



# SEMANA DO PYTHON DA HASHTAG

## Apostila Completa Aula 3

Aprenda a criar modelos de previsão usando  
Ciência de Dados.  
Impressionador do absoluto zero!



Parte 1

# Introdução

# O que vamos aprender

Na terceira aula da Semana do Python você vai aprender a criar um código de ciência de dados para criação de um modelo de previsão do **absoluto zero**. Para isso, vamos passar por conceitos como:

Jupyter Notebook

Etapas de um  
projeto de Ciência  
de Dados

Gráficos usando  
Python

Uso de bibliotecas

Modelos de  
Inteligência Artificial

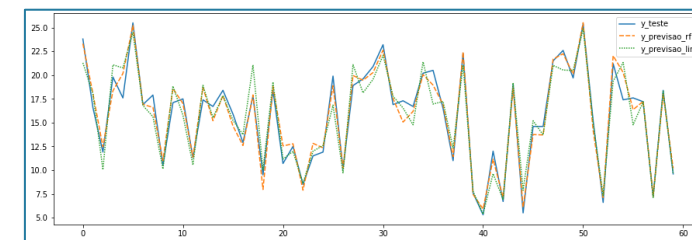
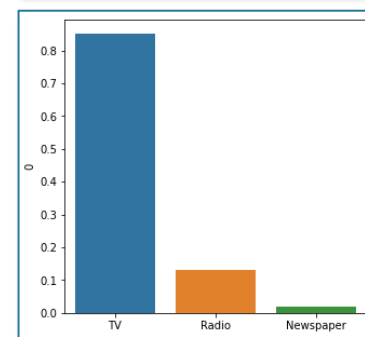
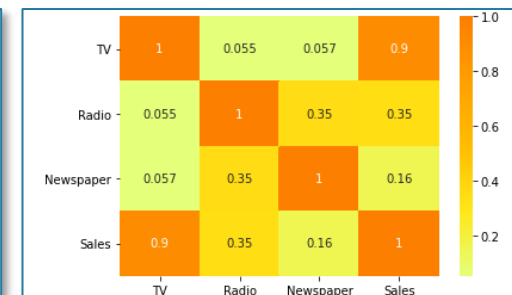
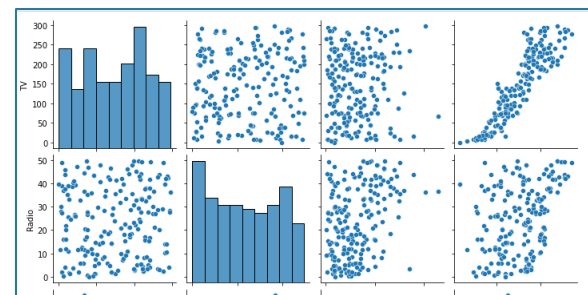
Após todos esses conhecimentos, seremos capazes de transformar uma tabela cheia de informações, nem um pouco fáceis de serem interpretadas ...

advertising.csv - Bloco de Notas

Arquivo	Editar	Formatar	Exibir	Ajuda
TV, Radio, Newspaper, Sales				
230.1, 37.8, 69.2, 22.1				
44.5, 39.3, 45.1, 10.4				
17.2, 45.9, 69.3, 12				
151.5, 41.3, 58.5, 16.5				
180.8, 10.8, 58.4, 17.9				
8.7, 48.9, 75, 7.2				
57.5, 32.8, 23.5, 11.8				
120.2, 19.6, 11.6, 13.2				
8.6, 2.1, 1, 4.8				
199.8, 2.6, 21.2, 15.6				
66.1, 5.8, 24.2, 12.6				
214.7, 24, 4, 17.4				
23.8, 35.1, 65.9, 9.2				



... em um modelo de previsão de Vendas utilizando Inteligência Artificial



# Entendendo a base de dados

As informações que vão alimentar o nosso código, serão dados dos investimentos em propaganda em diferentes canais e as vendas decorrentes desses investimentos.

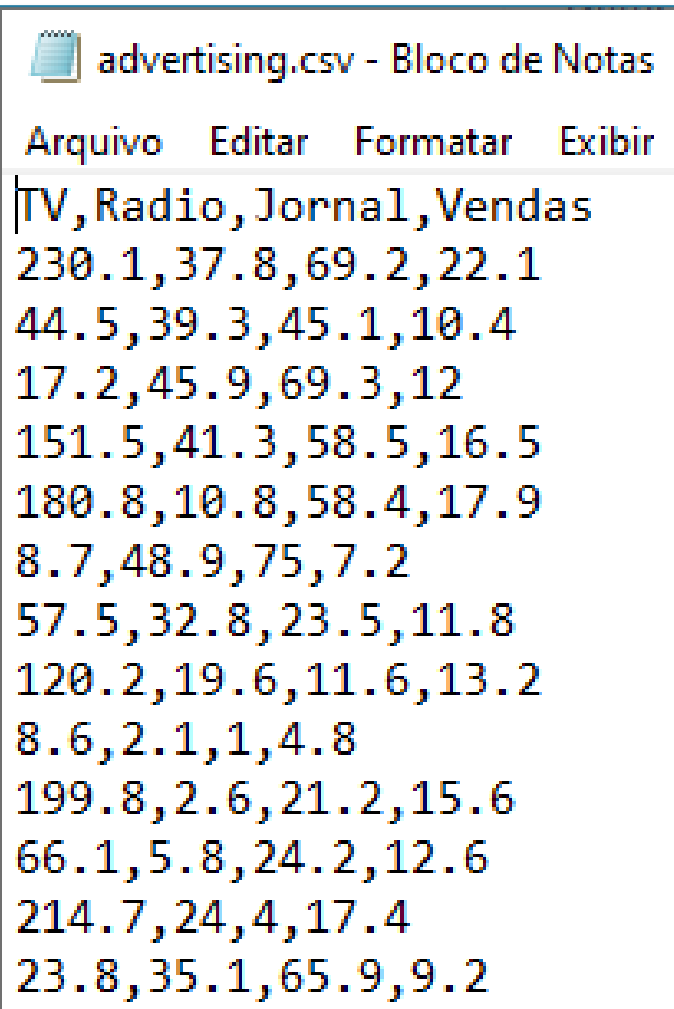
A imagem ao lado, mostra base “crua”. Apesar de não estar formatada de uma forma que nos ajuda a visualização, podemos perceber que existem 4 colunas (indicadas abaixo) separadas por vírgula:

✓ TV      ✓ Jornal      ✓ Radio      ✓ Vendas

Portanto, para um melhor entendimento, na primeira linha temos os 3 meios de comunicação que a Hashtag usa para vender (TV, Radio, Jornal) e o último item (Vendas) representa a quantidade em vendas resultante desses canais:

Usando a segunda linha como exemplo temos que investindo **230,1 kBRL no Canal TV**, **37,8 kBRL no Canal Radio**, **69,2 kBRL no Canal Jornal** foi gerada uma **Venda de 22,1 MBRL**.

Como você pode ver, temos muitas informações de vendas nessa tabela, e qualquer interpretação desses dados não é uma tarefa fácil, o que é um problema.



advertising.csv - Bloco de Notas			
Arquivo	Editar	Formatar	Exibir
TV	Radio	Jornal	Vendas
230.1	37.8	69.2	22.1
44.5	39.3	45.1	10.4
17.2	45.9	69.3	12
151.5	41.3	58.5	16.5
180.8	10.8	58.4	17.9
8.7	48.9	75	7.2
57.5	32.8	23.5	11.8
120.2	19.6	11.6	13.2
8.6	2.1	1	4.8
199.8	2.6	21.2	15.6
66.1	5.8	24.2	12.6
214.7	24	4	17.4
23.8	35.1	65.9	9.2

# Entendendo a solução final

Nosso objetivo é criar um modelo de previsão de vendas a partir do histórico de vendas e investimento em marketing.

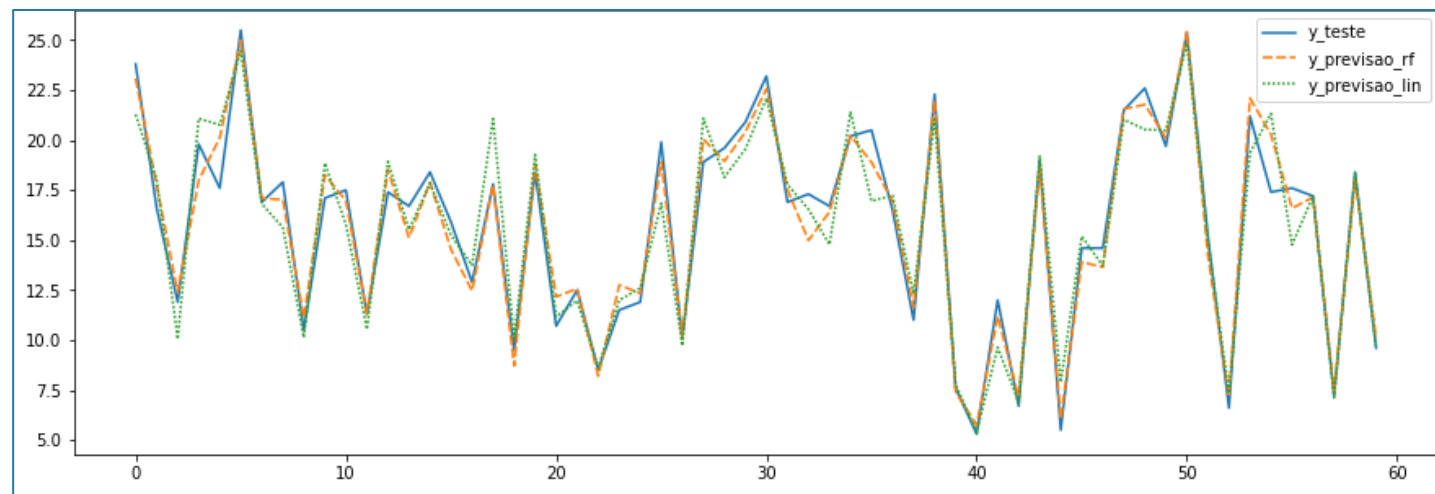
Para isso, usaremos alguns conhecimentos de Ciência de Dados.

Ou seja, baseada nesse modelo, poderemos projetar (com algum grau de incerteza) a quantidade em vendas baseada nos valores investidos.

Então, vamos começar!

### Projeto Ciência de Dados - Previsão de Vendas

- Nosso desafio é conseguir prever as vendas que vamos ter em determinado período com base nos gastos em anúncios nas 3 grandes redes que a empresa Hashtag investe: TV, Jornal e Rádio



Parte 2

# O que é um projeto de Ciência de Dados



# O que é um projeto de Ciência de dados

O conceito de Ciência de Dados cada dia passa a ser mais difundido nas empresas. Essencialmente, esse conceito, tem por objetivo gerar conclusões, Modelos, a partir de grande quantidade de dados.

Através de conhecimentos estatísticos alinhados ao conhecimento do negócio, a Ciência de dados permite “enxergar” além da superfície.

Para que isso aconteça, é muito interessante seguir alguns passos desse processos.

Preparamos um diagrama simples para você 😊



Parte 3

# Jupyter Notebooks



# O que é? Como acesso?

Os códigos em Python precisam de uma plataforma para serem escritos.

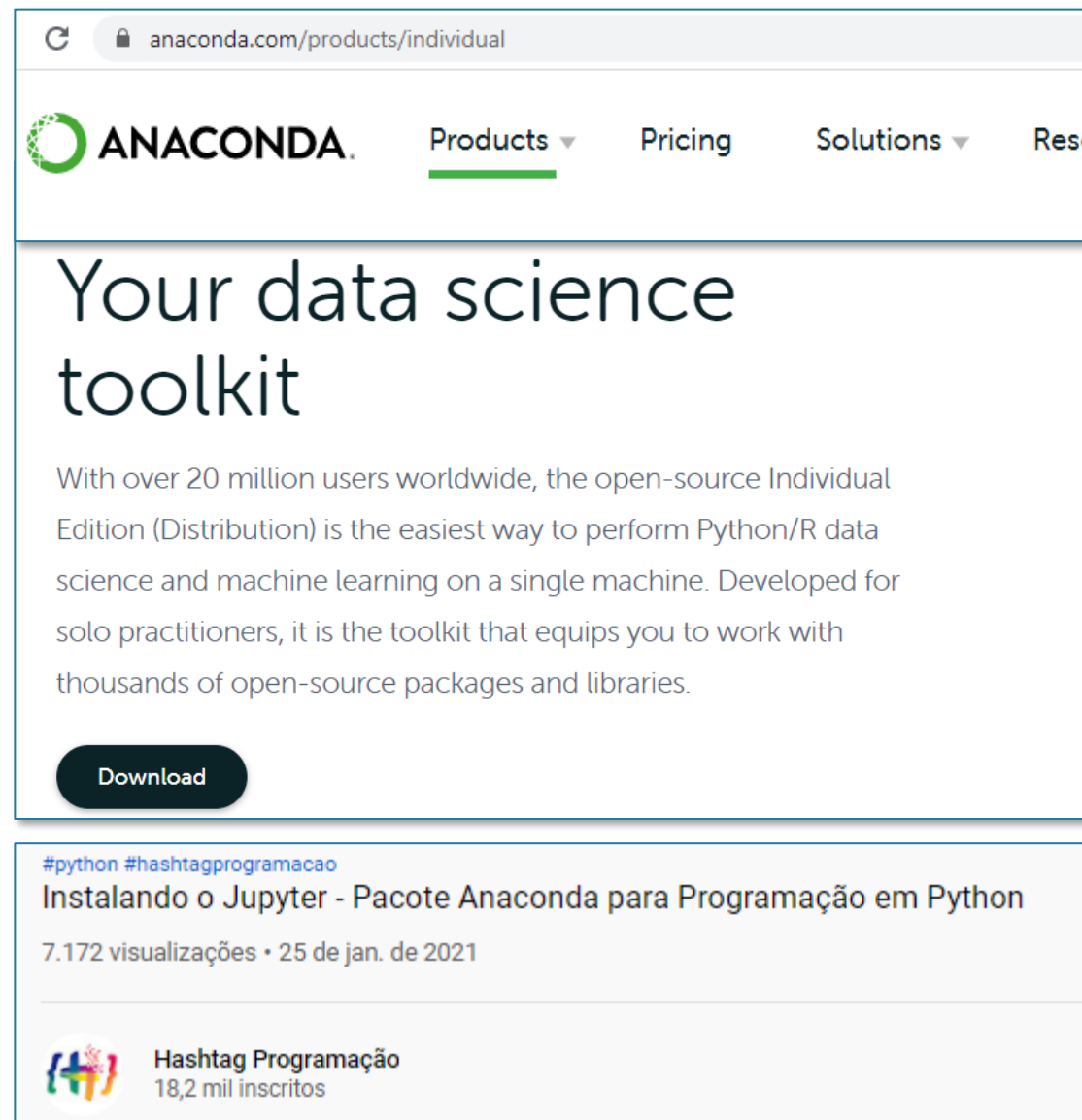
Essas plataformas na programação são chamadas de IDEs. Existem várias: Visual Studio, PyCharm, Atom, Google Colab, etc...

Todas podem ser utilizadas pra programação e execução dos códigos, usaremos o Jupyter Notebook que é uma ferramenta gratuita que existe dentro do Anaconda (uma espécie de grande caixa de ferramentas do Python).

Para usarmos o Jupyter Notebook, iremos instalar o Anaconda [\(link\)](#).

aAs próximas páginas desse capítulo são todos os passos para instalação correta do Jupyter mas caso você prefira, pode acessar nosso vídeo no Youtube explicando esse mesmo passo a passo:

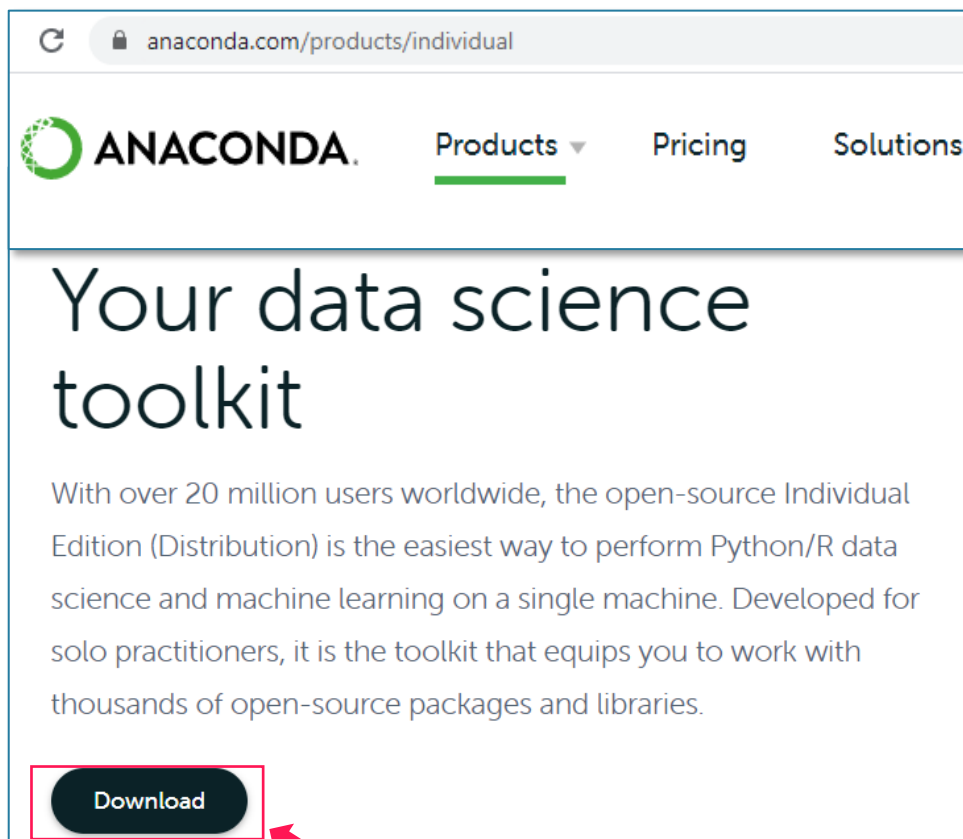
[Instalando o Jupyter - Pacote Anaconda para Programação em Python](#)



The screenshot shows the Anaconda website's product page for the Individual Edition. The URL in the browser is [anaconda.com/products/individual](https://anaconda.com/products/individual). The page features the Anaconda logo, navigation links for Products, Pricing, Solutions, and Resources. The main heading is "Your data science toolkit". Below this, a paragraph states: "With over 20 million users worldwide, the open-source Individual Edition (Distribution) is the easiest way to perform Python/R data science and machine learning on a single machine. Developed for solo practitioners, it is the toolkit that equips you to work with thousands of open-source packages and libraries." A prominent "Download" button is visible. At the bottom, there is a section for a video titled "Instalando o Jupyter - Pacote Anaconda para Programação em Python" with 7,172 views and a date of January 25, 2021. It also includes a hashtag logo for "#python #hashtagprogramacao" with 18,200 subscribers.

# Jupyter Notebook Instalação

1) No link indicado, clique em **Download**



anaconda.com/products/individual

ANACONDA. Products Pricing Solutions

## Your data science toolkit

With over 20 million users worldwide, the open-source Individual Edition (Distribution) is the easiest way to perform Python/R data science and machine learning on a single machine. Developed for solo practitioners, it is the toolkit that equips you to work with thousands of open-source packages and libraries.

**Download**

2) **Escolha** a opção adequada para seu computador

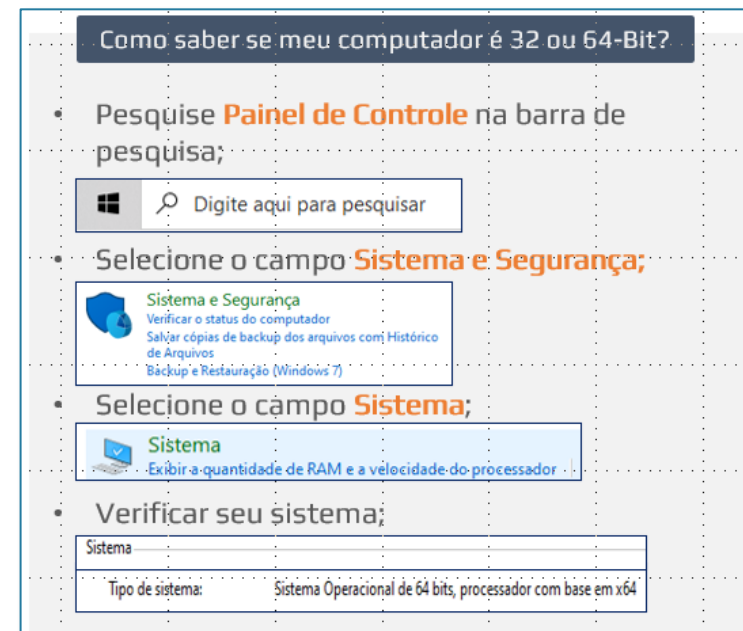


Windows

Python 3.8

64-Bit Graphical Installer (457 MB)

32-Bit Graphical Installer (403 MB)



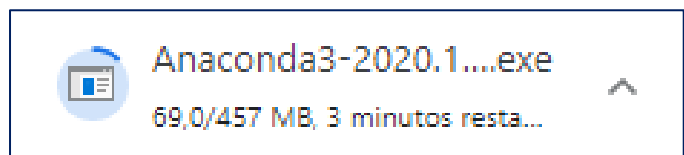
Como saber se meu computador é 32 ou 64-Bit?

- Pesquise **Painel de Controle** na barra de pesquisa;
- Selecione o campo **Sistema e Segurança**;
- Selecione o campo **Sistema**;
- Verificar seu sistema;

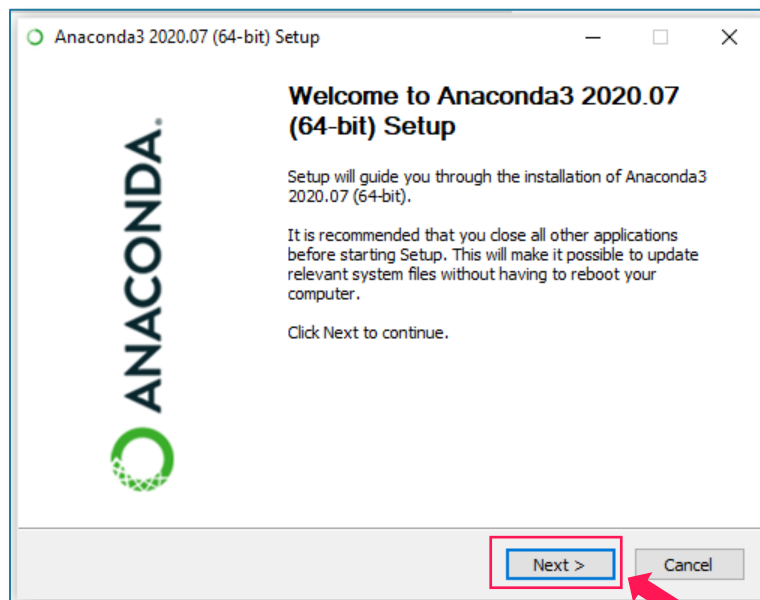
Sistema

Tipo de sistema: Sistema Operacional de 64 bits, processador com base em x64

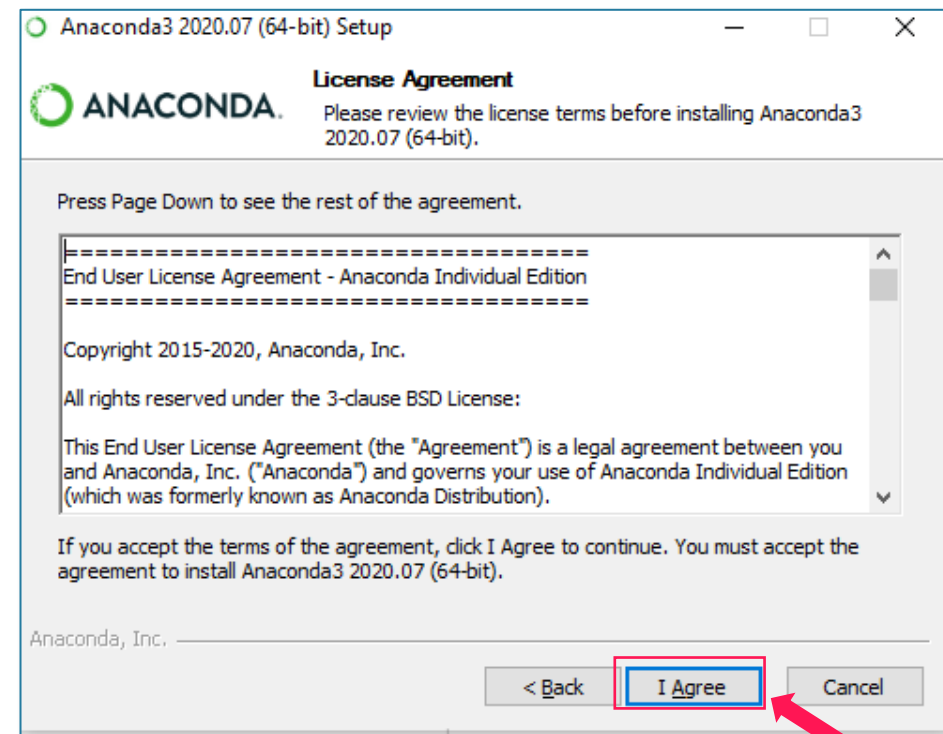
### 3) Fazer Download



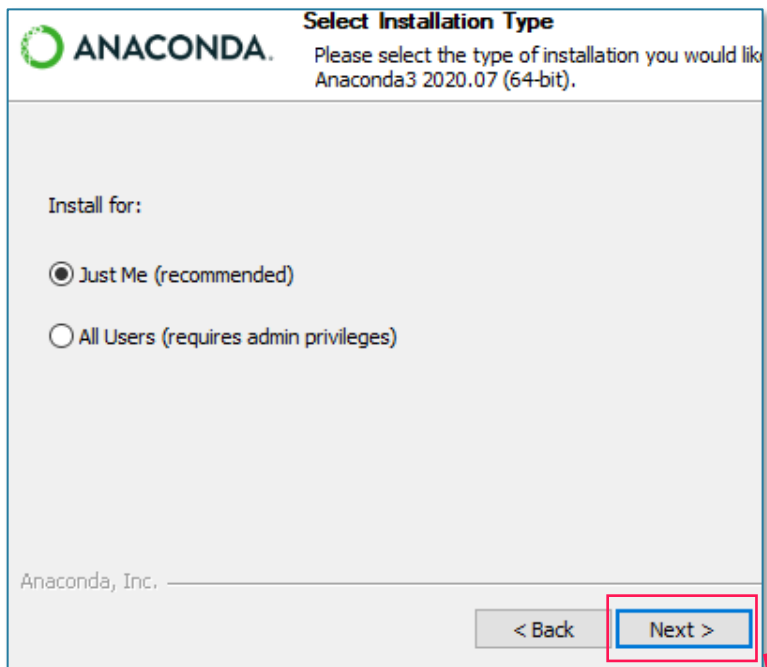
### 3) Abra o instalador do Anaconda



### 4) Aceite os termos de uso



5) No link indicado, clique em **Download**

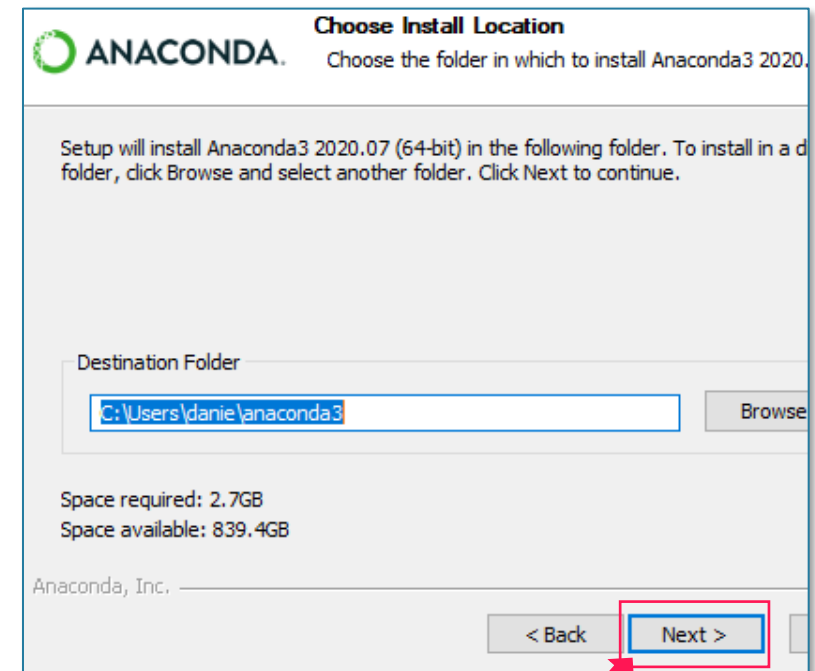


Nessa parte da instalação indicamos a opção **JUST ME** pois em teoria apenas o seu usuário precisa ter o Anaconda instalado.

**PORÉM**, em alguns casos, a instalação **JUST ME**, gera algumas falhas na inicialização do **JUPYTER NOTEBOOK** que vamos utilizar durante o curso.

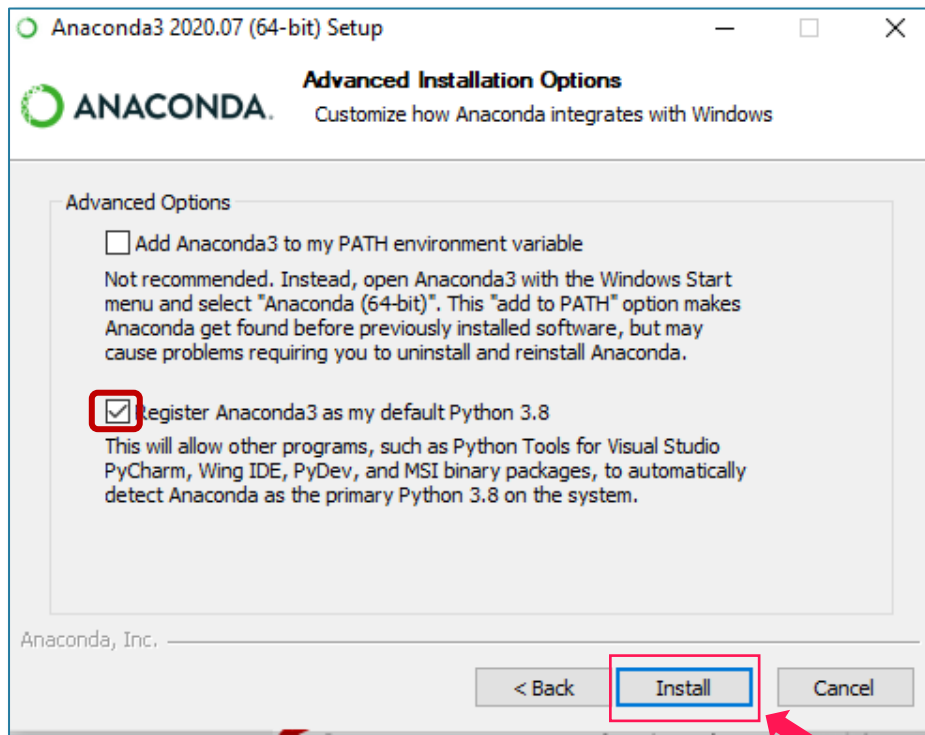
Caso aconteça com você, reinstale e utilize a opção **ALL USERS**.

6) Aperte **Next** e siga a pasta padrão definida pelo Anaconda

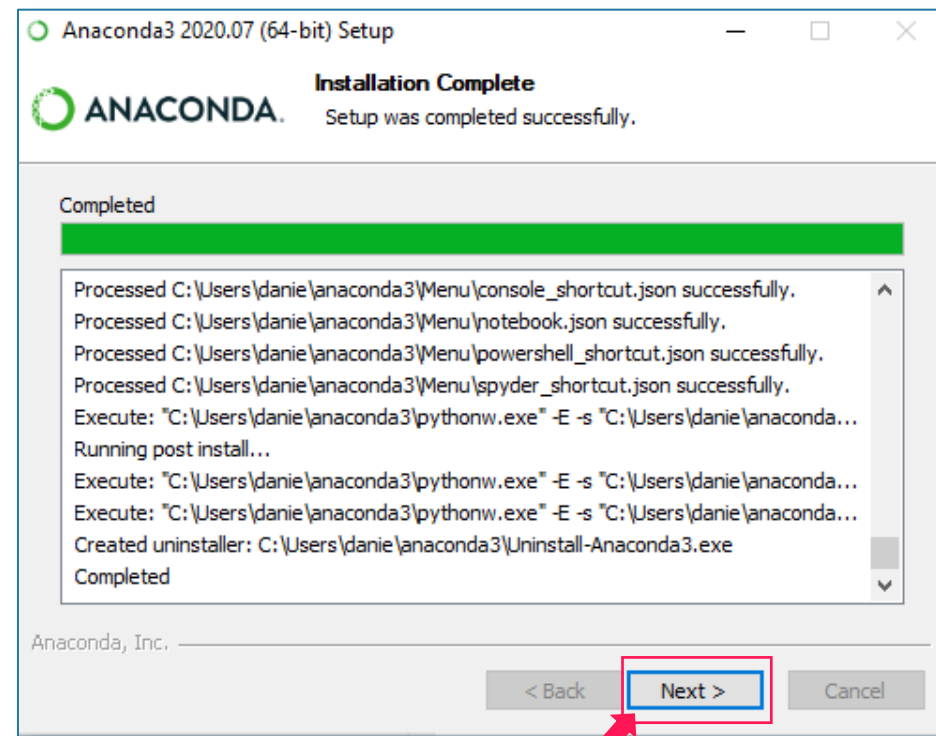


# Jupyter Notebook Instalação

7) Defina o Anaconda como seu Python padrão e siga com a instalação clicando em **Install**

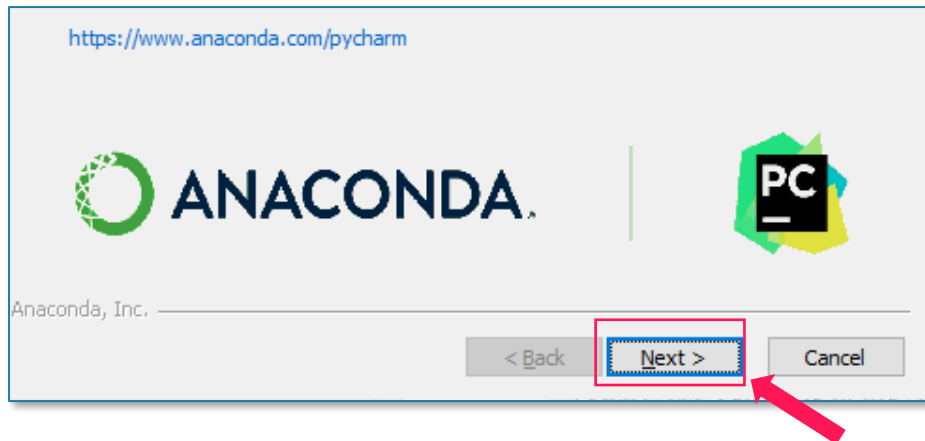


8) Ao fim da instalação clique em Next

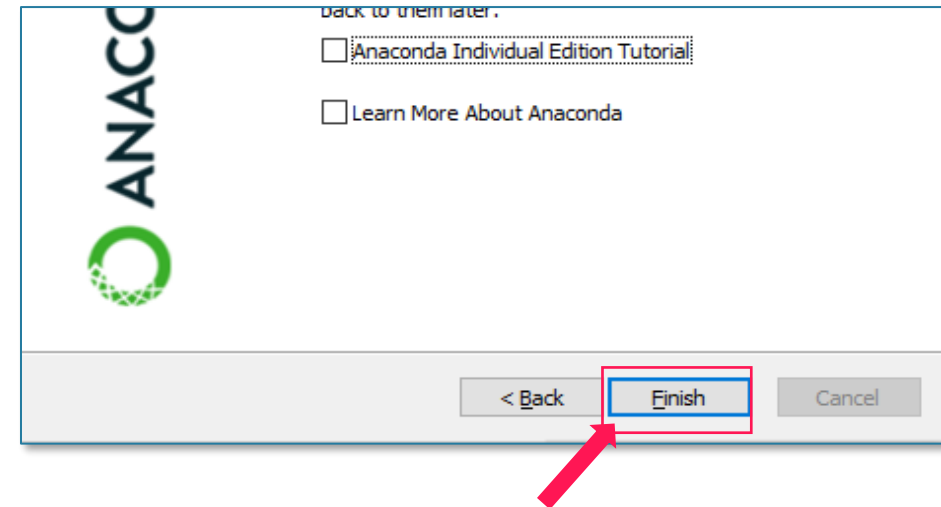


# Jupyter Notebook Instalação

9) Mais um **Next**



10) Clique em **Finish** para finalizar a instalação



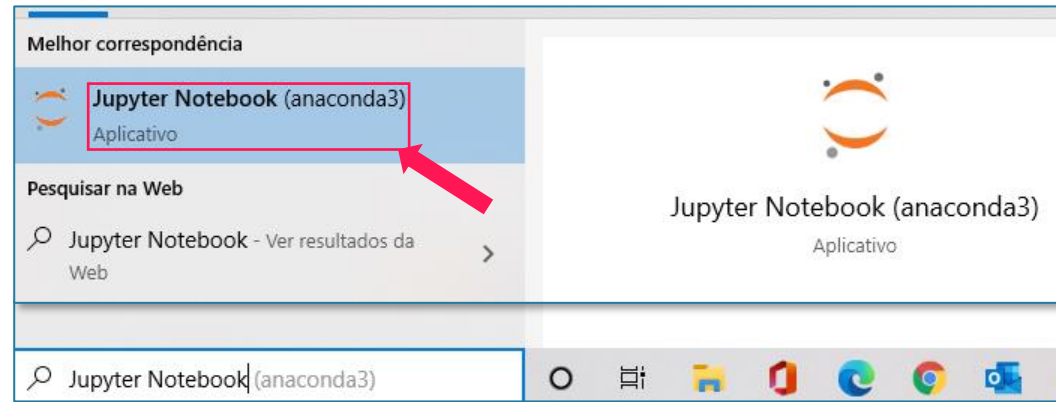
Pronto! Anaconda instalado. Agora vamos ver se está tudo OK para começarmos!

# Jupyter Notebook

## Inicializando o Jupyter

Para inicializarmos o Jupyter, basta digitar **Jupyter Notebook** no menu iniciar do seu computador.

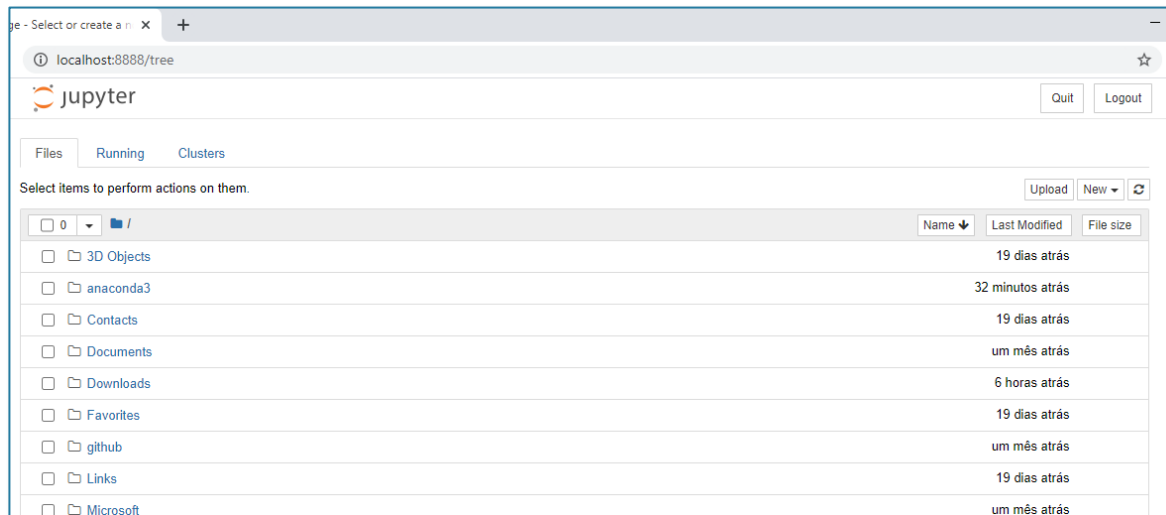
Uma nova janela será aberta no seu navegador padrão de internet.



## ATENÇÃO !

Ao clicar no ícone do Jupyter seu navegador padrão deverá abrir o Jupyter Notebook como no print 2.

Além disso, uma janela preta com o símbolo do Jupyter irá abrir. **Não feche esta janela!** Ela é o Jupyter Notebook sendo rodado pelo seu computador.





# Jupyter Notebook

## Inicializando o Jupyter

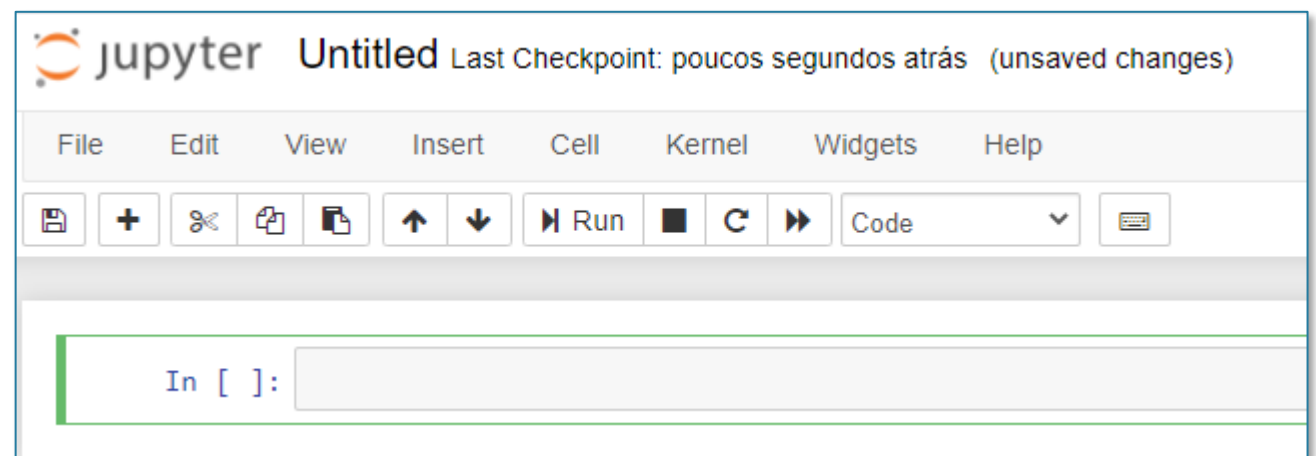
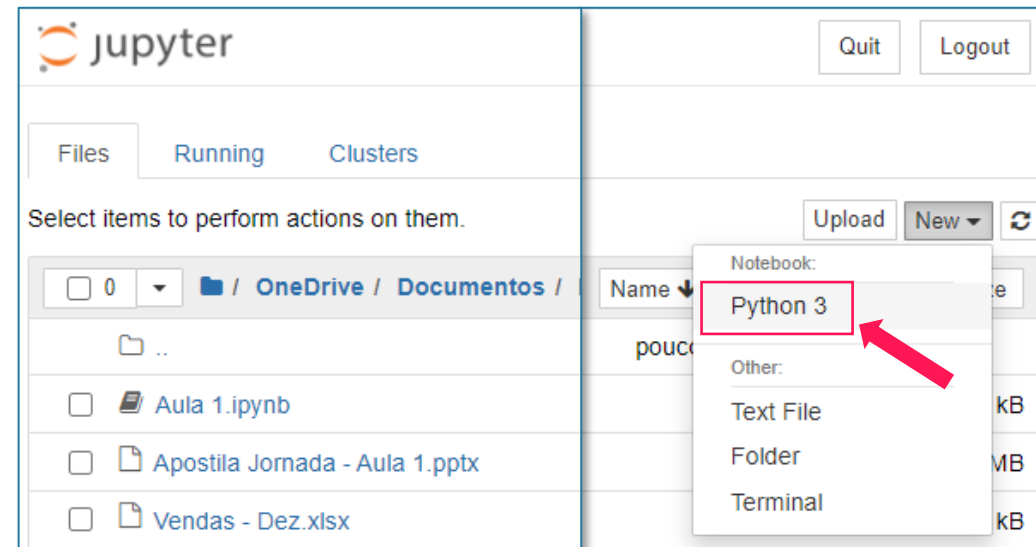
Após os preparativos, vamos acessar o nosso primeiro arquivo para criarmos nosso código em Python.

Esse arquivo se chama Notebook e possui o formato **.ipynb**. Ele só será aberto dentro de plataformas como o Google Colab ou Jupyter Notebook.

Para **criarmos um novo Notebook**, devemos clicar em **New > Python3**, assim como apresentado na figura ao lado.

Ao criar o novo Notebook, o Jupyter Notebook abrirá a janela ao lado. Aqui, é onde escreveremos nosso programa.

Mas antes de tudo, vamos entender a interface dessa plataforma.



# Entendendo a interface

The image shows the Jupyter Notebook interface with several annotations in Portuguese:

- Nome do arquivo. Formato ipynb.**: Points to the **Untitled** text in the top bar.
- Last Checkpoint: um minuto atrás (unsaved changes)**: Text displayed next to the filename.
- File Edit View Insert Cell Kernel Widgets Help**: The menu bar.
- +**: A button in the toolbar, annotated with **Cria nova célula** (Create new cell).
- Run**: A button in the toolbar, annotated with **Executa o código** (Execute code).
- In [ ]:**: The prompt for the code cell.
- Célula onde deve ser escrito o código**: Points to the text input area of the code cell.
- Região de Output**: Points to the output area below the code cell.

Parte 5

# Extração/Obtenção dos dados



**INTENSIVÃO DE {#}**  
**PYTHON{#}**  
100% ONLINE & GRATUITO

# Extração/Obtenção dos dados

## Importando o Pandas

Como vimos no nosso diagrama, as partes 1 e 2 do nosso processo de Ciência de Dados é realizada FORA do Python. De forma bem simples, são as informações que introduzimos no Capítulo 1 dessa apostila.

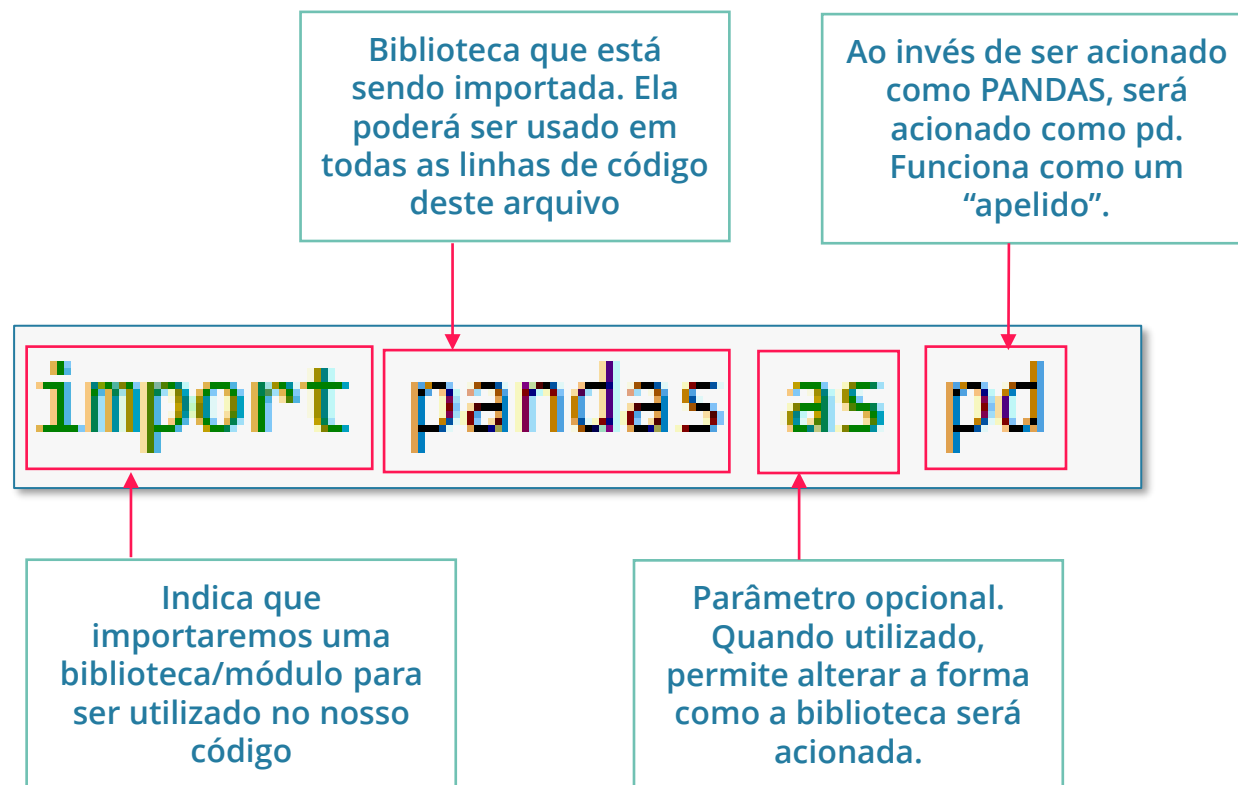
Agora vamos para a primeira etapa DENTRO do Python. A de obtenção de dados de fontes externas.

Apesar de agora ter um nome mais bonito, é algo que já vimos nas aulas 1 e 2 da Semana do Python. Usando o PANDAS podemos realizar essa importação dos dados do arquivo CSV e leva-los para o Jupyter Notebook.

Para realizar essa etapa, iremos usar a linha de código abaixo:

```
Import pandas as pd
```

Caso você ainda não conheça o Pandas, sugerimos dar uma olhadinha nas apostilas das aulas 1 e 2 😊.



# Importando os dados de um arquivo .csv

Como vimos no nosso diagrama, as partes 1 e 2 do nosso processo de Ciência de Dados é realizada FORA do Python. De forma bem simples, são as informações que introduzimos no Capítulo 1 dessa apostila.

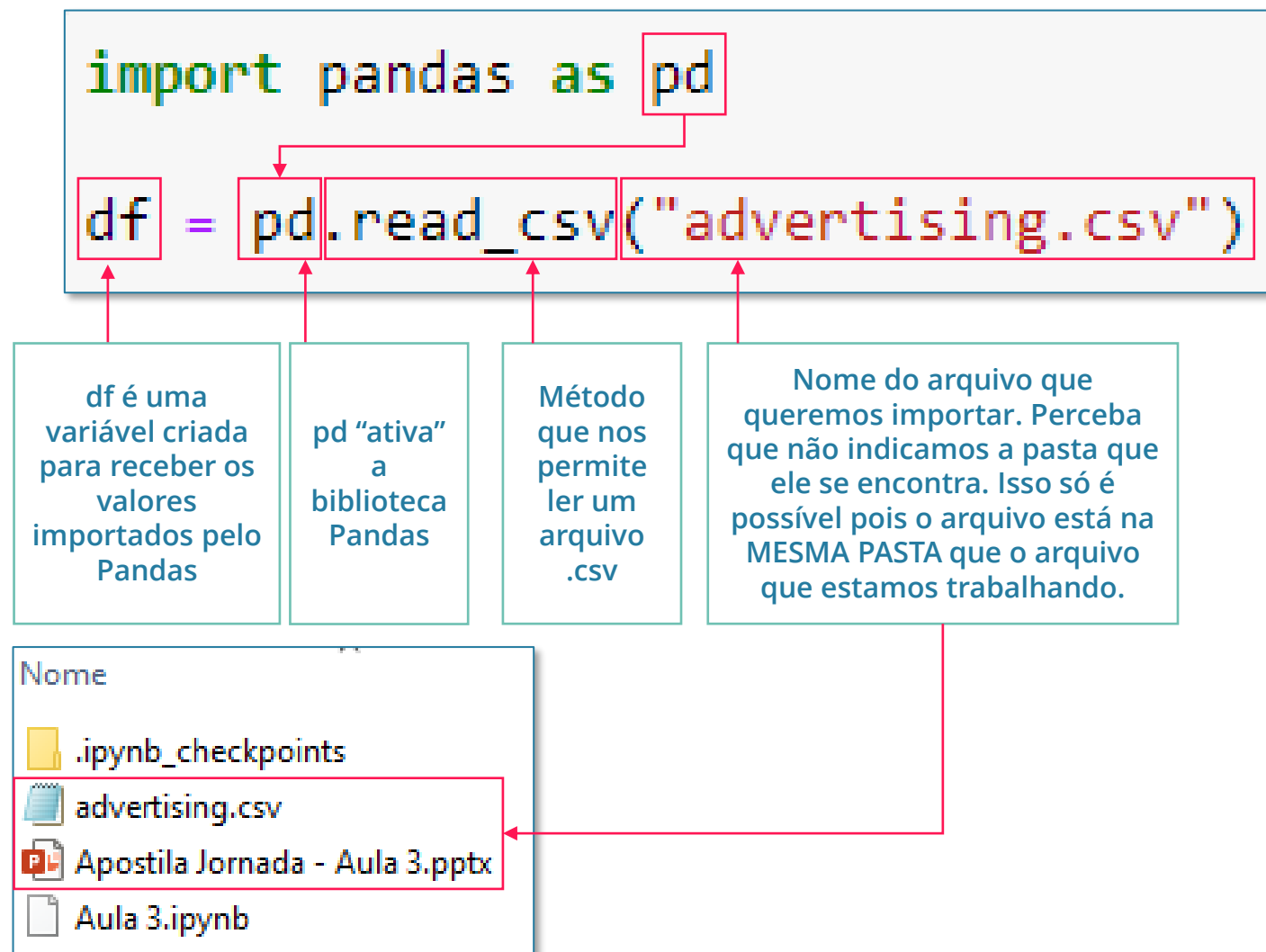
Agora vamos para a primeira etapa DENTRO do Python. A de obtenção de dados de fontes externas.

Apesar de agora ter um nome mais bonito, é algo que já vimos nas aulas 1 e 2 na Semana do Python. Usando o PANDAS podemos realizar essa importação dos dados do arquivo CSV e leva-los para o Jupyter Notebook.

Para realizar essa etapa, iremos usar a linha de código abaixo:

```
Import pandas as pd
```

Caso você ainda não conheça o Pandas, sugerimos dar uma olhadinha nas apostilas das aulas 1 e 2 😊.



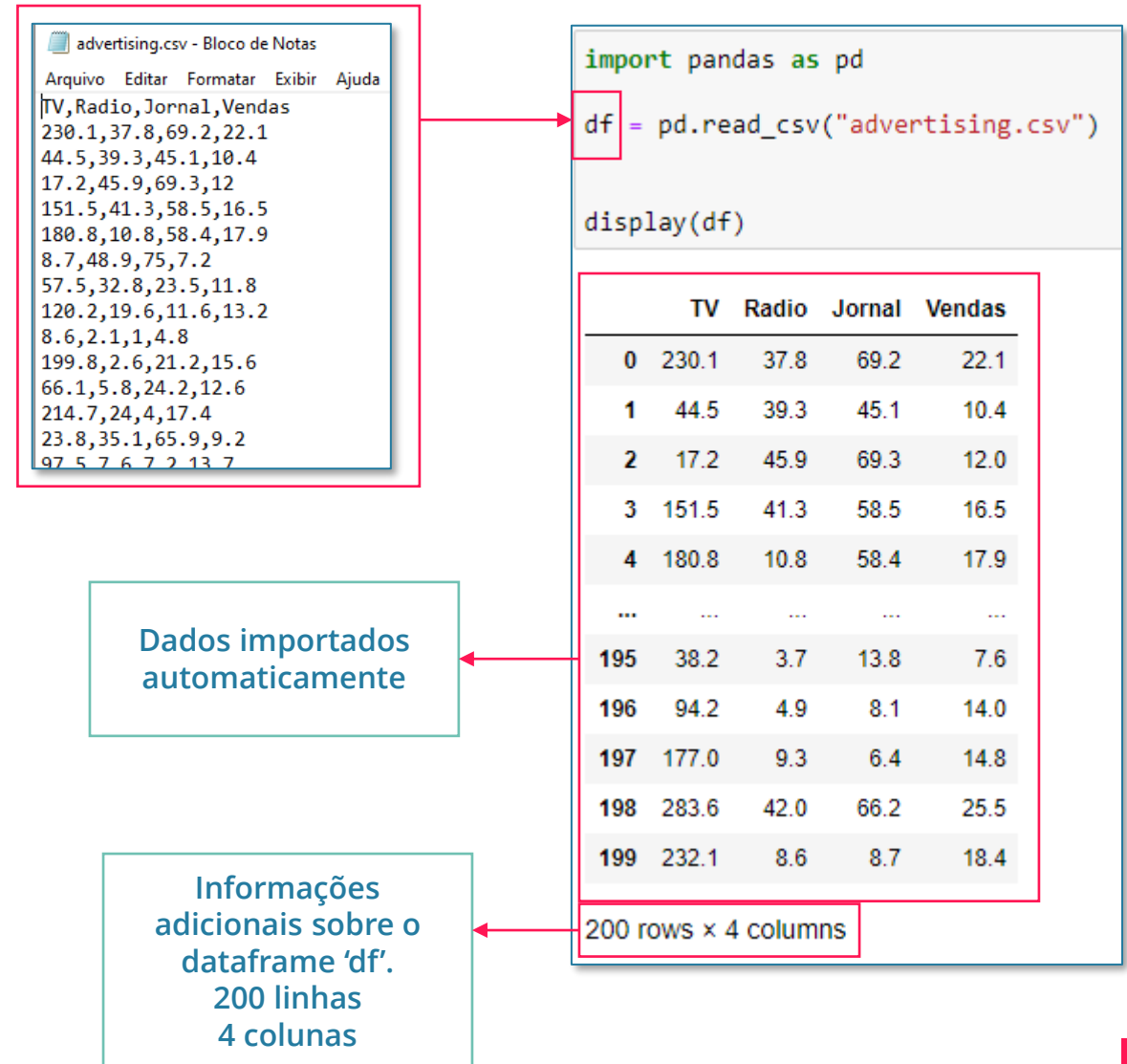
# Extração/Obtenção dos dados

## Visualização inicial dos dados

Feita a importação, todos os dados que existiam no arquivo **advertising.csv** agora foram armazenados na variável **df**.

Para visualizarmos essa informação, basta usarmos o **display(df)**.

Perceba no exemplo ao lado, que ao executarmos o código o próprio Pandas/Python, nos trouxe uma planilha formatada, auxiliando na análise preliminar dos dados.



Parte 6

# Ajustes de Dados (Limpeza de Dados)



**INTENSIVÃO DE**  
**PYTHON** {#}  
100% ONLINE & GRÁTUITO



# Tá ruim? Arruma! Tá Bom? Deixa do jeito que tá!

Essa é uma das fases mais operacionais e menos “divertidas” do processo de Ciência de Dados.

O que estamos buscando aqui de forma geral são:

- Dados faltantes;
- Dados com valor 0(ZERO) quando deveriam ser valores coerentes;
- Outliers que não representam bem o histórico e logo não ajudam a criação do modelo;
- Erros na base;
- Erros de importação;
- Etc...

Uma ferramenta que pode nos ajudar a ter uma visão bem geral dos dados é o método `.info()`.

Como podemos ver ao lado, nossos dados parecem coerentes. Portanto, nos livramos de fazer o tratamento mais pesado.

Se você acompanhou a aula 2 da Semana do Python, sabe do que estamos falando. Se não, dá uma olhadinha na apostila 😊.

```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 4 columns):
#   Column  Non-Null Count  Dtype  
---  -
0   TV       200 non-null    float64
1   Radio    200 non-null    float64
2   Jornal   200 non-null    float64
3   Vendas   200 non-null    float64
dtypes: float64(4)
memory usage: 6.4 KB
None
```

Todos os dados são numéricos como deveriam ser

Todos os dados são “não-nulos”

Parte 7

# Análise Exploratória



**INTENSIVÃO DE**  
**PYTHON** {#}  
100% ONLINE & GRATUITO

# Importando as bibliotecas gráficas

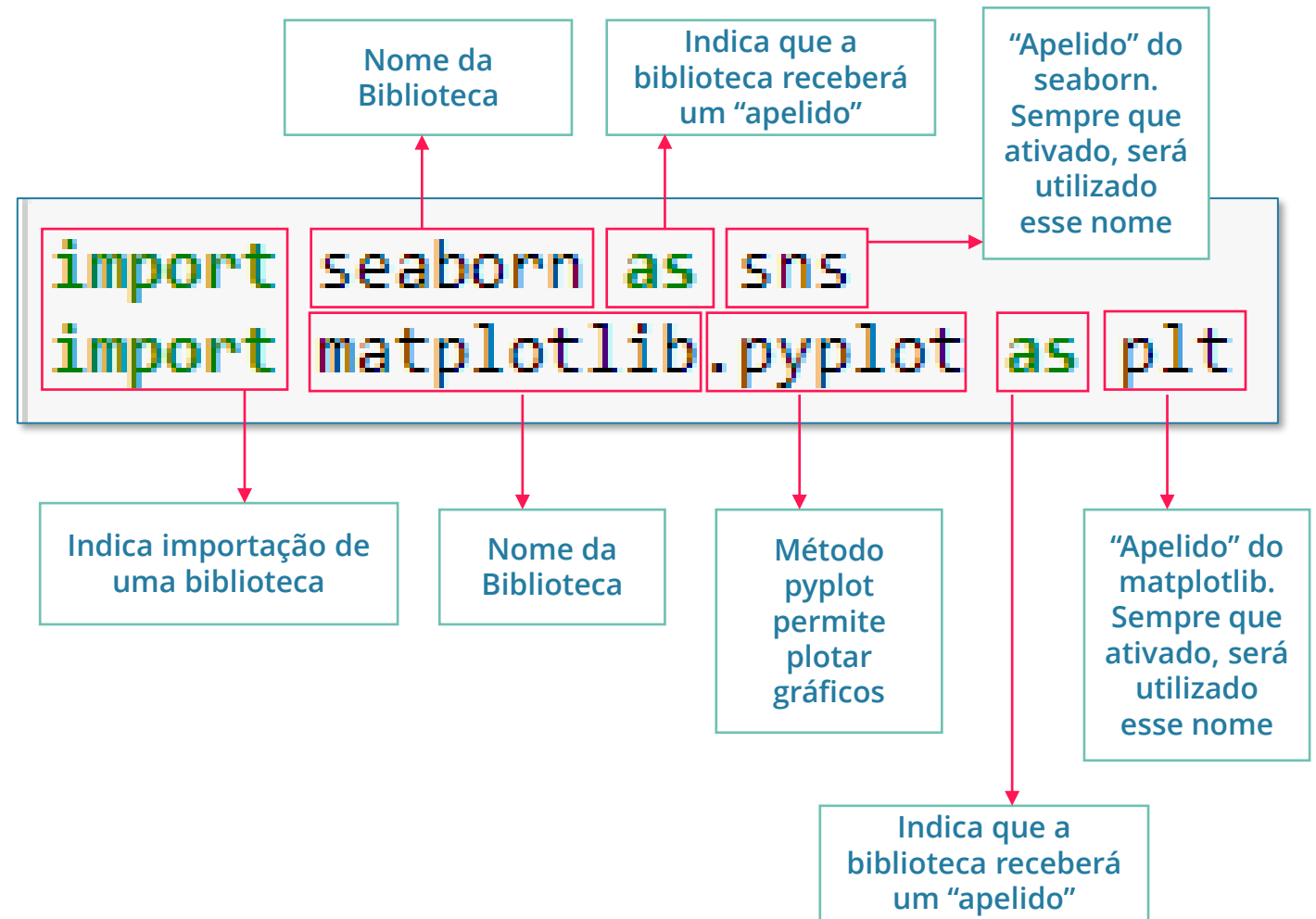
Se a parte anterior era uma das mais operacionais e braçais, essa é uma das mais divertidas 😊.

Com o uso do Python e do Pandas, vamos poder gerar uma série de análises estatísticas/gráficas que nos permita entender melhor como esses dados se comportam.

Para isso, vamos importar 2 bibliotecas que nos auxiliam a criar gráficos:

- **Seaborn** ([link](#));
- **Matplotlib** ([link](#));

Assim como fizemos na aula 2 com o Plotly para utilizarmos essas bibliotecas, é necessário importa-las para nosso código. Para isso usaremos os comandos ao lado.



# Análise Exploratória dos dados

## Análises Gráficas (1/4)

Talvez tenha ficado a dúvida:

“Mas para que usarmos 2 bibliotecas distintas para criação de gráficos? Uma não é suficiente?”

O que acontece aqui é que o Pandas/Python, já possuem uma integração com o Matplotlib. No entanto, seus gráficos são um pouco mais simples em termos visuais.

Vamos unir a beleza/complexidade do seaborn com a integração oferecida pelo Matplotlib.

Para isso, vamos criar um gráfico via Seaborn e exibi-lo via matplotlib.

Essas etapas estão apresentadas na imagem ao lado.

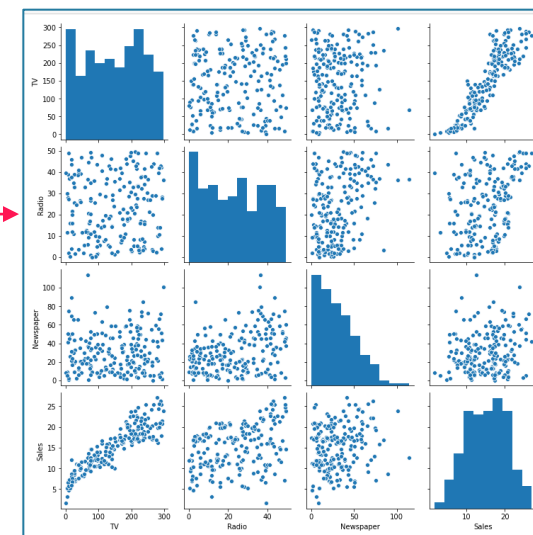
No próximo slide, vamos entender um pouco melhor sobre o gráfico que acabamos de gerar.

```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
sns.pairplot(df)
plt.show()
```

Indica o tipo de gráfico da biblioteca seaborn que será utilizado. Os dados base serão utilizados, são os do nosso dataframe df.

Exibe o gráfico criado



# Análise Exploratória dos dados

## Análises Gráficas (2/4)

Achou estranho com apenas 2 linhas gerar o que parecem ser 16 gráficos diferentes?

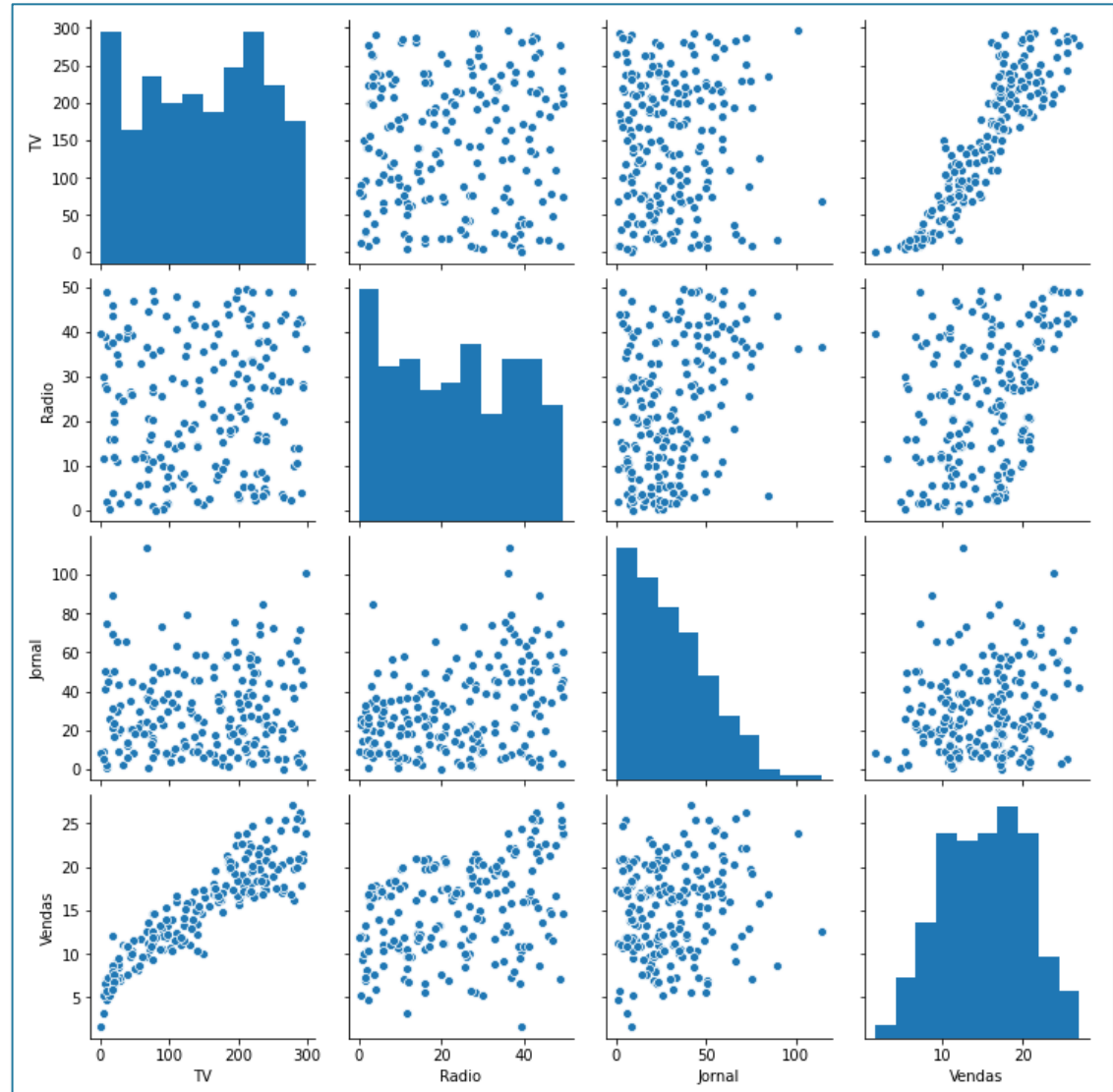
Legal né?

Aqui usamos um gráfico de dispersão para cada uma das combinações de colunas possíveis. Isso nos permite perceber se existem alguma relação entre elas.

Alguns casos, serão úteis, outros, não. Alguns nem sentido farão. **Saber diferenciar isso e entender o que está acontecendo é uma peça chave para um bom resultado no fim do processo.**

Lembre-se por enquanto estamos em uma **ANÁLISE EXPLORATÓRIA!** Estamos mais buscando caminhos do que grandes conclusões 😊.

Vamos pegar 2 dos 16 gráficos para entendê-los um pouco melhor.



# Análises Gráficas (3/4)

Antes da análise vamos entender o que são os pontos do gráfico.

Aqui, temos o resultado em vendas (eixo y) pelo investimento no canal (eixo x).

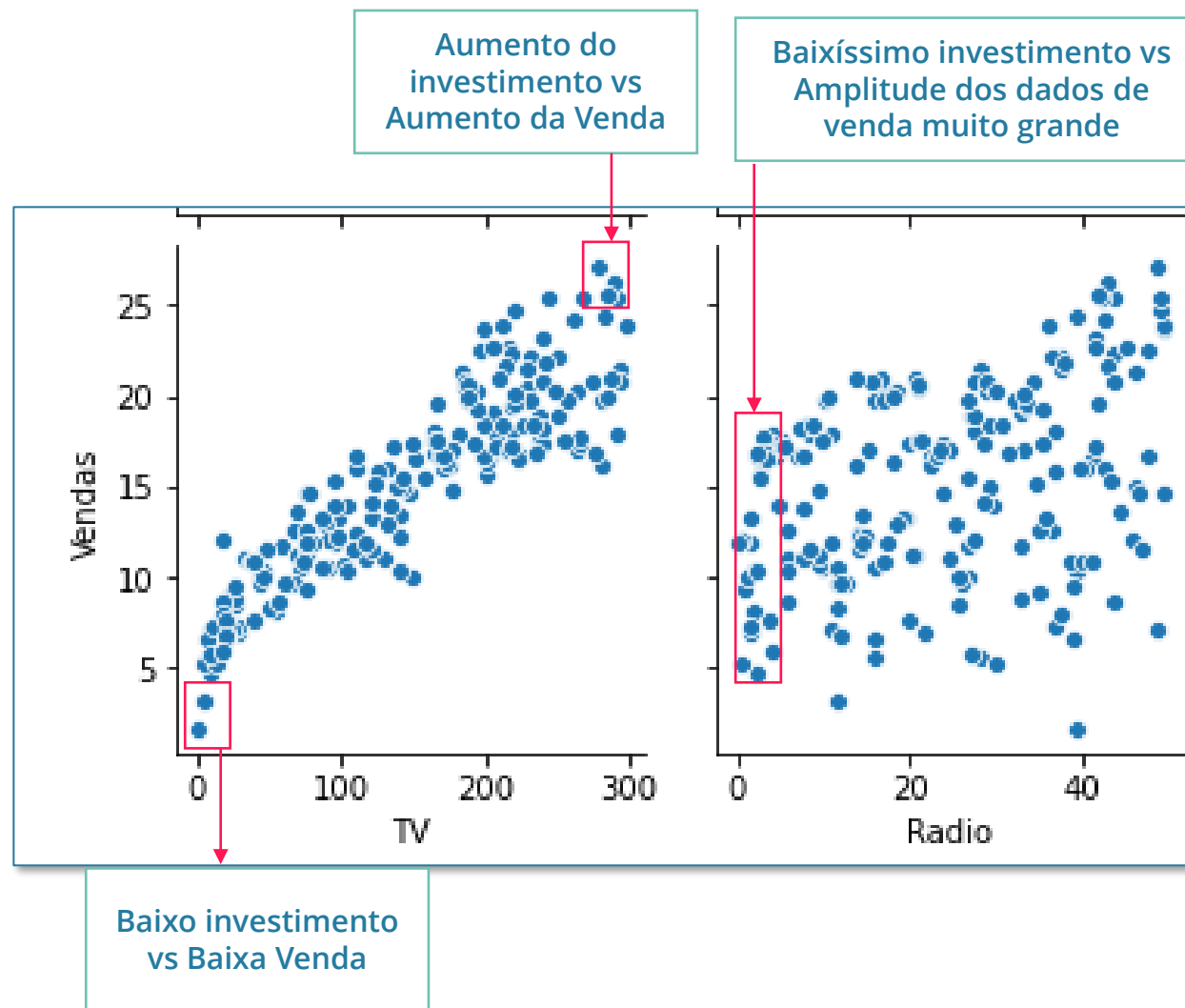
Só de olhar conseguimos ver uma diferença gritante no formato do gráfico.

No **gráfico TV X VENDAS**, podemos perceber que existe uma proporcionalidade no investimento e o aumento das vendas.

Ou seja, quando invisto em anúncios de TV, tendo a vender mais.

Já no **gráfico RADIO X VENDAS**, essa proporcionalidade não é clara. Na verdade, não é muito óbvia a relação entre elas.

Ou seja, as vendas tendem a existir com alto ou baixo investimento nos anúncios de rádio.



# Análises Gráficas (4/4)

Tivemos alguns direcionadores do que pode estar acontecendo, mas não faz muito sentido ficarmos avaliando os pontos 1 a 1 no olho...

Vamos usar outro método da biblioteca seaborn para conseguirmos avaliar melhor esses dados.

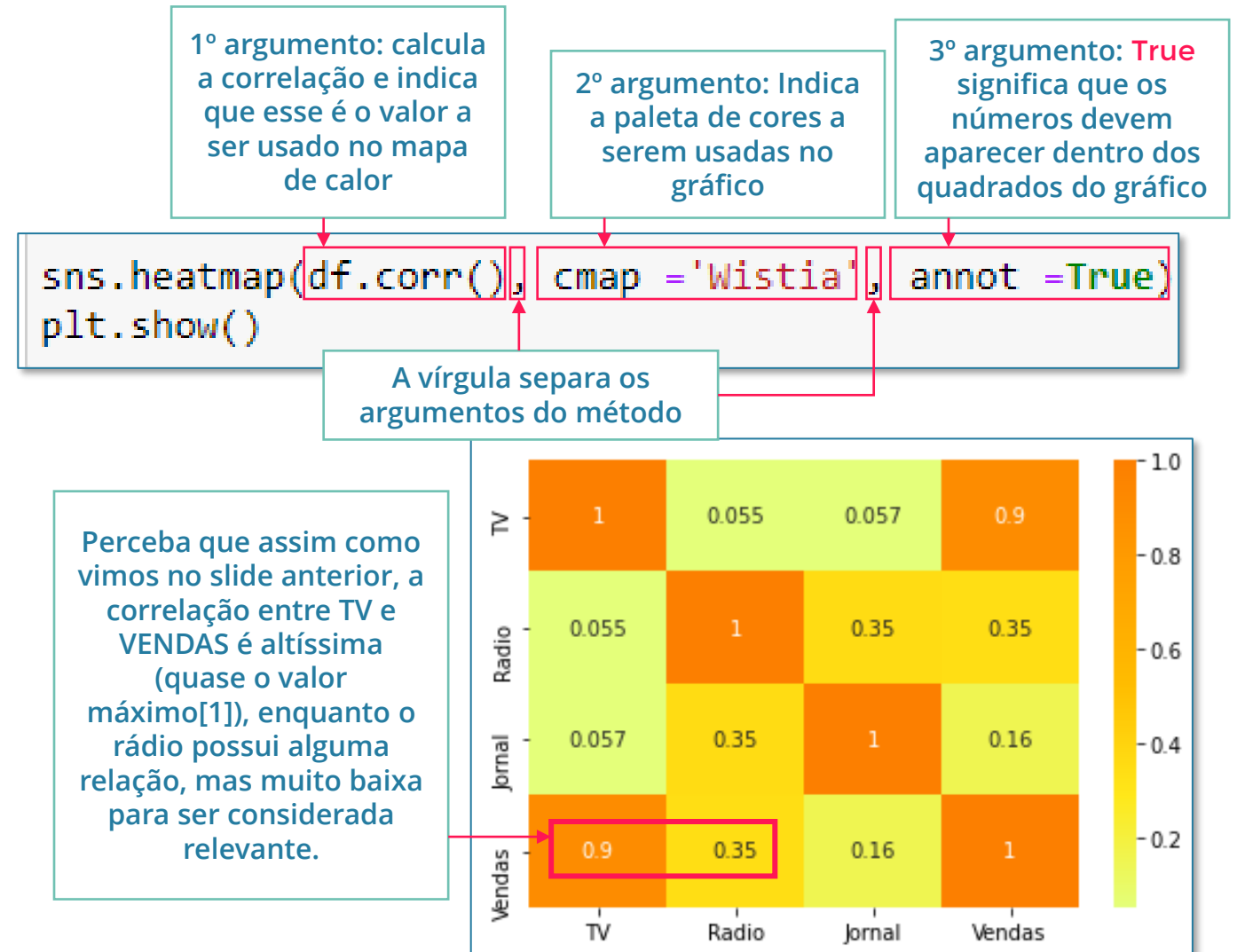
Este método é o `heatmap()`.

Aqui, vamos usar alguns **argumentos dentro do método**. Argumentos são essencialmente configurações que o método deverá seguir ao ser executado.

Vamos entendê-los um pouco melhor:

- `df.corr()` -> esse argumento indica o que está sendo plotado. Nesse caso, `df.corr()` indica que será calculado a CORRELAÇÃO\* entre os dados do dataframe `df`, que é uma
- `cmap = 'Wistia'` -> indica a paleta de cores a ser utilizada no gráfico;
- `annot=True` Escreve dentro dos quadrados o valor

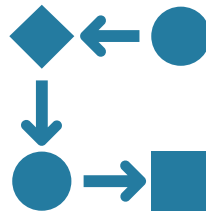
\*medida estatística que pode variar de -1(inversamente proporcional) a 1(diretamente proporcional);





Parte 8

# Modelagem + Algoritmos



# Criando base de dados para modelo (1/4)

Vamos agora para fase de criação de um modelo.

O que precisamos identificar são:

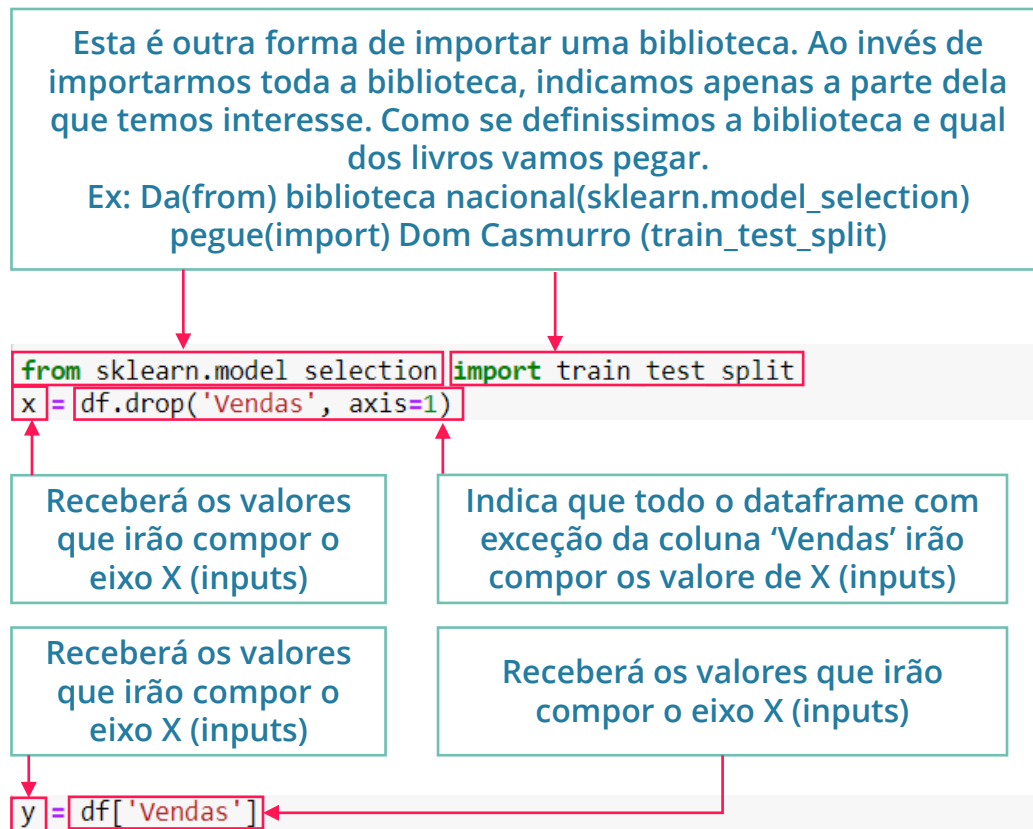
- Os inputs do modelo (eixo X);
- Os outputs do modelo (eixo Y);

Lembrando que nosso objetivo é criar um modelo de previsão de vendas baseado nos dados de investimento nos diferentes canais (TV, radio, jornais).

Ou seja, nossos inputs são valor do investimento nos diferentes canais e nosso output será a venda prevista dado os inputs informados.

Vamos iniciar este bloco de código informando exatamente isso. Usaremos para essa etapa uma nova biblioteca. E como sempre, vamos precisar importá-las.

`sklearn.model_selection`



# Criando base de dados para modelo (2/4)

Agora temos claro nas nossas variáveis X e Y quais são os Inputs e quais são os Outputs.

Para criarmos um modelo de previsão usando inteligência artificial vamos precisar de dados para:

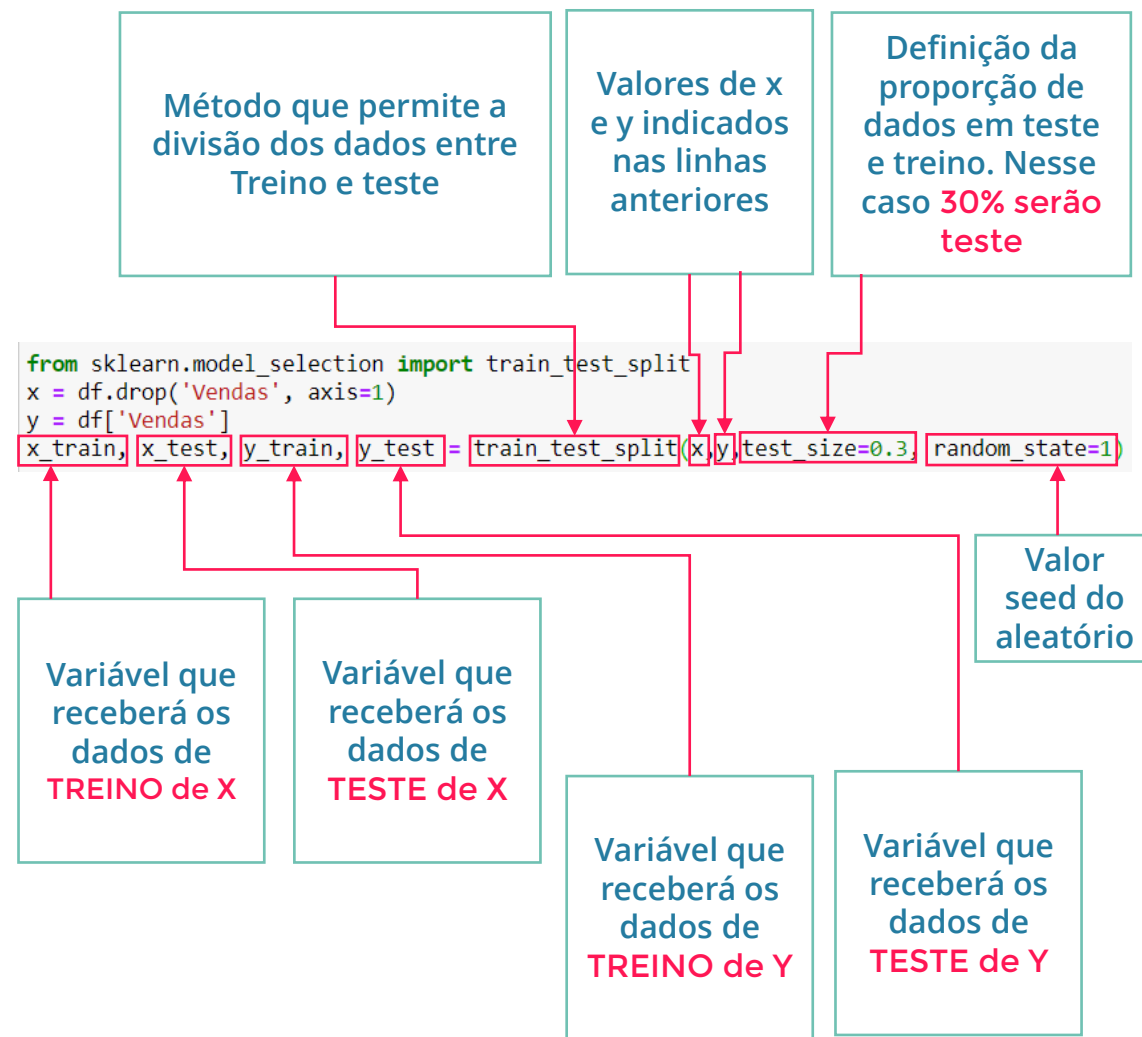
- 1) Treinar o modelo;
- 2) Testar o modelo.

Sabemos que temos 200 pontos não nulos na nossa base (informação dada pelo método info na fase de limpeza), temos algumas opções:

- 1) Usar 200 pontos para treino e 0 para testar;
- 2) Usar 0 pontos para treino e 200 para testar;
- 3) Dividir uma quantidade para treino e outra para teste.

Imagino que você escolheu a opção 3... Mas vamos entender por quê não as opções 1 e 2:

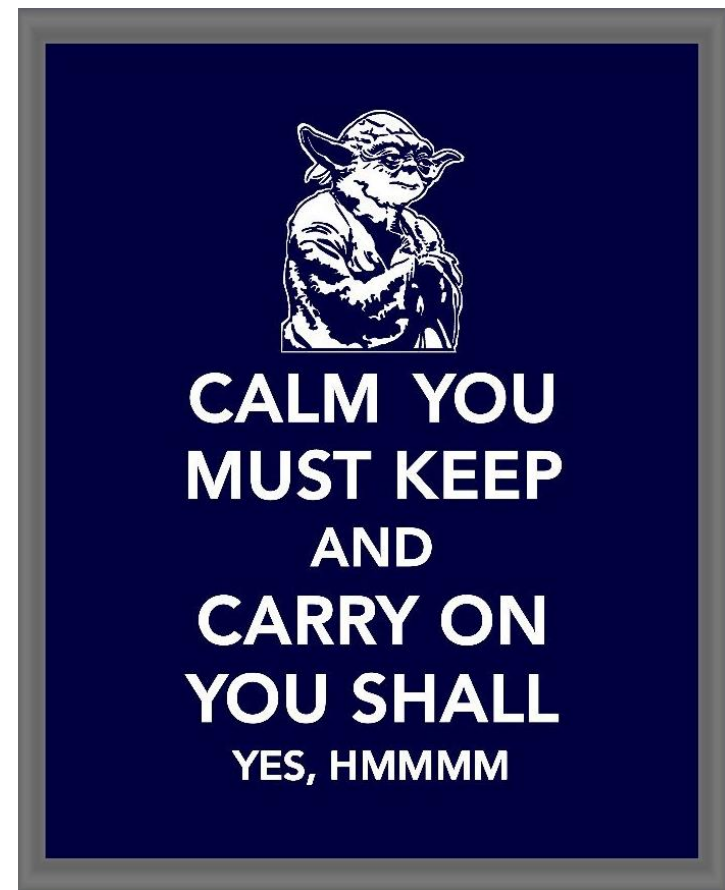
- 1) É como fazer 200 questões de Matemática, nunca fazer um simulado do ENEM e ir para prova. Pode funcionar, mas o risco é grande;
- 2) É como nunca ter feito um exercício e fazer todos os simulados do ENEM. O que você sabe? Em teoria, nada...



# PAUSA PARA REVISAR

Sabemos que a última etapa pode não ser tão direta em um primeiro momento, mas vamos revisar o que fizemos até agora e pensar o que vamos fazer para frente.

- 1) Importamos a base de dados do arquivo .CSV para nosso programa;
- 2) Pensamos em dar uma limpeza nos dados, mas como tudo estava OK, passamos para a próxima etapa;
- 3) Analisamos graficamente nossa base e usamos o mapa de calor juntamente com o conceito de correlação para entendermos que existia sim uma relação de proporcionalidade entre os investimentos e as vendas;
- 4) Começamos a pensar no nosso modelo de previsão que se utilizará de inteligência artificial;
- 5) Para gerar uma inteligência artificial precisamos treiná-la e testá-la. Como fazemos isso? Através de dados!
- 6) Criamos 4 variáveis(**x\_train**, **x\_test**, **y\_train**, **y\_test**) com objetivo de fornecê-las a nossa inteligência artificial;
- 7) Cá estamos! Usando nossa biblioteca vamos informar essas variáveis e iniciar o treinamento do nosso modelo. Feito isso, testar esse modelo.



# Criando base de dados para modelo (3/4)

Iniciamos por novas importações de dados. Aqui usaremos dois métodos distintos de regressão:

- Regressão Linear;
- Random Forest Regressor;

Além disso, usaremos o **numpy** que é uma biblioteca muito utilizada para grande volume de cálculos.

No código ressaltado ao lado, veja que iniciamos realizando o nosso **TREINO da inteligência artificial**. Primeiro com a regressão Linear e depois com o Random Forest.

Perceba que nessa fase usaremos apenas as variáveis do tipo **\_train**.

```
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn import metrics
import numpy as np

# treino AI
lin_reg = LinearRegression()
lin_reg.fit(x_train, y_train)

rf_reg = RandomForestRegressor()
rf_reg.fit(x_train, y_train)

# teste AI
test_pred_lin = lin_reg.predict(x_test)
test_pred_rf = rf_reg.predict(x_test)

r2_lin = metrics.r2_score(y_test, test_pred_lin)
rmse_lin = np.sqrt(metrics.mean_squared_error(y_test, test_pred_lin))
print(f"R² da Regressão Linear: {r2_lin}")
print(f"RSME da Regressão Linear: {rmse_lin}")
r2_rf = metrics.r2_score(y_test, test_pred_rf)
rmse_rf = np.sqrt(metrics.mean_squared_error(y_test, test_pred_rf))
print(f"R² do Random Forest: {r2_rf}")
print(f"RSME do Random Forest: {rmse_rf}")
```

Utilização apenas das variáveis de TREINO ('\_train')

```
R² da Regressão Linear: 0.9071151423684273
RSME da Regressão Linear: 1.5396947656031235
R² do Random Forest: 0.9658627724223138
RSME do Random Forest: 0.93341826994476
```

# Criando base de dados para modelo (4/4)

Feito o treino, agora precisamos iniciar os testes.

Perceba que agora, são utilizadas as variáveis finais **'\_test'** e não mais as **'\_train'**

Mas como saber se o teste foi bom ou não? Existem 2 indicadores que descrevem bem os resultados que se conseguiu.

Esses indicadores são:

- **R<sup>2</sup>**
- **RSME**

Sem entrar em detalhes estatísticos eles descrevem a qualidade da sua regressão.

Se olharmos o print do nosso código, veremos que o **valor de R<sup>2</sup> do Random Forest é MAIOR que o valor da regressão linear** (Esse valor varia entre 0 e 1).

Além disso, ao vermos que temos 0,96 como resultado, podemos inferir que temos um modelo confiável (Varia de indústria, mas para esse caso é um modelo aceitável).

```
# treino AI
lin_reg = LinearRegression()
lin_reg.fit(x_train, y_train)

rf_reg = RandomForestRegressor()
rf_reg.fit(x_train, y_train)

# teste AI
test_pred_lin = lin_reg.predict(x_test)
test_pred_rf = rf_reg.predict(x_test)

r2_lin = metrics.r2_score(y_test, test_pred_lin)
rmse_lin = np.sqrt(metrics.mean_squared_error(y_test, test_pred_lin))
print(f"R² da Regressão Linear: {r2_lin}")
print(f"RSME da Regressão Linear: {rmse_lin}")
r2_rf = metrics.r2_score(y_test, test_pred_rf)
rmse_rf = np.sqrt(metrics.mean_squared_error(y_test, test_pred_rf))
print(f"R² do Random Forest: {r2_rf}")
print(f"RSME do Random Forest: {rmse_rf}")
```

```
R² da Regressão Linear: 0.9071151423684273
RSME da Regressão Linear: 1.5396947656031235
R² do Random Forest: 0.9658627724223138
RSME do Random Forest: 0.93341826994476
```

Parte 9

# Interpretação do Resultado



**INTENSIVÃO DE {#}**  
**PYTHON{#}**  
100% ONLINE & GRATUITO



# Interpretação dos resultados

## Análise Gráfica

O gráfico ao lado mostra todos os 60\* pontos de teste que são dados reais extraídos da nossa base ao lado das duas curvas geradas pelos nossos modelos.

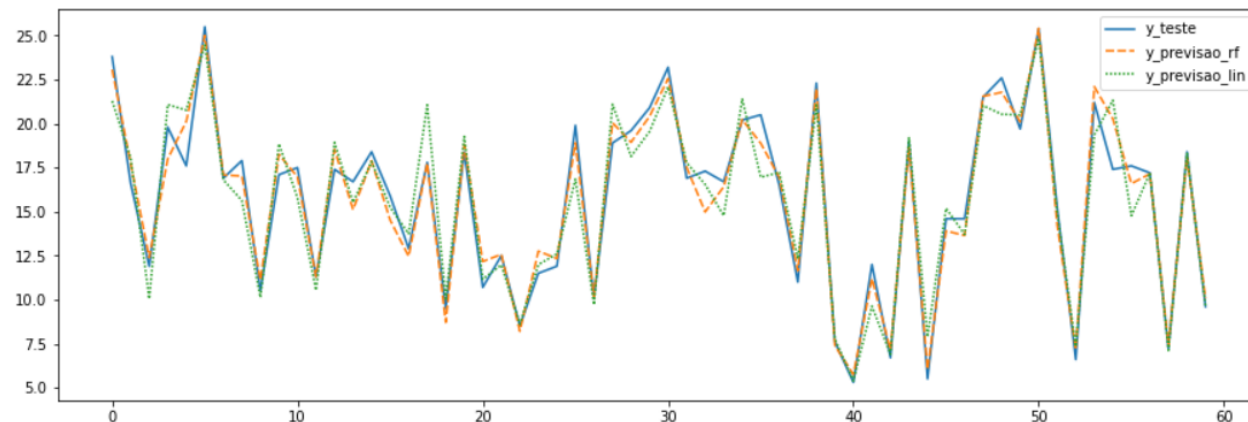
A linha azul representa os dados reais de teste;

A linha tracejada laranja representa os dados de previsão calculados pelo modelo de Random Forest

A linha verde tracejada representa os dados de previsão calculados pelo modelo de Regressão Linear.

Perceba que após a elaboração dos testes, utilizamos novamente o Pandas para criar um dataframe apenas com os resultados calculados e os dados de teste.

```
df_resultado = pd.DataFrame()
# df_resultado.index = x_test
df_resultado['y_teste'] = y_test
df_resultado['y_previsao_rf'] = test_pred_rf
df_resultado['y_previsao_lin'] = test_pred_lin
df_resultado = df_resultado.reset_index(drop=True)
fig = plt.figure(figsize=(15, 5))
sns.lineplot(data=df_resultado)
plt.show()
display(df_resultado)
```



\*lembrando que tínhamos 200 pontos e definimos 30% para teste, logo  $0,3 \times 200 = 60$

# Interpretação dos resultados

## Análise Gráfica

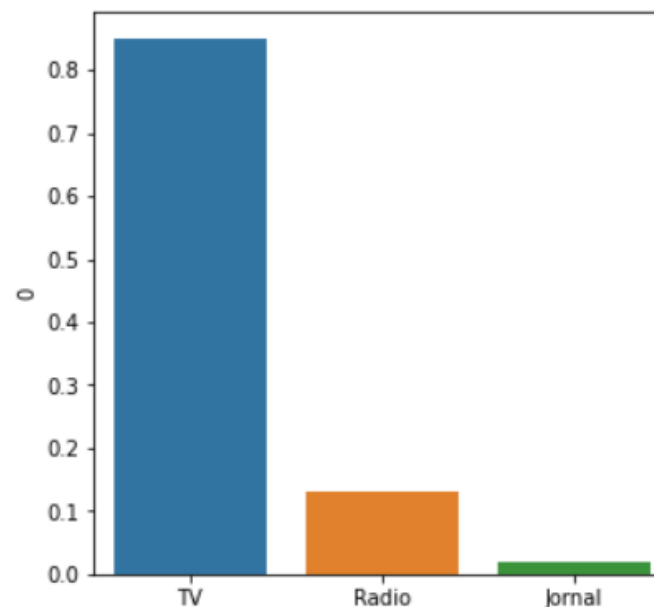
Lembra que na fase exploratória percebemos que o investimento em TV era o que possuía maior correlação com as vendas?

Será que nosso modelo possui a mesma característica?

Utilizando novamente nosso pandas e seaborn para auxiliar na elaboração do gráfico, podemos perceber que em termos de importância para o modelo via Random Forest:

- O investimento em TV é ~85% relevante;
- O Rádio que também avaliamos anteriormente, pouco mais de 10%;
- Jornal não chegando a mais de 5% de relevância

```
importancia_features = pd.DataFrame(rf_reg.feature_importances_, x_train.columns)
plt.figure(figsize=(5, 5))
sns.barplot(x=importancia_features.index, y=importancia_features[0])
plt.show()
```



Parte 10

# Deploy/Produção



**INTENSIVÃO DE**  
**PYTHON** {#}  
100% ONLINE & GRATUITO

# Deploy/Produção

E o Deploy? Ou melhor, o que é o Deploy?

Depois de criar um modelo e testar sua eficácia, chega a hora de colocar aquele modelo para rodar. Isso é o Deploy, ou colocar em modo de Produção.

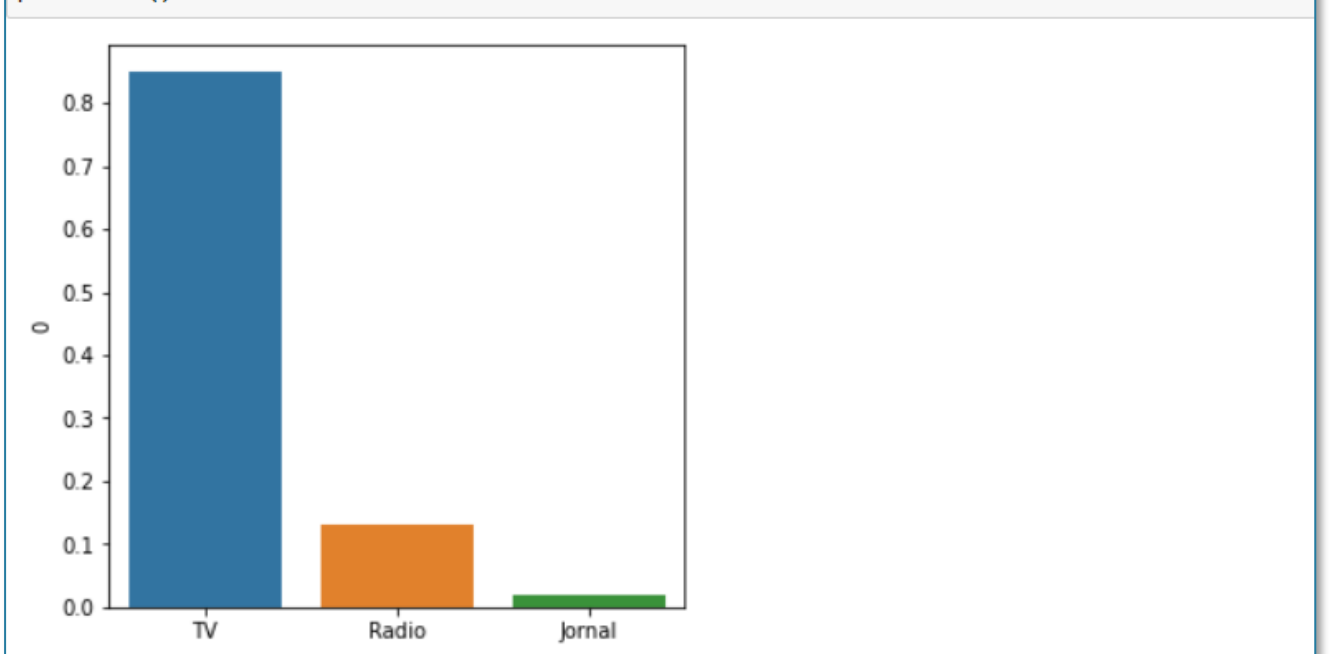
Aqui na Semana do Python, não vamos ver no detalhe ☹️

Mas em linhas gerais, o que precisamos fazer agora é criar uma espécie de sistema que possa ser executado por qualquer um que precise.

Assim você torna seu modelo escalável dentro da sua empresa.

Se tiver interesse de saber mais sobre Ciência de Dados e outros temas envolvendo Python fica de olho no nosso YouTube OU aproveita e já se inscreve no Python Impressionador 😊

```
importancia_features = pd.DataFrame(rf_reg.feature_importances_, x_train.columns)
plt.figure(figsize=(5, 5))
sns.barplot(x=importancia_features.index, y=importancia_features[0])
plt.show()
```



# INTENSIVÃO DE PYTHON {#}

100% ONLINE & GRATUITO

Ainda não segue a gente no Instagram e nem é inscrito no nosso canal do Youtube? Então corre lá!



@hashtagprogramacao



[youtube.com/hashtag-programacao](https://youtube.com/hashtag-programacao)

