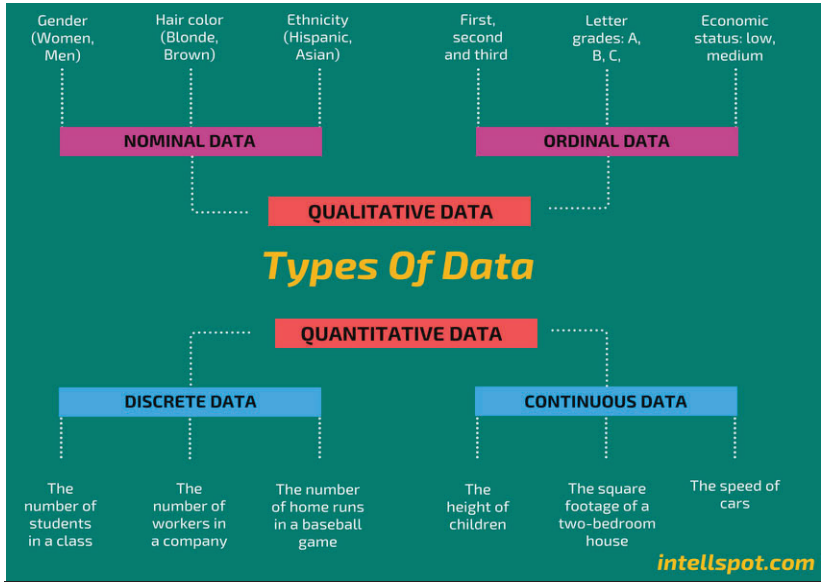
R will automatically perform an operation to all entries of a matrix, a vector, a data frame variable, or even a data frame:

```
(1:4)^2 # will return 1,4,9,16
log(B)  # will take natural log on each
entry of the matrix B
```

Careful, sometimes this is a blessing as it gives you speed, sometimes, not so much as you might have aimed for something else.

Check your outputs! That is the beauty of interpreted programming languages.



Qualitative data is also called **Categorical** data and should be stored as a *factor* in R. The factor function takes vectors of any data type and converts them to factors with *levels*.

- Examples:

```
f1 <- factor(rep(1:6, times = 3))
f1
f2 <- factor(c("a", 7, "blue", "blue"))
f2
is.factor(f2)
```

- Most R functions will automatically handle categorical variables when properly coded as a factor.
  That is often very useful when you fit a statistical model as it will automate handling of “labels only” variables.

- When plotting or modeling in R, if a character variable is included, R will treat it as a factor. By default, R will:
  - Sort the levels of the factor alphabetically by their label.
  - Leave the existing labels as they are.
  - Include all unique values.
    - That might be problematic in some settings.

Let’s consider a scenario where we have survey response data. In this survey, individuals could indicate their level of agreement with a statement by choosing one of the following options: "Agree", "Neither agree nor disagree", "Disagree", or "Not applicable". Additionally, individuals could omit answering so there is a "No answer" level too.

- Let’s simulate such data and use bar plot to display it.

5. Accessing Elements of a Data Structure

Subsets of the elements of a vector may be selected by appending to the name of the vector an index vector in square brackets.

```
a <- seq(2, 16, by = 2); a
```

Access the 2nd, 4th, and 6th elements of a.

```
a[c(2, 4, 6)]
```

Access all elements in a EXCEPT the 2nd, 4th, and 6th.

```
a[-c(2, 4, 6)]
```

#Access all elements in a except elements 3 through 6.

```
a[-(3:6)]
```

The following will produce an error (indices start from 1 in R)

```
a[-3:6]
```



```
B <- matrix(data=1:12, nrow=3, ncol=4,
byrow=T); B
```

To access the last entry in the last row of the matrix:

```
B[3,4]
```

To grab the second row only.

```
B[2,]
```

What is the median of the fourth column?

```
median(B[,4])
```

- What will happen if we don't provide enough values to the matrix command?
  - Using arrow up will give you quick way to edit old commands.

Sometimes we need to know if the elements of an object satisfy certain conditions. This can be determined using the logical operators <, <=, >, >=, ==, !=

"==" means "equal to" and "!=" means not equal to. This comparison Boolean operators return True or False.

- Checks which values of a are greater than 10

```
a > 10
```

- values of a less than or equal to 4

```
a <= 4
```

- values of v a equal to 10

```
a == 10
```

- values of a not equal to 10

```
a != 10
```


- More complicated logical arguments can be made using & and | characters.
  - & means "and"
  - | means "or"
  - ! means "not"
- Elements of a greater than 6 and less than or equal to 10

```
(a > 6)&(a <= 10)
```

- Elements of a less than or equal to 4 or greater than or equal to 12.

```
(a <= 4)|(a >= 12)
```

- Logical statements can be used to return parts of an object satisfying the appropriate criteria.
- Return elements of `a` less than 6.
  ```
  a[a < 6]
  ```
- Return elements of `a` equal to 10.
  ```
  a[a == 10]
  ```
- Return elements of `a` less than 6 or equal to 10.
  ```
  a[(a < 6)|(a == 10)]
  ```

- For practice, let's return the elements that are not divisible by 3.
  - Using indices will work but we want a solution that will work if we have say 10^9 data points.

6. Data Frames

- Date frames are created by passing vectors into the `data.frame` function or directly loading some types of data.
  - The names of the columns in the data frame are the names of the vectors you proved to the `data.frame` function.
- Example:
  ```
  d <- c(1, 2, 3, 4)
  e <- c("red", "white", "blue", NA)
  f <- c(TRUE, TRUE, TRUE, FALSE)
  mydataframe <- data.frame(d,e,f)
  mydataframe
  ```


- The columns of a data frame can be renamed using the `names` function on the data frame.
  ```
  names(mydataframe) <- c("ID", "Color",
  "Passed"); mydataframe
  ```

- The columns of a data frame can also be named when you are first creating the data frame by using "name =" for each vector of data.
  ```
  mydataframe2 <- data.frame(ID=d,
  Color=e, Passed=f); mydataframe2
  ```


- The vectors in a data frame may be accessed using "$" and specifying the name of the desired vector.
- Access the Color vector in `df`:
  ```
  mydataframe$Color
  ```
- The data and vectors of a data frame may also be accessed by specifying the desired row(s) or column(s) in square brackets.
- Access first row of `df`
  ```
  mydataframe[1,]
  ```
- Access third column of `mydataframe`
  ```
  mydataframe[,3]
  ```
- Access the ID column of `mydataframe2` and assign it (save it) to `newID`
  ```
  newID <- mydataframe2$ID
  ```

- Let's consider a dataset already preloaded in R.
- This data set was compiled in 1936 by Sir Ronald Fisher and has become a classic example in data mining/machine learning.

- The dataset is called "iris" in R:

```
iris
```

- R is case sensitive.

```
Iris # will give you an error
```


- To see just the first few rows, type:

```
head(iris)
```

- The left column displays the observation number in the dataset
- The next five columns are the variables in the dataset. Observe that the last column contains the names of the species of iris in this study.
  - Is this quantitative or qualitative variable?

Let's play with the `iris` dataset a bit more…

If you will be working with a single dataset it might be convenient to attach the data frame first.
Then you don't need to $ …

```
attach(iris)
mean(Sepal.Width) # average
…
detach(iris) # don't forget
```

- Is the iris object a data frame?
- How big is it?
- What are the names of the variables present?
- What are the variable values recorded for the 37th iris in the data?

- What are the species of `iris` present in the dataset? How many each?
- Extract the first 10 observations and store them in *DANvar*
- What are the species of `iris` present in the *DANvar*? How many each?
- What is the average petal width in *DANvar*?
- What is the average petal width overall?
- What are the average petal widths by species of iris?
  - Note that there are many flexible ways to do this and also data manipulation libraries that automate this.
    - "Get the job done and move on!" should be your attitude for school/research/prototyping.

7. Helpful Functions

General Functions

```
str(x)    # compact overview of object x
length(x)# length of x
sum(x)    # sum elements in x
mean(x)   # mean of elements in x
var(x)    # sample variance of elements in x
sd(x)  # standard deviation of elements in x
range(x) # range of elements in x
log(x)    # ln of elements in x
min(x) # minimum value in a vector
max(x) # maximum value in a vector
```


General Functions (cont.)

```
summary(x)# 5-number summary of x
unique(x) # each unique value in a vector
duplicated(x)# for each element: is it a
duplicate with regard to previous ones.
rev(x)    # reverse the order.
sort(x)   # sort the elements in a vector.
append(x)# append or insert in a vector.
```


You cannot "learn" all the functions in R. For example, these are just the built-into R functions to round numerical values:

Rounding Functions

```
round(x) # IEEE standard to round up or down
to specified decimal places
signif(x)# rounds to the specified digits=
number of digits
trunc(x) # rounds by removing non-integer
part of number
floor(x), ceiling(x) # rounds to integer
below or above, respectively
zapsmall(x) # remove numerical zeros, i.e.,
makes numbers close to zero (compared to
others in x) zero
```


Functions related to statistical distributions

Suppose that a random variable X has the "dist" distribution

`p[dist](q, ...)` – returns the cdf of X evaluated at q, i.e., $p = \Pr(X \leq q)$.

`q[dist](p, ...)` – returns the inverse cdf (or quantile function) of X evaluated at `p`, i.e., $q = \inf\{x: \Pr(X \leq x) \geq p\}$.

`d[dist](x, ...)` – returns the mass or density of X evaluated at `x` (depending on whether it's discrete or continuous).

`r[dist](n, ...)` – returns an i.i.d. random sample of size `n` having the same distribution as X.

... indicates that additional arguments describing the shape of the distribution (but default values are assumed).

In case you are currently enrolled in a statistics course, you don't need a z-table :)

`pnorm(1.96, mean = 0, sd = 1)`
returns the probability that a normal random variable with mean 0 and standard deviation 1 is less than or equal to 1.96. That is, for Z~N(0,1), the function returns P(Z≤ 1.96).

- Note that mean=0, sd=1 are the default values so you can skip them, `pnorm(1.96)` will give the same result.

- If you want upper tail, P(Z> 1.96), we can get it with
`pnorm(1.96, lower.tail=FALSE)`
or
`1-pnorm(1.96)`

More distributions

`qunif(0.6, min = 0, max = 1)`

returns the value x such that $P(X \leq x) = 0.6$ for a **uniform** random variable on the interval [0, 1].

- Here `min = 0, max = 1` are default bounds for the function so same result if you type `qunif(0.6)`.

`dbinom(2, size = 20, prob = .2)`
returns the probability that Pr $(X = 2)$ for $X \sim \text{Binomial}(n = 20, p = 0.2)$ random variable

- `dexp(1, rate = 4)`
returns the density of an exponential random variable with mean = ¼ evaluated at 1
- `rchisq(100, df = 5)`
returns a random sample of 100 observations from a chi-squared random variable with 5 df.

- For list of all built-in R distributions check:
https://cran.r-project.org/web/views/Distributions.html
- If not built-in already, any distribution that you might need is likely available as an external library.

More Functions

Perhaps thousands of functions are built in R. Just "several" of those are listed in the rather thorough "Short Reference Card": https://cran.r-project.org/doc/contrib/Baggott-refcard-v2.pdf

Tens of thousands are available in the external libraries (the R lingo for this is 'package').

You cannot "learn" them all!
Learn (what you need to learn) as you program.

Start working on a problem and seek help for the functions that you need.

Check your outputs at every step, that is the beauty of using an interpreted programming language as R.