0.0. Imports

0.1. Julia & Python Imports

In [5]:

```
using CSV;
using PyCall;
using PyPlot;
using Printf;
using StatsBase;
using DataFrames, FreqTables
using HypothesisTests

np  = pyimport("numpy");
sns = pyimport("seaborn");
ss  = pyimport("scipy.stats");
wrg = pyimport("warnings");
pd  = pyimport("pandas");
gs  = pyimport("matplotlib.gridspec");
```

In [2]:

```
ps = pyimport("pyspark.pandas");
pd = pyimport("databricks.koalas");
tree = pyimport("sklearn.tree");
layers = pyimport("keras.layers");
models = pyimport("keras.models");
prepro = pyimport("sklearn.preprocessing");
utils  = pyimport("sklearn.utils");
metrics = pyimport("sklearn.metrics");
model_selection = pyimport("sklearn.model_selection");
```

0.2. Aux Functions

0.2.1. Functions

In [294…

```
wrg.filterwarnings("ignore")

pearson_r(ti, df) = √( (ti^2) / (ti^2 + df ) );
glass_delt( x_bari, x_barj, stdi ) = (x_bari - x_barj) / stdi;
cohen_d(x_bari, x_barj, stdi, stdj) = ( x_bari - x_barj ) / √( ( stdi^2 + stdj^2 ) / 2 );
hedge_g( d, ni, nj ) = ( d * ( 1 - ( 3 / (4*(ni + nj - 9))))));

function cramer_v( x, y )
    cm = freqtable(x, y)
    n  = sum(cm)
    r, k = size(cm)

    chi2 = ChisqTest( cm ).stat
    chi2corr = max( 0, chi2 - ((( k-1 ) * ( r-1 )) / ( n-1 ) ))

    kcorr = k - ((( k-1 )^2) / ( n-1 ))
    rcorr = r - ((( r-1 )^2) / ( n-1 ))

    return sqrt( ( chi2corr / n ) / ( min( kcorr - 1, rcorr - 1 ) ) )
end;

function permutation(x, n_a, n_b)
    n = n_a + n_b
    rand_b = sample(1:n, n_b)
    rand_a = setdiff(1:n, rand_b)
    return np.mean(x[rand_b]) - np.mean(x[rand_a])
end;

function pages()
    info = []
    for i in ["Page A", "Page B"]
        mean = np.mean( df[df.Page .== i, 2] )
        std = np.std( df[df.Page .== i, 2] )
        n = length(df[df.Page .== i, 2])
        [append!(info, k) for k in [mean, std, n]];
    end
    return Dict("mean_a" => info[1], "std_a" => info[2], "n_a" => info[3],
                "mean_b" => info[4], "std_b" => info[5], "n_b" => info[6])
end;

function generate_data( x, y, z )
    res = []
    for k in [x, y, z]
```

```
        mean = np.mean( k )
        std = np.std( k )
        append!(res, [mean, std])
    end
    return Dict("mean_x1" => res[1], "std_x1" => res[2],
                "mean_x2" => res[3], "std_x2" => res[4],
                "mean_x3" => res[5], "std_x3" => res[6])
end

function permutation_chi2_julia()
    data = append!(vec(zeros(1, 2266)), vec(ones(1, 34)))
    data = round.(Int, data)
    values = [[ 14,   8,  12], [986, 992, 988]]
    expected = [34/3, 1000-(34/3)];
    function chi2( observed, expected )
        pearson_res = []
        for i in zip( observed, expected )
            append!(pearson_res, [(observe .- i[2])^2 / i[2] for observe in i[1]])
        end
        return sum(pearson_res)
    end
    chi2obsv = chi2(values, expected)
    function permutation_box( data )
        sample_click = [sum(py"samp"(sort(data))), sum(py"samp"(sort(data))), sum(py"samp"(sort(data)))]
        sample_noclick = [(1000 - i) for i in sample_click]
        return chi2([sample_click, sample_noclick], expected)
    end
    perm_chi2 = [permutation_box( data ) for _ in 1:2000]
    resample_p = sum(perm_chi2 .> chi2obsv) / length(perm_chi2)
    return [chi2obsv, resample_p]
end

py"""
import random
import numpy as np
import pandas as pd
import statsmodels.api as sm
import statsmodels.formula.api as api
def test(diff, diff_mean):
    return np.mean(diff > diff_mean)
def samp(x):
    return random.sample( sorted(x), 1000)
def anova(df, ty, te, frm):
    return sm.stats.anova_lm(api.ols(frm, data=df).fit(), typ=ty, test=te)
"""
```

0.2.1. Plots

In [33]:

```
function plot_linear(a, b, d, r1, r2)
    fig, ax = plt.subplots( figsize=(6, 5) );

    x = np.arange( r1, r2, 1 )
    y = a * x .+ b + np.random.normal( 0, d, length(x) );
    r = (length(x)sum(x .* y) - (sum(x)sum(y))) / (sqrt((length(x)sum(x.^2) - (sum(x))^2) * (length(y)sum(y.^2)

    ax = sns.regplot(x, y, color="r")
    if r >= 0
        ax.set_title(("Correlação Positiva: " * string(round(r, digits=4))))
    else
        ax.set_title(("Correlação Negativa: " * string(round(r, digits=4))))
    end
end;

function spearman_plot( size, power )
    log_a = [log1p(abs(j-10)) for j in 1:size]
    log_b = [log1p(j)^power for j in 1:size]
    cor, _ = ss.spearmanr( log_a, log_b );

    fig, ax = plt.subplots( figsize=(5, 5) )
    ax.plot( log_a, color="b", linestyle="--", label="Log - 6" )
    ax.plot( log_b, color="r", linestyle="--", label="Negative Log" )
    ax.set_title( "Correlação: " * string(round(cor, digits=4)) )
    plt.legend();
end;

function plot_density( x, y )
    fig, (ax1, ax2) = plt.subplots( 1, 2, figsize=(16, 7) )
    ax1.hexbin( np.random.randn(x), np.random.randn(y), gridsize=30, cmap="gist_heat" );
    ax2 = sns.kdeplot(np.random.randn(x), np.random.randn(y) );
end;
```

```
function plot_bootstrap( x, g_type, c )
    plt.hist( x, color=c, linewidth=2, histtype=g_type, bins=20 );
    plt.vlines( np.mean( x ), ymin=n_size-100, ymax=1., color="k", linestyle="--", label="Mean" );
    plt.legend();
end;

function plot_three_bn( prob, colors, label )
    for i in zip( [0.1, 0.5, 0.9], ["r", "g", "b"], ["Prob: 10%", "Prob: 50%", "Prob: 90%"] )
        res = [ss.binom.pmf(r, 10, i[1]) for r in 0:10]
        plt.bar(Array(1:11), res, color=i[2], label=i[3]);
        plt.legend()
    end
end;

function plot_normal_plus( x, x2, x3, x4, x5 )
    fig, ax = plt.subplots( figsize=(7, 4) )
    ax.plot(x, ss.norm.pdf(x), color="b", linewidth=2, label="Normal");
    for j in zip( [x2, x3, x4, x5], ["r", "b", "m", "c"])
        ax.plot(j[1], ss.norm.pdf(x2), color=j[2], linewidth=2 );
    end
    ax.hlines( .003, -4, 4, color="k")
    ax.vlines( 0, 0, 0.4, color="k", linestyle="--", label="Média")
    ax.set_title("Distribuição Normal")
    fig.legend();
end;

function plot_poisson( x, c )
    fig, ax = plt.subplots( figsize=(7, 4) )
    ax = sns.distplot(x, color=c, hist=false, label="Poisson");
    ax.vlines( np.mean(x), 0, 0.27, color="k", linestyle="--", label="Média" )
    ax.vlines( np.median(x), 0, 0.27, color="c", linestyle="--", label="Mediana" )
    ax.vlines( 0, -0.05, 0.28, color="#12004f", linewidth=2, linestyle="-")
    ax.hlines( -0.001, -1.7, 8.5, color="#12004f", linewidth=2, linestyle="-")
    ax.set_xlabel("Contagem");
    ax.set_ylabel("Densidade");
    ax.set_title("Distribuição Poisson")
    ax.legend();
end;

function plot_normal( x, sim=true )
    fig, ax = plt.subplots( figsize=(7, 4) )
    ax.plot(x, ss.norm.pdf(x), color="b", linewidth=2, label="Normal");
    ax.hlines( .003, -4, 4, color="k")
    ax.vlines( 0, 0, 0.4, color="k", linestyle="--", label="Média")
    ax.set_title("Distribuição Normal")
    fig.legend();
    if sim == true
        [ax.vlines(  i, 0, 0.24, color="r") for i in -1:1 if i != 0]
        [ax.vlines(  i, 0, 0.055, color="r") for i in -2:2 if i != 0]
        [ax.vlines( -i, 0, (0.50/(i*2.0)-.01), color="r", linestyle="--" ) for i in 1:.1:1.3];
        [ax.vlines( -i, 0, (0.50/(i*2.3)-.01), color="r", linestyle="--" ) for i in 1.3:.1:1.6];
        [ax.vlines( -i, 0, (0.50/(i*3)-.01), color="r", linestyle="--" )   for i in 1.6:.1:1.9];
        [ax.vlines(  i, 0, (0.50/(i*2.0)-.01), color="r", linestyle="--" ) for i in 1:.1:1.3];
        [ax.vlines(  i, 0, (0.50/(i*2.3)-.01), color="r", linestyle="--" ) for i in 1.3:.1:1.6];
        [ax.vlines(  i, 0, (0.50/(i*3)-.01), color="r", linestyle="--" )   for i in 1.6:.1:1.9];
    else
    end
end;

function plot_qq(x, y, z, g_type)
    fig, (ax1, ax2, ax3, ax4, ax5, ax6) = plt.subplots( 2, 3, figsize=(16, 7) )
    ss.probplot( x, dist="norm", plot=ax1 );
    ss.probplot( y, dist="norm", plot=ax5 );
    ss.probplot( z, dist="norm", plot=ax3 );
    for i in zip( [ax2, ax6, ax4], [x, y, z], ["Normal", "Log", "Skewed"], ["r", "g", "b"] )
        i[1].hist( i[2], histtype=g_type, linewidth=2, color=i[4], label=i[3] );
        i[1].legend()
        i[1].set_ylabel("Density")
        i[1].set_xlabel("Bins")
        plt.show()
    end;
end;

function plot_exp( x, c )
    fig, ax = plt.subplots( figsize=(7, 4) )
    ax = sns.distplot(x, color=c, hist=false, label="Exponencial");
    ax.set_xlabel("Contagem");
    ax.vlines( np.mean(x), 0, 0.4, color="k", linestyle="--", label="Média" )
    ax.vlines( np.median(x), 0, 0.55, color="c", linestyle="--", label="Mediana" )
    ax.vlines( 0, 0, 0.7, color="#12004f", linestyle="-")
    ax.vlines( 0, -0.05, 0.28, color="#12004f", linewidth=2, linestyle="-")
    ax.hlines( -0.001, -1.7, 8.5, color="#12004f", linewidth=2, linestyle="-")
    ax.set_title("Distribuição Exponencial")
    ax.legend();
end;
```

```
function plot_wei(x)
    fig, ax = plt.subplots( figsize=(7, 4) )
    ax = sns.distplot( x, color=c, label="Weibull");
    ax.hlines( -.1/(1000*100), 150*150, -200, color="#12004f", linewidth=2, linestyle="-")
    ax.vlines( -.1/(1000*100), -.1/(500*10), .1/(500), color="#12004f", linewidth=2, linestyle="-")
    ax.set_title("Distribuição Weibull")
    ax.legend();
end;

function plot_permutation(x, d, c1, c2, ex, with_ex=false)
    fig, ax = plt.subplots( figsize=(6, 5))
    ax.hist(x, histtype="step", color="r", linewidth=2)
    ax.axvline( x=d, c=c1, lw=1, ls="--", label="Diferença" );
    if with_ex == true
        ax.vlines( ex, 0, 100, color=c2 );
        plt.legend();
    else
        plt.legend();
    end
end;

function plot_page_diff()
    fig, ax = plt.subplots( figsize=(7, 5) )
    ax.hist( df[df.Page .== "Page A", 2], histtype="step", linewidth=2, color="r", label="Page A")
    ax.hist( df[df.Page .== "Page B", 2], histtype="step", linewidth=2, color="b", label="Page B")
    ax.hlines( y=2.0, xmin=-0, xmax=4., linestyle="--", color="k" )
    ax.set_xlabel("Time")
    ax.set_ylabel("Count")
    ax.legend();
end;
```

Capítulo 1

1.1. Correlação

1.1.1. R de Pearson

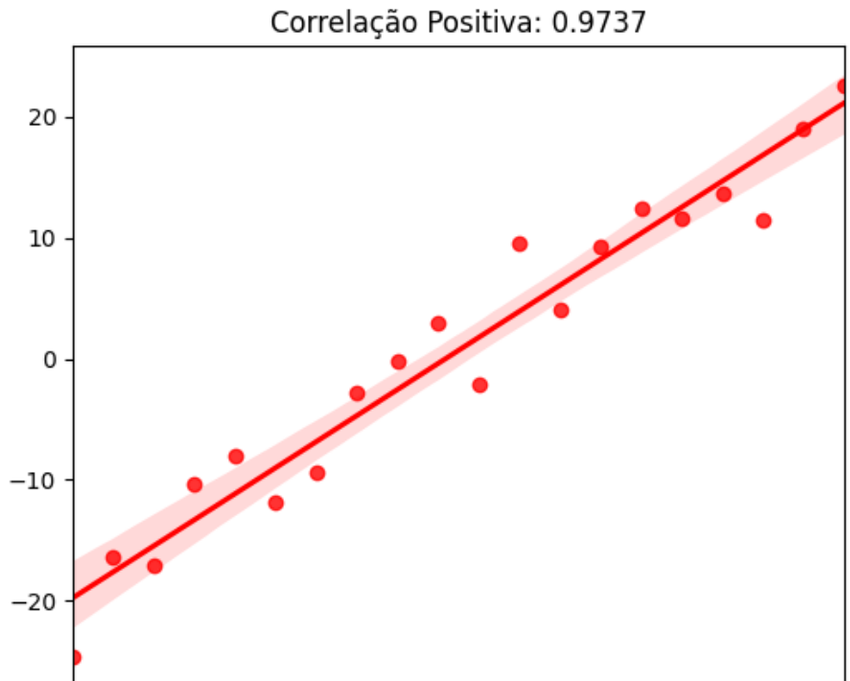
O coeficiente de correlação de pearson é muitas vezes o primeiro coeficiente estudado ou abordado em livros.
São ditos os dados que são positivamente correlacionados quando os valores de **x** acompanham os valores de **y** e negativamente correlacionados se os valores altos de **x** acompanharem os valores baixos de **y**.
Causalidade a variável **x** é a causa da variável **y**, logo por exemplo a correlação entre número de vendas e clientes é positiva, mas não quer dizer que quantos mais clientes existem mais vendas eu tenha.
Ex: O número de consumo de **margarina** e o número de **divórcios** em Maine.

Fórmula do coeficiente de pearson.

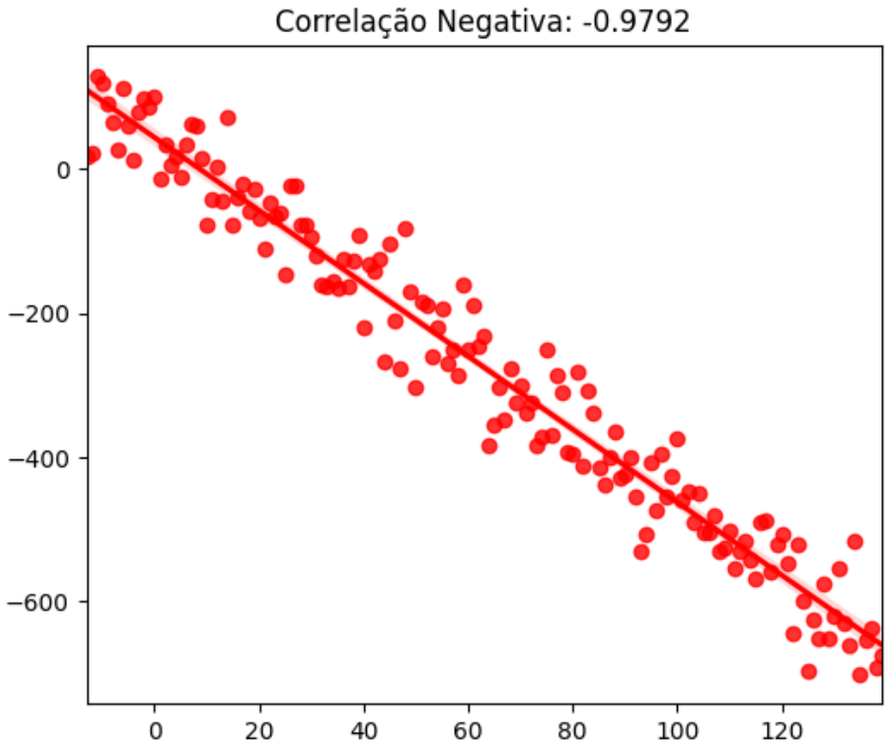
$$r = \frac{n(\sum xy) - (\sum x)(\sum y))}{\sqrt{(n\sum(x^2) - (\sum x)^2)\,(n\sum(y^2) - (\sum y)^2)}}$$

In [191…

```
ax1 = plot_linear(2, 2, 3, -10, 10);
```



In [8]:
```
plot_linear(-5, 40, 50, -13, 140);
```

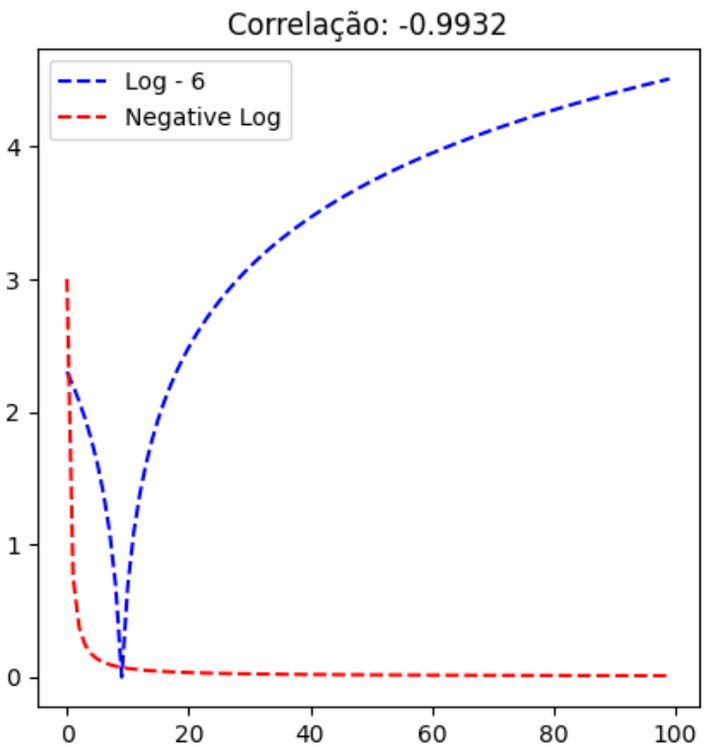


1.1.2. Rho de spearman

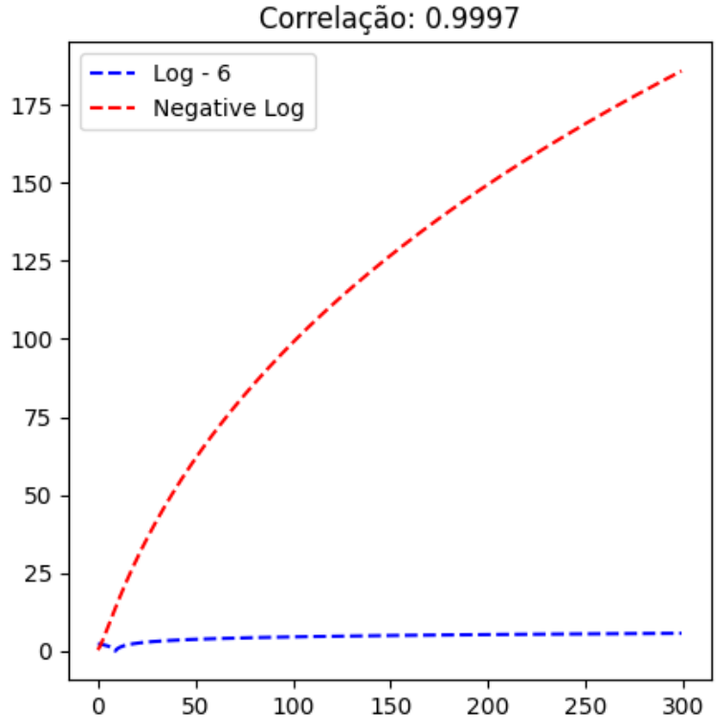
Robusto contra outliers e calculado em relação ao ranqueamento ou ordens dos dados, também mede relações lineares e não lineares.

$$r_s = 1 - \frac{6 \sum d^2}{n^3 - n}$$

In [96]:
```
spearman_plot( 100, -3 );
```



In [97]:
```
spearman_plot( 300, 3 );
```



1.1.4. V de Cramér

O V de cramér basicamente serve para calcular a correlação entre variaveis categoricas.

Existe a versão corrigida da fórmula de cramér que esta abaixo, k e r são as dimensões da matriz.

$$V = \sqrt{\frac{\varphi^2 \; ou \; X^2/n}{min(k-1, r-1)}}$$

$$\varphi^2 = max(0, \varphi^2 - \frac{(k-1)-(r-1)}{n-1}$$

$$cor \; k = k - \frac{(k-1)^2}{n-1}$$

$$cor \; r = r - \frac{(r-1)^2}{n-1}$$

In [226…

```
df = DataFrame(CSV.File("data/store.csv"))
df = hcat(df[:, 2:3], df.StoreType )
df = rename(df, Dict("x1" => "State"));

# Rename Rows
df.Assortment = [replace(i, "a" => "BASIC") for i in df.Assortment];
df.Assortment = [replace(i, "b" => "EXTRA") for i in df.Assortment];
df.Assortment = [replace(i, "c" => "EXTENDED") for i in df.Assortment];

# Generate FataFrame
results = []
data = DataFrame()

for i in ["StoreType", "Assortment", "State"]
    a = cramer_v( Array(df.StoreType), Array(df[:, i]) )
    b = cramer_v( Array(df.Assortment), Array(df[:, i]) )
    c = cramer_v( Array(df.State), Array(df[:, i]) )

    corr = Dict(i => [a, b, c])
    append!(results, corr)
end

df2 = DataFrame( results ) # Plotar um Mapa de Calor / Heatmap
```

Out[226…

3 rows × 3 columns

	StoreType	Assortment	State
	Float64	Float64	Float64
1	1.00135	0.54068	1.00135

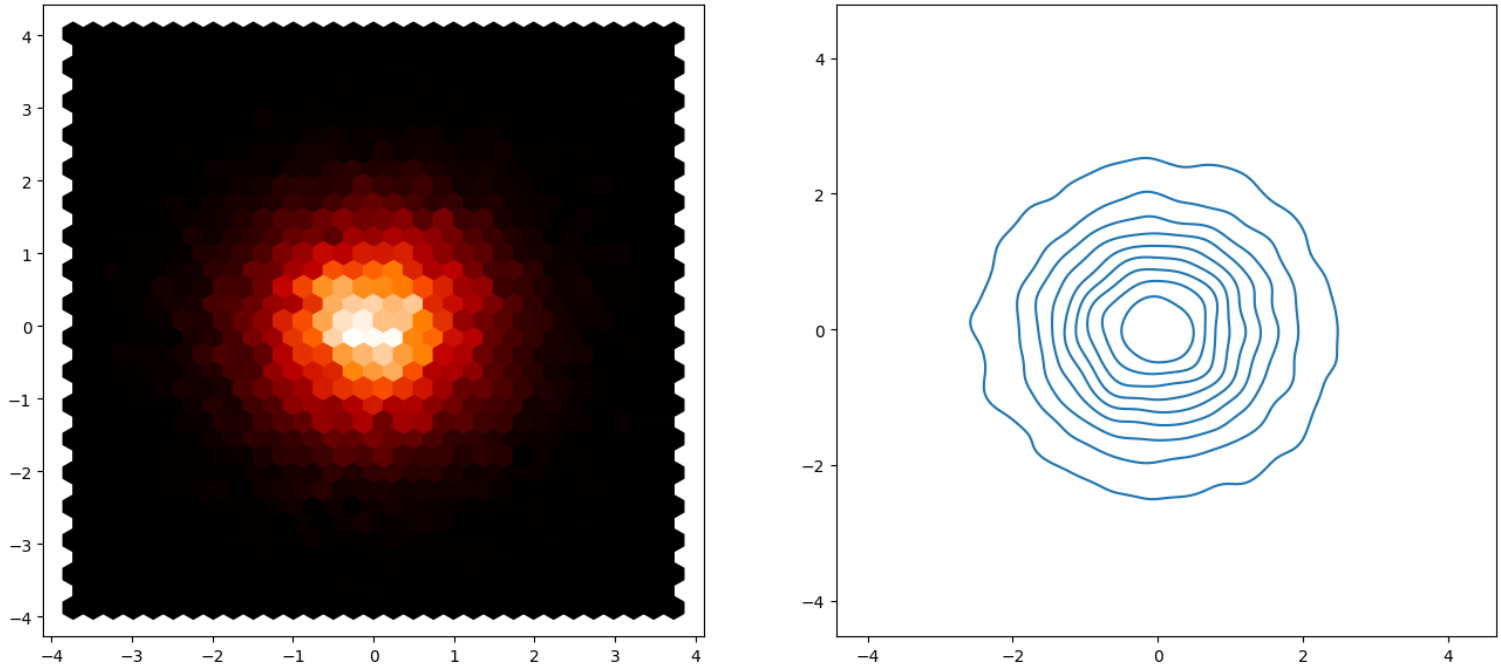
2	0.54068	1.0009	0.54068
3	1.00135	0.54068	1.00135

1.2. Dois Gráficos de Densidade

Hexagonal Binning relaciona as duas variaveis aleatorias normais em hexágonos, mesma coisa que o Histograma.
Kernel Density Estimate, Análogo análogo ao Hexagonal, porem em densidades com curvas.

In [160…

```
plot_density( 20000, 20000 )
plt.savefig("Density.png")
```



Capítulo 2

2.0. Distribuição de Amostragem de uma Estatística

A distribuição de uma estatística amostral como a média costuma ser mais regular e campanular do que a distribuição dos proprios dados quanto maior a amostra em que a estatística se baseia. Além disso, quanto maior a amostra, mais estreita é a distribuição da estatística amostral. "Tende a Normal"...

In [15]:

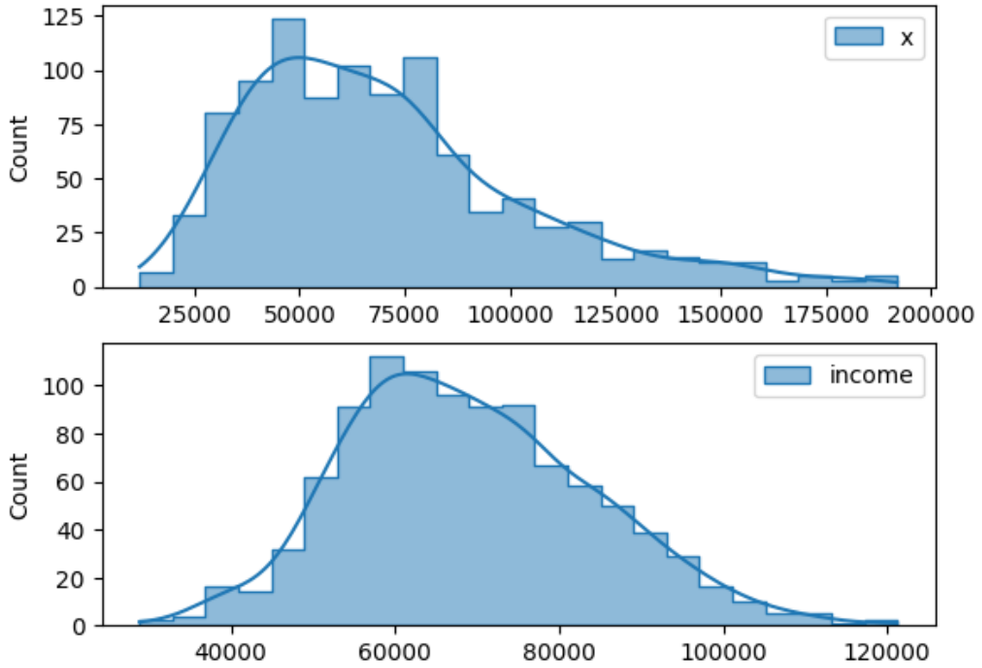
```
df = pd.read_csv("data/loans_income.csv");
```

In [16]:

```
sample = df.sample(1000)

sample_of_5 = pd.DataFrame( Dict("income" => [df["x"].sample(5).mean() for _ in 1:1000],
                                 "type" => "mean_of_5") )

sample_of_10 = pd.DataFrame( Dict("income" => [df["x"].sample(10).mean() for _ in 1:1000],
                                  "type" => "mean_of_10") )

sample_of_20 = pd.DataFrame( Dict("income" => [df["x"].sample(20).mean() for _ in 1:1000],
                                  "type" => "mean_of_20") )

result = pd.concat([sample, sample_of_5, sample_of_10, sample_of_20]);
```

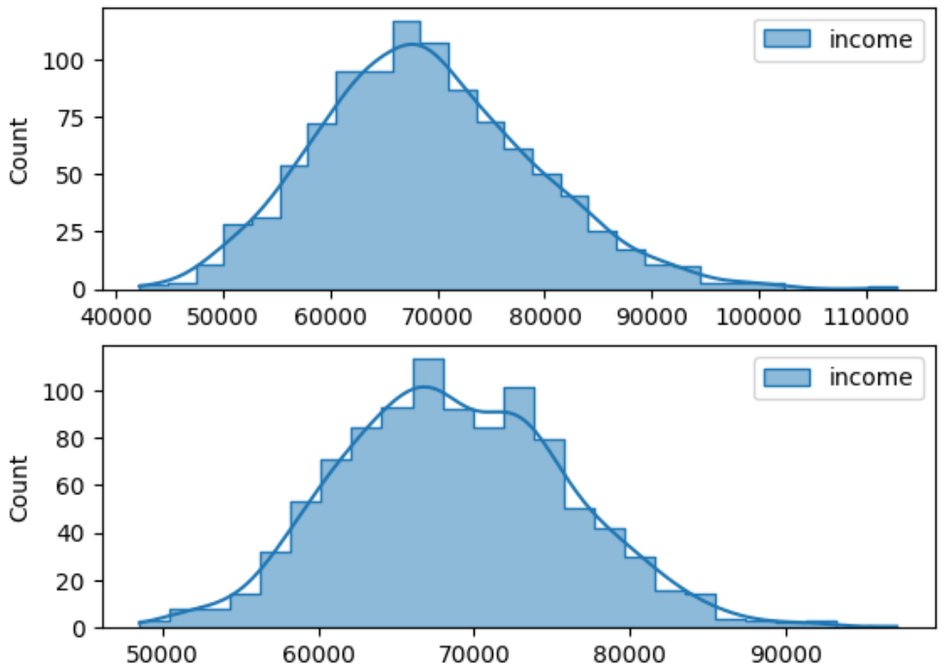
In [21]:

```
plt.subplot(2, 1, 1);
sns.histplot( sample, kde=true, element="step" );

plt.subplot(2, 1, 2);
sns.histplot( sample_of_5, kde=true, element="step" );
```



In [20]:

```
plt.subplot(2, 1, 1);
sns.histplot( sample_of_10, kde=true, element="step" );

plt.subplot(2, 1, 2);
sns.histplot( sample_of_20, kde=true, element="step" );
```



2.2. O Bootstrap

O bootstrap é uma forma eficiente e eficaz de estimar a distribuição amostral de uma estatística ou de parâmetros de modelo.
Conceitualmente pode-se imaginar o Bootstrap como uma replicação da amostra original várias vezes de modo a ter uma população hipotética que representa todo o conhecimento da amostra original só que maior.
Logo amostramos com reposição, dessa forma cria-se efetivamente uma população infinita na qual a probabilidade de um elemento ser extraído continua a mesma de extração por extração.
Com os resultados é possível encontrar um intervalo de confiança.

2.2.1. Sem Bootstrap

In [24]:

```
# Load and Prepare Dataset
df = DataFrame(CSV.File("data/diabetes.csv"));

x = df[:, 1:8];
x = hcat(x[:, 1], x[:, 2], x[:, 3], x[:, 4], x[:, 5], x[:, 6], x[:, 7]);

y = df[:, 9];

# Transform Variables
```

```
mms = prepro.MinMaxScaler()
x = mms.fit_transform( x );

# Split Dataset
x_train, x_test, y_train, y_test = model_selection.train_test_split( x, y, train_size=0.9 );
```

In [21]:

```
# Simple Logistic Regression
model = models.Sequential()
model.add( layers.Dense( 1, activation="sigmoid") )
model.compile( optimizer="sgd", loss="binary_crossentropy", metrics=["accuracy"])
```

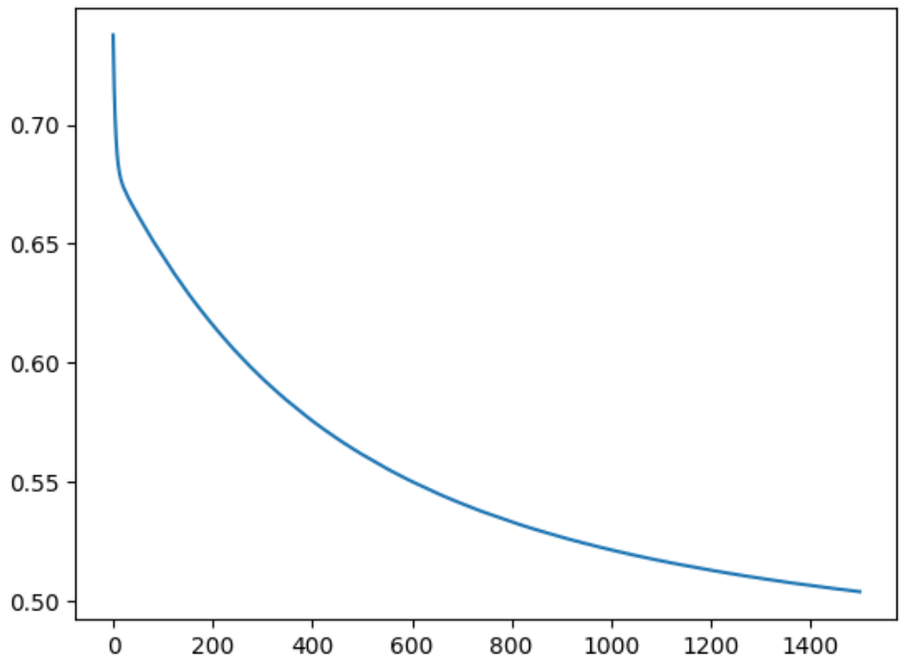
In [22]:

```
history = model.fit( x_train, y_train, epochs=2000, verbose=0 );
test = model.evaluate( x_test, y_test );
```

```
3/3 [==============================] - 0s 2ms/step - loss: 0.5010 - accuracy: 0.7143
```

In [117…

```
PyPlot.plot( history.history["loss"] );
```



Conclusão

In [116…

```
@printf "Accuracy %.2f%%" maximum(history.history["accuracy"])*100
```

```
Accuracy if 77.71%
```

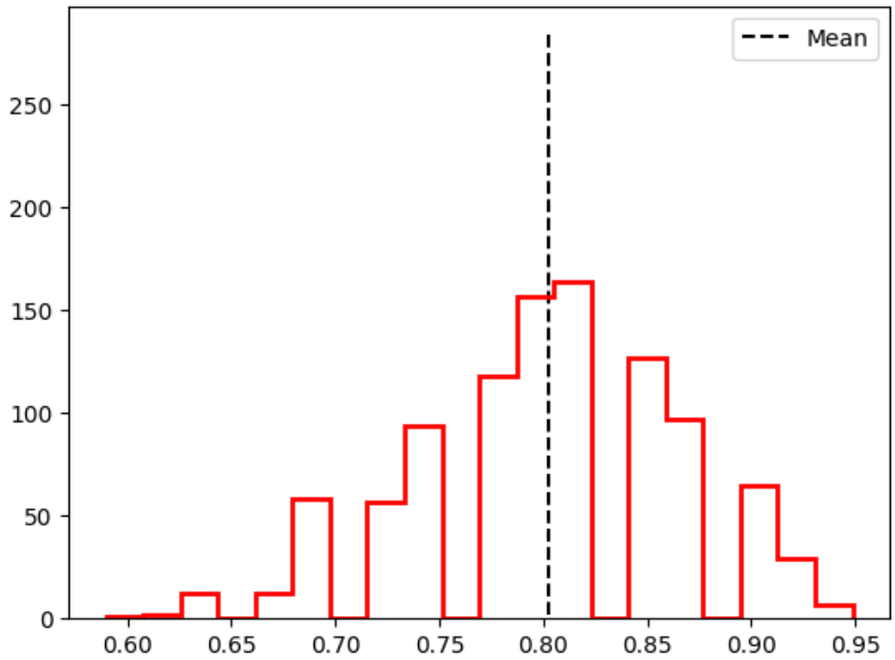
2.2.1. Com Bootstrap

In [25]:

```
# Configuration of Bootstrap
n_inter = 1000
n_size  = trunc(Int, (768 * 0.5))
stats   = []

# Set Up Data
df = hcat(df[:, 1], df[:, 2], df[:, 3], df[:, 4], df[:, 5], df[:, 6], df[:, 7], df[:, 8], df[:, 9])

for i in 1:n_inter
    sample = utils.resample( df, n_samples=n_size );
    y = sample[:, 9];
    x = sample[:, 1:8];

    x = mms.fit_transform( x );

    # Split Dataset
    x_train, x_test, y_train, y_test = model_selection.train_test_split( x, y, train_size=0.9 );

    model = tree.DecisionTreeClassifier() # Decision Tree, dont NN.
    model.fit( x_train, y_train )

    prediction = model.predict( x_test )
    score = metrics.accuracy_score( y_test, prediction )

    append!(stats, score)
end
```

In [106…

```
plot_bootstrap( stats, "step", "r" )
```



Conclusão

In [110…

```
α = 0.95

p = ((1.0 - α)/2.0)*100
lower = np.percentile(stats, p)*100

p = (α + (1.0 - α)/2.0) *100
max = np.percentile(stats, p)*100

@printf "%.0f%% Confidence Intervals %.2f%% and %.2f%%" α*100 lower max
```

```
95% Confidence Intervals 66.67% and 92.31%
```

2.3. Distribuições

2.3.1 Distribuição Binomial

A distribuição Binomial é, vamos dizer assim uma continuação da distribuição de Bernoulli, onde a distribuição de Bernoulli trabalha somente com duas possibilidades, ou 1 geralmente chamado de evento de sucesso ou 0 de fracasso contendo as probabilidades desses eventos ao lado.

In [17]:

```
bernoulli = pd.DataFrame([["Azul", 0.35], ["Vermelho", 0.65]]);
bernoulli.columns = ["X", "Probabilidade"];
bernoulli
```

Out[17]:

	X	Probabilidade
0	Azul	0.35
1	Vermelho	0.65

Formula da Distribuição Binomial

$$P(r/n,p) = \binom{n}{r} \cdot p^r \cdot (1-p)^{n-r}, onde \binom{n}{r} = C_{nk} = \frac{n!}{r! \cdot (n-r)!}$$

- r: Eventos de Interesse.
- n: Repetições.
- p: Probabilidade do Evento.

A média da distribuição binomial é dada pela formula: $n \cdot p$
A variância da distribuição binomial é dada pela formula: $\sqrt{n \cdot p \cdot (1-p)}$

Na formula da distribuição Binomial também pode se trabalhar com um conceito chamado de "Complementar", que nada mais é que uma forma mais rápida para calcular a binomial quando se tem um intervalo de eventos maior que um.
$P(x >= k) = 1 - P(x < k)$

$P(x > k) \ = 1 - P(x <= k)$
$P(x <= k) = 1 - P(x > k)$
$P(x < k) \ = 1 - P(x >= k)$

Por exemplo se você for calcular um **n = 6** para uma possibilidade de, digamos **P(x >= 2)**, você teria que calcular 5 vezes com a fórmula, logo
$P(x = 2) + P(x = 3) + P(x = 4) + P(x = 5) + P(x = 6)$

Porém, aplicando a regrazinha, inverte-se, logo:
$1 - (P(x = 0) + P(x = 1))$

In [16]:
```
P(r, n, p) = (factorial(n) / (factorial(r) * factorial(n-r))) * (p^r) * ((1 - p)^(n-r));
```

Qual é a probabilidade de sair 3 caras em 4 jogadas e a probabilidade de cada cara é $\frac{1}{2}$

$$P(2/4, \frac{1}{2}) = 6 \cdot (\frac{1}{2})^2 \cdot (\frac{1}{2})^2 = \frac{6}{16}$$

In [18]:
```
@printf "A probabilidade de sair três caras em quatro jogadas é: %.3f%% " P(2, 4, .5)*100
```

```
A probabilidade de sair três caras em quatro jogadas é: 37.500%
```

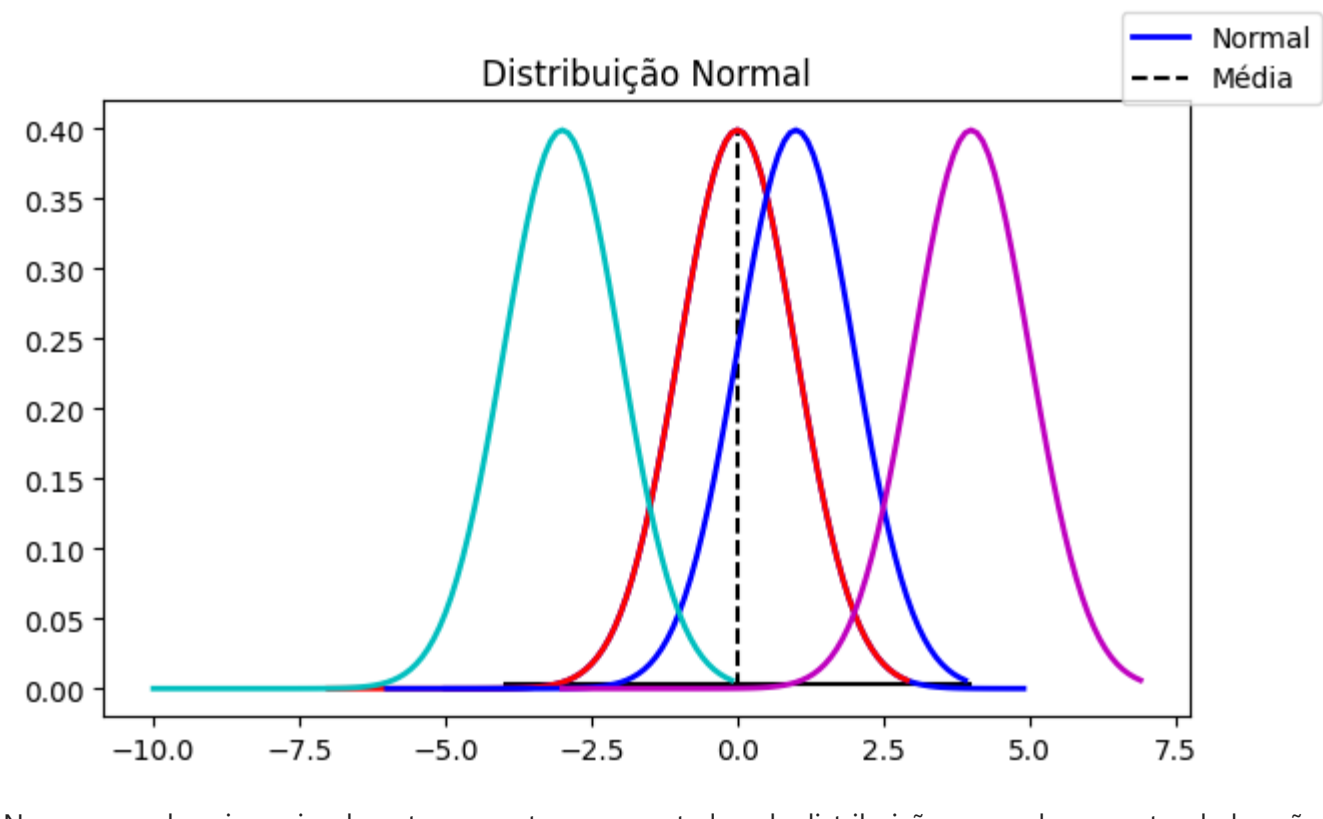
Qual é a chance de 3 pessoas que eu ligar das 10 entrarem em churn sabendo que a probabilidade de uma pessoa em churn na base de dados é 0.15%?

In [19]:
```
@printf "A probabilidade de 3 das 10 pessoas ligadas entrarem em churn é: %.3f%%" P(3, 10, .15)*100
```

```
A probabilidade de 3 das 10 pessoas ligadas entrarem em churn é: 12.983%
```

Confome a probabilidade tende a o equilibrio, ou seja, .5% de cair cara ou coroa, logo a distribuição parece uma Normal.

In [194…
```
plot_three_bn( [0.1, 0.5, 0.9], ["r", "g", "b"], ["Prob: 10%", "Prob: 50%", "Prob: 90%"] )
plt.savefig("Binomial.png")
```

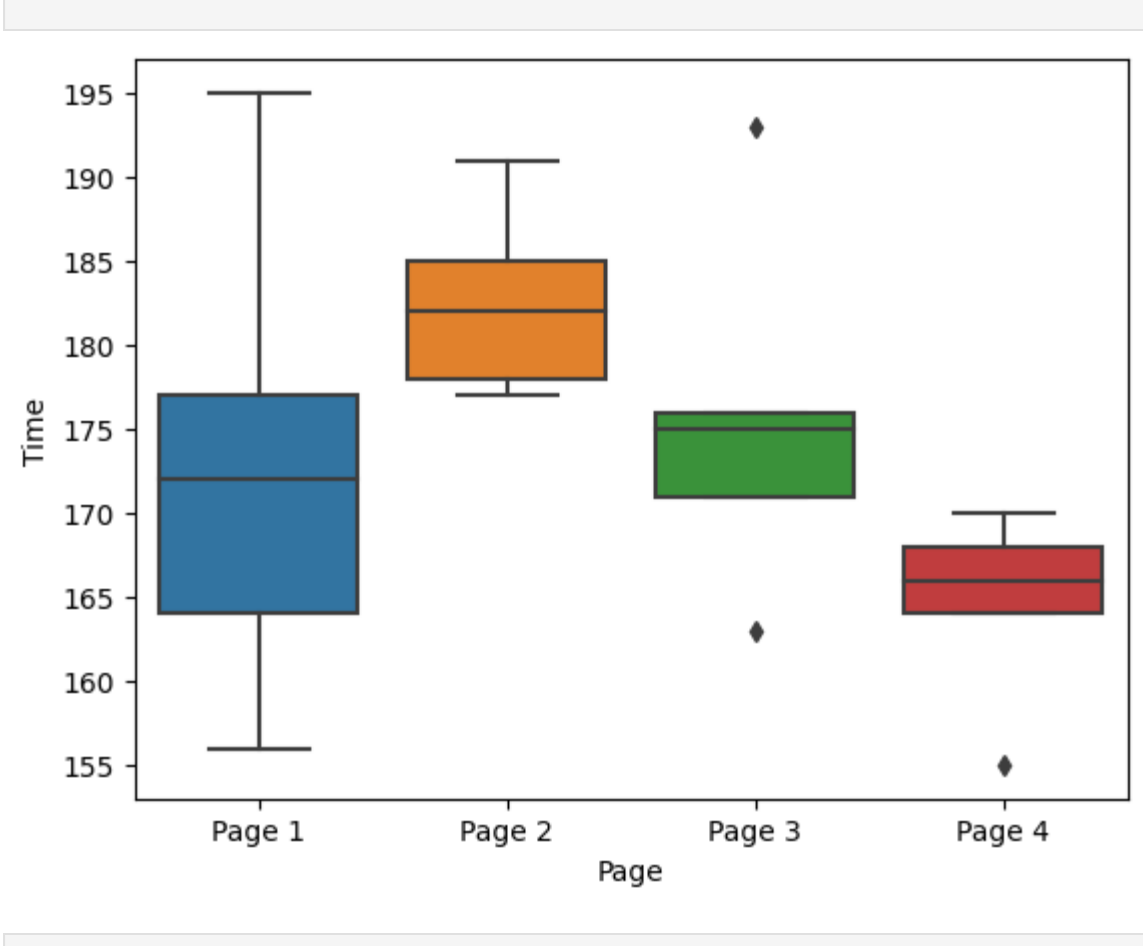


Nesse exemplo acima, visualmente a amostra representada pela distribuição em azul, parece ter dados não tão importantes quanto por exemplo a distribuição em azul marinho em relação a todos os dados.

H_o = **Não há diferenças estatisticamente significantes entre as médias dos grupos.**
H_a = **As médias dos grupos são diferentes.**
α = **0.05**

In [289…
```
df = DataFrame(CSV.File("data/four_sessions.csv"))
df1 = pd.read_csv("data/four_sessions.csv");
```

In [210…
```
sns.boxplot( df1.Page, df1.Time );
```



In [290…
```
f_stats, p_value = ss.f_oneway( df[df.Page .== "Page 1", 2], df[df.Page .== "Page 2", 2], df[df.Page .== "Page
@printf "Estatística F: %.4f\nP-Valor: %.4f" f_stats p_value
```

```
Estatística F: 2.7398
P-Valor: 0.0776
```

In [296…
```
f_stats, p_value = ss.kruskal( df[df.Page .== "Page 1", 2], df[df.Page .== "Page 2", 2], df[df.Page .== "Page 3
@printf "Estatística F: %.4f\nP-Valor: %.4f" f_stats p_value
```

```
Estatística F: 8.0893
P-Valor: 0.0442
```

No Statsmodels você desenha a matriz para o ordinary least-square regressor com as formulas.

- Nesse exemplo é um teste unicaldal com o mesmo p valor do scipy e so funciona com um dataframe do pandas.

In [293…
```
py"anova"(df1, 2, "F", "Time ~ Page")
```

Out[293…

	sum_sq	df	F	PR(>F)
Page	831.4	3.0	2.739825	0.077586
Residual	1618.4	16.0	NaN	NaN

Ok, você fez todos esses testes, mas qual dos grupos realmente é diferente logo que o p-valor foi pequeno?

In [362…
```
list = [df[df.Page .== "Page 1", 2], df[df.Page .== "Page 2", 2], df[df.Page .== "Page 3", 2], df[df.Page .== '
pg = ["Page 1", "Page 2", "Page 3", "Page 4"];
```

In [369…
```
for i in zip( pg, list ), k in 1:4
    @printf "\n%s - %s\n" i[1] pg[k]
    println(ss.ttest_ind(i[2], list[k]))
end
```

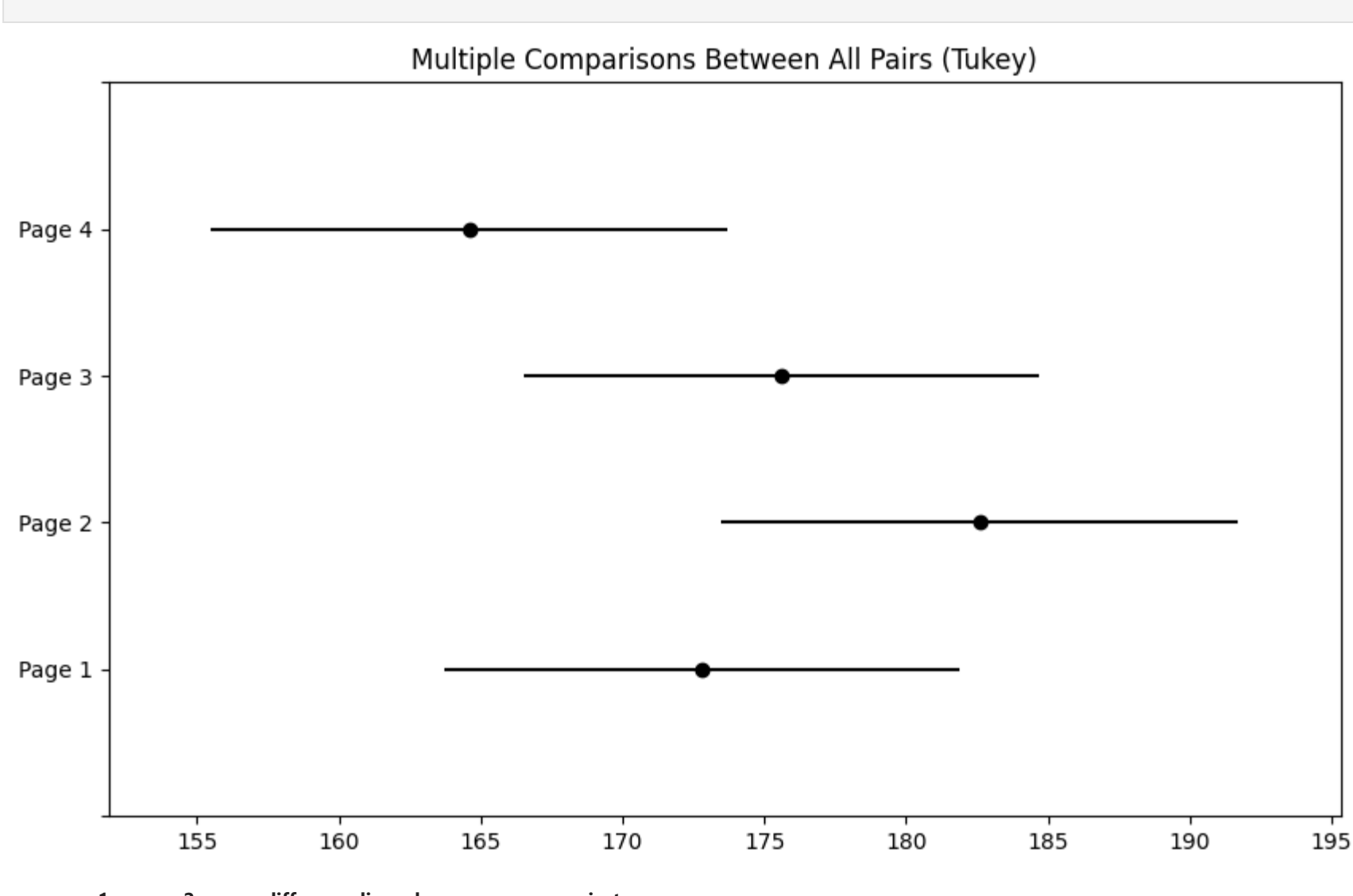
```
Page 1 - Page 1
(0.0, 1.0)

Page 1 - Page 2
(-1.3859292911256307, 0.20317499853824886)

Page 1 - Page 3
(-0.34030136960725227, 0.7423928129048489)

Page 1 - Page 4
(1.156191733258663, 0.28096106091142226)

Page 2 - Page 1
(1.3859292911256307, 0.20317499853824886)

Page 2 - Page 2
(0.0, 1.0)

Page 2 - Page 3
(1.2650143347589609, 0.24146881279653873)

Page 2 - Page 4
(4.950587917224828, 0.001120056434441126)

Page 3 - Page 1
(0.34030136960725227, 0.7423928129048489)

Page 3 - Page 2
(-1.2650143347589609, 0.24146881279653873)

Page 3 - Page 3
(0.0, 1.0)

Page 3 - Page 4
(1.9782125034804172, 0.08327876770509358)

Page 4 - Page 1
(-1.156191733258663, 0.28096106091142226)

Page 4 - Page 2
(-4.950587917224828, 0.001120056434441126)

Page 4 - Page 3
(-1.9782125034804172, 0.08327876770509358)

Page 4 - Page 4
(0.0, 1.0)
```

Somente em relação a página 4 com a página 2 o p-value ficou pequeno.

3.4.1. Bonferroni & Sidak Correction

Quando se trabalha com a Anova ou multiplos testes t, geralmente podem superestimar a significância, ou seja, maior a chance de encontrar p valores pequenos, uma forma de corrigir esse problema é com a correção de bonferroni, que nada mais é que dividir o nível de significância pelo número unicos de comparações feitas, por exemplo $0.05 \div 6 = 0.0083$
Com a correção de Sidak, a formula é $1 - (1 - 0.05)^{(1/6)} = 0.0085$.
Outro comuntemte utilizado tambem é o Tukey's test

In [427…
```
py"tukey"(df1.Time, df1.Page, 0.05)
```



Out[427…

	group1	group2	meandiff	p-adj	lower	upper	reject
1	Page 1	Page 2	9.8	0.4393	-8.3999	27.9999	False
2	Page 1	Page 3	2.8	0.9	-15.3999	20.9999	False
3	Page 1	Page 4	-8.2	0.5758	-26.3999	9.9999	False
4	Page 2	Page 3	-7.0	0.6774	-25.1999	11.1999	False
5	Page 2	Page 4	-18.0	0.0531	-36.1999	0.1999	False
6	Page 3	Page 4	-11.0	0.3419	-29.1999	7.1999	False

x.0. Referências

PETER BRUCE & ANDREW BRUCE **Estatística prática para cientistas de dados: 50 conceitos essenciais**.
Link: https://www.amazon.com.br/Estat%C3%ADstica-Pr%C3%A1tica-Para-Cientistas-Dados/dp/855080603X

DAVID MATOS **8 Conceitos Estatísticos Fundamentais Para Data Science**.
Link: https://www.cienciaedados.com/8-conceitos-estatisticos-fundamentais-para-data-science/

IGOR SOARES **Correlação não implica em Causalidade**.
Link: https://medium.com/@felipemaiapolo/correla%C3%A7%C3%A3o-n%C3%A3o-implica-em-causalidade-8459179ad1bc.

annahaensch **Número de Casos de Divórcio em Maine**
Link: https://blogs.ams.org/blogonmathblogs/2017/04/10/divorce-and-margarine/

Wikipédia **Cramer's_V**
Link: https://en.wikipedia.org/wiki/Cram%C3%A9r%27s_V

BURKEYACADEMY **What are Skewness and Kurtosis?**
Link: https://www.youtube.com/watch?v=IK7nLzxiAQQ

(Discourse) **qqnorm & qqplot**
Link: https://discourse.julialang.org/t/qqnorm-and-qqplot/6118/8

Professor Guru **Tabela Normal Padrão**
Link: https://professorguru.com.br/tabela-normal.html

Univesp **Probabilidade e Estatística**
Link: https://www.youtube.com/watch?v=7VQE278hIXc&list=PLxI8Can9yAHeeWqe3m9HZFiBhT33Mfxew

Todd Grande **Bonferroni**
Link: https://www.youtube.com/watch?v=pbWkwEz-XBY

Zach **Kruskall Wallis**
Link: https://www.statology.org/kruskal-wallis-test-python/#:~:text=A%20Kruskal%2DWallis%20Test%20is,of%20the%20One%2DWay%20ANOVA.

USP **Teste Exato de Fisher**
Link: http://wiki.icmc.usp.br/images/7/73/Chisq-fisher.pdf