Data Visualization and Analysis Tutorial

Chris Davis

20 August, 2016

The shortened URL for this page is [**http://is.gd/ODGU2016**](http://is.gd/ODGU2016)

The journal article [**The State of the States: Data-driven Analysis of the US Clean Power Plan**](http://www.sciencedirect.com/science/article/pii/S1364032116001271) shows applications of the analysis and visualization shown here

### Introduction

There are a variety of different tools available which are useful for data visualization and analysis. For dealing with large amounts of data you may see people using [R](https://www.r-project.org/) or [Python](https://www.python.org/). For visualization on the web, you often see JavaScript libraries just as [d3.js](https://d3js.org/).

What we'll focus on today is an example using the R programming language and two of its libraries which are commonly used for data analysis and visualization: dplyr and ggplot2. R is a programming language that is commonly used for data analysis. As R is open source, people are also free to contribute libraries to it that help to extend its functionality in ways that allow you to do new things, or make it easier to do existing things. Currently there are nearly [9000 packages](https://cran.r-project.org/web/packages/) meaning that you can often reuse code instead of having to create something yourself.

For both of the dplyr and ggplot2 libraries, there are reference sheets available which give a quick overview of what they're capable of. Today we'll be showing only a small subset of their functionality.

* [Data Visualization with ggplot2 Cheat Sheet](https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf)
* [Data Wrangling with dplyr and tidyr Cheat Sheet](https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf)

### Learning Objectives

In this tutorial we will show how to use these open source libraries to be able to explore large complex datasets. More specifically, after this lecture you should be familiar with several methods of filtering, grouping and summarizing data, in addition to different ways of visualizing the data. Given a complex dataset of your own, you should be able to describe different options that would allow you to explore the data.

### Examples

To start off, we'll look at a data set that is included with R. This is a [famous data set collected in 1936 by the statistician Ronald Fisher](https://en.wikipedia.org/wiki/Iris_flower_data_set) that documents measurements of different species of iris flowers. It's an interesting data set since the different species have similar measurements, and in some cases it can be difficult to tell them apart simply from their measurements.

Tabular data like you would find in spreadsheets is represented within R in what are known as data frames.

Looking at the first few lines of the iris data frame, we see the following. There are five columns: Sepal.Length, Sepal.Width, Petal.Length, Petal.Width and Species

head(iris)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Species |
| 5.1 | 3.5 | 1.4 | 0.2 | setosa |
| 4.9 | 3.0 | 1.4 | 0.2 | setosa |
| 4.7 | 3.2 | 1.3 | 0.2 | setosa |
| 4.6 | 3.1 | 1.5 | 0.2 | setosa |
| 5.0 | 3.6 | 1.4 | 0.2 | setosa |
| 5.4 | 3.9 | 1.7 | 0.4 | setosa |

One of the nice things about data frames is that you can use the names of the columns (combined with the $ sign) to directly access the values in that column:

iris$Sepal.Length

## [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4  
## [18] 5.1 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5  
## [35] 4.9 5.0 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0  
## [52] 6.4 6.9 5.5 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8  
## [69] 6.2 5.6 5.9 6.1 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4  
## [86] 6.0 6.7 6.3 5.6 5.5 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8  
## [103] 7.1 6.3 6.5 7.6 4.9 7.3 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7  
## [120] 6.0 6.9 5.6 7.7 6.3 6.7 7.2 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7  
## [137] 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8 6.7 6.7 6.3 6.5 6.2 5.9

### ggplot2

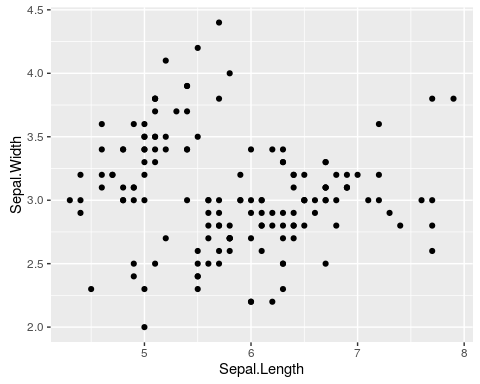
Many visualizations created with R are often created using the [ggplot2 library](http://ggplot2.org/). What's interesting about this library is that way in which it allows you to construct visualizations. The gg in ggplot2 stands for the Grammar of Graphics. The idea is that when you create plots, you are basically writing sentences that are of the form:

Here's my data frame + Here are the x and y columns + Apply this kind of plot to that data + These are the axis labels + here are some more additional transformations

The syntax may look strange at first, although it’s a very modular approach, and you can create very complex visualizations just by adding new parts to these sentences.

To visualize this, we can first do a simple scatter plot. You'll notice with the syntax that we first start with the iris data frame, then we specify which columns are to be associated with the x and y values, and then we specify that we want to plot the data as points by adding + geom\_point().

ggplot(iris, aes(x=Sepal.Length, y=Sepal.Width)) + geom\_point()

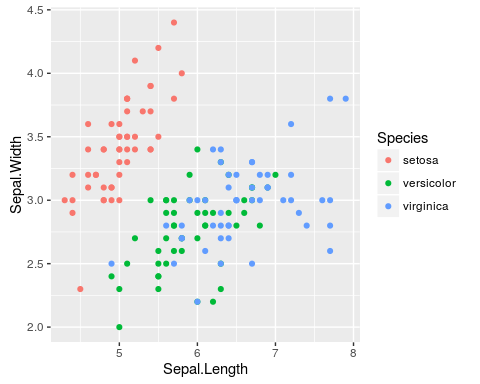


In the following examples, you may see the code examples split over multiple lines. The two statements below are actually equivalent, but by spreading the commands over multiple lines it can sometimes help to make things more readable by separating the code into its different functional pieces.

# same code in 1 line  
ggplot(iris, aes(x=Sepal.Length, y=Sepal.Width)) + geom\_point()  
  
# same code in 4 lines  
ggplot(iris,   
 aes(x=Sepal.Length,   
 y=Sepal.Width)) +   
 geom\_point()

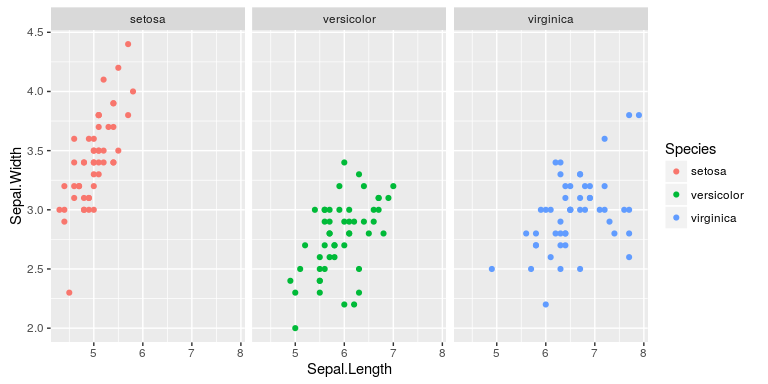
In addition to specifying which columns represent x and y values, we can also specify which one will be used for the colour of the points:

ggplot(iris, aes(x=Sepal.Length, y=Sepal.Width, colour=Species)) +   
 geom\_point()



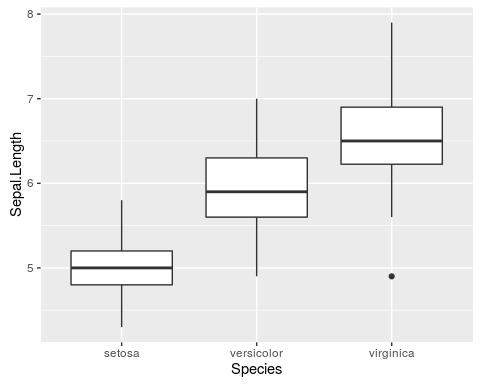
And we can also add an additional statement, facet\_wrap, to indicate that we want separate plots for each of the data per species.

ggplot(iris, aes(x=Sepal.Length,   
 y=Sepal.Width,   
 colour=Species)) +   
 geom\_point() +   
 facet\_wrap(~Species)



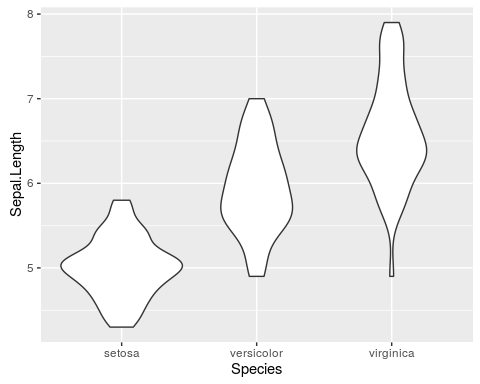
Instead of just having numeric values on the axes, we can also use categorical values such as the species, and then use this to create a box plot:

ggplot(iris, aes(x=Species, y=Sepal.Length)) + geom\_boxplot()



or change this to a violin plot, which is a similar concept, but shows the number of points at a particular value by the width of the shapes.

ggplot(iris, aes(x=Species, y=Sepal.Length)) + geom\_violin()



For analyzing the data, we can use the dplyr library. The dplyr library takes a very similar approach to what we see with ggplot. We start off with data represented in a table, or in what R calls a data frame, and then we perform a series of operations or data transformations which are connected together via the %>% symbol which acts as a sort of "pipe" through which data flows.

In very general terms, we may create a series of statements that look like:

data %>% operation 1 %>% operation 2

As an example of what this can look like, using the iris data set again, we do the following:

1. group the data by species using group\_by(Species)
2. per species we calculate the mean, standard deviation, minimum, and maximum sepal lengths using the summarize command.

This creates new columns such as mean\_sepal\_length which is calculated via the function mean(Sepal.Length)

iris %>%   
 group\_by(Species) %>%   
 summarize(mean\_sepal\_length = mean(Sepal.Length),   
 sd\_sepal\_length = sd(Sepal.Length),   
 min\_sepal\_length = min(Sepal.Length),  
 max\_sepal\_length = max(Sepal.Length))

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Species | mean\_sepal\_length | sd\_sepal\_length | min\_sepal\_length | max\_sepal\_length |
| setosa | 5.006 | 0.3524897 | 4.3 | 5.8 |
| versicolor | 5.936 | 0.5161711 | 4.9 | 7.0 |
| virginica | 6.588 | 0.6358796 | 4.9 | 7.9 |

### Advanced Example

For a more complicated example, we use a data set on electricity generation in the US. This data contains around 1.5 million observations on monthly generation from individual generators for the years 2001 through 2014. Here we see a preview of the data where we only look at the columns for plant\_name, state, general\_fuel\_name (the fuel type), date, and netgen (the amount of electricity generated).

head(generationAndFuelData[,c("plant\_name", "state", "general\_fuel\_name", "date", "netgen")])

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| plant\_name | state | general\_fuel\_name | date | netgen |
| Grand Tower | IL | Coal | 2001-06-15 | 0.000 |
| Weyerhaeuser Pine Hill Operations | AL | Coal | 2001-06-15 | 5733.327 |
| Hayden | CO | Coal | 2007-07-15 | 319839.745 |
| State-Fuel Level Increment | IN | Coal | 2012-08-15 | 0.000 |
| Rivesville | WV | Coal | 2007-08-15 | 28822.905 |
| Archer Daniels Midland Mankato | MN | Coal | 2002-02-15 | 2054.054 |

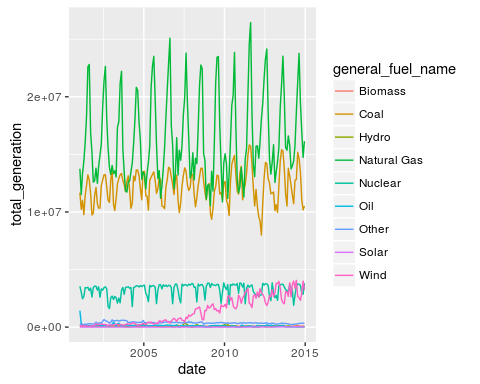
This is too much data to visualize, so we'll have to summarize it first. As an example, we

1. filter by all the generators in the state of Texas
2. group all values by date and general\_fuel\_name
3. sum up the total amount of electricity generation by date (per month in this case) and by the general\_fuel\_name.

tmp = generationAndFuelData %>%   
 filter(state == "TX") %>%   
 group\_by(date, general\_fuel\_name) %>%   
 summarize(total\_generation = sum(netgen, na.rm=TRUE))

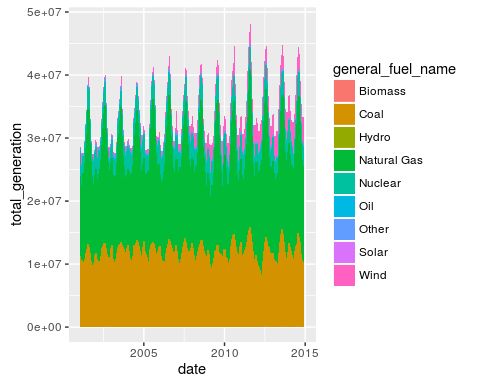
Next we plot the values where the x axis is the date, the y axis shows the total\_generation per date and fuel as calculated previously, and we color the data by the general\_fuel\_name. We then use + geom\_line() to specify that lines should be drawn for the data. Because we specified colour=general\_fuel\_name this means that the lines will be grouped by the general\_fuel\_name.

ggplot(tmp, aes(x=date,   
 y=total\_generation,   
 colour=general\_fuel\_name)) +   
 geom\_line()



We can also stack the values per date so we can get an overview of how much generation from each fuel type contributes to the whole.

ggplot(tmp, aes(x=date,   
 y=total\_generation,   
 fill=general\_fuel\_name)) +   
 geom\_area()

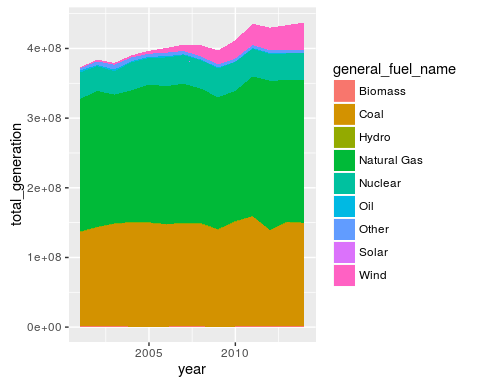


What we see with this is that there's a lot of variation over the course of a year, likely due to aspects such as increased air conditioning demand in the summer. To smooth this out, we can also sum up the total amount of generation per fuel type per year.

In the steps below we:

1. Keep all data for generators in Texas (state == "TX")
2. Use mutate to add a new column called year that is derived from the year value of the date
3. Group all data into blocks by the year and fuel type
4. Per block, sum up the total amount of generation.

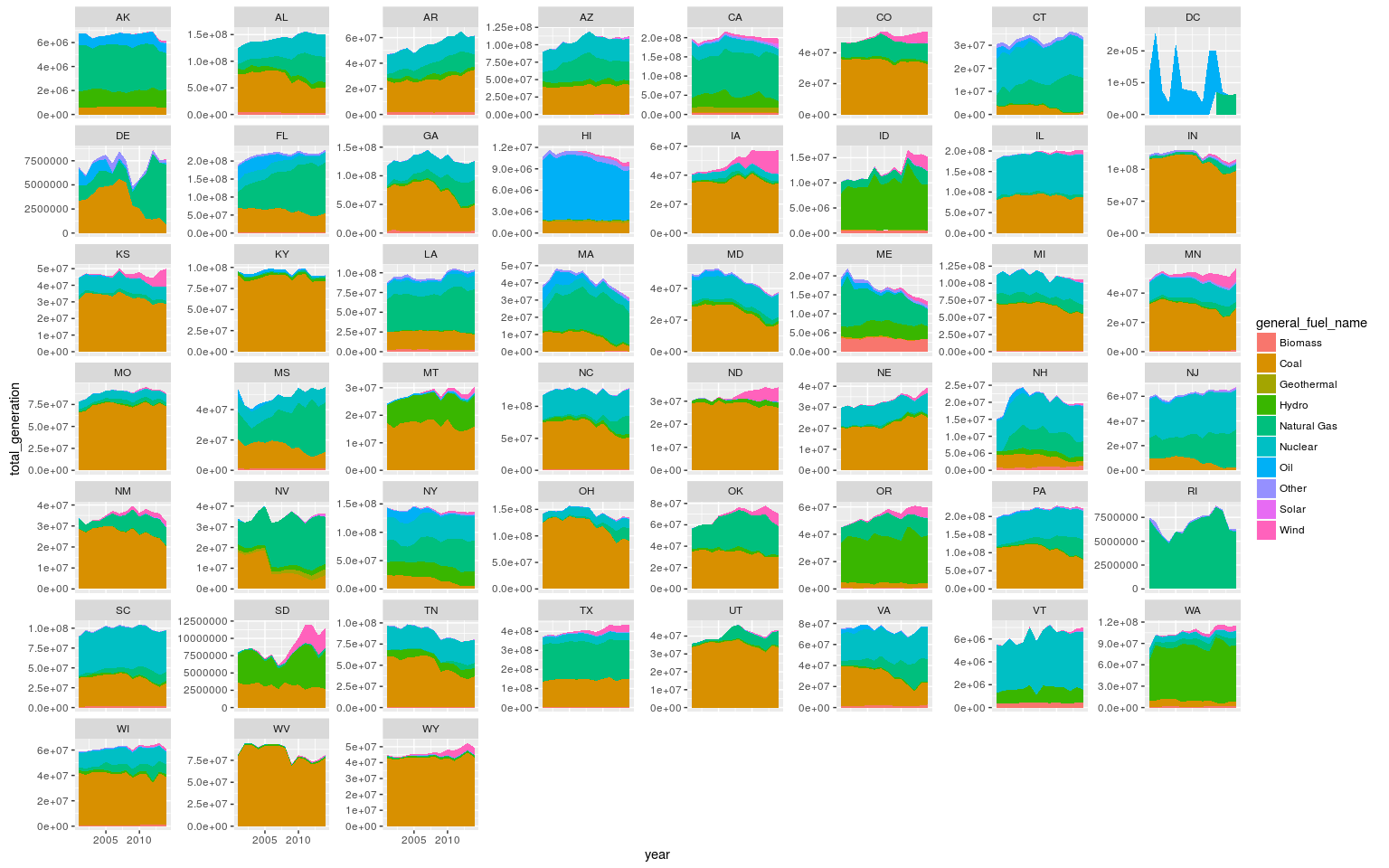
tmp = generationAndFuelData %>%   
 filter(state == "TX") %>%   
 mutate(year = year(date)) %>%  
 group\_by(year, general\_fuel\_name) %>%   
 summarize(total\_generation = sum(netgen))  
  
ggplot(tmp, aes(x=year,   
 y=total\_generation,   
 fill=general\_fuel\_name)) +   
 geom\_area()



We can also do this for the entire US by using the facet\_wrap command to create a plot for every single state. Note how we specify scales="free\_y" for the facet wrap so that the y axes for the plots are scaled to the maximum values in each states. Without this, we would only be able to clearly see the data for the states that are the largest producers of electricity.

From this plot, we can see the diversity across the US states in terms of the types of fuel used. It also becomes clear which states have changed their fuel types over time. In particular, you can see large amounts of wind capacity being installed in the states in the Midwest of the country, and a general shift away from coal to increased use of natural gas.

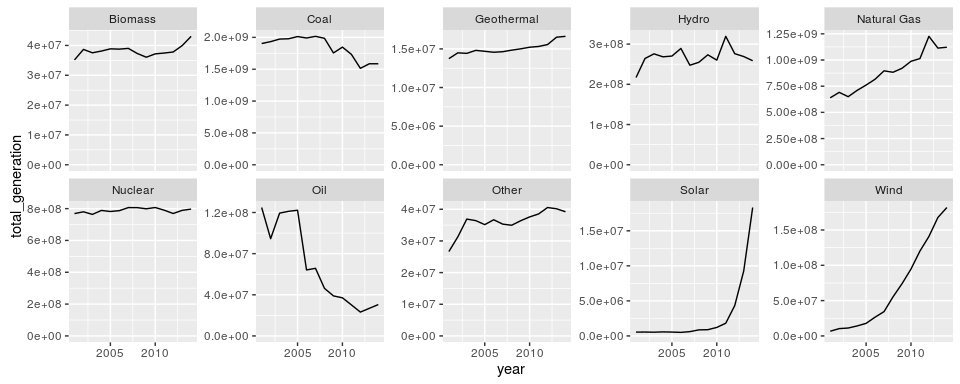
tmp = generationAndFuelData %>%   
 mutate(year = year(date)) %>%  
 group\_by(year, general\_fuel\_name, state) %>%   
 summarize(total\_generation = sum(netgen))  
  
ggplot(tmp, aes(x=year,   
 y=total\_generation,   
 fill=general\_fuel\_name)) +   
 geom\_area() +   
 facet\_wrap(~state, scales="free\_y")



Similarly, we can analyze yearly generation from different sources for the entire US. Again we use a facet\_wrap with independent scaling of the y axes. Additionally, expand\_limits is specified to make sure that the minimum value shown on the y axes is zero.

What we can see with these graphs is that generation from coal is going down, while that from natural gas is increasing. It's also interesting to note that generation from solar and wind are increasing at a very fast pace, although they are still only a small part of total

tmp = generationAndFuelData %>%   
 mutate(year = year(date)) %>%  
 group\_by(year, general\_fuel\_name) %>%   
 summarize(total\_generation = sum(netgen))  
  
ggplot(tmp, aes(x=year, y=total\_generation)) +   
 geom\_line() +   
 facet\_wrap(~general\_fuel\_name, scales="free\_y", ncol=5) +   
 expand\_limits(y=0)



We can also find out which states were the top generators of electricity from wind in 2014

1. mutate lets us add new columns based on values in other columns. The date column contains values such as 12-15-2014. By using year = year(date) we create a new column year that is based on the year value of the date, or in the example of 12-15-2014, 2014
2. We filter for all generation data where the general\_fuel\_name is Wind and for where the year is 2014.
3. We use group\_by to group all the data into blocks per state.
4. Per block, we summarize to create a new column total\_generation which is the sum of the values in the netgen column
5. Finally we sort the results based on the descending values of total\_generation

tmp = generationAndFuelData %>%   
 mutate(year = year(date)) %>%  
 filter(general\_fuel\_name == "Wind", year == 2014) %>%  
 group\_by(state) %>%  
 summarize(total\_generation = sum(netgen)) %>%   
 arrange(desc(total\_generation))

head(as.data.frame(tmp))

|  |  |
| --- | --- |
| state | total\_generation |
| TX | 39371136 |
| IA | 16295245 |
| CA | 13775969 |
| OK | 11861988 |
| KS | 10843900 |
| IL | 10076508 |

### Conclusion

This has been just a brief introduction into data analysis and visualization, and there is clearly much more that is possible. When choosing the types of tools that you will use for this, it is also important to think not just in terms of programming languages, but also about the communities around them.

With languages such as R, Python and JavaScript, there are large numbers of people who are actively using these, which means that it is easy to find documentation and help. Furthermore, the open source nature of these means that people are continually developing new libraries which help to provide additional functionality that may be useful for you. In practice, you may find yourself using several libraries developed by different people, which help you to manage aspects of data gathering, analysis and visualization.