- Consider an additional if-else exercise.

*Write an if statement that checks an integer vector **x** and prints "You shouldn't!" if the minimum value of **x** is < 1940, otherwise prints "No-way!" if the maximum value of **x** is > 1999 and prints "Cool!" if all the values of **x** are between 1940 and 1999.*

  - Check your code with different inputs to catch possible errors.

  - Can you now modify your code to print "You should not!" and "No-way!" both when appropriate?

12. Data and Plotting Example with Some Data Cleaning

It is critical to perform an initial data analysis of any real data.

Calculate numerical summaries:
- means
- standard deviations (SDs)
- maximum and minimum, correlations
- anything else that may be appropriate.

Construct the appropriate plots.
- For a single variable, consider boxplots, histograms, density plots, etc.
- For two variables, to explore possible relationships, scatterplots.

- For three or more variables, you can construct interactive and dynamic graphics.

Look for outliers, data-entry errors, skewness, unusual distributions, structure.

Make sure to **clean the data of any errors**.

- This may be very time consuming.

Kidney Example

The National Institute of Diabetes and Digestive and Kidney Diseases conducted a study on 768 adult female Pima Indians living near Phoenix. The following variables were recorded: *number of times pregnant*, *plasma glucose* concentration at 2 hours in an oral glucose tolerance test, *diastolic blood pressure* (mmHg), *triceps skin fold thickness* (mm), 2-hour serum *insulin* (mu U/ml), *body mass index* (weight in kg/(height in m2)), diabetes pedigree function, age (years) and a test whether the patient showed signs of diabetes (coded zero if negative, one if positive). The data may be obtained from the UCI Machine Learning Repository [http://archive.ics.uci.edu/ml].

This dataset is also available in the R package *faraway* by Julian Faraway. This example is loosely following his book LMR (p.13).

In principle, we should know more for the purpose of the study and much more about how the data were collected.

But let's do some analysis as an illustration here.

```
require(faraway) #load package faraway only
# if the dataset only if it is not
# already loaded in the environment

data(pima) # load the data from Faraway

head(pima) # first six rows of pima
  pregnant glucose diastolic triceps insulin
1        6     148        72      35       0
...
6        5     116        74       0       0
```


Recall, we can study the data with `str()`:

```
str(pima)
data.frame':  768 obs. of  9 variables:
 $ pregnant : int  6 1 8 1 0 5 3 10 2 8 ...
 $ glucose  : int  148 85 183 89 137 116 78 115 197
125 ...
 $ diastolic: int  72 66 64 66 40 74 50 0 70 96 ...
 $ triceps  : int  35 29 0 23 35 0 32 0 45 0 ...
 $ insulin  : int  0 0 0 94 168 0 88 0 543 0 ...
 $ bmi      : num  33.6 26.6 23.3 28.1 43.1 25.6 31
35.3 30.5 0 ...
 $ diabetes : num  0.627 0.351 0.672 0.167 2.288 ...
 $ age      : int  50 31 32 21 33 30 26 29 53 54 ...
 $ test     : int  1 0 1 0 1 0 1 0 1 1 ...
```


The `summary` command is a quick way to get the univariate five-number summary information and mean for each quantitative variable:

```
summary(pima)
     pregnant        glucose   ...
 Min.   : 0.00   Min.   :  0
 1st Qu.: 1.00   1st Qu.: 99
 Median : 3.00   Median :117
 Mean   : 3.85   Mean   :121
 3rd Qu.: 6.00   3rd Qu.:140
 Max.   :17.00   Max.   :199
```


- Look for anything unusual or unexpected, perhaps indicating a data-entry error.
- The minimum `diastolic` blood pressure is zero! (That's often an indication of a problem).

```
sort(pima$diastolic)
 [1]   0   0   0   0   0   0   0   0   0   0
[11]   0   0   0   0   0   0   0   0   0   0
[21]   0   0   0   0   0   0   0   0   0   0
[31]   0   0   0   0   0  24  30  30  38  40
```

The first 35 values are zero—there is a problem.

- It seems that 0 was used in place of a *missing value*.

This is very bad since 0 is a number, this problem may be easily overlooked, which can lead to faulty analysis!

- This is why we must check our data carefully for potential problems and things that don't make sense.
  - Fact: data collection and curation could be 90% of the effort or more for many projects.

Recall, the value for missing data in R is `NA`.

Several variables share this problem. Let's set the 0s that should be missing values to `NA`.

```
pima$diastolic[pima$diastolic == 0]  <- NA
pima$glucose[pima$glucose == 0] <- NA
pima$triceps[pima$triceps == 0]  <- NA
pima$insulin[pima$insulin == 0] <- NA
pima$bmi[pima$bmi == 0] <- NA
```

The variable `test` is a categorical variable, not numerical.

- R thinks it is a numerical variable but that is wrong.
- Recall, in R a categorical variable is a `factor`.

- We need to convert this variable to a factor.

```
pima$test <- factor(pima$test)

summary(pima$test)
  0   1
500 268
```

500 of the cases were negative and 268 were positive.

We can provide *more descriptive* labels using the `levels` function.

```
levels(pima$test) <-c("negative","positive")
```

Now the data is cleaned up…

Please note that this was relatively "easy" here as it is ultimately a pre-processed textbook example data.

Let's create some plots.

A histogram of diastolic blood pressure.

```
hist(pima$dias, xlab="Diastolic")
```

Recall, we use the `$` symbol to access the `dias` variable in the `pima` dataframe.

- This isn't a big deal for a few variables, but if we need to access multiple variables in a dataframe, this can be laborious.

An alternative way of doing this is to use the `with` function.

- The first argument to the `with` function is the dataframe of reference.

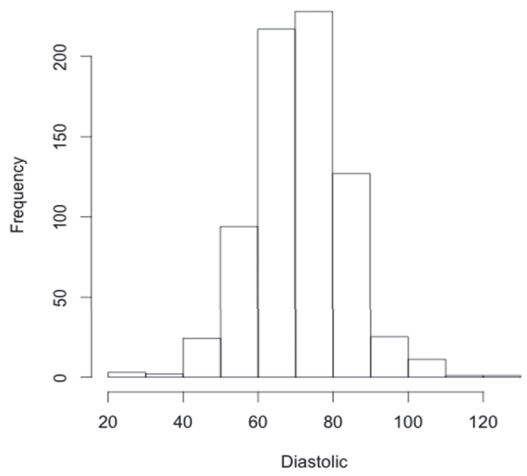
- The second argument is the command you want to execute, referencing the variables in the data frame directly instead of with the $ symbol.

```
with(pima, hist(diastolic)) #alternative 1
```

Recall, another way to do this is to use the `attach()` function, selecting the data frame for all the calls until you `detach()`.

```
attach(pima)     # alternative 2
hist(diastolic)
detach(pima)
```

`attach()` is useful if you need to play with a dataset but one often forgets to detach and some names can overlap too.



The histogram is approximately bell-shaped and roughly centered around 70.

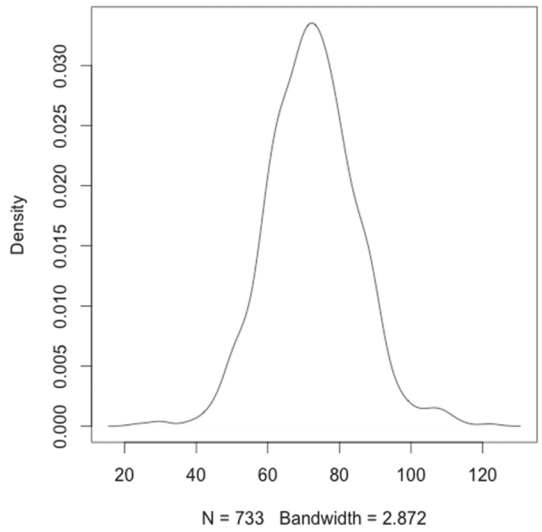
- A histogram can look very different depending on certain choices such as the *number of bins* and their *spacing*.

Many people prefer the density plot over the histogram since the histogram output is so sensitive to its options.

- A density plot is essentially a smoothed version of a histogram.
- A "kernel" smoother is used to construct a weighted average of data points and create a smoothed version of a histogram.

```
plot(density(pima$diastolic,na.rm=T),main="")
```

The density plot isn't as chunky (but you might see things at the boundaries since there are few data points).

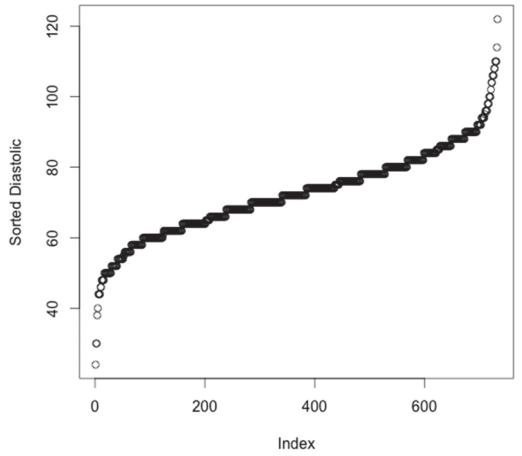


We could simply plot the sorted data against its index.

```
plot(sort(pima$diastolic),ylab="Sorted
Diastolic")
```

In this plot, we get to see

- the individual values
- the data distribution
- possible outliers
- the "discreteness" of the data.

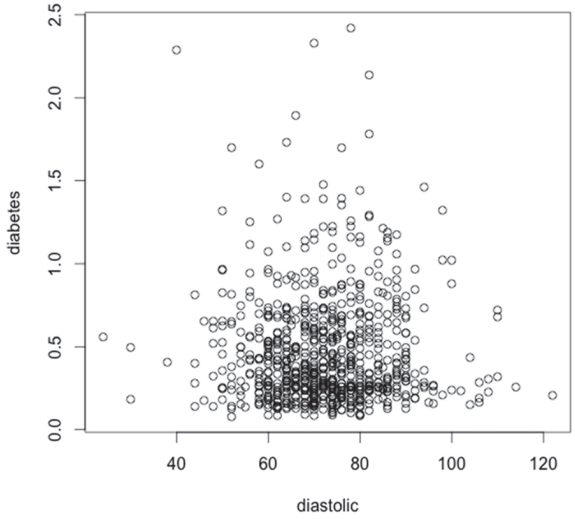


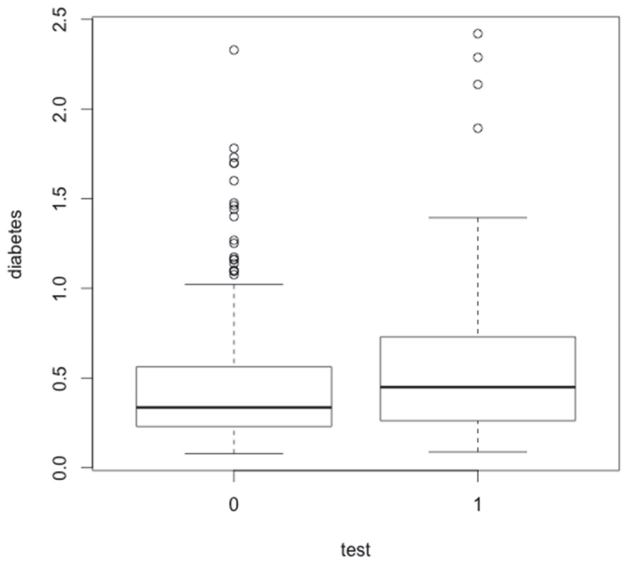
We can create useful bivariate plots using the plot function.

The first plot is a standard scatterplot of diabetes vs diastolic blood pressure.

The second is a parallel boxplot of diabetes vs test result.

```
plot(diabetes ~ diastolic, data = pima)
# this is the 4th way to specify the pima
data
plot(diabetes ~ test, data = pima)
```



The plots we have just created are using the built-in base graphics systems in R.

- These are very fast, simple, and yet professionally looking to obtain.

A fancier alternative is to construct plots using the `ggplot2` package, also allowed by SOA on the PA exam.

- The plots look perhaps more elegant with a bit more effort.
- First, we need to load the package that we already installed.
  - What is the first line below doing?

```
if(!require("ggplot2")){install.packages("ggplot2"
)}
library(ggplot2)
```


For `ggplot2`:

- You first need to create a ggplot object using the `ggplot` function that specifies where the data comes from (here it comes from the data frame `pima`) and an **aesthetic** using `aes`.
- The aesthetic specifies what you see such as position in the *x* or *y* direction or aspects such as shape or color.
- The second part of the command (after the +) in each case is specifying the particular geometry for the plot (how you want to map the aesthetics).

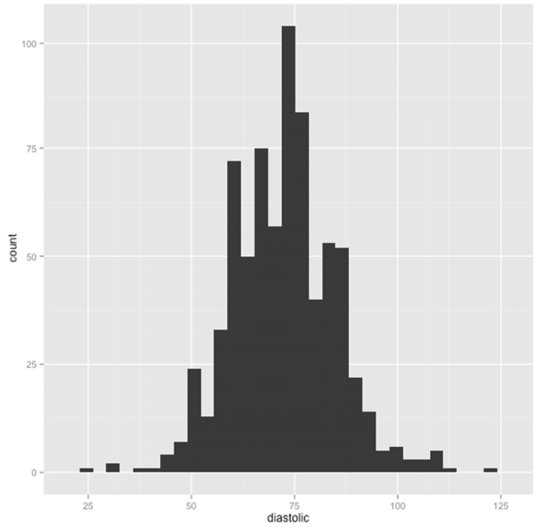
The same plots as before can now be constructed using `ggplot2`.

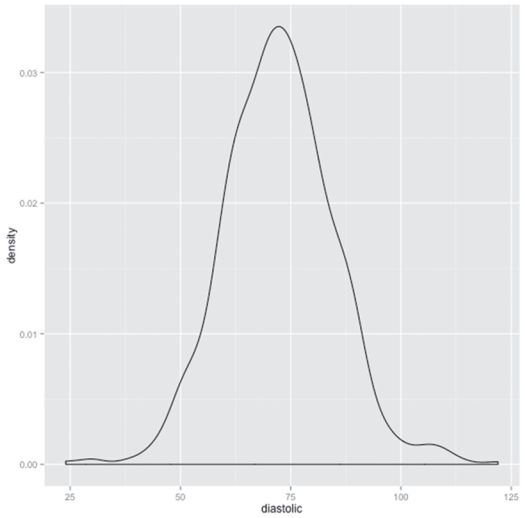
```
ggpimax  = ggplot(pima, aes(x=diastolic))
ggpimaxy = ggplot(pima, aes(x=diastolic, y=diabetes))
ggpimax + geom_histogram()
ggpimax + geom_density()
ggpimaxy + geom_point()
ggplot(pima,aes(x=diastolic,y=diabetes,shape=test)) +
  geom_point() +
  theme(legend.position = "top", legend.direction =
"horizontal")
ggpimaxy + geom_point(size=1) + facet_grid(~ test)
```

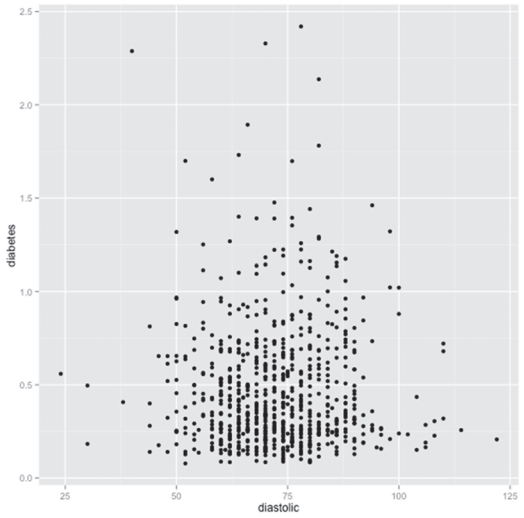
- A **theme** specifies options for the appearance of the plot.
  - We specified where the legend should appear in one plot and to use more than one panel (facets) in another.

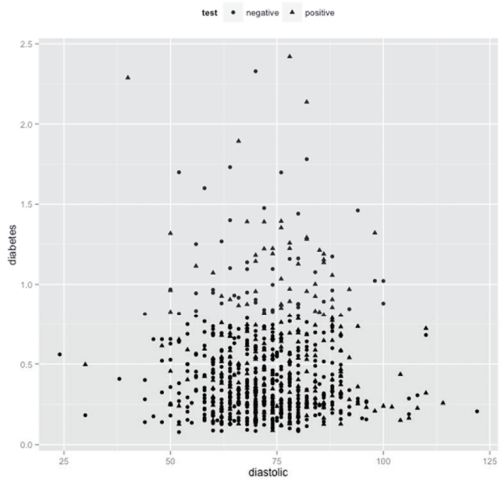
- As true for any good package, there is a very good documentation available:

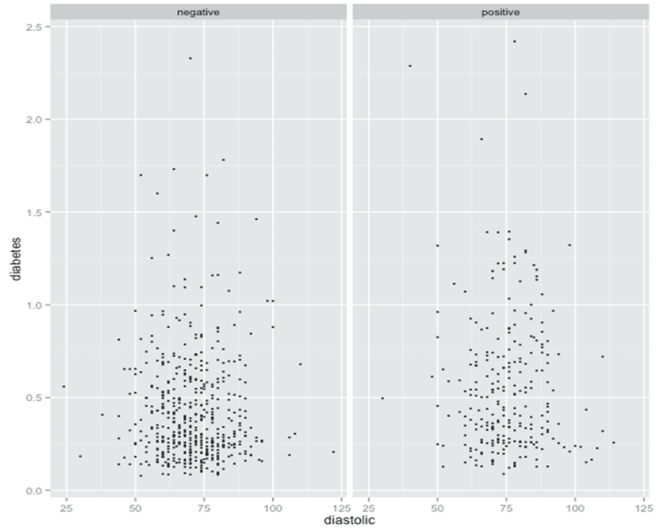
```
require(ggplot2)
help("ggplot2")#starts from the beginning
help(package="ggplot2")# alphabetical list
```









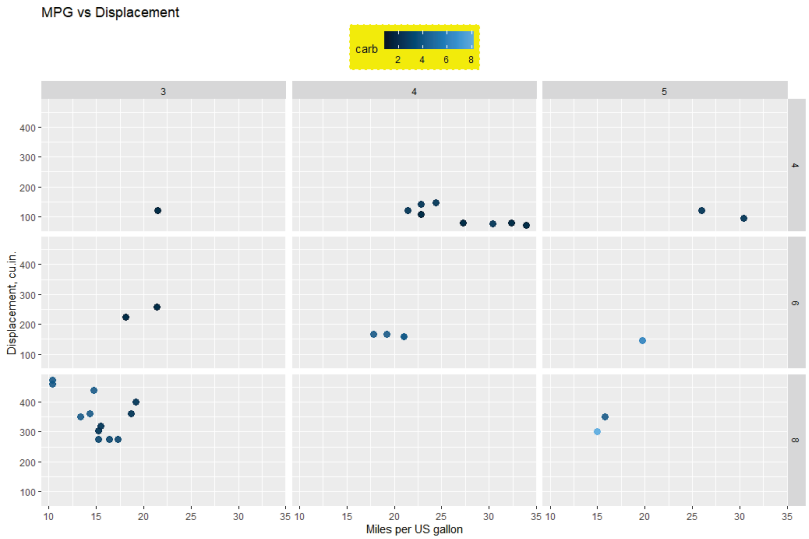
- The advantage of `ggplot2` is more apparent when producing more complex plots involving more than just two variables.
- Besides the built-in datasets, i.e., `data()` R has `library(datasets)` with even more data to play with.
- One of the datasets in `library(datasets)` is `mtcars`. Let's read the description:

```
library(datasets)
?mtcars
```


- Let us produce a fancy graphical display:

```
library(ggplot2)

ggplot(mtcars, aes(mpg, disp)) +
geom_point(aes(color = carb), size =3, alpha =0.9) +
facet_grid(cyl ~ gear) + xlab('Miles per US gallon')
+ ylab('Displacement, cu.in.') +  ggtitle('MPG vs
Displacement') +
theme(legend.background = element_rect(color =
'yellow', fill = 'yellow', size = 1, linetype =
'dotted'), legend.key = element_rect(fill = 'red'),
legend.position = 'top')
```



- Again, most packages are big "programs" themselves.
  - Practically any tool that you might need is available… but you need to "learn" this new tool to be able to use it.
    - Don't be afraid to start with an example from a textbook or a manual and to modify the code.
- More practice with `ggplot2` will be provided later in the class.
- You could also study Introduction to Data Visualization with `ggplot2`, a full DataCamp course:

https://www.DataCamp.com/courses/data-visualization-with-ggplot2-1

Back to our beautiful MPG vs Displacement plot, awesome, but how to save it? Recall:

  - If you are in RStudio, we used *Export* and selected *Save as Image…* or you could select *Save as PDF too.*
  - Select "view plot" after saving it in order to preview what was saved.

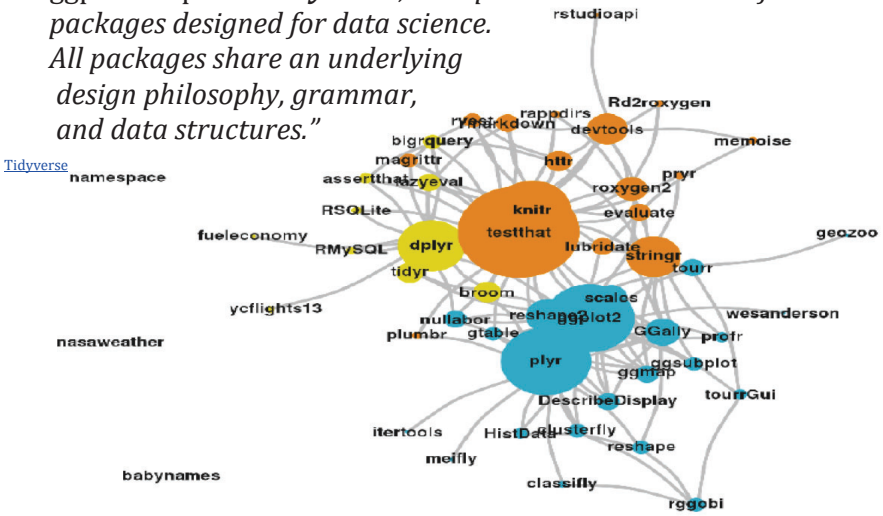
- For a quick draft, you could simply copy the plot to the clipboard and paste as needed.
- For much more control, you can save an image with specific functions from the console too.

- It is also easy to publish any image or any report on the web with **RPubs**, a free service provided by Posit (formerly RStudio) company.

- Related:
  - The company making **RStudio** changed its name to **Posit** last year.
    - They support Python and Jupyter Interactive Computing Notebooks now (dominating the Python landscape).
  - **RStudio IDE** that we use should continue to exist.
  - It seems that all functionality will remain the same and the software will remain free with Open Source License.

- Using **RPubs** is a pretty fast way to provide a weblink with your results.
  - RPubs uses R Markdown (different type of notebooks) that you will need for sine classes…
    - Packages will be automatically installed and/or updated only the first time you use **RPubs**.
- Select **Publish** and next select **RPubs**.
  - Create a *free account* the first time you are using the service.
  - Careful: my understanding is that anything published in RPubs is automatically in the *public domain*. Don't publish any confidential company data…

- ggplot2 is part of ***tidyverse***, *"an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures."*



13. Environmental Data (and File System)

- Recall, you can type and edit your R commands in the RStudio editor, typically the upper-left window.
  - To open a new R script file press `Ctrl+Shift+N` or select File->New File->New R Script from the menu
  - I myself use a different text editor, but this is mostly a matter of habit. The RStudio editor is very nice and convenient to use as you can immediately execute.
- Customary, R-scripts containing commands and comments have a file extension "`.R`".

- Simply hit `<Ctrl+Enter>` or click `<run>` at the taskbar of the text editor window to execute in the console the row the cursor is at in the editor.
  - Select a few rows in the editor window to execute the selected or all the commands in the console.
  - Or select all with `Ctrl+A` in the editor window and hit `Ctrl+Enter`
  - Save your changes and open the `.R` file again when needed.
    - You have *save as* functionality too.