

**EDUCAR PARA  
TRANSFORMAR**



**Estácio**

**CIÊNCIA DE DADOS E BIG DATA ANALYTICS  
LINGUAGEM PYTHON - NPG7875**

**Prof. Raphael Mauricio Sanches de Jesus  
Maio 2025**

## Objetivo 1

Tema 1: Python orientado a objeto.

- 1.1 Orientação a objetos
- 1.2 Orientação a objetos na linguagem Python
- 1.3 Orientação a objetos com herança e polimorfismo
- 1.4 Orientação a objetos aplicados a Python e outras existentes no mercado

## Objetivo 2

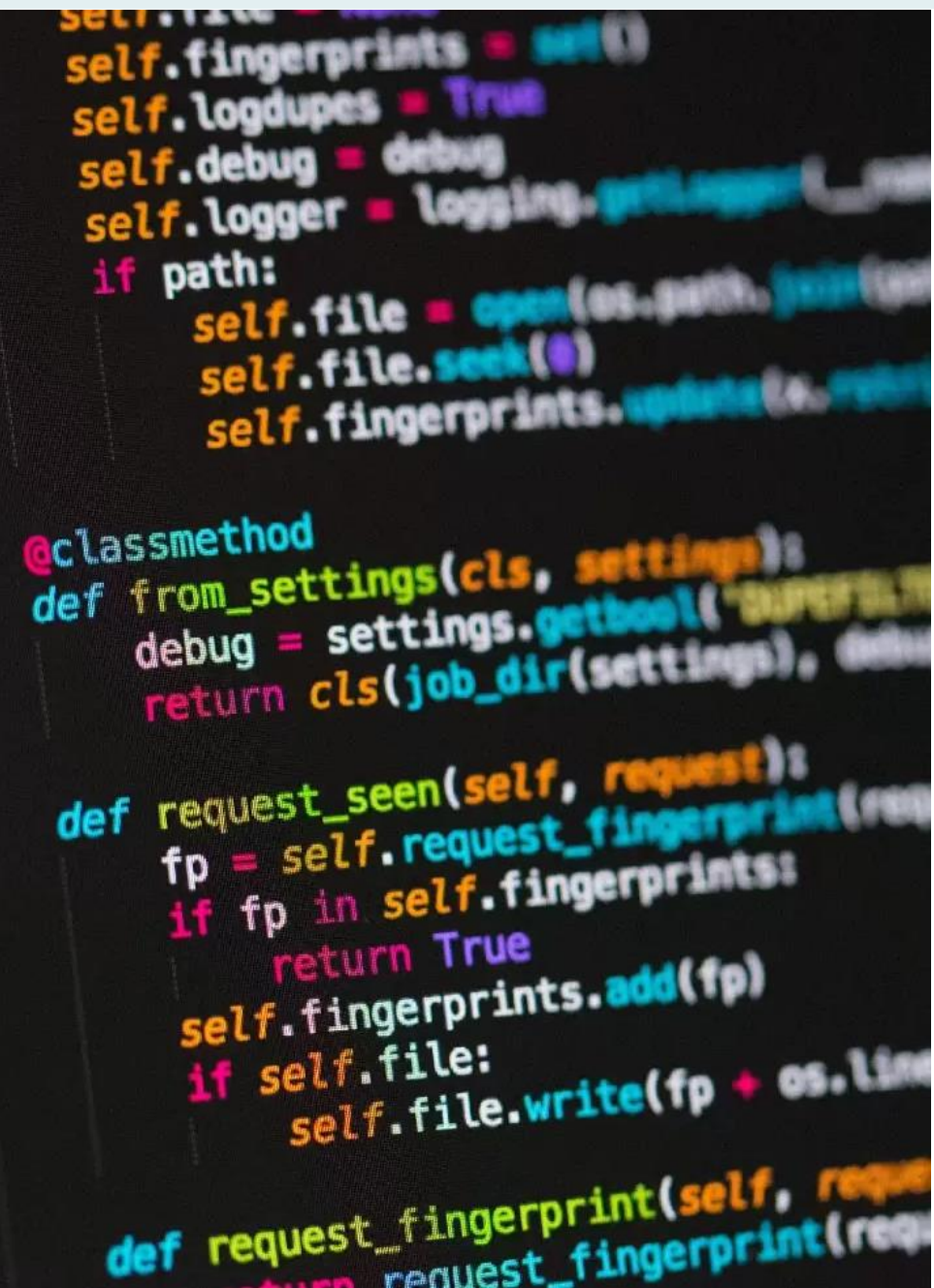
Tema 2: Python em outros paradigmas.

- 2.1 Linguagem funcional no Python
- 2.2 Computação concorrente em Python
- 2.3 Desenvolvimento Web com Python
- 2.4 Ciência de Dados em Python



# — Objetivos - Agora

1. Linguagem funcional no Python
2. Computação concorrente em Python
3. Desenvolvimento Web com Python
4. Ciência de Dados em Python



```
self.file = None
self.fingerprints = set()
self.logdups = True
self.debug = debug
self.logger = logging.getLogger(__name__)
if path:
    self.file = open(os.path.join(path, 'fingerprint.log'), 'a')
    self.file.seek(0)
    self.fingerprints.update(request_fingerprint(request))

    @classmethod
    def from_settings(cls, settings):
        debug = settings.getbool('debug')
        return cls(job_dir(settings), debug)

    def request_seen(self, request):
        fp = self.request_fingerprint(request)
        if fp in self.fingerprints:
            return True
        self.fingerprints.add(fp)
        if self.file:
            self.file.write(fp + os.linesep)

    def request_fingerprint(self, request):
        return request_fingerprint(request)
```

# — Linguagem Funcional no Python

Python oferece suporte ao paradigma funcional, permitindo o uso de funções de primeira classe, funções de ordem superior, e técnicas como mapeamento, filtragem, e redução.

- **Tópicos principais:**

- **Funções de alta ordem:** Funções que recebem outras funções como argumentos (ex.: `map`, `filter`, `reduce`).
- **Funções `lambda`:** Funções anônimas para expressões simples.
- **Imutabilidade:** Preferência por dados imutáveis em paradigmas funcionais.



```
from functools import reduce

def uso_reduce(lista: list) -> int:
    """
    Recebe uma lista de inteiros e retorna a soma de todos os elementos da lista.
    args: lista: uma lista de inteiros
    return: a soma de todos os elementos da lista
    """
    return reduce(lambda x, y: x + y, lista)

def uso_map(lista: list) -> list:
    """
    Recebe uma lista de inteiros e retorna uma nova lista com os quadrados dos
    elementos da lista.
    args: lista: uma lista de inteiros
    return: uma nova lista com os quadrados dos elementos da lista
    """
    return list(map(lambda x: x**2, lista))
```

```

def uso_filter(lista: list) -> list:
    '''
    Recebe uma lista de inteiros e retorna uma nova lista com os elementos pares da lista.
    args: lista: uma lista de inteiros
    return: uma nova lista com os elementos pares da lista
    '''
    return list(filter(lambda x: x % 2 == 0, lista))

if __name__ == '__main__':
    lista = [1, 2, 3, 4, 5]
    soma = uso_reduce(lista) # Resultado: 15
    lista_quadrados = uso_map(lista) # Resultado: [1, 4, 9, 16, 25]
    pares = uso_filter(lista) # Resultado: [2, 4]
    print(f'Resultado do uso_reduce: {soma}\nResultado do uso_map: {lista_quadrados}\nResultado do uso_filter: {pares}')

```

```

PS D:\Estacio\2024_02\codigos_2024_2> python
Resultado do uso_reduce: 15
Resultado do uso_map: [1, 4, 9, 16, 25]
Resultado do uso_filter: [2, 4]
PS D:\Estacio\2024_02\codigos_2024_2>

```

# — Computação Concorrente em Python

Python permite a execução de tarefas concorrentes através de threads, processos e a biblioteca **asyncio**, apesar da Global Interpreter Lock (GIL) limitar o paralelismo verdadeiro em threads.

- **Tópicos principais:**

- **Módulo `threading`**: Para tarefas leves que precisam rodar em paralelo.
- **Módulo `multiprocessing`**: Para tarefas que exigem verdadeiro paralelismo (bom para CPU-bound).
- **`asyncio`**: Programação assíncrona com `async` e `await` para operações I/O-bound.

```
import asyncio

async def tarefa(numero: int) -> None:
    '''
    Função assíncrona para executar uma tarefa
    Args:
        numero (int): Numero da tarefa
    Returns:
        None
    '''
    print(f"Inicio da tarefa {numero}")
    await asyncio.sleep(1)
    print(f"Fim da tarefa {numero}")

async def main() -> None:
    '''
    Função principal para executar as tarefas
    Args:
        None
    Returns:
        None
    '''
    await asyncio.gather(tarefa(1), tarefa(2), tarefa(3))

if __name__ == "__main__":
    asyncio.run(main())
```

[https://docs.python.org/pt-br/3.12/reference/compound\\_stmts.html#index-49](https://docs.python.org/pt-br/3.12/reference/compound_stmts.html#index-49)



```
import threading
import time

def tarefa(numero: int) -> None:
    '''
    Função para executar uma tarefa em uma thread
    '''
    print(f"Início da tarefa {numero}")
    time.sleep(1)
    print(f"Fim da tarefa {numero}")

def main() -> None:
    '''
    Função principal para executar tarefas usando threading
    '''
    threads = []
    for i in range(1, 4):
        t = threading.Thread(target=tarefa, args=(i,))
        threads.append(t)
        t.start()
    for t in threads:
        t.join()

if __name__ == "__main__":
    main()
```

<https://docs.python.org/3/library/threading.html>

```
import multiprocessing
```

```
import time
```

```
def tarefa(numero: int) -> None:
```

```
    '''
```

```
    Função para executar uma tarefa em um processo separado
```

```
    '''
```

```
    print(f"Início da tarefa {numero}")
```

```
    time.sleep(1)
```

```
    print(f"Fim da tarefa {numero}")
```

```
def main() -> None:
```

```
    '''
```

```
    Função principal para executar tarefas usando multiprocessing
```

```
    '''
```

```
    processos = []
```

```
    for i in range(1, 4):
```

```
        p = multiprocessing.Process(target=tarefa, args=(i,))
```

```
        processos.append(p)
```

```
        p.start()
```

```
    for p in processos:
```

```
        p.join()
```

```
if __name__ == "__main__":
```

```
    main()
```

<https://docs.python.org/3/library/multiprocessing.html>

# — Desenvolvimento Web com Python

Python é amplamente usado para desenvolvimento web, especialmente com frameworks como Django e Flask, que permitem criar APIs e aplicações web completas.

- **Tópicos principais:**

- **Flask:** Framework minimalista para APIs e aplicações pequenas.

<https://flask.palletsprojects.com/en/stable/>

- **Django:** Framework completo, com ORM, sistema de autenticação, e administração embutidos.

<https://www.djangoproject.com>

- **FastAPI:** Framework moderno para APIs rápidas e assíncronas.

<https://fastapi.tiangolo.com>

# pip install flask

```
from flask import Flask, jsonify
app = Flask(__name__)
@app.route('/')
def home() -> str:
    '''
    Retorna uma mensagem de boas-vindas.
    args:
        None
    return:
        uma mensagem de boas-vindas
    '''
    return jsonify(message="Oi mamãe e papai!")
if __name__ == '__main__':
    app.run(debug=True)
```

<https://flask.palletsprojects.com/en/stable/api/#flask.json.jsonify>

# `pip install fastapi uvicorn`

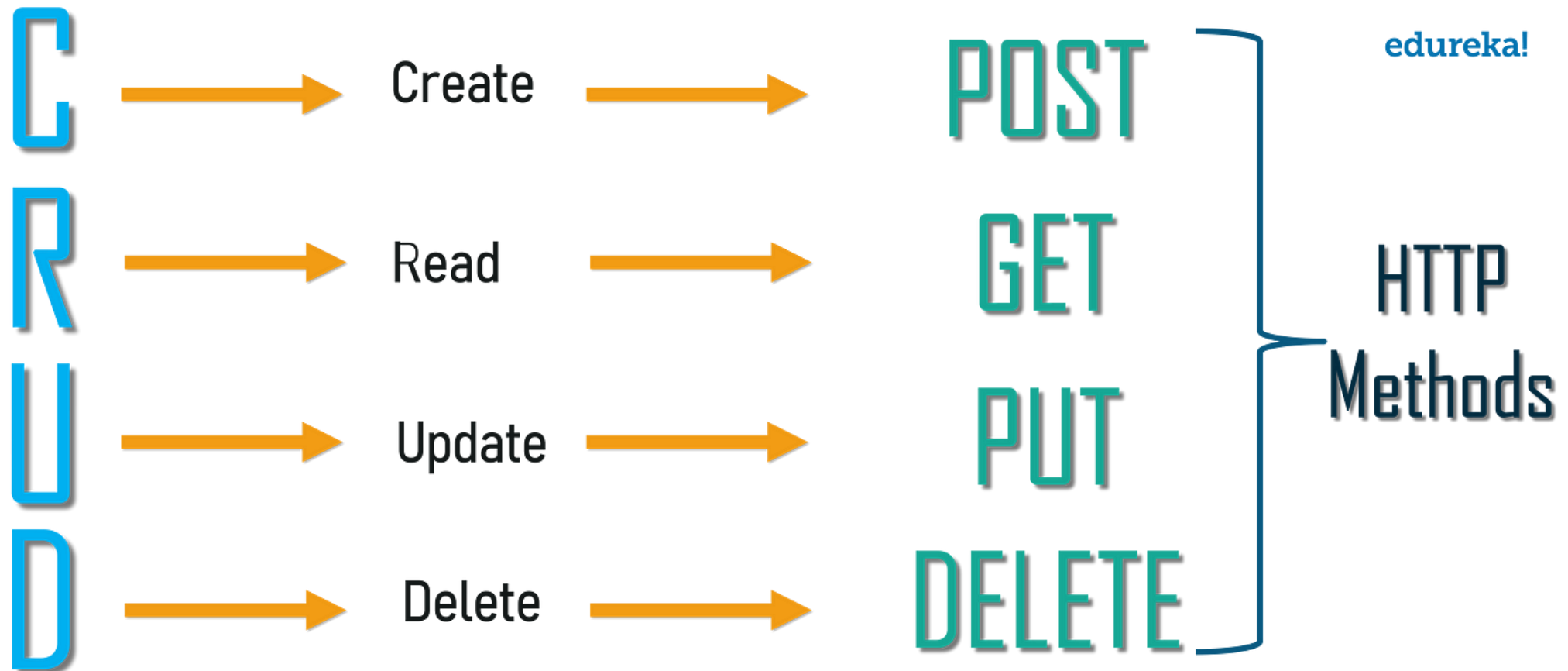
```
from fastapi import FastAPI
from fastapi.responses import JSONResponse

app = FastAPI()
@app.get("/")
async def home() -> JSONResponse:
    """
    Retorna uma mensagem de boas-vindas.
    Args:
        None
    Returns:
        JSONResponse: Mensagem de boas-vindas
    """
    return JSONResponse(content={"message": "Oi mamãe e papai!"})
```

`uvicorn nome_do_arquivo:app --reload`

## Criação de endpoints RESTful para operações **CRUD**:

- **Create:** POST /items
- **Read:** GET /items, GET /items/<id>
- **Update:** PUT /items/<id>
- **Delete:** DELETE /items/<id>





```
from flask import Flask, jsonify, request

app = Flask(__name__)

# Dados fictícios (em memória) para demonstração
items = [
    {"id": 1, "nome": "Item 1", "preco": 10.0},
    {"id": 2, "nome": "Item 2", "preco": 20.0},
]

@app.route('/items', methods=['GET'])
def get_items():
    return jsonify(items), 200

@app.route('/items/<int:item_id>', methods=['GET'])
def get_item(item_id):
    item = next((item for item in items if item["id"] == item_id), None)
    if item:
        return jsonify(item), 200
    else:
        return jsonify({"erro": "Item não encontrado"}), 404
```

```
@app.route('/items', methods=['POST'])
def create_item():
    data = request.json
    new_item = {
        "id": len(items) + 1,
        "nome": data["nome"],
        "preco": data["preco"]
    }
    items.append(new_item)
    return jsonify(new_item), 201

@app.route('/items/<int:item_id>', methods=['PUT'])
def update_item(item_id):
    data = request.json
    item = next((item for item in items if item["id"] == item_id), None)
    if item:
        item["nome"] = data["nome"]
        item["preco"] = data["preco"]
        return jsonify(item), 200
    else:
        return jsonify({"error": "Item não encontrado"}), 404
```

```
@app.route('/items/<int:item_id>', methods=['DELETE'])
def delete_item(item_id):
    global items
    items = [item for item in items if item["id"] != item_id]
    return jsonify({"message": "Item deletado"}), 200

if __name__ == '__main__':
    app.run(debug=True)
```

<https://insomnia.rest>



<https://www.thunderclient.com>



<https://www.postman.com>



# — Ciência de Dados em Python

Python é uma das linguagens mais usadas em ciência de dados, graças a bibliotecas robustas para manipulação de dados, visualização e machine learning.

- **Tópicos principais:**

- **Pandas:** Manipulação de dados em DataFrames.

<https://pandas.pydata.org>

- **NumPy:** Operações matemáticas e manipulação de arrays.

<https://numpy.org>

- **Matplotlib e Seaborn:** Framework moderno para APIs rápidas e assíncronas.

<https://matplotlib.org> & <https://seaborn.pydata.org>

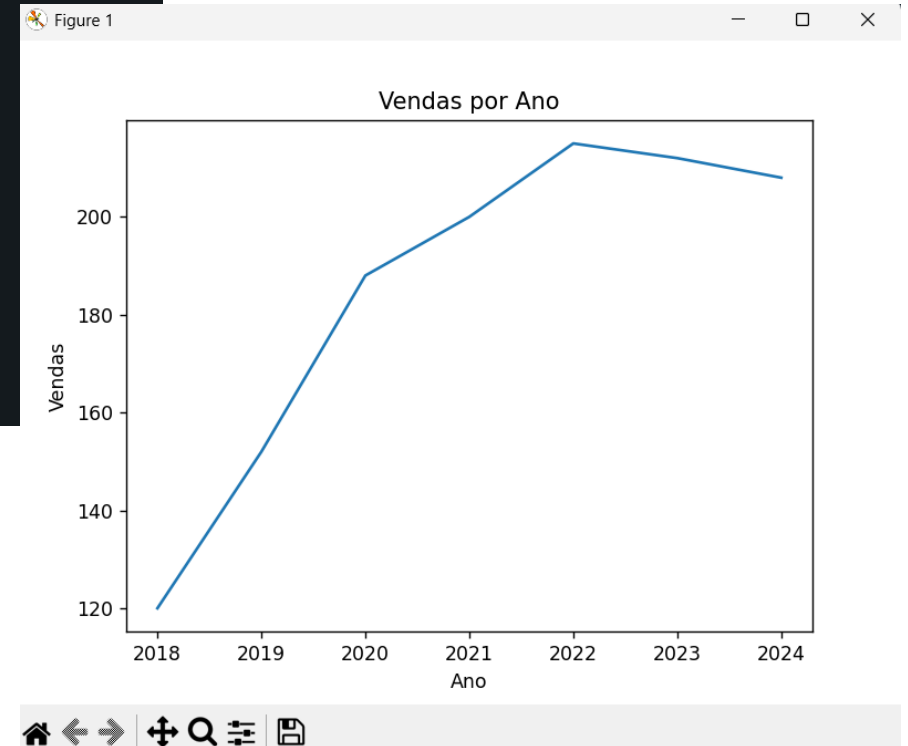
- **Scikit-Learn:** Ferramentas para machine learning.

<https://scikit-learn.org>

```
import pandas as pd
import matplotlib.pyplot as plt
# Criação de um DataFrame
dados = {'Ano': [2018, 2019, 2020, 2021, 2022, 2023, 2024],
        'Vendas': [120, 152, 188, 200, 215, 212, 208]}
df = pd.DataFrame(dados)

# Plotando os dados
plt.plot(df['Ano'], df['Vendas'])
plt.xlabel('Ano')
plt.ylabel('Vendas')
plt.title('Vendas por Ano')
plt.show()
```

pip install pandas  
pip install matplotlib



```
import numpy as np

# Criação de um array NumPy
dados = np.array([10, 20, 30, 40, 50])

# Operações básicas
soma = np.sum(dados)
media = np.mean(dados)
desvio_padrao = np.std(dados)

print("Dados:", dados)
print("Soma:", soma)
print("Média:", media)
print("Desvio Padrão:", desvio_padrao)
```







pip install seaborn

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

```
# Criação de um DataFrame
```

```
dados = {
    'Ano': [2018, 2019, 2020, 2021, 2022, 2023, 2024],
    'Vendas': [120, 152, 188, 200, 215, 212, 208]
}
df = pd.DataFrame(dados)
```

```
# Estilo do seaborn
```

```
sns.set_theme(style="whitegrid")
```

```
# Gráfico de linha
```

```
plt.figure(figsize=(8, 4))
sns.lineplot(x='Ano', y='Vendas', data=df, marker='o')
```

```
plt.title('Vendas por Ano')
plt.xlabel('Ano')
plt.ylabel('Vendas')
plt.tight_layout()
plt.show()
```



# pandas



# plotly

```
import pandas as pd
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
```

```
# Dados fictícios
```

```
dados = {
    'Ano': [2018, 2019, 2020, 2021, 2022, 2023, 2024],
    'Vendas': [120, 152, 188, 200, 215, 212, 208]
}
```

```
df = pd.DataFrame(dados)
```

```
# Preparar dados para o modelo
```

```
X = df[['Ano']]
y = df['Vendas']
```

```
# Criar e treinar modelo
```

```
modelo = LinearRegression()
modelo.fit(X, y)
```

pip install scikit-learn



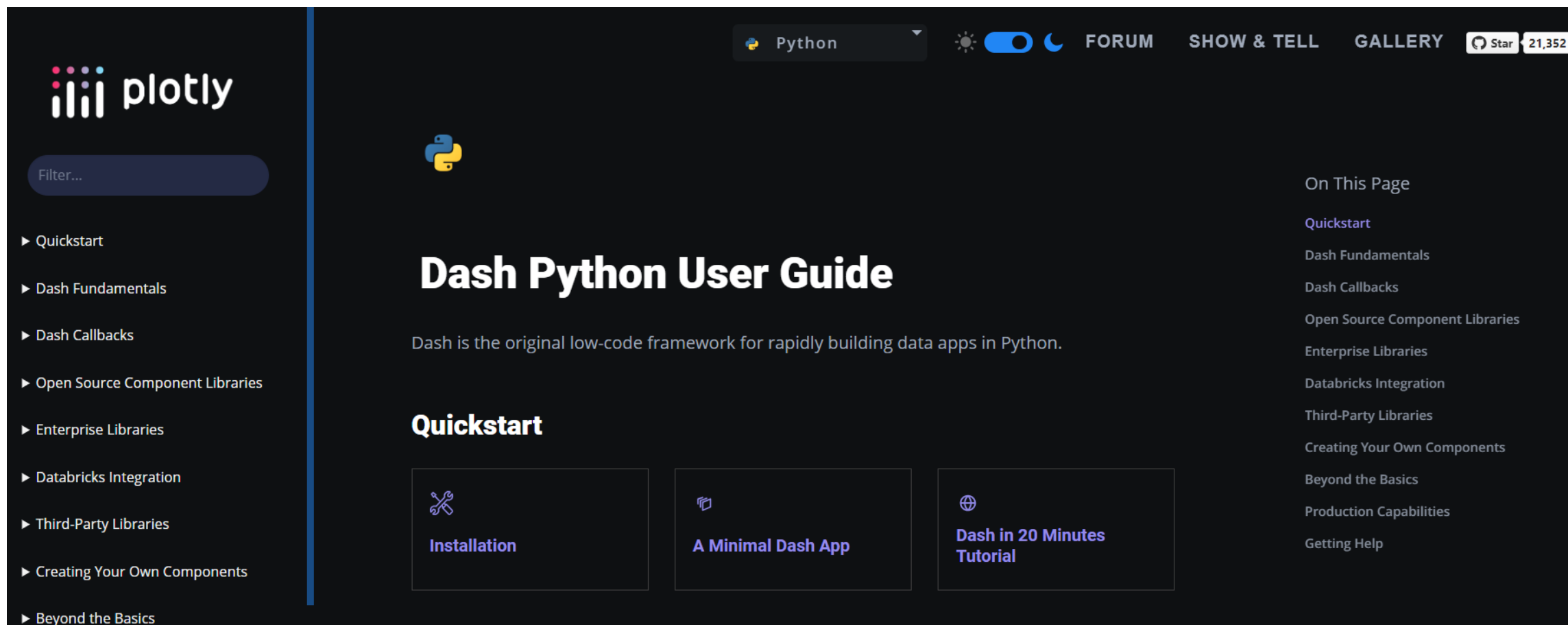
```
df['Previsão'] = modelo.predict(X)
```

```
# Visualização
```

```
plt.figure(figsize=(8, 4))
plt.plot(df['Ano'], df['Vendas'], label='Real', marker='o')
plt.plot(df['Ano'], df['Previsão'], label='Regressão
Linear', linestyle='--')
plt.xlabel('Ano')
plt.ylabel('Vendas')
plt.title('Vendas Reais vs. Previsão')
plt.legend()
plt.tight_layout()
plt.show()
```

# — Visualizador de dados

<https://dash.plotly.com>



The screenshot shows the Dash Python User Guide website. The header includes the Plotly logo, a search bar, and navigation links for Python, FORUM, SHOW & TELL, GALLERY, and a Star button with 21,352 stars. The main content area features the Python logo, the title 'Dash Python User Guide', and a description: 'Dash is the original low-code framework for rapidly building data apps in Python.' Below this is a 'Quickstart' section with three cards: 'Installation', 'A Minimal Dash App', and 'Dash in 20 Minutes Tutorial'. The left sidebar contains a list of navigation links, and the right sidebar contains a list of links for 'On This Page'.

**plotly**

Filter...

- ▶ Quickstart
- ▶ Dash Fundamentals
- ▶ Dash Callbacks
- ▶ Open Source Component Libraries
- ▶ Enterprise Libraries
- ▶ Databricks Integration
- ▶ Third-Party Libraries
- ▶ Creating Your Own Components
- ▶ Beyond the Basics

**Python**

**FORUM** **SHOW & TELL** **GALLERY** **Star 21,352**

**Dash Python User Guide**

Dash is the original low-code framework for rapidly building data apps in Python.

**Quickstart**

- Installation**
- A Minimal Dash App**
- Dash in 20 Minutes Tutorial**

**On This Page**

- Quickstart**
- Dash Fundamentals
- Dash Callbacks
- Open Source Component Libraries
- Enterprise Libraries
- Databricks Integration
- Third-Party Libraries
- Creating Your Own Components
- Beyond the Basics
- Production Capabilities
- Getting Help

# — Visualizador de dados

<https://streamlit.io>

📌 See what's new in Release 1.39 and register for the Quarterly Showcase on Oct. 30!



[Cloud](#)

[Gallery](#)

[Components](#)

[Generative AI](#)

[Community](#)

[Docs](#)

[Blog](#)

[Sign in](#)

[Sign up](#)

# A faster way to build and share data apps

Streamlit turns data scripts into shareable web apps in minutes.

All in pure Python. No front-end experience required.

[Try Streamlit now](#)

[Deploy on Community Cloud \(it's free!\)](#)

15

3

Search

Sign In

Register

MUHAMMAD TALHA AWAN · UPDATED 2 YEARS AGO

▲

52

<>

Code

Download

World Export & Import Dataset (1989 - 2023)

34 Year Export and Import of Countries, Territories and Overall World in USD\$

Data Card


Code (4)

Discussion (1)

Suggestions (0)

About Dataset

Usability ⓘ  
10.00



```
import streamlit as st
import pandas as pd
import plotly.express as px

# Carregar o dataset
data = pd.read_csv("34_years_world_export_import_dataset.csv")

# Padronizar nomes das colunas
data.columns = data.columns.str.strip()

# Renomear colunas para português
data.rename(columns={
    'Partner Name': 'País',
    'Year': 'Ano',
    'Export (US$ Thousand)': 'Exportação',
    'Import (US$ Thousand)': 'Importação'
}, inplace=True)

# Título do app
st.title("Análise de Exportações e Importações Mundiais")
```



```
# Barra lateral de filtros
st.sidebar.header("Filtros")
países_selecionados = st.sidebar.multiselect(
    "Selecione os Países",
    options=data["País"].unique(),
    default=data["País"].unique()[:3]
)

ano_selecionado = st.sidebar.selectbox(
    "Selecione o Ano",
    options=sorted(data["Ano"].unique(), reverse=True)
)

valor_tipo = st.sidebar.radio(
    "Tipo de Valor",
    options=["Exportação", "Importação"]
)
```

```
# Filtrar os dados
dados_filtrados = data[
    (data["País"].isin(países_selecionados)) &
    (data["Ano"] == ano_selecionado)
].copy()

# Exibir tabela
st.write(f"### {valor_tipo} em {ano_selecionado} para os países selecionados")
st.dataframe(dados_filtrados[["País", "Ano", valor_tipo]])

# Gráfico de barras
fig_bar = px.bar(
    dados_filtrados,
    x="País",
    y=valor_tipo,
    color="País",
    title=f"{valor_tipo} por País - {ano_selecionado}",
    labels={valor_tipo: "Valor (US$ Milhares)"}
)
st.plotly_chart(fig_bar)
```

```
# Gráfico de linha (série histórica)
dados_historico = data[
    (data["País"].isin(países_selecionados))
].groupby(["Ano", "País"])[valor_tipo].sum().reset_index()

fig_line = px.line(
    dados_historico,
    x="Ano",
    y=valor_tipo,
    color="País",
    title=f"Evolução Histórica de {valor_tipo} por País",
    markers=True,
    labels={valor_tipo: "Valor (US$ Milhares)"}
)
st.plotly_chart(fig_line)

# Rodapé
st.write("Aplicação desenvolvida com Streamlit, Pandas e Plotly.")
```



https://www.kaggle.com/c/titanic/data?select=train.csv

# kaggle

≡ kaggle

+ Create

Home

Competitions

Datasets

Models

<> Code

Discussions

Learn

More

Your Work

VIEWED

Search



KAGGLE · GETTING STARTED PREDICTION COMPETITION · ONGOING

Submit Prediction

...

## Titanic - Machine Learning from Disaster

Start here! Predict survival on the Titanic and get familiar with ML basics



Overview Data Code Models Discussion Leaderboard Rules

### Dataset Description

#### Overview

The data has been split into two groups:

- training set (train.csv)

(test set (test.csv))

#### Files

3 files

#### Size

93.08 kB

#### Type

CSV

# — Controlador – Geração de API

<https://flask.palletsprojects.com/en/stable/>

## Project Links

[Donate](#)

[PyPI Releases](#)

[Source Code](#)

[Issue Tracker](#)

[Chat](#)

## Contents

[Welcome to Flask](#)

[User's Guide](#)

[API Reference](#)

[Additional Notes](#)

## Quick search



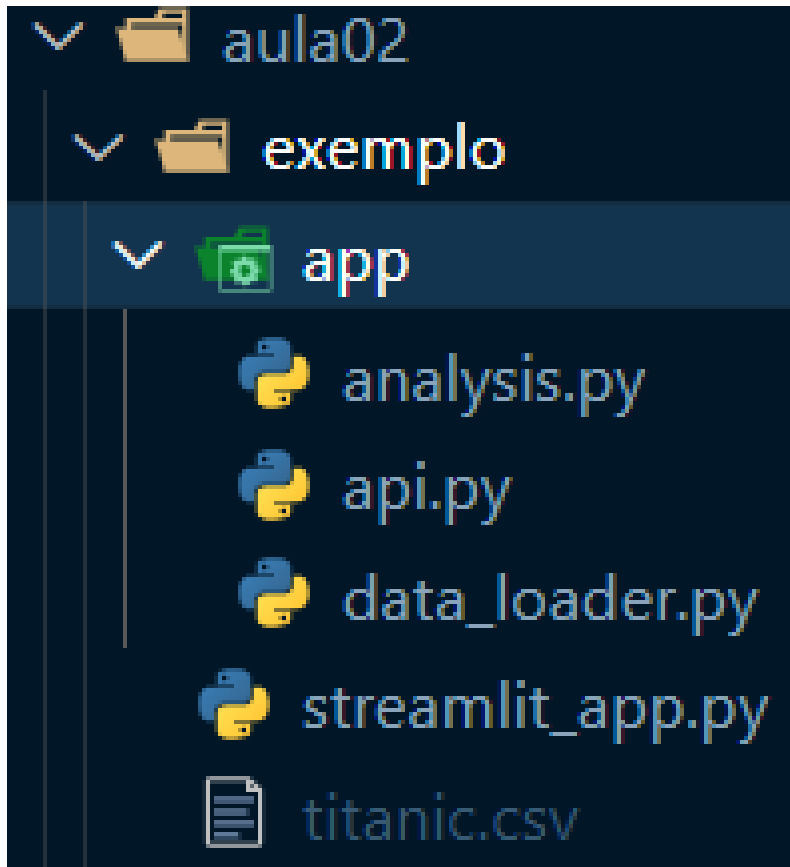
# Flask

Welcome to Flask's documentation. Get started with [Installation](#) and then get an overview with the [Quickstart](#). There is also a more detailed [Tutorial](#) that shows how to create a small but complete application with Flask. Common patterns are described in the [Patterns for Flask](#) section. The rest of the docs describe each component of Flask in detail, with a full reference in the [API](#) section.

Flask depends on the [Werkzeug](#) WSGI toolkit, the [Jinja](#) template engine, and the [Click](#) CLI toolkit. Be sure to check their documentation as well as Flask's when looking for information.

## User's Guide





# Classe para realizar análises nos dados

# API Flask

# Classe para carregar e pré-processar os dados

# Interface Streamlit

# Dados do Titanic (download do Kaggle)

```
import pandas as pd

class DataLoader:
    def __init__(self, file_path):
        self.file_path = file_path
        self.data = None

    def load_data(self):
        self.data = pd.read_csv(self.file_path)
        return self.data

    def preprocess_data(self):
        # Exemplo de pré-processamento
        self.data = self.data.copy()
        self.data['Age'] = self.data['Age'].fillna(self.data['Age'].mean())
        self.data['Fare'] = self.data['Fare'].fillna(self.data['Fare'].mean())
        self.data = self.data.dropna(subset=['Embarked'])
        return self.data
```

```
import pandas as pd
import matplotlib.pyplot as plt

class Analysis:
    def __init__(self, data):
        self.data = data

    def survival_rate(self):
        return self.data['Survived'].mean()

    def plot_age_distribution(self):
        fig, ax = plt.subplots()
        ax.hist(self.data['Age'], bins=20, color='skyblue', edgecolor='black')
        ax.set_title('Distribuição de Idade')
        ax.set_xlabel('Idade')
        ax.set_ylabel('Frequência')
        return fig
```

```
from flask import Flask, jsonify
from data_loader import DataLoader
from analysis import Analysis
```

```
app = Flask(__name__)
```

```
# Carregar e pré-processar os dados
data_loader = DataLoader('titanic.csv')
data = data_loader.load_data()
data_loader.preprocess_data()
analysis = Analysis(data)
```

```
@app.route('/api/survival_rate', methods=['GET'])
def survival_rate():
    rate = analysis.survival_rate()
    return jsonify({"survival_rate": rate})
```

```
@app.route('/api/passenger/<int:id>', methods=['GET'])
def passenger(id):
    passenger_data = data.loc[data['PassengerId'] == id].to_dict(orient='records')
    return jsonify(passenger_data)
```

```
if __name__ == '__main__':
    app.run(debug=True)
```

```
# streamlit_app.py
import streamlit as st
import pandas as pd
import requests
from app.data_loader import DataLoader
from app.analysis import Analysis

# Carregar e pré-processar os dados
data_loader = DataLoader('titanic.csv')
data = data_loader.load_data()
data_loader.preprocess_data()
analysis = Analysis(data)

# Título
st.title("Análise do Titanic com Flask e Streamlit")

# Visualizar dados
st.header("Dados do Titanic")
st.write(data.head())
```

```
# Taxa de sobrevivência
st.header("Taxa de Sobrevivência")
rate =
requests.get("http://127.0.0.1:5000/api/survival_rate").json()["survival_rate"]
st.write(f"Taxa de sobrevivência: {rate:.2%}")

# Filtro por passageiro
st.header("Informações do Passageiro")
passenger_id = st.number_input("ID do Passageiro", min_value=1,
max_value=int(data['PassengerId'].max()), value=1)
passenger_data =
requests.get(f"http://127.0.0.1:5000/api/passenger/{passenger_id}").json()
st.write(passenger_data)

# Distribuição de idade
st.header("Distribuição de Idade dos Passageiros")
fig = analysis.plot_age_distribution()
st.pyplot(fig)
```

# Terminal 1

```
PS D:\Estacio\2024_02\codigos_2024_2\python\pos\aula02\exemplo> python app/api.py
```

```
* Serving Flask app 'api'
```

```
* Debug mode: on
```

```
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
```

```
* Running on http://127.0.0.1:5000
```

```
Press CTRL+C to quit
```

```
* Restarting with watchdog (windowsapi)
```

```
* Debugger is active!
```

```
* Debugger PIN: 729-657-499
```

```
127.0.0.1 - - [08/Nov/2024 20:23:55] "GET /api/survival_rate HTTP/1.1" 200 -
```

# Terminal 2

```
PS D:\Estacio\2024_02\codigos_2024_2\python\pos\aula02\exemplo> streamlit run streamlit_app.py
```

You can now view your Streamlit app in your browser.

Local URL: <http://localhost:8501>

Network URL: <http://172.16.2.5:8501>

# Análise do Titanic com Flask e Streamlit

## Dados do Titanic

	PassengerId	Survived	Pclass	Name	Sex	Age	S
0	1	0	3	Braund, Mr. Owen Harris	male	22	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Thayer)	female	38	
2	3	1	3	Heikkinen, Miss. Laina	female	26	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35	
4	5	0	3	Allen, Mr. William Henry	male	35	

## Taxa de Sobrevivência

Taxa de sobrevivência: 38.38%





# kaggle

# pandas



# Streamlit

# plotly

## Supermarket Sales Analysis

Python · Supermarket sales

Notebook Input Output Logs Comments (0)

### Input Data

supermarket\_sales - Sheet1.csv (131.53 kB)



Detail Compact Column

10 of 17 columns

Input (131.53 kB)

Data Sources

Supermarket sales

supermarket\_sales - St

<https://www.kaggle.com/code/kareemadel20/supermarket-sales-analysis>

<https://pandas.pydata.org/>

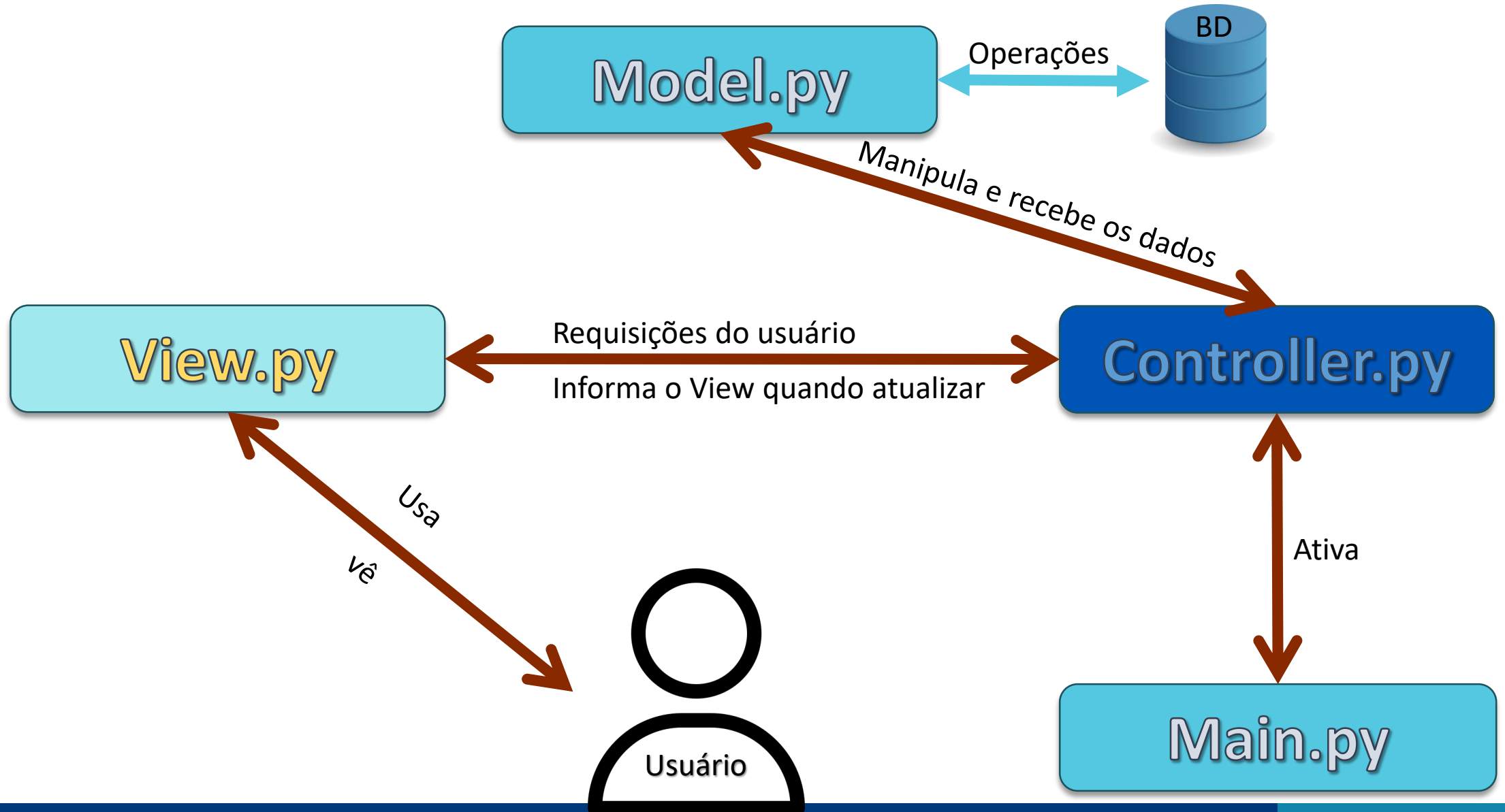
<https://streamlit.io/>

<https://plotly.com/>

<https://plotly.com/python/plotly-express/>



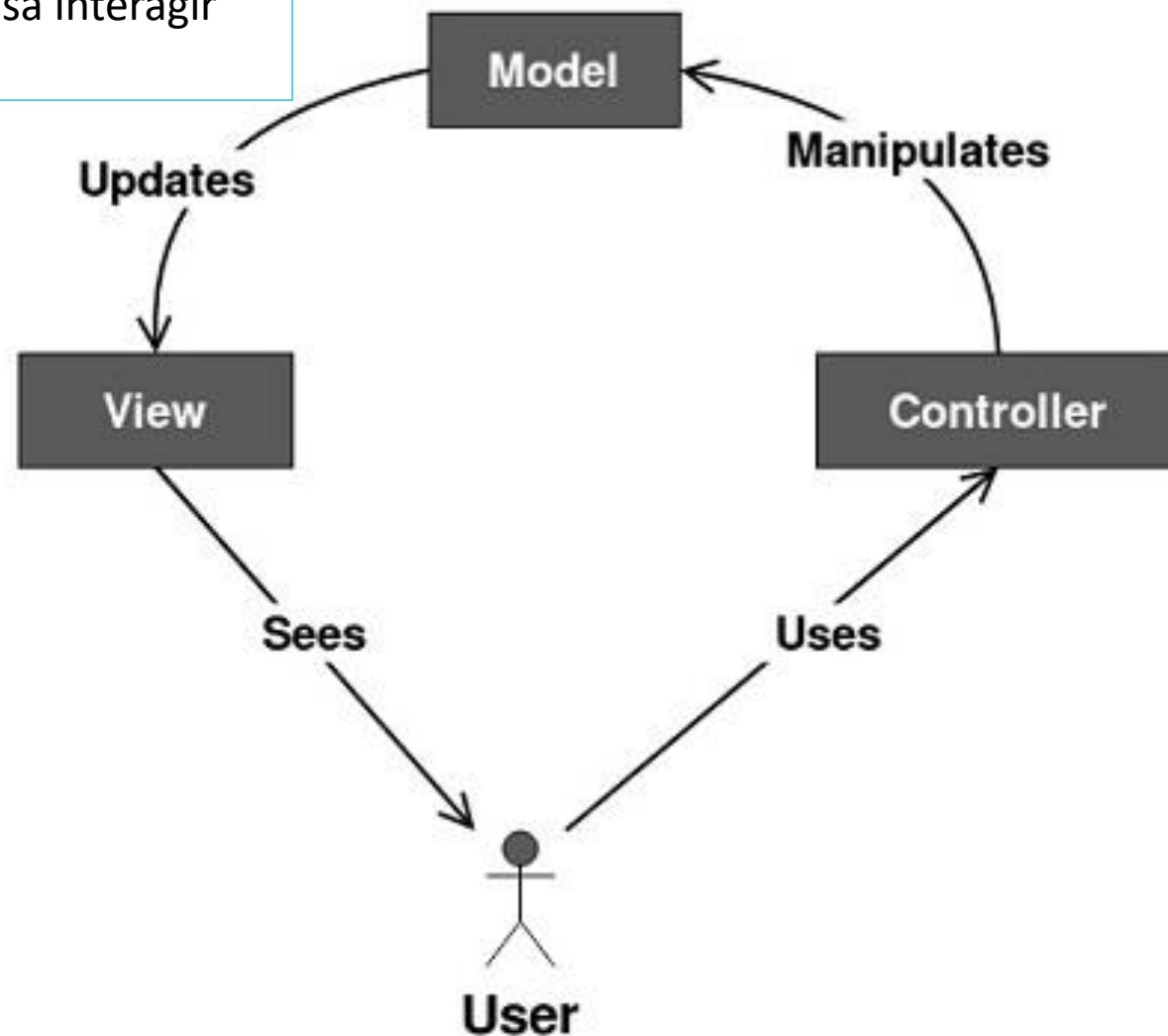
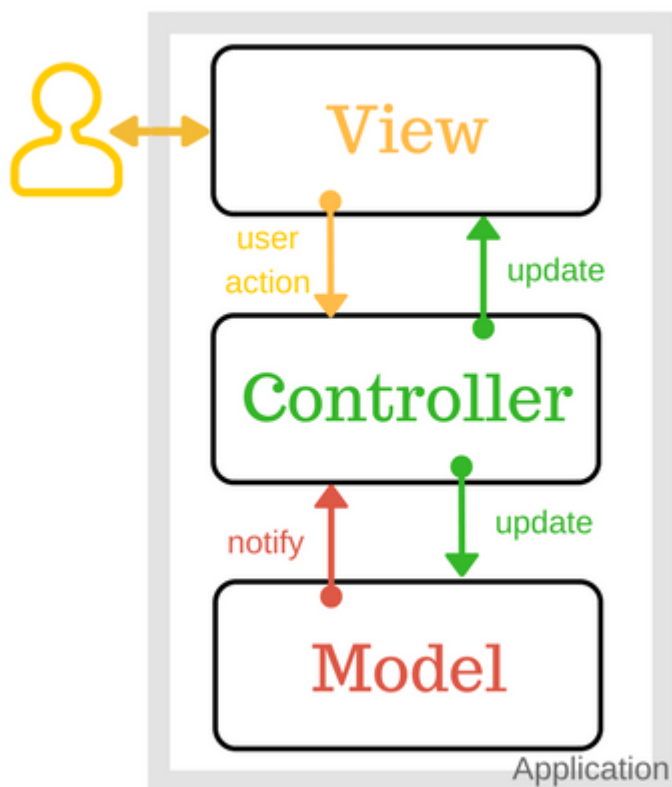
## Padrão arquitetônico Model View Controller (MVC)





# MVC

O padrão de design final é MVC - ou o padrão model-view-controller. Como você verá, é uma ótima escolha sempre que um usuário precisa interagir frequentemente com o sistema que você está codificando!

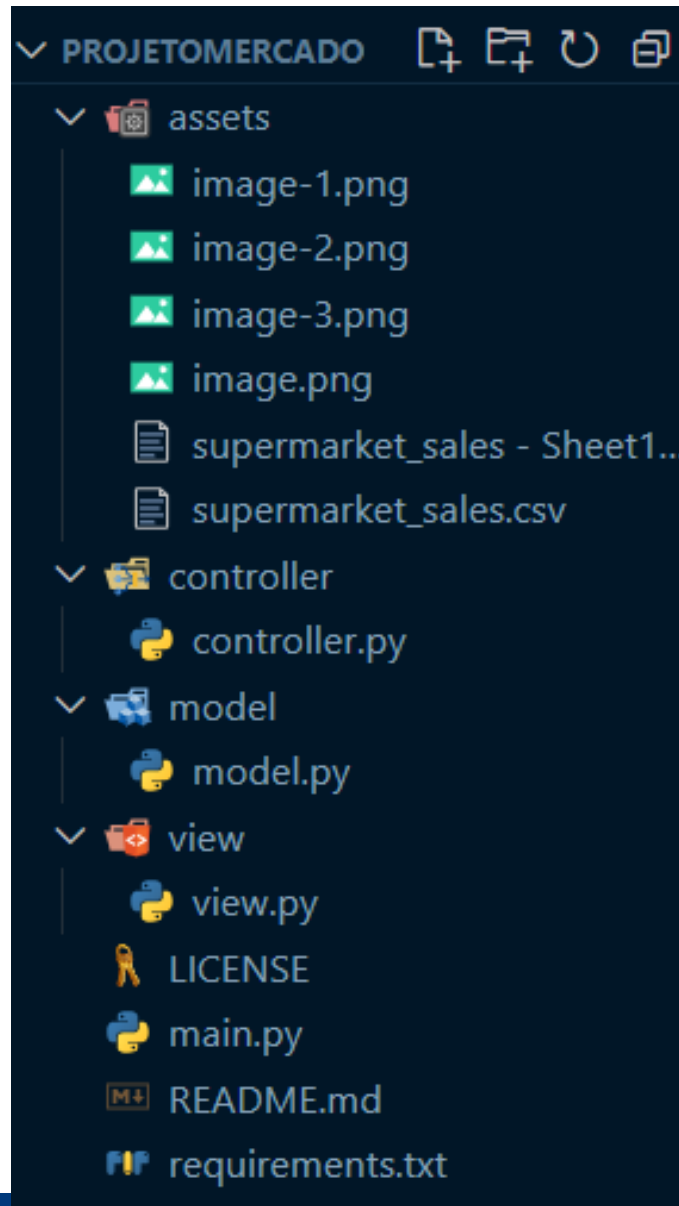




 pandas

 Streamlit

 plotly





pandas



Streamlit



plotly

```
import pandas as pd

class ModeloSupermercado:
    def __init__(self, caminho_dados: str):
        try:
            self.df = pd.read_csv(caminho_dados, sep=";", decimal=".", thousands=".")
        except FileNotFoundError:
            raise FileNotFoundError(f"Arquivo não encontrado em {caminho_dados}")
        except pd.errors.ParserError:
            raise ValueError(f"Erro ao processar o arquivo {caminho_dados}")
        self.df["Date"] = pd.to_datetime(self.df["Date"], errors="coerce")
        self.df = self.df.dropna(subset=["Date"])
        self.df = self.df.sort_values("Date")

    def criar_coluna_mes(self) -> None:
        self.df["Mes"] = self.df["Date"].dt.strftime("%Y-%m")
```



pandas



Streamlit



plotly

```
def obter_dados_por_mes(self, mes: str) -> pd.DataFrame:
    return self.df[self.df["Mes"] == mes]

def calcular_total_do_mes(self, mes: str, coluna_alvo: str = "Total") -> float:
    df_mes = self.obter_dados_por_mes(mes)
    total_mes = df_mes[coluna_alvo].sum()
    return total_mes

def calcular_total_cogs(self, mes: str) -> float:
    df_mes = self.obter_dados_por_mes(mes)
    total_cogs = df_mes["cogs"].sum()
    return total_cogs

def calcular_margem_de_lucro_bruta(self, mes: str) -> float:
    total_do_mes = self.calcular_total_do_mes(mes)
    total_cogs = self.calcular_total_cogs(mes)
    if total_do_mes == 0:
        return 0.0
    margem_de_lucro_bruta = ((total_do_mes - total_cogs) / total_do_mes) * 100
    return margem_de_lucro_bruta
```



```
import streamlit as st
import plotly.express as px
import pandas as pd

class VisualizacaoSupermercado:
    def exibir_pagina(
        self,
        df_filtrado: pd.DataFrame,
        total_do_mes: float,
        margem_de_lucro_bruta: float,
        total_cogs: float,
    ) -> None:
        col1, col2 = st.columns(2)
        col3 = st.columns(1)[0]
        col4, col5, col6 = st.columns(3)
        col7 = st.columns(1)[0]
```

<https://docs.streamlit.io/library/api-reference/layout/st.columns>

<https://plotly.com/python/plotly-express/>



pandas



Streamlit



plotly

```
fig_data = px.bar(
    df_filtrado, x="Date", y="Total", color="City", title="Faturamento por dia"
)
fig_data.update_xaxes(title="Data")
col1.plotly_chart(fig_data, use_container_width=True)

fig_produto = px.bar(
    df_filtrado,
    x="Date",
    y="Product line",
    color="City",
    title="Faturamento por tipo de produto",
    orientation="h",
)
fig_produto.update_xaxes(title="Data")
fig_produto.update_yaxes(title="Linha do produto")
col2.plotly_chart(fig_produto, use_container_width=True)
```





pandas



Streamlit



plotly

```
fig_crescimento = px.line(  
    df_filtrado,  
    x="Total",  
    y="cogs",  
    color="Branch",  
    symbol="Branch",  
    title="Custo dos bens vendidos",  
)  
fig_crescimento.update_xaxes(title="Total")  
fig_crescimento.update_yaxes(title="Custo dos bens vendidos")  
col3.plotly_chart(fig_crescimento, use_container_width=True)
```

<https://plotly.com/python/line-charts/>



pandas



Streamlit



plotly

```
cidade_total = df_filtrado.groupby("City")[["Total"]].sum().reset_index()
fig_cidade = px.bar(
    cidade_total, x="City", y="Total", title="Faturamento por filial"
)
fig_cidade.update_xaxes(title="Cidade")
col4.plotly_chart(fig_cidade, use_container_width=True)
```



```
fig_tipo = px.pie(  
    df_filtrado,  
    values="Total",  
    names="Payment",  
    title="Faturamento por forma de pagamento",  
)  
col5.plotly_chart(fig_tipo, use_container_width=True)
```

<https://plotly.com/python/pie-charts/>



pandas



Streamlit



plotly

```
fig_margem_bruta = px.bar(  
    df_filtrado,  
    x="Mes",  
    y="gross income",  
    color="City",  
    title="Margem de Lucro Bruta por Mês",  
)  
fig_margem_bruta.update_yaxes(title="Renda bruta")  
col6.plotly_chart(fig_margem_bruta, use_container_width=True)
```

<https://plotly.com/python/bar-charts/>



```
st.write("Dados Analisados:")
st.write(df_filtrado)

st.write(f"Total do Mês: ${total_do_mes:,.2f}")
st.write(f"Total CoG: ${total_cogs:,.2f}")
st.write(f"Margem de Lucro Bruta Média: {margem_de_lucro_bruta:,.2f}%")

fig_avaliacao_faturamento = px.scatter(
    df_filtrado,
    x="Rating",
    y="Total",
    title="Relação entre Avaliação e Faturamento",
)
fig_avaliacao_faturamento.update_xaxes(title="Avaliação")
fig_avaliacao_faturamento.update_yaxes(title="Faturamento Total")
col7.plotly_chart(fig_avaliacao_faturamento, use_container_width=True)
```

<https://plotly.com/python/line-and-scatter/>



```
from model.model import ModeloSupermercado
from view.view import VisualizacaoSupermercado

class ControladorSupermercado:
    def __init__(self, caminho_dados: str):
        self.modelo = ModeloSupermercado(caminho_dados)
        self.visualizacao = VisualizacaoSupermercado()

    def executar(self, mes: str) -> None:
        total_cogs = self.modelo.calcular_total_cogs(mes)
        df_filtrado = self.modelo.obter_dados_por_mes(mes)
        total_do_mes = self.modelo.calcular_total_do_mes(mes)
        margem_de_lucro_bruta = self.modelo.calcular_margem_de_lucro_bruta(mes)
        self.visualizacao.exibir_pagina(
            df_filtrado, total_do_mes, margem_de_lucro_bruta, total_cogs
        )
```



pandas



Streamlit



plotly

<https://docs.streamlit.io/library/api-reference/layout/st.sidebar>

```
import streamlit as st
from controller.controller import ControladorSupermercado
import os

def main(caminho: str) -> None:
    controlador = ControladorSupermercado(caminho)
    controlador.modelo.criar_coluna_mes()
    st.sidebar.markdown(" :balloon: __Nossas opções__ :balloon: ")

    mes_radio = st.sidebar.radio(
        "Mês (Radio Button)", controlador.modelo.df["Mes"].unique()
    )
    st.write(f"Mês selecionado (Radio Button): {mes_radio}")

    mes_selectbox = st.sidebar.selectbox(
        "Mês (Selectbox)", controlador.modelo.df["Mes"].unique()
    )
    st.write(f"Mês selecionado (Selectbox): {mes_selectbox}")
```



pandas

 Streamlit



plotly

<https://docs.streamlit.io/library/api-reference/layout/st.sidebar>

```
mes_multiselect = st.sidebar.multiselect(
    "Mês (Multiselect)", controlador.modelo.df["Mes"].unique()
)
st.write(f"Mês selecionado (Multiselect): {mes_multiselect}")

mes_slider = st.sidebar.slider(
    "Mês (Slider)",
    min_value=int(controlador.modelo.df["Date"].dt.month.min()),
    max_value=int(controlador.modelo.df["Date"].dt.month.max()),
    step=1,
)
st.write(f"Ano selecionado (Slider): {mes_slider}")

controlador.executar(mes_radio)
```





```
if __name__ == "__main__":  
    st.set_page_config(page_title="Exemplo Final", page_icon="😊", layout="wide")  
    caminho = os.path.join(os.getcwd(), "assets", "supermarket_sales.csv")  
    if not os.path.exists(caminho):  
        raise FileNotFoundError(f"Arquivo não encontrado em {caminho}")  
    main(caminho)
```

<https://docs.streamlit.io/library/api-reference/text/st.markdown>

```
PS D:\Codigos\workspace\BigData\ProjetoMercado> streamlit run main.py  
  
You can now view your Streamlit app in your browser.  
  
Local URL: http://localhost:8501  
Network URL: http://172.16.0.50:8501
```

Python -m streamlit run main.py

<https://pandas.pydata.org/>

<https://streamlit.io/>

<https://plotly.com/python/plotly-express/>

<https://cheat-sheet.streamlit.app/>

<https://docs.streamlit.io/library/advanced-features/theming>



pandas



Streamlit



plotly

<https://github.com/professorRaphael/exemplo-streamlit>

<https://github.com/professorRaphael/ProjetoMercado>

[https://github.com/professorRaphael/streamlit\\_flask](https://github.com/professorRaphael/streamlit_flask)

[github.com/professorRaphael/Trabalho-pos/](https://github.com/professorRaphael/Trabalho-pos/)

# Perguntas



A group of diverse students are gathered in a modern library or study hall. Some are sitting on the floor in a circle, while others are standing or sitting at tables. They are engaged in various activities like reading, writing, and talking. Bookshelves filled with books line the walls, and large windows provide natural light. The overall atmosphere is one of active learning and collaboration.

# OBRIGADO!

**EDUCAR** PARA  
TRANS**FORMAR**



**Estácio**