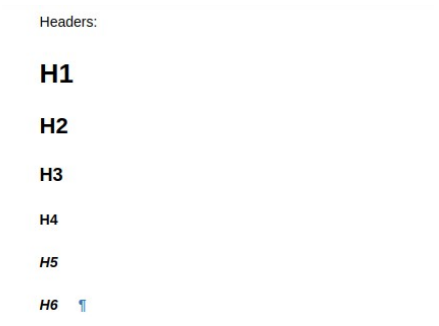
Útil

https://www.amazon.com/How-Lie-Statistics-Darrell-Huff/dp/0393310728
https://www.nytimes.com/column/whats-going-on-in-this-graph
https://www.nytimes.com/interactive/2018/05/03/learning/08WGOITGraphLN.html

Bivariate Distributions
https://markkurzejaumich.shinyapps.io/bivariate_analysis/

Jupyter notebooks

Markdown language



```
Headers:

# H1
## H2
### H3
#### H4
##### H5
###### H6
```

Text modifications:

Emphasis, aka italics, with *asterisks* or *underscores*.

Strong emphasis, aka bold, with **asterisks** or **underscores**.

Combined emphasis with **asterisks and *underscores***.

Strikethrough uses two tildes. ~~Scratch this.~~

```
Text modifications:

Emphasis, aka italics, with *asterisks* or _underscores_.

Strong emphasis, aka bold, with **asterisks** or __underscores__.

Combined emphasis with **asterisks and _underscores_**.

Strikethrough uses two tildes. ~~Scratch this.~~
```

Lists:

1. First ordered list item
2. Another item
    - Unordered sub-list.
3. Actual numbers don't matter, just that it's a number
    A. Ordered sub-list
4. And another item.
5. Unordered list can use asterisks
6. Or minuses
7. Or pluses

```
Lists:

1. First ordered list item
2. Another item
  * Unordered sub-list.
1. Actual numbers don't matter, just that it's a number
  1. Ordered sub-list
4. And another item.

* Unordered list can use asterisks
- Or minuses
+ Or pluses
```

Or:

- Headers
- Text modifications such as italics and bold
- Ordered and Unordered lists
- Links
- Tables
- Images
- Etc.

```
* Headers
* Text modifications such as italics and bold
* Ordered and Unordered lists
* Links
* Tables
* Images
* Etc.
```

Python useful libraries

For this course, we will focus primarily on the following libraries:

- **Numpy** is a library for working with arrays of data.
- **Pandas** provides high-performance, easy-to-use data structures and data analysis tools.
- **Scipy** is a library of techniques for numerical and scientific computing.
- **Matplotlib** is a library for making graphs.
- **Seaborn** is a higher-level interface to Matplotlib that can be used to simplify many graphing tasks.
- **Statsmodels** is a library that implements many statistical techniques.

Numpy

Permite trabalhar com matrizes (arrays).

Arrays

import numpy as np

```
### Create a 3x2 numpy array
c = np.array([[1,2],[3,4],[5,6]])
#3 linhas x 2 colunas
#[1,2]
#[3,4]
#[5,6]

### Print shape
print(c.shape)

### Print some values in c
print(c[0,1], c[1,0], c[2,0], c[2,1])
```

```
(3, 2)
2 3 5 6
```

```
### Create a 2x2 numpy array
b = np.array([[1,2],[3,4]])
#[1,2]
#[3,4]

### Print shape
print(b.shape)

## Print some values in b
print(b[0,0], b[0,1], b[1,1])
```

```
(2, 2)
1 2 4
```

Algumas funções úteis para criar matrizes preenchidas com valores pré-definidos:

```
In [4]: ### 2x3 zero array
        d = np.zeros((2,3))
        d

Out[4]: array([[0., 0., 0.],
               [0., 0., 0.]])
```

```
In [6]: ### 4x2 array of ones
        e = np.ones((4,2))

        print(e)

        ### 2x2 constant array
        f = np.full((2,2), 9)

        print(f)

        ### 3x3 random array
        g = np.random.random((3,3))

        print(g)
```

```
[[1. 1.]
 [1. 1.]
 [1. 1.]
 [1. 1.]]
[[9 9]
 [9 9]]
[[0.08575662 0.13558526 0.0585958 ]
 [0.93469334 0.33128823 0.00713129]
 [0.33550695 0.53203299 0.51145875]]
```

Slice de array:

```
In [9]: ### Create 3x4 array
        h = np.array([[1,2,3,4,], [5,6,7,8], [9,10,11,12]])

        print(h)

        ### Slice array to make a 2x4 sub-array
        i = h[:2]

        print(i)

        ### Slice array to make a 2x2 sub-array
        x = h[:2, 1:3]
        print (x)
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
[[1 2 3 4]
 [5 6 7 8]]
[[2 3]
 [6 7]]
```

Modificando o valor de um elemento:

```
print(h[0,1])

### Modify the slice
i[0,1] = 1738

### Print to show how modifying the slice also changes the base object
print(h[0,1])
```

```
2
1738
```

Consultar o tipo de dado do array:

```
In [6]: ### Integer
        j = np.array([1, 2])
        print(j.dtype)


        ### Float
        k = np.array([1.0, 2.0])
        print(k.dtype)

        ### Force Data Type
        l = np.array([1.0, 2.0], dtype=np.int64)
        print(l.dtype)
```

```
int64
float64
int64
```

Operações matemáticas

Soma e diferença

```
x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)

# Elementwise sum; both produce the array
# [[ 6.0  8.0]
#  [10.0 12.0]]
print(x + y)
print(np.add(x, y))

# Elementwise difference; both produce the array
# [[-4.0 -4.0]
#  [-4.0 -4.0]]
print(x - y)
print(np.subtract(x, y))
```

Soma total e por coluna:

```
In [8]: x = np.array([[1,2],[3,4]])

        ### Compute sum of all elements; prints "10"
        print(np.sum(x))

        ### Compute sum of each column; prints "[4 6]"
        print(np.sum(x, axis=0))

        ### Compute sum of each row; prints "[3 7]"
        print(np.sum(x, axis=1))
```

```
10
[4 6]
[3 7]
```

Produto e divisão

```
x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)
```

```
# Elementwise product; both produce the array
# [[ 5.0 12.0]
#  [21.0 32.0]]
print(x * y)
print(np.multiply(x, y))

# Elementwise division; both produce the array
# [[ 0.2         0.33333333]
#  [ 0.42857143  0.5       ]]
print(x / y)
print(np.divide(x, y))
```

Raiz quadrada

```
x = np.array([[1,2],[3,4]], dtype=np.float64)
```

```
# Elementwise square root; produces the array
# [[ 1.          1.41421356]
#  [ 1.73205081  2.        ]]
print(np.sqrt(x))
```

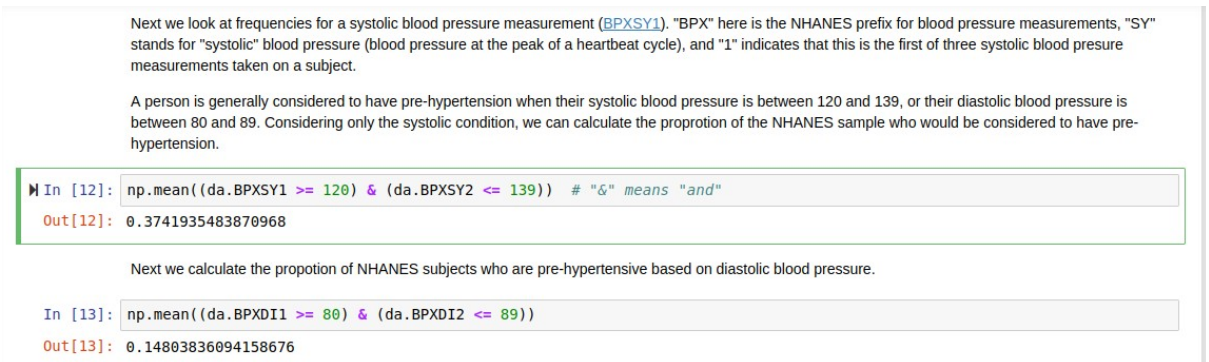
Média

In [9]:
```
x = np.array([[1,2],[3,4]])

### Compute mean of all elements; prints "2.5"
print(np.mean(x))

### Compute mean of each column; prints "[2 3]"
print(np.mean(x, axis=0))

### Compute mean of each row; prints "[1.5 3.5]"
print(np.mean(x, axis=1))
```

```
2.5
[2. 3.]
[1.5 3.5]
```

Mean to get Proportion

The np.mean() function is generally used to get the average of all values in a dataset. We can apply this function with a logical statement to get the percent of values that satisfy the logical statement.
https://discuss.codecademy.com/t/how-does-np-mean-give-us-the-probability/362762



Next we look at frequencies for a systolic blood pressure measurement (BPXSY1). "BPX" here is the NHANES prefix for blood pressure measurements, "SY" stands for "systolic" blood pressure (blood pressure at the peak of a heartbeat cycle), and "1" indicates that this is the first of three systolic blood presure measurements taken on a subject.

A person is generally considered to have pre-hypertension when their systolic blood pressure is between 120 and 139, or their diastolic blood pressure is between 80 and 89. Considering only the systolic condition, we can calculate the proprotion of the NHANES sample who would be considered to have pre-hypertension.

In [12]:
```
np.mean((da.BPXSY1 >= 120) & (da.BPXSY2 <= 139))  # "&" means "and"
```

Out[12]:
```
0.3741935483870968
```

Next we calculate the propotion of NHANES subjects who are pre-hypertensive based on diastolic blood pressure.

In [13]:
```
np.mean((da.BPXDI1 >= 80) & (da.BPXDI2 <= 89))
```

Out[13]:
```
0.14803836094158676
```

Finally we calculate the proportion of NHANES subjects who are pre-hypertensive based on either systolic or diastolic blood pressure. Since some people are pre-hypertensive under both criteria, the proportion below is less than the sum of the two proportions calculated above.

Since the combined systolic and diastolic condition for pre-hypertension is somewhat complex, below we construct temporary variables 'a' and 'b' that hold the systolic and diastolic pre-hypertensive status separately, then combine them with a "logical or" to obtain the final status for each subject.

In [14]:
```
a = (da.BPXSY1 >= 120) & (da.BPXSY2 <= 139)
b = (da.BPXDI1 >= 80) & (da.BPXDI2 <= 89)
print(np.mean(a | b))  # "|" means "or"
```

```
0.43975588491717527
```

Exponencial (e^x)

Python number method **exp()** returns returns exponential of x: e^x.

Syntax

Following is the syntax for **exp()** method −

```
import math

math.exp( x )
```

```
Y  = exp(-np.arange(-2,2,dx)**2)
print(Y)
```

```
[0.01831564 0.01906121 0.01983316 0.02063225 0.02145924 0.02231491
 0.02320007 0.02411551 0.02506206 0.02604056 0.02705185 0.02809679
 0.02917626 0.03029114 0.03144234 0.03263076 0.03385732 0.03512297
 0.03642864 0.03777529 0.0391639  0.04059542 0.04207086 0.0435912
 0.04515745 0.04677062 0.04843173 0.05014181 0.05190189 0.05371301
 0.05557621 0.05749254 0.05946306 0.06148881 0.06357087 0.06571027
 0.0679081  0.0701654  0.07248323 0.07486266 0.07730474 0.07981052
 0.08238104 0.08501734 0.08772047 0.09049144 0.09333128 0.09624098
 0.09922155 0.10227398 0.10539922 0.10859825 0.11187199 0.11522138
```

...

Arange

Gera um array.
Sintaxe: arange([start,] stop, [step,] dtype=None)

```
>>> arange(5)
array([0, 1, 2, 3, 4])
```

Se quisermos criar a mesma seqüência mas com números em pontos flutuantes, temos duas opções: usar o argumento `dtype` ou então especificar o parâmetro na forma de um número inteiro:

```
>>> arange(5.)
array([ 0.,  1.,  2.,  3.,  4.])
>>> arange(5, dtype=float)
array([ 0.,  1.,  2.,  3.,  4.])
```

Para criarmos uma seqüência com início e intervalo diferentes, basta especificarmos os parâmetros correspondentes. Por exemplo, para criar uma seqüência que vai de -5 a 5 (excluindo este último) com um intervalo de 0.5 entre cada elemento, fazemos:

```
>>> arange(-5, 5, 0.5)
array([-5. , -4.5, -4. , -3.5, -3. , -2.5, -2. , -1.5, -1. , -0.5,  0. ,
        0.5,  1. ,  1.5,  2. ,  2.5,  3. ,  3.5,  4. ,  4.5])
```

```
In [5]: X  = np.arange(-2,2,dx)
        print(X)
```

```
[-2.00000000e+00 -1.99000000e+00 -1.98000000e+00 -1.97000000e+00
 -1.96000000e+00 -1.95000000e+00 -1.94000000e+00 -1.93000000e+00
 -1.92000000e+00 -1.91000000e+00 -1.90000000e+00 -1.89000000e+00
 -1.88000000e+00 -1.87000000e+00 -1.86000000e+00 -1.85000000e+00
 -1.84000000e+00 -1.83000000e+00 -1.82000000e+00 -1.81000000e+00
 -1.80000000e+00 -1.79000000e+00 -1.78000000e+00 -1.77000000e+00
 -1.76000000e+00 -1.75000000e+00 -1.74000000e+00 -1.73000000e+00
 -1.72000000e+00 -1.71000000e+00 -1.70000000e+00 -1.69000000e+00
 -1.68000000e+00 -1.67000000e+00 -1.66000000e+00 -1.65000000e+00
 -1.64000000e+00 -1.63000000e+00 -1.62000000e+00 -1.61000000e+00
 -1.60000000e+00 -1.59000000e+00 -1.58000000e+00 -1.57000000e+00
 -1.56000000e+00 -1.55000000e+00 -1.54000000e+00 -1.53000000e+00
 -1.52000000e+00 -1.51000000e+00 -1.50000000e+00 -1.49000000e+00
 -1.48000000e+00 -1.47000000e+00 -1.46000000e+00 -1.45000000e+00
 -1.44000000e+00 -1.43000000e+00 -1.42000000e+00 -1.41000000e+00
 -1.40000000e+00 -1.39000000e+00 -1.38000000e+00 -1.37000000e+00
 -1.36000000e+00 -1.35000000e+00 -1.34000000e+00 -1.33000000e+00
 -1.32000000e+00 -1.31000000e+00 -1.30000000e+00 -1.29000000e+00
 -1.28000000e+00 -1.27000000e+00 -1.26000000e+00 -1.25000000e+00
 -1.24000000e+00 -1.23000000e+00 -1.22000000e+00 -1.21000000e+00
```

....

Random

Passando uma seed é possível gerar um array idêntico em outra execução

```
np.random.seed(282629734)
```

Normal

```
import numpy as np
np.random.seed(123)
random_normal_variables = np.random.normal(100,1,3)
print(random_normal_variables)
```

Choice

```
import numpy as np
np.random.seed(123)
population = np.arange(1,100)
sample = np.random.choice(population, 10)
sample
```

Scipy

The SciPy.Stats module contains a large number of probability distributions as well as a growing library of statistical functions such as:

* Continuous and Discrete Distributions (i.e Normal, Uniform, Binomial, etc.)

* Descriptive Statistcs

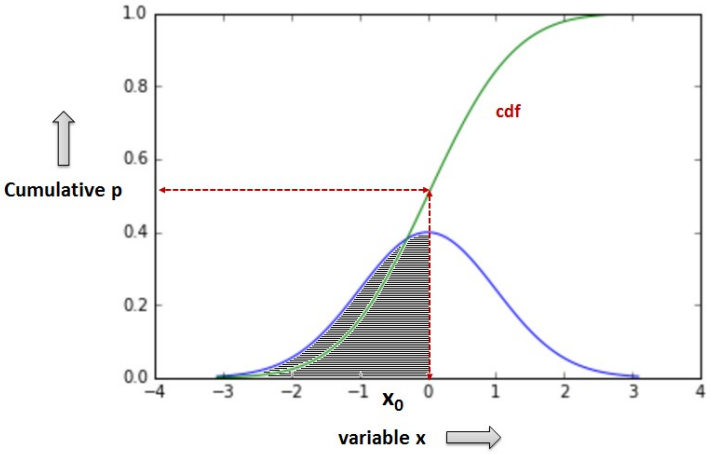
* Statistical Tests (i.e T-Test)

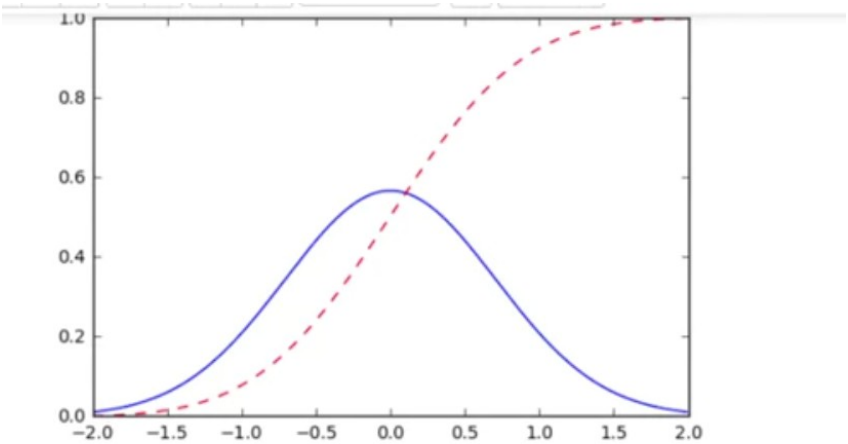
Gerando uma distribuição normal aleatória de n=10:

from scipy import stats
import numpy as np

```
In [14]: ### Print Normal Random Variables
         print(stats.norm.rvs(size = 10))

[ 2.03472027 -0.41029035 -0.42205721 -0.29678088 -2.00753762 -0.84212236
 -0.61006664 -0.34858785 -0.48534383 -1.34572375]
```

CDF: cumulative density function






```
]: print(stats.norm.cdf(np.array([-1.5,-.5,0,1,1.5,2, 6])))

[ 0.0668072   0.30853754  0.5         0.84134475  0.9331928   0.97724987
  1.        ]
```

T-Student

https://docs.scipy.org/doc/scipy/reference/tutorial/stats.html

Gerando uma distribuição T-Student:

```
>>> x = stats.t.rvs(10, size=1000)
```

Here, we set the required shape parameter of the t distribution, which in statistics corresponds to the degrees of freedom, to 10. Using size=1000 means that our sample consists of 1000 independently drawn (pseudo) random numbers. Since we did not specify the keyword arguments *loc* and *scale*, those are set to their default values zero and one.

In [15]:
```
np.random.seed(282629734)

# Generate 1000 Student's T continuous random variables.
x = stats.t.rvs(10, size=1000)
print(x)
```

```
[-2.08627648e-01 -1.29471404e+00 -1.63750825e+00 -1.40201994e+00
 -9.78474650e-01  1.50921872e-01 -6.03228797e-01 -9.84282743e-03
  1.74312761e+00 -1.13979432e+00 -4.66677733e-01  4.86407881e-01
 -2.36650679e+00  6.77389379e-01  5.16035617e-02  8.72665270e-01
 -1.39692591e+00 -8.63605331e-02 -1.40892128e+00 -4.06199875e-01
 -6.42117306e-01  6.00697501e-01  2.09925227e-01  4.00177781e-01
 -1.23045890e+00  7.02140030e-01  1.50683645e-01  1.42444397e+00
 -3.73827782e-01 -1.36380861e+00 -5.51851062e-01 -1.73761084e+00
 -1.55252456e-01 -6.37074241e-02 -8.36353978e-01 -5.84233524e-01
 -5.44322070e-01 -1.72065894e+00 -1.81223936e-01  2.66857789e-01
 -1.59705579e+00 -1.02180890e-01 -2.41532906e-01 -6.71904774e-01
 -4.71684857e-01 -3.83783319e-01  1.10563139e+00  1.53254855e+00
 -1.15696045e+00  7.20068291e-01  8.11219763e-01  1.71006867e+00
 -2.18827710e+00 -1.51987441e+00 -2.10188850e+00 -4.81147788e-01
 -2.94612350e-02  7.04107081e-01  9.41338072e-01  5.40415266e-01
 -1.39426696e+00 -3.48237194e-01 -8.60729960e-01 -3.61157904e+00
 -2.43137060e+00  2.19992321e+00 -1.32335446e+00 -1.19870982e+00
  5.49169478e-02  1.00194448e+00 -7.24698675e-01  2.96714967e+00
 -2.27306395e+00  5.79143502e-01  4.84816852e-01  3.37292663e+00
```

Estatística descritiva

In [16]:
```
# Do some descriptive statistics
print(x.min())   # equivalent to np.min(x)

print(x.max())   # equivalent to np.max(x)

print(x.mean())  # equivalent to np.mean(x)

print(x.var())   # equivalent to np.var(x))

stats.describe(x)
```

```
-3.7897557242248197
5.263277329807165
0.014061066398468422
1.288993862079285
```

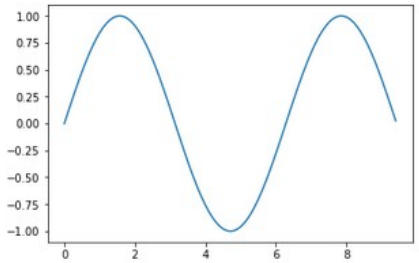
Out[16]:
```
DescribeResult(nobs=1000, minmax=(-3.7897557242248197, 5.263277329807165), mean=0.014061066398468422, variance=1.290
2841462255106, skewness=0.21652778283120955, kurtosis=1.055594041706331)
```

MatplotLib

Matplotlib is a plotting library.

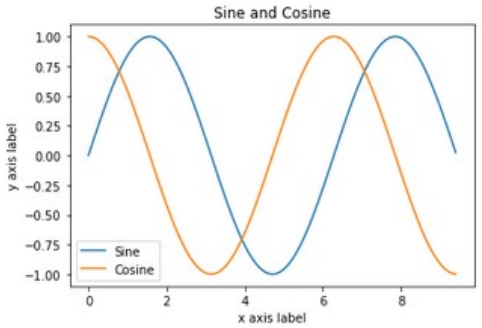
import numpy as np
import matplotlib.pyplot as plt

```
In [17]: # Compute the x and y coordinates for points on a sine curve
         x = np.arange(0, 3 * np.pi, 0.1)
         y = np.sin(x)

         # Plot the points using matplotlib
         plt.plot(x, y)
         plt.show()  # You must call plt.show() to make graphics appear.
```



Plotando mais de uma variável no gráfico:

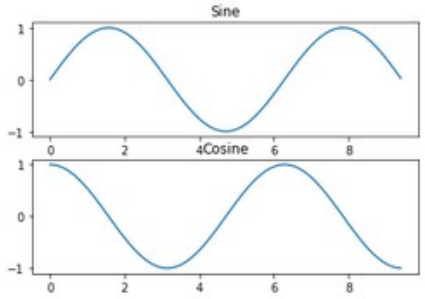
```
In [18]: # Compute the x and y coordinates for points on sine and cosine curves
         x = np.arange(0, 3 * np.pi, 0.1)
         y_sin = np.sin(x)
         y_cos = np.cos(x)

         # Plot the points using matplotlib
         plt.plot(x, y_sin)
         plt.plot(x, y_cos)
         plt.xlabel('x axis label')
         plt.ylabel('y axis label')
         plt.title('Sine and Cosine')
         plt.legend(['Sine', 'Cosine'])
         plt.show()
```



Subplots:

https://matplotlib.org/stable/gallery/subplots_axes_and_figures/subplots_demo.html

In [19]:
```
import numpy as np
import matplotlib.pyplot as plt

# Compute the x and y coordinates for points on sine and cosine curves
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Set up a subplot grid that has height 2 and width 1,
# and set the first such subplot as active.
plt.subplot(2, 1, 1)

# Make the first plot
plt.plot(x, y_sin)
plt.title('Sine')

# Set the second subplot as active, and make the second plot.
plt.subplot(2, 1, 2)
plt.plot(x, y_cos)
plt.title('Cosine')

# Show the figure.
plt.show()
```



Seaborn

https://seaborn.pydata.org/introduction.html

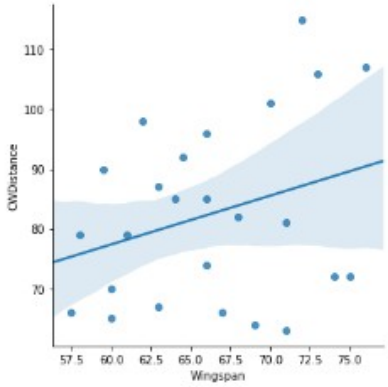
Examples dataset: https://github.com/mwaskom/seaborn-data

Seaborn is complementary to Matplotlib and it specifically targets statistical data visualization. But it goes even further than that: Seaborn extends Matplotlib and makes generating visualizations convenient.

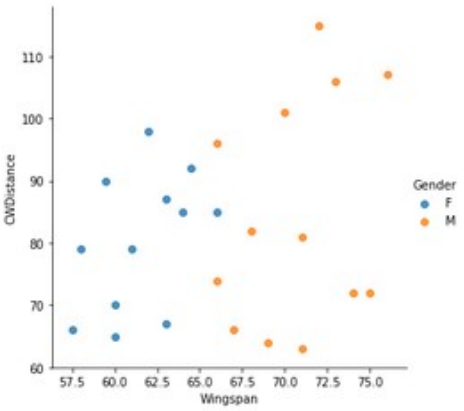
While Matplotlib is a robust solution for various problems, Seaborn utilizes more concise parameters for ease-of-use.

Scatter Plot

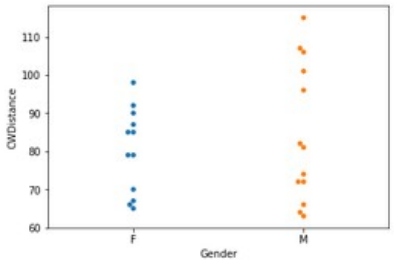
In [20]:
```
# Import necessary libraries
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Store the url string that hosts our .csv file
url = "Cartwheeldata.csv"

# Read the .csv file and store it as a pandas Data Frame
df = pd.read_csv(url)

# Create Scatterplot
sns.lmplot(x='Wingspan', y='CWDistance', data=df)

plt.show()
```



In [21]:
```
# Scatterplot arguments
sns.lmplot(x='Wingspan', y='CWDistance', data=df,
           fit_reg=False, # No regression line
           hue='Gender')   # Color by evolution stage

plt.show()
```



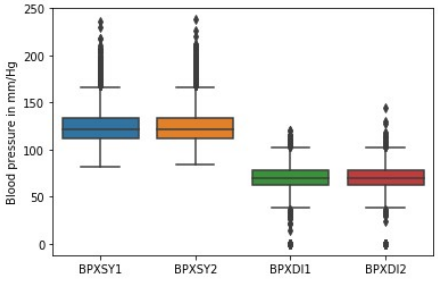
In [22]:
```
# Construct Cartwheel distance plot
sns.swarmplot(x="Gender", y="CWDistance", data=df)

plt.show()
```



Box plot

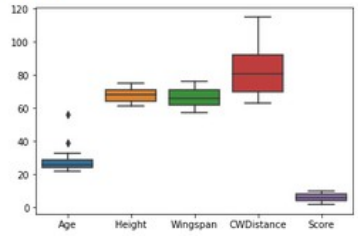
To compare several distributions, we can use side-by-side boxplots. Below we compare the distributions of the first and second systolic blood pressure measurements (BPXSY1, BPXSY2), and the first and second diastolic blood pressure measurements (BPXDI1, BPXDI2). As expected, diastolic measurements are substantially lower than systolic measurements. Above we saw that the second blood pressure reading on a subject tended on average to be slightly lower than the first measurement. This difference was less than 1 mm/Hg, so is not visible in the "marginal" distributions shown below.

In [18]:
```
bp = sns.boxplot(data=da.loc[:, ["BPXSY1", "BPXSY2", "BPXDI1", "BPXDI2"]])
_ = bp.set_ylabel("Blood pressure in mm/Hg")
```

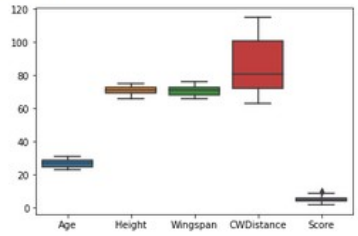


Boxplots

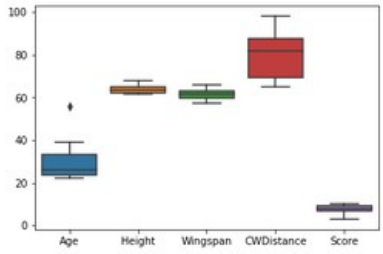
In [23]:
```
sns.boxplot(data=df.loc[:, ["Age", "Height", "Wingspan", "CWDistance", "Score"]])

plt.show()
```



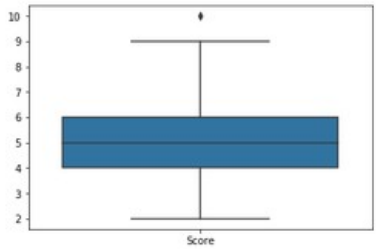
In [24]:
```
# Male Boxplot
sns.boxplot(data=df.loc[df['Gender'] == 'M', ["Age", "Height", "Wingspan", "CWDistance", "Score"]])

plt.show()
```



In [25]:
```
# Female Boxplot
sns.boxplot(data=df.loc[df['Gender'] == 'F', ["Age", "Height", "Wingspan", "CWDistance", "Score"]])

plt.show()
```



In [26]:
```
# Male Boxplot
sns.boxplot(data=df.loc[df['Gender'] == 'M', ["Score"]])

plt.show()
```



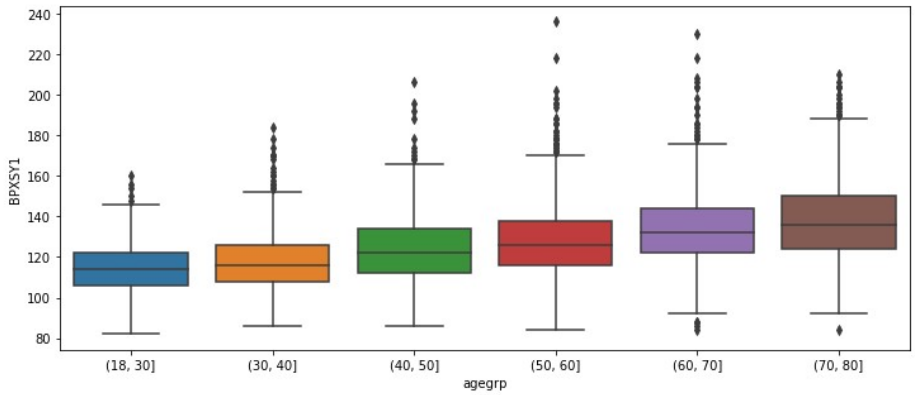
Stratification

One of the most effective ways to get more information out of a dataset is to divide it into smaller, more uniform subsets, and analyze each of these "strata" on its own. We can then formally or informally compare the findings in the different strata. When working with human subjects, it is very common to stratify on demographic factors such as age, sex, and race.

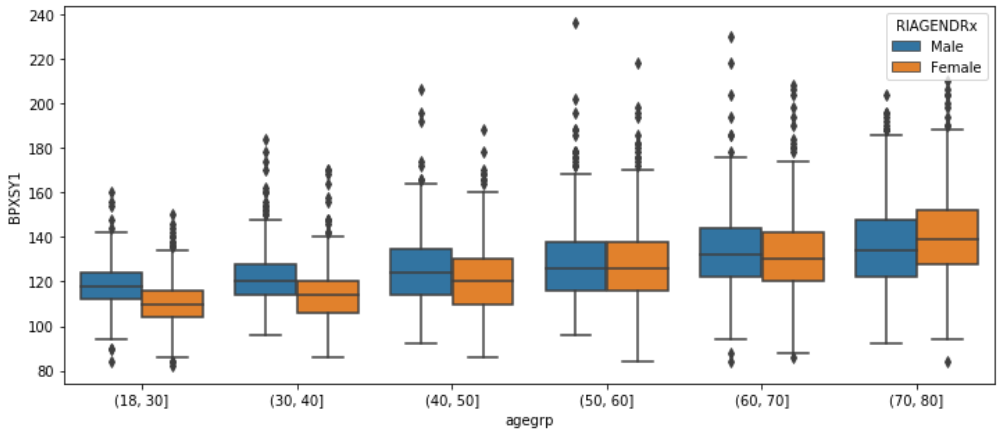
To illustrate this technique, consider blood pressure, which is a value that tends to increase with age. To see this trend in the NHANES data, we can partition the data into age strata, and construct side-by-side boxplots of the systolic blood pressure (SBP) distribution within each stratum. Since age is a quantitative variable, we need to create a series of "bins" of similar SBP values in order to stratify the data. Each box in the figure below is a summary of univariate data within a specific population stratum (here defined by age).

In [19]:
```
da["agegrp"] = pd.cut(da.RIDAGEYR, [18, 30, 40, 50, 60, 70, 80]) # Create age strata based on these cut points
plt.figure(figsize=(12, 5))  # Make the figure wider than default (12cm wide by 5cm tall)
sns.boxplot(x="agegrp", y="BPXSY1", data=da)  # Make boxplot of BPXSY1 stratified by age group
```

Out[19]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8388799ef0>
```

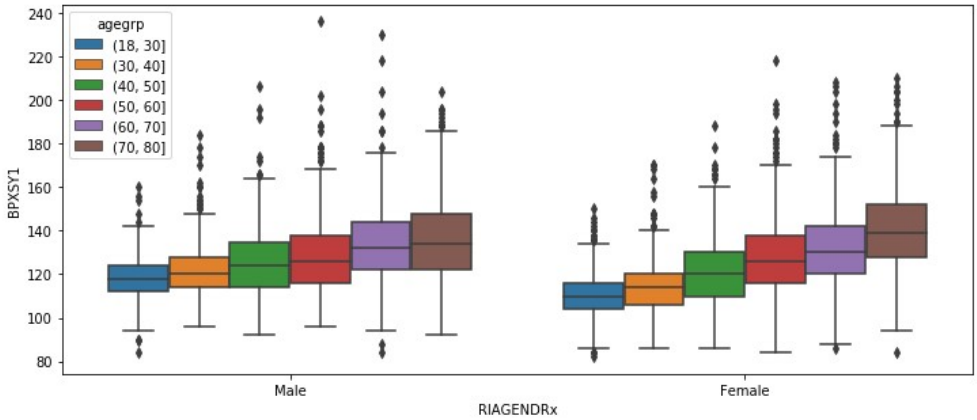


```
In [20]: da["agegrp"] = pd.cut(da.RIDAGEYR, [18, 30, 40, 50, 60, 70, 80])
         plt.figure(figsize=(12, 5))
         sns.boxplot(x="agegrp", y="BPXSY1", hue="RIAGENDRx", data=da)

Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x7f83886830f0>
```



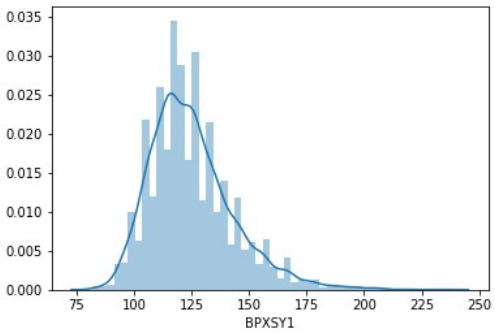
When stratifying on two factors (here age and gender), we can group the boxes first by age, and within age bands by gender, as above, or we can do the opposite -- group first by gender, and then within gender group by age bands. Each approach highlights a different aspect of the data:

```
In [21]: da["agegrp"] = pd.cut(da.RIDAGEYR, [18, 30, 40, 50, 60, 70, 80])
         plt.figure(figsize=(12, 5))
         sns.boxplot(x="RIAGENDRx", y="BPXSY1", hue="agegrp", data=da)

Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x7f838880ed68>
```



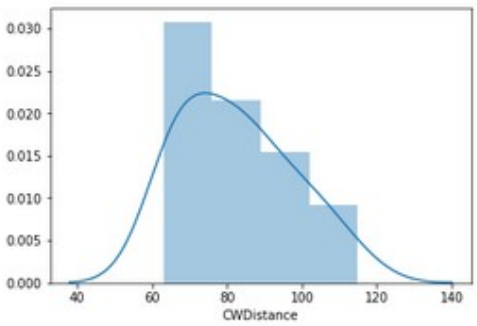
Histogram

```
In [17]: sns.distplot(da.BPXSY1.dropna())
Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x7f83e7142550>
```



Histogram

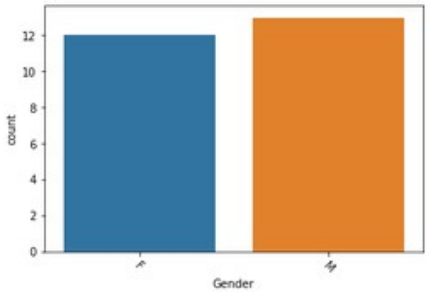
```
In [28]: # Distribution Plot (a.k.a. Histogram)
sns.distplot(df.CWDistance)

plt.show()
```



Count Plot

Count Plot

```
In [29]: # Count Plot (a.k.a. Bar Plot)
sns.countplot(x='Gender', data=df)

plt.xticks(rotation=-45)

plt.show()
```



Pandas

https://pandas.pydata.org/docs/user_guide/index.html#user-guide

```
In [2]: import numpy as np
        import pandas as pd
```

Data Management

Data management is a crucial component to statistical analysis and data science work. The following code will show how to import data via the pandas library, view your data, and transform your data.

The main data structure that Pandas works with is called a **Data Frame**. This is a two-dimensional table of data in which the rows typically represent cases (e.g. Cartwheel Contest Participants), and the columns represent variables. Pandas also has a one-dimensional data structure called a **Series** that we will encounter when accesing a single column of a Data Frame.

Pandas has a variety of functions named '`read_xxx`' for reading data in different formats. Right now we will focus on reading '`csv`' files, which stands for comma-separated values. However the other file formats include excel, json, and sql just to name a few.

This is a link to the .csv that we will be exploring in this tutorial: Cartwheel Data (Link goes to the dataset section of the Resources for this course)

There are many other options to '`read_csv`' that are very useful. For example, you would use the option `sep='\t'` instead of the default `sep=','` if the fields of your data file are delimited by tabs instead of commas. See here for the full documentation for '`read_csv`'.

Importando dados

Importing Data

```
]: # Store the url string that hosts our .csv file (note that this is a different url than in the video)
   url = "Cartwheeldata.csv"

   # Read the .csv file and store it as a pandas Data Frame
   df = pd.read_csv(url)

   # Output object type
   type(df)
```

Colunas

```
In [7]: df.columns

Out[7]: Index([u'ID', u'Age', u'Gender', u'GenderGroup', u'Glasses', u'GlassesGroup',
               u'Height', u'Wingspan', u'CWDistance', u'Complete', u'CompleteGroup',
               u'Score'],
              dtype='object')
```

Para ver os valores únicos numa coluna:

```
In [19]: # List unique values in the df['Gender'] column
         df.Gender.unique()

Out[19]: array(['F', 'M'], dtype=object)

In [20]: # Lets explore df["GenderGroup] as well
         df.GenderGroup.unique()

Out[20]: array([1, 2])
```

Sintaxe alternativa: df['Gender'].unique()

Tipos de dados

```
In [18]: df.dtypes

Out[18]: ID                int64
         Age               int64
         Gender           object
         GenderGroup       int64
         Glasses          object
         GlassesGroup      int64
         Height          float64
         Wingspan        float64
         CWDistance        int64
         Complete         object
         CompleteGroup     int64
         Score             int64
         dtype: object
```

Visualizando

Viewing Data

```
In [5]: # We can view our Data Frame by calling the head() function
        df.head()
```

Out[5]:

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistance	Complete	CompleteGroup	Score
0	1	56	F	1	Y	1	62.0	61.0	79	Y	1	7
1	2	26	F	1	Y	1	62.0	60.0	70	Y	1	8
2	3	33	F	1	Y	1	66.0	64.0	85	Y	1	7
3	4	39	F	1	N	0	64.0	63.0	87	Y	1	10
4	5	27	M	2	N	0	73.0	75.0	72	N	0	4

The head() function simply shows the first 5 rows of our Data Frame. If we wanted to show the entire Data Frame we would simply write the following:

Lets say we would like to splice our data frame and select only specific portions of our data. There are three different ways of doing so.

1. .loc()
2. .iloc()
3. .ix()

We will cover the .loc() and .iloc() splicing functions.

Filtro .loc()

The .loc() function requires two arguments, the indices of the rows and the column names you wish to observe. For instance:

> **: specifies all rows, and our column is CWDistance**. df.loc[**:,**"CWDistance"]

.loc()

.loc() takes two single/list/range operator separated by ','. The first one indicates the row and the second one indicates columns.

In [9]:
```
# Return all observations of CWDistance
df.loc[:,"CWDistance"]
```

Out[9]:
```
0      79
1      70
2      85
3      87
4      72
5      81
6     107
7      98
8     106
9      65
10     96
11     79
12     92
13     66
14     72
15    115
16     90
17     74
18     64
19     85
20     66
21    101
22     82
23     63
24     67
Name: CWDistance, dtype: int64
```

In [10]:
```
# Select all rows for multiple columns, ["CWDistance", "Height", "Wingspan"]
df.loc[:,["CWDistance", "Height", "Wingspan"]]
```

Out[10]:

	CWDistance	Height	Wingspan
0	79	62.00	61.0
1	70	62.00	60.0
2	85	66.00	64.0
3	87	64.00	63.0
4	72	73.00	75.0
5	81	75.00	71.0
6	107	75.00	76.0
7	98	65.00	62.0
8	106	74.00	73.0
9	65	63.00	60.0
10	96	69.50	66.0
11	79	62.75	58.0
12	92	65.00	64.5
13	66	61.50	57.5
14	72	73.00	74.0
15	115	71.00	72.0
16	90	61.50	59.5
17	74	66.00	66.0
18	64	70.00	69.0
19	85	68.00	66.0
20	66	69.00	67.0
21	101	71.00	70.0
22	82	70.00	68.0
23	63	69.00	71.0

```
In [11]: # Select few rows for multiple columns, ["CWDistance", "Height", "Wingspan"]
         df.loc[:9, ["CWDistance", "Height", "Wingspan"]]
```

Out[11]:

	CWDistance	Height	Wingspan
0	79	62.0	61.0
1	70	62.0	60.0
2	85	66.0	64.0
3	87	64.0	63.0
4	72	73.0	75.0
5	81	75.0	71.0
6	107	75.0	76.0
7	98	65.0	62.0
8	106	74.0	73.0
9	65	63.0	60.0

```
In [22]: # Use .loc() to specify a list of mulitple column names
         df.loc[:,["Gender", "GenderGroup"]]
```

Out[22]:

	Gender	GenderGroup
0	F	1
1	F	1
2	F	1
3	F	1
4	M	2
5	M	2
6	M	2
7	F	1
8	M	2
9	F	1

```
In [12]: # Select range of rows for all columns
         df.loc[10:15]
```

Out[12]:

	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistance	Complete	CompleteGroup	Score
10	11	30	M	2	Y	1	69.50	66.0	96	Y	1	6
11	12	28	F	1	Y	1	62.75	58.0	79	Y	1	10
12	13	25	F	1	Y	1	65.00	64.5	92	Y	1	6
13	14	23	F	1	N	0	61.50	57.5	66	Y	1	4
14	15	31	M	2	Y	1	73.00	74.0	72	Y	1	9
15	16	26	M	2	Y	1	71.00	72.0	115	Y	1	6

Filtro .iloc()

.iloc() is integer based slicing, whereas .loc() used labels/column names. Here are some examples:

```
In [14]: df.iloc[:4]
```

Out[14]:

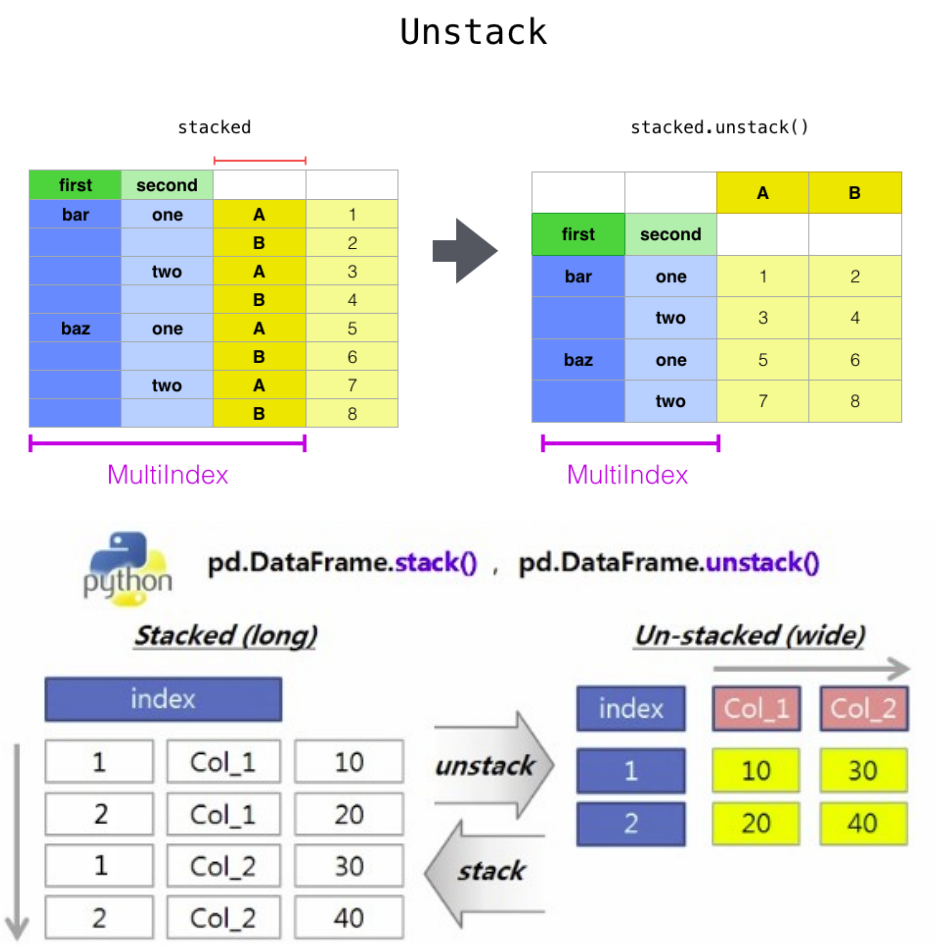
	ID	Age	Gender	GenderGroup	Glasses	GlassesGroup	Height	Wingspan	CWDistance	Complete	CompleteGroup	Score
0	1	56	F	1	Y	1	62.0	61.0	79	Y	1	7
1	2	26	F	1	Y	1	62.0	60.0	70	Y	1	8
2	3	33	F	1	Y	1	66.0	64.0	85	Y	1	7
3	4	39	F	1	N	0	64.0	63.0	87	Y	1	10

```
In [15]: df.iloc[1:5, 2:4]
```

Out[15]:

	Gender	GenderGroup
1	F	1
2	F	1
3	F	1
4	M	2

Unstack



Group by

From eyeballing the output, it seems to check out. We can streamline this by utilizing the groupby() and size() functions.

In [23]:
```
df.groupby(['Gender','GenderGroup']).size()
```

Out[23]:
```
Gender  GenderGroup
F       1              12
M       2              13
dtype: int64
```

This output indicates that we have two types of combinations.

- Case 1: Gender = F & Gender Group = 1
- Case 2: Gender = M & GenderGroup = 2.

This validates our initial assumption that these two fields essentially portray the same information.

In [22]:
```
da.groupby("agegrp")["DMDEDUC2x"].value_counts()
```

Out[22]:
```
agegrp    DMDEDUC2x
(18, 30]  Some college/AA    364
          College            278
          HS/GED             237
          Missing            128
          9-11                99
          <9                  47
(30, 40]  Some college/AA    282
          College            264
          HS/GED             182
          9-11               111
          <9                  93
(40, 50]  Some college/AA    262
          College            260
          HS/GED             171
          9-11               112
          <9                  98
(50, 60]  Some college/AA    258
          College            220
          HS/GED             220
          9-11               122
          <9                 104
(60, 70]  Some college/AA    238
          HS/GED             192
```

In [33]:
```
da.groupby(["agegrp", "RIAGENDRx"])["DMDEDUC2x"].value_counts()
```

Out[33]:
```
agegrp    RIAGENDRx  DMDEDUC2x
(18, 30]  Female     Some college/AA    207
                     College            156
                     HS/GED             119
                     Missing             56
                     9-11                44
                     <9                  27
          Male       Some college/AA    157
                     College            122
                     HS/GED             118
                     Missing             72
                     9-11                55
                     <9                  20
(30, 40]  Female     Some college/AA    159
                     College            149
                     HS/GED              78
                     <9                  46
                     9-11                42
          Male       Some college/AA    123
                     College            115
                     HS/GED             104
                     9-11                69
                     <9                  47
(40, 50]  Female     Some college/AA    157
```

```
dx = da.loc[~da.DMDEDUC2x.isin(["Don't know", "Missing"]), :]  # Eliminate rare/missing values
dx = dx.groupby(["agegrp", "RIAGENDRx"])["DMDEDUC2x"]
dx = dx.value_counts()
dx = dx.unstack() # Restructure the results from 'long' to 'wide'
dx = dx.apply(lambda x: x/x.sum(), axis=1) # Normalize within each stratum to get proportions
print(dx.to_string(float_format="%.3f"))  # Limit display to 3 decimal places
```

```
DMDEDUC2x           9-11    <9  College  HS/GED  Some college/AA
agegrp   RIAGENDRx
(18, 30] Female    0.080 0.049    0.282   0.215            0.374
         Male      0.117 0.042    0.258   0.250            0.333
(30, 40] Female    0.089 0.097    0.314   0.165            0.335
         Male      0.151 0.103    0.251   0.227            0.269
(40, 50] Female    0.110 0.106    0.299   0.173            0.313
         Male      0.142 0.112    0.274   0.209            0.262
(50, 60] Female    0.117 0.102    0.245   0.234            0.302
         Male      0.148 0.123    0.231   0.242            0.256
(60, 70] Female    0.118 0.188    0.195   0.206            0.293
         Male      0.135 0.151    0.233   0.231            0.249
(70, 80] Female    0.105 0.225    0.149   0.240            0.281
         Male      0.113 0.180    0.237   0.215            0.255
```

Value_counts

This method can be used to determine the number of times that each distinct value of a variable occurs in a data set. The value_counts method produces a table with two columns. The first column contains all distinct observed values for the variable. The second column contains the number of times each of these values occurs. This method excludes missing values

In statistical terms, this is the "frequency distribution" of the variable.

Below we show the frequency distribution of the DMDEDUC2 variable, which is a variable that reflects a person's level of educational attainment.

```
In [3]: da.DMDEDUC2.value_counts()

Out[3]: 4.0    1621
        5.0    1366
        3.0    1186
        1.0     655
        2.0     643
        9.0       3
        Name: DMDEDUC2, dtype: int64
```

Em proporção:

For many purposes it is more relevant to consider the proportion of the sample with each of the possible category values, rather than the number of people in each category. We can do this as follows:

```
In [8]: x = da.DMDEDUC2x.value_counts()  # x is just a name to hold this value temporarily
        x / x.sum()

Out[8]: Some college/AA    0.296127
        College            0.249543
        HS/GED             0.216661
        <9                 0.119657
        9-11               0.117464
        Don't know         0.000548
        Name: DMDEDUC2x, dtype: float64
```

Dealing with missing values

Note that the `value_counts` method excludes missing values. We confirm this below by adding up the number of observations with a DMDEDUC2 value equal to 1, 2, 3, 4, 5, or 9 (there are 5474 such rows), and comparing this to the total number of rows in the data set, which is 5735. This tells us that there are 5735 - 5474 = 261 missing values for this variable (other variables may have different numbers of missing values).

In [4]:
```
print(da.DMDEDUC2.value_counts().sum())
print(1621 + 1366 + 1186 + 655 + 643 + 3) # Manually sum the frequencies
print(da.shape)
```

```
5474
5474
(5735, 28)
```

Another way to obtain this result is to locate all the null (missing) values in the data set using the isnull Pandas function, and count the number of such locations.

In [5]:
```
pd.isnull(da.DMDEDUC2).sum()
```

Out[5]:
```
261
```

Preenchendo missing values com um valor personalizado:

In some cases we will want to treat the missing response category as another category of observed response, rather than ignoring it when creating summaries. Below we create a new category called "Missing", and assign all missing values to it usig fillna. Then we recalculate the frequency distribution. We see that 4.6% of the responses are missing.

In [9]:
```
da["DMDEDUC2x"] = da.DMDEDUC2x.fillna("Missing")
x = da.DMDEDUC2x.value_counts()
x / x.sum()
```

Out[9]:
```
Some college/AA    0.282650
College            0.238187
HS/GED             0.206800
<9                 0.114211
9-11               0.112119
Missing            0.045510
Don't know         0.000523
Name: DMDEDUC2x, dtype: float64
```

Replace

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.replace.html

In some cases it is useful to replace integer codes with a text label that reflects the code's meaning. Below we create a new variable called 'DMDEDUC2x' that is recoded with text labels, then we generate its frequency distribution.

In [6]:
```
da["DMDEDUC2x"] = da.DMDEDUC2.replace({1: "<9", 2: "9-11", 3: "HS/GED", 4: "Some college/AA", 5: "College",
                                       7: "Refused", 9: "Don't know"})
da.DMDEDUC2x.value_counts()
```

Out[6]:
```
Some college/AA    1621
College            1366
HS/GED             1186
<9                  655
9-11                643
Don't know            3
Name: DMDEDUC2x, dtype: int64
```

We will also want to have a relabeled version of the gender variable, so we will construct that now as well. We will follow a convention here of appending an 'x' to the end of a categorical variable's name when it has been recoded from numeric to string (text) values.

In [7]:
```
da["RIAGENDRx"] = da.RIAGENDR.replace({1: "Male", 2: "Female"})
```

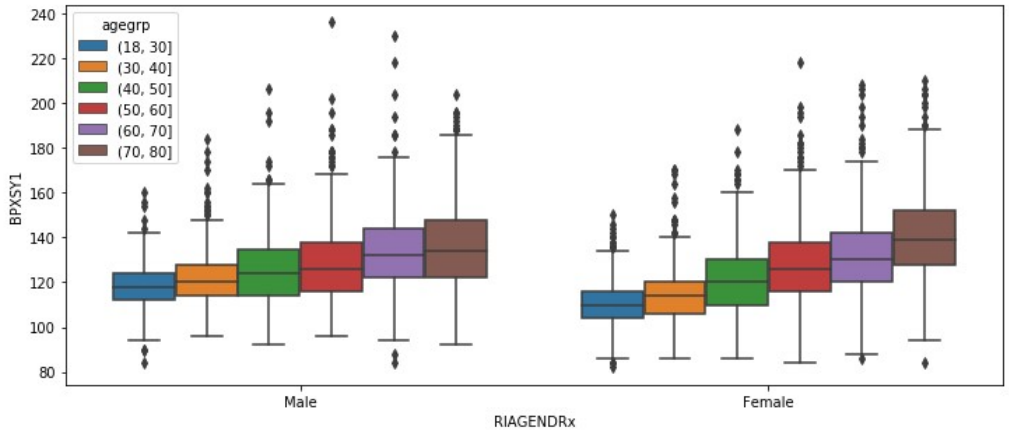
For many purposes it is more relevant to consider the proportion of the sample with each of the possible category values, rather than the number of people in each category. We can do this as follows:

```
In [14]: da['DMDMARTL'].value_counts()

Out[14]: 1.0     2780
         5.0     1004
         3.0      579
         6.0      527
         2.0      396
         4.0      186
         77.0       2
         Name: DMDMARTL, dtype: int64

In [24]: da['DMDMARTL_LABEL']=da['DMDMARTL'].replace({1:"Married", 2:"Widowed", 3:"Divorced", 4:"Separated", 5:"Never married",
                        6:"Living with partner", 77:"Refused",99:"Dont Know"})

In [26]: da['DMDMARTL_LABEL'].value_counts()

Out[26]: Married                2780
         Never married          1004
         Divorced                579
         Living with partner     527
         Widowed                 396
         Separated               186
         Refused                   2
         Name: DMDMARTL_LABEL, dtype: int64
```

Cut

Use cut when you need to segment and sort data values into bins. This function is also useful for going from a continuous variable to a categorical variable. For example, cut could convert ages to groups of age ranges. Supports binning into an equal number of bins, or a pre-specified array of bins.
https://pandas.pydata.org/docs/reference/api/pandas.cut.html

```
In [21]: da["agegrp"] = pd.cut(da.RIDAGEYR, [18, 30, 40, 50, 60, 70, 80])
         plt.figure(figsize=(12, 5))
         sns.boxplot(x="RIAGENDRx", y="BPXSY1", hue="agegrp", data=da)

Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x7f838880ed68>
```



da['agegr'] = pd.cut(da.RIDAGEYR,[10,20,30,40,50,60,70,80])
da_female_age = da.groupby(['RIAGENDR','agegr'])['DMDMARTL_LABEL'].value_counts()
da_female_age

```
In [17]: da['agegr'] = pd.cut(da.RIDAGEYR,[10,20,30,40,50,60,70,80])
         da_female_age = da.groupby(['RIAGENDR','agegr'])['DMDMARTL_LABEL'].value_counts()
         da_female_age
```

```
Out[17]: RIAGENDR  agegr     DMDMARTL_LABEL
         Female    (10, 20]  Never married          30
                             Living with partner     8
                             Married                 1
                   (20, 30]  Never married         229
                             Married               157
                             Living with partner   106
                             Divorced               11
                             Separated              11
                   (30, 40]  Married               258
                             Never married          97
                             Living with partner    57
                             Divorced               43
                             Separated              17
                             Widowed                 2
                   (40, 50]  Married               288
                             Divorced               69
                             Never married          63
                             Living with partner    37
                             Separated              33
                             Widowed                12
                   (50, 60]  Married               257
```

Describe

Retorna um resumo numérico com as principais estatísticas para variáveis quantitativas.

```
In [10]: da.BMXWT.dropna().describe()
```

```
Out[10]: count    5666.000000
         mean       81.342676
         std        21.764409
         min        32.400000
         25%        65.900000
         50%        78.200000
         75%        92.700000
         max       198.900000
         Name: BMXWT, dtype: float64
```

It's also possible to calculate individual summary statistics from one column of a data set. This can be done using Pandas methods, or with numpy functions:

```
In [11]: x = da.BMXWT.dropna()  # Extract all non-missing values of BMXWT into a variable called 'x'
         print(x.mean()) # Pandas method
         print(np.mean(x)) # Numpy function

         print(x.median())
         print(np.percentile(x, 50))  # 50th percentile, same as the median
         print(np.percentile(x, 75))  # 75th percentile
         print(x.quantile(0.75)) # Pandas method for quantiles, equivalent to 75th percentile
```

```
81.34267560889516
81.34267560889516
78.2
78.2
92.7
92.7
```

Dummy Variables

Permite pegar valores de uma coluna e transformar em colunas e contabilizar “presença” (1) e “ausência” (0).

Making a dummy variable will take all the `k` distinct values in one column and make `k` columns out of them.

Let's look at an example below:

```
In [24]: tips.head()
```

Out[24]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

```
In [25]: pd.get_dummies(tips.head(), columns=['sex'])
```

Out[25]:

	total_bill	tip	smoker	day	time	size	sex_Male	sex_Female
0	16.99	1.01	No	Sun	Dinner	2	0	1
1	10.34	1.66	No	Sun	Dinner	3	1	0
2	21.01	3.50	No	Sun	Dinner	3	1	0
3	23.68	3.31	No	Sun	Dinner	2	1	0
4	24.59	3.61	No	Sun	Dinner	4	0	1

Notice the sex column was split into the sex_Male and sex_Female column. When the sex is female the sex_Female is 1 and 0 otherwise. And similarly for the sex_Male column.

This can be very useful for ML models and doing some types of analysis.

Pivot

What about Melting and Pivoting?

That is about it when it comes to stacking and unstacking. Anything you can do with melting and pivoting ca
a single example from pandas:

```
In [17]: cheese = pd.DataFrame({'first': ['John', 'Mary'],
                                'last': ['Doe', 'Bo'],
                                'height': [5.5, 6.0],
                                'weight': [130, 150]})
         cheese
```

Out[17]:

	first	last	height	weight
0	John	Doe	5.5	130
1	Mary	Bo	6.0	150

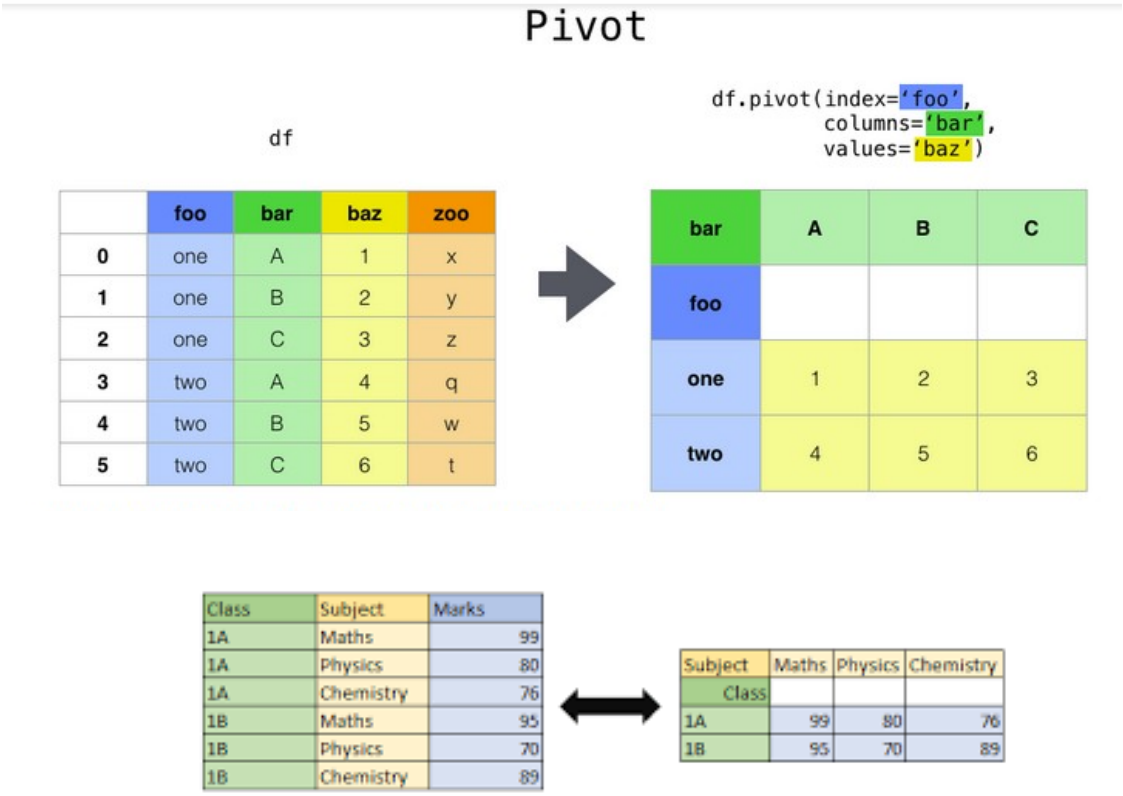
```
In [18]: # melt does stacking in one operation
         cheese.melt(id_vars=['first', 'last'])
```

Out[18]:

	first	last	variable	value
0	John	Doe	height	5.5
1	Mary	Bo	height	6.0
2	John	Doe	weight	130.0
3	Mary	Bo	weight	150.0

To do this with stacking we just need to do it in two steps:

```
In [ ]: cheese.set_index(['first', 'last'], inplace=True)
        cheese.stack().reset_index()
```



https://www.geeksforgeeks.org/python-pandas-pivot_table/
https://kanoki.org/2019/07/24/how-to-create-pandas-pivot-table-and-crosstab/

Create a spreadsheet-style pivot table as a DataFrame.

	site	Product_Category	Product	Sales
0	walmart	Kitchen	Oven	2000
1	amazon	Home-Decor	Sofa-set	3000
2	alibaba	Gardening	digging spade	4000
3	flipkart	Health	fitness band	5000
4	alibaba	Beauty	sunscreen	6000
5	flipkart	Garments	pyjamas	9000
6	walmart	Gardening	digging spade	3000
7	amazon	Health	fitness band	2500
8	alibaba	Beauty	sunscreen	1020
9	flipkart	Garments	pyjamas	950

Create Pivot Table

```
ivot_table( index=['Product_Category', 'Product'], values=['Sales'], columns=['site'])
```

		Sales			
	site	alibaba	amazon	flipkart	walmart
Product_Category	Product				
Beauty	sunscreen	3510.0	NaN	NaN	NaN
Gardening	digging spade	4000.0	NaN	NaN	3000.0
Garments	pyjamas	NaN	NaN	4975.0	NaN
Health	fitness band	NaN	2500.0	5000.0	NaN
Home-Decor	Sofa-set	NaN	3000.0	NaN	NaN
Kitchen	Oven	NaN	NaN	NaN	2000.0

df.pivot_table(index=['Product_Category', 'Product'], values=['Sales'], columns=['site'])

Dados Categóricos

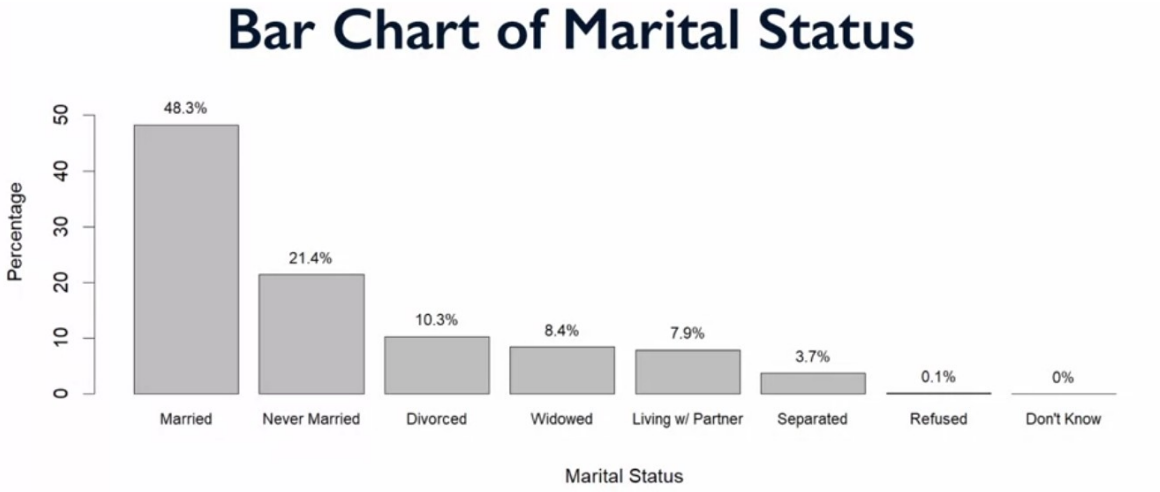
- **Frequency Tables**
  - Great for numerical summaries
- **Bar Charts**
  - Great for visualization

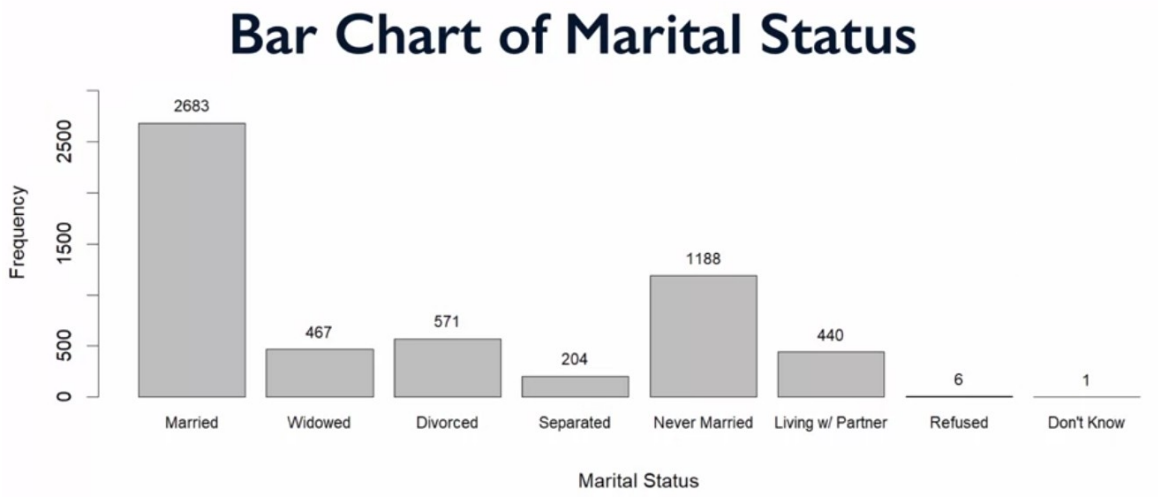
Use gráficos de pizza com moderação.

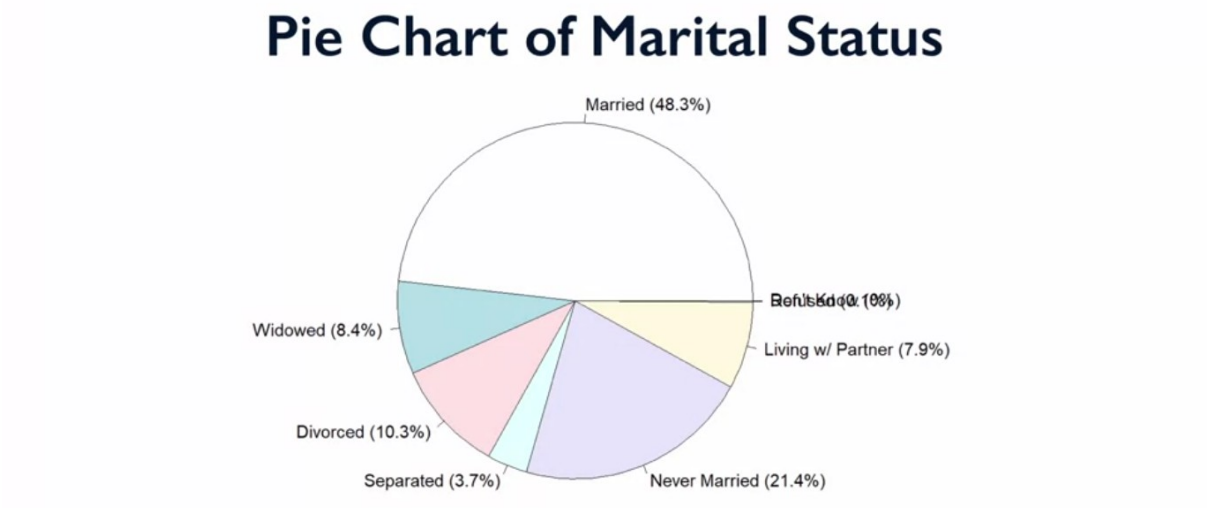
- Counts
- Percentages

Frequency Table

Marital Status	Count	Percent
Married	2683	48.3%
Widowed	467	8.4%
Divorced	571	10.3%
Separated	204	3.7%
Never Married	1188	21.4%
Living w/ Partner	440	7.9%
Refused	6	0.1%
Don't Know	1	0.0%
Total	5560	100%


Bar Chart of Marital Status
Percentage
0
10
20
30
40
50
48.3%
21.4%
10.3%
8.4%
7.9%
3.7%
0.1%
0%
Married
Never Married
Divorced
Widowed
Living w/ Partner
Separated
Refused
Don't Know
Marital Status


Bar Chart of Marital Status
Frequency
0
500
1500
2500
2683
467
571
204
1188
440
6
1
Married
Widowed
Divorced
Separated
Never Married
Living w/ Partner
Refused
Don't Know
Marital Status


Pie Chart of Marital Status
Married (48.3%)
Widowed (8.4%)
Divorced (10.3%)
Separated (3.7%)
Never Married (21.4%)
Living w/ Partner (7.9%)

Dados Quantitativos

Histogramas

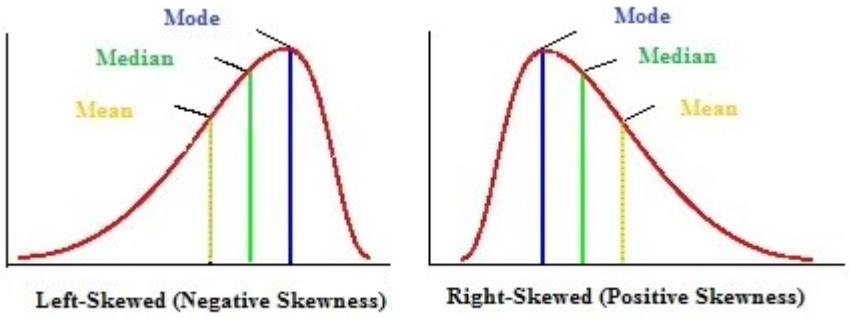
- Histograms allow us to display data graphically
- 4 main aspects we use to describe the data
  - **Shape**
  - **Center**
  - **Spread**
  - **Outliers**
- Your one sentence summary should allow for any person to read it and have a general understand of what your data looks like

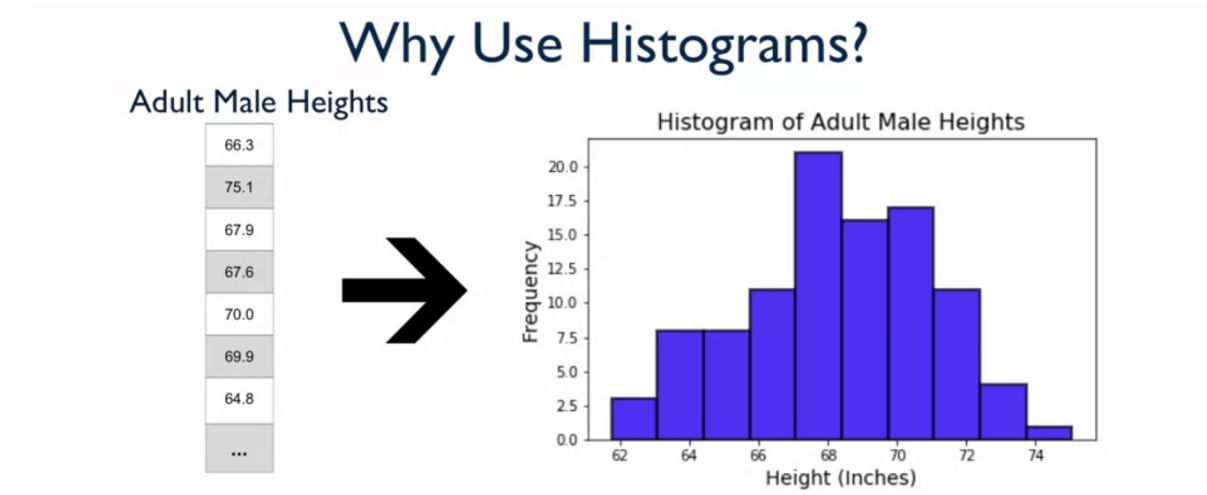
Shape - Overall appearance of histogram. Can be symmetric, bell-shaped, left skewed, right skewed, etc

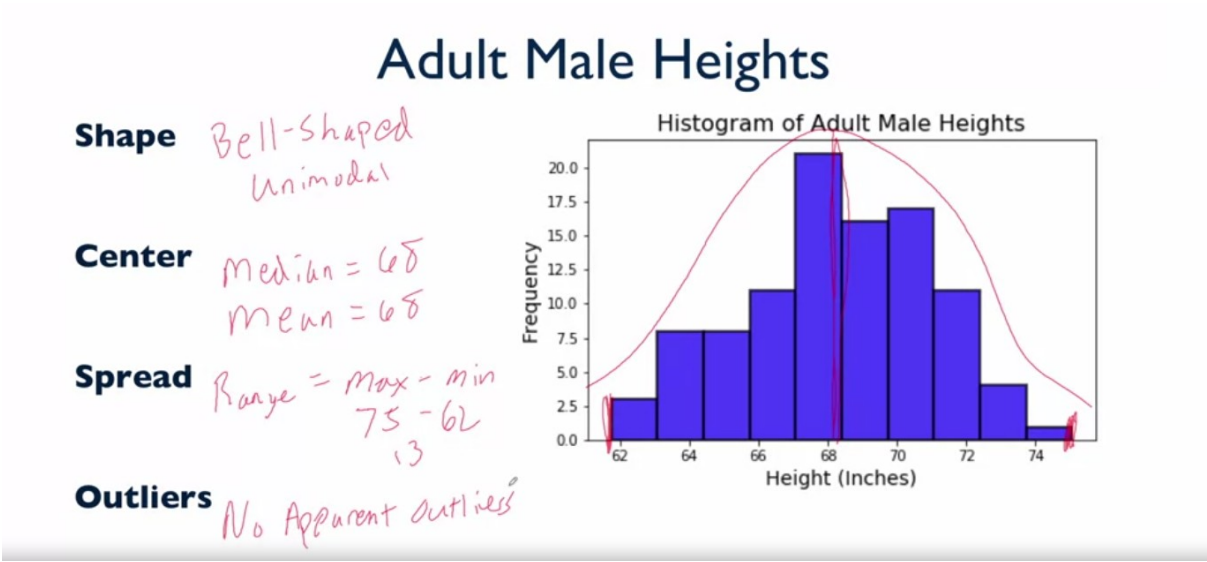
Center - Mean or Median

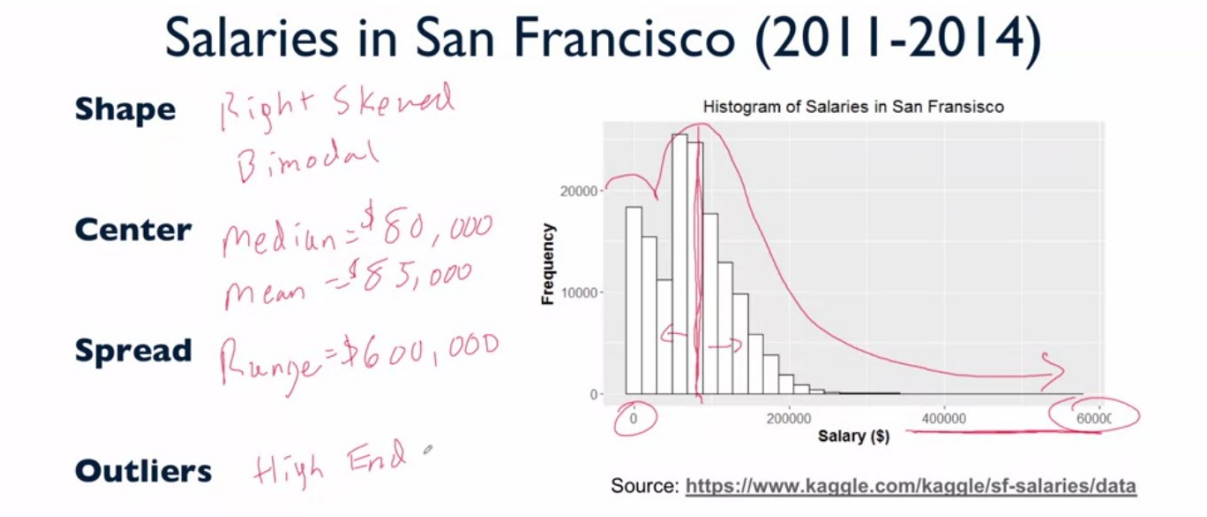
Spread - How far our data spreads. Range, Interquartile Range (IQR), standard deviation, variance.

Outliers - Data points that fall far from the bulk of the data



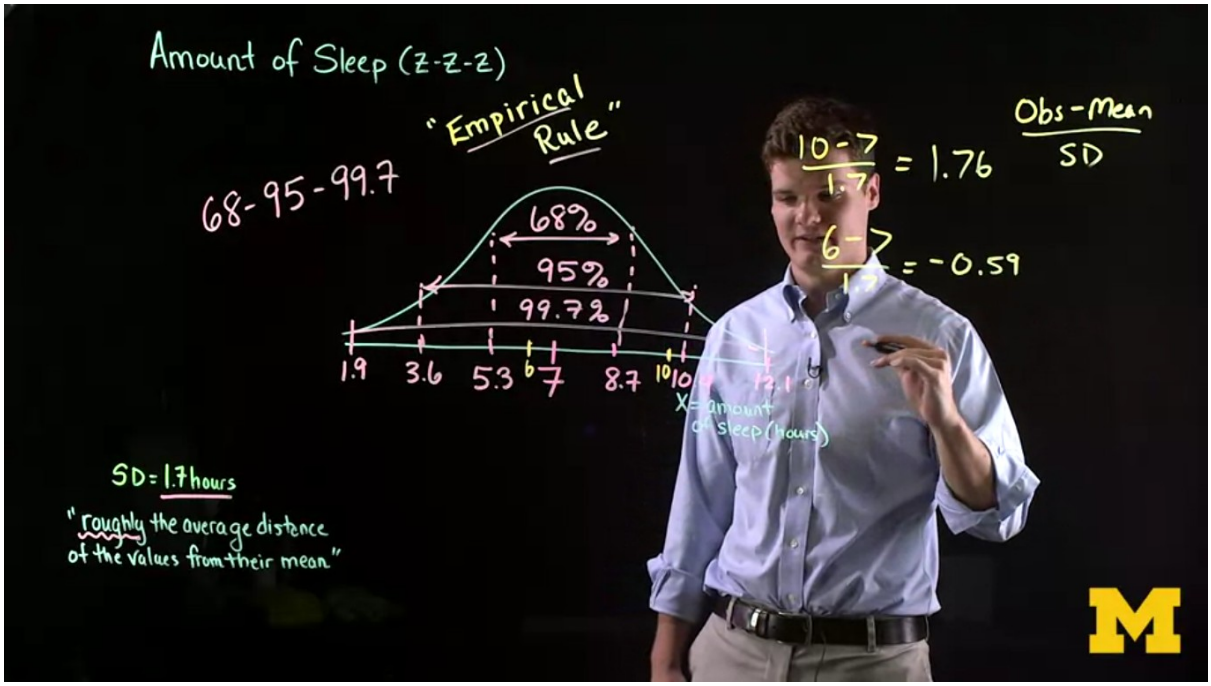




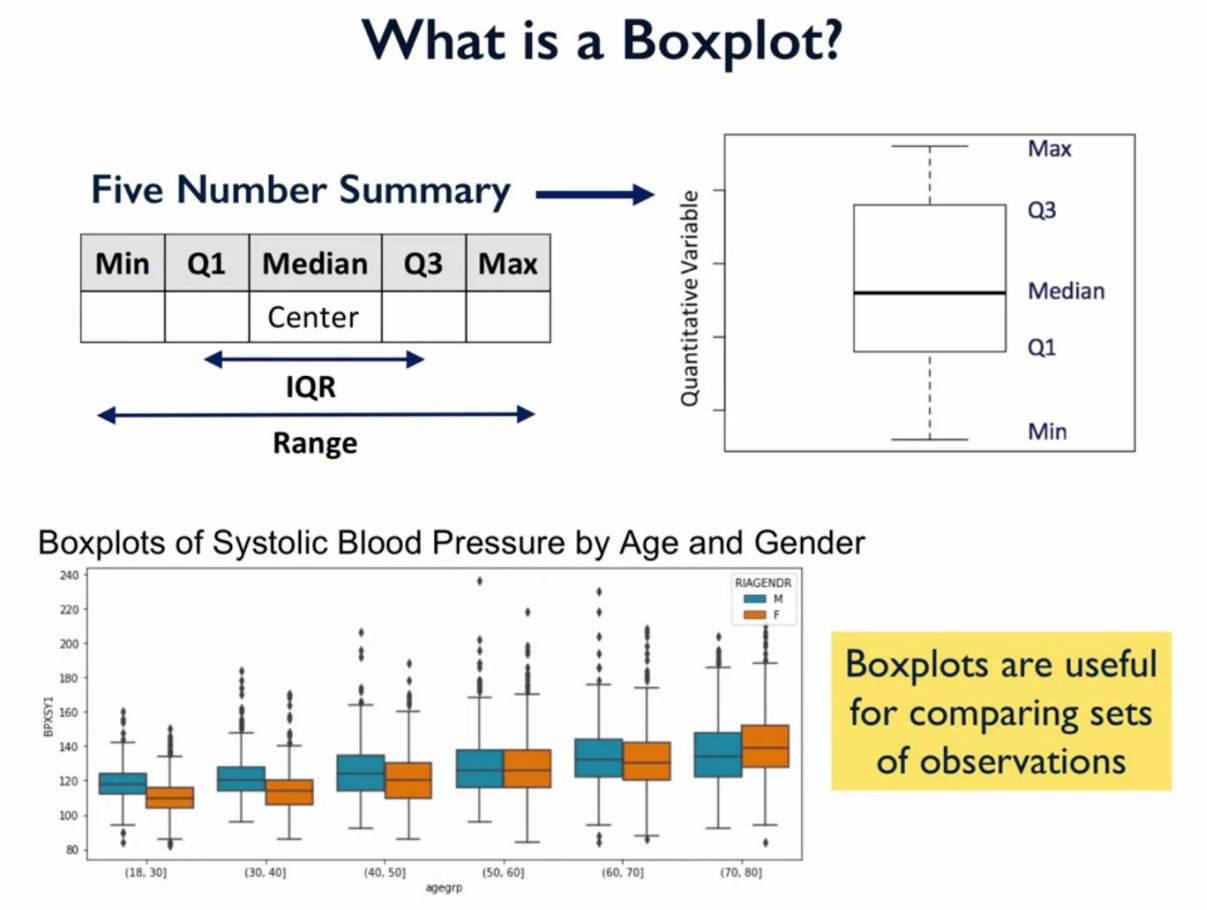


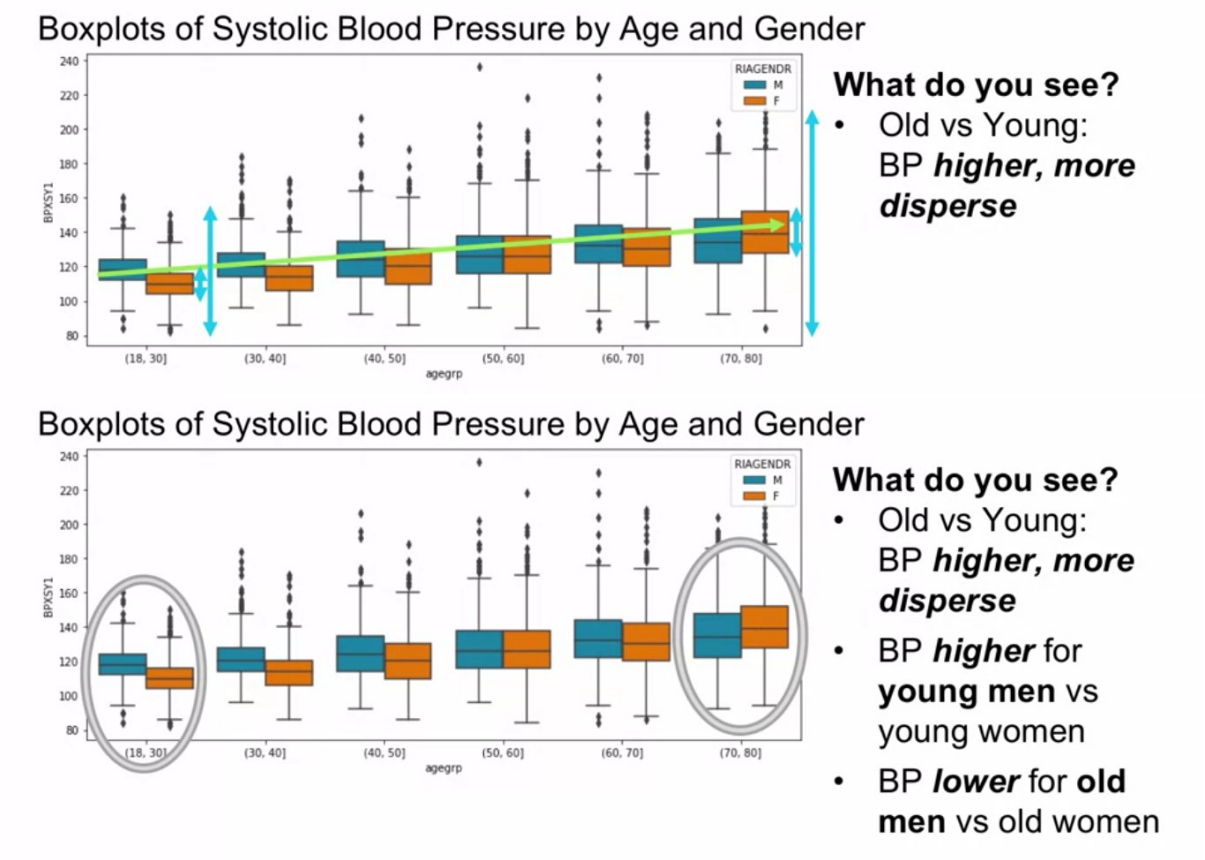
Normalização (Standard Score)

Valor Z = (valor observado - média) / desvio padrão


Amount of Sleep (z-z-z)
"Empirical Rule"
68-95-99.7
68%
95%
99.7%
1.9
3.6
5.3
6
7
8.7
10
10
12.1
X = amount of sleep (hours)
10-7/1.7 = 1.76
Obs-Mean/SD
6-7/1.7 = -0.59
SD = 1.7 hours
"roughly the average distance of the values from their mean"

Boxplots


What is a Boxplot?
Five Number Summary
Min
Q1
Median
Q3
Max
Center
IQR
Range
Quantitative Variable
Max
Q3
Median
Q1
Min
Boxplots of Systolic Blood Pressure by Age and Gender
BPXSY1
RIAGENDR
M
F
(18, 30]
(30, 40]
(40, 50]
(50, 60]
(60, 70]
(70, 80]
agegrp
Boxplots are useful for comparing sets of observations



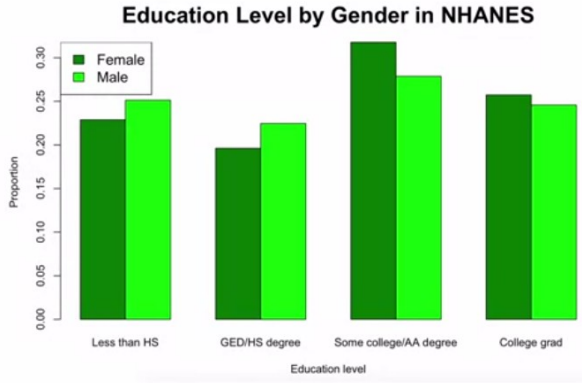
Dados Multivariados

https://flowingdata.com/2016/03/03/marrying-age/

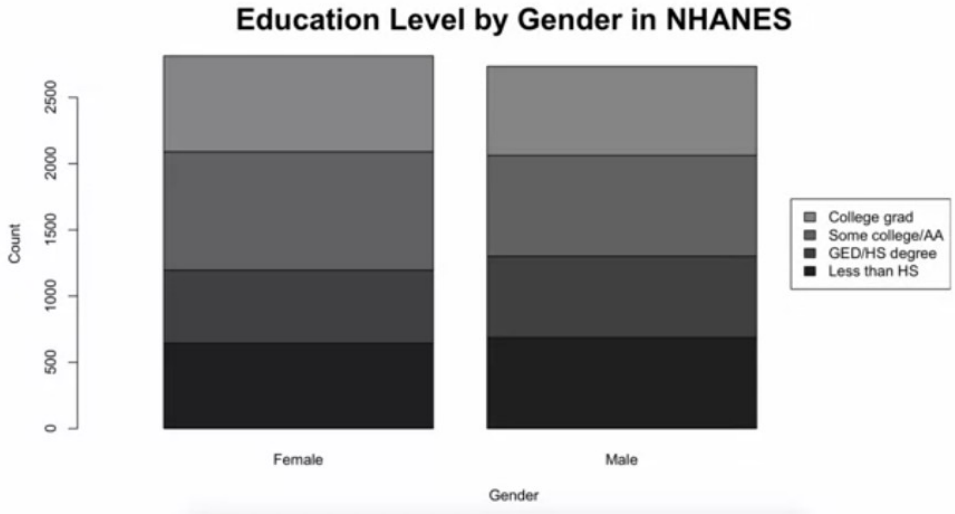
Marginal Distribution of Education Level

		Female	Male	Total
Highest Education Level Attained	Less than HS degree	644	687	1331 (24.0%)
	HS degree or GED	552	614	1166 (21.0%)
	Some college or AA	894	762	1656 (29.8%)
	College degree or above	724	672	1396 (25.2%)
	Total	2814	2735	5549 (100%)

Side-by-side Bar Chart of Education Level



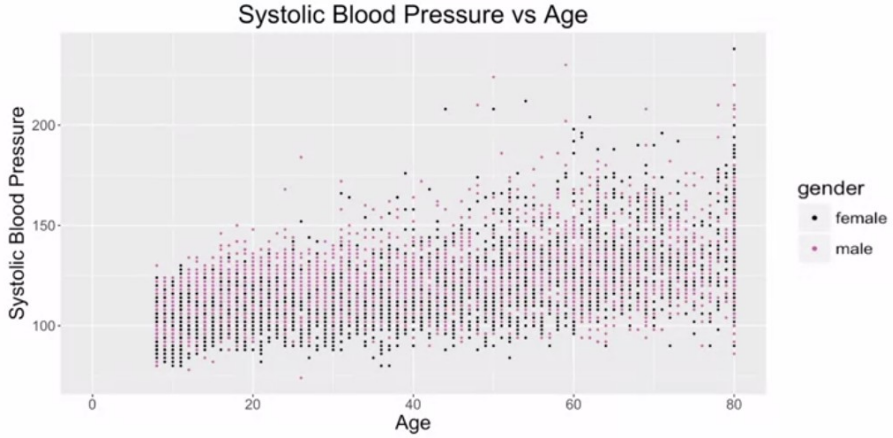
Stacked Bar Chart of Education Level



Mosaic Plot of Education Level, Age, and Gender

- Scatterplots for visualization
- Describing association through
  - Type
  - Direction
  - Strength
- Correlation as a way to numerically describe association
- Identifying potential outliers

Displaying Quantitative and Categorical Data

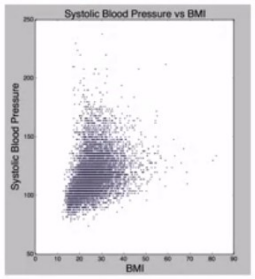


Correlation

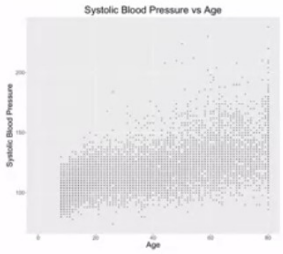
Pearson correlation (R or ρ): number between -1 and 1 indicating the strength and sign of association between 2 variables

The sign of the correlation is the sign of the association

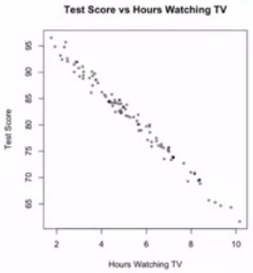
The closer the number is to 1 or -1, the stronger the association



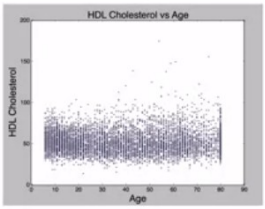
R = 0.30



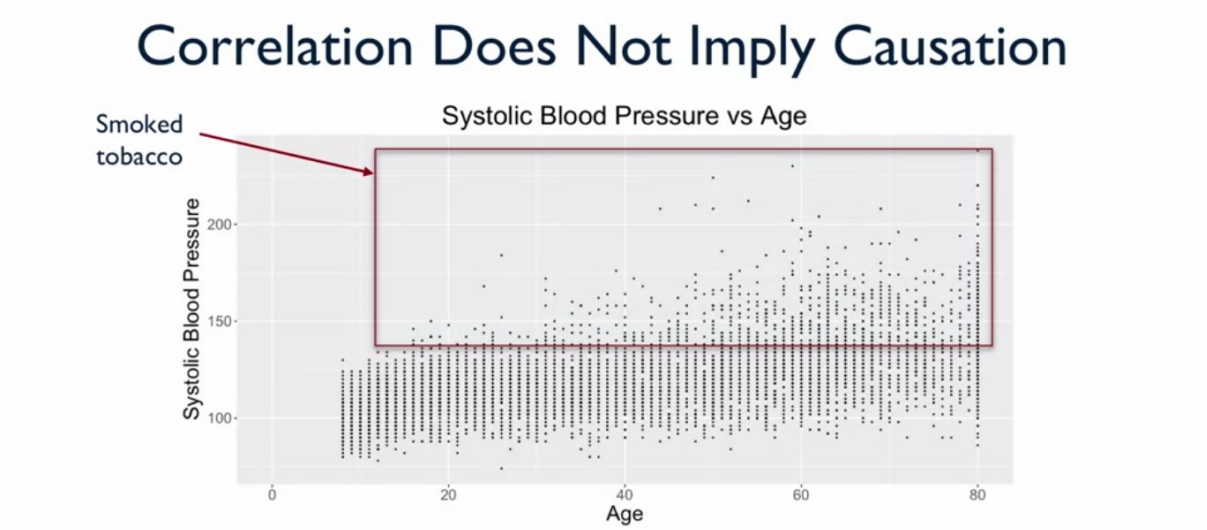
R = 0.58



R = -0.99

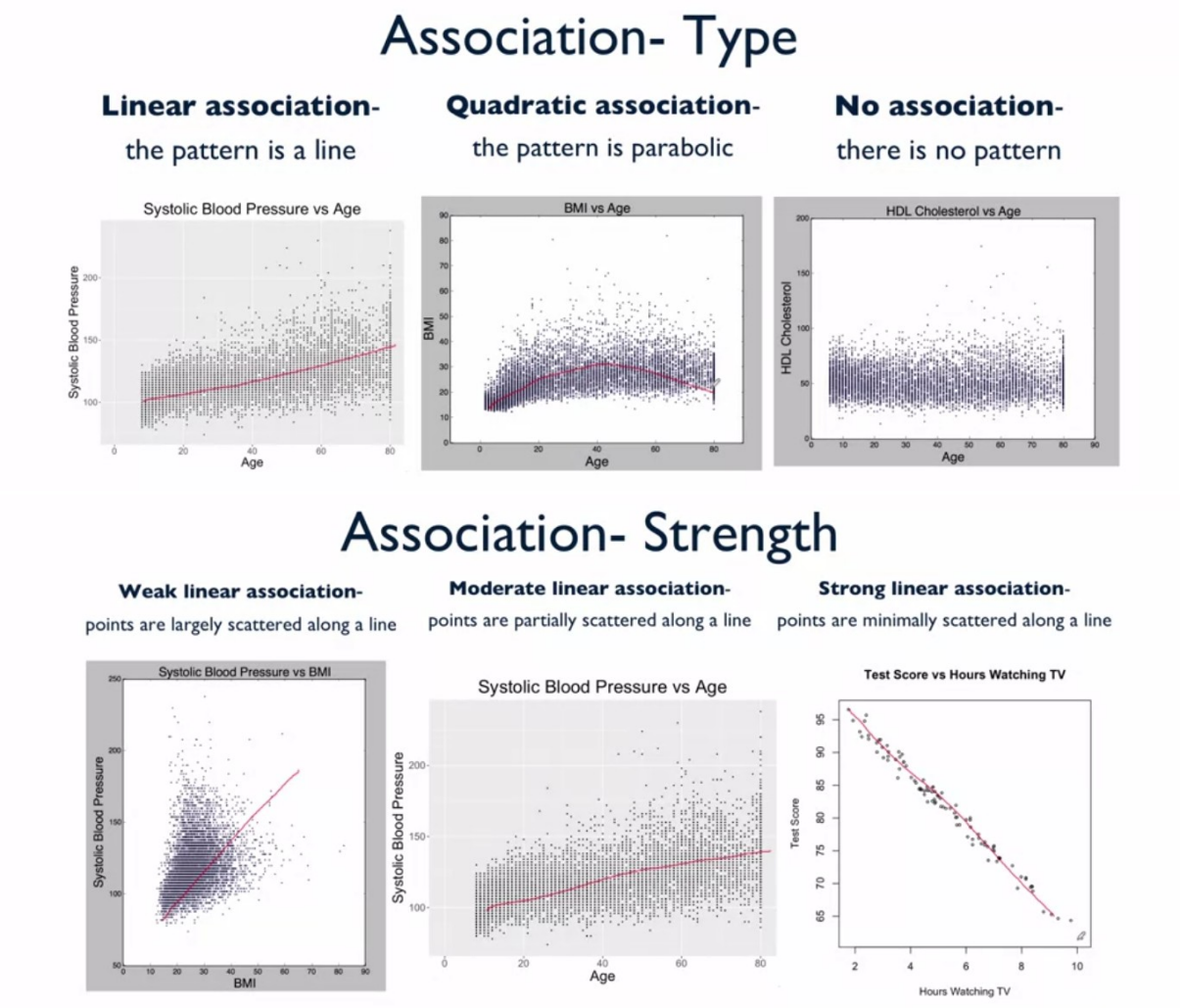


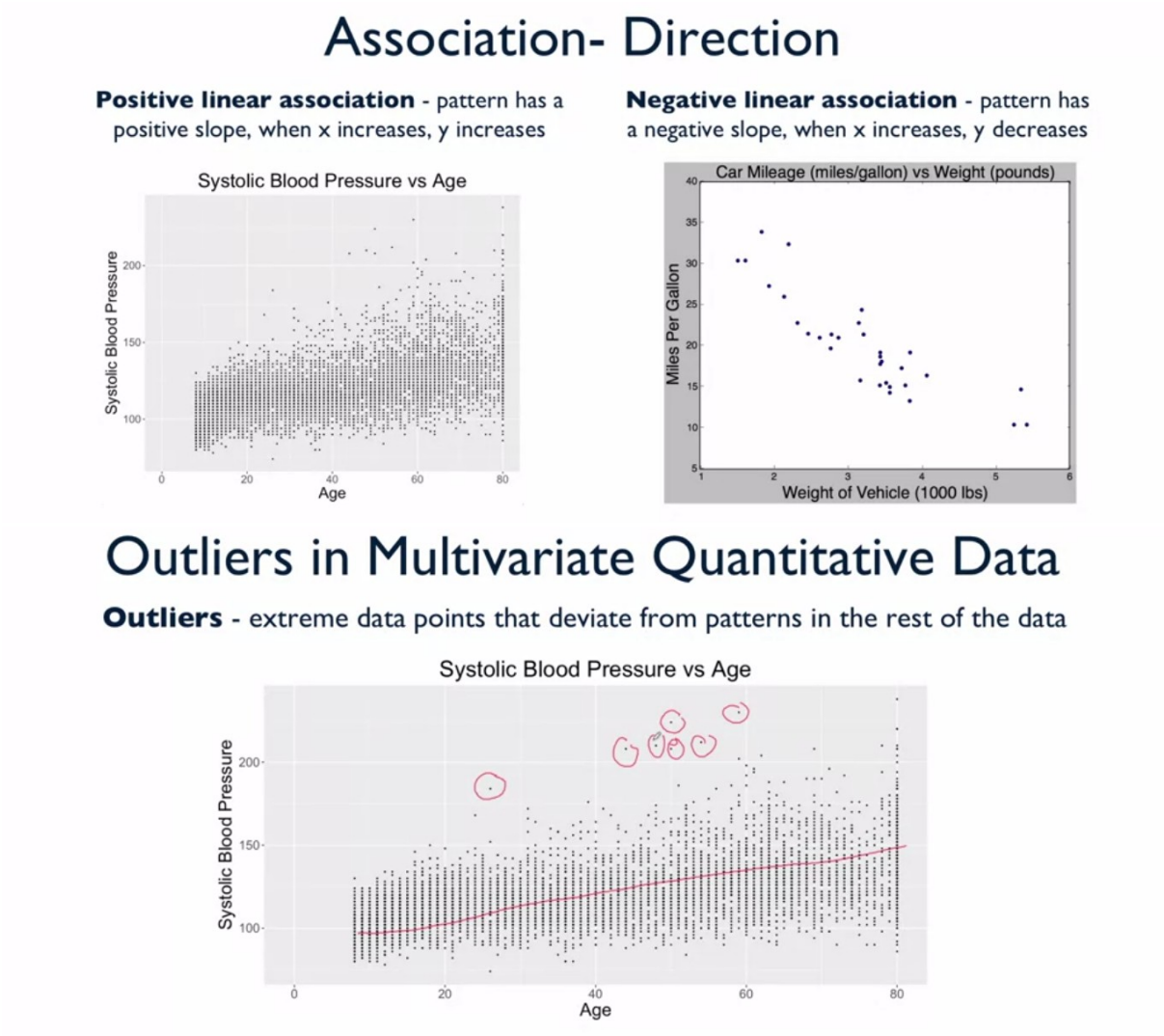
R = -.01



Outros fatores podem estar relacionados ao padrão observado.

Atente-se a outliers, tipo, direção e força da correlação…





Paradoxo de Simpson

Será que a forma em que coletamos os dados de um processo, pode influenciar nossas conclusões?

A confounding variable is an outside influence that changes the relationship between the independent and the dependent variable. It oftentimes works by affecting the causal relationship between the primary independent variable and the dependent variable. This confounding variable confuses the relationship between two other variables; it may act by hiding, obscuring, or enhancing the existing relationship.
For example, suppose that you are interested in examining how activity level affects weight change. Other factors, like diet, age, and gender may also affect weight change and come into play when looking at the relationship between activity level and weight change. If one doesn't control for these factors, the relationship between activity level and weight change can be distorted.

Sampling Distribution

The “mean of the sampling distribution of the means”. In a nutshell, this is the same as the population mean. For example, if your population mean (μ) is 99, then the mean of the sampling distribution of the mean, μ_m, is also 99 (as long as you have a sufficiently large sample size). You can think of a sampling distribution as a relative frequency distribution with a large number of samples.
It is a distribution created by every possible mean from every possible sample. And it resembles a normal distribution.

The sampling distribution of the sample mean is very useful because it can tell us the probability of getting any specific mean from a random sample. Put more simply, we can use this distribution to tell us how far off our own sample mean is from all other possible means, and use this to inform decisions and estimates in null hypothesis statistical testing.

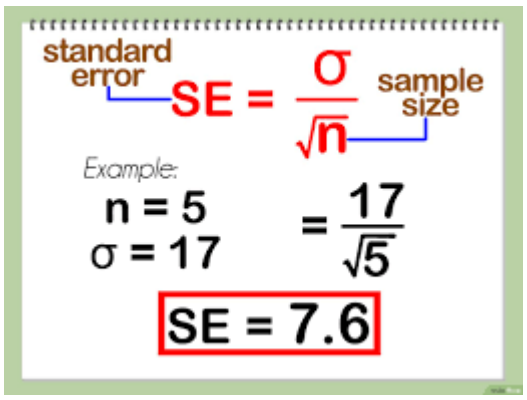
As the sample size increases, distribution of the mean will approach the population mean of μ, and the variance will approach σ^2/N, where N is the sample size.

Central Limit Theorem

The **central limit theorem** tells us that if we have a **large number** of independent, identically distributed variables, the **distribution will approximately follow a normal distributio**n. It doesn’t matter what the underlying distribution is.

Standard Error of the Mean (SE)

It is the standard deviation of the sampling distribution, calculated by dividing the standard deviation by the square root of the sample size (n) for a given sample.



https://www.psychologyinaction.org/psychology-in-action-1/2016/08/13/what-is-a-sampling-distribution

T test

The *t* statistic for an independent samples *t* test uses the Standard Error as the denominator. The numerator of this *t* statistic is the difference in means between group 1 and group 2, and the denominator is the standard deviation of all possible means from all possible samples. What the *t* value then represents is how different the means of group 1 and group 2 are in standard units.

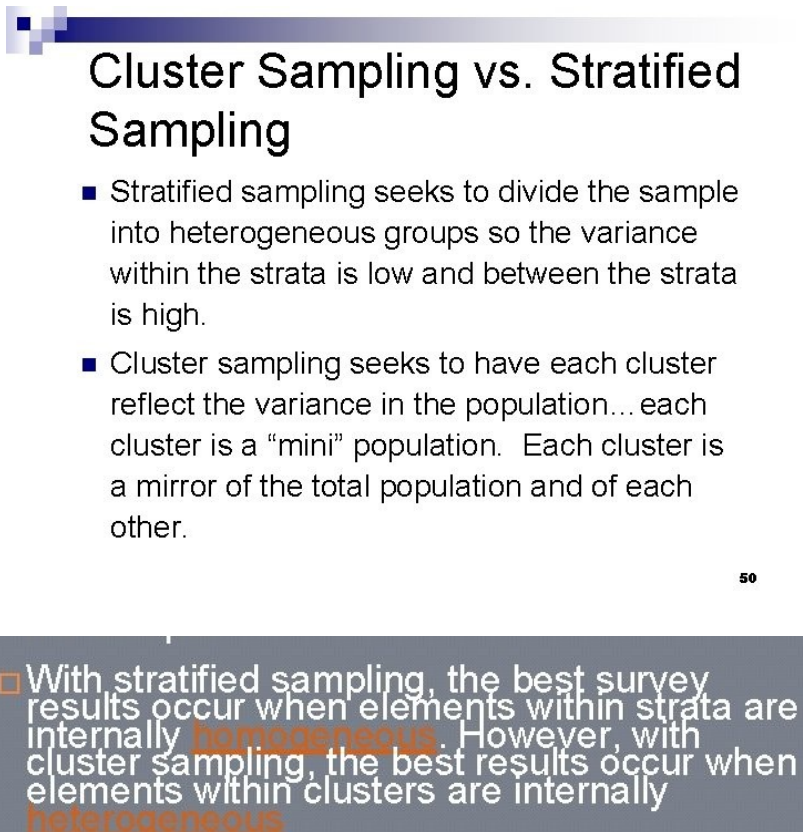
$$t = \frac{\overline{X}_1 - \overline{X}_2}{SEM}$$

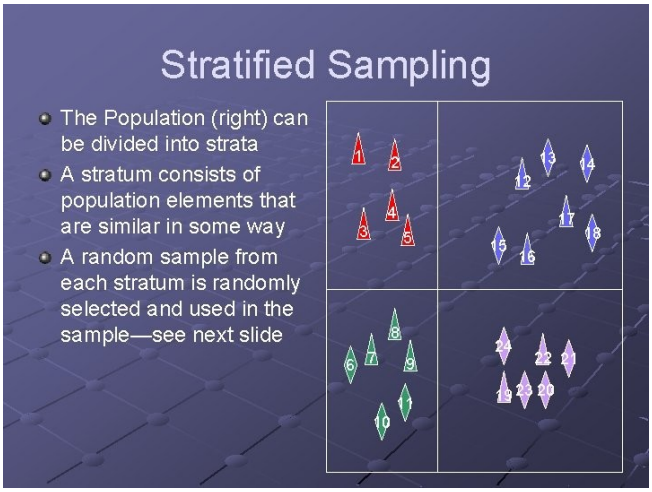
Further, to get a confidence interval of your mean estimate for an independent samples *t* test, you also use the Standard Error:

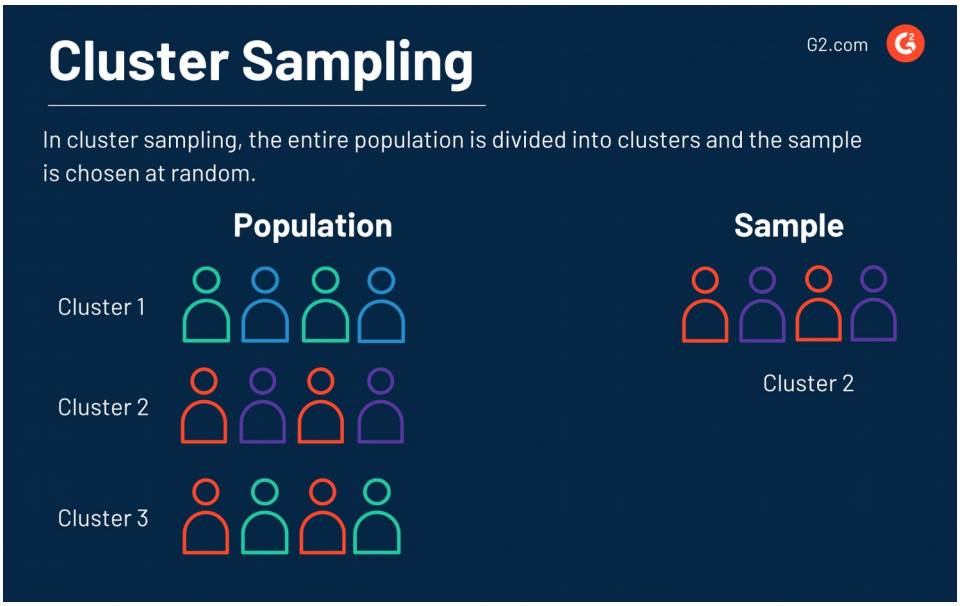
$$\text{Upper Limit} = \text{Mean } (\bar{x}) + t_{\alpha_2} * SEM$$

$$\text{Lower Limit} = \text{Mean } (\bar{x}) - t_{\alpha_2} * SEM$$

Cluster Sampling x Stratified Sampling







Complex Samples

Example of finding a unit's probability of selection:

- Select a out of A clusters at random in a given stratum
- then select b out of B units at random from within a selected cluster

Probability of selection: $\left(\frac{a}{A}\right)\left(\frac{b}{B}\right)$

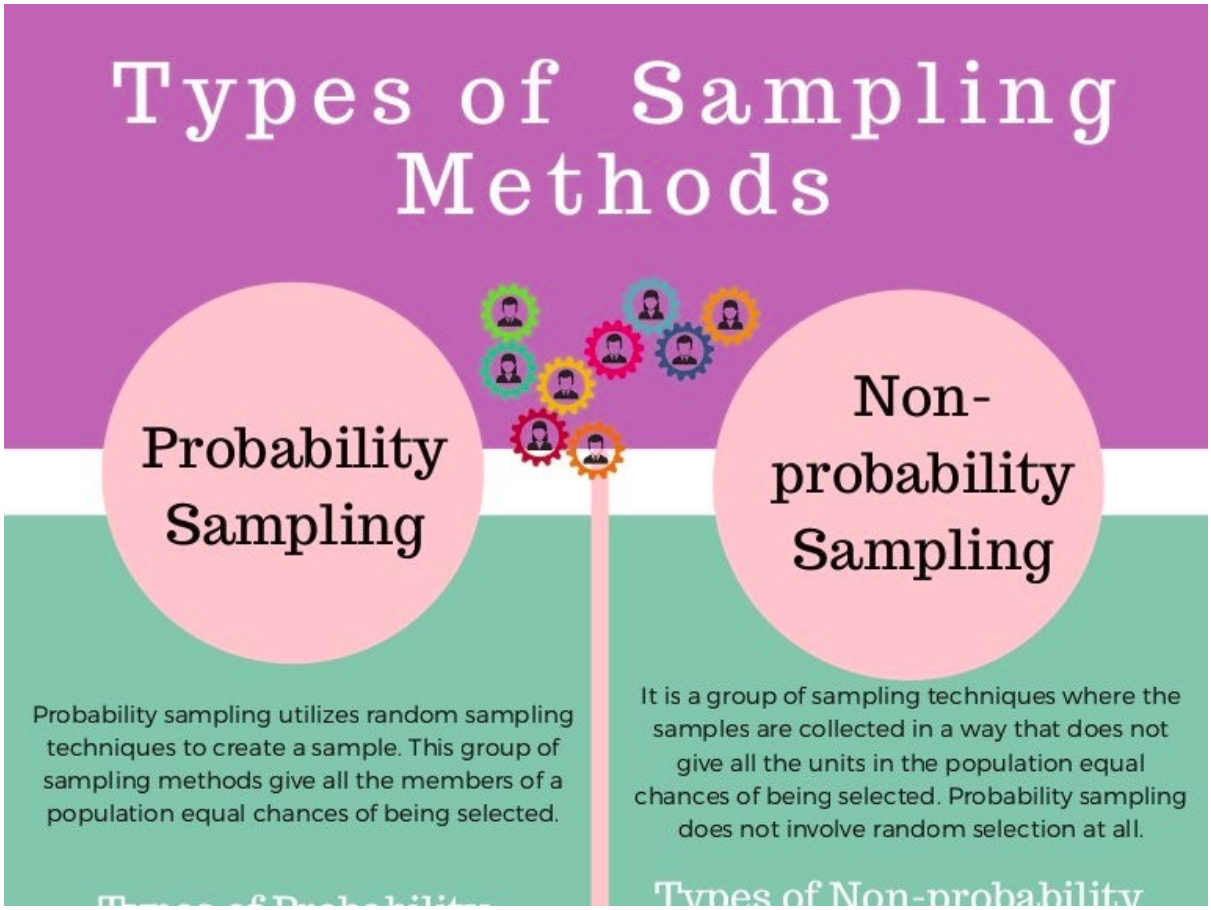
Questionar

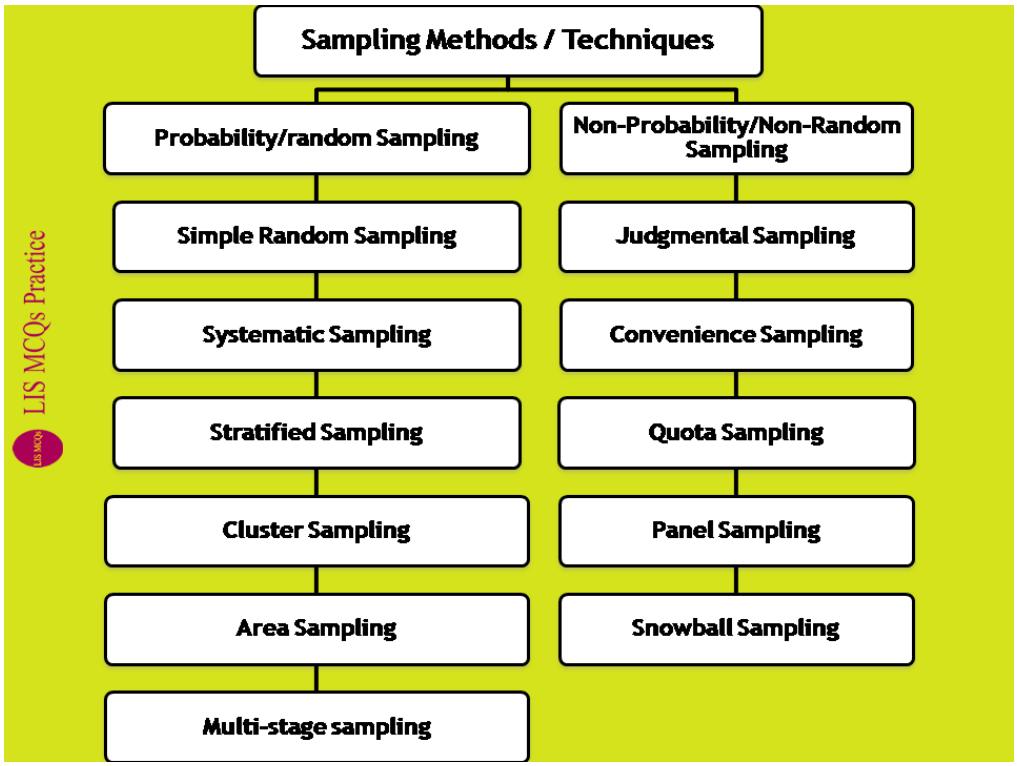
In the northeastern region of the United States (a stratum), suppose that 20 counties (clusters) are sampled at random from a list of 300 counties, and 100 housing units (elements) are sampled from a purchased list of housing units in each sampled county. In the southeastern region of the United States, suppose that 10 counties are sampled at random from a list of 200 counties, and 100 housing units are sampled from a list of housing units in each county. What are the probabilities of selection for housing units in each of the two strata?

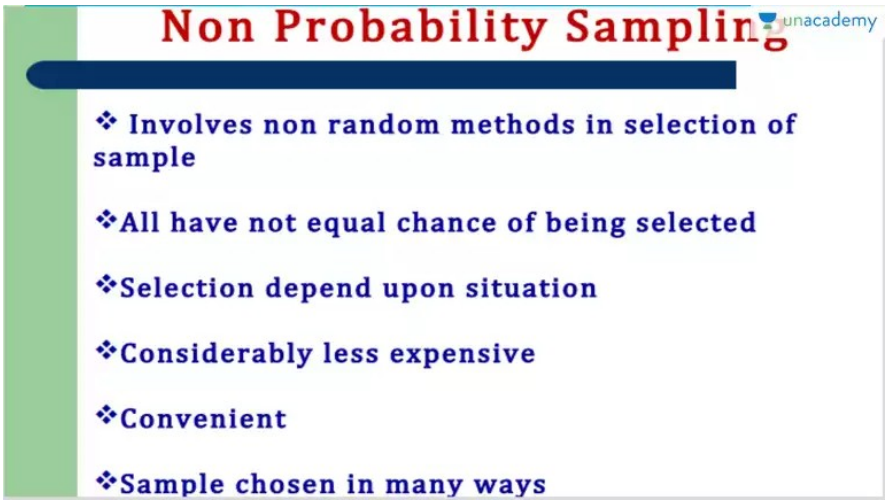
- ○ 20/100 and 10/100
- ○ 20/300 and 10/200
- ○ 100/300 and 100/200
- ◉ We cannot determine the probabilities of selection from the information provided.

✓ **Correto**
To determine the probabilities of selection for housing units based on this complex sample design, we would need to know the total number of housing units on the list in each county, and the county to which a given housing unit belonged.

Types of sampling




Sampling Methods / Techniques
Probability/random Sampling
Non-Probability/Non-Random Sampling
Simple Random Sampling
Judgmental Sampling
Systematic Sampling
Convenience Sampling
Stratified Sampling
Quota Sampling
Cluster Sampling
Panel Sampling
Area Sampling
Snowball Sampling
Multi-stage sampling
LIS MCQs Practice


Non Probability Sampling
unacademy
❖ Involves non random methods in selection of sample
❖All have not equal chance of being selected
❖Selection depend upon situation
❖Considerably less expensive
❖Convenient
❖Sample chosen in many ways

Sr. no	Probability Sampling	Non Probability Sampling
1.	It is used when we have a complete sampling frame.	It is used when an exhaustive population list is not available.(no cmplt sampling frame)
2.	Random selection of samples from population. All persons/ units have an equal chance of being selected.	Not random. Each person/ unit is not given a chance of being selected.
3.	Results can be generalized. Results can be applied to the entire population.	Can be effective when trying to generate ideas. Results may not be generalizable to the population.
4.	Can be expensive and time consuming.	More convenient and less costly

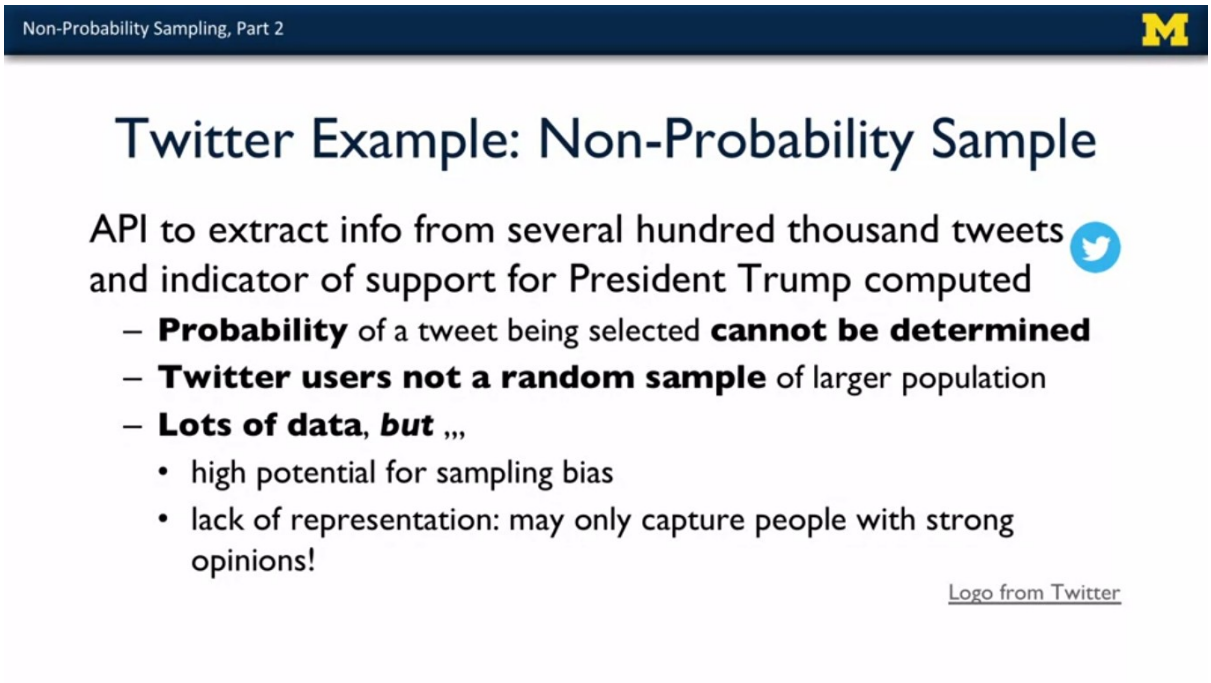
So What Is The Problem?

- Sampled units **not selected at random** → strong risk of **sampling bias**
  (e.g., people actually interested in visiting particular web site)

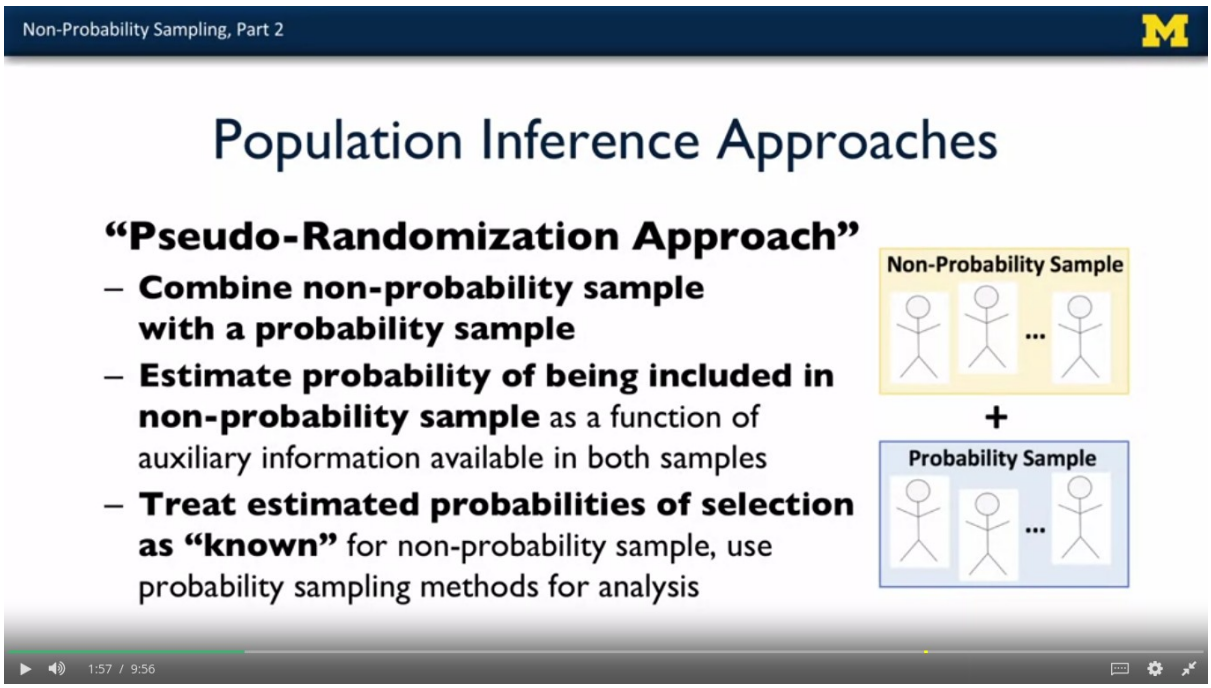
While non-probability sampling methods can generate a lot of data very quickly and at low cost, analysts of the data must be very careful when making population inferences based on the data. The point of this example is not to say that all non-probability samples will lead to erroneous conclusions. When carefully and properly applying one of the two inferential approaches introduced in Lecture #3, one can make sound conclusions about the features of a larger population.

In short, it is always important to first ask what type of sampling mechanism was used to generate the data set that is presently under consideration. Then, if non-probability sampling methods were used, careful analyses of how representative that sample is with respect to

the target population of interest are necessary before proceeding.



Como fazer inferências sobre a população usando non-probability samples?



Population Inference Approaches

"Calibration" Approach

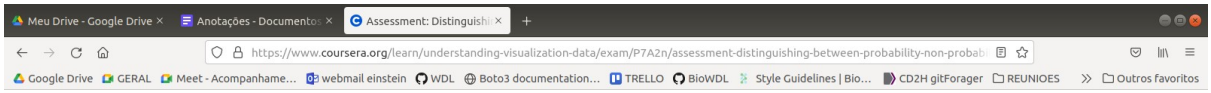
- **Compute weights for responding units** in non-probability sample that allow weighted sampled to mirror a known population

- **Limitation:** if weighting factor not related to variable(s) of interest → will not reduce possible sampling bias

Wang and colleagues analyzed hundreds of thousands of survey responses from Xbox users in 2012, and used a type of calibration weighting approach to make conclusions about voting intentions of the population. Their estimates of preference for presidential candidates in the 2012 election were almost spot-on with forecasts based on aggregation of polling data. For more on this study, please see this link: https://www.sciencedirect.com/science/article/pii/S0169207014000879?via%3Dihub

What's Next?

- **Sampling distributions and sampling variance ~** how to estimate features of these distributions *based on only one probability sample*
- **Examples of making population inferences** *based on type of sample selected*
- Introduce **model-based** approaches to analyzing data



Voltar

Assessment: Distinguishing Between Probability & Non-Probability Samples

Teste valendo nota • 10 min

Vencimento Dec 27, 5:59 AM -02

1. *In each of the questions in this assessment, you'll read a description of a sample, and decide whether or not it is a probably or non-probability sample.*

1 / 1 ponto

A random sample of U.S. households is selected from a population address list, and households in lower-income areas are randomly sampled at a higher rate than households in higher income areas. Both sampling rates are known. The households are then mailed a paper survey asking about employment status.

- (●) Probability
- ( ) Non-Probability

Correto

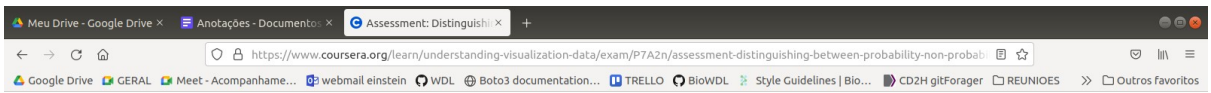
2.

1 / 1 ponto

The telephone surveying technique known as random digit dialing (RDD) is used to select a random sample of households from two different lists: a list of randomly generated landline telephone numbers, and a list of randomly generated mobile phone numbers. Mobile phone numbers are sampled at a higher rate than landline numbers. Both rates are known.

- (●) Probability
- ( ) Non-Probability

Correto



Voltar

Assessment: Distinguishing Between Probability & Non-Probability Samples

Teste valendo nota • 10 min

Vencimento Dec 27, 5:59 AM -02

3.

1 / 1 ponto

A doctoral student in psychology wants to collect opinion information from the general campus population, but doesn't have a large budget for her research. She decides to go out on one of the busiest campus street, wait on the corner, and ask people walking by if they would like to answer a few brief questions. She ultimately speaks with 100 people and analyzes the data.

- ( ) Probability
- (●) Non-Probability

Correto

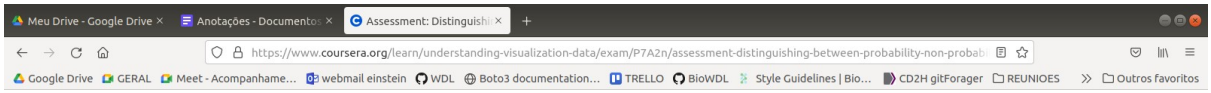
4.

1 / 1 ponto

A University survey research center selects a random sample of counties in the U.S. using probability proportionate to size, with a certain number of counties to be sampled from each of the four major regions of the United States. One hundred housing units are then selected at random within each randomly selected county, from all available housing units within each county, and one adult is selected at random within each household and invited to participate in a survey.

- (●) Probability
- ( ) Non-Probability

Correto



5. **1 / 1 ponto**

After randomly selected adults from the randomly sampled housing units in Question 4 above have completed a survey, they are invited to join a web panel that will receive invitations to complete web surveys throughout the year.

- (●) Probability
- ( ) Non-Probability

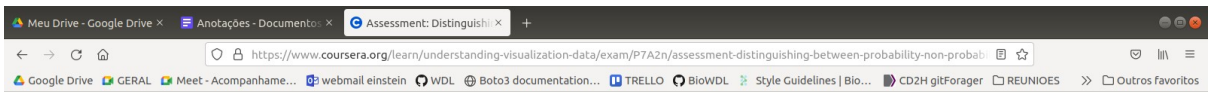
Correto

6. **1 / 1 ponto**

Visitors to a sports information web site click on an advertisement that says they can get paid for providing their opinions about current events.

- ( ) Probability
- (●) Non-Probability

Correto



7. **1 / 1 ponto**

A researcher visits a homeless shelter in a nearby city, tells some individuals currently residing in the center that they can receive compensation for participating in a survey, and indicates that they should also tell everyone in their social networks about this opportunity.

- ( ) Probability
- (●) Non-Probability

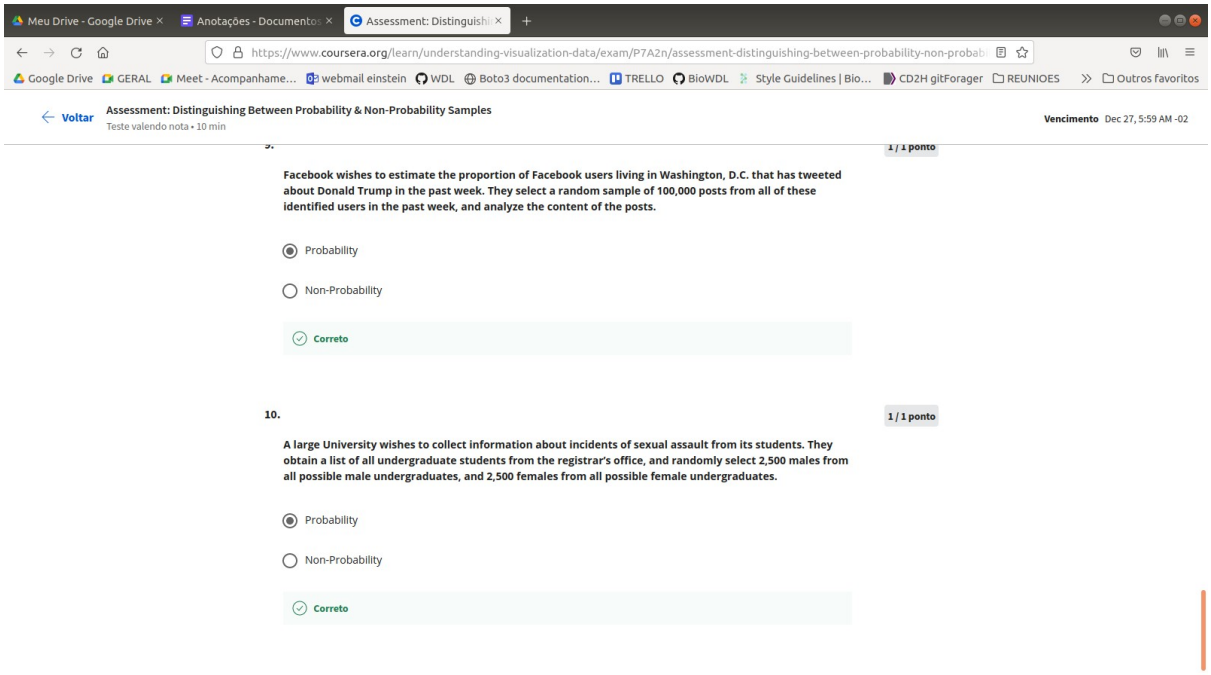
Correto

8. **1 / 1 ponto**

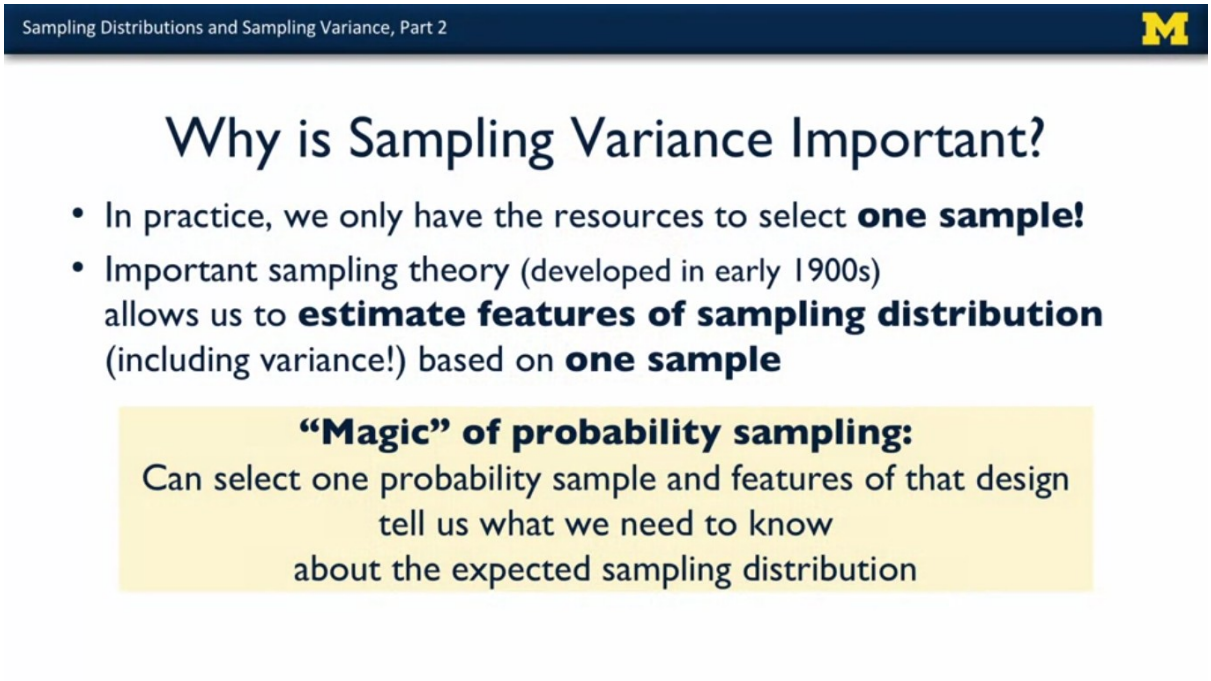
A psychology professor is told by a statistical consultant that she needs 100 males and 100 females to have enough statistical power to compare the two groups in terms of mean scores on a new scale that she is developing. She actively recruits student volunteers for the study, and turns away all interested volunteers after she hits her target of 100 people in each group.

- ( ) Probability
- (●) Non-Probability

Correto



Making Population Inference Based on Only One Sample



Questionar

You speculate that the mean test performance in an undergraduate psychology class is 80 / 100. Based on a single sample of 30 students, you estimate the mean test score to be 90. The estimated sampling variance based on that one sample suggests that most estimates from repeated hypothetical samples of size 30 will lie between 85 and 95. How confident are you about your speculation?

- (●) We have fairly strong evidence against our speculation: a mean of 80 seems unrealistic.
- ( ) We have evidence in support of our speculation: 80 seems like a plausible mean value.
- ( ) We need to draw several more samples of size 30 before we can make a conclusion.

✓ **Correto**
The estimating sampling variance suggests that most estimates of the mean test performance will lie between 85 and 95, which makes 80 seem like an unrealistic value for the mean. We do not need to draw several samples of size 30, because sampling theory allows us to estimate the variance of sampling distribution based on one sample only!

General approaches to making population inferences based on estimated features of sampling distributions

- Confidence Interval Estimate for Parameters of Interest
- Hypothesis Testing about Parameters of Interest

Examples of Parameters of Interest:
a mean, a proportion, a regression coefficient, an odds ratio, and many more!

Key Assumption: Normality

These approaches assume that sampling distributions for the estimate are (approximately) normal, which is often met if sample sizes are "large"

All possible values of estimate

Q: What if sampling distribution is not (approximately) normal?

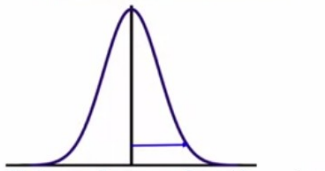
A: Alternative inferential approaches discussed in later course

Step 2: Estimate the Sampling Variance of the Point Estimate

Compute an **unbiased estimate of the variance** of the sampling distribution for the particular point estimate

Unbiased Variance Estimate:
Correctly describes variance of the sampling distribution *under the sample design used*

Square root of variance = **Standard Error of the Point Estimate**



Making Population Inference Based on Only One Sample

To Test Hypotheses

hypothesized or 'null' value

- Hypothesis: Could the value of the parameter be _____?
- Is point estimate for parameter close to this null value or far away?
- Use standard error of point estimate as yardstick

$$Test\ Statistic = \frac{(estimate - null\ value)}{standard\ error}$$

- If the null is true, what is the probability of seeing a test statistic this extreme (or more extreme)? If probability small, reject the null!

https://seeing-theory.brown.edu/basic-probability/index.html

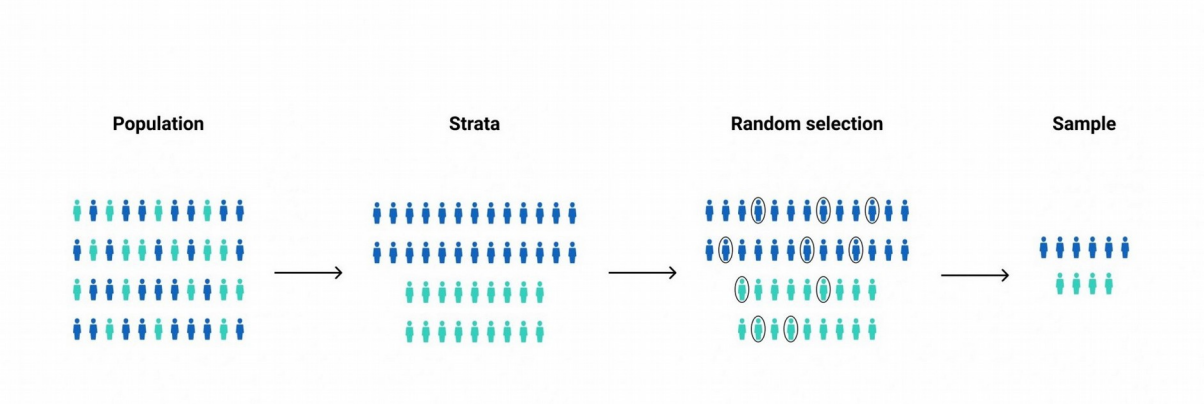
"standard" statistical analyses that are presented and discussed in introductory statistics

courses make the assumption that the data of interest are **independent and identically distributed (or "i.i.d.") observations**. As discussed in the lectures earlier this week, **simple random sampling (SRS)** is the closest probability sampling analog to i.i. because sampling mechanism used to generate the observations will produce independent and identically distributed observations. While this type of sampling will produce samples with this nice "i.i.d." statistical property, facilitating "standard" statistical analyses, SRS is seldom used when sampling from real populations.

One of the reasons for this is that SRS, while producing estimates that are unbiased in nature (which recall means that the estimates based on hypothetical repeated samples using SRS will have a mean equal to the true population mean), has the potential to generate "bad" samples with substantial sampling error (where an estimate based on the sample is quite different from the population parameter of interest).

Consider, for example, a national sample of 1,000 cell phone numbers selected using SRS. While in expectation any one given sample will include a representative random sampling of numbers from area codes across the nation, **all possible random samples using SRS are equally likely.** What this means is that a simple random sample of cell phone numbers that only includes area codes from Florida is just as likely as a simple random sample of numbers that includes a representative selection across the states. Ideally, we would like to use design strategies to reduce the chances of such a "bad sample" occurring, especially if our variable of interest tends to take on very different values in the state of Florida!
The major statistical problem with the simple random "Florida" sample is that any estimate that we compute after collecting data from the sample will likely be very different from the true population parameter that we are trying to estimate (especially if the variable of interest tends to take on very different values in Florida relative to the rest of the nation).
Because the probability of selecting these extreme samples is equal to the probability of selecting more representative samples, the sampling distribution for simple random samples can tend to be quite variable.

A very common sampling technique used to minimize the sampling variance that can arise from these so-called "bad samples" in SRS is **stratification**. You've already been introduced to stratification in an earlier lecture. When we conduct stratified sampling, we first allocate portions of our sample to all possible divisions (or "strata") of the population of interest (e.g., states). This ensures that some sample will be selected from all of these possible divisions, and that the overall sample will therefore be representative of the target population. For example, using a technique known as *proportionate allocation*, suppose that we knew that 55% of students enrolled in a particular college were females, and 45% were males. If we wanted to draw a sample of 1,000 students from this college, we would randomly selected 550 females from a list of all females enrolled, and 450 males from a list of all males enrolled. This ensures that our entire sample of size 1,000 won't include only females!



If we only draw our sample from one stratum of the overall population (e.g., gym goers), and the units in that stratum tend to have values on a variable of interest that *differ* from the values for the variable in other strata, then the estimates that we compute based on that sample will be biased, and will not represent the overall target population. This is an example of **selection bias;** on average, estimates computed from repeated samples of gym goers will *not* be equal to the true population parameter of interest. Stratified sampling ensures that we would select a sample of gym goers *and* a sample of non-gym goers, increasing the representativeness of our sample and potentially reducing bias. Another nice property of **stratified sampling is that it shrinks the variance of sampling distributions**.

In SRS, all of the variance **within strata and between strata** in terms of the variable of interest contributes to the overall sampling variance. In stratified sampling, when we allocate a certain number of sampled units to be selected from each stratum, we ***remove the between-stratum variance from the overall sampling variance!*** This is because every hypothetical repeated sample would use the same stratified design, and the same allocation; assuming reasonable response rates, we will have representation from each of the strata where we allocated a portion of the sample. There is no uncertainty in whether we will have sampled units from a particular stratum, and there is nothing random about the allocations; these are **fixed by design**! The only **uncertainty arises from the random sampling that occurs *within* strata** from one hypothetical sample to another. Each sample will always feature random selections from the same strata; what happens within the strata will change from sample to sample.

We will revisit the idea of stratified sampling in an upcoming lecture, but you will often hear sampling statisticians say **"when in doubt, stratify."** We can use this technique to prevent bad samples, and decrease the variance of our sampling distributions.

Nonresponse bias occurs when there is a tendency for the units in a sample that agree to provide data to be systematically different from the units in the sample that do not provide data (in terms of the variable of interest). This type of bias can also occur for estimates based on specific variables, when sampled units may agree to provide data in general, but not on specific variables. For instance, a survey respondent may agree to participate in the survey, but refuse to share their income. This type of nonresponse is known as **item nonresponse**.

Suppose that people with lower income tend to respond to a survey of a nationally representative sample of individuals at higher rates than people with higher income.

Because the resulting sample of *respondents* to the survey request tends to feature people with lower income, any estimates related to income (which will always be computed using data from the respondents, or the units that agree to provide data!) will be subject to another form of **selection bias**, namely **nonresponse bias**.

Whereas stratified sampling is a design tool that can be used to reduce selection bias from a sampling perspective, the selection bias introduced by unit or item nonresponse can either be addressed during the data collection process or via post-survey adjustments to the estimates based on a respondent sample.

Inference from non-probability samples

- **Problem: Non-probability samples do not let us rely on sampling theory** for making population inferences based on expected sampling distributions

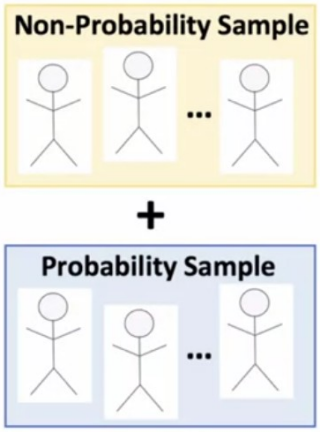
Two Approaches:

I. Quasi-Randomization (or pseudo-randomization)
II. Population Modelling

For any of these estimation techniques for non-probability samples, **we need to have common variables in the two data sets**. Simple estimation of the mean based on the non-probability sample may lead to a biased estimate, and we can't estimate sampling variance from the non-probability sample.

Approach 1: "Quasi-Randomization"

Big Idea: Combine data from non-probability sample with data from probability sample that collected same types of measures



Example:  __ years-old? White/Black/Asian/...

if we measure blood pressure, age, and race/ethnicity on a sample of **volunteers**,

→ combine with prior data from a probability sample (e.g., NHANES) that collected the same three measures

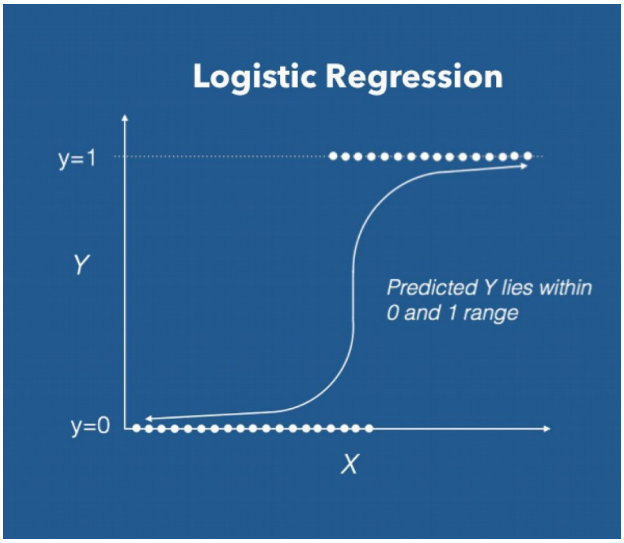
Exemplo: casos em que você depende de voluntários

- **Stack** the two data sets; non-probability sample may have other response variables we are really interested in
- **Code** NPSAMPLE = 1 if member of non-probability sample
  NPSAMPLE = 0 if member of probability sample

NPSAMPLE	BloodPressure	Age	Race/Ethnicity	Response1	Response2
0	100	52	White	83	Yes
0	120	45	Asian	92	No
⋮	⋮	⋮	⋮	⋮	⋮
1	130	64	Black	91	No
1	110	38	White	79	No
⋮	⋮	⋮	⋮	⋮	⋮

Fit **logistic regression model**

→ predicting NPSAMPLE with common variables
weighting non-probability cases by 1 and
weighting probability cases by their survey weights



Big Idea:

1. **Can predict probability of being in non-probability sample**, within whatever population is represented by probability sample!
2. **Invert predicted probabilities** for non-probability sample, **treat as survey weights** in standard weighted survey analysis

$$Survey\ Weight = \frac{1}{Predicted\ Probability}$$

Issue: How to estimate sampling variance?

Not entirely clear …

Some kind of **replication method** is recommended
(e.g. computing weighted estimates based on **bootstrap samples** or **jackknife samples** of the original units)

Para usar esse método, é preciso haver variáveis em comum entre os dois datasets

For a deep (and technical) dive into this approach, see the following article:

Elliott, M.R. and Valliant, R. (2017).
Inference for Non-Probability Samples.
Statistical Science, 32(2), 249-264.

Approach 2: Population Modeling

Big Idea:

1. **Use predictive modeling** to **predict aggregate sample quantities (usually totals) on key variables** of interest for population units **not** included in the non-probability sample
2. **Compute estimates of interest using estimated totals**

e.g $Weighted\ Mean = \frac{Predicted\ Total\ Estimate}{Estimated\ Population\ Size}$

Note: Don't need probability sample with same measures

Idea: dados de uma população maior, mas não estão todas as variáveis preenchidas para todos os indivíduos.

- **Need good regression models** to predict key variables using other auxiliary information available at aggregate level (e.g., totals for overall population)

...more about good models later!

- **Standard errors** can be based on fitted regression models, or using similar replication methods!

Inferential methods for non-probability samples need to:

- **Leverage other auxiliary information** (reference probability samples or regression models)
- **Predict values** for population cases not included in probability sample (or at least probability of being included in non-probability sample!)

In absence of this information …
we will have a **hard time** making good population inferences!

Complex samples

Complex Sample = any probability sample where design involves more than Simple Random Sampling (SRS)!

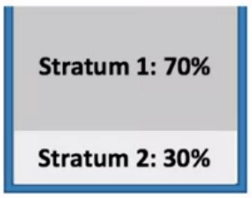
Features of Complex Samples: Stratification

- **Stratification**: Allocation of overall sample to different "strata", or mutually exclusive divisions of the population (e.g., regions of the United States)

- Several different allocation schemes are possible; Aim → minimize sampling variance for particular variables given fixed costs

Example: *Proportionate* Allocation

- If 70% of a population appears in one stratum and 30% in the other;
- Then 70% of the overall sample would be allocated to the first stratum, and 30% to the second



Population

- Stratification will eliminate between-stratum variance in means (or totals) on variable from the sampling variance!
- Important to account for stratification in analysis; else sampling variance may be artificially large → inferences too conservative, confidence intervals too wide!

Features of Complex Samples: Clustering

- **Clustering**: Random sampling of larger clusters of population elements, possibly across multiple stages (e.g., counties, then segments, then households)

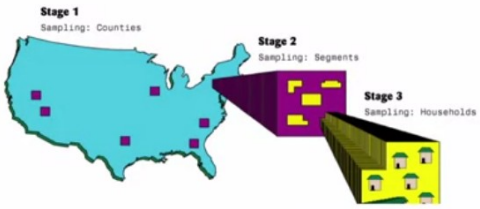


Image Credit: L. Mahadjer, Westat

- Reduces cost of data collection: expensive $$$ to visit *n* randomly sampled units from large and widespread population

- Clustering *reduces* costs 😀
  BUT tends to *increase* sampling variance of estimates 😔
  Why? Units within same cluster have similar (correlated) Values on variables of interest → don't measure unique info!

- **Important** to account for cluster sampling in analysis, else inferences too *liberal*, confidence intervals too *narrow*!

Features of Complex Samples: Weighting

Complex samples are still probability samples, but if …

- Multiple stages of cluster sampling within strata
- Or certain subgroups sampled at higher rates (oversampling)

→ **Unequal probabilities of selection** for different units

Need to account for these unequal probabilities to make **unbiased** population inferences

- **How**? Use of **weights** in analysis … (partly) defined by **inverse of probability of selection**

If my probability is 1/100 → my weight is 100, I represent ***myself*** and **99 others** in the population!

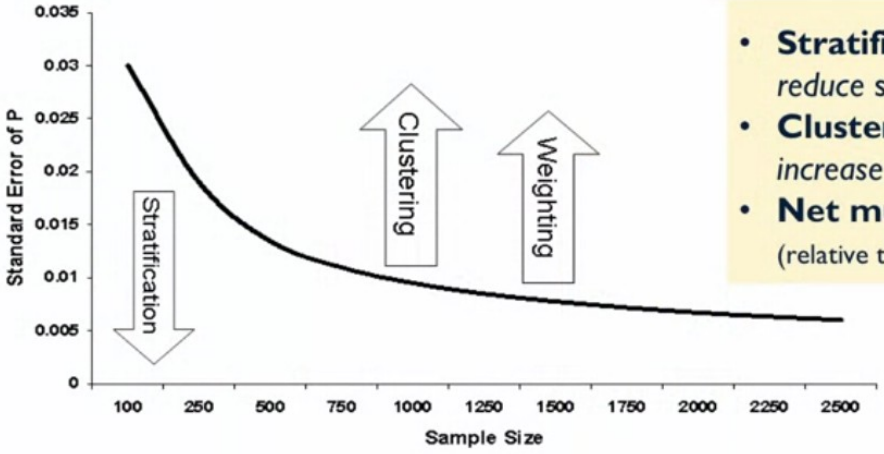
- Weights also **adjusted** for different probabilities of responding in different **subgroups**

If my probability of selection = 1/100
and I belong to subgroup where only 50% responded
→ my adjusted weight = (1/0.01) x (1/0.5) = 200

- **Important** need to use weights so estimates are unbiased with respect to the sample design; else possible serious bias!

- **Drawback**: like cluster sampling, highly variable adjusted survey weights tend to increase sampling variance of weighted estimates *(even if they produce unbiased estimates!)*

Visualizing Design Effects



- **Stratification:** *reduce* sampling variance
- **Cluster** and **Weighting:** *increase* sampling variance
- **Net multiplicative change** (relative to SRS) = **design effect**

urce: *Applied Survey Data Analysis (Heeringa et al., 2017)*

- Most "survey analysis" procedures in statistical software compute unbiased point estimates (using final survey weights) and unbiased estimates of sampling variance (using stratum and cluster information, or *replicate sampling weights*)

- **Important** need to use appropriate software procedures, and identify all of these features to the software!