Graphs and Data Visualisation in R

By the end of this section, you will be able to:

- Describe the ggplot "grammar of visualisation": coordinates and geoms
- Write a graph function to display multiple variables on a plot
- Amend the titles and legends of a plot
- Save plots in PDF or image formats

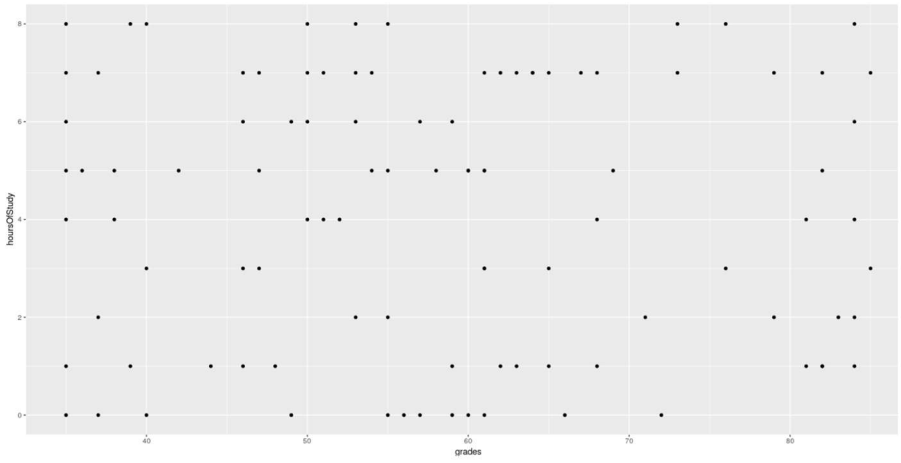
```
## Parsed with column specification:
## cols(
##   X1 = col_double(),
##   route = col_character(),
##   grades = col_double(),
##   hoursOfStudy = col_double(),
##   hasDepdendants = col_character(),
##   satisfactionLevel = col_character()
## )
```

The "grammar of visualisation"

- Graphs are made up of 3 components:
  - A dataset
  - A coordinate system
  - Visual marks to represent data **(geoms)**

The "grammar of visualisation" #2

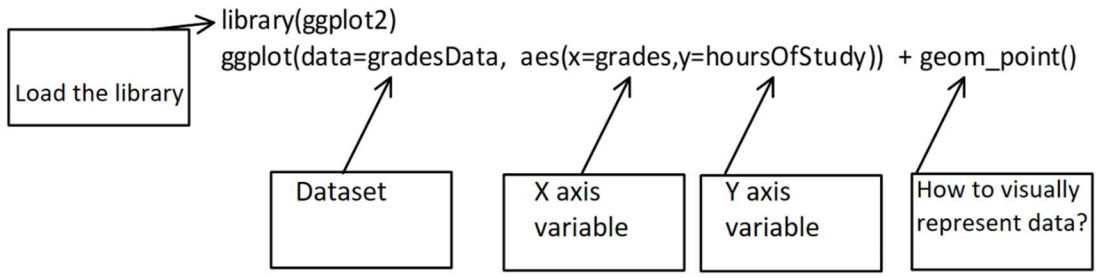
grades	hoursOfStudy
35	6
35	8
82	7
56	0
61	5
72	0
38	4
82	5
39	1



- In the above example, the dataset is the *studentData* that we used previously. - The *grades* variable is mapped to the X axis - The *hoursOfStudy* variable is mapped to the Y axis

How to code a graph

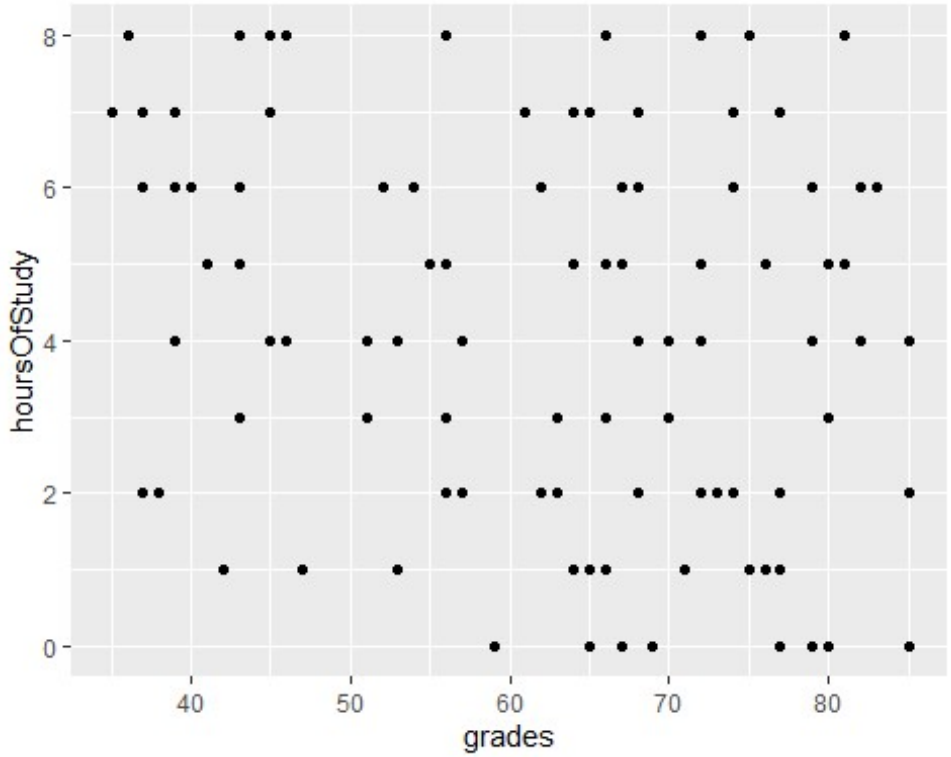
- The graph is created using the following code:



- In this code, we specify the dataset, the variables for the X and Y axes and the **geom** that will represent the data points visually (in this case, each datum is a point)

The graph output

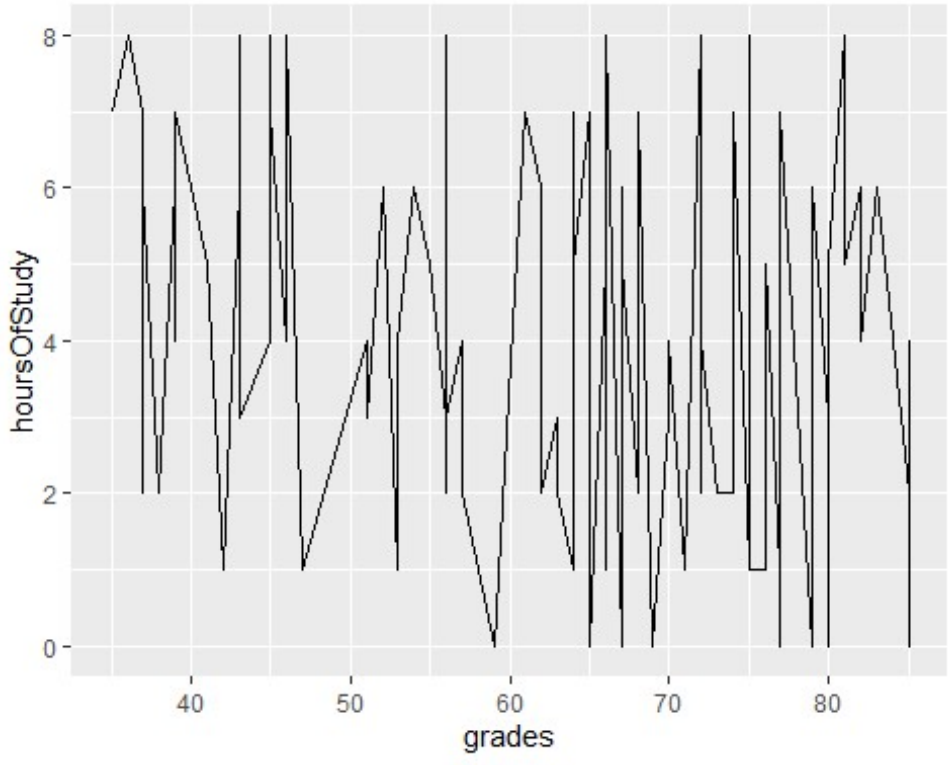
```
library(ggplot2)

ggplot(data=studentData, aes(x=grades,y=hoursOfStudy)) + geom_point()
```



Changing the geoms leads to different visualisations

- If we change from points to lines, for example we get a different plot:

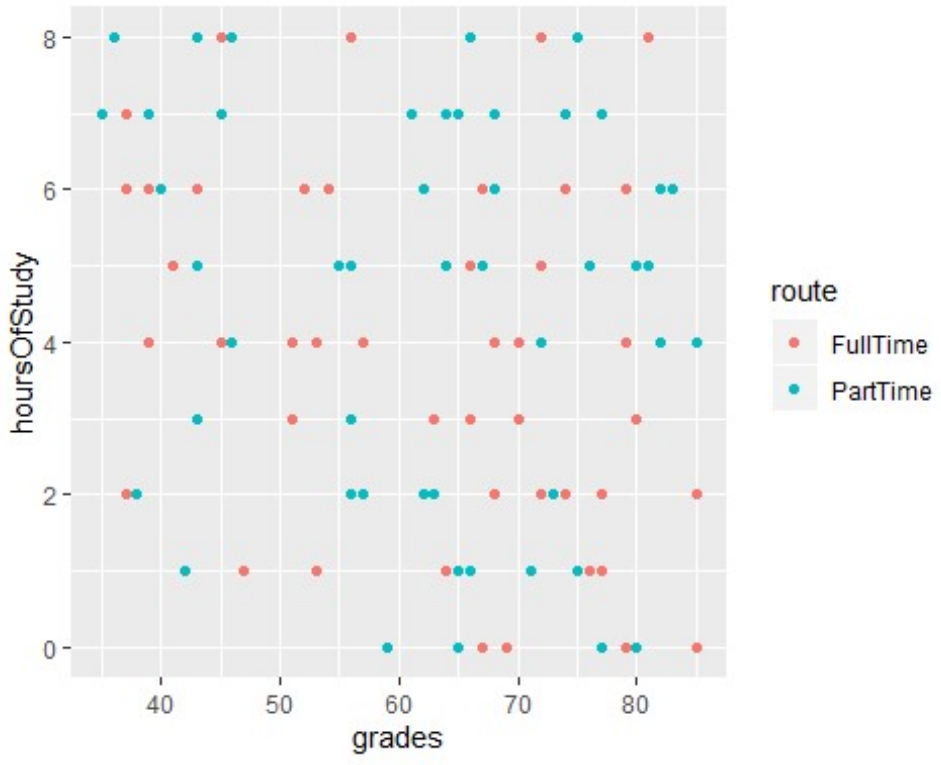
```
library(ggplot2)

ggplot(data=studentData, aes(x=grades,y=hoursOfStudy)) + geom_line()
```



It is possible to represent more variables on the plot

- By specifying that colours of our points should be attached to the **route** variable, the data is now colour-coded

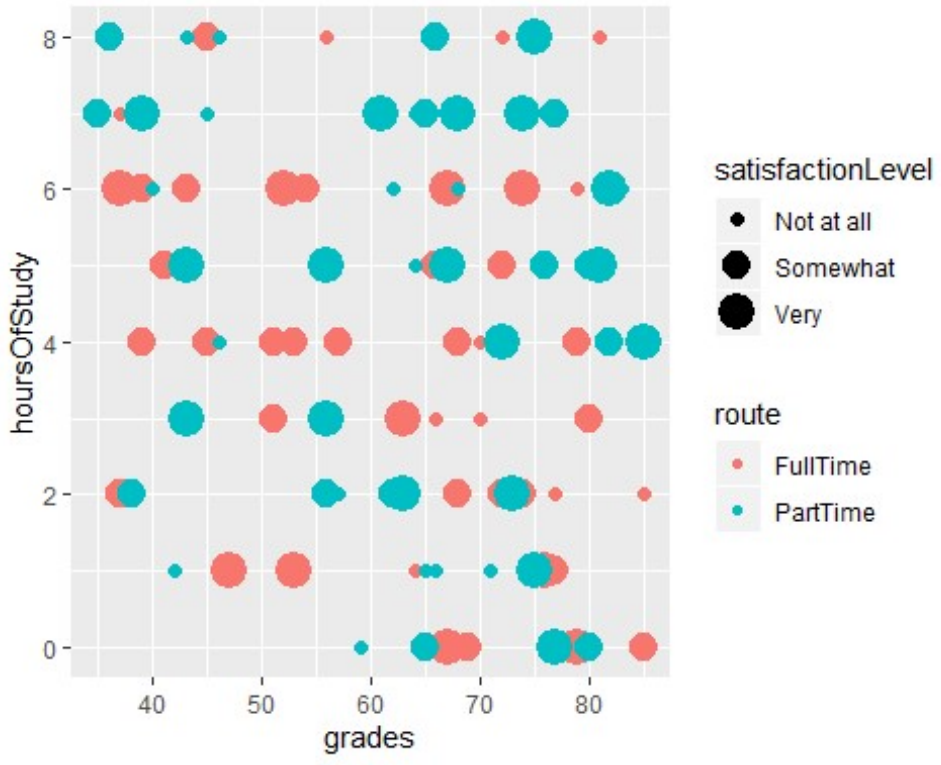
```
library(ggplot2)

ggplot(data=studentData, aes(x=grades,y=hoursOfStudy)) + geom_point(aes(color
= route))
```



It is possible to represent more variables on the plot #2

- By specifying that size of our points should be attached to the **satisfactionLevel** variable, the size of the points adjusts

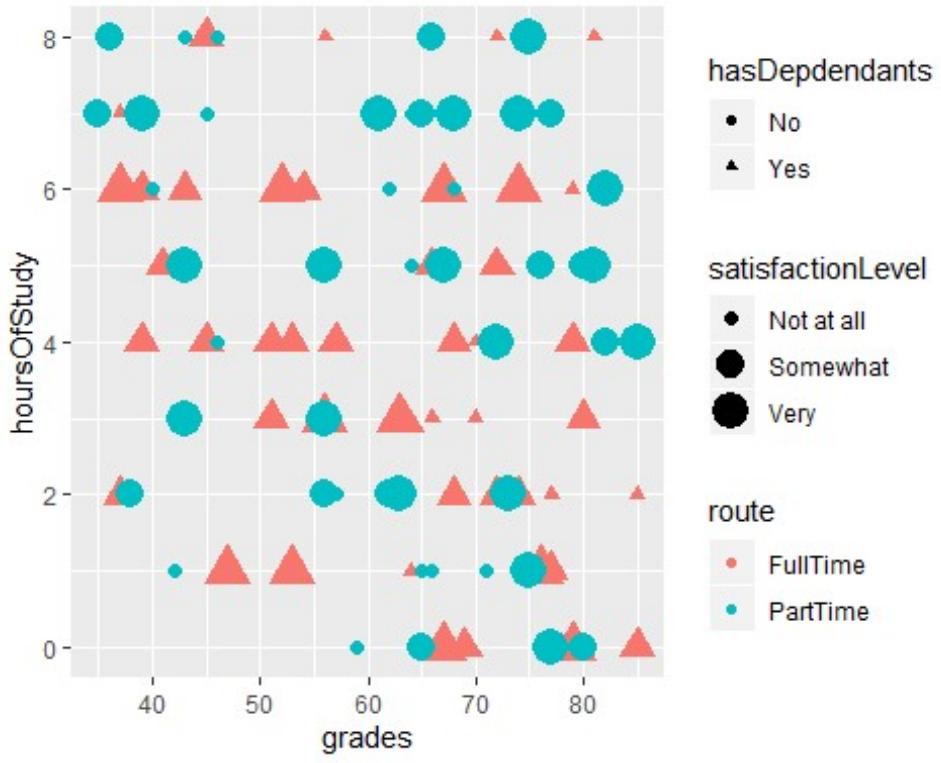
```
library(ggplot2)

ggplot(data=studentData, aes(x=grades,y=hoursOfStudy)) + geom_point(aes(color
= route, size=satisfactionLevel))
```



It is possible to represent more variables on the plot #3

- By specifying that shape of our points should be attached to the **hasDependents** variable, the shape of the points changes accordingly

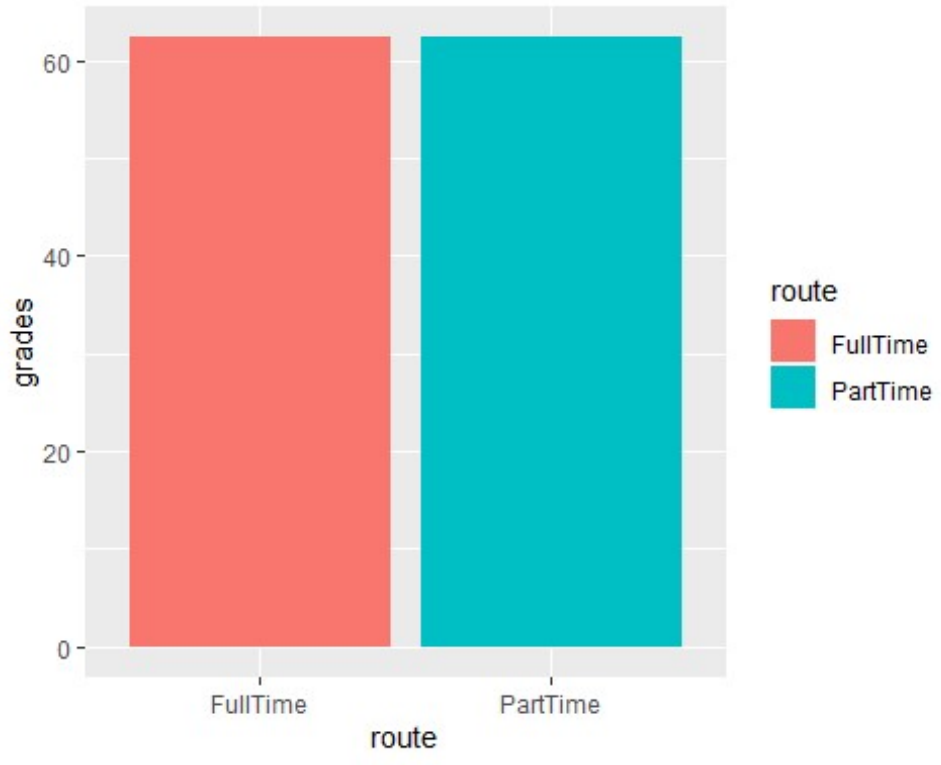
```
library(ggplot2)

ggplot(data=studentData, aes(x=grades,y=hoursOfStudy)) + geom_point(aes(color
= route, size=satisfactionLevel, shape=hasDepdendants))
```



Plotting summaries of data

- We can summarise the data (e.g. get the mean or sd) using the *stat_summary()* function
- Below we are making a bar chart with the mean grade for each route

```
ggplot(data=studentData, aes(x=route, y= grades, fill=route)) + stat_summary(
fun.y = "mean", geom = "bar")
```

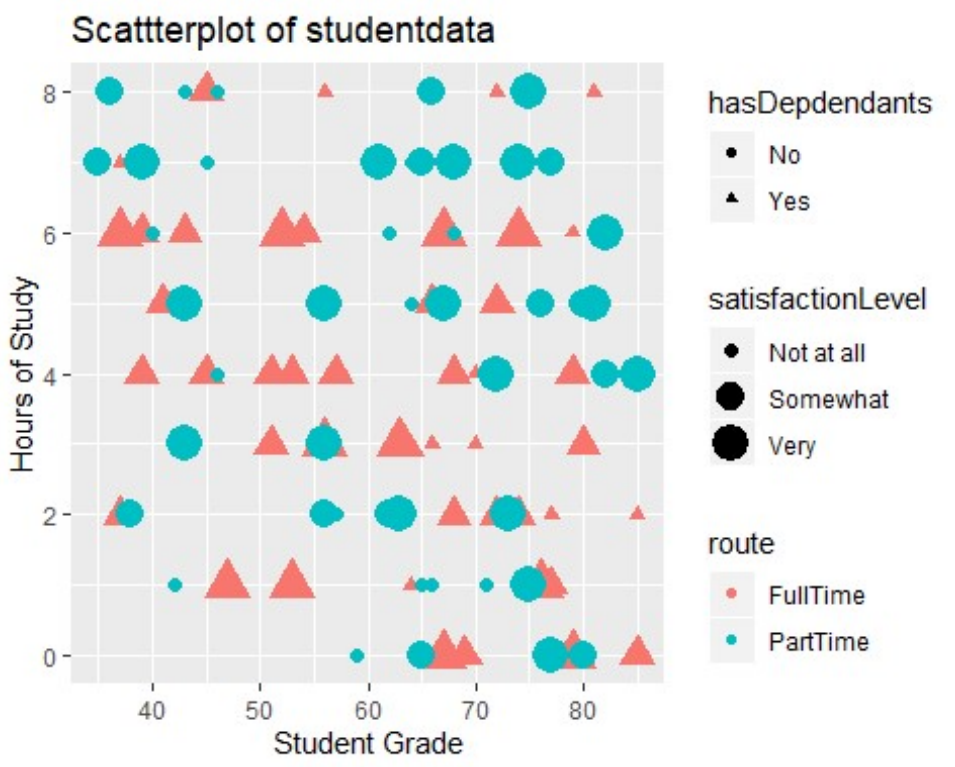


Changing the axis labels and title on a plot

We can change the axis labels and title using the **labs()** command:

labs(x="Student Grade", y="Hours of Study", title = "Scattterplot of student data")

```
library(ggplot2)

ggplot(data=studentData, aes(x=grades,y=hoursOfStudy)) + geom_point(aes(color
= route, size=satisfactionLevel, shape=hasDepdendants)) + labs(x="Student Gra
de", y="Hours of Study", title = "Scattterplot of studentdata")
```

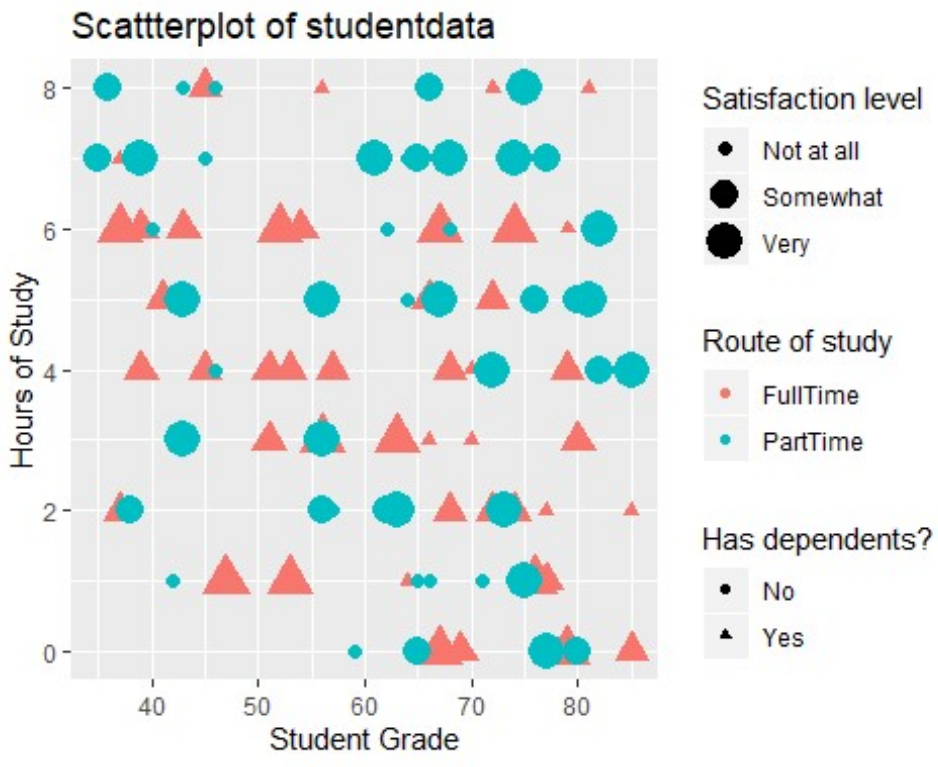


Changing the legend on a plot

To change the legend, we use the **labs()** command too, and reference the relevant property (e.g. size, shape, colour)

labs(x="Student Grade", y="Hours of Study", title = "Scattterplot of student data", color="Route of study", size="Satisfaction level", shape="Has dependents?")

```
library(ggplot2)

  ggplot(data=studentData, aes(x=grades,y=hoursOfStudy)) +
    geom_point(aes(color = route, size=satisfactionLevel, shape=hasDepdendant
s)) +
  labs(x="Student Grade", y="Hours of Study", title = "Scattterplot of studen
tdata", color="Route of study", size="Satisfaction level", shape="Has depende
nts?")
```



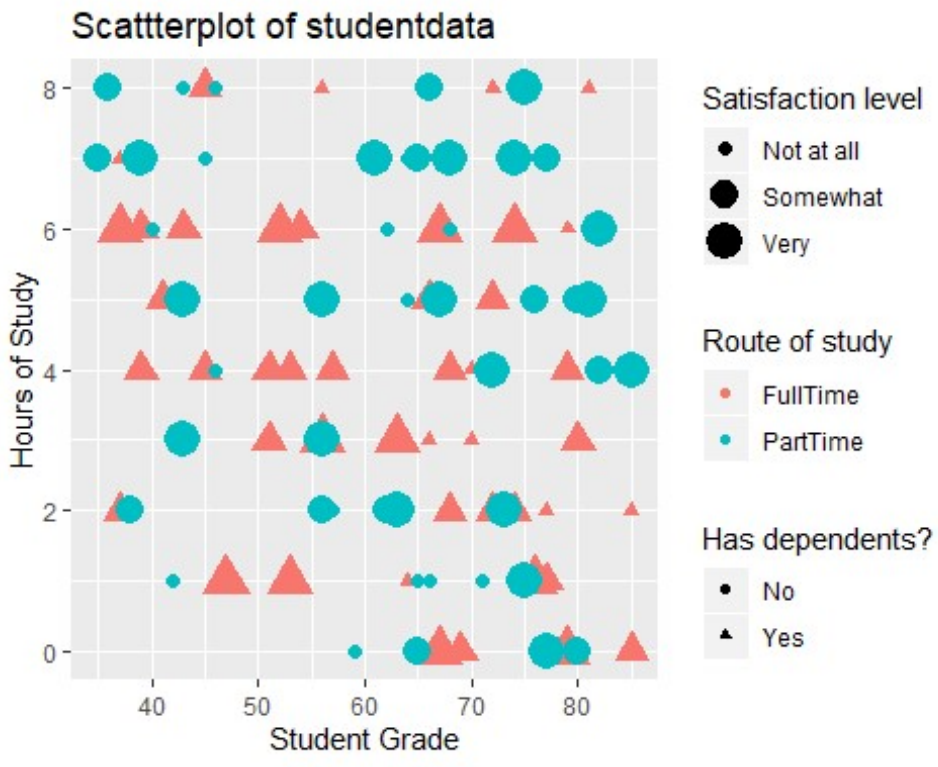
Storing plots to be recalled later

- Plots can be assigned to objects in R and recalled later, just like any other piece of data

```
library(ggplot2)

## Create plot and store it as "myPlot" object

myPlot <- ggplot(data=studentData, aes(x=grades,y=hoursOfStudy)) +
  geom_point(aes(color = route, size=satisfactionLevel, shape=hasDepdendants)
) +
  labs(x="Student Grade", y="Hours of Study", title = "Scattterplot of studen
tdata", color="Route of study", size="Satisfaction level", shape="Has depende
nts?")
```

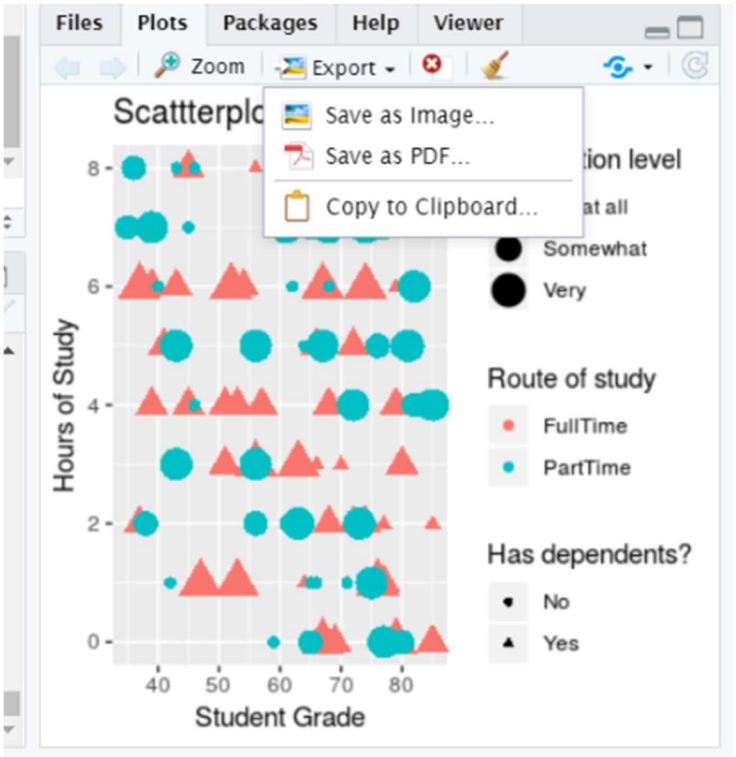
Recalling a stored plot

```
 #Recall myPlot
myPlot
```



Saving plots # 1

- Plots can be save using the **export** button in the plots tab



Plots can also be saved using code

- You might want to include code to save your plot in a script, for example
- This can allow greater control over the output file and plot dimensions:

```
ggsave(plot= myPlot, file="myPlot.pdf", width = 4, height = 4)
ggsave(plot= myPlot, file="myPlot.png", width = 4, height = 4, units="cm", dp
i=320)
```

Plotting with R Exercises

0. Install the ggplot2 package and then load it using the library() command
1. Import the **SalesData.sav** dataset
2. Change the ***married*** variable to a factor
3. Create a plot using the **salesData** dataset - put the ***salary*** variable on the X axis and the ***valueOfSales*** variable on the Y axis.
4. Add the geom "point" and assign colour to the ***married*** variable and size to the ***numberOfVisits*** variable.
5. Rename the plot title, axes labels and legend labels to something appropriate
6. Store this plot as an object named “plot1”
7. Create a different plot using the **salesData** dataset - the ***married*** variable on the X axis and use the *stat_summary()* function to show the mean ***valueOfSales*** on the Y axis, with the geom bar
8. Inside the *stat_summary()* function, add the option fill=“***married***” to the code, to change the fill colour of the bars based on the ***married*** variable
9. Store this plot as a plot named “plot2” 8 Save “plot1” as a pdf file
10. Save “plot2” as a png file