Introduction to GGPLOT2

Hadley Wickham's **ggplot2** package is a very popular alternative to R's base graphics package. The **ggplot2** package is an implementation of the ideas in the book, The Grammar of Graphics, by Leland Wilkison, whose goal was to develop a set of general unifying principles for the visualization of data.

For this reason, **ggplot2** offers a more elegant and arguably more natural approach than does the base R graphics package. The package has a relatively small number of primitive functions, making it relatively easy to master. Through combining these functions in various ways, a very large number of types of graphs may be produced. It is considered especially good in setting reasonable default values of parameters, and much is done without the user's asking. Legends are automatically added to graphs, for instance.

The package is quite extensive (only a few functions, but lots of options). The basic concept of a ggplot2 graphic contains information about the following:

- **The data that you want to plot**: For ggplot(), this must be a data frame.
- **A mapping from the data to your plot:** This usually is as simple as telling ggplot() what goes on the x-axis and what goes on the y-axis. The aes() function is used to set up the mapping.
- **A geometric object, or geom in ggplot terminology:** the geom defines the overall look of the layer (e.g. whether the plot is made up of bar points, or lines)
- **A statistical summary, called a stat in ggplot:** This describes how you want the data to be summarised (e.g. binning for histograms or smoothing to draw regression lines)
-

Installation and Use

If you have not done so already, download and install ggplot2 with the usual install.packages() function, and then at each usage, load via library() i.e..

```
install.packages("ggplot2")
library("ggplot2")
```

Let us read in the numeric weather data and separately the nominal data (change the filepath to suit)

```
weathernom <- read.csv("c:/weathernominal.txt")
weathernum <- read.csv("c:/weathernumeric.txt")
```

Basic Structures

A call to ggplot() takes the form:

ggplot (yourdataframe) or

ggplot (yourdataframe , aes (your args))

e.g.

```
p <- ggplot(weathernom, aes(x = windy))
p + geom_bar()
```

- Weathernom is a data frame with your dataset. The result p is an R S3 object of class ”ggplot”, consisting of a component named data, and other components containing information about the plot. Note that at this point, though, there is no plot;
- Almost every plot maps a variable to x and y, so naming these aesthetics is important. To add additional variables to a plot, we can use other aesthetics like colour, shape, and size.
- One adds features to—or even changes—the plot via the + operator. One can do this repeatedly, e.g. to superimpose several curves on the same graph. Each invocation of + adds a new layer to the graph.

```
p <- ggplot(weathernom, aes(x = windy)) +geom_bar()
p
```

- The function aes() (“aesthetics”) is used to map your data variables to graph attributes. The first one or two arguments specify the data variables to be plotted, in one dimension (say for the remaining arguments are name/value pairs, where the name is an attribute such as colour and the value is a data variable. This could be, for example, to indicate a variable that will determine the colour of any given point in the graph. One can set attributes in this way at various levels:
  - We can set attributes for the entire graph by calling **aes()** within our call to ggplot(), e.g. to specify which variables from our dataset we wish to plot.
  - We can set attributes specific to one of the + actions, such as specifying the fill of the bars in a barchart or point color in a scatterplot.

```
p <- ggplot(weathernom, aes(x = windy))
p + geom_bar(fill="yellow",color="blue")
```

```
-- a scatterplot
p <- ggplot(weathernum, aes(x = humidity, y=temperature ))
p + geom_point(color="green", size=3) + geom_smooth(method="loess")
```

So for instance we could use aes() to specify our data variables either when we call ggplot(), so these variables will be used in all following operations, or when we call, say, geom point(), to indicate data variables for this specific operation.

```
-- A scatterplot with points colour-coded using the Play attribute.
ggplot(weathernum, aes(x=humidity, y=temperature)) + geom_point(aes
(color= play), size =6)
```

There are various types of objects that can be used as the second operand for the + operator.

Examples are:

- **geoms** ("geometrics"): Geometric objects to be drawn, such as points, lines, bars, polygons and text.
- **position adjustments:** For instance, in a bar graph, this controls whether bars should be side by side, or stacked on top of each other.
- **facets:** Specifications to draw many graphs together, as panels in a large graph. You can have rows of panels, columns of panels, and rows and columns of panels.
- **themes:** These can be useful if you do not like the gray background in a graph and want nicer labelling, etc. You can set each of these individually or use one of the built-in themes, or a user-contributed one..

There are a wide selection of geoms. Each geom has a default stat, and each stat has a default geom. In practice, you have to specify only one of these. Here are some of the popular ones

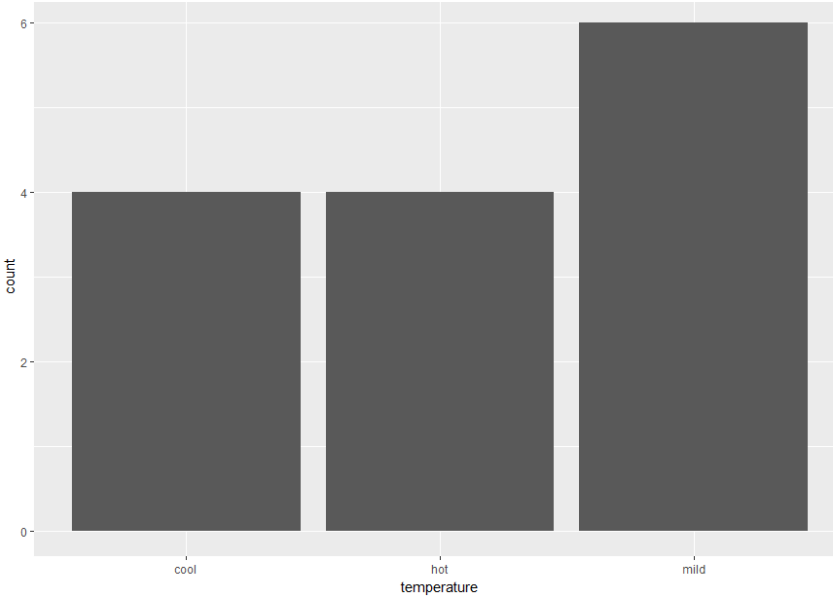
Geom	Description	Default Stat
geom_bar()	Bar Chart	stat_bin()
geom_point()	Scatteplot	stat_identity()
geom_line()	Line Diagram connecting observations in ordered by x-value	stat_identity()
geom_boxplot	Box and Whisker plot	stat_boxplot()
geom_smooth()	Add a smoothed conditioned mean	stat_smooth()
geom_histogram()	An alias for geom_bar() and stat_bin()	stat_bin()

Here is a link to a summary of the layers in ggplot2

Bar Graphs

Bar graphs (or bar charts) are the best way to display categorical variables (one variable). As we know aesthetics are characteristics that each plotted object can have, such as an x-coordinate, a y-coordinate, a color, a shape, and so on. The layers we will use are all geometric representations of the data and have function names that have the form geom_XXX() where XXX is the name of the type of plot. This first example will use geom_bar() for a bar graph. With a bar graph, we set x to be the name of the categorical variable and y is automatically chosen to be the count.

```
g <-ggplot(weathernom, aes(x = temperature))
g + geom_bar()
```



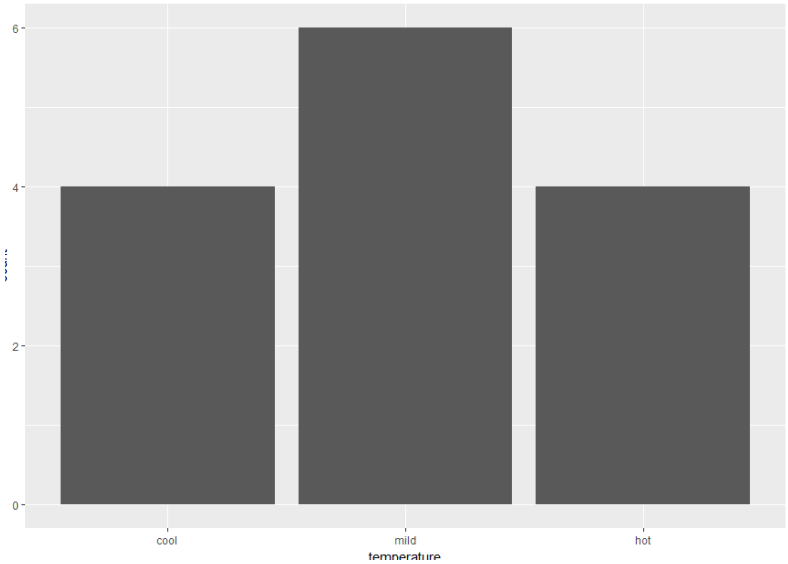
reorder()

The preceding graph would be improved if the order temperature was not alphabetical. Temperature has a natural order (at least in part) from cool to hot. The function reorder() can be used to change the order of the levels of a factor. The way it works is to include a second argument which is a quantitative variable of the same length: the levels are ordered so that the mean value for each group goes from smallest to largest. As there is no variable in the data set that we can be sure will put temperature in the desired order, we will create one with 1 for cool, 2 for mild, and so on. Watch the use of with() and square brackets to select parts of objects. A single = is for assignment or setting the value of an argument to a function. The double == is to check equality. Here is the R code to do it for a temporary variable named temporder that we create and then discard.

```
tempOrder <- rep(0, nrow(weathernom))
tempOrder
 [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0

#Change tempOrder vector positions to values 1,2,3 to represent order
tempOrder[with(weathernom, temperature == "cool")] <- 1
tempOrder[with(weathernom, temperature == "mild")] <- 2
tempOrder[with(weathernom, temperature == "hot")]  <- 3
tempOrder

 [1] 3 3 3 2 1 1 1 2 1 2 2 2 3 2

# change weathernom$temperature to the reordered version and discard
tempOrder. Note levels are introduced
weathernom$temperature = with(weathernom, reorder(temperature,
tempOrder))

rm(tempOrder)
#check the levels

weathernom$temperature
```

Now, redo the plot.

```
ggplot(weathernom, aes(x = temperature)) + geom_bar()
```



Try these variations out:

```
g <-ggplot(weathernom, aes(x = temperature))
g + geom_bar(width=.5)
g + geom_bar(width=.5) + coord_flip()
g + geom_bar(fill="cyan", colour="darkgreen")
```

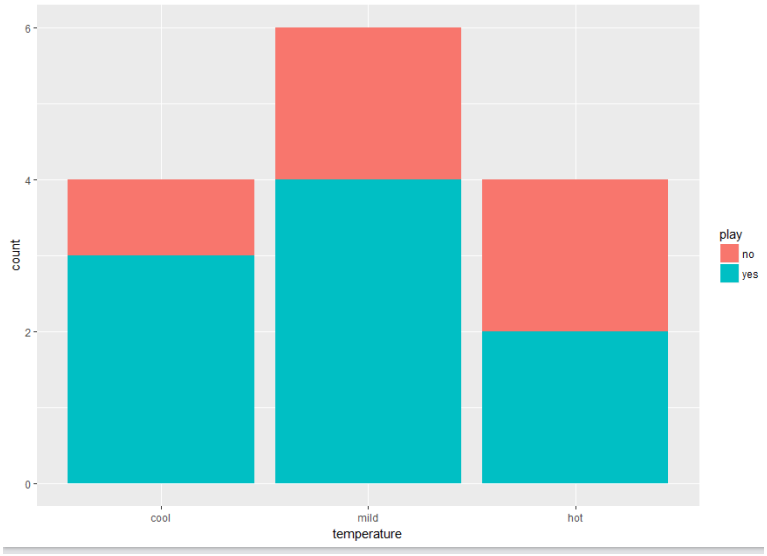
```
# plyr allows the user to split a data set apart into smaller subsets, app
ly methods to the subsets, and combine the results. When the data contains
y values in a column, use stat="identity"
library(plyr)


# Calculate the mean temperature when games are played and when games were
not played and then create a bar chart

mtemp <-ddply(weathernum, "play",summarise,mtemp=mean(temperature))

ggplot(mtemp, aes(x = play, y = mtemp)) + geom_bar(stat = "identity
")


ggplot(mtemp, aes(x = play, y = mtemp, fill=play)) + geom_bar(stat
= "identity")
```

Note: The heights of the bars commonly represent one of two things: either a count of cases in each group, or the values in a column of the data frame. By default, `geom_bar` uses `stat="bin"`. This makes the height of each bar equal to the number of cases in each group, and it is incompatible with mapping values to the `y` aesthetic. If you want the heights of the bars to represent values in the data, use `stat="identity"` and map a value to the `y` aesthetic.

Bar plots for 2 Categorical Variables

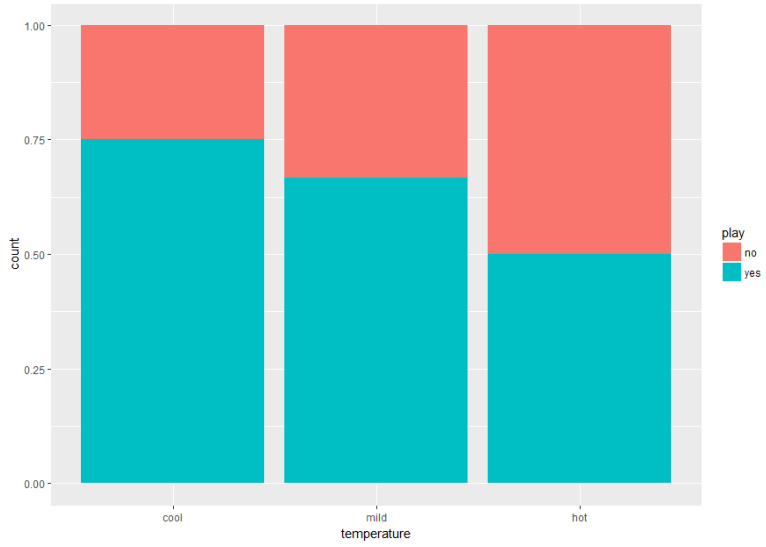
If we want to examine the **play** distribution by **temperature**, we can tabulate it and then make a stacked bar plot. We want to see the distribution of **play** by **temperature**, but the alternative plot is also legitimate.

```
with(weathernom, table(play, temperature))
     temperature
play  cool mild hot
  no     1    2   2
  yes    3    4   2
ggplot(weathernom, aes(x = temperature, fill = play)) + geom_bar()
```

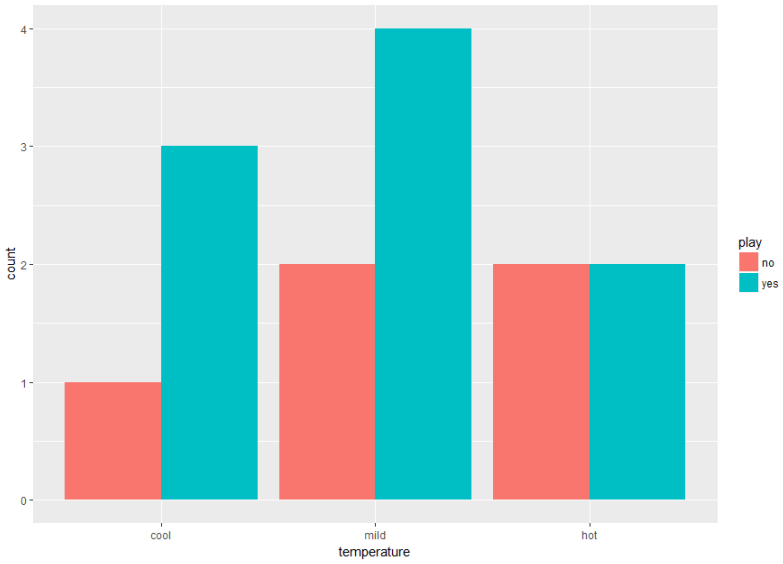


If we want to see the **proportion** of play and don't play for each temperature, we change the position attribute of the bar plot to "fill".

```
ggplot(weathernom, aes(x = temperature, fill = play)) +
geom_bar(position="fill")
```



Now change the position to "dodge" if we want to have separate bars for play (no, yes) within each temperature. (The bars dodge each other to avoid overlapping.)



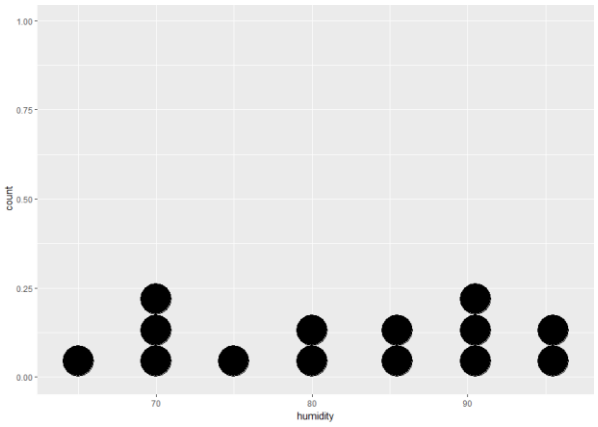
One Quantitative Variable

There are multiple ways to graphically display the distribution of a single quantitative variable. They differ in how clearly they show various features of the distribution.

Dot plots

For small data sets, a dot plot is an effective way to visualize all of the data. A dot is placed at the appropriate value on the x axis for each case, and dots stack up. Here is how to make a dotplot with ggplot2 for the weathernum data humidity variable.

```
ggplot(weathernum, aes(x = humidity)) + geom_dotplot(binwidth = 2)
```



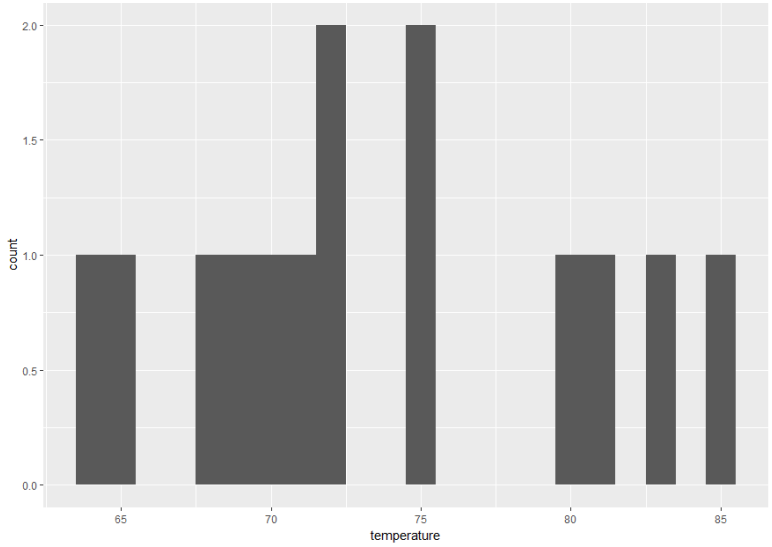
Here are more examples!

```
# y axis isn't really meaningful, so hide it

ggplot(weathernum, aes(x = temperature))
+ geom_dotplot(binwidth = 2)
+ scale_y_continuous(NULL, breaks = NULL)

ggplot(weathernum, aes(x = temperature, fill=play))
+ geom_dotplot(binwidth = 2)
+ scale_y_continuous(NULL, breaks = NULL)

ggplot(weathernum, aes(x =humidity, fill = factor(outlook)))
+ geom_dotplot(binwidth = 2)

# Overlap dots vertically
ggplot(weathernum, aes(x = temperature))
+ geom_dotplot(binwidth = 1.5, stackratio = .8)

# Expand dot diameter
ggplot(weathernum, aes(x = temperature))
+ geom_dotplot(binwidth = 1.5, dotsize = 1.25)

# Stacking multiple groups, with different fill
ggplot(weathernum, aes(x = humidity, fill = play))
+ geom_dotplot(stackgroups = TRUE, binwidth = 1,
  binpositions = "all")

# Try these out

ggplot(weathernum, aes(x = humidity, fill = play))
+ geom_dotplot(stackgroups = TRUE, binwidth = 1, method =
"histodot")
+ scale_y_continuous(NULL, breaks = NULL)

ggplot(weathernum, aes(x = 1, y=humidity, fill =outlook))
+ geom_dotplot(binaxis = "y", stackgroups = TRUE, binwidth = 1,
  method = "histodot")
+ scale_x_continuous(NULL, breaks = NULL)
```

Histograms

Another popular way to graph a single quantitative variable is with a histogram. Just use geom_histogram() instead. There are ways to specify the breakpoints. It is often useful to specify the binwidth and/or origin manually.

```
ggplot(weathernum, aes(x = temperature)) + geom_histogram(binwidth = 1)
```



In the previous plot, the y-axis corresponds to the count of observations in each bin. Another common way to present histograms is so that the total area is one, so that the area of a bin represents the proportion of data in the corresponding interval. Adding an aesthetic y=..density.. makes this change.

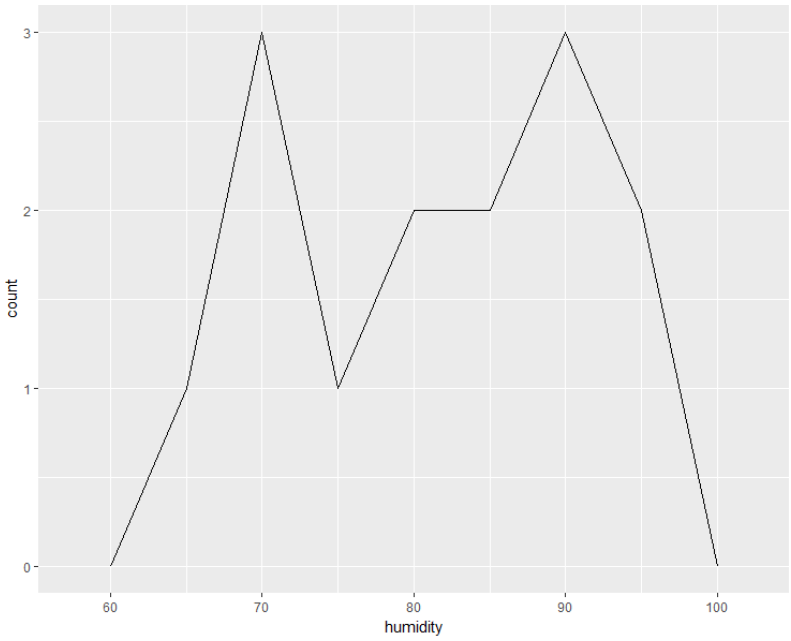
```
ggplot(weathernum, aes(x = temperature))
+ geom_histogram(binwidth = 1, aes(y=..density..))
```

```
Try these out!
ggplot(weathernum, aes(humidity, fill = play))
+ geom_histogram(binwidth = 5)

ggplot(weathernum,aes(temperature, fill = outlook))
+ geom_histogram(binwidth = 5)

g <- ggplot(weathernum, aes(x=humidity))
g + geom_histogram(binwidth = 5)
g + geom_histogram(aes(y = ..density..),binwidth=5)+ geom_density()


#adjusting colour and transparency and a different way to adjust the x axis
g +geom_histogram(breaks=seq(60, 100, by = 5),
                      col="red",
                      fill="green",
                      alpha = .4)
+ labs(title="Histogram for humidity")
+ labs(x="Humidity", y="Count")
+ xlim(c(60,100))
+ ylim(c(0,5))
```

Frequency Plots

Freqency polygons also show the distribution of a single numeric variable. They provide more information about the distribution of a single group than boxplots do, at the expense of needing more space.

```
ggplot(weathernum, aes(humidity)) + geom_freqpoly( binwidth=5)
```



We can compare frequency polygons for target variable play

```
ggplot(weathernum, aes(humidity, colour = play)) +
geom_freqpoly(binwidth = 5)
```

Both histograms and frequency polygons work in the same way: they bin the data, then count the number of observations in each bin. The only difference is the display: histograms use bars and frequency polygons use lines. You can control the width of the bins with the binwidth argument (if you dont want evenly spaced bins you can use the breaks argument). It is **very important** to experiment with the bin width. The default just splits your data into 30 bins, which is unlikely to be the best choice. You should always try many bin widths, and you may find you need multiple bin widths to tellthe full story of your data.

```
ggplot(weathernum, aes(humidity)) + geom_freqpoly( binwidth=2.5)

ggplot(weathernum, aes(humidity)) + geom_freqpoly( binwidth=1)
```

(An alternative to the frequency polygon is the density plot, geom_density(). Density plots are harder to interpret since the underlying computations are more complex. They also make assumptions that are not true for all data, namely that the underlying distribution is continuous, unbounded, and smooth.)

To compare the distributions of different subgroups, you can map a categorical variable to either fill (for geom_histogram()) or colour (for geom_freqpoly()). It is easier to compare distributions using the frequency polygon because the underlying perceptual task is easier. You can also use

facetting: this makes comparisons a little harder, but it's easier to see the distribution of each group.

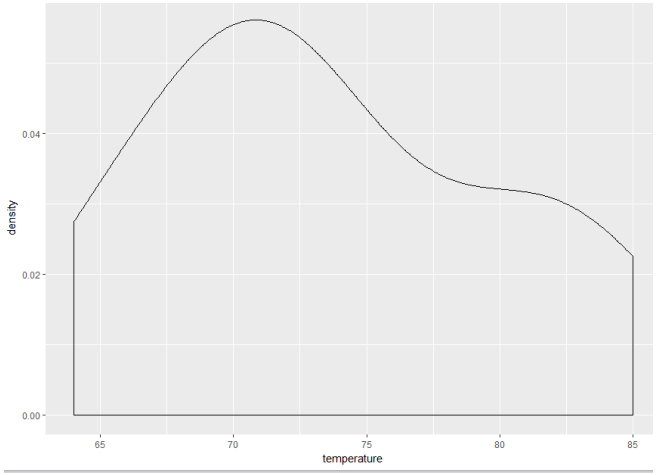
```
ggplot(weathernum, aes(humidity, colour = outlook))
+ geom_freqpoly(binwidth = 5)
```

```
ggplot(weathernum, aes(humidity, fill = outlook))
+ geom_histogram(binwidth = 5)
+ facet_wrap(~outlook, ncol = 1)
```

Density Plots

Density plots are similar to histograms on a density scale, but instead of fixed bins or intervals with jumps at the boundaries, are smooth. The argument adjust to geom_density regulates how smooth the density estimate is, with larger values resulting in smoother graphs.

```
ggplot(weathernum, aes(x = temperature))  + geom_density()
```

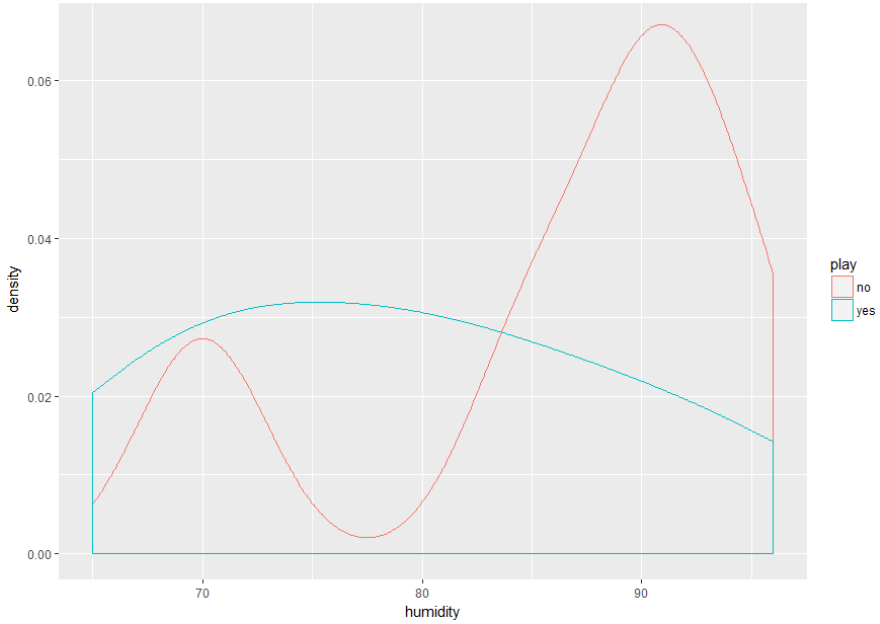


Try these out!

```
ggplot(weathernum, aes(x = temperature)) + geom_density(adjust=1)
ggplot(weathernum, aes(x = temperature)) + geom_density(adjust=.25)
```

Because density plots are lines, a few of them can be drawn together on the same plot, making the m easy to compare. Here is an example that uses the color aesthetic to distinguish play = no and play=yes.

```
ggplot(weathernum, aes(humidity, fill = play))
+ geom_density(alpha = 0.1)
+ xlim(60, 100)
```



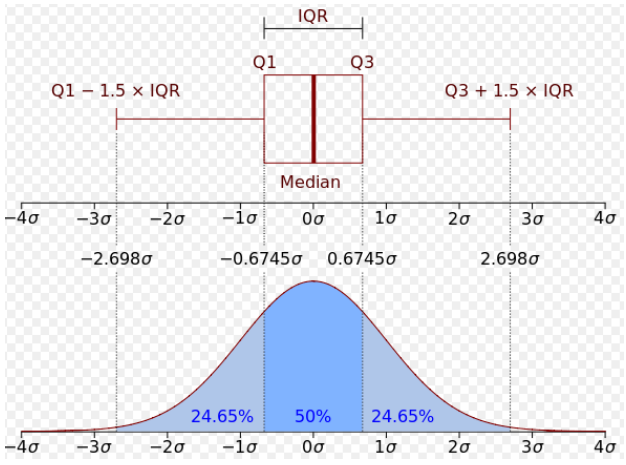
You can use position="fill" to produce a conditional density estimate.

```
ggplot(weathernum, aes(humidity ,fill = play))
+ geom_density(position = "fill")
```

Density plots and histograms can show many features of a distribution, but can be too messy if there are many groups.

Box and Whisker Plots

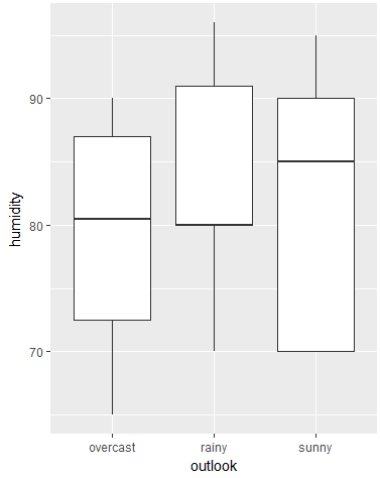
Box and whisker plots are highly summarized representations of the data, essentially condensing the entire sample to a five number summary, plus locations of outliers as defined by extreme distance 11 outside the range of the middle half of the data.



While many statistical software packages use boxplots to summarize single variables, ggplot2 by construction only uses box plots for comparing two or more distributions. It is, however, possible to force the functions to handle a single variable by using an artificial grouping variable. First, here

is an example of side-by-side boxplots with the grouping variable on the x-axis and the quantitative variable on the y-axis.

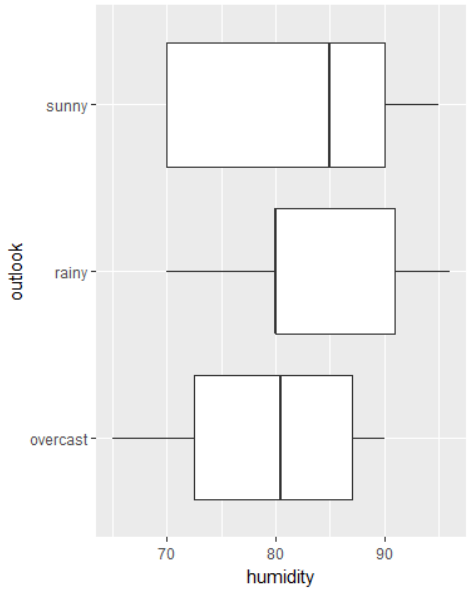
```
g <-ggplot(weathernum, aes(x = outlook, y=humidity))
g + geom_boxplot()
```



Explain what the boxplot is displaying for overcast.

One additional layer command flips the coordinates so that the quantitative variable is horizontal (which often makes comparisons about centre and spread easier to perceive).

```
g + geom_boxplot() +coord_flip()
```



Some variations

```
g + geom_boxplot() + geom_jitter(width = 0.2)
```

```
g + geom_boxplot(fill = "white", colour = "#3366FF")
g + geom_boxplot(outlier.colour = "red", outlier.shape = 1)
```

Note there are no outliers

Quantitative Summaries

The function to compute the quantiles in R is called quantile(). The mean height of all students and computed separately for females and males are computed like this

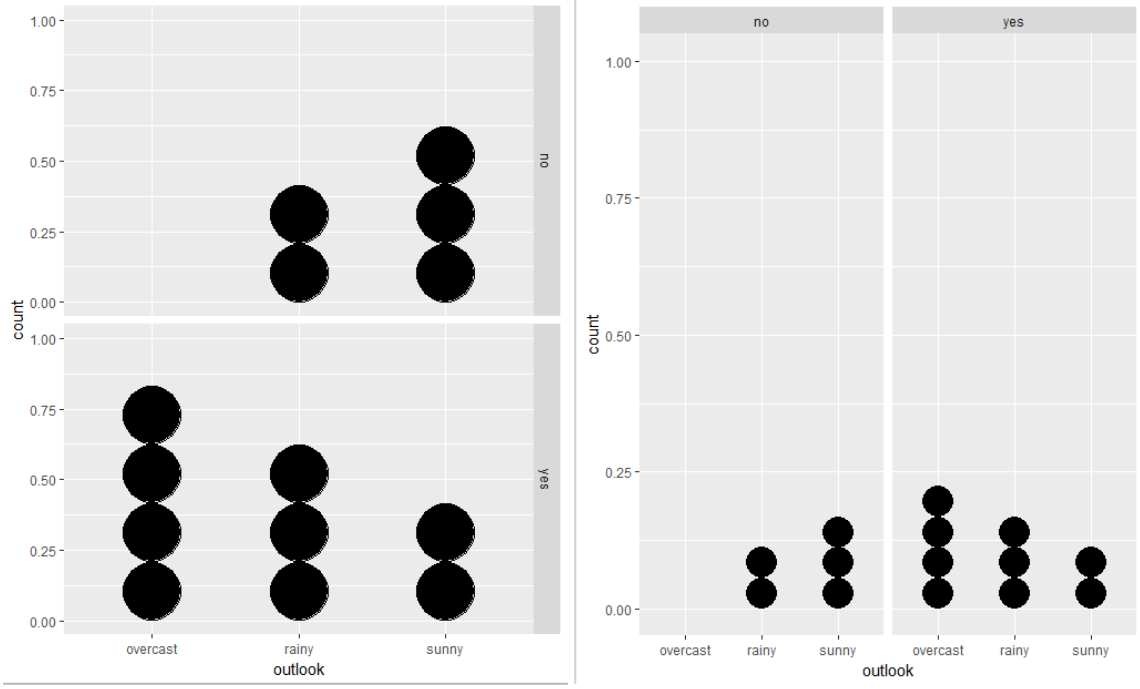
```
quantile(weathernum$humidity)
   0%   25%   50%   75%  100%
65.00 71.25 82.50 90.00 96.00
```

Relationships Between a Categorical and Quantitative Variable

It is very common in statistical settings to be interested in how a quantitative variable differs among groups. We have seen temperature versus play = yes/no. Now, we will review graphs and summary calculations for this situation.

Faceting One thing ggplot2 does very well is to make it easy to split a variable among the groups of a quantitative variable to show univariate displays for each group separately using common scaling to allow easy visual comparisons. For something different, let's look at the distributions of the number of hours of sleep students get each night versus their level in school. Let us first show this as a dotplot. Here is the command that shows outlook against play for the entire sample. I have adjusted the dot size. Games seem to be played more when the outlook is overcast and rainy.

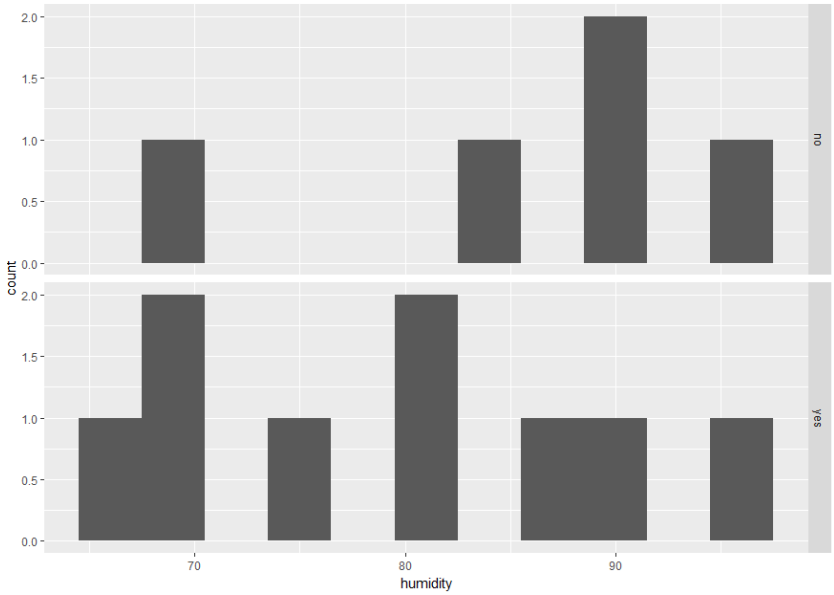
```
ggplot(weathernom, aes(x = outlook))
+ geom_dotplot(dotsize = 0.2, binwidth=1)
+ facet_grid( play~ .)    # try facet_grid( ~play)    for different orientation
```



There are two types of facetting: grid which we have seen and wrapped. To facet a plot you simply add a facetting specification with facet_wrap(), which takes the name of a variable preceded by ~.

Stacking histograms is another way to compare distributions of a quantitative variable for different groups. Here is code to compare humidity when games are not played

```
ggplot(weathernum, aes(x = humidity))
+ geom_histogram(binwidth=3)
+ facet_grid( play ~ .)
```



Two Quantitative Variables

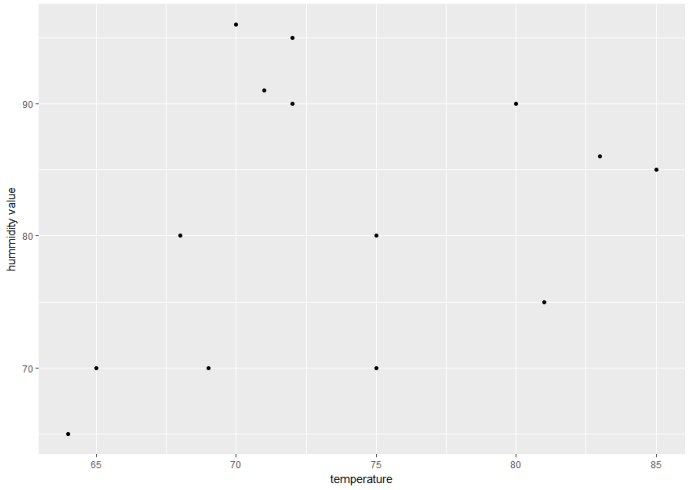
A scatterplot is the most useful way to display two quantitative variables. More information can be a dded to plots by using different colors or symbols for different groups. Overlapping points can be ha ndled by jittering (moving the positions a small random amount) or by using partially opaque poi nts.

For numerical summaries of pairs of quantitative variables, we will look at the correlation coefficient as a measure of the strength of the linear relationship between two variables . Here are some examples.

Scatterplots

In the weather data set, we can examine the relationship between temperature and humidity

```
ggplot(weathernum, aes(x=temperature,y=humidity)) + geom_point() +
ylab('hummidity value')
```



There is a weak positive correlation between the variables. We can measure this with the Pearson'scorrelation coefficient.

```
with(weathernum, cor(temperature,humidity))
[1] 0.3150818
```

If the graph has a fair amount of overplotting (not so in this case). The following command will jitter all of the points in a controlled way. We choose to jitter only the width by a small amount and not the height so the humidity remains accurate and we can still see the true humidity

```
ggplot(weathernum, aes(x=temperature,y=humidity))
+ geom_point(position= position_jitter(w=0.1, h=0))
+ ylab('Humidity Value')+ xlab('Degrees Farenheit')
```

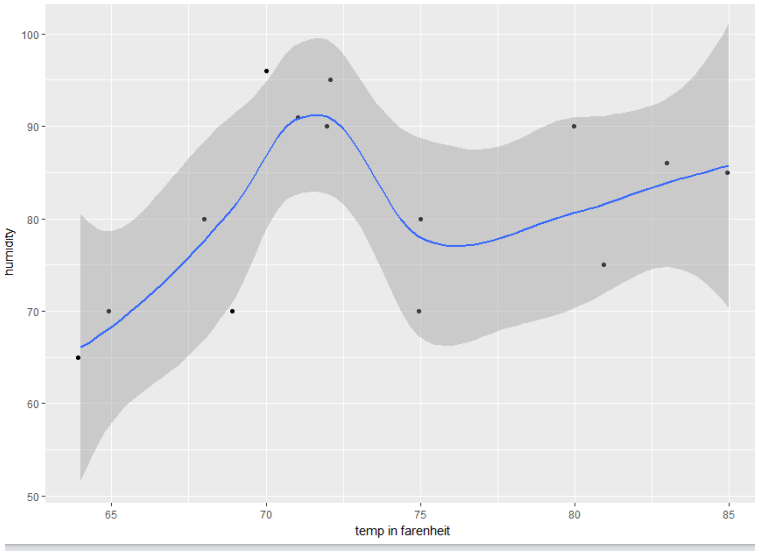
Least-squares Lines

Regression is a special case of a linear model. The R function for fitting a linear mode is lm() and we can use it like this. The function coef() returns the estimated coefficients of the fitted linear model. Below, fit is the name for the object that contains the fitted model.

```
fit = lm(humidity ~ temperature, data = weathernum)
coef(fit)
(Intercept) temperature
 45.3625954   0.4931298
```

We can add the line to the plot using geom_smooth(). The default behaviour is to add a locally smooth regression line with a shaded gray area to represent pointwise 95 percent confidence regions for the true mean humidity as a function of temperature. If you are not interested in the confidence interval, turn it off with geom_smooth(se= FALSE). We first show this plot and then the options to override the default and just plot the simple regression line.

```
ggplot(weathernum, aes(x=temperature,y=humidity))
+ geom_point(position=position_jitter(w=0.1,h=0))
+ geom_smooth()
+ xlab('temp in farenheit')
```

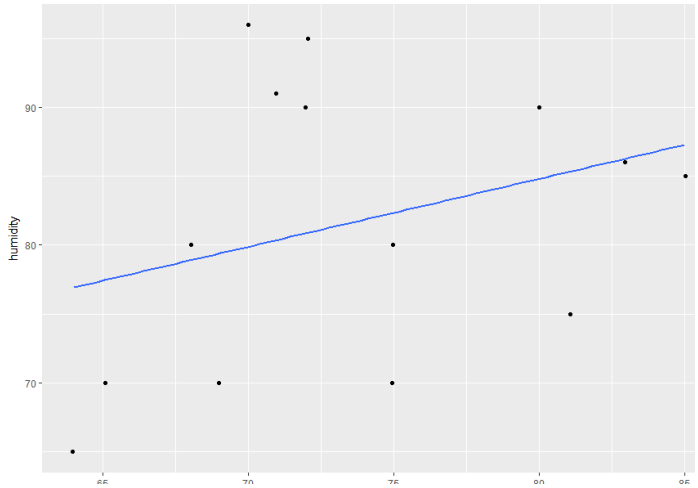


An important argument to geom_smooth() is the method, which allows you to choose which type of model is used to fit the smooth curve:
method = "loess", the default for small n, uses a smooth local regression (as described in ?loess). The wiggliness of the line is controlled by the span parameter, which ranges from 0 (exceedingly wiggly) to 1 (not so wiggly)!

```
ggplot(weathernum, aes(x=temperature,y=humidity))
+ geom_point(position=position_jitter(w=0.1,h=0))
+ geom_smooth(span=.5)
+ xlab('temp in farenheit')
```

Note: Loess does not work well for large datasets (it's *O*(*n*2) in memory), so an alternative smoothing alg-o rithm is used when *n* is greater than 1,000. method = "gam" fits a generalised additive model provided by t he **mgcv** package. You need to first load mgcv, then use a formula like formula = y~ s(x) or y ~ s(x, bs = "cs") (for large data). This is what ggplot2 uses when there are more than 1,000 points.

```
ggplot(weathernum, aes(x=temperature,y=humidity))
+ geom_point(position=position_jitter(w=0.1,h=0))
+ geom_smooth(method="lm", se=FALSE)
+  xlab('temp in farenheit')
```



Let us look at a the a subset of the points where outlook is rainy or overcast

```
g <-ggplot(subset(weathernum, outlook %in% c("overcast", "rainy")))
g + geom_point(aes(x= temperature, y=humidity, color=play))
```

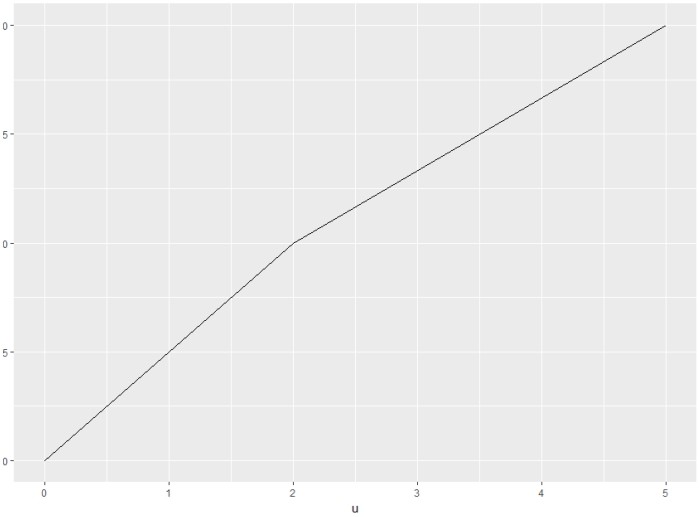
You should plot the temperature and humidity as above but looks at the different outlooks (i.e. sunny, overcast and rainy) where outlook is rainy or overcast. What can you conclude?

Example: Simple Line Graphs

Let us create some data in a data frame and plot it using a geom_line()

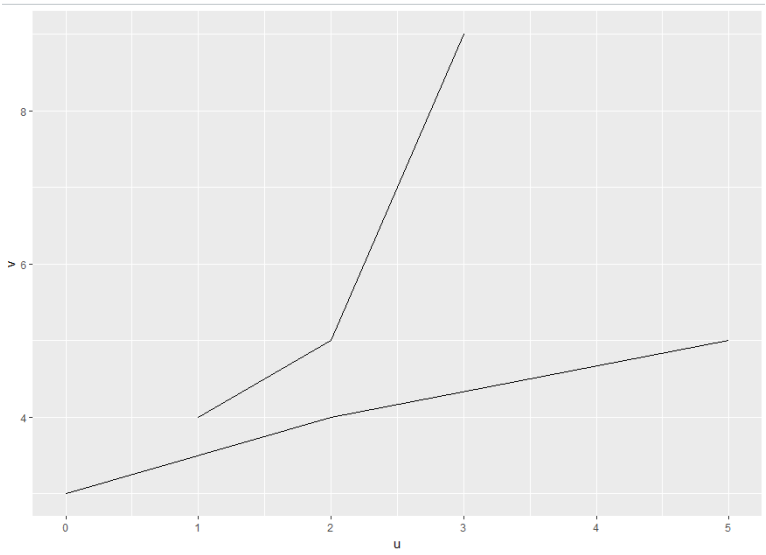
```
u<- c(0,2,5)
v<- c(3,4,5)
df1 <- cbind(u,v)
df1 <-data.frame(df1)

ggplot(df1) + geom_line (aes ( x=u , y=v ) )
```

Here aes() was called from geom line() rather than from ggplot(), so as to apply just to this line. The result is



Now let us add a second line from a different data frame.

```
w<- c(1,2,3)
z<- c(4,5,9)
df2 <- cbind(w,z)
df2 <-data.frame(df2)
ggplot(df1) + geom_line(aes(x=u ,y=v))+ geom_line (data=df2,
aes(x=w,y=z))
```



It worked as long as we specified data in the second line. Note that ggplot2 automatically adjusted that second graph, to make room for the "taller" second line.

A few Exercises - If you have time

Examine the housing data (see moodle)

housing <- read.csv(file = "E:/R/Housing.csv")

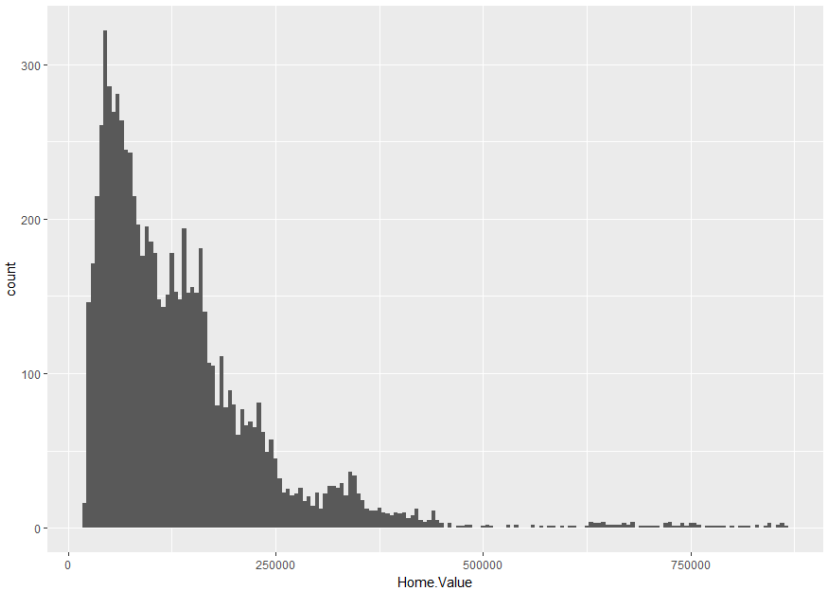
1. Using a histogram, examine the home value variable. Note any observations.
2. Examine the home value and land value variables by using a scatterplot. Is there any correlation? Verify your answer mathematically
3. Examine these state variable CA, DC and HI for home value and land value. Note any observations
4. Create a scatterplot that shows the land price index against the home price index for the years following 2008. Colour code it by state.
5. Using two line graphs, plot year on the x axis and home value on the y-axis for CA and them AK. What are your findings?

Now let us use the cars dataset cars2 <- read.csv("E:/…/Rdatasets/cars2.txt")
[Check out scale_y_continuous() function]

6. Create an overlayed bar chart for car cylinders to get a sense of the counts based on the different car brands.
7. Normalise the barchart (show proportion on the y axis) now showing the proportions for no of cylinders based on brand

1. Using a histogram, examine the home value variable. Note any observations.

```
p <-ggplot(housing)
p  + geom_histogram(aes(x = Home.Value), binwidth=5000)
```



Positively skewed with most house under 250K

2. Examine the home value and land value variables. Use a scatterplot. Is there any correlation? Verify your answer mathematically

You should notice there is a strong positive correlation

```
with(housing, cor(Home.Value, Land.Value))
```

```
p <-ggplot(housing)
p + geom_point ( aes ( x= Home.Value , y= Land.Value))
```

Here there are three data variables informing aes(): Home Value, Land Value and State. The argument color here means that we want the state to be used for color coding the points.

```
p <-ggplot(housing)
p + geom_point ( aes ( x= Home.Value , y= Land.Value, color=State) )
```

We can examine the correlation for each state separately

```
p <-ggplot(housing)
p + geom_point ( aes ( x= Home.Value , y= Land.Value))

p <-ggplot(housing)
p + geom_point ( aes ( x= Home.Value , y= Land.Value, color=State)) +
facet_wrap(~State)
```

3 Examine these state variable CA, DC and HI for home value and land value. Note any observations

We can also look at certain states as follows:

```
p <-ggplot(subset(housing, State %in% c("CA", "DC","HI")))
p +geom_point(aes(x= Home.Value, y=Land.Value, color=State))
```

We can also subset the data like this

```
p <-ggplot(subset(housing, State %in% c("CA")))
p +geom_point(aes(x= Home.Value, y=Land.Value, color=State))
```

We can also look at correlations with state as follows:

```
CA<- subset(housing, State %in% c("CA"))
with(CA, cor(Home.Value, Land.Value))
```

4. Create a scatterplot that shows the land price index against the home price index for the years following 2008. Colour code it by state.

```
with(housing, cor(Land.Price.Index, Home.Price.Index))
```

```
p <- ggplot(subset(housing, Year > 2008))
p +geom_point(aes(x= Land.Price.Index, y=Home.Price.Index, color=Year))
```

```
ggplot(housing, aes(year, Land.Value)) + geom_line()
ggplot(economics_long, aes(date, value01, colour = variable)) +
  geom_line()
```

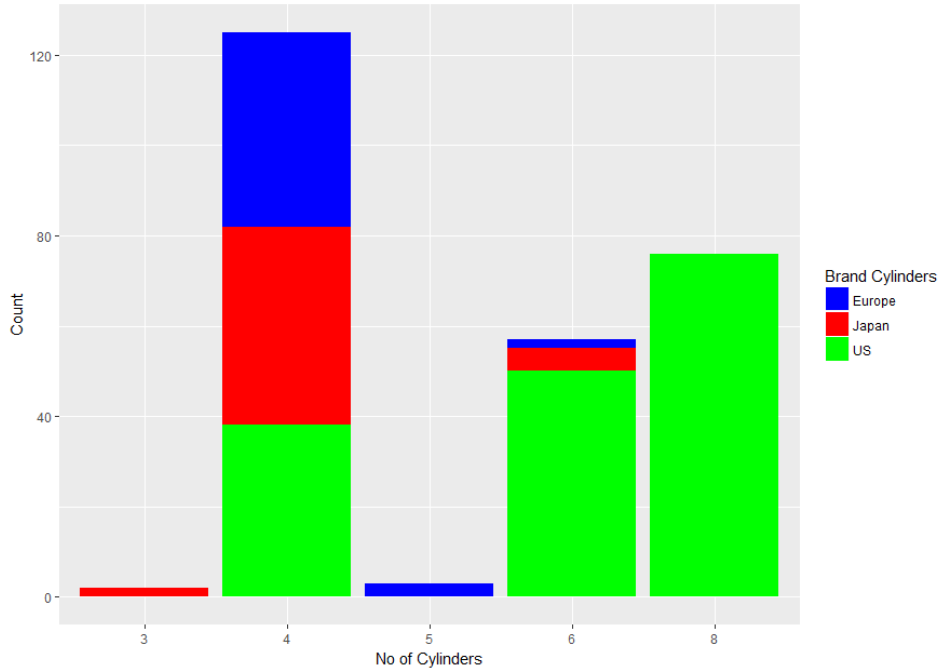
5. Using two line graphs, plot year on the x axis and home value on the y-axis for CA and them AK. What are your findings

```
ggplot(subset(housing, State %in% c("CA")), aes(Date, Home.Value)) +
geom_line()

ggplot(subset(housing, State %in% c("AK")), aes(Date, Home.Value)) +
geom_line()

   with(subset(housing, State %in% c("CA")), cor(Home.Value, Land.Value))
   with(subset(housing, State %in% c("AK")), cor(Home.Value, Land.Value))
```

Overall there is a strong correlation between date and home value for both states with CA close to a perfect correlation. CA values plateaued in the 90s and there was a dip from roughly 2007 on. In AK ibn the late '80s until 1990 there was a drop in value as well a small drop in the late 2000s.

6. Create an overlayed bar chart for car cylinders to get a sense of the counts based on the different car brands.

```
ggplot() + geom_bar(data = cars2, aes(x = factor(cylinders), fill = factor(brand)), position = "stack") +
scale_x_discrete("No of Cylinders") +
scale_y_continuous("Count") +
guides(fill=guide_legend(title="Brand Cylinders")) +
scale_fill_manual(values=c("blue", "red", "green"))
```



6. Normalise the barchart (show proportion on the y axis) now showing the proportions for no of cylinders based on brand

```
ggplot() + geom_bar(data = cars2, aes(x = factor(cylinders), fill = facto
r(brand)), position = "fill") +    scale_x_discrete("No of Cylinders") +
scale_y_continuous("Proportion") +
guides(fill=guide_legend(title="Brand Cylinders")) +
scale_fill_manual(values=c("blue", "red", "green"))
```