

# **Inferência/ modelação estatística**

Métodos e sua implementação em R/ Python

# Inferência/ modelação estatística

- Objetivos:
  - Descrever a distribuição das variáveis
  - Descrever relação entre as variáveis
  - Realizar inferências sobre a distribuição das variáveis e sobre a sua relação -> a partir de uma amostra inferir sobre uma população

Em R, muitas das funções e distribuições incluídas na distribuição base

Packages do Python:

- Package ***scipy.stats*** - <https://docs.scipy.org/doc/scipy/reference/stats.html>
- Package ***statsmodels*** - <https://www.statsmodels.org/stable/index.html>

# Distribuições de probabilidade: variáveis discretas

- **Função (massa) de probabilidade**: para uma variável discreta  $X$

$$f(x) = P(X=x)$$

- dá a probabilidade associada ao valor  $x$
- soma dos valores de  $f$  terá que ser 1

- **Função de probabilidade acumulada** da variável aleatória  $X$

$$F(x) = P(X \leq x)$$

- dá a probabilidade que  $X$  tome um valor menor ou igual a  $x$
- se  $X$  é uma variável discreta,  $F$  é dada pelo somatório de  $f$ , para valores menores ou iguais a  $x$

# Distribuições de probabilidade: variáveis contínuas

- **Função de densidade de probabilidade (f.d.p.):  $f(x)$**  pode ser fdp para uma variável contínua sse

$$P(a \leq X \leq b) = \int_a^b f(x) dx$$

- $f(x)$  não é  $P(X=x)$  como no caso discreto, pois este é 0, sendo as probabilidades neste caso sempre medidas em intervalos
- integral de  $f(x)$  sobre todo o intervalo de valores possíveis = 1

- **Função de probabilidade acumulada** da variável aleatória  $X$

$$F(x) = P(X \leq x)$$

- é dada pelo integral de  $f$  no intervalo de  $-\infty$  a  $x$

$$P(a \leq X \leq b) = F(b) - F(a)$$

# Distribuições de probabilidade: funções em R

- O R tem um conjunto de 4 funções para cada distribuição para as seguintes tarefas:
  - Funções iniciadas por **d**: dão a função de densidade de probabilidade
  - Funções iniciadas por **p**: dão a função acumulativa de probabilidade
  - Funções iniciadas por **r**: geram n<sup>º</sup>s aleatórios com base na distribuição
  - Funções iniciadas por **q**: dão os quantis da distribuição
- Estas funções estão disponíveis para as distribuições mais usadas

# Distribuições de probabilidade: funções em python

- O package **scipy.stats** inclui inúmeras distribuições quer contínuas quer discretas que podem ser usadas:
  - Ver lista em: <https://docs.scipy.org/doc/scipy/reference/stats.html>
- Cada distribuição tem associadas funções para diversas tarefas:
  - Funções **pdf**: dão a função de densidade de probabilidade
  - Funções **cdf**: dão a função acumulativa de probabilidade
  - Funções **rvs**: geram n<sup>º</sup>s aleatórios com base na distribuição
  - Funções **ppf**: dão os percentis da distribuição

# Distribuições de probabilidade: distribuição uniforme

Variáveis discretas: todos os valores têm a mesma probabilidade de ocorrência

$$f(x) = \frac{1}{k} \quad \begin{array}{l} x = x_1, x_2, \dots, x_k \\ x_i \neq x_j, i \neq j \end{array} \quad f(x) = \begin{cases} \frac{1}{\beta - \alpha} & \alpha < x < \beta \\ 0 & \text{outros valores} \end{cases}$$
$$\mu = \frac{1}{k} \sum_{i=1}^k x_i \quad \mu = \frac{\beta + \alpha}{2}$$

Variáveis contínuas: valores podem ocorrer de forma uniforme num intervalo  $]\alpha, \beta[$

– Para  $[0,1]$ , média é 0.5 e variância  $1/12$

Funções em R:

- `dunif`
- `punif`
- `runif`
- `qunif`

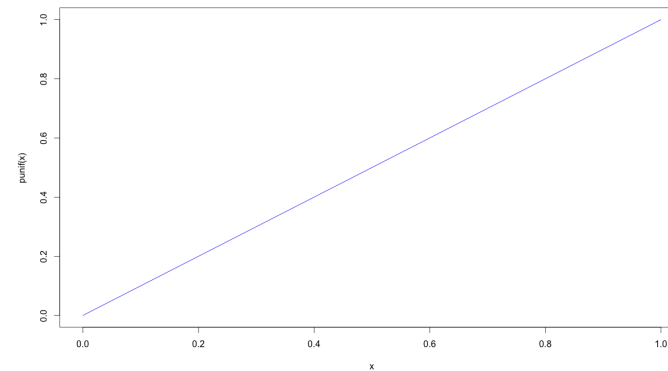
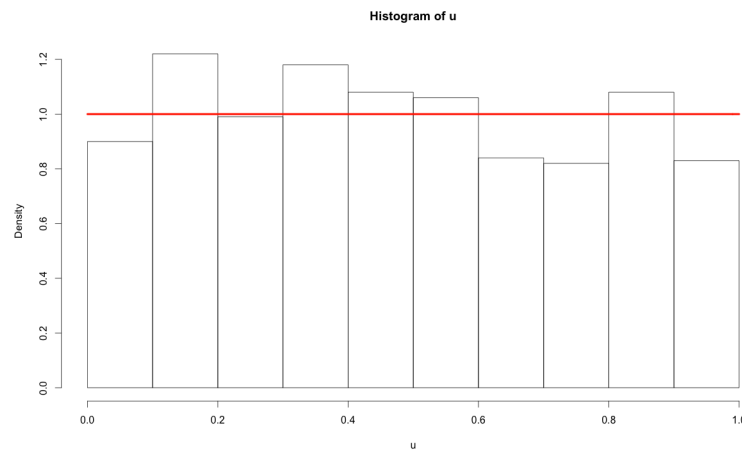
Funções em python:

- `uniform.pdf`
- `uniform.cdf`
- `uniform.rvs`
- `uniform.ppf`

# Distribuição uniforme: exemplos

```
> u = runif(1000)
> hist(u, prob=T)
> curve(dunif(x,0,1),add=T, col="red")
> curve(punif(x), col="blue")
> summary(u)
```

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
	0.001214	0.236300	0.456100	0.482600	0.725700	0.999300





# Distribuição uniforme: exemplos

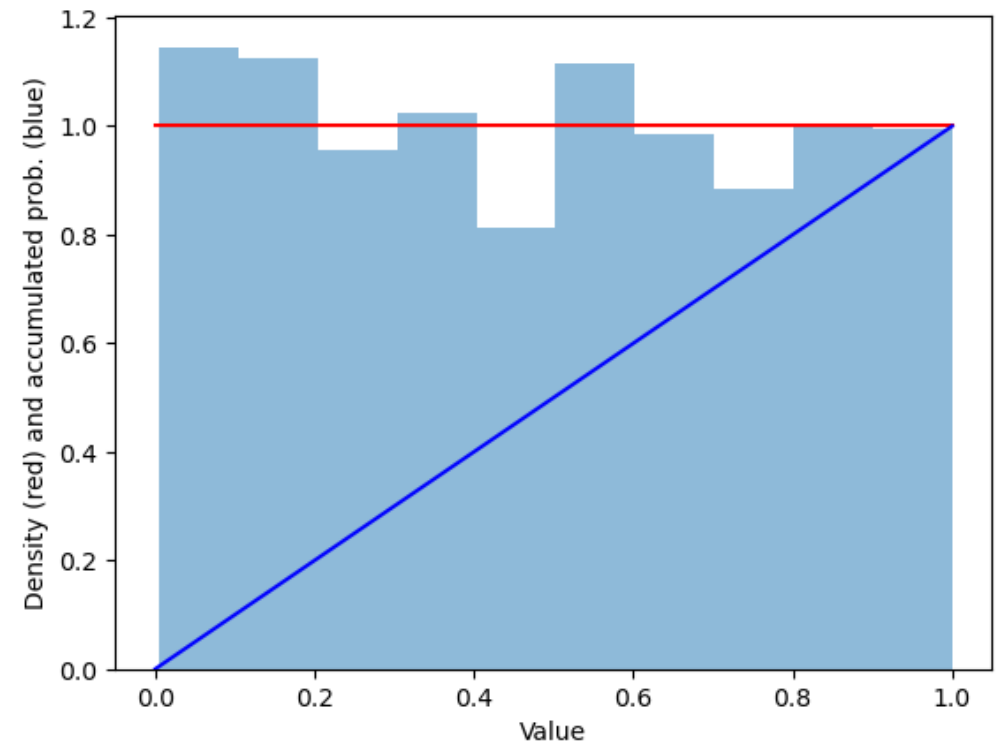
```
np.random.seed(42)
data = np.random.uniform(size=1000)

plt.hist(data, density=True, alpha=0.5)
x = np.linspace(0, 1, 100)

density = stats.uniform.pdf(x)
plt.plot(x, density, color='red')

cdf = stats.uniform.cdf(x)
plt.plot(x, cdf, color='blue')

plt.xlabel("value")
plt.ylabel("Density (red) ...")
plt.show()
```



# Distribuições de probabilidade: distribuição normal

- Variável contínua
- Distribuição mais usada na Estatística e análise de dados
- Parâmetros: média -  $\mu$ ; desvio padrão-  $\sigma$

$$f(x; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \quad \begin{array}{l} -\infty < x < \infty \\ \sigma > 0 \end{array}$$

Funções em R:

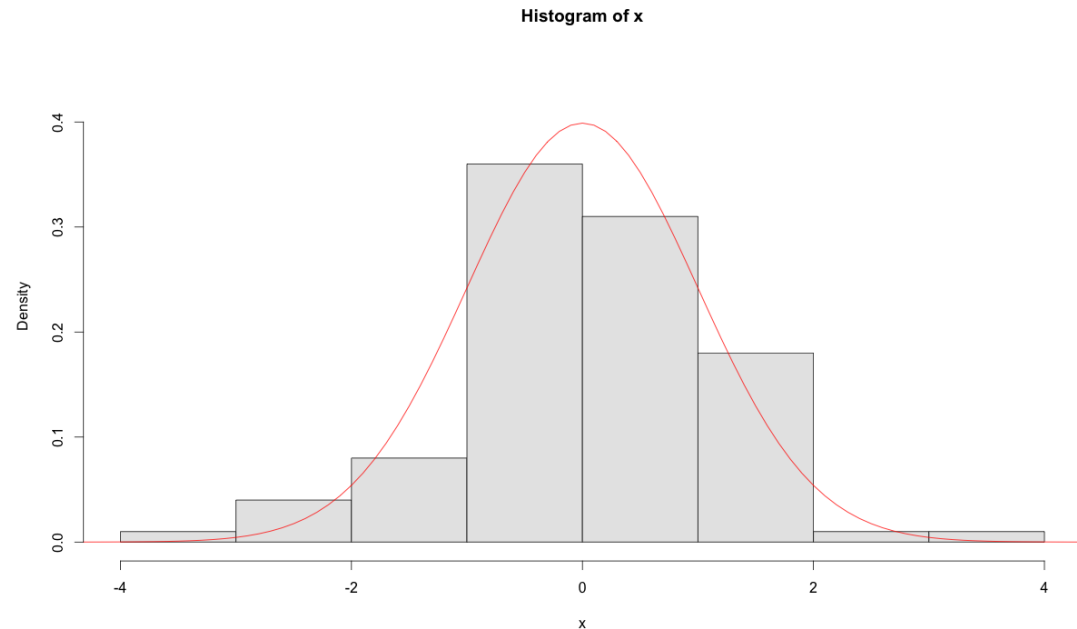
- *dnorm*
- *pnorm*
- *rnorm*
- *qnorm*

Funções em python:

- *norm.pdf*
- *norm.cdf*
- *norm.rvs*
- *norm.ppf*

# Distribuições de probabilidade: distribuição normal

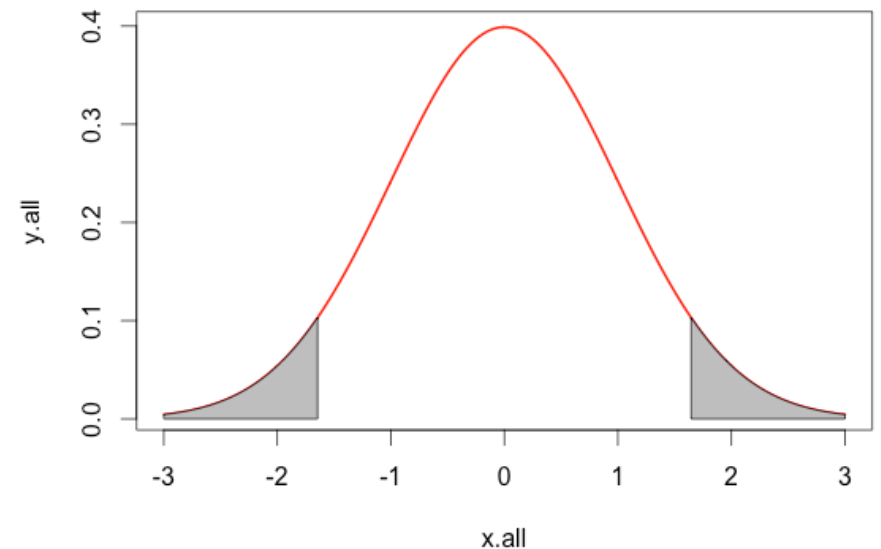
```
> x = rnorm(100)
> mean(x)
[1] 0.00897715
> sd(x)
[1] 1.081045
> y = rnorm(100,
mean=10, sd = 3)
> mean(y)
[1] 9.70692
> sd(y)
[1] 2.927611
> pnorm(1)
[1] 0.8413447
> qnorm(0.975)
[1] 1.959964
```



```
> hist(x,probability=T,col=gray(.9), ylim=c(0,0.45))
> curve(dnorm(x),-5,5, add=T, col="red")
```

# Distribuições de probabilidade: distribuição normal

```
> x.all = seq(-3, 3, by = 0.01)
> y.all = dnorm(x.all)
> plot(x.all, y.all, lwd = 2, type = "l", col="red")
> x.p = seq(-3, qnorm(.05), length = 100)
> y.p = dnorm(x.p)
> polygon(c(-3, x.p, qnorm(.05)), c(0, y.p, 0), col = "gray")
> x.p.up = seq(qnorm(.95), 3, length = 100)
> y.p.up = dnorm(x.p.up)
> polygon(c(qnorm(.95), x.p.up, 3), c(0, y.p.up, 0), col = "gray")
```



# Distribuições de probabilidade: distribuição normal

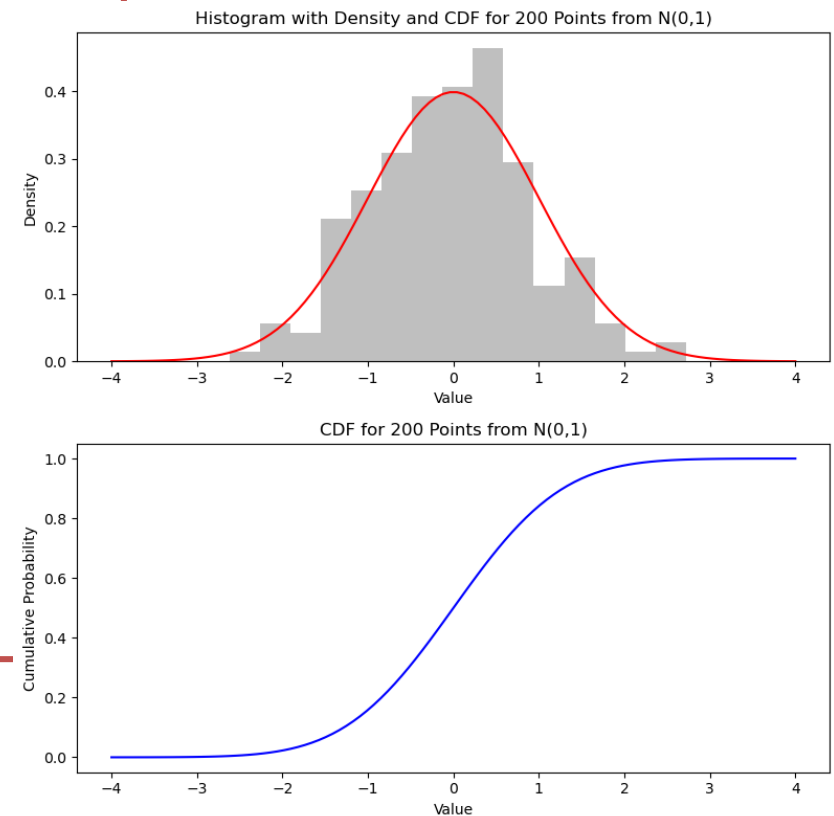
```
data = np.random.normal(loc=0, scale=1, size=200)

x = np.linspace(-4, 4, 100)
pdf = stats.norm.pdf(x)
cdf = stats.norm.cdf(x)

fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(8, 8))
ax1.hist(data, bins=15, density=True, alpha=0.5, color='grey')
ax1.plot(x, pdf, color='red')
ax1.set_xlabel("value")
ax1.set_ylabel("Density")
ax1.set_title("Histogram with Density and CDF ...")

ax2.plot(x, cdf, color='blue')
ax2.set_xlabel("value")
ax2.set_ylabel("Cumulative Probability")
ax2.set_title("CDF for 200 Points from N(0,1)")

fig.tight_layout()
plt.show()
```



# Distribuições de probabilidade: distribuição t de student

- Variável contínua
- Distribuição semelhante à normal mas com “caudas” mais longas, i.e. com mais valores extremos
- Parâmetro: nº de graus de liberdade -  $\nu$
- Quanto maior  $\nu$  mais próximo da distribuição normal

$$f(t) = \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\sqrt{\pi\nu}\Gamma\left(\frac{\nu}{2}\right)} \left(1 + \frac{t^2}{\nu}\right)^{-\frac{\nu+1}{2}} \quad -\infty < t < \infty$$

Funções em R:

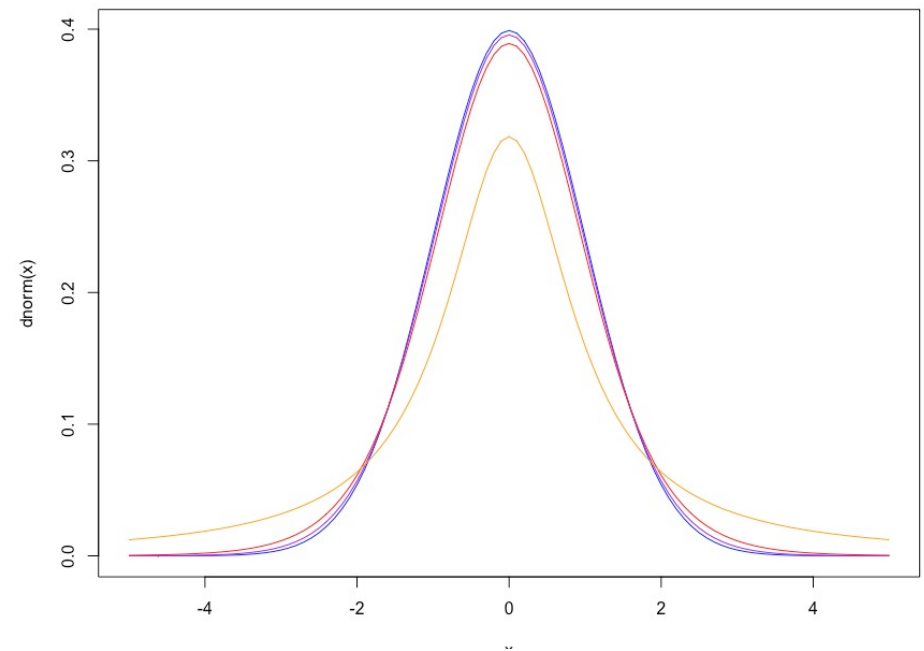
- *dt*
- *pt*
- *rt*
- *qt*

Funções em python:

- *t.pdf*
- *t.cdf*
- *t.rvs*
- *t.ppf*

# Distribuição t de student

```
> curve(dnorm, -5, 5, col="blue")  
> curve(dt(x, df=30), -5, 5, col="purple", add=T)  
> curve(dt(x, df=10), -5, 5, col="red", add=T)  
> curve(dt(x, df=1), -5, 5, col="orange", add=T)
```



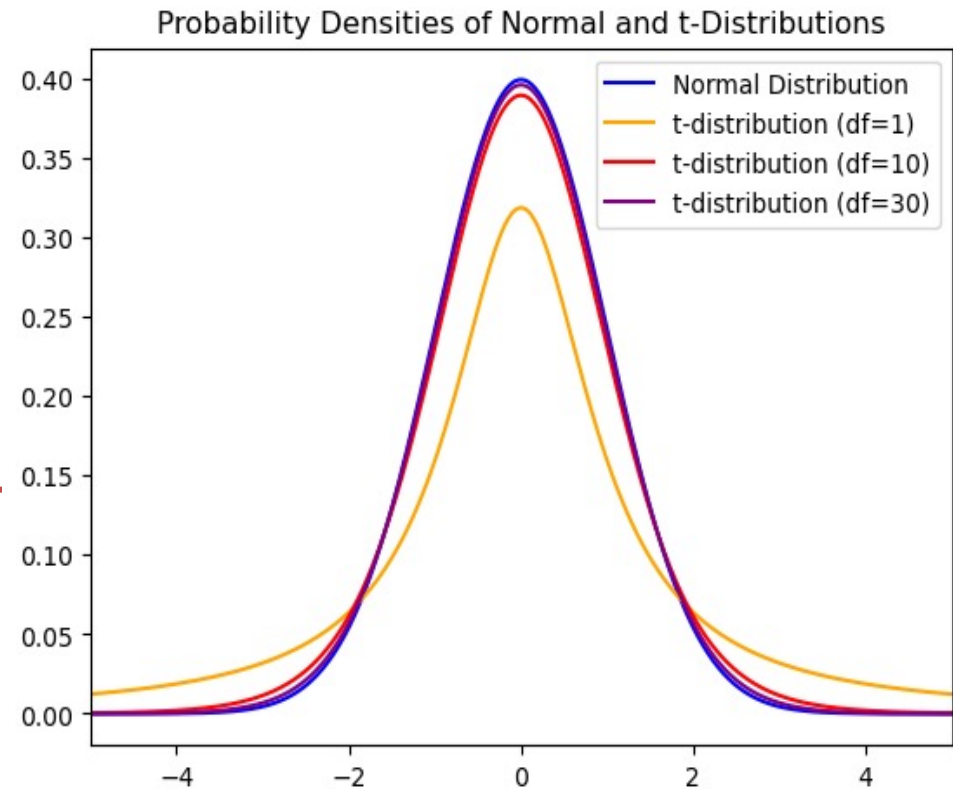
# Distribuição t de student

```
x = np.linspace(-5, 5, 500)

normal = stats.norm.pdf(x)
t1 = stats.t.pdf(x, df=1)
t10 = stats.t.pdf(x, df=10)
t30 = stats.t.pdf(x, df=30)

fig, ax = plt.subplots()
ax.plot(x, normal, color='blue', label='Normal ...')
ax.plot(x, t1, color='orange', label='t... (df=1)')
ax.plot(x, t10, color='red', label='t-... (df=10)')
ax.plot(x, t30, color='purple', label='t-... (df=30)')

ax.set_xlim(-5, 5)
ax.legend()
ax.set_title('Probability Densities ...')
plt.show()
```





# Teorema do limite central

É um resultado muito importante para a inferência estatística

A sua principal consequência afirma que dado um conjunto de valores com média e variância conhecidas, à medida que o tamanho da amostra aumenta, a **distribuição amostral da sua média aproxima-se de uma distribuição normal**

- cuja média é igual à média da distribuição original
- cujo desvio padrão (*standard error*) é o desvio padrão da distribuição original dividido pela raiz quadrada do tamanho das amostras

# Simulação TLC: exemplo

Objetivo: mostrar que a distribuição da média de amostras de tamanho  $n$  provindo de uma distribuição  $N(0,1)$  tem média 0 e desvio padrão  $1 / \sqrt{n}$

```
> numsims = 1000
> tam = 10
> values = rnorm(numsims * tam)
> m = matrix(values, numsims, tam)

> mean(m)
[1] 0.00712001
> sd(m)
[1] 0.9897006
```

```
> medias = apply(m, 1, mean)
> mean(medias)
[1] 0.00712001

> sd(medias)
[1] 0.3166327
> 1 / sqrt(tam)
[1] 0.3162278
```

## Simulação TLC: exemplo

Objetivo: mostrar que a distribuição da média de amostras de tamanho  $n$  provindo de uma distribuição uniforme  $[0,1]$  tem media igual à original (0.5) e desvio padrão  $1 / \sqrt{12*n}$

```
> numsims = 1000
> tam = 10
> values_u = np.random.uniform(size=numsims * tam)
> m_u = np.reshape(values_u, (numsims, tam))

> print(np.mean(m_u))
> print(np.std(m_u))
> print(1 / np.sqrt(12))
```

```
0.5080396
0.2896496
0.2886751
```

```
> medias_u = np.mean(m_u, axis=1)

> print(np.mean(medias_u))
> print(np.std(medias_u))
> print(1 / np.sqrt(12*tam))
```

```
0.5080396
0.0945325
0.0912871
```

# Amostragem

Em muitos casos de Análise de Dados, o objetivo passa por ser capaz de extrair padrões e relações de um conjunto de dados disponível (designado por amostra) que provém de um conjunto maior (a população)

Em muitos casos de simulação estatística, é necessário realizar a amostragem de valores a partir de conjuntos de dados

Existem duas formas distintas de realizar a amostragem: com reposição (i.e. o valor selecionado é recolocado nos dados e pode ser escolhido de novo) ou sem reposição

Função mais simples do R: *sample*

- Argumentos obrigatórios: *x* – dados de onde escolher; *size* – nº de itens a escolher;
- Argumentos opcionais: *replace* – com ou sem reposição (T ou F); por omissão F; *prob* – vetor com probabilidades para cada item

# Amostragem: exemplos

```
> sample(1:6, 5, replace = T)
[1] 3 6 5 3 3
> sample(c("cara","coroa"),4,replace=TRUE)
[1] "cara" "coroa" "cara" "cara"
> sample(1:49, 6)
[1] 16 2 43 27 13 33
> cartas =
paste(rep(c("A",2:10,"J","Q","K"),4),
c("C","P","E","O"), sep="")
> sample(cartas, 3)
[1] "2C" "QP" "4E"
> sample(1:10,10)
[1] 6 8 2 10 9 3 4 7 1 5
```

← Simular lançamento de um dado 5 vezes

← Simular lançamento de uma moeda 4 vezes

← Chave do totoloto

← Selecionar cartas de um baralho

← Permutação

# Estimação de intervalos de confiança

Uma das tarefas essenciais na análise de dados passa pela necessidade de estimar parâmetros da distribuição a partir dos dados (e.g. estimar a média de uma distribuição, assumindo que é normal)

A partir dos dados conhecidos (amostra) é normalmente impossível saber com certeza os valores dos parâmetros, mas podemos chegar a estimativas dos seus valores, bem como quantificar a incerteza

Os **intervalos de confiança** constituem esta estimativa, constituindo uma gama de valores que contém o parâmetro procurado com um dado **nível de confiança  $\alpha$**

IC para um parâmetro  $\theta$

$$P(\hat{\theta}_I < \theta < \hat{\theta}_S) = 1 - \alpha$$

# Intervalo de confiança para a média

- Uma tarefa muito comum passa por calcular uma estimativa para a média da distribuição de valores, assumindo que estes seguem uma distribuição normal
- Dado que no caso mais comum não se conhece o desvio padrão da distribuição, este terá que ser aproximado pela estimativa da amostra
- Assim, usa-se a estatística  $T$ , que:
  - Para valores pequenos de  $n$ , segue a distribuição  $t$  de student com  $n-1$  graus de liberdade ( $n$  – tamanho da amostra)
  - Para valores grandes de  $n$ , segue uma distribuição normal

$$t = \frac{\bar{X} - \mu}{s/\sqrt{n}}$$

Média da amostra

Desvio padrão da amostra

# Intervalo de confiança para a média

**Com desvio padrão conhecido:**

$$\bar{x} - z_{1-\alpha/2} \frac{\sigma}{\sqrt{n}} < \mu < \bar{x} + z_{1-\alpha/2} \frac{\sigma}{\sqrt{n}}$$

**Sem desvio padrão conhecido ( $n$  pequeno):**

$$P\left(\bar{x} - t_{\alpha/2, n-1} \frac{s}{\sqrt{n}} < \mu < \bar{x} + t_{\alpha/2, n-1} \frac{s}{\sqrt{n}}\right) = 1 - \alpha$$

**Sem desvio padrão conhecido ( $n$  grande):**

Usar primeira expressão com  $s$  (valor do desvio padrão na amostra no lugar de  $\sigma$  (valor na população)



## Exemplo – intervalo confiança médias

```
> x = c(175,176,173,175,174,173,173,176,173,179)
> mean(x)
[1] 174.7
> se = sd(x) / sqrt(10)
> se
[1] 0.6155395
> qt(0.975,9)
[1] 2.262157
> intconf = mean(x) + c(-se*qt(0.975,9), se*qt(0.975,9))
> intconf
[1] 173.3076 176.0924
```

Função **t.test**:  
Permite calcular  
o valor do IC

```
> t.test(x)
      One Sample t-test
data:  x
t = 283.8161, df = 9, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 173.3076 176.0924
sample estimates:
mean of x
 174.7
```

## Exemplo – intervalo confiança médias

```
from scipy.stats import t
x = np.array([175, 176, 173, 175, 174, 173, 173, 176, 173, 179])

sample_mean = np.mean(x)
sample_std = np.std(x, ddof=1)

t_value = t.ppf(0.975, df=9)

margin_of_error = t_value * sample_std / np.sqrt(len(x))

confidence_interval = (sample_mean - margin_of_error, sample_mean + margin_of_error)
confidence_interval
```

173.30755288755185, 176.09244711244813

Função ***ttest\_1samp***:  
Permite calcular o  
valor do IC

```
from scipy.stats import ttest_1samp

ttest_result = ttest_1samp(x, popmean = 0)

ttest_result.confidence_interval()
```

# Testes de hipóteses

Como já foi claro, os testes de hipóteses têm relações com os intervalos de confiança mostrados atrás

Neste caso, geramos uma **hipótese  $H_0$**  (dita nula) sobre os valores de parâmetros e depois calculamos uma estatística de teste que nos permite calcular a probabilidade de podermos rejeitar esta hipótese

O principal resultado da maioria dos testes é o chamado ***p-value***. Este valor diz-nos a **probabilidade de obtermos o valor calculado da estimativa ou um valor pior, se assumirmos a hipótese nula**.

Valores suficientemente pequenos permitem assim **rejeitar a hipótese nula ( $H_0$ )**. O que é considerado suficientemente pequeno é determinado pelo nível de significância (confiança) assumido

Se não rejeitarmos a hipótese nula, o teste não é conclusivo

Muitas funções para testes de hipóteses são as mesmas usadas para cálculo de intervalos de confiança

# Testes paramétricos e não paramétricos

Testes **paramétricos**: e.g. t-test

- Assumem distribuição normal dos dados
- Dizem respeito a parâmetros especificados

Testes **não paramétricos**

- Não impõem qualquer condição relativa à distribuição dos dados
- Não há parâmetros especificados

# Teste às médias

Hipótese nula: média = 0 (por omissão)

Função: ***t.test***

– Argumentos: dados, *alternative* – diz se o teste é “two-sided”, “greater”, “less”

```
> t.test(x, mu=175)
      One Sample t-test
data:  x
t = -0.4874, df = 9, p-value = 0.6376
alternative hypothesis: true mean is not
equal to 175
95 percent confidence interval:
 173.3076 176.0924
sample estimates:
mean of x
 174.7
```

```
> t.test(x)
      One Sample t-test
data:  x
t = 283.8161, df = 9, p-value < 2.2e-16
alternative hypothesis: true mean is not
equal to 0
95 percent confidence interval:
 173.3076 176.0924
sample estimates:
mean of x
 174.7
```

# Teste às médias

Hipótese nula: média = valor a testar

Função: *ttest\_1samp*

- Argumentos: dados, *popmean* – média a testar ( $H_0$ );  
*alternative* – diz se o teste é “two-sided”, “greater”, “less”

```
> res = ttest_1samp(x, popmean = 175)
> print(res.pvalue)
> print(res.statistic)
> print(res.df)
```

```
0.6376422052940726
-0.4873773249665126
9
```

```
> res = ttest_1samp(x, popmean = 0)
> print(res.pvalue)
```

```
4.259313480395569e-19
```

## Testes à mediana

Calculada de forma diferente usando testes não paramétricos

Teste mais usado: **Wilcoxon test (ou Mann Whitney)**

```
> wilcox.test(x, conf.int=T)

      wilcoxon signed rank test with continuity correction

data:  x
V = 55, p-value = 0.005541
alternative hypothesis: true location is not equal to 0
95 percent confidence interval:
 173 176
sample estimates:
(pseudo)median
 174.5
```

# Testes à mediana

Teste mais usado: **Wilcoxon test (ou Mann Whitney)**

```
> from scipy.stats import wilcoxon  
  
> test_statistic, p_value = wilcoxon(x)  
  
> print("wilcoxon test statistic: {:.2f}".format(test_statistic))  
> print("p-value: {:.4f}".format(p_value))  
  
wilcoxon test statistic: 0.00  
p-value: 0.0020
```



# Testes a duas amostras

Usando variantes dos testes referidos podemos comparar duas amostras

Neste caso, a hipótese nula é tipicamente enunciada como a **igualdade da estimativa nas duas populações subjacentes**

Estas variantes podem ser aplicadas aos testes às proporções, às médias e às medianas vistos anteriormente, usando as mesmas funções vistas atrás

## Exemplo: testes às médias com duas amostras

H0: média(drug) – média(placebo) = 0

H1: média(drug) – média(placebo) < 0

Assume-se que as populações  
têm variâncias iguais

```
> drug = c(15, 10, 13, 7, 9, 8, 21, 9, 14, 8)
> placebo = c(15, 14, 12, 8, 14, 7, 16, 10, 15, 12)
> t.test(drug, placebo, alt="less", var.equal=TRUE)

Two Sample t-test
data: drug and placebo
t = -0.5331, df = 18, p-value = 0.3002
alternative hypothesis: true difference in means is less than 0
95 percent confidence interval:
 -Inf 2.027436
sample estimates:
mean of x mean of y
 11.4      12.3
```

Não rejeitar H0

## Exemplo: testes às médias com duas amostras

H0: média(diag\_med1) – média(diag\_med2) = 0

H1: média(diag\_med1) – média(diag\_med2) != 0

Assume que as amostras estão emparelhadas: cada valor da primeira corresponde a um da segunda

```
> diag_med1 = c(3, 0, 5, 2, 5, 5, 5, 4, 4, 5)
> diag_med2 = c(2, 1, 4, 1, 4, 3, 3, 2, 3, 5)
> t.test(diag_med1, diag_med2, paired=T)
      Paired t-test
data:  diag_med1 and diag_med2
t = 3.3541, df = 9, p-value = 0.008468
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 0.325555 1.674445
sample estimates:
mean of the differences
                1
```

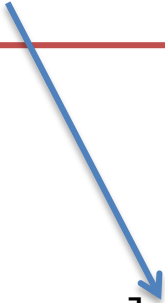
Rejeitar H0 -> médias são significativamente diferentes

## Exemplo: testes às médias com duas amostras

H0: média(drug) – média(placebo) = 0

H1: média(drug) – média(placebo) < 0

Assume-se que as populações  
têm variâncias iguais



```
from scipy.stats import ttest_ind

drug = np.array([15, 10, 13, 7, 9, 8, 21, 9, 14, 8])
placebo = np.array([15, 14, 12, 8, 14, 7, 16, 10, 15, 12])

test_statistic, p_value = ttest_ind(drug, placebo, alternative='less', equal_var=True)

print("t-test statistic: {:.2f}".format(test_statistic))
print("p-value: {:.4f}".format(p_value))
```

t-test statistic: -0.53

p-value: 0.3002

Não rejeitar H0

## Exemplo: testes às médias com duas amostras

H0: média(diag\_med1) – média(diag\_med2) = 0

H1: média(diag\_med1) – média(diag\_med2) != 0

Função *ttest\_rel*:

Assume que as amostras estão emparelhadas: cada valor da primeira corresponde a um da segunda

```
from scipy.stats import ttest_rel

diag_med1 = [3, 0, 5, 2, 5, 5, 5, 4, 4, 5]
diag_med2 = [2, 1, 4, 1, 4, 3, 3, 2, 3, 5]

t_stat, p_val = ttest_rel(diag_med1, diag_med2)

print("T-statistic:", t_stat)
print("p-value:", p_val)
```

T-statistic: 3.3541019662496843

p-value: 0.00846815040315423

Rejeitar H0 -> médias são significativamente diferentes

# Teste do qui-quadrado: ajuste

Teste do qui-quadrado: permite verificar se os dados provêm de uma dada população

Aplicado a variáveis categóricas

Estatística do teste:

$$\chi^2 = \sum_{i=1}^n \frac{(f_i - e_i)^2}{e_i}$$

$f_i$  – frequência da categoria  $i$ ;  $e_i$  – valor esperado

Hipótese nula: cada categoria tem probabilidade  $p_i$  (usada para calcular os valores esperados)

Função R: ***chisq.test***

– Argumentos: dados, vetor de probabilidades (por omissão: distribuição uniforme)

## Exemplos: teste qui-quadrado

```
> freq_dados = c(22,21,22,27,22,36)
> probs = rep(1/6,6)
> probs
[1] 0.1666667 0.1666667 0.1666667 0.1666667 0.1666667 0.1666667
> chisq.test(freq_dados, p=probs)
      Chi-squared test for given probabilities
data:  freq_dados
X-squared = 6.72, df = 5, p-value = 0.2423
```

H0: probabilidade de cada categoria = 1/6

Não rejeitar H0

```
> freqs_letras = c(100,110,80,55,14)
> probs = c(29, 21, 17, 17, 16)/100
> chisq.test(freqs_letras, p=probs)
      Chi-squared test for given probabilities
data:  x
X-squared = 55.3955, df = 4, p-value = 2.685e-11
```

Rejeitar H0

# Exemplos: teste qui-quadrado

```
> from scipy.stats import chisquare  
  
> freq_dados = [22, 21, 22, 27, 22, 36]  
> res = chisquare(freq_dados)  
> print(res.pvalue)  
  
0.24231086039631416
```

Por omissão probabilidades iguais em todos  
H0: probabilidade de cada categoria = 1/6  
Não rejeitar H0

Probabilidades distintas: temos  
que especificar as frequências  
esperadas

Rejeitar H0

```
> freqs_letras = [100,110,80,55,14]  
> sum_fl = sum(freqs_letras)  
> expected = np.array([0.29,0.21,0.17,0.17,0.16])*sum_fl  
> res = chisquare(freqs_letras, expected)  
> print(res.pvalue)  
  
2.6845339866838983e-11
```



# Aproximação à distribuição normal: visualização com *QQ plots*

Dada a sua importância em vários tipos de análise de dados, determinar a conformidade de um conjunto de dados com a distribuição normal é essencial

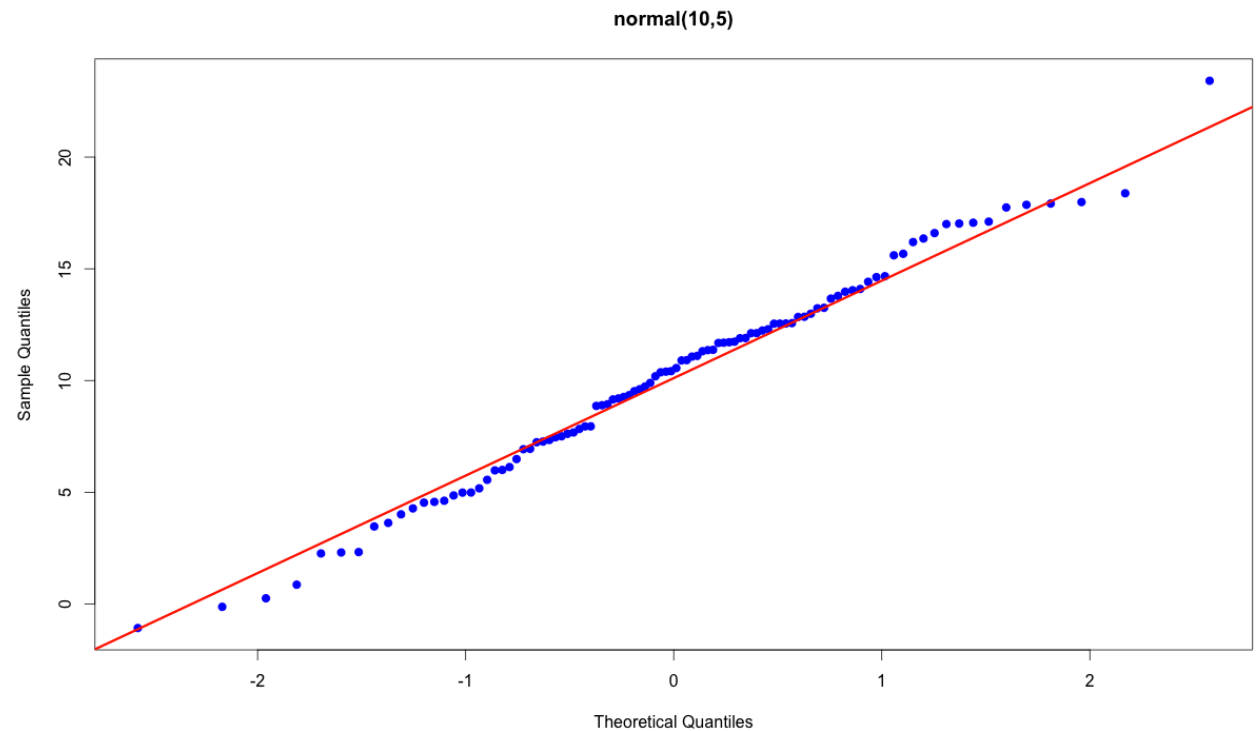
Uma forma alternativa aos histogramas para a visualização de dados e sua conformidade com uma distribuição normal são os gráficos quantil-quantil (**QQ plots**)

Para comparação genérica de duas distribuições, o R tem a função **qqplot**

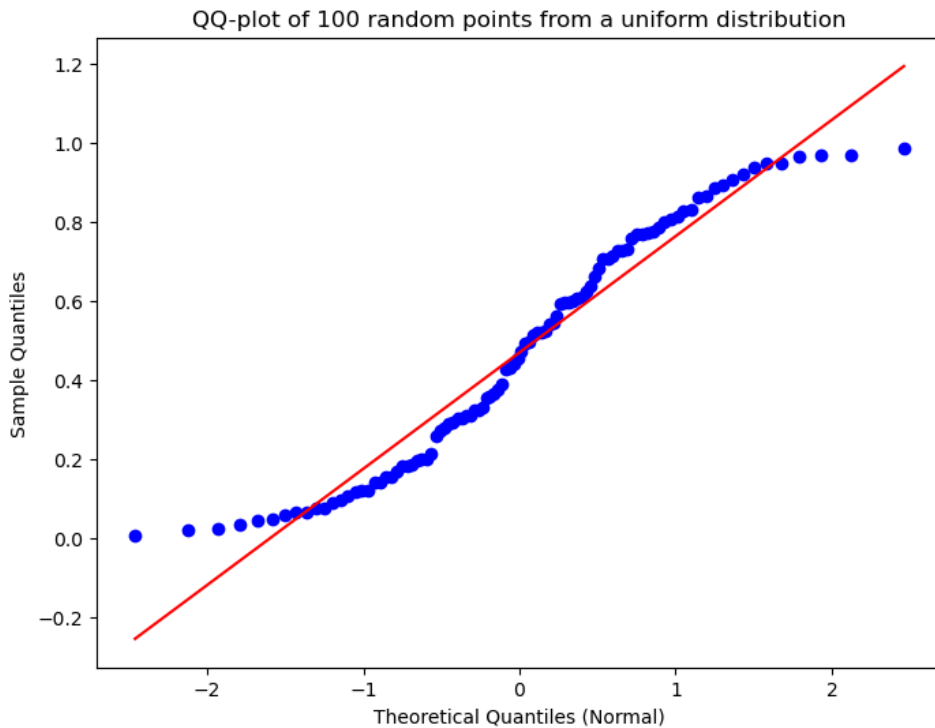
Para representar os quantis de um conjunto de dados vs os quantis teóricos da distribuição normal, o que permite verificar, em termos visuais, o ajuste dos dados usam-se as funções : **qqnorm** e **qqline**

# QQ plots: exemplos

```
> x = rnorm(100, 10, 5)
> qqnorm(x, main='normal(10,5)', pch=19, col="blue")
> qqline(x, col="red", lwd=3)
```



# QQ plots: exemplos



```
# Create a quantile-quantile plot of the data
fig = plt.figure(figsize=(8,6))
ax = fig.add_subplot(111)
stats.probplot(data, dist="norm", plot=ax)

plt.xlabel("Theoretical Quantiles (Normal)")
plt.ylabel("Sample Quantiles")
plt.title("QQ-plot of 100 random points from a
uniform distribution")
plt.show()
```

# Testes Kolmogorov-Smirnov

Testes não paramétricos que permitem:

- Verificar se uma amostra está de acordo com uma distribuição teórica
- Comparar duas amostras verificando se podem advir da mesma distribuição

Hipótese nula:

- A amostra segue a distribuição teórica
- As duas amostras provêm da mesma distribuição

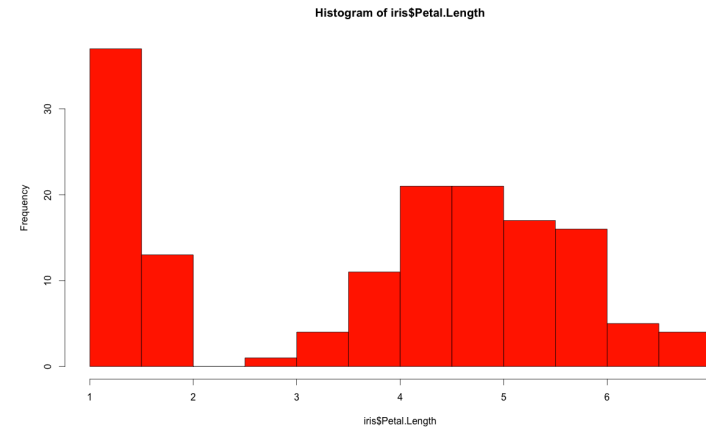
Função: *ks.test*

# Exemplos: Kolmogorov-Smirnov

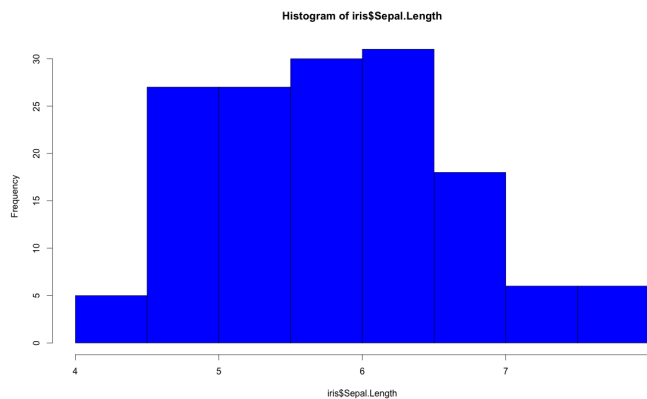
```
> ks.test(as.vector(scale(iris$Petal.Length)), "pnorm")
```

One-sample Kolmogorov-Smirnov test

```
data:
as.vector(scale(iris$Petal.Length))
D = 0.1982, p-value = 1.532e-05
alternative hypothesis: two-sided
```



Qual será a variável que tem melhor aproximação à distribuição normal ?



```
> ks.test(as.vector(scale(iris$Sepal.Length)), "pnorm")
```

One-sample Kolmogorov-Smirnov

test

```
data:
as.vector(scale(iris$Sepal.Length))
D = 0.0887, p-value = 0.1891
alternative hypothesis: two-sided
```

Em Python: **ks\_test**  
package `scipy.stats`

# Teste à normalidade dos dados: Shapiro

```
> n = rnorm(100)
> shapiro.test(n)
      Shapiro-Wilk normality test
data:  n
W = 0.9864, p-value = 0.396
```

Não se rejeita H0

**Hipótese nula:**

Dados seguem uma distribuição normal

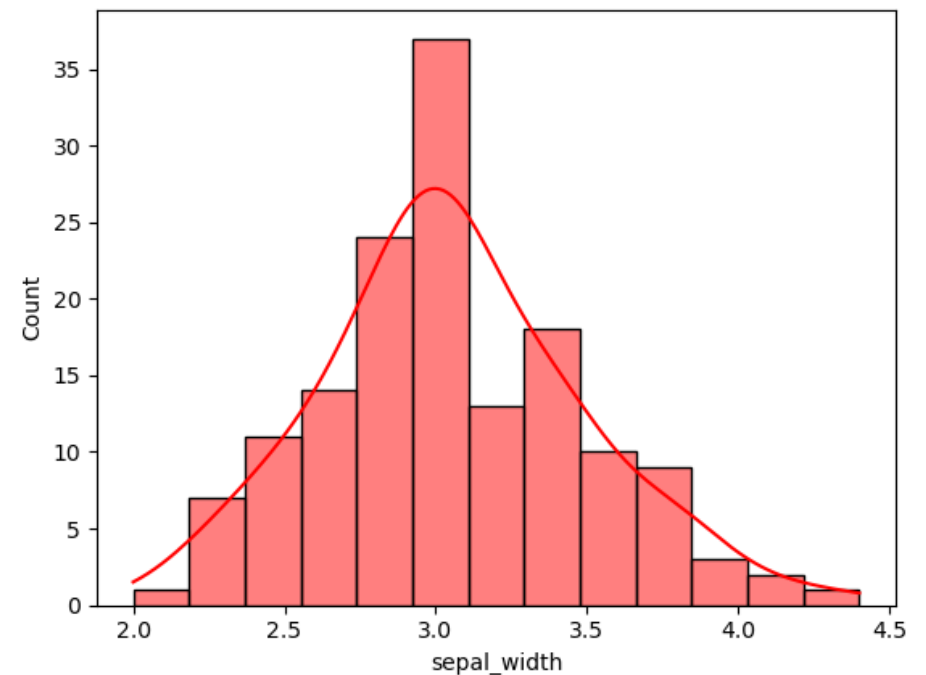
```
> n = runif(100)
> shapiro.test(n)
      Shapiro-Wilk normality test
data:  n
W = 0.9546, p-value = 0.001685
```

Rejeita-se H0

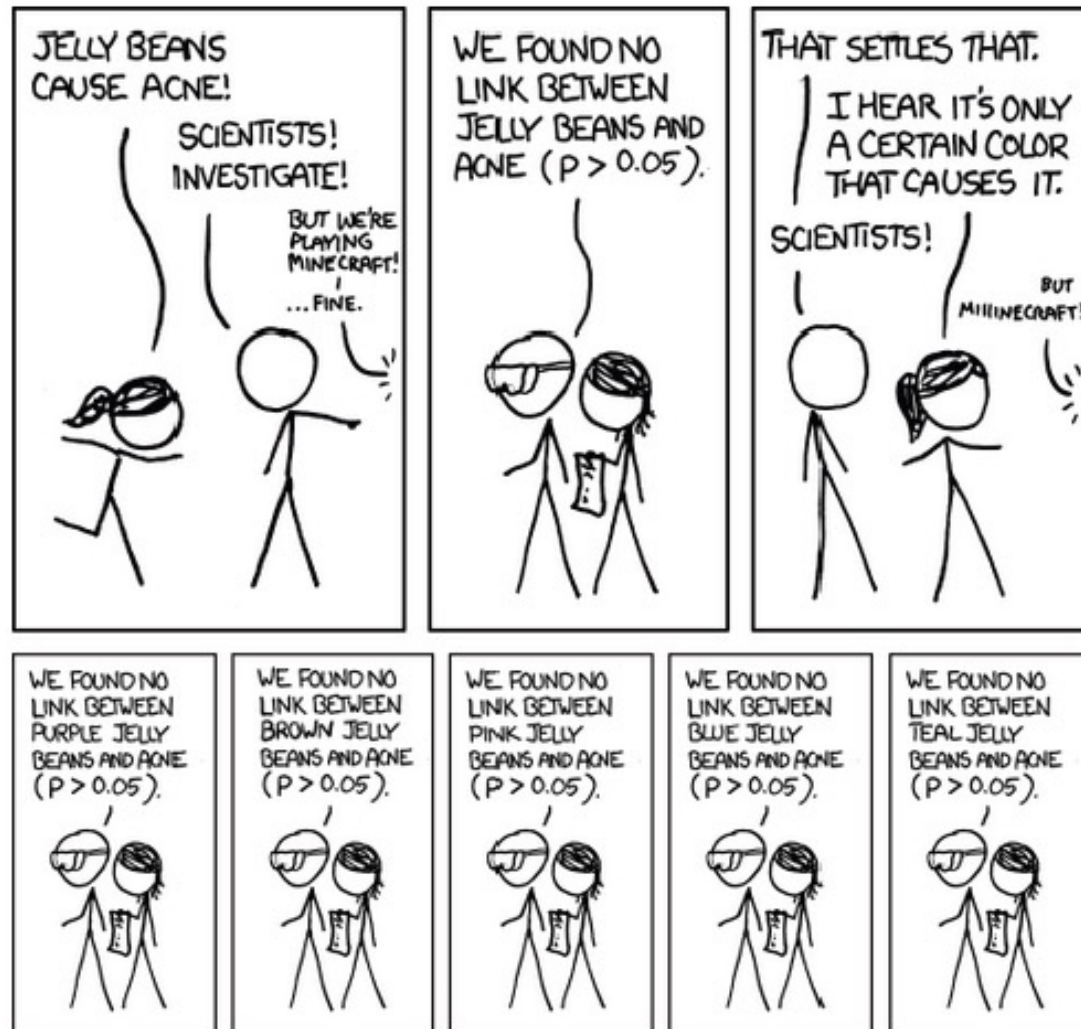
# Teste de Shapiro: exemplo

```
iris = sns.load_dataset('iris')  
  
sns.histplot(data=iris, x='sepal_width', color='red', kde=True)  
  
_, p_value = stats.shapiro(iris['sepal_width'])  
print("Shapiro-wilk p-value: ", p_value)
```

Shapiro-wilk p-value: 0.10112646222114563



# Alguns cuidados a ter

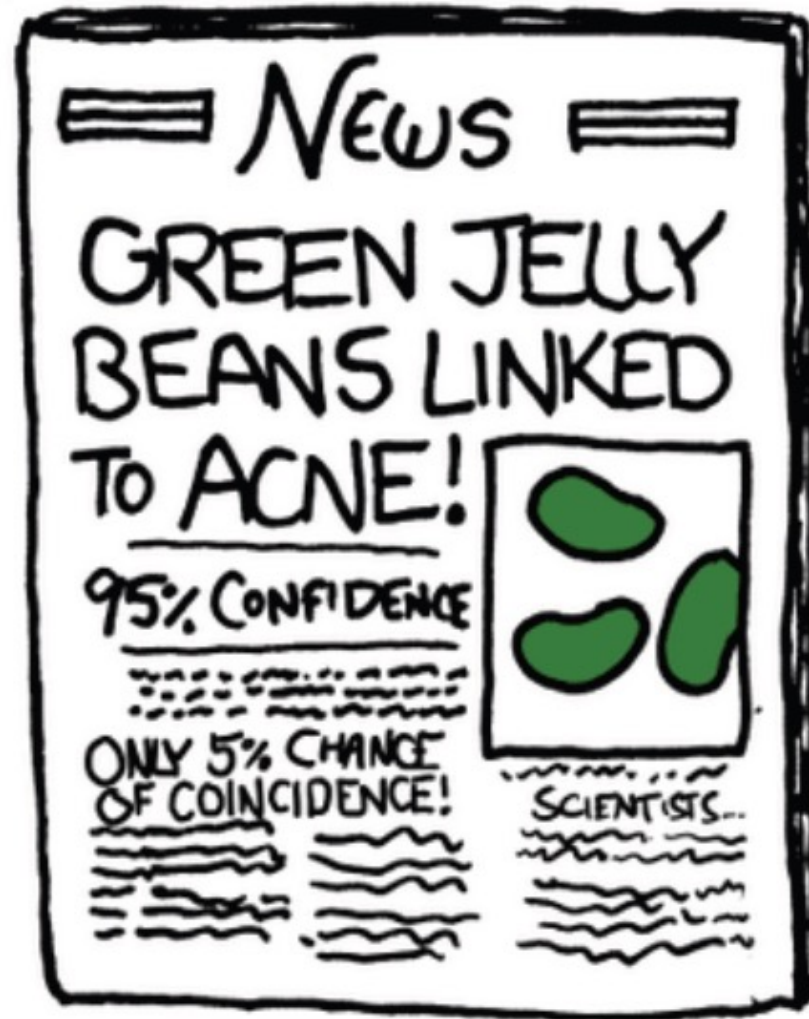


(...)



# Alguns cuidados a ter

(...)



<http://xkcd.com/882>

# Testes múltiplos

Se um teste tem  $\alpha = 1\%$  - erros (tipo I) esperados são de 1 em 100

Se fizermos 100 testes, teremos expectavelmente 1 falso positivo; se fizermos 10000 testes teremos 100 ...

Fazendo N testes, para um dado  $\alpha$  teremos  $(1-\alpha)*N$  erros esperados (falsos positivos)

Necessidade de fazer correções !!

# Testes múltiplos: soluções

Controlar a FWER (*family wise error rate*):

- Definir um novo  $\alpha$  dividindo o original pelo nº de testes realizados – correção de Bonferroni
- Método bastante conservador

Controlar a FDR (*false discovery rate*): correções BH (Benjamini / Hochberg) e BY

- Menos conservador
- Calcula p-values e ordena-os; nível  $\alpha$  de significância varia ao longo deste ranking

Função **p.adjust** em R faz a correção dos p.values de acordo com um destes métodos (argumento *method*)

Em Python, o package statsmodels tem um módulo stats.multitest, que inclui a função **multipletests** com um comportamento similar

# Testes múltiplos: exemplos

```
> set.seed(12345)
> m = matrix(rnorm(100*60),100,60)
> tt = function(x) t.test(x[1:30],x[31:60])$p.value
> pvalues = apply(m, 1, tt)
> sum(pvalues < 0.05)
[1] 5
> adj.bonf = p.adjust(pvalues, "bonferroni")
> sum(adj.bonf < 0.05)
[1] 0
> adj.bh = p.adjust(pvalues, "BH")
> sum(adj.bh < 0.05)
[1] 0
```

Não há verdadeiros positivos

# Testes múltiplos: exemplos

```
> set.seed(12345)
> m1 = matrix(rnorm(50*60),50,60)
> m2 = cbind(matrix(rnorm(50*30),50,30),
                matrix(rnorm(50*30, mean = 1),50,30) )
> mt = rbind(m1, m2)
> pvalues.2 = apply(mt, 1, tt)
> res.verd = c(rep("nao rej",50), rep("rej",50))
> table(pvalues.2 < 0.05, res.verd)
      nao rej rej
FALSE      47   3
TRUE        3  47
> adj.bonf.2 = p.adjust(pvalues.2, "bonferroni")
> table(adj.bonf.2 < 0.05, res.verd)
      nao rej rej
FALSE      50  20
TRUE        0  30
> adj.bh.2 = p.adjust(pvalues.2, "BH")
> table(adj.bh.2 < 0.05, res.verd)
      nao rej rej
FALSE      49   4
TRUE        1  46
```

50% de verdadeiros positivos

# Análise de variância simples

**Análise de variância** constitui um conjunto de modelos estatísticos e testes associados que pretende particionar a variância de cada observação nos diversos componentes atribuídos a cada fonte de variação

Versão mais simples da análise de variância: pretende analisar se a média de vários grupos de observações é a mesma

Generalização dos testes à média com duas amostras para o caso de  $N (>2)$  amostras

Realizado como um teste estatístico, assumindo que os dados seguem a distribuição normal, cuja hipótese nula é a de que as médias de cada um dos grupos é igual

Estatística F: compara variabilidade entre cada grupo com variabilidade entre grupos, seguindo a distribuição F de Fisher (funções R: **df**, **pf**, **qf**, **rf**)

# Análise de variância simples: exemplo

```
> fligner.test(values ~ ind, data= notas_st)
Fligner-Killeen test of homogeneity of variances
data:  values by ind
Fligner-Killeen:med chi-squared = 1.3956, df = 2,
p-value = 0.4977
```

Teste prévio à  
homogeneidade  
das variâncias

```
> prof1 = c(4,3,4,5,2,3,4,5)
> prof2 = c(4,4,5,5,4,5,4,4)
> prof3 = c(3,4,2,4,5,5,4,4)
> notas = data.frame(prof1, prof2, prof3)
> notas_st = stack(notas)
> notas_st
  values ind
1      4 prof1
2      3 prof1
...
16     4 prof2
17     3 prof3
...
> oneway.test(values ~ ind, data=notas_st, var.equal=T)
One-way analysis of means
data:  values and ind
F = 1.1308, num df = 2, denom df = 21, p-value = 0.3417
```

Função *oneway.test*  
Calcula estatística F  
e a partir desta o valor do p-value

Não rejeitar  
hipótese nula

# Análise de variância simples: exemplo

```
> anv$coeff
(Intercept)      indprof2      indprof3
          3.750           0.625           0.125
```

```
> anv = aov(values ~ ind, data=notas_st)
> anv
Call:
aov(formula = values ~ ind, data = notas_st)
Terms:
          ind Residuals
Sum of Squares  1.75      16.25
Deg. of Freedom    2       21

Residual standard error: 0.8796644
Estimated effects may be unbalanced
> summary(anv)
```

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
ind	2	1.75	0.8750	1.131	0.342
Residuals	21	16.25	0.7738		

Função **aov**

Retorna detalhes sobre a análise de variância

Linha *ind* referente à variação devida ao “tratamento” (diferenças entre os grupos)

Resultados idênticos com **anova(lm(values~ind, data= notas\_st))**



## Análise de variância simples: exemplo

```
from scipy.stats import f_oneway

prof1 = [4,3,4,5,2,3,4,5]
prof2 = [4,4,5,5,4,5,4,4]
prof3 = [3,4,2,4,5,5,4,4]
notas = pd.DataFrame({'prof1': prof1, 'prof2': prof2, 'prof3': prof3})
F, p = f_oneway(notas.prof1, notas.prof2, notas.prof3)
print('One-way analysis of means')
print(f'F = {F}, p-value = {p}')
```

One-way analysis of means  
F = 1.1307692307692307, p-value = 0.34166385145199196

Não rejeitar  
hipótese nula

# Análise de variância não paramétrica

Quando os dados não seguem uma distribuição normal há necessidade de usar testes **não paramétricos**

O teste não paramétrico mais conhecido para análise de variância é o teste de **Kruskal-Wallis**:

```
> kruskal.test(values ~ ind, data= notas_st)
```

```
    Kruskal-Wallis rank sum test
```

```
data:  values by ind
```

```
Kruskal-Wallis chi-squared = 1.9387, df = 2, p-value = 0.3793
```

R

python

```
> stats.kruskal(notas.prof1, notas.prof2, notas.prof3)
```

```
KruskalResult(statistic=1.938748079877106, pvalue=0.3793204032280171)
```

# Análise de variância com fatores

```
> anvf = aov(iris$Petal.Length ~ iris$Species)
> summary(anvf)
```

	Df	Sum Sq	Mean Sq	F value	Pr(>F)	
iris\$Species	2	437.1	218.55	1180	<2e-16	***
Residuals	147	27.2	0.19			

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
> TukeyHSD(anvf)
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = iris$Petal.Length ~ iris$Species)

$`iris$Species`
```

	diff	lwr	upr	p adj
versicolor-setosa	2.798	2.59422	3.00178	0
virginica-setosa	4.090	3.88622	4.29378	0
virginica-versicolor	1.292	1.08822	1.49578	0

## ***Tukey's HSD test***

(honestly significant difference):

Usado em conjunto com a ANOVA para identificar médias significativamente diferentes entre si

# Análise de variância com fatores

```
import statsmodels.stats.multicomp as mc
from statsmodels.formula.api import ols
from statsmodels.stats.anova import anova_lm

model = ols('petal_length ~ species', data=iris).fit()
anova_results = anova_lm(model)

# perform Tukey HSD test
tukey_results = mc.pairwise_tukeyhsd(endog=iris['petal_length'], groups=iris['species'])

print(anova_results)
print(tukey_results)
```

	df	sum_sq	mean_sq	F	PR(>F)
species	2.0	437.1028	218.551400	1180.161182	2.856777e-91
Residual	147.0	27.2226	0.185188	NaN	NaN

Multiple Comparison of Means - Tukey HSD, FWER=0.05

```
=====
```

group1	group2	meandiff	p-adj	lower	upper	reject
setosa	versicolor	2.798	0.0	2.5942	3.0018	True
setosa	virginica	4.09	0.0	3.8862	4.2938	True
versicolor	virginica	1.292	0.0	1.0882	1.4958	True

```
=====
```

# Análise de Regressão (linear)

Objetivo: determinar relações (lineares) entre variáveis

- Identificar se variáveis estão relacionadas
- Estimar parâmetros da relação que liga as variáveis
- Definir modelos para prever uma variável (dependente) a partir de outra(s)

# Regressão linear

Caso geral: modelos de **regressão**:

$$\hat{y} = \beta_0 + \sum_{i=1}^p \beta_i x_i$$

Se  $p = 1$  : **regressão linear**

Se  $p \geq 2$ : **regressão linear múltipla**

$\beta_i$  - parâmetros do modelo

Valor esperado de aumento de  $y$ , se  $x$  aumenta 1 unidade

Valor esperado de  $y$  se  $x = 0$

# Regressão linear simples

Assume duas variáveis numéricas que estão correlacionadas linearmente (x – variável independente; y – variável dependente)

Determina melhor modelo linear que permite calcular y a a partir de x

São determinados o declive da reta e a interseção com o eixo dos yy (coordenada na origem)

Parâmetros determinados com base no método dos mínimos quadrados, i.e. aqueles que minimizam a soma do quadrado dos erros para todos os dados conhecidos (diferença entre o valor previsto para a variável dependente pelo modelo linear e os valores reais)

Modelo:

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$

Estimados

Resíduos

# Regressão linear em R

Função R: *lm*

- Argumento obrigatório: *formula*
- Outros: *data* – conjunto de dados, ...

Resultado é uma lista com diversos campos:

- *coefficients*: os valores estimados (betas)
- *residuals*: erros
- *fitted.values*: valores estimados

Plot do resultado dá alguns gráficos que podem dar uma ideia do erro

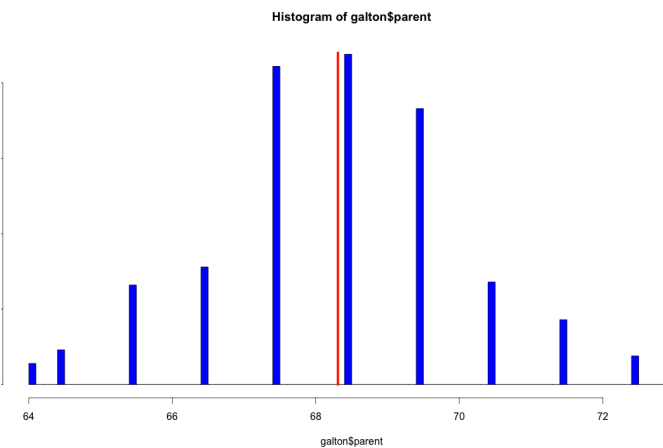
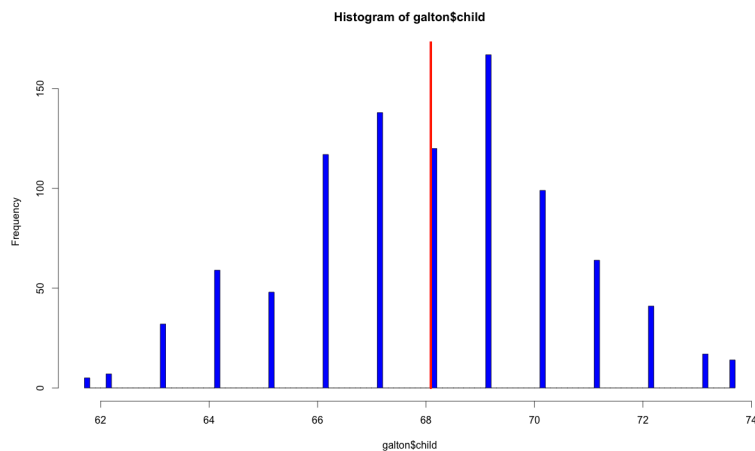
Função alternativa mais robusta (quando há outliers) – função *rlm* do package MASS



# Regressão linear: exemplo

```
> library(UsingR)
> data(galton)
> names(galton)
[1] "child" "parent"
> hist(galton$child,col="blue",breaks=100)
> abline(v=mean(galton$child), col = "red", lwd = 4)
> hist(galton$parent,col="blue",breaks=100)
> abline(v=mean(galton$parent), col = "red", lwd = 4)
```

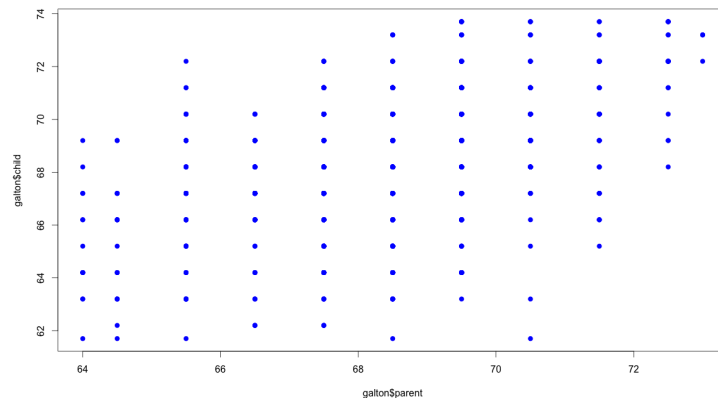
Dataset de alturas  
de pais e filhos  
recolhido por Galton



# Regressão linear: exemplo

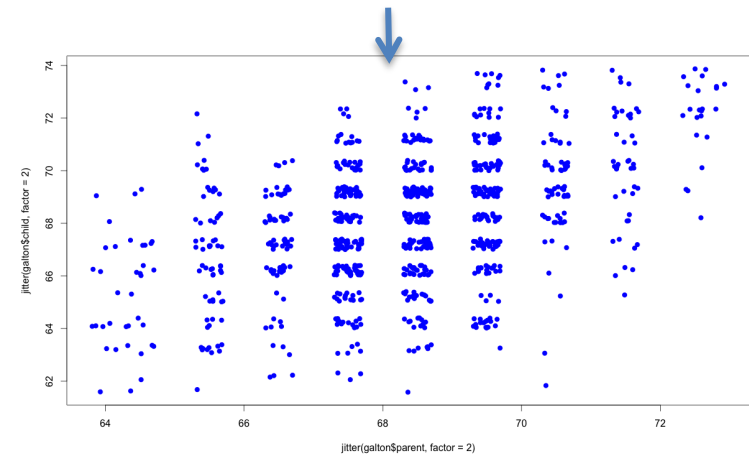
Correlação

```
> plot(galton$parent, galton$child, pch=19, col="blue")  
> cor(galton$child, galton$parent)  
[1] 0.4587624  
> plot(jitter(galton$parent, factor=2), jitter(galton$child, factor=2),  
pch=19, col="blue")
```



No gráfico original há muitos pontos sobrepostos devido ao arredondamento dos valores medidos por Galton

Gráfico de dispersão com ruído: ajuda a perceber a densidade de pontos quando estes estão sobrepostos

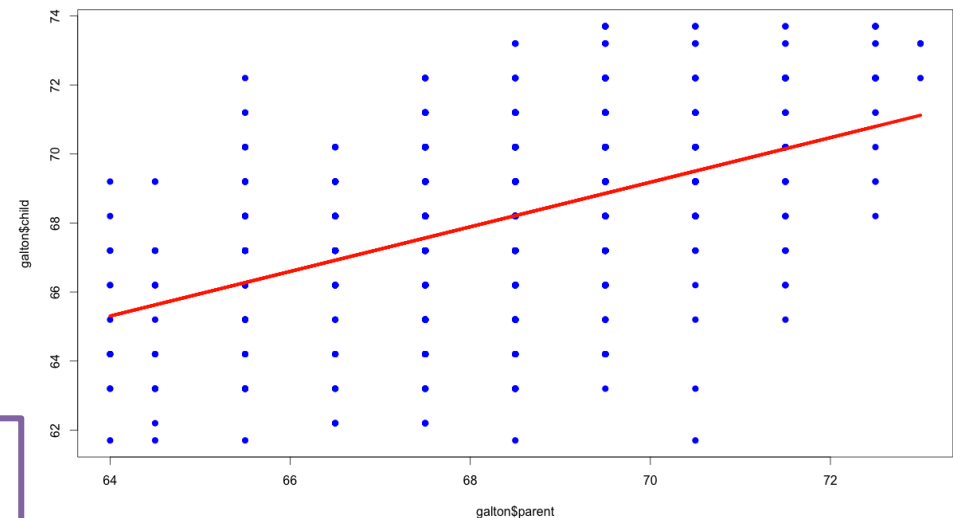


# Regressão linear: exemplo

```
> lm1 <- lm(child ~ parent, data=galton)
> lm1$coefficients
(Intercept)      parent 
 23.9415302    0.6462906 
> plot(galton$parent, galton$child, pch=19, col="blue")
> lines(galton$parent, lm1$fitted, col="red", lwd=5)
```

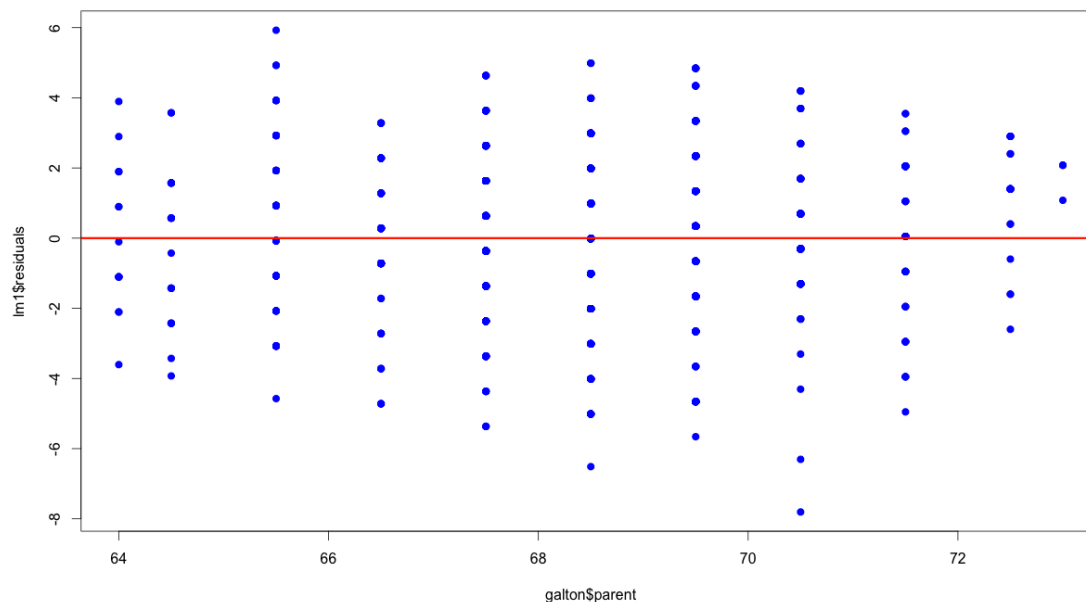
$$C = 23.94 + 0.646 P$$

```
> attach(galton)
> beta1 = cor(child, parent) * sd(child) / sd(parent)
> beta0 = mean(child) - beta1 * mean(parent)
> c(beta0, beta1)
[1] 23.9415302 0.6462906
```



# Regressão linear: exemplo

```
> plot(galton$parent, lm1$residuals, col="blue", pch=19)  
> abline(c(0,0), col="red", lwd=3)
```



Diferença entre os pontos e a linha – erros do modelo (resíduos)

# Regressão linear: exemplo

Intervalos confiança

```
> summary(lm1)
Residuals:
    Min       1Q   Median       3Q      Max
-7.8050 -1.3661  0.0487  1.6339  5.9264
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 23.94153    2.81088   8.517  <2e-16 ***
parent       0.64629    0.04114  15.711  <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.239 on 926 degrees of freedom
Multiple R-squared:  0.2105,    Adjusted R-squared:  0.2096
F-statistic: 246.8 on 1 and 926 DF,  p-value: < 2.2e-16
```

```
> confint(lm1)
                2.5 %    97.5 %
(Intercept) 18.4250996 29.4579608
parent      0.5655602  0.7270209
```

Distribuição dos erros

Testes hipóteses  
 $H_0: \beta_0 = 0$   
 $H_0: \beta_1 = 0$

Em ambos os casos:  
Rejeitar  $H_0$

```
> cor(child, parent) ^2
[1] 0.2104629
> anova(lm(child ~ parent))
            Df Sum Sq Mean Sq F value    Pr(>F)
parent      1 1236.9  1236.93   246.84 < 2.2e-16 ***
Residuals 926 4640.3     5.01
> 1237/(4640+1237)
[1] 0.2104815
```

Em regressão linear  
simples, o coeficiente de  
determinação é igual ao  
quadrado da correlação

# Regressão linear: previsão

```
> predict(lm1, data.frame(parent=c(64, 65, 66, 67, 68)))  
      1      2      3      4      5  
65.30413 65.95042 66.59671 67.24300 67.88929
```

Função **predict** permite prever valores usando o modelo criado  
Estes podem ser calculados com intervalos de confiança

```
> predict(lm1, data.frame(parent=c(64, 65, 66, 67, 68)),  
  interval = "confidence")  
      fit      lwr      upr  
1 65.30413 64.92761 65.68064  
2 65.95042 65.64690 66.25394  
3 66.59671 66.36108 66.83234  
4 67.24300 67.06425 67.42175  
5 67.88929 67.74294 68.03563
```

# Regressão múltipla

```
> fit1 = lm(iris$Petal.Length ~ iris$Petal.width + iris$Sepal.Length)
> fit1
Coefficients:
      (Intercept)  iris$Petal.width  iris$Sepal.Length
        -1.5071         1.7481         0.5423
> summary(fit1)
Residuals:
      Min       1Q   Median       3Q      Max
-1.15506 -0.21920 -0.02115  0.25986  1.35204
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)   -1.50714    0.33696  -4.473 1.54e-05 ***
iris$Petal.width  1.74810    0.07533  23.205 < 2e-16 ***
iris$Sepal.Length 0.54226    0.06934   7.820 9.41e-13 ***
```

```
> anv2 = aov(iris$Petal.Length ~ iris$Petal.width + iris$Sepal.Length)
> summary(anv2)
              Df Sum Sq Mean Sq F value    Pr(>F)
iris$Petal.width    1  430.5    430.5 2647.53 < 2e-16 ***
iris$Sepal.Length    1    9.9     9.9   61.15 9.41e-13 ***
Residuals          147   23.9     0.2
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

# Regressão múltipla (com interações)

```
> fit2 = lm(iris$Petal.Length ~ iris$Petal.Width * iris$Sepal.Length)
> fit2
Coefficients:
                (Intercept)                iris$Petal.Width
                -3.2480                  2.9712
    iris$Sepal.Length  iris$Petal.Width:iris$Sepal.Length
                0.8755                  -0.2225

> summary(fit2)
Residuals:
    Min       1Q   Median       3Q      Max
-0.99588 -0.24329  0.00355  0.29735  1.24780

Coefficients:
                (Intercept)                iris$Petal.Width
                -3.24804                  2.97115
    iris$Sepal.Length  iris$Petal.Width:iris$Sepal.Length
                0.87551                  -0.22248
---
Estimate Std. Error t value Pr(>|t|)
(Intercept) -3.24804    0.59586  -5.451 2.08e-07 ***
iris$Petal.Width  2.97115    0.35836   8.291 6.74e-14 ***
iris$Sepal.Length  0.87551    0.11667   7.504 5.60e-12 ***
iris$Petal.Width:iris$Sepal.Length -0.22248    0.06384  -3.485 0.00065 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.3888 on 146 degrees of freedom
Multiple R-squared:  0.9525,    Adjusted R-squared:  0.9515
F-statistic: 975.4 on 3 and 146 DF,  p-value: < 2.2e-16
```



# Regressão com fatores

```
> fit3 = lm(iris$Petal.Length ~ iris$Species)
> fit3
Coefficients:
            (Intercept)  iris$Speciesversicolor  iris$Speciesvirginica
                1.462                2.798                4.090
> summary(fit3)

Residuals:
    Min       1Q   Median       3Q      Max
-1.260 -0.258  0.038  0.240  1.348

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)    1.46200    0.06086   24.02  <2e-16 ***
iris$Speciesversicolor  2.79800    0.08607   32.51  <2e-16 ***
iris$Speciesvirginica   4.09000    0.08607   47.52  <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.4303 on 147 degrees of freedom
Multiple R-squared:  0.9414,    Adjusted R-squared:  0.9406
F-statistic: 1180 on 2 and 147 DF,  p-value: < 2.2e-16
```

O comando *anova(fit3)* permite fazer a análise de variância

# Regressão múltipla com fatores

```
> fit4 = lm(iris$Petal.Length ~ iris$Petal.width + iris$Species)
> fit4
Coefficients:
            (Intercept)            iris$Petal.width  iris$Speciesversicolor
                1.211                  1.019                  1.698
iris$Speciesvirginica
                2.277
> summary(fit4)
Residuals:
    Min       1Q   Median       3Q      Max
-1.02977 -0.22241 -0.01514  0.18180  1.17449

Coefficients:
                Estimate Std. Error t value Pr(>|t|)
(Intercept)      1.21140    0.06524   18.568 < 2e-16 ***
iris$Petal.width  1.01871    0.15224    6.691 4.41e-10 ***
iris$Speciesversicolor 1.69779    0.18095    9.383 < 2e-16 ***
iris$Speciesvirginica  2.27669    0.28132    8.093 2.08e-13 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

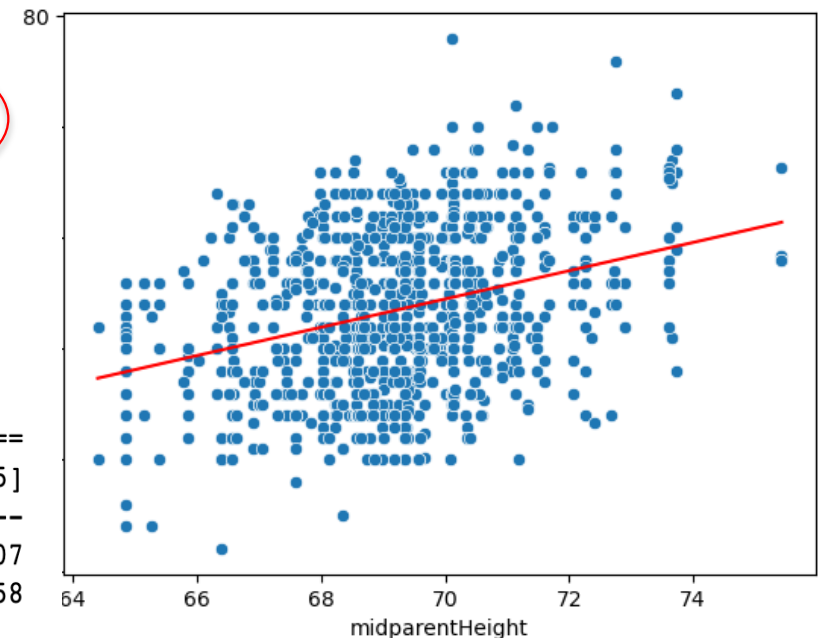
Residual standard error: 0.3777 on 146 degrees of freedom
Multiple R-squared:  0.9551,    Adjusted R-squared:  0.9542
F-statistic: 1036 on 3 and 146 DF,  p-value: < 2.2e-16
```

# Regressão linear: exemplo

```
# Load the Galton dataset
galton_df = sm.datasets.get_rdataset('GaltonFamilies', package='HistData').data
# Plot the data
sns.scatterplot(data=galton_df, x='midparentHeight', y='childHeight')
# Fit the linear regression model
model = ols('childHeight ~ midparentHeight', data=galton_df).fit()
print(model.summary())
# Plot the regression line
sns.lineplot(data=galton_df, x='midparentHeight', y=model.fittedvalues, color='red')
plt.show()
```

```
=====
Dep. Variable:          childHeight    R-squared:                0.103
Model:                  OLS           Adj. R-squared:            0.102
Method:                 Least Squares   F-statistic:             107.0
Date:                   Wed, 01 Mar 2023 Prob (F-statistic):      8.05e-24
Time:                   22:33:34        Log-Likelihood:         -2465.0
No. Observations:       934            AIC:                     4934.
Df Residuals:           932            BIC:                     4944.
Df Model:               1
Covariance Type:        nonrobust
C = 22.64 + 0.637 P
=====
```

	coef	std err	t	P> t	[0.025	0.975]
Intercept	22.6362	4.265	5.307	0.000	14.266	31.007
midparentHeight	0.6374	0.062	10.345	0.000	0.516	0.758



# Regressão múltipla

```
X = iris[['petal_width', 'sepal_length']]
y = iris['petal_length']

X = sm.add_constant(X) # add an intercept term to the model
model = sm.OLS(y, X).fit()

print(model.summary())
```

```
=====
Dep. Variable:          petal_length    R-squared:                0.949
Model:                  OLS             Adj. R-squared:           0.948
Method:                 Least Squares   F-statistic:              1354.
Date:                   Wed, 01 Mar 2023 Prob (F-statistic):       2.01e-95
Time:                   22:40:56         Log-Likelihood:           -75.090
No. Observations:      150              AIC:                    156.2
Df Residuals:           147              BIC:                    165.2
Df Model:               2
Covariance Type:        nonrobust
=====
```

	coef	std err	t	P> t	[ 0.025	0.975]
const	-1.5071	0.337	-4.473	0.000	-2.173	-0.841
petal_width	1.7481	0.075	23.205	0.000	1.599	1.897
sepal_length	0.5423	0.069	7.820	0.000	0.405	0.679

# Regressão múltipla

```
# fit linear regression model with interaction term
fit2 = sm.OLS.from_formula("petal_length ~ petal_width * sepal_length", iris).fit()

# print summary
print(fit2.summary())
```

```
=====
Dep. Variable:          petal_length    R-squared:                0.952
Model:                  OLS            Adj. R-squared:           0.952
Method:                 Least Squares   F-statistic:              975.4
Date:                   Wed, 01 Mar 2023 Prob (F-statistic):       2.45e-96
Time:                   22:50:36        Log-Likelihood:          -69.096
No. Observations:      150             AIC:                    146.2
Df Residuals:          146             BIC:                    158.2
Df Model:               3
Covariance Type:       nonrobust
=====
```

	coef	std err	t	P> t	[0.025	0.975]
Intercept	-3.2480	0.596	-5.451	0.000	-4.426	-2.070
petal_width	2.9712	0.358	8.291	0.000	2.263	3.679
sepal_length	0.8755	0.117	7.504	0.000	0.645	1.106
petal_width:sepal_length	-0.2225	0.064	-3.485	0.001	-0.349	-0.096