Ggplot2 Guide

Posted on *May 7, 2013* by *George*

Most ggplot2 tutorials start off using the qplot function or only use qplot. I find this confusing as qplot is a wrapper around the more versatile ggplot command. So to add to the multitude of ggplot2 tutorials here are my notes on getting to grips with ggplot2 without any further mention of qplot.

We now offer a Add In that lets you build ggplot charts in Excel without having to export your data or write any R code.

ggplot2 is a large package and it is impossible to cover it all, so you need to refer to the ggplot2 website for details of all the geoms, scales and stats. There is also the ggplot2 book, 'Elegant Graphics for Data Analysis' (details on the ggplot2 website) is worth a read if you want to know a bit more about the package.

The Basics

At the heart of every ggplot2 chart is the data which must be in the form of a data frame. Plots are built up in layers, first the default data is specified and then the graphical objects, geoms and statistical views, stats are put on top. Many different layers can be added as required and each layer can use data from a different data frame. I'm using the iris data set that is part of the base R package for these examples. To start load ggplot2 and the data.

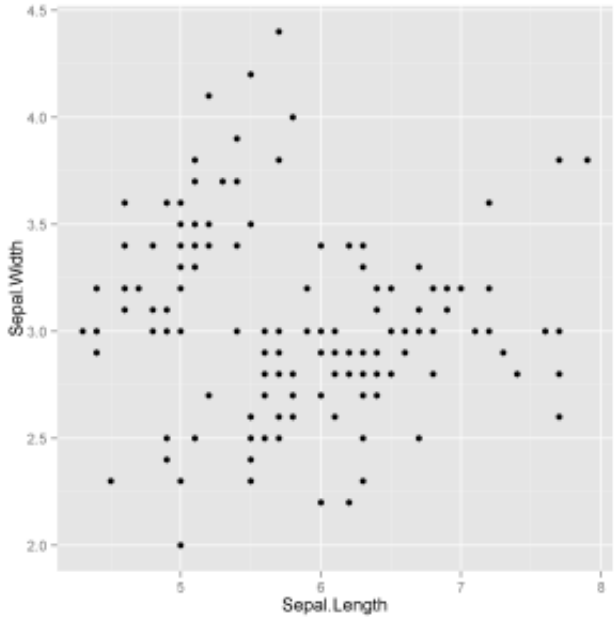
```
library(ggplot2)
data(iris)
```

To produce a plot fist call the ggplot command, which can also be used used to specify the default data frame

```
ggplot(iris,aes(Sepal.Length,Sepal.Width))
```

which promptly produces an error,'No layers in plot'. This is because the data frame, iris has been specified along with which 2 variables to plot, Sepal.Length along the x axis and Sepal.Width along the y axis but no information on what graphical component to use. To produce a simple scatter plot as shown below, add a layer that displays points for each data point, using geom_point()

```
ggplot(iris, aes(Sepal.Length, Sepal.Width)) + geom_point()
```



The aes function is the aesthetic function and is used to detail what is displayed and how it is displayed. There is no need to use the typical R notation dataFrame$Variable when specifying variables. The variables can be specifically assigned to an axis if required.

```
ggplot(iris,aes(x=Sepal.Length,y=Sepal.Width))+geom_point()
```

In the previous example the data has been specified in the ggplot command, but each layer can use it's own data frame,variables and aesthetics, so the following 2 lines produce the same plot.

```
ggplot(iris,aes(Sepal.Length,Sepal.Width))+geom_point()
ggplot()+geom_point(data=iris,aes(Sepal.Length,Sepal.Width))
```

Important point to note is that the layer specifications overrides those detailed in ggplot() so unexpected results are possible,

```
ggplot(iris,aes(Sepal.Length,Sepal.Width))+geom_point(aes(Sepal.Width,Se
```

produces a plot with what looks like the axis are mixed up as the geom_point aes has over ridden the order specified in the default ggplot call. This is a little confusing at first due to the use of that '+' symbol, which means more than just 'add a layer'.

The use of the '+' operator is useful as it allows for the code to be split into separate lines as if often seen in examples,

```
p<-ggplot(iris,aes(Sepal.Length,Sepal.Width))
p+geom_point()
```

Using this method also allows for easier reading and re use of code, for example a chart can be updated with another data set using the odd looking operator,

%+%

To illustrate this create two data sets,

```
setosaData <- subset(iris, Species == "setosa")
verData <- subset(iris, Species == "versicolor")
```

Produce a plot of one data set, although note that now the plot is begin assigned you may have to type p or print(p) to display the plot, depending on your environment.

```
p<-ggplot(setosaData,aes(Sepal.Length,Sepal.Width))+geom_point()
```

Now to produce the same chart but using the other data set

```
p %+% verData
```

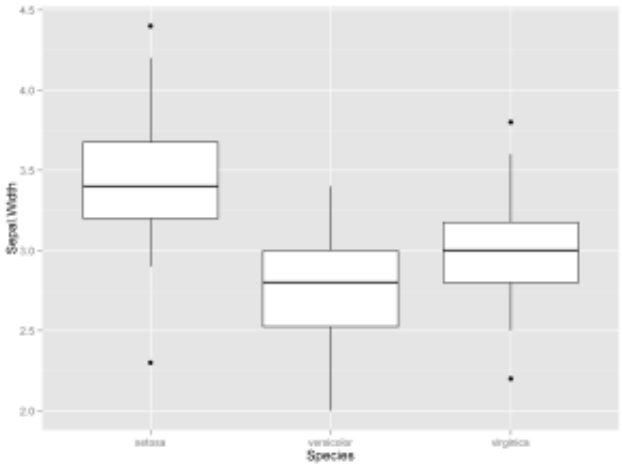
With ggplot2 the data frame is part of the plot so when the plot is saved so is the data. This stops the old problem of trying to match data sets to charts when you revisit analysis done 6 months ago.

```
p<-ggplot(iris,aes(Sepal.Length,Sepal.Width))+geom_point()
#Save the plot
save(p,file="FirstGgplot.RData")
#Load
load("FirstGgplot.RData")
#Save to image
ggsave("PlotImage.png";)
```

Geoms

Geoms are the names for the types of shapes that represent the data on the chart, and there are two main types. The geom_point is an example of geom that works on individual data points, and so is straight forward to use as already shown. However a collective geom such as geom_boxplot requires some collection of data to work on. So to produce a box plot,

```
p <- ggplot(iris, aes(Species, Sepal.Width))
p + geom_boxplot()
```



The above plot shows a box plot of the Sepal.Width for each species. Note that ggplot2 has automatically divided the data into the correct categories as the x axis variable is discrete. Now it is also equally valid to produce a box plot with continuous variable say Sepal.Length along the x axis.

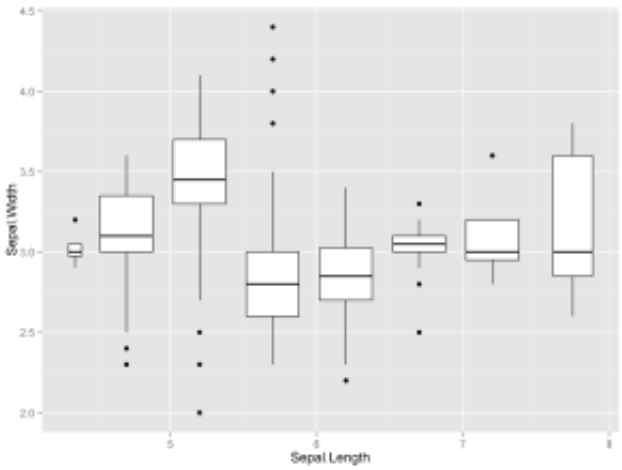
```
p<-ggplot(iris,aes(sepal.Length,Sepal.Width))
p+geom_boxplot()
```

which if tried produces a single box plot using all the data, not particularly useful. However use of the group command results in something a bit more useful

```
p<-ggplot(iris,aes(Sepal.Length,Sepal.Width))
p+geom_boxplot(aes(group=Sepal.Length))
```

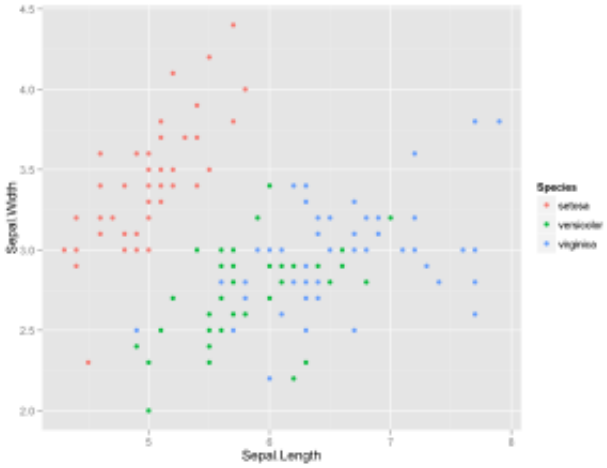
Making sure you have the plyr package (which is a prerequisite of ggplot2) loaded the size of the bins used for the box plot are easily adjusted with the help of the round_any function. Note this type of thing often gets a warning as typically the bins at the edge might not have enough data, as in this case where there is only one value of greater than 7.75.

```
library(plyr)
p <- ggplot(iris, aes(Sepal.Length, Sepal.Width))
p + geom_boxplot(aes(group = round_any(Sepal.Length, 0.5)))
```



If you check out the ggplot2 site, there are detailed notes on each of the geoms available and there are some 30 odd different geoms so plenty of options to play with. On the geom_point page it specifies that the aesthetics has options shape, colour, size, fill, and alpha. How do we make use of this? In the aes settings on the geom each property can be set to a different value or variable, so as example the following code

```
p<-ggplot(iris,aes(Sepal.Length,Sepal.Width))
p+geom_point(aes(color=Species))
```

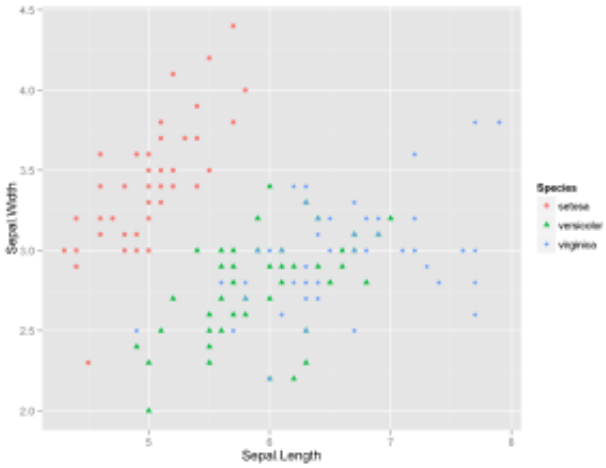


produces a plot as shown where the colour of each point indicates the Species. Note that the legend is automatically added.Each of the options can be assigned to variables in the same aes function so some interesting plots can quickly be produced.

Scales

The colours and shapes used in the chart can be manually adjusted if you don't like the defaults, so the following code and chart demonstrates the use of manual scales.

```
p <- ggplot(iris, aes(Sepal.Length, Sepal.Width)) + geom_point(aes(color
shape = Species))
p + scale_colour_manual(values = c("red", "green", "blue"))
p + scale_shape_manual(values = c(16, 17, 18))
```



Note that geom_point aes call specifies colour and shape, if either specification wasn't there the corresponding manual scale tweak would make no difference. The shape specification refers to the R graphics option pch.

Scales are also used on the axis, and are again split into two types, continuous and discrete. The limit and break options allow the adjustment of the tick marks and the range.

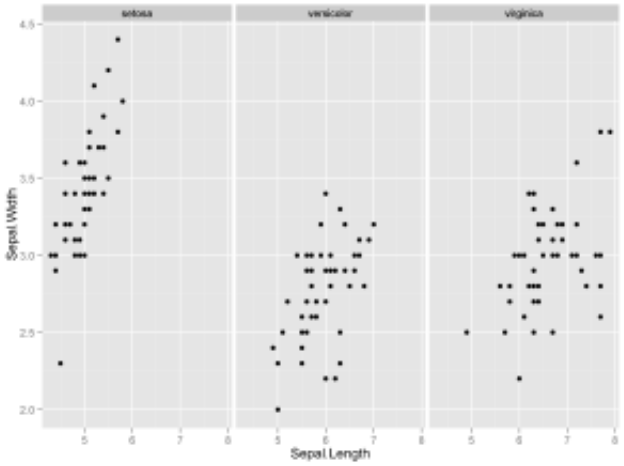
```
p<-ggplot(iris,aes(Sepal.Length,Sepal.Width))
p+geom_point(aes(color=Species))
p+scale_x_continuous(limits=c(5,7),breaks=c(5:7))
```

The above code customises the x axis. There is much more to scales than the above examples, you can reverse the axis, have logarithmic scales and specify the axis labels to mention just a few options.

Faceting

Instead of grouping data on a single plot faceting is a easy way of groups of data on different plots, as shown below where the Species is used to show the plots as a single row of columns.

```
p<- ggplot(iris, aes(Sepal.Length, Sepal.Width))
p + geom_point() + facet_grid(. ~ Species)
```



If you want the plots in rows then use,

```
+facet_grid(Species~.)
```

When there are two discrete variables a grid can be laid our by specifying a variable either side of '~'.

Stats

Stats apply statistical transformations that are used to summarise the data, and allows a huge range of possibilities. Stat_smooth is a nice stat to illustrate the principles, which fits a line and a shaded band to indicate some specified level of uncertainty, as shown in the following example which fits a linear regression line.

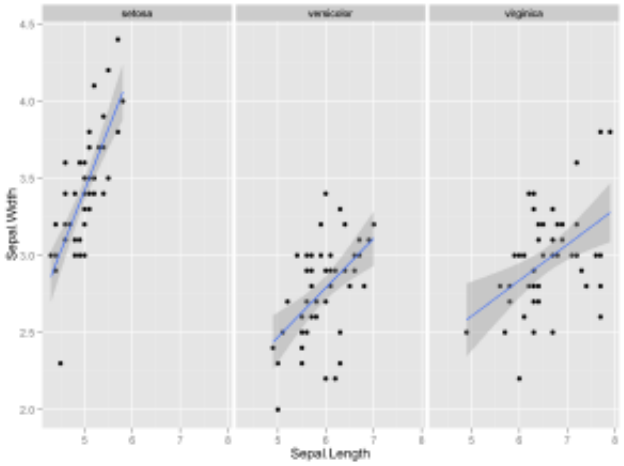
```
ggplot(iris,aes(Sepal.Length,Sepal.Width))+geom_point()+ stat_smooth(met
```

Stats are automatically split according to the factors so

```
,aes(Sepal.Length,Sepal.Width,color=factor(Species)))+geom_point()+ stat
```

produces a plot with 3 sets of fitted lines on which could be considered a little confusing. As faceting is similar to using factors the stat is also automatically split over the plots as shown below.

```
p <- ggplot(iris, aes(Sepal.Length, Sepal.Width)) + geom_point()
p + facet_grid(. ~ Species) + stat_smooth(method = "lm")
```



There are various options for the model, which again is listed in the notes on the ggplot2 website. For example if you want a quadratic model, use the standard R notation. Note that the variables are referred to as x and y not by name.

```
stat_smooth(method="lm",formula=y~I(x^2))
```

As the stat is just another layer on the plot the original geom can be omitted if required just to show the stat.

```
ggplot(iris,aes(Sepal.Length,Sepal.Width,color=factor(Species)))+ stat_s
```

A very flexible stat is the stat_summary function which allows definition of custom summary statistics. As an example to produce a bar chart of the mean value use,

```
ggplot(iris,aes(Species,Sepal.Width))+stat_summary(fun.y=mean,geom="bar"
```

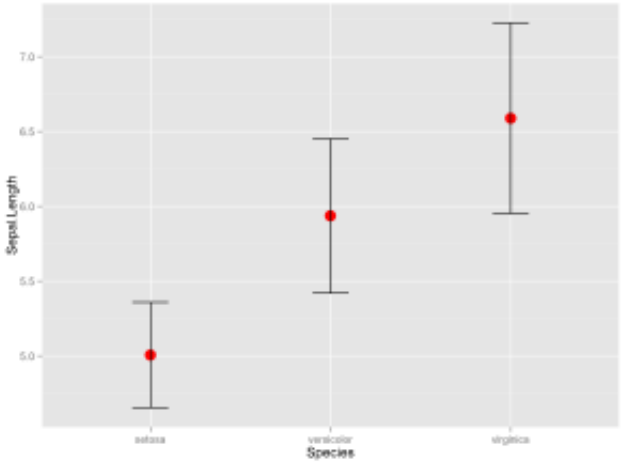
which works as mean takes a vector and returns as single value that defines the y value for each bar. Stat_summary can also take 2 other values ymin and ymax which define the upper and lower bounds.The following function shows how to specify your own summaries. Define a function that takes a vector and returns the mean plus and minus 1 standard deviation, and names the two results as ymin and ymax

```
myFunc = function(x) {
result = c(mean(x) - sd(x), mean(x) + sd(x))
names(result) = c("ymin", "ymax")
result
}
```

then pass this function into stat_summary and the result is used to define the limits for error bars.

```
p <- ggplot(iris, aes(Species, Sepal.Length))
p + stat_summary(fun.y = mean, geom = "point", color = "red", size = 5)
```

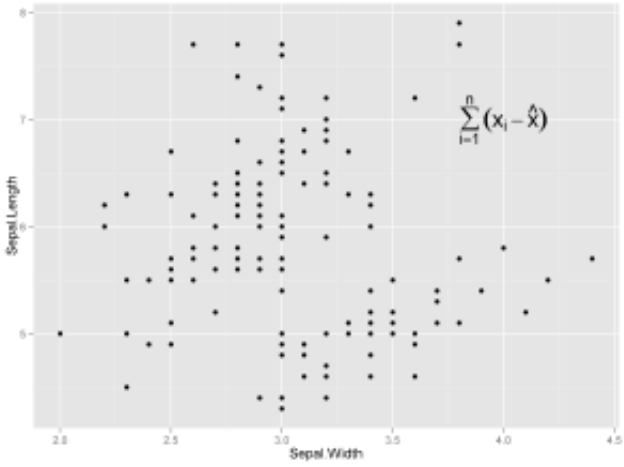
```
geom = "errorbar", width = 0.2)
```



Annotation

Text annotations are simple to apply to a plot and using plotmath notation decent math notation is also easy to include in annotation

```
standard_text = "Some Annotation"
plotmath_text <- "sum((x[i]-hat(x)), i==1, n)"
p<- ggplot(iris, aes(Sepal.Width, Sepal.Length)) + geom_point()
p + annotate("text", x = 3, y = 6, label = standard_text) + annotate("te
x = 4, y = 7, cex = 7, label = plotmath_text, parse = TRUE)
```



It is also easy to add lines and such to plots

```
ggplot(iris, aes(Sepal.Width, Sepal.Length)) + geom_point() + annotate("
xmin = 3, xmax = 3.5, ymin = 6, ymax = 7, fill = "red", alpha = 0.2) + a
x = 2, xend = 4, y = 6, yend = 8, colour = "blue")
```

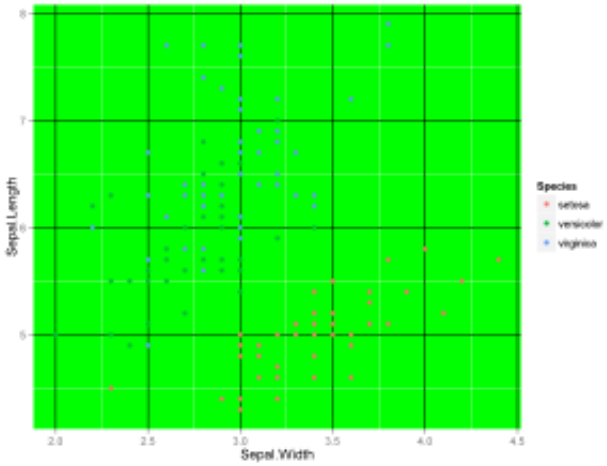
Themes

The default look of ggplot2 charts are not to everyone taste but there are several default themes that make changing things easy. For example the minimal theme,

```
p<-ggplot(iris,aes(Sepal.Width,Sepal.Length))+geom_point(aes(color=Speci
p + theme_minimal()
```

As with all things ggplot2 it is easy to change most aspects of a theme, it just takes a while to learn the details of what can be adjusted, and doesn't guarantee pleasant results.

```
p<- ggplot(iris, aes(Sepal.Width, Sepal.Length)) + geom_point(aes(color
p + theme(panel.background = element_rect(fill = "green"), panel.grid.ma
```



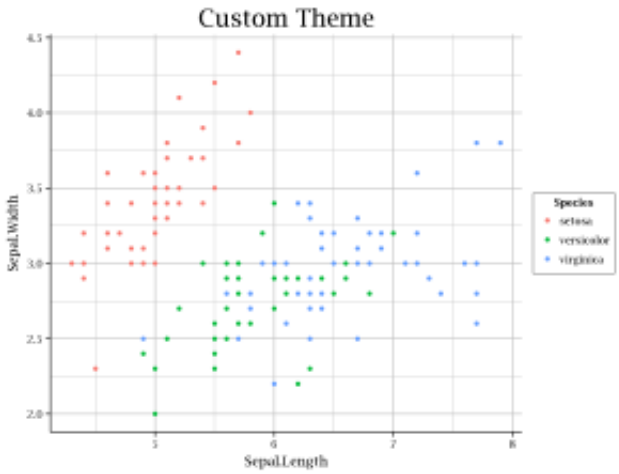
It is also possible to define your own theme, just take the code for an existing theme and change what you need. For fonts look at this article. Note that there have been recent changes to the theme aspect of Ggplot2 and so code from old articles might need updating. The following example is a customised theme is based on the default grey theme

```
theme_sharp<-function(base_size = 12, base_family = "Trebuchet MS")
{
theme(line = element_line(colour = "black", size = 0.5, linetype = 1,
lineend = "butt"),
rect = element_rect(fill = "white", colour = "black", size = 0.5, linety
text = element_text(family = base_family, face = "plain",
colour = "black", size = base_size,
hjust = 0.5, vjust = 0.5, angle = 0, lineheight = 0.9),
axis.text = element_text(size = rel(0.8), colour = "black"),
strip.text = element_text(size = rel(0.8)),
axis.line = element_line(),
axis.text.x = element_text(vjust = 1),
axis.text.y = element_text(hjust = 1),
axis.ticks = element_line(colour = "black"),
axis.title.x = element_text(vjust=0),
```

```
axis.title.y = element_text(angle = 90,vjust=0.3),
axis.ticks.length = unit(0.15, "cm"),
axis.ticks.margin = unit(0.1, "cm"),
legend.background = element_rect(colour = "black",size=0.25),
legend.margin = unit(0.2, "cm"),
legend.key = element_rect(fill = "white", colour = "white"),
legend.key.size = unit(1.2, "lines"),
legend.key.height = NULL,
legend.key.width = NULL,
legend.text = element_text(size = rel(0.8)),
legend.text.align = NULL,
legend.title = element_text(size = rel(0.8), face = "bold", hjust = 0),
legend.title.align = 0.5,
legend.position = "right",
legend.direction = NULL,
legend.justification = "center",
legend.box = NULL,</code>

panel.background = element_rect(fill = "white", colour = NA),
panel.border = element_blank(),
panel.grid.major = element_line(colour = "grey",size=0.5),
panel.grid.minor = element_line(colour = "grey", size = 0.25),
panel.margin = unit(0.25, "lines"),

strip.background = element_rect(fill = "white", colour = NA),
strip.text.x = element_text(),
strip.text.y = element_text(angle = -90),

plot.background = element_rect(colour = "white"),
plot.title = element_text(size = rel(1.8),vjust=1),
plot.margin = unit(c(1, 1, 0.5, 0.5), "lines"),

complete = TRUE
)
}
```

and to use the custom theme, assuming you have the grid package loaded

```
p<- ggplot(iris, aes(Sepal.Width, Sepal.Length)) + geom_point(aes(color
p + theme_sharp() + ggtitle("Custom Theme")
```



ggplot2 is a very flexible library and this flexibility can make learning to use it a confusing but once you get familiar with the functions and the documentation on the ggplot2 website some very useful plots can be quickly produced.

This entry was posted in *R*. Bookmark the *permalink*.

← Non Linear Curve Fitting in Excel

Multiple Pie Charts →

One thought on "Ggplot2 Guide"

Pingback: All about ggplot2 in R | Xizeng Mao's Blog

Comments are closed.

SEARCH

Search …

RECENT ARTICLES

- Sharp-R, An Excel Statistical Add In
- When Excel goes bad
- Sharp-R Data Update
- EARL Conference 2015
- Bar Charts, Error Bars and R

SUBSCRIBE

Sign up to our newsletter.

BLOGROLL

- Cesar Souza
- Eric Lippert
- Geertjan's Blog
- R Bloggers

CATEGORIES

- Case study
- Charts
- Demo
- Excel
- Java
- Latex
- News
- R

- Statistics
- Testimonial
- VSTO

ARCHIVES

- February 2016
- January 2016
- October 2015
- September 2015
- August 2015
- July 2015
- March 2015
- October 2014
- February 2014
- December 2013
- November 2013
- October 2013
- September 2013
- August 2013
- July 2013
- May 2013
- March 2013

Email Us

+44 (0) 7513 410395

Sharp Statistics Limited
Registered in the UK, No 8602876 Registered Office:
714 London Rd Aylesford
Maidstone KENT