

Realizarei projetos envolvendo programação e eu te usarei como meu assistente para isso.

O projeto envolve, em destaque, os seguintes termos:

```
---
---
WEB
HTML
CSS
Python (Microsoft)
Tailwind
Jupyter (Microsoft)
Python Debugger (Microsoft)
pandas
DOM
Artifact
csv
svg
Animação
uuid
XML
ASCII
TSX
Mermaid
PlantUML
Markdown
py
code
TEST
Debugging
Debug
Inspect
sys
platform
Warnings
Erros
Importlib
Import
Lib
libraries
TypeScript
JSX
TSX
github
Viz
numpy
"plotly>=5.0.0"
"google-cloud-storage<2.0.0"
google-auth
aleido
import json
graph
Data
import os
from datetime import datetime, timedelta
import
pandas
as
pd
Generated Data URI
import json
plotly.graph_objects
hooks
Vetor
upload.py
Vi
viz.py
as
go
dashboard.py
plotly.subplots
plotly
data.py
Reference Image URL
make_subplots
import os
kpi
Additional Libraries
Plot Types
webbrowser
df
df.to_csv
DataFrame
create_data
Data Row ID
dash
VS
create_directory
pd
kpi_data
```

```
pd.dataframe
data_gen.py URI
ls
gs
plot
outputs
inputs
assets
dir
CSS
dashboard
file
generate
print
VISUALIZATION
JavaScript
JS
DOM
HTML
---
---
```

Preciso saber tudo o que voce tem em seu ambiente de codigo e programação relacionado a esses termos.

Preciso entender tambem sobre eles, pois sou iniciante.

Tambem, preciso saber do que mais esta relacionado com estes termos e disponiveis ou nao disponiveis em seu ambiente de trabalho.

Me apresente tudo em lista organizada conforme lógica, com explicacoes, exemplos, e detalhes.

Neste primeiro momento, voce nao deve gerar nenhum codigo ainda, nem artifãt, ou similar. Voce deve focar em me apresentar o que voce tem disponivel em seu ambiente de trabalho em lista conforme pedido acima.

 Monica Sonnet

## Ambiente de Código e Programação Disponível

Olá! Vou explicar detalhadamente as ferramentas e tecnologias disponíveis no meu ambiente relacionadas aos termos que você mencionou. Como você é iniciante, incluirei explicações claras e organizadas por categorias lógicas.

### 1. Linguagens de Programação Disponíveis

#### Python

- **Disponível:** Sim, posso executar código Python através do Code Interpreter
- **Descrição:** Linguagem de programação versátil, de alto nível e interpretada
- **Ambiente:** Jupyter Notebook integrado com suporte a bibliotecas comuns
- **Uso:** Análise de dados, automação, desenvolvimento web, machine learning

#### JavaScript/TypeScript

- **Disponível:** Posso gerar código, mas não executar diretamente
- **Descrição:** JavaScript é a linguagem principal para web; TypeScript adiciona tipagem estática
- **Relacionados:** DOM (Document Object Model), JSX/TSX (React), Hooks

#### HTML/CSS

- **Disponível:** Posso gerar código, mas não renderizar diretamente
- **Descrição:** Linguagens de marcação e estilo para desenvolvimento web
- **Relacionados:** Tailwind CSS (framework de utilidades CSS)

### 2. Bibliotecas Python Disponíveis

#### Análise de Dados

- **pandas:** Biblioteca para manipulação e análise de dados estruturados
  - Funções: DataFrame, Series, read\_csv, to\_csv, merge, groupby
- **numpy:** Computação científica, arrays multidimensionais e funções matemáticas
- **datetime:** Manipulação de datas e horas

#### Visualização de Dados

- **plotly:** Biblioteca para gráficos interativos
  - Módulos: graph\_objects, subplots (make\_subplots)
- **matplotlib:** Biblioteca de visualização estática (disponível no Code Interpreter)
- **seaborn:** Visualizações estatísticas baseadas em matplotlib

#### Sistema e Utilitários

- **os:** Interações com o sistema operacional
- **sys:** Acesso a variáveis e funções específicas do sistema
- **platform:** Informações sobre a plataforma de execução
- **importlib:** Importação dinâmica de módulos
- **json:** Processamento de dados JSON
- **webbrowser:** Interface para navegadores web

#### Google Cloud

- **google-cloud-storage:** Interação com Google Cloud Storage
- **google-auth:** Autenticação para serviços Google

### 3. Ferramentas de Desenvolvimento

#### Ambientes de Execução

- **Code Interpreter:** Ambiente para executar código Python em tempo real
- **Jupyter:** Ambiente interativo para execução de código Python em células

### Debugging

- **Python Debugger:** Ferramentas para identificar e corrigir erros
- **Warnings:** Sistema de alertas para possíveis problemas no código
- **print:** Função básica para debugging

### Formatos de Dados

- **CSV:** Formato de arquivo para dados tabulares
- **JSON:** Formato de intercâmbio de dados leve
- **XML:** Linguagem de marcação para documentos estruturados
- **DataFrame:** Estrutura de dados bidimensional do pandas

## 4. Visualização e Diagramas

### Ferramentas de Diagramação

- **Mermaid:** Linguagem para criação de diagramas a partir de texto
- **PlantUML:** Linguagem para criação de diagramas UML a partir de texto

### Gráficos e Visualizações

- **SVG:** Formato de imagem vetorial para web
- **Plotly:** Biblioteca para gráficos interativos
- **Matplotlib:** Biblioteca para visualizações estáticas

## 5. Desenvolvimento Web

### Frontend

- **HTML/CSS:** Estrutura e estilo de páginas web
- **Tailwind CSS:** Framework CSS baseado em classes utilitárias
- **JavaScript:** Linguagem de programação para web
- **DOM:** Interface de programação para documentos HTML e XML
- **React:** Biblioteca JavaScript para construção de interfaces (via TSX/JSX)

### Ferramentas Web

- **Artifact:** Sintaxe especial para gerar conteúdo como SVG, HTML, etc.
- **UUID:** Geração de identificadores únicos

## 6. Limitações do Ambiente

- Não posso executar código JavaScript, HTML ou CSS diretamente
- Não tenho acesso direto à internet para baixar pacotes não instalados
- Não posso acessar ou modificar arquivos fora do ambiente do Code Interpreter
- Não posso manter estado entre sessões diferentes
- Não posso executar servidores web ou aplicações interativas em tempo real

## 7. Exemplos de Uso

### Exemplo de pandas para análise de dados

```
python

import pandas as pd
# Criar um DataFrame
data = {'Nome': ['Ana', 'Bruno', 'Carlos'],
        'Idade': [25, 30, 35],
        'Cidade': ['São Paulo', 'Rio', 'Belo Horizonte']}
df = pd.DataFrame(data)
# Salvar em CSV
df.to_csv('dados.csv', index=False)
```

Copiar

### Exemplo de plotly para visualização

```
python

import plotly.graph_objects as go
from plotly.subplots import make_subplots

# Criar um gráfico simples
fig = go.Figure(data=go.Scatter(x=[1, 2, 3], y=[4, 5, 6]))
fig.update_layout(title="Gráfico Simples")
fig.show()
```

Copiar

### Exemplo de manipulação de arquivos

```
python

import os
import json

# Criar diretório
os.makedirs('outputs', exist_ok=True)

# Salvar dados em JSON
dados = {'chave': 'valor', 'lista': [1, 2, 3]}
with open('outputs/dados.json', 'w') as f:
    json.dump(dados, f)
```

Copiar

## 8. Recursos Relacionados Disponíveis

- **Artifact Syntax:** Permite criar conteúdo especializado como SVG, HTML, TSX, etc.
- **Markdown:** Formatação de texto para documentação

- **Code Highlighting:** Destaque de sintaxe para diferentes linguagens
- **Data URI:** Codificação de dados em strings para uso em HTML/CSS

## 9. Ferramentas de Teste e Qualidade de Código

- **Debugging:** Identificação e correção de erros
- **Inspeção:** Análise de estruturas de dados e estado do programa
- **Warnings:** Alertas sobre possíveis problemas no código
- **Testes:** Verificação de funcionalidades do código

Estou à disposição para ajudá-lo com qualquer um desses tópicos. Quando estiver pronto, podemos começar a trabalhar em exemplos práticos ou aprofundar qualquer área específica que você tenha interesse.

continue

 Monica Sonnet

## Ambiente de Código e Programação Disponível (Continuação)

### 10. Ferramentas de Visualização Avançadas

#### Dashboards e Aplicações Interativas

- **Dash:** Framework para criar aplicações web analíticas em Python
  - **Disponibilidade:** Posso gerar código, mas execução requer ambiente externo
  - **Relacionados:** `dash.dependencies`, `dash_core_components`, `dash_html_components`
- **Plotly Express:** API de alto nível para Plotly
  - **Uso:** Criação rápida de visualizações complexas com menos código

#### Tipos de Gráficos Suportados

- **Gráficos de Linha:** Tendências ao longo do tempo
- **Gráficos de Barra/Coluna:** Comparações entre categorias
- **Gráficos de Dispersão:** Relações entre variáveis
- **Mapas de Calor:** Visualização de matrizes de dados
- **Gráficos 3D:** Visualizações tridimensionais
- **Gráficos de Área:** Magnitudes ao longo do tempo
- **Box Plots:** Distribuições estatísticas
- **Gráficos de Pizza:** Proporções de um todo

### 11. Manipulação de Dados Avançada

#### Operações com pandas

- **Filtros e Consultas:** `df[df['coluna'] > valor]`, `df.query()`
- **Agregações:** `groupby()`, `agg()`, `pivot_table()`
- **Junções:** `merge()`, `join()`, `concat()`
- **Transformações:** `apply()`, `map()`, `transform()`
- **Tratamento de Dados Ausentes:** `fillna()`, `dropna()`, `interpolate()`
- **Reshaping:** `melt()`, `pivot()`, `stack()`, `unstack()`

#### Geração de Dados

- **Dados Sintéticos:** Criação de conjuntos de dados para testes
- **Dados Aleatórios:** `numpy.random` para geração de valores
- **Séries Temporais:** Dados com índices de tempo
- **KPIs:** Métricas de desempenho para dashboards

### 12. Desenvolvimento Web Frontend

#### Componentes React (via TSX)

- **Disponibilidade:** Posso gerar código TSX através do Artifact
- **Bibliotecas Disponíveis:**
  - Base React
  - `lucide-react` (ícones)
  - `recharts` (gráficos)
  - `shadcn/ui` (componentes de UI)
- **Limitações:** Não posso usar `react-icons`, `antd` ou outras bibliotecas não listadas

#### CSS e Estilização

- **Tailwind CSS:** Framework de utilidades CSS
  - **Vantagens:** Desenvolvimento rápido, design responsivo
  - **Classes:** `flex`, `grid`, `text-{size}`, `bg-{color}`, etc.
- **CSS Puro:** Estilização tradicional
- **Animações:** Transições e animações CSS

### 13. Integração com Sistemas Externos

#### Armazenamento em Nuvem

- **Google Cloud Storage:** Armazenamento de objetos na nuvem Google
  - **Operações:** upload, download, listagem de arquivos
  - **Prefixos:** `gs://` para caminhos do Google Storage

- **Limitações:** Execução real requer credenciais e ambiente configurado

## Sistemas de Arquivos

- **Operações Locais:** Leitura/escrita de arquivos no ambiente do Code Interpreter
- **Estrutura de Diretórios:** Criação e navegação (assets, inputs, outputs)
- **Manipulação de Arquivos:** Abrir, ler, escrever, fechar

## 14. Ferramentas de Documentação e Diagramação

### Markdown Avançado

- **Tabelas:** Organização de dados em formato tabular
- **Links:** Referências a recursos externos ou internos
- **Imagens:** Incorporação de imagens
- **Blocos de Código:** Destaque de sintaxe para diferentes linguagens
- **Listas:** Ordenadas e não-ordenadas

### Diagramas Avançados

- **Mermaid:**
  - **Fluxogramas:** Processos e decisões
  - **Diagramas de Sequência:** Interações entre componentes
  - **Diagramas de Classe:** Estrutura de classes e relacionamentos
  - **Gantt:** Cronogramas de projeto
- **PlantUML:**
  - **Diagramas UML:** Casos de uso, classes, sequência, etc.
  - **Diagramas de Atividade:** Fluxos de trabalho
  - **Diagramas de Componentes:** Arquitetura de sistemas

## 15. Debugging e Testes

### Técnicas de Debugging

- **Print Debugging:** Uso de `print()` para visualizar valores
- **Logging:** Registro estruturado de eventos e erros
- **Inspecção de Variáveis:** Análise de valores durante execução
- **Tratamento de Exceções:** `try/except` para capturar e lidar com erros

### Testes

- **Testes Unitários:** Verificação de componentes individuais
- **Assertions:** Verificações de condições esperadas
- **Mocks:** Simulação de componentes externos

## 16. Formatos de Dados Especializados

### ASCII Art

- **Disponibilidade:** Posso gerar representações textuais de imagens ou diagramas
- **Uso:** Visualizações simples em ambientes somente texto

### SVG (Scalable Vector Graphics)

- **Disponibilidade:** Posso gerar através do Artifact
- **Características:** Gráficos vetoriais escaláveis
- **Uso:** Imagens de alta qualidade, gráficos interativos, ícones

### Data URI

- **Formato:** `scheme:data;base64,{dados}`
- **Uso:** Incorporar imagens ou outros dados diretamente em HTML/CSS

## 17. Integrações e Workflows

### GitHub

- **Disponibilidade:** Posso gerar código para interagir com GitHub, mas não executar
- **Operações:** Clone, push, pull, criação de issues (código apenas)
- **CI/CD:** Workflows de integração contínua (apenas conceitual)

### Automação

- **Scripts:** Automação de tarefas repetitivas
- **Pipelines de Dados:** Processamento sequencial de dados
- **Agendamento:** Conceitos de execução programada (cron)

## 18. Ambiente Microsoft

### Jupyter no Ambiente Microsoft

- **Disponibilidade:** Integrado ao Code Interpreter
- **Características:** Execução de código em células, visualização inline
- **Extensões:** Suporte a widgets e visualizações interativas

### Python Debugger (Microsoft)

- **Disponibilidade:** Conceitual apenas, não executável diretamente
- **Funcionalidades:** Breakpoints, step-through, inspeção de variáveis

19. Exemplos Práticos Adicionais

Geração de Dados e Visualização

python

```
import pandas as pd
import numpy as np
import plotly.express as px
from datetime import datetime, timedelta

# Gerar dados de série temporal
def create_data(n_days=30):
    dates = [datetime.now() - timedelta(days=i) for i in range(n_days)]
    values = np.random.normal(100, 15, n_days).cumsum() + 1000
    df = pd.DataFrame({'data': dates, 'valor': values})
    return df

# Criar KPIs
def create_kpi_data():
    kpi_data = {
        'métrica': ['Vendas', 'Conversão', 'Usuários', 'Tempo Médio'],
        'valor': [12500, 3.5, 45000, 2.7],
        'unidade': ['R$', '%', 'pessoas', 'minutos'],
        'variação': [12, -2.5, 7.8, 0.3]
    }
    return pd.DataFrame(kpi_data)

# Visualizar dados
df = create_data()
fig = px.line(df, x='data', y='valor', title="Tendência ao Longo do Tempo")
# fig.show() seria usado em ambiente real
```

Copiar

Estrutura de Diretórios para Projeto

python

```
import os

def create_directory_structure():
    # Criar estrutura básica de projeto
    directories = [
        'assets',
        'data',
        'outputs',
        'scripts',
        'notebooks',
        'docs'
    ]

    for directory in directories:
        os.makedirs(directory, exist_ok=True)
        print(f"Diretório '{directory}' criado com sucesso.")

# create_directory_structure()
```

Copiar

20. Recursos Não Disponíveis (Limitações)

- **Servidores Web Ativos:** Não posso hospedar aplicações web em tempo real
- **Bancos de Dados:** Não posso conectar diretamente a bancos de dados externos
- **Instalação de Pacotes:** Não posso instalar pacotes adicionais além dos disponíveis
- **Acesso a API Externas:** Limitado pela falta de acesso direto à internet
- **WebSockets:** Não posso manter conexões persistentes
- **Execução de Código Compilado:** Não posso executar binários ou código compilado
- **Acesso ao Sistema de Arquivos do Usuário:** Restrito ao ambiente do Code Interpreter

Estou à disposição para ajudá-lo a explorar qualquer um desses tópicos em maior profundidade ou para começarmos a trabalhar em projetos práticos utilizando as ferramentas disponíveis. Como iniciante, recomendo começarmos com exemplos simples e progredirmos gradualmente para conceitos mais avançados.

continue

Monica Sonnet

Ambiente de Código e Programação Disponível (Continuação)

21. Frameworks e Bibliotecas Web

React e Componentes

- **Disponibilidade:** Posso gerar código TSX via Artifact
- **Componentes Funcionais:** Uso de hooks e props
- **Hooks Comuns:**
  - `useState` : Gerenciamento de estado
  - `useEffect` : Efeitos colaterais
  - `useContext` : Contexto compartilhado
  - `useRef` : Referências persistentes
- **Exemplo de Estrutura:**

tsx

```
import React, { useState } from 'react';

const Component = () => {
  const [count, setCount] = useState(0);
  return (
    <div className="p-4">
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
};
```

Copiar

Tailwind CSS Detalhado

- **Classes Utilitárias:**

- Layout: flex, grid, container
- Espaçamento: p-4, m-2, gap-3
- Tipografia: text-lg, font-bold, italic
- Cores: bg-blue-500, text-red-700
- Responsividade: md:flex, lg:text-xl
- **Limitações:** Não suporta valores arbitrários como h-[600px]

## 22. Manipulação de Arquivos e Dados

### Operações com Arquivos

- **Leitura/Escrita:**

```
python

# Leitura de arquivo
with open('arquivo.txt', 'r') as f:
    conteudo = f.read()

# Escrita em arquivo
with open('saida.txt', 'w') as f:
    f.write('Conteúdo do arquivo')
```

[Copiar](#)

### Formatos de Dados Comuns

- **CSV:**

```
python

import pandas as pd

# Leitura
df = pd.read_csv('dados.csv')

# Escrita
df.to_csv('saida.csv', index=False)
```

[Copiar](#)

- **JSON:**

```
python

import json

# Leitura
with open('dados.json', 'r') as f:
    dados = json.load(f)

# Escrita
with open('saida.json', 'w') as f:
    json.dump(dados, f, indent=4)
```

[Copiar](#)

- **XML:**

```
python

import xml.etree.ElementTree as ET

# Criação de XML
root = ET.Element("root")
child = ET.SubElement(root, "child")
child.text = "Conteúdo"
tree = ET.ElementTree(root)
tree.write("saida.xml")
```

[Copiar](#)

## 23. Visualização de Dados Avançada

### Plotly Detalhado

- **Gráficos Interativos:**

```
python

import plotly.graph_objects as go

# Gráfico de barras interativo
fig = go.Figure(data=[
    go.Bar(name='Grupo A', x=['Jan', 'Fev', 'Mar'], y=[20, 14, 23]),
    go.Bar(name='Grupo B', x=['Jan', 'Fev', 'Mar'], y=[12, 18, 29])
])

fig.update_layout(
    title='Comparação por Mês',
    xaxis_title='Mês',
    yaxis_title='Valores',
    barmode='group'
)

# fig.show()
```

[Copiar](#)

### Subplots e Layouts Complexos

- **Múltiplos Gráficos:**

```
python

from plotly.subplots import make_subplots
import plotly.graph_objects as go

# Criar layout com 2 linhas e 2 colunas
fig = make_subplots(rows=2, cols=2,
                    subplot_titles=('Gráfico 1', 'Gráfico 2',
                                   'Gráfico 3', 'Gráfico 4'))

# Adicionar gráficos
fig.add_trace(go.Scatter(x=[1, 2, 3], y=[4, 5, 6]), row=1, col=1)
fig.add_trace(go.Bar(x=[1, 2, 3], y=[7, 8, 9]), row=1, col=2)
fig.add_trace(go.Pie(labels=['A', 'B', 'C'], values=[10, 20, 30]), row=2, col=1)
fig.add_trace(go.Scatter(x=[1, 2, 3], y=[10, 11, 12], mode='markers'), row=2, col=2)

fig.update_layout(height=600, width=800, title_text="Dashboard de Exemplo")
# fig.show()
```

[Copiar](#)

## 24. Análise de Dados com pandas

### Operações Avançadas

• **Agregações Personalizadas:**

```
python

import pandas as pd
import numpy as np

# Dados de exemplo
data = {
    'categoria': ['A', 'A', 'B', 'B', 'C'],
    'valor1': [10, 15, 20, 25, 30],
    'valor2': [100, 150, 200, 250, 300]
}
df = pd.DataFrame(data)

# Agregações por categoria
resultado = df.groupby('categoria').agg({
    'valor1': ['sum', 'mean', 'min', 'max'],
    'valor2': ['mean', lambda x: x.max() - x.min()]
})
```

[Copiar](#)

• **Pivotagem e Reshape:**

```
python

# Dados de vendas
vendas = {
    'data': ['2023-01-01', '2023-01-01', '2023-01-02', '2023-01-02'],
    'produto': ['A', 'B', 'A', 'B'],
    'valor': [100, 150, 120, 180]
}
df_vendas = pd.DataFrame(vendas)

# Pivot para formato largo
tabela_pivot = df_vendas.pivot(index='data', columns='produto', values='valor')

# Melt para formato longo
df_long = pd.melt(tabela_pivot.reset_index(), id_vars='data',
                  var_name='produto', value_name='valor')
```

[Copiar](#)

25. Diagramas e Visualizações Estruturais

Mermaid Avançado

• **Fluxogramas Complexos:**

```
css

graph TD
    A[Início] --> B{Decisão}
    B -->|Sim| C[Processo 1]
    B -->|Não| D[Processo 2]
    C --> E[Resultado 1]
    D --> F[Resultado 2]
    E --> G[Fim]
    F --> G
```

[Copiar](#)

• **Diagramas de Sequência:**

```
rust

sequenceDiagram
    participant Cliente
    participant Servidor
    participant Banco de Dados

    Cliente->>Servidor: Requisição
    Servidor->>Banco de Dados: Consulta
    Banco de Dados-->>Servidor: Resultados
    Servidor-->>Cliente: Resposta
```

[Copiar](#)

PlantUML Avançado

• **Diagramas de Classe:**

```
php

@startuml
class Usuario {
    -id: int
    -nome: string
    -email: string
    +autenticar(): boolean
    +atualizarPerfil(): void
}

class Produto {
    -id: int
    -nome: string
    -preco: float
    +calcularDesconto(): float
}

Usuario "1" -- "n" Produto: compra
@enduml
```

[Copiar](#)

26. Integração com APIs e Serviços

Conceitos e Estruturas

• **Requisições HTTP:**

```
python

import requests # Disponível no Code Interpreter

# GET request
response = requests.get('https://api.exemplo.com/dados')
dados = response.json()

# POST request
payload = {'chave': 'valor'}
response = requests.post('https://api.exemplo.com/enviar', json=payload)
```

[Copiar](#)

• **Autenticação:**

```
python

# Autenticação básica
response = requests.get('https://api.exemplo.com/dados',
                        auth=('usuario', 'senha'))
```

[Copiar](#)



```
# Autenticação com token
headers = {'Authorization': 'Bearer seu_token_aqui'}
response = requests.get('https://api.exemplo.com/dados', headers=headers)
```

## 27. Desenvolvimento de Dashboards

### Estrutura de Aplicação Dash

• Componentes Básicos:

```
python

import dash
import dash_core_components as dcc
import dash_html_components as html
from dash.dependencies import Input, Output
import plotly.express as px
import pandas as pd

# Dados de exemplo
df = pd.DataFrame({
    'x': range(10),
    'y': range(10)
})

# Inicializar app
app = dash.Dash(__name__)

# Layout
app.layout = html.Div([
    html.H1("Dashboard de Exemplo"),
    dcc.Dropdown(
        id='dropdown',
        options=[{'label': i, 'value': i} for i in ['Linear', 'Quadrático']],
        value='Linear'
    ),
    dcc.Graph(id='grafico')
])

# Callbacks
@app.callback(
    Output('grafico', 'figure'),
    Input('dropdown', 'value')
)
def update_graph(value):
    if value == 'Linear':
        fig = px.line(df, x='x', y='y')
    else:
        df['y_quad'] = df['x'] ** 2
        fig = px.line(df, x='x', y='y_quad')
    return fig

# Em ambiente real: app.run_server(debug=True)
```

 Copiar

## 28. Ferramentas de Geração de Conteúdo

### Artifact Syntax Detalhado

• Tipos Suportados:

- svg : Gráficos vetoriais
- code.[lang] : Blocos de código com destaque de sintaxe
- mermaid : Diagramas Mermaid
- plantuml : Diagramas UML
- html : Conteúdo HTML
- tsx : Componentes React
- markdown : Conteúdo formatado

• Exemplo de SVG:

```
arduino

id: circulo_exemplo
name: Círculo Vermelho
type: svg
content: |-
<svg width="100" height="100">
  <circle cx="50" cy="50" r="40" stroke="black" stroke-width="3" fill="red" />
</svg>
```

 Copiar

• Exemplo de TSX:

```
javascript

id: contador_exemplo
name: Contador React
type: tsx
content: |-
import React, { useState } from 'react';
import { Button } from "@/components/ui/button";

const Counter = () => {
  const [count, setCount] = useState(0);

  return (
    <div className="p-4 border rounded">
      <h2 className="text-xl font-bold mb-2">Contador: {count}</h2>
      <div className="flex gap-2">
        <Button onClick={() => setCount(count - 1)}>Diminuir</Button>
        <Button onClick={() => setCount(count + 1)}>Aumentar</Button>
      </div>
    </div>
  );
};

export default Counter;
```

 Copiar

## 29. Depuração e Tratamento de Erros

### Técnicas Avançadas

• Tratamento de Exceções Estruturado:

```
python
```

 Copiar

```
try:
    # Código que pode gerar erro
    resultado = 10 / 0
except ZeroDivisionError as e:
    print(f"Erro de divisão por zero: {e}")
except Exception as e:
    print(f"Erro genérico: {e}")
else:
    print("Execução bem-sucedida")
finally:
    print("Este bloco sempre é executado")
```

- **Logging Estruturado:**

```
python

import logging

# Configurar logger
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s'
)

logger = logging.getLogger('minha_aplicacao')

# Usar logger
logger.info("Aplicação iniciada")
logger.warning("Aviso: recurso quase esgotado")
logger.error("Erro ao processar dados")
```

[Copiar](#)

## 30. Recursos Adicionais e Integrações

### UUID (Identificadores Únicos)

- **Geração de IDs:**

```
python

import uuid

# Gerar UUID aleatório
id_aleatorio = uuid.uuid4()
print(id_aleatorio) # Ex: 123e4567-e89b-12d3-a456-426614174000

# Gerar UUID baseado em nome
id_namespace = uuid.uuid5(uuid.NAMESPACE_DNS, "exemplo.com")
print(id_namespace)
```

[Copiar](#)

### Manipulação de Datas e Horas

- **Operações com datetime:**

```
python

from datetime import datetime, timedelta

# Data e hora atual
agora = datetime.now()

# Formatação
data_formatada = agora.strftime("%d/%m/%Y %H:%M:%S")

# Operações
amanha = agora + timedelta(days=1)
proxima_semana = agora + timedelta(weeks=1)

# Diferença entre datas
diferenca = amanhã - agora
print(f"Segundos até amanhã: {diferenca.total_seconds()}")
```

[Copiar](#)

Todos estes recursos estão disponíveis para ajudá-lo em seus projetos de programação. Como iniciante, recomendo começar com conceitos básicos de Python e pandas para análise de dados, ou HTML/CSS/JavaScript para desenvolvimento web, dependendo do seu interesse principal.

Posso ajudá-lo a criar exemplos práticos, explicar conceitos em detalhes ou desenvolver projetos completos utilizando estas ferramentas. Quando estiver pronto para começar a programar, podemos utilizar o Code Interpreter para executar código Python ou criar artifacts para visualizações e componentes web.

Há alguma área específica que você gostaria de explorar primeiro ou algum projeto que gostaria de iniciar?

[Tornar mais longo](#)

 **Monica** Sonnet

## Ambiente de Código e Programação Disponível: Guia Completo

### 1. Linguagens de Programação Disponíveis

#### Python

- **Disponibilidade:** Totalmente disponível através do Code Interpreter integrado
- **Versão:** Python 3.x com suporte a bibliotecas modernas
- **Características:**
  - Linguagem interpretada de alto nível
  - Tipagem dinâmica e forte
  - Multiparadigma: suporta programação orientada a objetos, funcional e procedural
  - Sintaxe limpa e legível com uso de indentação para blocos de código
- **Casos de Uso:**
  - Análise e visualização de dados
  - Automação de tarefas
  - Desenvolvimento web (backend)
  - Machine learning e inteligência artificial
  - Processamento de texto e manipulação de arquivos
- **Limitações no Ambiente:**
  - Sem acesso direto à internet para instalação de pacotes adicionais
  - Sem persistência de estado entre sessões diferentes

- Tempo de execução limitado a 120 segundos

## JavaScript/TypeScript

- **Disponibilidade:** Posso gerar código, mas não executar diretamente
- **Características:**
  - JavaScript: Linguagem de programação interpretada, de tipagem dinâmica
  - TypeScript: Superset de JavaScript que adiciona tipagem estática opcional
- **Componentes Relacionados:**
  - DOM (Document Object Model): Interface para interagir com elementos HTML
  - JSX/TSX: Extensões de sintaxe para incorporar HTML em JavaScript/TypeScript (React)
  - Hooks: Funções especiais do React para gerenciamento de estado e efeitos
- **Casos de Uso:**
  - Desenvolvimento frontend
  - Interatividade em páginas web
  - Aplicações de página única (SPAs)
  - Manipulação do DOM
- **Limitações no Ambiente:**
  - Não é possível executar código JavaScript diretamente
  - Geração apenas via Artifact para componentes TSX

## HTML/CSS

- **Disponibilidade:** Posso gerar código, mas não renderizar diretamente
- **Características:**
  - HTML: Linguagem de marcação para estruturar conteúdo web
  - CSS: Linguagem de estilo para definir a apresentação visual
- **Frameworks Relacionados:**
  - Tailwind CSS: Framework CSS baseado em classes utilitárias
- **Casos de Uso:**
  - Estruturação de páginas web
  - Estilização e design responsivo
  - Layouts e interfaces de usuário
- **Limitações no Ambiente:**
  - Não é possível visualizar a renderização diretamente
  - Geração apenas via Artifact

## 2. Bibliotecas Python Disponíveis

### Análise de Dados

- **pandas:**
  - **Descrição:** Biblioteca para manipulação e análise de dados estruturados
  - **Componentes Principais:**
    - DataFrame: Estrutura de dados bidimensional com colunas potencialmente de diferentes tipos
    - Series: Array unidimensional rotulado
    - Index: Objeto para indexação de dados
  - **Funcionalidades:**
    - Leitura/escrita de dados: read\_csv(), to\_csv(), read\_excel(), to\_excel(), etc.
    - Seleção e filtragem: loc[], iloc[], query(), boolean indexing
    - Agregação: groupby(), agg(), pivot\_table(), crosstab()
    - Junção de dados: merge(), join(), concat()
    - Tratamento de dados ausentes: fillna(), dropna(), interpolate()
    - Reshaping: melt(), pivot(), stack(), unstack()
    - Janelas deslizantes: rolling(), expanding(), ewm()

- **Exemplo Básico:**

```
python

import pandas as pd

# Criar DataFrame
df = pd.DataFrame({
    'Nome': ['Ana', 'Bruno', 'Carlos', 'Diana'],
    'Idade': [25, 30, 35, 28],
    'Departamento': ['TI', 'Marketing', 'TI', 'Financeiro'],
    'Salário': [5000, 6000, 7000, 5500]
})

# Operações básicas
media_por_depto = df.groupby('Departamento')['Salário'].mean()
filtro_idade = df[df['Idade'] > 30]
```

 Copiar

- **numpy:**
  - **Descrição:** Biblioteca fundamental para computação científica em Python
  - **Componentes Principais:**
    - ndarray: Array multidimensional eficiente
    - Funções universais (ufuncs): Operações vetorizadas
  - **Funcionalidades:**
    - Operações matemáticas: sin(), cos(), exp(), log(), etc.
    - Álgebra linear: dot(), matmul(), eig(), svd(), etc.
    - Estatística: mean(), std(), var(), min(), max(), etc.

- Manipulação de arrays: reshape(), transpose(), concatenate(), split(), etc.
- Geração de números aleatórios: random.rand(), random.normal(), etc.

• **Exemplo Básico:**

```
python

import numpy as np

# Criar arrays
arr1 = np.array([1, 2, 3, 4, 5])
arr2 = np.random.normal(0, 1, (3, 3)) # Matriz 3x3 com distribuição normal

# Operações
soma = arr1.sum()
media = arr1.mean()
matriz_transposta = arr2.T
```

 Copiar

• **datetime:**

- **Descrição:** Módulo para manipulação de datas e horas
- **Componentes Principais:**
  - datetime: Classe para representar data e hora
  - timedelta: Classe para representar diferenças entre datas/horas
  - date: Classe para representar apenas datas
  - time: Classe para representar apenas horas
- **Funcionalidades:**
  - Criação de datas e horas: datetime.now(), datetime.strptime()
  - Formatação: strftime()
  - Operações: adição/subtração de timedeltas
  - Comparações entre datas

• **Exemplo Básico:**

```
python

from datetime import datetime, timedelta

# Data atual
agora = datetime.now()

# Formatação
data_formatada = agora.strftime("%d/%m/%Y %H:%M:%S")

# Operações
amanha = agora + timedelta(days=1)
uma_semana_atras = agora - timedelta(weeks=1)
```

 Copiar

## Visualização de Dados

• **plotly:**

- **Descrição:** Biblioteca para criação de gráficos interativos
- **Módulos Principais:**
  - graph\_objects: API de baixo nível para controle detalhado
  - express: API de alto nível para criação rápida
  - subplots: Criação de layouts com múltiplos gráficos
- **Tipos de Gráficos:**
  - Gráficos básicos: linha, barra, dispersão, área, pizza
  - Gráficos estatísticos: boxplot, histograma, violino, heatmap
  - Gráficos 3D: superfície, dispersão 3D, linha 3D
  - Mapas: geográficos, choropleth, scatter\_geo
  - Gráficos financeiros: candlestick, OHLC
- **Funcionalidades:**
  - Interatividade: zoom, pan, hover, seleção
  - Personalização: layouts, cores, fontes, legendas
  - Animações: frames e transições
  - Subplots: múltiplos gráficos em uma figura

• **Exemplo Básico:**

```
python

import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots

# Gráfico simples com Express
df = px.data.gapminder().query("continent=='Europe' and year==2007")
fig1 = px.bar(df, x='country', y='pop', title='População na Europa (2007)')

# Gráfico personalizado com Graph Objects
fig2 = go.Figure()
fig2.add_trace(go.Scatter(x=[1, 2, 3], y=[4, 5, 6], mode='lines+markers'))
fig2.update_layout(title='Gráfico de Linha Personalizado')

# Múltiplos gráficos
fig3 = make_subplots(rows=1, cols=2)
fig3.add_trace(go.Bar(x=['A', 'B', 'C'], y=[1, 2, 3]), row=1, col=1)
fig3.add_trace(go.Scatter(x=[1, 2, 3], y=[4, 5, 6]), row=1, col=2)
```

 Copiar

• **matplotlib:**

- **Descrição:** Biblioteca de visualização estática tradicional
- **Componentes Principais:**
  - pyplot: Interface para criação rápida de gráficos
  - figure: Objeto para controle detalhado
  - axes: Áreas individuais de plotagem
- **Tipos de Gráficos:**

- Gráficos básicos: linha, barra, dispersão, histograma
- Gráficos estatísticos: boxplot, violino, densidade
- Gráficos especializados: contorno, stream, quiver

• **Funcionalidades:**

- Personalização detalhada
- Múltiplos subplots
- Anotações e textos
- Salvamento em diversos formatos

• **Exemplo Básico:**

```
python

import matplotlib.pyplot as plt
import numpy as np

# Dados
x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.cos(x)

# Criar figura e subplots
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 8))

# Primeiro subplot
ax1.plot(x, y1, 'b-', label='sen(x)')
ax1.set_title('Função Seno')
ax1.legend()

# Segundo subplot
ax2.plot(x, y2, 'r-', label='cos(x)')
ax2.set_title('Função Cosseno')
ax2.legend()

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

 Copiar

• **seaborn:**

- **Descrição:** Biblioteca para visualizações estatísticas baseada em matplotlib

• **Tipos de Gráficos:**

- Gráficos de distribuição: histplot, kdeplot, ecdfplot
- Gráficos relacionais: scatterplot, lineplot, relplot
- Gráficos categóricos: barplot, boxplot, violinplot, stripplot
- Matrizes: heatmap, clustermap

• **Funcionalidades:**

- Temas visuais predefinidos
- Paletas de cores
- Visualização de dados multivariados
- Integração com pandas

• **Exemplo Básico:**

```
python

import seaborn as sns
import matplotlib.pyplot as plt

# Carregar dataset de exemplo
tips = sns.load_dataset('tips')

# Configurar estilo
sns.set_theme(style="whitegrid")

# Criar visualização
plt.figure(figsize=(10, 6))
sns.boxplot(x='day', y='total_bill', hue='sex', data=tips)
plt.title('Distribuição de Contas por Dia e Gênero')
plt.show()
```

 Copiar

## Sistema e Utilitários

• **os:**

- **Descrição:** Módulo para interações com o sistema operacional
- **Funcionalidades:**
  - Manipulação de caminhos: os.path.join(), os.path.exists(), os.path.dirname()
  - Manipulação de diretórios: os.mkdir(), os.makedirs(), os.listdir(), os.walk()
  - Manipulação de arquivos: os.remove(), os.rename()
  - Variáveis de ambiente: os.environ
  - Execução de comandos: os.system()

• **Exemplo Básico:**

```
python

import os

# Criar diretório
os.makedirs('dados/processados', exist_ok=True)

# Listar arquivos
arquivos = os.listdir('.')

# Verificar se arquivo existe
if os.path.exists('arquivo.txt'):
    print('O arquivo existe')
```

 Copiar

• **sys:**

- **Descrição:** Módulo para acesso a variáveis e funções específicas do sistema
- **Funcionalidades:**
  - Argumentos de linha de comando: sys.argv
  - Entrada/saída padrão: sys.stdin, sys.stdout, sys.stderr

- Informações sobre o interpretador: `sys.version`, `sys.platform`
- Controle de execução: `sys.exit()`

- **Exemplo Básico:**

```
python

import sys

# Verificar versão do Python
print(f"Versão do Python: {sys.version}")

# Plataforma
print(f"Plataforma: {sys.platform}")

# Saída formatada
sys.stdout.write("Mensagem formatada\n")
```

 Copiar

- **platform:**

- **Descrição:** Módulo para obter informações sobre a plataforma de execução
- **Funcionalidades:**
  - Informações sobre o sistema: `platform.system()`, `platform.release()`
  - Informações sobre a máquina: `platform.machine()`, `platform.processor()`
  - Informações sobre o Python: `platform.python_version()`

- **Exemplo Básico:**

```
python

import platform

# Sistema operacional
sistema = platform.system()
versao = platform.release()
print(f"Sistema: {sistema} {versao}")

# Arquitetura
arquitetura = platform.machine()
print(f"Arquitetura: {arquitetura}")
```

 Copiar

- **importlib:**

- **Descrição:** Módulo para importação dinâmica de módulos
- **Funcionalidades:**
  - Importação dinâmica: `import_module()`
  - Recarregamento de módulos: `reload()`
  - Metadados de módulos: `metadata`

- **Exemplo Básico:**

```
python

import importlib

# Importar módulo dinamicamente
math = importlib.import_module('math')
resultado = math.sqrt(16)

# Recarregar módulo
importlib.reload(math)
```

 Copiar

- **json:**

- **Descrição:** Módulo para processamento de dados JSON
- **Funcionalidades:**
  - Decodificação (parsing): `loads()`, `load()`
  - Codificação (serialização): `dumps()`, `dump()`
  - Personalização: `JSONEncoder`, `JSONDecoder`

- **Exemplo Básico:**

```
python

import json

# Dicionário Python
dados = {
    'nome': 'João',
    'idade': 30,
    'cidade': 'São Paulo',
    'interesses': ['programação', 'música', 'esportes']
}

# Serializar para string JSON
json_str = json.dumps(dados, indent=4, ensure_ascii=False)
print(json_str)

# Deserializar de string JSON
dados_recuperados = json.loads(json_str)
```

 Copiar

- **webbrowser:**

- **Descrição:** Módulo para interface com navegadores web
- **Funcionalidades:**
  - Abrir URLs: `open()`, `open_new()`, `open_new_tab()`
  - Registrar navegadores: `register()`

- **Exemplo Básico:**

```
python

import webbrowser

# Abrir URL em navegador padrão
url = 'https://www.exemplo.com'
webbrowser.open(url) # Não funciona no ambiente do Code Interpreter
```

 Copiar

## Google Cloud

- **google-cloud-storage:**

- **Descrição:** Cliente para Google Cloud Storage
- **Funcionalidades:**
  - Criação de buckets: create\_bucket()
  - Upload de objetos: upload\_from\_filename(), upload\_from\_string()
  - Download de objetos: download\_to\_filename(), download\_as\_string()
  - Listagem de objetos: list\_blobs()
  - Gerenciamento de buckets e objetos: delete\_blob(), copy\_blob()

• **Exemplo Conceitual:**

```
python

from google.cloud import storage

# Cliente
# client = storage.Client()

# Operações com buckets
# bucket = client.get_bucket('nome-do-bucket')

# Upload de arquivo
# blob = bucket.blob('caminho/arquivo.txt')
# blob.upload_from_filename('arquivo_local.txt')

# Download de arquivo
# blob.download_to_filename('arquivo_baixado.txt')
```

 Copiar

- **google-auth:**
  - **Descrição:** Biblioteca para autenticação em serviços Google
  - **Funcionalidades:**
    - Autenticação com credenciais: service\_account.Credentials
    - Autenticação com token: oauth2.credentials.Credentials
    - Fluxos de autenticação: default(), get\_application\_default()

• **Exemplo Conceitual:**

```
python

from google.oauth2 import service_account

# Carregar credenciais
# credentials = service_account.Credentials.from_service_account_file(
#     'caminho/para/credenciais.json'
# )

# Definir escopo
# scoped_credentials = credentials.with_scopes(
#     ['https://www.googleapis.com/auth/cloud-platform']
# )
```

 Copiar

3. Ferramentas de Desenvolvimento

Ambientes de Execução

- **Code Interpreter:**
  - **Descrição:** Ambiente para execução de código Python em tempo real
  - **Características:**
    - Execução interativa de código Python
    - Suporte a bibliotecas científicas e de análise de dados
    - Visualização de resultados inline
    - Persistência temporária de arquivos
    - Limite de tempo de execução de 120 segundos
  - **Uso Típico:**
    - Análise exploratória de dados
    - Criação de visualizações
    - Prototipagem rápida
    - Demonstração de conceitos
- **Jupyter:**
  - **Descrição:** Ambiente interativo para execução de código em células
  - **Características:**
    - Documentos que combinam código, texto e visualizações
    - Execução não-linear (células podem ser executadas em qualquer ordem)
    - Suporte a markdown para documentação
    - Visualização rica de resultados
  - **Integração no Ambiente:**
    - O Code Interpreter é baseado em tecnologia similar ao Jupyter
    - Suporta a mesma sintaxe e comportamento de células

• **Exemplo de Uso:**

```
python

# Esta é uma célula Jupyter
import pandas as pd
import matplotlib.pyplot as plt

# Criar dados
dados = {'x': range(10), 'y': [i**2 for i in range(10)]}
df = pd.DataFrame(dados)

# Visualizar
plt.figure(figsize=(8, 6))
plt.plot(df['x'], df['y'], 'ro-')
plt.title('Função Quadrática')
plt.xlabel('x')
plt.ylabel('y = x²')
```

 Copiar

```
plt.grid(True)
plt.show()
```

### Debugging

- **Python Debugger:**

- **Descrição:** Ferramentas para identificar e corrigir erros
- **Componentes:**
  - pdb: Módulo de debugging integrado do Python
  - Breakpoints: Pontos de parada para inspeção
  - Step-through: Execução passo a passo
  - Inspeção de variáveis: Visualização de valores em tempo de execução
- **Técnicas Disponíveis:**
  - Print debugging: Uso de print() para visualizar valores
  - Logging: Registro estruturado de eventos
  - Tratamento de exceções: try/except para capturar erros

- **Exemplo Conceitual:**

```
python

# Debugging com print
def funcao_problematica(x):
    print(f"Valor de entrada: {x}")
    resultado = x * 2
    print(f"Resultado intermediário: {resultado}")
    return resultado + 10

# Debugging com tratamento de exceções
def divisao_segura(a, b):
    try:
        resultado = a / b
        return resultado
    except ZeroDivisionError:
        print("Erro: Divisão por zero!")
        return None
    except TypeError as e:
        print(f"Erro de tipo: {e}")
        return None
```

 Copiar

- **Warnings:**

- **Descrição:** Sistema de alertas para possíveis problemas no código
- **Funcionalidades:**
  - Emissão de avisos: warn()
  - Filtragem de avisos: filterwarnings(), simplefilter()
  - Captura de avisos: catch\_warnings()

- **Exemplo Básico:**

```
python

import warnings

# Emitir aviso
warnings.warn("Esta função está obsoleta", DeprecationWarning)

# Filtrar avisos
warnings.filterwarnings("ignore", category=DeprecationWarning)

# Capturar avisos
with warnings.catch_warnings():
    warnings.simplefilter("error")
    try:
        warnings.warn("Este aviso será tratado como erro")
    except Warning as w:
        print(f"Capturado: {w}")
```

 Copiar

### Formatos de Dados

- **CSV:**

- **Descrição:** Formato de arquivo para dados tabulares
- **Características:**
  - Valores separados por vírgula (ou outro delimitador)
  - Formato texto simples
  - Suporte universal
  - Fácil de ler e escrever
- **Manipulação com pandas:**
  - read\_csv(): Leitura de arquivos CSV
  - to\_csv(): Escrita em arquivos CSV

- **Exemplo Básico:**

```
python

import pandas as pd

# Criar DataFrame
dados = {
    'Nome': ['Ana', 'Bruno', 'Carlos'],
    'Idade': [25, 30, 35],
    'Salário': [5000, 6000, 7000]
}
df = pd.DataFrame(dados)

# Salvar em CSV
df.to_csv('funcionarios.csv', index=False)

# Ler de CSV
df_lido = pd.read_csv('funcionarios.csv')
```

 Copiar

- **JSON:**

- **Descrição:** Formato de intercâmbio de dados leve



• **Características:**

- Baseado em sintaxe JavaScript
- Suporta objetos, arrays, strings, números, booleanos e null
- Formato texto
- Amplamente utilizado em APIs web

• **Manipulação com Python:**

- json.loads(): Parsing de string JSON
- json.dumps(): Serialização para string JSON
- json.load(): Parsing de arquivo JSON
- json.dump(): Serialização para arquivo JSON

• **Exemplo Básico:**

```
python

import json

# Objeto Python
pessoa = {
    'nome': 'Maria',
    'idade': 28,
    'profissao': 'Engenheira',
    'habilidades': ['Python', 'JavaScript', 'SQL'],
    'ativo': True
}

# Serializar para JSON
json_str = json.dumps(pessoa, indent=2)

# Salvar em arquivo
with open('pessoa.json', 'w') as f:
    json.dump(pessoa, f, indent=2)

# Ler de string
obj = json.loads(json_str)

# Ler de arquivo
with open('pessoa.json', 'r') as f:
    pessoa_lida = json.load(f)
```

 Copiar

• **XML:**

- **Descrição:** Linguagem de marcação para documentos estruturados

• **Características:**

- Baseado em tags
- Estrutura hierárquica
- Suporte a atributos
- Formato texto

• **Manipulação com Python:**

- xml.etree.ElementTree: API para parsing e criação
- xml.dom: API DOM para XML
- xml.sax: API SAX para parsing baseado em eventos

• **Exemplo Básico:**

```
python

import xml.etree.ElementTree as ET

# Criar documento XML
root = ET.Element("pessoas")

# Adicionar elemento
pessoa = ET.SubElement(root, "pessoa", attrib={"id": "1"})
ET.SubElement(pessoa, "nome").text = "João"
ET.SubElement(pessoa, "idade").text = "25"

# Converter para string
xml_str = ET.tostring(root, encoding='unicode')

# Criar árvore
tree = ET.ElementTree(root)
# tree.write("pessoas.xml")

# Parsing de string
root_parsed = ET.fromstring(xml_str)
```

 Copiar

• **DataFrame:**

- **Descrição:** Estrutura de dados bidimensional do pandas

• **Características:**

- Colunas potencialmente de diferentes tipos
- Índices rotulados para linhas e colunas
- Operações vetorizadas
- Manipulação eficiente de dados

• **Operações Comuns:**

- Criação: pd.DataFrame()
- Seleção: df['coluna'], df.loc[], df.iloc[]
- Filtragem: df[df['coluna'] > valor]
- Agregação: df.groupby().agg()
- Transformação: df.apply(), df.transform()

• **Exemplo Avançado:**

```
python

import pandas as pd
import numpy as np

# Criar DataFrame complexo
np.random.seed(42)
datas = pd.date_range('20230101', periods=100)
```

 Copiar

```
df = pd.DataFrame({
    'data': datas,
    'categoria': np.random.choice(['A', 'B', 'C'], 100),
    'valor1': np.random.normal(100, 15, 100),
    'valor2': np.random.uniform(0, 1, 100)
})

# Operações avançadas
# Agregação por categoria e mês
resultado = df.assign(mes=df['data'].dt.month) \
    .groupby(['categoria', 'mes']) \
    .agg({
        'valor1': ['mean', 'std', 'min', 'max'],
        'valor2': ['mean', 'sum']
    })

# Pivotagem
pivot = df.pivot_table(
    index='data',
    columns='categoria',
    values=['valor1', 'valor2'],
    aggfunc={'valor1': 'mean', 'valor2': 'sum'}
)
```

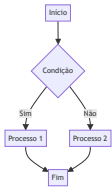
4. Visualização e Diagramas

Ferramentas de Diagramação

- **Mermaid:**
  - **Descrição:** Linguagem para criação de diagramas a partir de texto
  - **Tipos de Diagramas:**
    - Fluxogramas (flowchart): Processos e decisões
    - Diagramas de Sequência (sequenceDiagram): Interações entre componentes
    - Diagramas de Classe (classDiagram): Estrutura de classes
    - Diagramas de Estado (stateDiagram): Máquinas de estado
    - Diagramas de Gantt (gantt): Cronogramas de projeto
    - Diagramas de Entidade-Relacionamento (erDiagram): Modelagem de dados
    - Diagramas de Jornada do Usuário (journey): Experiência do usuário
- **Sintaxe Básica:**

mermaid

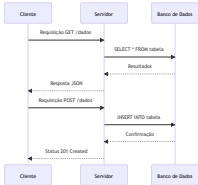
Visualizar Copiar



- **Exemplo de Diagrama de Sequência:**

mermaid

Visualizar Copiar



- **PlantUML:**
  - **Descrição:** Linguagem para criação de diagramas UML a partir de texto
  - **Tipos de Diagramas:**
    - Diagramas de Classe: Estrutura de classes e relacionamentos
    - Diagramas de Sequência: Interações entre objetos
    - Diagramas de Caso de Uso: Requisitos e atores
    - Diagramas de Atividade: Fluxos de trabalho
    - Diagramas de Componentes: Arquitetura de sistemas
    - Diagramas de Estado: Máquinas de estado
    - Diagramas de Implantação: Infraestrutura física

continue

Monica Sonnet

Ambiente de Código e Programação Disponível (Continuação)

4. Visualização e Diagramas (Continuação)

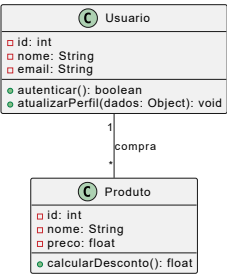
Ferramentas de Diagramação (Continuação)

- **PlantUML (Continuação):**

- **Sintaxe Básica:**

plantuml

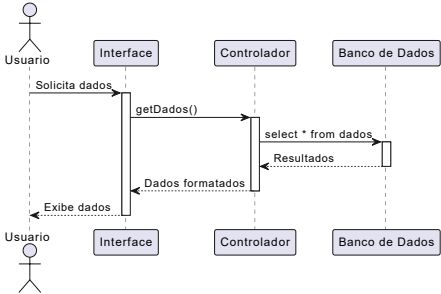
Visualizar Copiar



Exemplo de Diagrama de Sequência:

plantuml

Visualizar Copiar



Vantagens:

- Suporte completo a UML 2.x
- Personalização avançada
- Integração com ferramentas de desenvolvimento
- Exportação para diversos formatos

Gráficos e Visualizações

SVG:

- Descrição: Formato de imagem vetorial para web

Características:

- Baseado em XML
- Escalável sem perda de qualidade
- Manipulável via CSS e JavaScript
- Suporte a animações

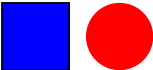
Elementos Básicos:

- <svg> : Elemento raiz
- <rect> : Retângulos
- <circle> , <ellipse> : Círculos e elipses
- <line> , <polyline> : Linhas
- <polygon> , <path> : Formas complexas
- <text> : Texto
- <g> : Agrupamento

Exemplo Básico:

svg

Visualizar Copiar



SVG Exemplo

Geração via Artifact:

arduino

Copiar

```
id: svg_exemplo
name: Exemplo SVG
type: svg
content: |-
<svg width="200" height="200" xmlns="http://www.w3.org/2000/svg">
  <rect x="10" y="10" width="80" height="80" fill="blue" stroke="black" stroke-width="2" />
  <circle cx="150" cy="50" r="40" fill="red" />
  <text x="100" y="150" font-size="20" text-anchor="middle">SVG Exemplo</text>
</svg>
```

Plotly:

- Descrição: Biblioteca para gráficos interativos
- Características Avançadas:
  - Interatividade completa: zoom, pan, hover, seleção
  - Animações e transições
  - Layouts responsivos
  - Temas e estilos personalizados
  - Exportação para diversos formatos

- **Gráficos Especializados:**
  - Mapas de calor (heatmap): Visualização de matrizes
  - Gráficos de contorno (contour): Superfícies 2D
  - Gráficos 3D: Superfícies, dispersão, malhas
  - Gráficos financeiros: Candlestick, OHLC
  - Gráficos estatísticos: Box plot, violin plot, distplot

• **Exemplo Avançado:**

```
python

import plotly.graph_objects as go
from plotly.subplots import make_subplots
import numpy as np

# Criar dados
x = np.linspace(-5, 5, 100)
y1 = np.sin(x)
y2 = np.cos(x)
y3 = np.exp(-x**2/10)

# Criar layout com 3 subplots
fig = make_subplots(
    rows=2, cols=2,
    subplot_titles=("Função Seno", "Função Cosseno", "Função Gaussiana", ""),
    specs=[[{}], {}], [{"colspan": 2}, None]],
    vertical_spacing=0.1
)

# Adicionar gráficos
fig.add_trace(go.Scatter(x=x, y=y1, mode='lines', name='sin(x)', row=1, col=1))
fig.add_trace(go.Scatter(x=x, y=y2, mode='lines', name='cos(x)', row=1, col=2))
fig.add_trace(go.Scatter(x=x, y=y3, mode='lines', name='exp(-x^2/10)',
                        fill='tozeroy'), row=2, col=1)

# Atualizar layout
fig.update_layout(
    title_text="Funções Matemáticas",
    height=600,
    showLegend=True,
    legend=dict(orientation="h", yanchor="bottom", y=1.02, xanchor="right", x=1)
)

# Atualizar eixos
fig.update_xaxes(title_text="x", showgrid=True, gridwidth=1, gridcolor='LightGray')
fig.update_yaxes(title_text="y", showgrid=True, gridwidth=1, gridcolor='LightGray')

fig.show() # Em ambiente real
```

 Copiar

• **Matplotlib:**

- **Descrição:** Biblioteca para visualizações estáticas
- **Arquitetura:**
  - Figure: Contêiner de alto nível
  - Axes: Área individual de plotagem
  - Axis: Eixos x e y
  - Artist: Elementos visuais (linhas, pontos, textos)
- **Personalização Avançada:**
  - Estilos: plt.style.use()
  - Cores e paletas: cmap
  - Anotações e textos: annotate(), text()
  - Layouts complexos: GridSpec
  - Elementos 3D: Axes3D

• **Exemplo Avançado:**

```
python

import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import numpy as np
from mpl_toolkits.mplot3d import Axes3D

# Configurar estilo
plt.style.use('seaborn-v0_8-whitegrid')

# Criar figura com layout complexo
fig = plt.figure(figsize=(12, 8))
gs = gridspec.GridSpec(2, 3, height_ratios=[1, 1.5])

# Gráfico de linha
ax1 = fig.add_subplot(gs[0, 0])
x = np.linspace(0, 10, 100)
ax1.plot(x, np.sin(x), 'r-', label='sin(x)')
ax1.set_title('Função Seno')
ax1.legend()

# Gráfico de barras
ax2 = fig.add_subplot(gs[0, 1:])
categorias = ['A', 'B', 'C', 'D', 'E']
valores = [3, 7, 2, 5, 9]
ax2.bar(categorias, valores, color='skyblue')
ax2.set_title('Gráfico de Barras')

# Gráfico 3D
ax3 = fig.add_subplot(gs[1, :], projection='3d')
x = np.linspace(-5, 5, 50)
y = np.linspace(-5, 5, 50)
X, Y = np.meshgrid(x, y)
Z = np.sin(np.sqrt(X**2 + Y**2))
surf = ax3.plot_surface(X, Y, Z, cmap='viridis', edgecolor='none')
ax3.set_title('Superfície 3D')
fig.colorbar(surf, ax=ax3, shrink=0.5, aspect=10)

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

 Copiar

5. Desenvolvimento Web

Frontend

• **HTML/CSS:**

- **Descrição:** Linguagens fundamentais para estrutura e estilo de páginas web

• **HTML5 Features:**

- Elementos semânticos: <header> , <footer> , <nav> , <section> , <article>
- Elementos de mídia: <audio> , <video> , <canvas>
- Formulários avançados: novos tipos de input, validação
- APIs web: Storage, Geolocation, Drag and Drop

• **CSS3 Features:**

- Flexbox: Layout unidimensional
- Grid: Layout bidimensional
- Transformações: translate, rotate, scale, skew
- Transições e animações
- Media queries: Design responsivo
- Variáveis CSS: Personalização dinâmica

• **Exemplo Básico:**

```
html

<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Exemplo HTML/CSS</title>
  <style>
    :root {
      --cor-primaria: #3498db;
      --cor-secundaria: #2ecc71;
    }

    body {
      font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
      line-height: 1.6;
      max-width: 800px;
      margin: 0 auto;
      padding: 20px;
    }

    header {
      background-color: var(--cor-primaria);
      color: white;
      padding: 1rem;
      border-radius: 5px;
      margin-bottom: 20px;
    }

    .container {
      display: grid;
      grid-template-columns: 1fr 1fr;
      gap: 20px;
    }

    .card {
      border: 1px solid #ddd;
      border-radius: 5px;
      padding: 15px;
      transition: transform 0.3s ease;
    }

    .card:hover {
      transform: translateY(-5px);
      box-shadow: 0 5px 15px rgba(0,0,0,0.1);
    }

    @media (max-width: 600px) {
      .container {
        grid-template-columns: 1fr;
      }
    }
  </style>
</head>
<body>
  <header>
    <h1>Exemplo de HTML5 e CSS3</h1>
  </header>

  <main>
    <section class="container">
      <div class="card">
        <h2>Card 1</h2>
        <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.</p>
      </div>
      <div class="card">
        <h2>Card 2</h2>
        <p>Sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.</p>
      </div>
    </section>
  </main>

  <footer>
    <p>&copy; 2025 Exemplo HTML/CSS</p>
  </footer>
</body>
</html>
```

 Copiar

• **Tailwind CSS:**

- **Descrição:** Framework CSS baseado em classes utilitárias

• **Características:**

- Abordagem "utility-first": Classes pequenas e específicas
- Design responsivo integrado
- Personalização via configuração
- Baixo overhead de CSS

• **Classes Comuns:**

- Layout: container , flex , grid , block , hidden
- Flexbox: flex-row , flex-col , justify-center , items-center

- Grid: `grid-cols-{n}` , `gap-{n}` , `col-span-{n}`
- Espaçamento: `p-{n}` , `m-{n}` , `space-x-{n}` , `space-y-{n}`
- Tipografia: `text-{size}` , `font-{weight}` , `text-{color}` , `text-center`
- Cores: `bg-{color}` , `text-{color}` , `border-{color}`
- Bordas: `border` , `border-{width}` , `rounded-{size}`
- Efeitos: `shadow-{size}` , `opacity-{value}` , `blur-{amount}`
- Transições: `transition` , `duration-{time}` , `ease-{type}`
- Responsividade: `sm:` , `md:` , `lg:` , `xl:` , `2xl:`

• **Exemplo Básico:**

html

 Copiar

```
<div class="container mx-auto px-4 py-8">
  <header class="bg-blue-500 text-white p-4 rounded-lg mb-6">
    <h1 class="text-2xl font-bold">Dashboard</h1>
  </header>

  <main class="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-6">
    <div class="bg-white p-6 rounded-lg shadow-md hover:shadow-lg transition duration-300">
      <h2 class="text-xl font-semibold text-gray-800 mb-2">Card 1</h2>
      <p class="text-gray-600">Conteúdo do card com classes utilitárias do Tailwind.</p>
      <button class="mt-4 bg-blue-500 hover:bg-blue-600 text-white px-4 py-2 rounded transition">
        Ação
      </button>
    </div>

    <div class="bg-white p-6 rounded-lg shadow-md hover:shadow-lg transition duration-300">
      <h2 class="text-xl font-semibold text-gray-800 mb-2">Card 2</h2>
      <p class="text-gray-600">Design responsivo sem escrever CSS personalizado.</p>
      <button class="mt-4 bg-green-500 hover:bg-green-600 text-white px-4 py-2 rounded transition">
        Ação
      </button>
    </div>

    <div class="bg-white p-6 rounded-lg shadow-md hover:shadow-lg transition duration-300">
      <h2 class="text-xl font-semibold text-gray-800 mb-2">Card 3</h2>
      <p class="text-gray-600">Facilmente extensível e customizável.</p>
      <button class="mt-4 bg-purple-500 hover:bg-purple-600 text-white px-4 py-2 rounded transition">
        Ação
      </button>
    </div>
  </main>

  <footer class="mt-8 text-center text-gray-500 text-sm">
    &copy; 2025 Exemplo Tailwind CSS
  </footer>
</div>
```

• **Limitações no Ambiente:**

- Não suporta valores arbitrários como `h-[600px]`
- Disponível apenas via Artifact para componentes TSX/HTML

• **JavaScript:**

- **Descrição:** Linguagem de programação para web

• **Características Modernas (ES6+):**

- Arrow functions: `() => {}`
- Template literals: ``texto ${variavel}``
- Desestruturação: `const { prop1, prop2 } = objeto`
- Spread/rest: `...array` , `...objeto`
- Classes: `class Exemplo {}`
- Módulos: `import` , `export`
- Promises: `new Promise()` , `async/await`

• **APIs Web:**

- DOM: Manipulação de elementos HTML
- Fetch: Requisições HTTP
- localStorage/sessionStorage: Armazenamento no navegador
- Canvas: Desenho 2D
- Web APIs: Geolocation, WebSockets, Web Workers

• **Exemplo Básico:**

javascript

 Copiar

```
// Seleção de elementos
const form = document.querySelector('#form');
const input = document.querySelector('#input');
const lista = document.querySelector('#lista');

// Array para armazenar itens
let itens = JSON.parse(localStorage.getItem('itens')) || [];

// Renderizar itens
const renderizarItens = () => {
  lista.innerHTML = '';

  itens.forEach((item, index) => {
    const li = document.createElement('li');
    li.className = 'item';
    li.innerHTML = `
      <span>${item.texto}</span>
      <button data-index="${index}" class="btn-remover">Remover</button>
    `;
    lista.appendChild(li);
  });
};

// Adicionar item
form.addEventListener('submit', (e) => {
  e.preventDefault();

  const texto = input.value.trim();
  if (texto === '') return;

  itens.push({ texto, concluido: false });
```

```

localStorage.setItem('itens', JSON.stringify(itens));

input.value = '';
renderizarItens();
});

// Remover item
lista.addEventListener('click', (e) => {
  if (e.target.classList.contains('btn-remover')) {
    const index = parseInt(e.target.dataset.index);
    itens.splice(index, 1);
    localStorage.setItem('itens', JSON.stringify(itens));
    renderizarItens();
  }
});

// Inicializar
renderizarItens();

```

#### • DOM:

- **Descrição:** Interface de programação para documentos HTML e XML

#### • Operações Principais:

- Seleção de elementos: `querySelector()`, `getElementById()`, `getElementsByClassName()`
- Manipulação de conteúdo: `innerHTML`, `textContent`, `innerText`
- Manipulação de atributos: `setAttribute()`, `getAttribute()`, `removeAttribute()`
- Manipulação de classes: `classList.add()`, `classList.remove()`, `classList.toggle()`
- Manipulação de estilos: `style.property`
- Eventos: `addEventListener()`, `removeEventListener()`
- Navegação: `parentNode`, `children`, `nextSibling`, `previousSibling`

#### • Exemplo Básico:

```

javascript

// Seleção de elementos
const titulo = document.querySelector('h1');
const botoes = document.querySelectorAll('.btn');
const container = document.getElementById('container');

// Manipulação de conteúdo
titulo.textContent = 'Novo Título';

// Manipulação de atributos
const imagem = document.querySelector('img');
imagem.setAttribute('src', 'nova-imagem.jpg');
imagem.alt = 'Descrição da imagem';

// Manipulação de classes
container.classList.add('ativo');
container.classList.remove('inativo');
container.classList.toggle('visivel');

// Manipulação de estilos
titulo.style.color = 'blue';
titulo.style.fontSize = '24px';

// Eventos
botoes.forEach(botao => {
  botao.addEventListener('click', (evento) => {
    console.log('Botão clicado:', evento.target.textContent);
    evento.target.classList.add('clicado');
  });
});

// Criação de elementos
const novoElemento = document.createElement('div');
novoElemento.className = 'card';
novoElemento.innerHTML = 'ch2>Novo Card</h2><p>Conteúdo do card</p>';
container.appendChild(novoElemento);

```

 Copiar

#### • React:

- **Descrição:** Biblioteca JavaScript para construção de interfaces

#### • Conceitos Fundamentais:

- Componentes: Blocos de construção reutilizáveis
- Props: Dados passados de componente pai para filho
- Estado: Dados gerenciados internamente pelo componente
- Ciclo de vida: Montagem, atualização, desmontagem
- Renderização condicional: Exibição baseada em condições
- Listas e chaves: Renderização de coleções

#### • Hooks:

- `useState`: Gerenciamento de estado
- `useEffect`: Efeitos colaterais
- `useContext`: Contexto compartilhado
- `useRef`: Referências persistentes
- `useMemo`: Memorização de valores
- `useCallback`: Memorização de funções

#### • Exemplo Básico (TSX):

```

tsx

import React, { useState, useEffect } from 'react';

interface TodoItem {
  id: number;
  text: string;
  completed: boolean;
}

const TodoApp: React.FC = () => {
  const [todos, setTodos] = useState<TodoItem[]>([]);
  const [input, setInput] = useState('');
  const [filter, setFilter] = useState<'all' | 'active' | 'completed'>('all');

  // Adicionar todo

```

 Copiar

```

const handleAddTodo = () => {
  if (input.trim() === '') return;

  const newTodo: TodoItem = {
    id: Date.now(),
    text: input.trim(),
    completed: false
  };

  setTodos([...todos, newTodo]);
  setInput('');
};

// Alternar status
const toggleTodo = (id: number) => {
  setTodos(todos.map(todo => {
    todo.id === id ? { ...todo, completed: !todo.completed } : todo
  }));
};

// Remover todo
const removeTodo = (id: number) => {
  setTodos(todos.filter(todo => todo.id !== id));
};

// Filtrar todos
const filteredTodos = todos.filter(todo => {
  if (filter === 'active') return !todo.completed;
  if (filter === 'completed') return todo.completed;
  return true;
});

// Efeito para salvar no localStorage
useEffect(() => {
  const savedTodos = localStorage.getItem('todos');
  if (savedTodos) {
    setTodos(JSON.parse(savedTodos));
  }
}, []);

useEffect(() => {
  localStorage.setItem('todos', JSON.stringify(todos));
}, [todos]);

return (
  <div className="p-4 max-w-md mx-auto">
    <h1 className="text-2xl font-bold mb-4">Lista de Tarefas</h1>

    <div className="flex mb-4">
      <input
        type="text"
        value={input}
        onChange={(e) => setInput(e.target.value)}
        className="flex-1 border p-2 rounded-l"
        placeholder="Adicionar tarefa..."
      />
      <button
        onClick={handleAddTodo}
        className="bg-blue-500 text-white px-4 py-2 rounded-r"
      >
        Adicionar
      </button>
    </div>

    <div className="flex space-x-2 mb-4">
      <button
        onClick={() => setFilter('all')}
        className={`${filter === 'all' ? 'bg-blue-500 text-white' : 'bg-gray-200'} px-3 py-1 rounded`}
      >
        Todas
      </button>
      <button
        onClick={() => setFilter('active')}
        className={`${filter === 'active' ? 'bg-blue-500 text-white' : 'bg-gray-200'} px-3 py-1 rounded`}
      >
        Ativas
      </button>
      <button
        onClick={() => setFilter('completed')}
        className={`${filter === 'completed' ? 'bg-blue-500 text-white' : 'bg-gray-200'} px-3 py-1 rounded`}
      >
        Concluídas
      </button>
    </div>

    <ul className="space-y-2">
      {filteredTodos.map(todo => (
        <li
          key={todo.id}
          className="flex items-center justify-between p-3 border rounded"
        >
          <div className="flex items-center">
            <input
              type="checkbox"
              checked={todo.completed}
              onChange={() => toggleTodo(todo.id)}
              className="mr-2"
            />
            <span className={todo.completed ? 'line-through text-gray-500' : ''}>
              {todo.text}
            </span>
          </div>
          <button
            onClick={() => removeTodo(todo.id)}
            className="text-red-500"
          >
            Remover
          </button>
        </li>
      ))}
    </ul>

    <div className="mt-4 text-sm text-gray-500">
      {todos.length} itens • {todos.filter(t => !t.completed).length} restantes
    </div>
  </div>
);
};

export default TodoApp;

```

## Ferramentas Web

- **Artifact:**



- **Descrição:** Sintaxe especial para gerar conteúdo como SVG, HTML, etc.

• **Estrutura:**

```
yaml

id: identificador_unico
name: nome_de_exibicao
type: tipo_de_conteudo
content: |-
  conteúdo_aqui
```

Copiar

• **Tipos Suportados:**

- svg : Gráficos vetoriais
- code.[lang] : Blocos de código com destaque de sintaxe
- mermaid : Diagramas Mermaid
- plantuml : Diagramas UML
- html : Conteúdo HTML
- tsx : Componentes React
- markdown : Conteúdo formatado

• **Exemplo de HTML:**

```
php

id: pagina_exemplo
name: Página de Exemplo
type: html
content: |-
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Exemplo de Página</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      line-height: 1.6;
      max-width: 800px;
      margin: 0 auto;
      padding: 20px;
    }
    header {
      background-color: #4CAF50;
      color: white;
      padding: 1rem;
      text-align: center;
    }
    .container {
      display: flex;
      flex-wrap: wrap;
      gap: 20px;
      margin-top: 20px;
    }
    .card {
      flex: 1 1 300px;
      border: 1px solid #ddd;
      border-radius: 5px;
      padding: 15px;
    }
  </style>
</head>
<body>
  <header>
    <h1>Página de Exemplo</h1>
  </header>

  <main>
    <p>Esta é uma página de exemplo gerada com Artifact.</p>

    <div class="container">
      <div class="card">
        <h2>Seção 1</h2>
        <p>Conteúdo da primeira seção.</p>
      </div>
      <div class="card">
        <h2>Seção 2</h2>
        <p>Conteúdo da segunda seção.</p>
      </div>
    </div>

  <footer>
    <p>&copy; 2025 Exemplo Artifact</p>
  </footer>
</body>
</html>
```

Copiar

• **UUID:**

- **Descrição:** Geração de identificadores únicos
- **Formatos:**
  - UUID v1: Baseado em timestamp e endereço MAC
  - UUID v3: Baseado em namespace e nome (MD5)
  - UUID v4: Aleatório
  - UUID v5: Baseado em namespace e nome (SHA-1)

• **Exemplo em Python:**

```
python

import uuid

# UUID v4 (aleatório)
id_aleatorio = uuid.uuid4()
print(f"UUID v4: {id_aleatorio}")

# UUID v5 (baseado em namespace e nome)
namespace = uuid.NAMESPACE_DNS
nome = "exemplo.com"
id_namespace = uuid.uuid5(namespace, nome)
print(f"UUID v5: {id_namespace}")
```

Copiar

continue

- **Execução de Código:**
    - **Python:** Executável via Code Interpreter
- ## Ambiente de Código e Programação Disponível (Continuação)

### 6. Limitações do Ambiente (Continuação)

- **Tempo de Execução:** Execução limitada a 120 segundos
- **Memória:** Limite de memória para execução de processos
- **Processamento:** Recursos computacionais compartilhados
- **Paralelismo:** Limitado a processamento sequencial na maioria dos casos
- **Persistência:** Estado não é mantido entre sessões diferentes
- **Acesso a Recursos:**
  - **Internet:** Sem acesso direto à internet para execução de código
  - **Instalação de Pacotes:** Limitado aos pacotes pré-instalados
  - **Sistema de Arquivos:** Acesso apenas ao ambiente sandbox
  - **APIs Externas:** Sem conexão direta a serviços externos
  - **Bancos de Dados:** Sem conexão direta a bancos de dados externos
  - **WebSockets:** Sem suporte a conexões persistentes
- **Renderização:**
  - **Visualizações Interativas:** Limitadas às capacidades do ambiente
  - **Aplicações Web:** Apenas geração de código, sem hospedagem
  - **Interfaces Gráficas:** Sem suporte a GUI tradicional
  - **Vídeo/Áudio:** Sem capacidade de reprodução direta

### 7. Análise de Dados Avançada

#### Manipulação de Dados Complexos

- **pandas Avançado:**

- **Operações com MultiIndex:**

```
python

import pandas as pd
import numpy as np

# Criar MultiIndex
arrays = [
    ['A', 'A', 'B', 'B'],
    ['um', 'dois', 'um', 'dois']
]
index = pd.MultiIndex.from_arrays(arrays, names=('letra', 'numero'))

# DataFrame com MultiIndex
df = pd.DataFrame({
    'valor1': np.random.randn(4),
    'valor2': np.random.randn(4)
}, index=index)

# Seleção com MultiIndex
df.loc['A'] # Selecionar todos os itens do nível 'A'
df.loc['A', 'um'] # Selecionar item específico

# Operações por nível
df.groupby(level=0).sum() # Agrupar pelo primeiro nível
```

 Copiar

- **Janelas Deslizantes:**

```
python

import pandas as pd
import numpy as np

# Criar série temporal
datas = pd.date_range('20230101', periods=100)
serie = pd.Series(np.random.randn(100).cumsum(), index=datas)

# Média móvel
media_movel = serie.rolling(window=7).mean()

# Janela expansiva
media_expansiva = serie.expanding().mean()

# Média móvel exponencialmente ponderada
media_exp = serie.ewm(span=7).mean()
```

 Copiar

- **Operações com Strings:**

```
python

import pandas as pd

# DataFrame com strings
df = pd.DataFrame({
    'texto': ['abc_123', 'def_456', 'ghi_789', 'jkl_000'],
    'email': ['usuario1@exemplo.com', 'usuario2@exemplo.com',
             'usuario3@exemplo.com', 'usuario4@exemplo.com']
})

# Operações com strings
df['prefixo'] = df['texto'].str.split('_').str[0]
df['sufixo'] = df['texto'].str.split('_').str[1]
df['dominio'] = df['email'].str.extract(r'@(.+)\.')
df['maiusculo'] = df['texto'].str.upper()
```

 Copiar

- **Operações com Datas:**

```
python

import pandas as pd
import numpy as np

# DataFrame com datas
df = pd.DataFrame({
    'data': pd.date_range('20230101', periods=100),
    'valor': np.random.randn(100).cumsum()
})

# Extrair componentes de data
```

 Copiar

```
df['ano'] = df['data'].dt.year
df['mes'] = df['data'].dt.month
df['dia'] = df['data'].dt.day
df['dia_semana'] = df['data'].dt.day_name()

# Filtrar por período
inicio = pd.Timestamp('20230115')
fim = pd.Timestamp('20230131')
filtro_periodo = df[(df['data'] >= inicio) & (df['data'] <= fim)]

# Agrupar por período
media_mensal = df.groupby(df['data'].dt.to_period('M')).mean()
```

## Análise Estatística

### • Estatística Descritiva:

python

 Copiar

```
import pandas as pd
import numpy as np
import scipy.stats as stats

# Gerar dados
np.random.seed(42)
dados = np.random.normal(100, 15, 1000)

# Estatísticas básicas
media = np.mean(dados)
mediana = np.median(dados)
desvio_padrao = np.std(dados)
variancia = np.var(dados)
minimo = np.min(dados)
maximo = np.max(dados)

# Quartis e percentis
q1 = np.percentile(dados, 25)
q3 = np.percentile(dados, 75)
iqr = q3 - q1

# Assimetria e curtose
assimetria = stats.skew(dados)
curtose = stats.kurtosis(dados)

# Resumo estatístico com pandas
df = pd.DataFrame({'valor': dados})
resumo = df.describe()
```

### • Testes de Hipótese:

python

 Copiar

```
import numpy as np
import scipy.stats as stats

# Gerar duas amostras
np.random.seed(42)
grupo_a = np.random.normal(100, 15, 100)
grupo_b = np.random.normal(105, 15, 100)

# Teste t para amostras independentes
t_stat, p_valor = stats.ttest_ind(grupo_a, grupo_b)
print(f"Estatística t: {t_stat}, p-valor: {p_valor}")

# Teste de normalidade (Shapiro-Wilk)
_, p_normal_a = stats.shapiro(grupo_a)
_, p_normal_b = stats.shapiro(grupo_b)
print(f"Teste de normalidade A: p-valor = {p_normal_a}")
print(f"Teste de normalidade B: p-valor = {p_normal_b}")

# Teste não-paramétrico (Mann-Whitney)
_, p_mw = stats.mannwhitneyu(grupo_a, grupo_b)
print(f"Teste Mann-Whitney: p-valor = {p_mw}")

# ANOVA para múltiplos grupos
grupo_c = np.random.normal(110, 15, 100)
_, p_anova = stats.f_oneway(grupo_a, grupo_b, grupo_c)
print(f"ANOVA: p-valor = {p_anova}")
```

### • Correlação e Regressão:

python

 Copiar

```
import numpy as np
import pandas as pd
import scipy.stats as stats
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

# Gerar dados correlacionados
np.random.seed(42)
x = np.random.uniform(0, 10, 100)
y = 2 * x + 1 + np.random.normal(0, 2, 100)

# Correlação de Pearson
corr, p_valor = stats.pearsonr(x, y)
print(f"Correlação de Pearson: {corr:.4f}, p-valor: {p_valor:.4f}")

# Correlação de Spearman (rank)
corr_s, p_valor_s = stats.spearmanr(x, y)
print(f"Correlação de Spearman: {corr_s:.4f}, p-valor: {p_valor_s:.4f}")

# Regressão linear
X = x.reshape(-1, 1) # Reshape para formato esperado pelo sklearn
modelo = LinearRegression()
modelo.fit(X, y)

# Coeficientes
print(f"Intercepto: {modelo.intercept_:.4f}")
print(f"Coeficiente: {modelo.coef_[0]:.4f}")

# R²
r2 = modelo.score(X, y)
print(f"R²: {r2:.4f}")

# Visualização
plt.figure(figsize=(10, 6))
plt.scatter(x, y, alpha=0.7)
plt.plot(x, modelo.predict(X), 'r-', linewidth=2)
plt.title(f'Regressão Linear (y = {modelo.coef_[0]:.2f}x + {modelo.intercept_:.2f})')
plt.xlabel('x')
plt.ylabel('y')
plt.grid(True, alpha=0.3)
plt.show()
```

## Machine Learning Básico

### • Classificação:

```
python

import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Gerar dados sintéticos
np.random.seed(42)
n_samples = 1000

# Características
X = np.random.normal(size=(n_samples, 2))

# Gerar alvos (círculo)
y = (X[:, 0]**2 + X[:, 1]**2 < 1).astype(int)

# Adicionar ruído
y = np.logical_xor(y, np.random.random(size=n_samples) < 0.1).astype(int)

# Dividir em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Normalização
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Treinar modelo
modelo = RandomForestClassifier(n_estimators=100, random_state=42)
modelo.fit(X_train_scaled, y_train)

# Prever
y_pred = modelo.predict(X_test_scaled)

# Avaliar
acuracia = accuracy_score(y_test, y_pred)
print(f"Acurácia: {acuracia:.4f}")
print("\nRelatório de Classificação:")
print(classification_report(y_test, y_pred))

# Matriz de confusão
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Matriz de Confusão')
plt.xlabel('Previsto')
plt.ylabel('Real')
plt.show()

# Visualizar fronteira de decisão
def plot_decision_boundary(X, y, model, scaler):
    h = 0.02 # Tamanho da grade
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

    # Escalar pontos da grade
    grid_scaled = scaler.transform(np.c_[xx.ravel(), yy.ravel()])

    # Prever classes
    Z = model.predict(grid_scaled)
    Z = Z.reshape(xx.shape)

    # Plotar
    plt.figure(figsize=(10, 8))
    plt.contourf(xx, yy, Z, alpha=0.3, cmap='RdBu')
    plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='RdBu', edgecolor='k')
    plt.title('Fronteira de Decisão')
    plt.xlabel('Característica 1')
    plt.ylabel('Característica 2')
    plt.show()

plot_decision_boundary(X, y, modelo, scaler)
```

 Copiar

### • Clustering:

```
python

import numpy as np
import pandas as pd
from sklearn.cluster import KMeans, DBSCAN
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt

# Gerar dados com clusters
np.random.seed(42)

# Cluster 1
n1 = 200
x1 = np.random.normal(0, 0.5, n1)
y1 = np.random.normal(0, 0.5, n1)

# Cluster 2
n2 = 200
x2 = np.random.normal(3, 0.5, n2)
y2 = np.random.normal(0, 0.5, n2)

# Cluster 3
n3 = 200
x3 = np.random.normal(1.5, 0.5, n3)
y3 = np.random.normal(2, 0.5, n3)

# Combinar dados
X = np.vstack([np.column_stack((x1, y1)),
               np.column_stack((x2, y2)),
               np.column_stack((x3, y3))])

# Normalizar
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# K-Means
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans_labels = kmeans.fit_predict(X_scaled)

# DBSCAN
```

 Copiar

```

dbscan = DBSCAN(eps=0.5, min_samples=10)
dbscan_labels = dbscan.fit_predict(X_scaled)

# Avaliar K-Means
silhouette_kmeans = silhouette_score(X_scaled, kmeans_labels)
print(f"Silhouette Score (K-Means): {silhouette_kmeans:.4f}")

# Avaliar DBSCAN (ignorando ruído)
if np.any(dbscan_labels != -1):
    silhouette_dbscan = silhouette_score(
        X_scaled[dbscan_labels != -1],
        dbscan_labels[dbscan_labels != -1]
    )
    print(f"Silhouette Score (DBSCAN): {silhouette_dbscan:.4f}")

# Visualizar resultados
fig, axes = plt.subplots(1, 2, figsize=(14, 6))

# K-Means
axes[0].scatter(X[:, 0], X[:, 1], c=kmeans_labels, cmap='viridis', s=50, alpha=0.8)
axes[0].scatter(kmeans.cluster_centers[:, 0], kmeans.cluster_centers[:, 1],
                c='red', marker='x', s=200, alpha=0.8)
axes[0].set_title('K-Means Clustering')

# DBSCAN
axes[1].scatter(X[:, 0], X[:, 1], c=dbscan_labels, cmap='viridis', s=50, alpha=0.8)
axes[1].set_title('DBSCAN Clustering')

plt.tight_layout()
plt.show()

```

## 8. Desenvolvimento de Aplicações

### Dashboards e Visualizações Interativas

- Dash:

```

python


Copiar


import dash
from dash import dcc, html
from dash.dependencies import Input, Output
import plotly.express as px
import pandas as pd
import numpy as np

# Gerar dados de exemplo
np.random.seed(42)
datas = pd.date_range('20230101', periods=100)
df = pd.DataFrame({
    'data': datas,
    'vendas': np.random.normal(100, 15, 100).cumsum(),
    'custos': np.random.normal(80, 10, 100).cumsum(),
    'regiao': np.random.choice(['Norte', 'Sul', 'Leste', 'Oeste'], 100)
})

# Inicializar app
app = dash.Dash(__name__)

# Layout
app.layout = html.Div([
    html.H1("Dashboard de Vendas", style={'textAlign': 'center'}),

    html.Div([
        html.Label("Selecione a Região:"),
        dcc.Dropdown(
            id='regiao-dropdown',
            options=[{'label': r, 'value': r} for r in df['regiao'].unique()],
            value=None,
            multi=True,
            placeholder="Todas as Regiões"
        ),

        html.Label("Intervalo de Datas:"),
        dcc.DatePickerRange(
            id='date-picker',
            start_date=df['data'].min(),
            end_date=df['data'].max(),
            display_format='DD/MM/YYYY'
        )
    ], style={'padding': '20px', 'backgroundColor': '#f8f9fa', 'borderRadius': '5px'}),

    html.Div([
        dcc.Graph(id='vendas-grafico')
    ], style={'padding': '20px'}),

    html.Div([
        html.Div([
            dcc.Graph(id='vendas-por-regiao')
        ], style={'width': '48%', 'display': 'inline-block'}),

        html.Div([
            dcc.Graph(id='lucro-grafico')
        ], style={'width': '48%', 'display': 'inline-block', 'float': 'right'})
    ])
], style={'padding': '20px', 'fontFamily': 'Arial'})

# Callbacks
@app.callback(
    [Output('vendas-grafico', 'figure'),
     Output('vendas-por-regiao', 'figure'),
     Output('lucro-grafico', 'figure')],
    [Input('regiao-dropdown', 'value'),
     Input('date-picker', 'start_date'),
     Input('date-picker', 'end_date')]
)
def update_graphs(regioes_selecionadas, data_inicio, data_fim):
    # Filtrar por data
    mask_data = (df['data'] >= data_inicio) & (df['data'] <= data_fim)
    df_filtrado = df[mask_data]

    # Filtrar por região
    if regioes_selecionadas and len(regioes_selecionadas) > 0:
        df_filtrado = df_filtrado[df_filtrado['regiao'].isin(regioes_selecionadas)]

    # Gráfico de vendas ao longo do tempo
    fig_vendas = px.line(
        df_filtrado,
        x='data',
        y='vendas',
        title='Vendas ao Longo do Tempo',
        labels={'data': 'Data', 'vendas': 'Vendas (R$)'}
    )

    # Gráfico de vendas por região

```

```

fig_regiao = px.bar(
    df_filtrado.groupby('regiao').sum().reset_index(),
    x='regiao',
    y='vendas',
    title='Vendas por Região',
    labels={'regiao': 'Região', 'vendas': 'Vendas (R$)'},
    color='regiao'
)

# Gráfico de lucro
df_filtrado['lucro'] = df_filtrado['vendas'] - df_filtrado['custos']
fig_lucro = px.area(
    df_filtrado,
    x='data',
    y='lucro',
    title='Lucro ao Longo do Tempo',
    labels={'data': 'Data', 'lucro': 'Lucro (R$)'})

)

return fig_vendas, fig_regiao, fig_lucro

# Em ambiente real: app.run_server(debug=True)

```

## Aplicações Web React

### • Exemplo de Aplicação TSX:

```

tsx
Copiar

import React, { useState, useEffect } from 'react';
import { Card } from "@components/ui/card";
import { Button } from "@components/ui/button";
import { Input } from "@components/ui/input";
import { Select } from "@components/ui/select";
import { LineChart, Line, XAxis, YAxis, CartesianGrid, Tooltip, Legend, ResponsiveContainer } from 'recharts';

// Tipos
interface Task {
  id: string;
  title: string;
  status: 'pending' | 'in-progress' | 'completed';
  priority: 'low' | 'medium' | 'high';
  dueDate: string;
}

interface ChartData {
  name: string;
  pending: number;
  inProgress: number;
  completed: number;
}

const TaskManager: React.FC = () => {
  // Estados
  const [tasks, setTasks] = useState<Task[]>([]);
  const [newTask, setNewTask] = useState('');
  const [filter, setFilter] = useState<'all' | 'pending' | 'in-progress' | 'completed'>('all');
  const [chartData, setChartData] = useState<ChartData[]>([]);

  // Efeito para gerar dados iniciais
  useEffect(() => {
    const initialTasks: Task[] = [
      {
        id: '1',
        title: 'Completar relatório',
        status: 'pending',
        priority: 'high',
        dueDate: '2025-07-20'
      },
      {
        id: '2',
        title: 'Preparar apresentação',
        status: 'in-progress',
        priority: 'medium',
        dueDate: '2025-07-25'
      },
      {
        id: '3',
        title: 'Revisar documentação',
        status: 'completed',
        priority: 'low',
        dueDate: '2025-07-15'
      }
    ];

    setTasks(initialTasks);
  }, []);

  // Efeito para atualizar dados do gráfico
  useEffect(() => {
    // Gerar dados para os últimos 7 dias
    const today = new Date();
    const data: ChartData[] = [];

    for (let i = 6; i >= 0; i--) {
      const date = new Date();
      date.setDate(today.getDate() - i);
      const dateStr = date.toLocaleDateString('pt-BR', { month: 'short', day: 'numeric' });

      data.push({
        name: dateStr,
        pending: Math.floor(Math.random() * 5) + 1,
        inProgress: Math.floor(Math.random() * 4) + 1,
        completed: Math.floor(Math.random() * 6) + 1
      });
    }

    setChartData(data);
  }, []);

  // Adicionar tarefa
  const handleAddTask = () => {
    if (newTask.trim() === '') return;

    const task: Task = {
      id: Date.now().toString(),
      title: newTask.trim(),
      status: 'pending',
      priority: 'medium',
      dueDate: new Date(Date.now() + 7 * 24 * 60 * 60 * 1000).toISOString().split('T')[0]
    };

    setTasks([...tasks, task]);
    setNewTask('');
  };
}

```

```

// Atualizar status da tarefa
const updateTaskStatus = (id: string, status: Task['status']) => {
  setTasks(tasks.map(task =>
    task.id === id ? { ...task, status } : task
  ));
};

// Remover tarefa
const removeTask = (id: string) => {
  setTasks(tasks.filter(task => task.id !== id));
};

// Filtrar tarefas
const filteredTasks = tasks.filter(task => {
  if (filter === 'all') return true;
  return task.status === filter;
});

return (
  <div className="p-6 max-w-6xl mx-auto">
    <h1 className="text-3xl font-bold mb-6">Gerenciador de Tarefas</h1>

    {/ * Estatísticas */}
    <div className="grid grid-cols-1 md:grid-cols-4 gap-4 mb-6">
      <Card className="p-4 bg-blue-50">
        <h3 className="text-lg font-medium">Total</h3>
        <p className="text-2xl font-bold">{tasks.length}</p>
      </Card>
      <Card className="p-4 bg-yellow-50">
        <h3 className="text-lg font-medium">Pendentes</h3>
        <p className="text-2xl font-bold">{tasks.filter(t => t.status === 'pending').length}</p>
      </Card>
      <Card className="p-4 bg-purple-50">
        <h3 className="text-lg font-medium">Em Progresso</h3>
        <p className="text-2xl font-bold">{tasks.filter(t => t.status === 'in-progress').length}</p>
      </Card>
      <Card className="p-4 bg-green-50">
        <h3 className="text-lg font-medium">Concluídas</h3>
        <p className="text-2xl font-bold">{tasks.filter(t => t.status === 'completed').length}</p>
      </Card>
    </div>

    {/ * Gráfico */}
    <Card className="p-4 mb-6">
      <h2 className="text-xl font-bold mb-4">Progresso da Semana</h2>
      <div className="h-64">
        <ResponsiveContainer width="100%" height="100%">
          <LineChart data={chartData}>
            <CartesianGrid strokeDasharray="3 3" />
            <XAxis dataKey="name" />
            <YAxis />
            <Tooltip />
            <Legend />
            <Line type="monotone" dataKey="pending" stroke="#eab308" name="Pendentes" />
            <Line type="monotone" dataKey="inProgress" stroke="#8b5cf6" name="Em Progresso" />
            <Line type="monotone" dataKey="completed" stroke="#22c55e" name="Concluídas" />
          </LineChart>
        </ResponsiveContainer>
      </div>
    </Card>

    {/ * Adicionar Tarefa */}
    <div className="flex gap-2 mb-6">
      <Input
        value={newTask}
        onChange={(e) => setNewTask(e.target.value)}
        placeholder="Nova tarefa..."
        className="flex-1"
      />
      <Button onClick={handleAddTask}>Adicionar</Button>
    </div>

    {/ * Filtros */}
    <div className="flex gap-2 mb-4">
      <Button
        variant={filter === 'all' ? 'default' : 'outline'}
        onClick={() => setFilter('all')}
      >
        Todas
      </Button>
      <Button
        variant={filter === 'pending' ? 'default' : 'outline'}
        onClick={() => setFilter('pending')}
      >
        Pendentes
      </Button>
      <Button
        variant={filter === 'in-progress' ? 'default' : 'outline'}
        onClick={() => setFilter('in-progress')}
      >
        Em Progresso
      </Button>
      <Button
        variant={filter === 'completed' ? 'default' : 'outline'}
        onClick={() => setFilter('completed')}
      >
        Concluídas
      </Button>
    </div>

    {/ * Lista de Tarefas */}
    <div className="space-y-3">
      {filteredTasks.length === 0 ? (
        <p className="text-center text-gray-500 py-4">Nenhuma tarefa encontrada.</p>
      ) : (
        filteredTasks.map(task => (
          <Card key={task.id} className="p-4">
            <div className="flex items-center justify-between">
              <div>
                <h3 className="text-lg font-medium">{
                  task.status === 'completed' ? 'line-through text-gray-500' : ''
                }</h3>
                {task.title}
              </div>
              <div className="flex items-center gap-2 text-sm text-gray-500">
                <span className="px-2 py-1

```

continue

8. Desenvolvimento de Aplicações (Continuação)

Aplicações Web React (Continuação)

- Exemplo de Aplicação TSX (Continuação):

tsx

```
// Continuação do componente anterior
    <span className={`px-2 py-1 rounded-full text-xs ${
      task.priority === 'high' ? 'bg-red-100 text-red-800' :
      task.priority === 'medium' ? 'bg-yellow-100 text-yellow-800' :
      'bg-blue-100 text-blue-800'
    }`}>
      {task.priority === 'high' ? 'Alta' :
      task.priority === 'medium' ? 'Média' : 'Baixa'}
    </span>
    <span>Vencimento: {new Date(task.dueDate).toLocaleDateString('pt-BR')}</span>
  </div>
</div>

<div className="flex items-center gap-2">
  <Select
    value={task.status}
    onChange={(e) => updateTaskStatus(task.id, e.target.value as Task['status'])}
    className="w-32"
  >
    <option value="pending">Pendente</option>
    <option value="in-progress">Em Progresso</option>
    <option value="completed">Concluída</option>
  </Select>

  <Button
    variant="destructive"
    size="sm"
    onClick={() => removeTask(task.id)}
  >
    Remover
  </Button>
</div>
</div>
</div>
</div>
);
};
};

export default TaskManager;
```

Copiar

Processamento de Dados em Lote

- Exemplo de Pipeline de Dados:

python

```
import pandas as pd
import numpy as np
from datetime import datetime, timedelta
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

# Configurações
np.random.seed(42)

# 1. Geração de Dados Simulados
def gerar_dados_vendas(n_dias=180, n_produtos=10, n_lojas=5):
    """Gera dados simulados de vendas para análise."""
    datas = pd.date_range(end=datetime.now(), periods=n_dias).tolist()
    produtos = [f"Produto-{i}" for i in range(1, n_produtos + 1)]
    lojas = [f"Loja-{i}" for i in range(1, n_lojas + 1)]
    categorias = ["Eletrônicos", "Alimentos", "Vestuário", "Casa", "Lazer"]

    registros = []

    for data in datas:
        for produto in np.random.choice(produtos, size=np.random.randint(5, 20)):
            loja = np.random.choice(lojas)
            categoria = np.random.choice(categorias)
            preco_unitario = np.random.uniform(10, 1000)
            quantidade = np.random.randint(1, 10)

            # Adicionar sazonalidade
            if data.month in [11, 12]: # Fim de ano
                quantidade *= np.random.uniform(1.2, 1.5)
            elif data.weekday() >= 5: # Fim de semana
                quantidade *= np.random.uniform(1.1, 1.3)

            registros.append({
                "data": data,
                "produto": produto,
                "loja": loja,
                "categoria": categoria,
                "preco_unitario": preco_unitario,
                "quantidade": quantidade,
                "valor_total": preco_unitario * quantidade
            })

    return pd.DataFrame(registros)

# 2. Processamento e Limpeza de Dados
def processar_dados(df):
    """Processa e limpa os dados para análise."""
    # Cópia para não modificar o original
    df_proc = df.copy()

    # Converter data para datetime se não for
    if not pd.api.types.is_datetime64_any_dtype(df_proc['data']):
        df_proc['data'] = pd.to_datetime(df_proc['data'])

    # Extrair componentes de data
    df_proc['ano'] = df_proc['data'].dt.year
    df_proc['mes'] = df_proc['data'].dt.month
    df_proc['dia'] = df_proc['data'].dt.day
    df_proc['dia_semana'] = df_proc['data'].dt.dayofweek
    df_proc['fim_de_semana'] = df_proc['dia_semana'].apply(lambda x: 1 if x >= 5 else 0)

    # Arredondar valores monetários
    df_proc['preco_unitario'] = df_proc['preco_unitario'].round(2)
```

Copiar



```

df_proc['valor_total'] = df_proc['valor_total'].round(2)

# Remover outliers (valores extremos de quantidade)
q1 = df_proc['quantidade'].quantile(0.01)
q3 = df_proc['quantidade'].quantile(0.99)
iqr = q3 - q1

df_proc = df_proc[
    (df_proc['quantidade'] >= q1 - 1.5 * iqr) &
    (df_proc['quantidade'] <= q3 + 1.5 * iqr)
]

return df_proc

# 3. Análise Exploratória
def analise_exploratoria(df):
    """Realiza análise exploratória dos dados."""
    # Estatísticas descritivas
    print("=== Estatísticas Descritivas ===")
    print(df[['preco_unitario', 'quantidade', 'valor_total']].describe().round(2))

    # Vendas por categoria
    vendas_categoria = df.groupby('categoria')['valor_total'].sum().sort_values(ascending=False)
    print("\n=== Vendas por Categoria ===")
    print(vendas_categoria)

    # Vendas por loja
    vendas_loja = df.groupby('loja')['valor_total'].sum().sort_values(ascending=False)
    print("\n=== Vendas por Loja ===")
    print(vendas_loja)

    # Vendas por mês
    vendas_mes = df.groupby('mes')['valor_total'].sum()
    print("\n=== Vendas por Mês ===")
    print(vendas_mes)

    # Visualizações
    plt.figure(figsize=(12, 8))

    # Gráfico 1: Vendas por categoria
    plt.subplot(2, 2, 1)
    sns.barplot(x=vendas_categoria.index, y=vendas_categoria.values)
    plt.title('Vendas por Categoria')
    plt.xticks(rotation=45)
    plt.tight_layout()

    # Gráfico 2: Vendas por loja
    plt.subplot(2, 2, 2)
    sns.barplot(x=vendas_loja.index, y=vendas_loja.values)
    plt.title('Vendas por Loja')
    plt.tight_layout()

    # Gráfico 3: Vendas por mês
    plt.subplot(2, 2, 3)
    sns.lineplot(x=vendas_mes.index, y=vendas_mes.values, marker='o')
    plt.title('Vendas por Mês')
    plt.xticks(range(1, 13))
    plt.tight_layout()

    # Gráfico 4: Distribuição de preços
    plt.subplot(2, 2, 4)
    sns.histplot(df['preco_unitario'], bins=30, kde=True)
    plt.title('Distribuição de Preços')
    plt.tight_layout()

    plt.show()

    return {
        'vendas_categoria': vendas_categoria,
        'vendas_loja': vendas_loja,
        'vendas_mes': vendas_mes
    }

# 4. Segmentação de Produtos
def segmentar_produtos(df):
    """Segmenta produtos com base em preço e quantidade vendida."""
    # Agregar dados por produto
    produtos_df = df.groupby('produto').agg({
        'preco_unitario': 'mean',
        'quantidade': 'sum',
        'valor_total': 'sum'
    }).reset_index()

    # Normalizar dados para clustering
    features = ['preco_unitario', 'quantidade', 'valor_total']
    scaler = StandardScaler()
    produtos_scaled = scaler.fit_transform(produtos_df[features])

    # Aplicar K-Means
    kmeans = KMeans(n_clusters=4, random_state=42)
    produtos_df['cluster'] = kmeans.fit_predict(produtos_scaled)

    # Visualizar clusters
    plt.figure(figsize=(10, 8))
    sns.scatterplot(
        x='preco_unitario',
        y='quantidade',
        size='valor_total',
        hue='cluster',
        palette='viridis',
        data=produtos_df
    )
    plt.title('Segmentação de Produtos')
    plt.xlabel('Preço Unitário Médio')
    plt.ylabel('Quantidade Vendida')
    plt.tight_layout()
    plt.show()

    # Caracterização dos clusters
    print("\n=== Caracterização dos Clusters ===")
    print(produtos_df.groupby('cluster')[features].mean().round(2))

    return produtos_df

# 5. Previsão de Tendências
def analisar_tendencias(df):
    """Análisa tendências de vendas ao longo do tempo."""
    # Agregar vendas diárias
    vendas_diarias = df.groupby('data')['valor_total'].sum().reset_index()
    vendas_diarias.set_index('data', inplace=True)

    # Média móvel de 7 dias
    vendas_diarias['mm_7d'] = vendas_diarias['valor_total'].rolling(window=7).mean()

    # Média móvel de 30 dias
    vendas_diarias['mm_30d'] = vendas_diarias['valor_total'].rolling(window=30).mean()

```

```

# Visualizar tendências
plt.figure(figsize=(12, 6))
plt.plot(vendas_diarias.index, vendas_diarias['valor_total'], 'b-', alpha=0.3, label='Diário')
plt.plot(vendas_diarias.index, vendas_diarias['mm_7d'], 'r-', label='Média Móvel (7d)')
plt.plot(vendas_diarias.index, vendas_diarias['mm_30d'], 'g-', label='Média Móvel (30d)')
plt.title('Tendência de Vendas ao Longo do Tempo')
plt.xlabel('Data')
plt.ylabel('Valor Total de Vendas')
plt.legend()
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

return vendas_diarias

# 6. Geração de Relatório
def gerar_relatorio(df, resultados):
    """Gera um relatório resumido com os principais insights."""
    total_vendas = df['valor_total'].sum()
    ticket_medio = df['valor_total'].mean()
    n_transacoes = len(df)

    # Produtos mais vendidos
    produtos_mais_vendidos = df.groupby('produto')['quantidade'].sum().sort_values(ascending=False).head(5)

    # Produtos com maior faturamento
    produtos_maior_faturamento = df.groupby('produto')['valor_total'].sum().sort_values(ascending=False).head(5)

    # Dias da semana com mais vendas
    dias_semana = ['Segunda', 'Terça', 'Quarta', 'Quinta', 'Sexta', 'Sábado', 'Domingo']
    vendas_dia_semana = df.groupby('dia_semana')['valor_total'].sum()
    vendas_dia_semana.index = [dias_semana[i] for i in vendas_dia_semana.index]

    print("\n" + "="*50)
    print("RELATÓRIO DE VENDAS")
    print("="*50)

    print(f"\nPeríodo: {df['data'].min().strftime('%d/%m/%Y')} a {df['data'].max().strftime('%d/%m/%Y')}")
    print(f"Total de Vendas: R$ {total_vendas:,.2f}")
    print(f"Número de Transações: {n_transacoes:,}")
    print(f"Ticket Médio: R$ {ticket_medio:,.2f}")

    print("\n--- TOP 5 PRODUTOS MAIS VENDIDOS ---")
    for produto, qtd in produtos_mais_vendidos.items():
        print(f"{produto}: {qtd:,.0f} unidades")

    print("\n--- TOP 5 PRODUTOS COM MAIOR FATURAMENTO ---")
    for produto, valor in produtos_maior_faturamento.items():
        print(f"{produto}: R$ {valor:,.2f}")

    print("\n--- VENDAS POR DIA DA SEMANA ---")
    for dia, valor in vendas_dia_semana.sort_values(ascending=False).items():
        print(f"{dia}: R$ {valor:,.2f}")

    print("\n--- VENDAS POR CATEGORIA ---")
    for categoria, valor in resultados['vendas_categoria'].items():
        print(f"{categoria}: R$ {valor:,.2f}")

    print("\n--- VENDAS POR LOJA ---")
    for loja, valor in resultados['vendas_loja'].items():
        print(f"{loja}: R$ {valor:,.2f}")

    print("\n" + "="*50)

# Pipeline principal
def executar_pipeline():
    """Executa o pipeline completo de análise de dados."""
    print("Iniciando pipeline de análise de dados de vendas...\n")

    # 1. Gerar dados
    print("Gerando dados simulados...")
    dados_brutos = gerar_dados_vendas()
    print(f"Gerados {len(dados_brutos)} registros de vendas.\n")

    # 2. Processar dados
    print("Processando e limpando dados...")
    dados_processados = processar_dados(dados_brutos)
    print(f"Dados processados. Restaram {len(dados_processados)} registros após limpeza.\n")

    # 3. Análise exploratória
    print("Realizando análise exploratória...")
    resultados_exploratorios = analise_exploratoria(dados_processados)

    # 4. Segmentação de produtos
    print("Realizando segmentação de produtos...")
    produtos_segmentados = segmentar_produtos(dados_processados)

    # 5. Análise de tendências
    print("Analisando tendências temporais...")
    tendencias = analisar_tendencias(dados_processados)

    # 6. Gerar relatório
    print("Gerando relatório final...")
    gerar_relatorio(dados_processados, resultados_exploratorios)

    print("\nPipeline concluído com sucesso!")

    return {
        'dados_brutos': dados_brutos,
        'dados_processados': dados_processados,
        'resultados_exploratorios': resultados_exploratorios,
        'produtos_segmentados': produtos_segmentados,
        'tendencias': tendencias
    }

# Executar pipeline
# resultados = executar_pipeline()

```

## 9. Manipulação de Texto e Linguagem Natural

### Processamento de Texto

- **Análise de Sentimento:**

python

 Copiar

```

import pandas as pd
import numpy as np
import re
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from collections import Counter

```

```

import matplotlib.pyplot as plt
import seaborn as sns

# Configurações
np.random.seed(42)

# Dados de exemplo (simulados)
def gerar_comentarios_simulados(n=100):
    """Gera comentários simulados para análise de sentimento."""
    comentarios_positivos = [
        "Adorei o produto, superou minhas expectativas!",
        "Excelente atendimento e entrega rápida.",
        "O melhor produto que já comprei, recomendo muito.",
        "Qualidade impressionante, vale cada centavo.",
        "Estou muito satisfeito com minha compra.",
        "Produto incrível, funcionou perfeitamente.",
        "Atendimento ao cliente nota 10, muito atenciosos.",
        "Chegou antes do prazo e em perfeitas condições.",
        "Ótimo custo-benefício, certamente comprei mais.",
        "Superou todas as minhas expectativas, perfeito!"
    ]

    comentarios_neutros = [
        "O produto chegou conforme o esperado.",
        "Funciona como deveria, nada de especial.",
        "Atendeu às minhas necessidades básicas.",
        "Preço razoável para o que oferece.",
        "Produto ok, mas nada extraordinário.",
        "Entrega dentro do prazo previsto.",
        "Corresponde à descrição do anúncio.",
        "Qualidade média, serve para o propósito.",
        "Nem bom nem ruim, apenas satisfatório.",
        "Embalagem simples mas adequada."
    ]

    comentarios_negativos = [
        "Produto de péssima qualidade, não recomendo.",
        "Chegou com defeito e o suporte não ajudou.",
        "Muito caro para o que oferece, decepcionante.",
        "Não funciona como prometido, perda de dinheiro.",
        "Péssimo atendimento ao cliente, nunca mais compro.",
        "Quebrou após uma semana de uso, qualidade terrível.",
        "Entrega atrasada e produto danificado.",
        "Totalmente diferente da descrição, me senti enganado.",
        "Experiência frustrante, não vale o preço cobrado.",
        "Arrepentido da compra, não recomendo a ninguém."
    ]

    # Distribuição de sentimentos (60% positivo, 20% neutro, 20% negativo)
    todos_comentarios = []
    sentimentos = []

    for _ in range(n):
        rand = np.random.random()
        if rand < 0.6:
            comentario = np.random.choice(comentarios_positivos)
            sentimento = "positivo"
        elif rand < 0.8:
            comentario = np.random.choice(comentarios_neutros)
            sentimento = "neutro"
        else:
            comentario = np.random.choice(comentarios_negativos)
            sentimento = "negativo"

        # Adicionar variações aleatórias
        palavras = comentario.split()
        if len(palavras) > 5 and np.random.random() < 0.3:
            # Remover algumas palavras
            n_remove = np.random.randint(1, 3)
            for _ in range(n_remove):
                if len(palavras) > 3:
                    idx = np.random.randint(0, len(palavras))
                    palavras.pop(idx)

            comentario_modificado = " ".join(palavras)

        # Adicionar erros de digitação ocasionais
        if np.random.random() < 0.2:
            chars = list(comentario_modificado)
            idx = np.random.randint(0, len(chars))
            if idx < len(chars) - 1:
                chars[idx], chars[idx + 1] = chars[idx + 1], chars[idx]
            comentario_modificado = "".join(chars)

        todos_comentarios.append(comentario_modificado)
        sentimentos.append(sentimento)

    return pd.DataFrame({
        'comentario': todos_comentarios,
        'sentimento': sentimentos
    })

# Pré-processamento de texto
def preprocessar_texto(texto):
    """Realiza pré-processamento básico de texto."""
    # Converter para minúsculas
    texto = texto.lower()

    # Remover caracteres especiais e números
    texto = re.sub(r'[^a-záâãäåêëîíîöôõçñ]', '', texto)

    # Tokenização
    tokens = word_tokenize(texto, language='portuguese')

    # Remover stopwords
    stop_words = set(stopwords.words('portuguese'))
    tokens = [token for token in tokens if token not in stop_words]

    # Lematização
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(token) for token in tokens]

    return tokens

# Análise de frequência de palavras
def analisar_frequencia_palavras(df, coluna_texto, n=20):
    """Análise a frequência de palavras nos textos."""
    # Combinar todos os tokens
    todos_tokens = []
    for texto in df[coluna_texto]:
        tokens = preprocessar_texto(texto)
        todos_tokens.extend(tokens)

    # Contar frequência
    contador = Counter(todos_tokens)
    palavras_comuns = contador.most_common(n)

```

```

# Converter para DataFrame
df_freq = pd.DataFrame(palavras_comuns, columns=['palavra', 'frequencia'])

# Visualizar
plt.figure(figsize=(12, 6))
sns.barplot(x='frequencia', y='palavra', data=df_freq)
plt.title(f'Top {n} Palavras Mais Frequentes')
plt.xlabel('Frequência')
plt.ylabel('Palavra')
plt.tight_layout()
plt.show()

return df_freq

# Análise de palavras por sentimento
def analisar_palavras_por_sentimento(df, coluna_texto, coluna_sentimento, n=10):
    """Analisar palavras mais frequentes por categoria de sentimento."""
    sentimentos = df[coluna_sentimento].unique()

    plt.figure(figsize=(15, 10))

    for i, sentimento in enumerate(sentimentos, 1):
        # Filtrar por sentimento
        textos = df[df[coluna_sentimento] == sentimento][coluna_texto]

        # Combinar tokens
        tokens_sentimento = []
        for texto in textos:
            tokens = preprocesar_texto(texto)
            tokens_sentimento.extend(tokens)

        # Contar frequência
        contador = Counter(tokens_sentimento)
        palavras_comuns = contador.most_common(n)

        # Converter para DataFrame
        df_freq = pd.DataFrame(palavras_comuns, columns=['palavra', 'frequencia'])

        # Plotar
        plt.subplot(len(sentimentos), 1, i)
        sns.barplot(x='frequencia', y='palavra', data=df_freq)
        plt.title(f'Top {n} Palavras - Sentimento: {sentimento.capitalize()}')
        plt.tight_layout()

    plt.show()

# Visualização da distribuição de sentimentos
def visualizar_distribuicao_sentimentos(df, coluna_sentimento):
    """Visualiza a distribuição de sentimentos nos comentários."""
    contagem = df[coluna_sentimento].value_counts()

    plt.figure(figsize=(10, 6))
    sns.barplot(x=contagem.index, y=contagem.values)
    plt.title('Distribuição de Sentimentos')
    plt.xlabel('Sentimento')
    plt.ylabel('Contagem')

    # Adicionar rótulos de valor
    for i, v in enumerate(contagem.values):
        plt.text(i, v + 0.5, str(v), ha='center')

    plt.tight_layout()
    plt.show()

# Gráfico de pizza
plt.figure(figsize=(8, 8))
plt.pie(contagem.values, labels=contagem.index, autopct='%1.1f%%',
        colors=['green', 'gray', 'red'])
plt.title('Proporção de Sentimentos')
plt.tight_layout()
plt.show()

# Executar análise de sentimento
def analisar_sentimentos():
    """Executa o pipeline de análise de sentimento."""
    print("Iniciando análise de sentimento em comentários...\n")

    # Gerar dados
    print("Gerando dados simulados...")
    df_comentarios = gerar_comentarios_simulados(n=200)
    print(f"Gerados {len(df_comentarios)} comentários para análise.\n")

    # Visualizar distribuição de sentimentos
    print("Analisando distribuição de sentimentos...")
    visualizar_distribuicao_sentimentos(df_comentarios, 'sentimento')

    # Análise de frequência geral de palavras
    print("\nAnalisando frequência geral de palavras...")
    freq_palavras = analisar_frequencia_palavras(df_comentarios, 'comentario', n=15)

    # Análise por sentimento
    print("\nAnalisando palavras por sentimento...")
    analisar_palavras_por_sentimento(df_comentarios, 'comentario', 'sentimento', n=10)

    print("\nAnálise de sentimento concluída!")

    return df_comentarios

# Executar análise
df_analise = analisar_sentimentos()

```

## Geração de Documentação

- Exemplo de Geração de Documentação:

python

 Copiar

```

import pandas as pd
import numpy as np
import os
from datetime import datetime
import matplotlib.pyplot as plt
import seaborn as sns

def gerar_documentacao_projeto(nome_projeto, autor, descricao):
    """
    Gera documentação estruturada para um projeto de análise de dados.

    Parameters:
    -----
    nome_projeto : str
        Nome do projeto
    autor : str
        Nome do autor ou equipe
    """

```

```
descricao : str
    Descrição breve do projeto

Returns:
-----
str
    Texto formatado da documentação
"""
data_atual = datetime.now().strftime("%d/%m/%Y")

# Estrutura da documentação
documentacao = f"""
# {nome_projeto}

**Autor:** {autor}
**Data:** {data_atual}

## Descrição do Projeto

{descricao}

## Estrutura do Projeto

projeto/ |—— data/ # Dados brutos e processados | |—— raw/ # Dados originais, imutáveis | |—— processed/ # Dados processados |—— notebooks/ # Jupyter notebooks | |——
01_exploracao.ipynb # Análise exploratória | |—— 02_processamento.ipynb # Processamento de dados | |—— 03_modelagem.ipynb # Modelagem e avaliação |—— src/ # Código
fonte | |—— init.py | |—— data/ # Scripts para processamento de dados | |—— features/ # Scripts para engenharia de features | |—— models/ # Scripts para treinamento e
avaliação |—— reports/ # Relatórios e visualizações | |—— figures/ # Figuras e gráficos gerados |—— requirements.txt # Dependências do projeto |—— README.md # Descrição geral
do projeto

markdown
```

 Copiar

## ## Fluxo de Trabalho

- Coleta de Dados**
  - Fonte dos dados
  - Processo de extração
  - Armazenamento inicial
- Exploração e Limpeza**
  - Análise exploratória inicial
  - Identificação de valores ausentes e outliers
  - Limpeza e transformação básica
- Engenharia de Features**
  - Criação de novas variáveis
  - Seleção de features relevantes
  - Normalização/padronização
- Modelagem**
  - Seleção de algoritmos
  - Treinamento e validação
  - Otimização de hiperparâmetros
- Avaliação**
  - Métricas de desempenho
  - Validação cruzada
  - Análise de erros
- Implantação**
  - Estratégia de implantação
  - Monitoramento
  - Manutenção

## ## Dependências

- Python 3.8+
- pandas
- numpy
- scikit-learn
- matplotlib
- seaborn

## ## Como Executar

- Clone o repositório
- Instale as dependências: `pip install -r requirements.txt`
- Execute os notebooks na ordem numérica

## ## Resultados Esperados

[Descrição dos resultados esperados do projeto]

continue

 Monica Sonnet

## Ambiente de Código e Programação Disponível (Continuação)

## 9. Manipulação de Texto e Linguagem Natural (Continuação)

### Geração de Documentação (Continuação)

- Exemplo de Geração de Documentação** (Continuação):

```
python

## Resultados Esperados

[Descrição dos resultados esperados do projeto]

## Referências

- [Referência 1]
- [Referência 2]
- [Referência 3]
"""

    return documentacao

# Exemplo de uso da função
def criar_documentacao_exemplo():
    """Cria um exemplo de documentação para um projeto de análise de dados."""
    nome_projeto = "Análise Preditiva de Churn de Clientes"
    autor = "Equipe de Ciência de Dados"
    descricao = """
    Este projeto visa desenvolver um modelo preditivo para identificar clientes com alto risco de churn (cancelamento).
    Utilizando técnicas de machine learning, analisaremos padrões de comportamento dos clientes
    para prever com antecedência possíveis cancelamentos, permitindo ações preventivas.
    """
```

 Copiar

```
"""

documentacao = gerar_documentacao_projeto(nome_projeto, autor, descricao)

# Adicionar seção de resultados específicos
resultados = """
## Resultados Específicos

### Métricas de Desempenho do Modelo

| Modelo | Acurácia | Precisão | Recall | F1-Score | AUC-ROC |
|-----|-----|-----|-----|-----|-----|
| Regressão Logística | 0.82 | 0.76 | 0.69 | 0.72 | 0.85 |
| Random Forest | 0.87 | 0.81 | 0.74 | 0.77 | 0.91 |
| XGBoost | 0.89 | 0.84 | 0.78 | 0.81 | 0.93 |
| Rede Neural | 0.86 | 0.79 | 0.76 | 0.77 | 0.90 |

### Principais Fatores de Churn

1. **Tempo de inatividade**: Clientes com mais de 30 dias sem uso do serviço
2. **Problemas de suporte**: Clientes com múltiplos tickets de suporte não resolvidos
3. **Utilização decrescente**: Padrão de queda no uso do serviço nos últimos 3 meses
4. **Feedback negativo**: Avaliações negativas recentes em pesquisas de satisfação
5. **Concorrência**: Surgimento de ofertas competitivas no mercado

### Impacto Financeiro Esperado

Com a implementação do modelo preditivo e das estratégias de retenção associadas, estimamos:

- Redução de 15-20% na taxa de churn
- Aumento de 10% na receita média por cliente
- ROI estimado de 300% no primeiro ano
"""

documentacao_completa = documentacao.replace("[Descrição dos resultados esperados do projeto]", resultados)

# Adicionar referências
referencias = """
## Referências

- Vafeiadis, T., Diamantaras, K. I., Sarigiannidis, G., & Chatzisavvas, K. C. (2015). A comparison of machine learning techniques for customer churn prediction. Simulation Modelling Practice and Theory, 55, 1-9.
- Verbeke, W., Dejaeger, K., Martens, D., Hur, J., & Baesens, B. (2012). New insights into churn prediction in the telecommunication sector: A profit driven data mining approach. European Journal of Operational Research, 218(1), 211-229.
- Hadden, J., Tiwari, A., Roy, R., & Ruta, D. (2007). Computer assisted customer churn management: State-of-the-art and future trends. Computers & Operations Research, 34(10), 2902-2917.
- Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.
"""

documentacao_completa = documentacao_completa.replace("- [Referência 1]\n- [Referência 2]\n- [Referência 3]", referencias.strip())

return documentacao_completa

# documentacao_exemplo = criar_documentacao_exemplo()
# print(documentacao_exemplo)
```

Processamento de Linguagem Natural

- Análise de Tópicos:



```
python

import pandas as pd
import numpy as np
import re
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.decomposition import LatentDirichletAllocation, NMF
import matplotlib.pyplot as plt
import seaborn as sns

# Configurações
np.random.seed(42)

# Gerar dados simulados (artigos de notícias)
def gerar_artigos_simulados(n=100):
    """Gera artigos de notícias simulados para modelagem de tópicos."""
    # Tópicos e palavras relacionadas
    topicos = {
        "tecnologia": [
            "A Apple lançou hoje seu novo iPhone com recursos avançados de inteligência artificial. O dispositivo conta com processador mais rápido e melhor duração de bateria.",
            "A Microsoft anunciou atualizações para o Windows que incluem integração profunda com ferramentas de IA generativa e assistentes virtuais.",
            "Google apresenta novo algoritmo de busca baseado em redes neurais que promete resultados mais precisos e personalizados.",
            "A Tesla revelou seu mais recente sistema de direção autônoma que utiliza visão computacional avançada e aprendizado profundo.",
            "Pesquisadores desenvolveram chip quântico capaz de processar informações 100 vezes mais rápido que processadores convencionais."
        ],
        "economia": [
            "O Banco Central decidiu manter a taxa de juros em 10,5% ao ano, contrariando expectativas do mercado que esperava um corte.",
            "A inflação acumulada nos últimos 12 meses atingiu 4,2%, ficando dentro da meta estabelecida pelo Conselho Monetário Nacional.",
            "O dólar fechou em alta de 1,2% frente ao real, refletindo incertezas sobre a política fiscal do governo.",
            "A bolsa de valores registrou queda de 0,8% hoje, puxada principalmente por ações do setor bancário e de commodities.",
            "O PIB cresceu 0,9% no último trimestre, superando as expectativas dos analistas que previam alta de 0,6%."
        ],
        "saude": [
            "Novo estudo publicado na revista Science demonstra eficácia de vacina experimental contra o câncer de pâncreas em testes iniciais.",
            "Pesquisadores identificaram gene associado ao desenvolvimento precoce de Alzheimer, abrindo caminho para novos tratamentos.",
            "A Organização Mundial da Saúde alertou sobre o aumento de casos de dengue em países tropicais devido às mudanças climáticas.",
            "Estudo com 10 mil participantes confirma benefícios da dieta mediterrânea na prevenção de doenças cardiovasculares.",
            "Nova técnica cirúrgica minimamente invasiva reduz em 50% o tempo de recuperação de pacientes com problemas na coluna."
        ],
        "esportes": [
            "O Brasil venceu a Argentina por 3 a 1 em partida decisiva pelas eliminatórias da Copa do Mundo, com gols de Neymar e Vinicius Jr.",
            "Lewis Hamilton conquistou sua 100ª vitória na Fórmula 1 no Grande Prêmio da Itália, ampliando seu recorde na categoria.",
            "O time de basquete dos Estados Unidos garantiu medalha de ouro nas Olimpíadas ao derrotar a França na final por 87 a 82.",
            "Novak Djokovic venceu Rafael Nadal em uma épica final de cinco sets em Wimbledon, conquistando seu 25º título de Grand Slam.",
            "A seleção brasileira feminina de vôlei sagrou-se campeã mundial após vitória sobre a China por 3 sets a 0 na final."
        ],
        "politica": [
            "O presidente anunciou hoje um pacote de medidas econômicas que inclui redução de impostos para pequenas empresas e incentivos à exportação.",
            "O Congresso aprovou projeto de lei que altera regras de aposentadoria para servidores públicos, gerando protestos nas principais capitais.",
            "Ministro da Justiça pediu demissão após divergências com o presidente sobre políticas de segurança pública.",
            "Pesquisa de opinião mostra queda de 5 pontos na aprovação do governo, atingindo o menor índice desde o início do mandato.",
            "Oposição protocolou pedido de impeachment contra o presidente, alegando crime de responsabilidade na gestão da pandemia."
        ]
    }

    # Gerar artigos completos
    artigos = []
    categorias = []

    for _ in range(n):
```

```

# Escolher tópico principal
topico = np.random.choice(list(topicos.keys()))
categorias.append(topico)

# Escolher artigo base do tópico principal
artigo_base = np.random.choice(topicos[topico])

# Adicionar conteúdo de outro tópico em 30% dos casos (mistura de tópicos)
if np.random.random() < 0.3:
    topico_secundario = np.random.choice([t for t in topicos.keys() if t != topico])
    artigo_secundario = np.random.choice(topicos[topico_secundario])

    # Extrair algumas frases do artigo secundário
    frases_secundarias = artigo_secundario.split('.')
    frases_selecionadas = np.random.choice(frases_secundarias, size=min(2, len(frases_secundarias)), replace=False)

    # Combinar com artigo principal
    artigo_final = artigo_base + " " + " ".join(frases_selecionadas) + "."
else:
    artigo_final = artigo_base

# Adicionar variação
artigos.append(artigo_final)

return pd.DataFrame({
    'artigo': artigos,
    'categoria': categorias
})

# Pré-processamento de texto
def preprocessar_texto(texto):
    """Realiza pré-processamento de texto para modelagem de tópicos."""
    # Converter para minúsculas
    texto = texto.lower()

    # Remover caracteres especiais e números
    texto = re.sub(r'[^a-záâãäåêëéíîóôõöçñ]', '', texto)

    # Tokenização
    tokens = word_tokenize(texto, language='portuguese')

    # Remover stopwords
    stop_words = set(stopwords.words('portuguese'))
    tokens = [token for token in tokens if token not in stop_words and len(token) > 2]

    # Lematização
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(token) for token in tokens]

    return ' '.join(tokens)

# Vetorização de texto
def vetorizar_textos(textos_processados, metodo='tfidf', max_features=1000):
    """Vetoriza textos usando CountVectorizer ou TfidfVectorizer."""
    if metodo == 'count':
        vectorizer = CountVectorizer(max_features=max_features)
    else: # tfidf
        vectorizer = TfidfVectorizer(max_features=max_features)

    X = vectorizer.fit_transform(textos_processados)

    return X, vectorizer

# Modelagem de tópicos com LDA
def modelar_topicos_lda(X, vectorizer, n_topicos=5, n_palavras=10):
    """Aplica modelagem de tópicos usando LDA."""
    # Treinar modelo LDA
    lda = LatentDirichletAllocation(
        n_components=n_topicos,
        random_state=42,
        max_iter=10
    )

    lda.fit(X)

    # Extrair palavras mais importantes por tópico
    feature_names = vectorizer.get_feature_names_out()

    topicos = []
    for i, topic in enumerate(lda.components_):
        top_words_idx = topic.argsort()[::-n_palavras:-1]
        top_words = [feature_names[idx] for idx in top_words_idx]
        topicos.append({
            'id': i,
            'palavras': top_words
        })

    return lda, topicos

# Modelagem de tópicos com NMF
def modelar_topicos_nmf(X, vectorizer, n_topicos=5, n_palavras=10):
    """Aplica modelagem de tópicos usando NMF."""
    # Treinar modelo NMF
    nmf = NMF(
        n_components=n_topicos,
        random_state=42,
        max_iter=200
    )

    nmf.fit(X)

    # Extrair palavras mais importantes por tópico
    feature_names = vectorizer.get_feature_names_out()

    topicos = []
    for i, topic in enumerate(nmf.components_):
        top_words_idx = topic.argsort()[::-n_palavras:-1]
        top_words = [feature_names[idx] for idx in top_words_idx]
        topicos.append({
            'id': i,
            'palavras': top_words
        })

    return nmf, topicos

# Visualizar tópicos
def visualizar_topicos(topicos, titulo="Palavras-chave por Tópico"):
    """Visualiza as palavras-chave de cada tópico."""
    n_topicos = len(topicos)
    n_palavras = len(topicos[0]['palavras'])

    fig, axes = plt.subplots(n_topicos, 1, figsize=(10, n_topicos * 2.5), sharex=True)

    for i, topico in enumerate(topicos):
        ax = axes[i] if n_topicos > 1 else axes
        palavras = topico['palavras']

```

```

        pesos = np.linspace(1, 0.4, len(palavras)) # Pesos decrescentes para visualização

        ax.barh(range(n_palavras), pesos, align='center')
        ax.set_yticks(range(n_palavras))
        ax.set_yticklabels(palavras)
        ax.set_title(f'Tópico {i+1}')
        ax.invert_yaxis()

    plt.tight_layout()
    plt.suptitle(titulo, y=1.02, fontsize=16)
    plt.show()

# Classificar documentos por tópico
def classificar_documentos_por_topico(modelo, X):
    """Classifica cada documento pelo tópico mais provável."""
    # Obter distribuição de tópicos para cada documento
    doc_topico = modelo.transform(X)

    # Atribuir tópico mais provável
    topico_principal = doc_topico.argmax(axis=1)

    return topico_principal, doc_topico

# Visualizar distribuição de tópicos
def visualizar_distribuicao_topicos(topico_principal, n_topicos):
    """Visualiza a distribuição de documentos por tópico."""
    contagem = pd.Series(topico_principal).value_counts().sort_index()

    plt.figure(figsize=(10, 6))
    sns.barplot(x=contagem.index, y=contagem.values)
    plt.title('Distribuição de Documentos por Tópico')
    plt.xlabel('Tópico')
    plt.ylabel('Número de Documentos')
    plt.xticks(range(n_topicos), [f'Tópico {i+1}' for i in range(n_topicos)])

    # Adicionar rótulos de valor
    for i, v in enumerate(contagem.values):
        plt.text(i, v + 0.5, str(v), ha='center')

    plt.tight_layout()
    plt.show()

# Comparar tópicos com categorias reais
def comparar_topicos_categorias(df, topico_principal):
    """Compara os tópicos descobertos com as categorias reais."""
    df_comparacao = df.copy()
    df_comparacao['topico'] = topico_principal

    # Tabela de contingência
    tabela = pd.crosstab(df_comparacao['categoria'], df_comparacao['topico'])

    # Normalizar por linha
    tabela_norm = tabela.div(tabela.sum(axis=1), axis=0)

    # Visualizar mapa de calor
    plt.figure(figsize=(10, 8))
    sns.heatmap(tabela_norm, annot=tabela.values, fmt='d', cmap='YlGnBu')
    plt.title('Correspondência entre Categorias e Tópicos')
    plt.xlabel('Tópico Descoberto')
    plt.ylabel('Categoria Real')
    plt.xticks(np.arange(tabela.shape[1]) + 0.5, [f'Tópico {i+1}' for i in range(tabela.shape[1])])
    plt.tight_layout()
    plt.show()

    return tabela

# Pipeline completo
def analisar_topicos():
    """Executa o pipeline completo de modelagem de tópicos."""
    print("Iniciando análise de tópicos em artigos de notícias...\n")

    # Gerar dados
    print("Gerando dados simulados...")
    df_artigos = gerar_artigos_simulados(n=200)
    print(f"Gerados {len(df_artigos)} artigos para análise.\n")

    # Pré-processamento
    print("Realizando pré-processamento de texto...")
    df_artigos['texto_processado'] = df_artigos['artigo'].apply(preprocessar_texto)

    # Vetorização
    print("Vetorizando textos...")
    X, vectorizer = vetorizar_textos(df_artigos['texto_processado'], metodo='tfidf', max_features=1000)

    # Número de tópicos (igual ao número de categorias reais para comparação)
    n_topicos = len(df_artigos['categoria'].unique())
    print(f"Número de tópicos a serem descobertos: {n_topicos}\n")

    # Modelagem com LDA
    print("Aplicando modelagem de tópicos com LDA...")
    modelo_lda, topicos_lda = modelar_topicos_lda(X, vectorizer, n_topicos=n_topicos)

    # Visualizar tópicos LDA
    print("\nTópicos descobertos pelo LDA:")
    visualizar_topicos(topicos_lda, titulo="Palavras-chave por Tópico (LDA)")

    # Classificar documentos
    print("\nClassificando documentos por tópico...")
    topico_principal_lda, _ = classificar_documentos_por_topico(modelo_lda, X)

    # Visualizar distribuição
    print("\nDistribuição de documentos por tópico:")
    visualizar_distribuicao_topicos(topico_principal_lda, n_topicos)

    # Comparar com categorias reais
    print("\nComparando tópicos descobertos com categorias reais:")
    tabela_comparacao = comparar_topicos_categorias(df_artigos, topico_principal_lda)

    # Modelagem com NMF (opcional)
    print("\nAplicando modelagem de tópicos com NMF...")
    modelo_nmf, topicos_nmf = modelar_topicos_nmf(X, vectorizer, n_topicos=n_topicos)

    # Visualizar tópicos NMF
    print("\nTópicos descobertos pelo NMF:")
    visualizar_topicos(topicos_nmf, titulo="Palavras-chave por Tópico (NMF)")

    print("\nAnálise de tópicos concluída!")

    return {
        'df_artigos': df_artigos,
        'modelo_lda': modelo_lda,
        'topicos_lda': topicos_lda,
        'modelo_nmf': modelo_nmf,
        'topicos_nmf': topicos_nmf,
        'tabela_comparacao': tabela_comparacao
    }
}

```



```
# resultados_topicos = analisar_topicos()
```

## 10. Integração com Ferramentas Externas

### Manipulação de Arquivos

- Operações com Arquivos:

```
python  Copiar
```

```
import os
import json
import csv
import shutil
import glob
import pandas as pd
from datetime import datetime

# Configurações
diretorio_base = '/home/user'

# Criar estrutura de diretórios
def criar_estrutura_diretorios():
    """Cria uma estrutura de diretórios para o projeto."""
    diretorios = [
        'dados',
        'dados/brutos',
        'dados/processados',
        'resultados',
        'resultados/graficos',
        'resultados/relatorios',
        'logs'
    ]

    for diretorio in diretorios:
        caminho = os.path.join(diretorio_base, diretorio)
        os.makedirs(caminho, exist_ok=True)
        print(f"Diretório criado: {caminho}")

# Salvar dados em diferentes formatos
def salvar_dados_exemplo():
    """Salva dados de exemplo em diferentes formatos."""
    # Gerar dados de exemplo
    dados = {
        'nome': ['Ana', 'Bruno', 'Carlos', 'Diana', 'Eduardo'],
        'idade': [25, 32, 45, 28, 36],
        'departamento': ['TI', 'Marketing', 'Financeiro', 'RH', 'TI'],
        'salario': [5000, 6000, 8000, 4500, 5500]
    }

    df = pd.DataFrame(dados)

    # Salvar como CSV
    caminho_csv = os.path.join(diretorio_base, 'dados/processados/funcionarios.csv')
    df.to_csv(caminho_csv, index=False)
    print(f"Dados salvos em CSV: {caminho_csv}")

    # Salvar como JSON
    caminho_json = os.path.join(diretorio_base, 'dados/processados/funcionarios.json')
    df.to_json(caminho_json, orient='records', indent=4)
    print(f"Dados salvos em JSON: {caminho_json}")

    # Salvar como Excel
    caminho_excel = os.path.join(diretorio_base, 'dados/processados/funcionarios.xlsx')
    df.to_excel(caminho_excel, index=False, sheet_name='Funcionários')
    print(f"Dados salvos em Excel: {caminho_excel}")

    # Salvar como HTML
    caminho_html = os.path.join(diretorio_base, 'dados/processados/funcionarios.html')
    df.to_html(caminho_html, index=False)
    print(f"Dados salvos em HTML: {caminho_html}")

    # Salvar como arquivo de texto
    caminho_txt = os.path.join(diretorio_base, 'dados/processados/funcionarios.txt')
    with open(caminho_txt, 'w') as f:
        for indice, linha in df.iterrows():
            f.write(f"{linha['nome']}, {linha['idade']}, {linha['departamento']}, {linha['salario']}\n")
    print(f"Dados salvos em TXT: {caminho_txt}")

    return df

# Ler dados de diferentes formatos
def ler_dados_exemplo():
    """Lê dados de exemplo em diferentes formatos."""
    resultados = {}

    # Ler CSV
    caminho_csv = os.path.join(diretorio_base, 'dados/processados/funcionarios.csv')
    if os.path.exists(caminho_csv):
        df_csv = pd.read_csv(caminho_csv)
        resultados['csv'] = df_csv
        print(f"Dados lidos de CSV: {len(df_csv)} registros")

    # Ler JSON
    caminho_json = os.path.join(diretorio_base, 'dados/processados/funcionarios.json')
    if os.path.exists(caminho_json):
        df_json = pd.read_json(caminho_json)
        resultados['json'] = df_json
        print(f"Dados lidos de JSON: {len(df_json)} registros")

    # Ler Excel
    caminho_excel = os.path.join(diretorio_base, 'dados/processados/funcionarios.xlsx')
    if os.path.exists(caminho_excel):
        df_excel = pd.read_excel(caminho_excel)
        resultados['excel'] = df_excel
        print(f"Dados lidos de Excel: {len(df_excel)} registros")

    # Ler HTML
    caminho_html = os.path.join(diretorio_base, 'dados/processados/funcionarios.html')
    if os.path.exists(caminho_html):
        df_html = pd.read_html(caminho_html)[0]
        resultados['html'] = df_html
        print(f"Dados lidos de HTML: {len(df_html)} registros")

    # Ler TXT
    caminho_txt = os.path.join(diretorio_base, 'dados/processados/funcionarios.txt')
    if os.path.exists(caminho_txt):
        registros = []
        with open(caminho_txt, 'r') as f:
            for linha in f:
                campos = linha.strip().split(',')
                if len(campos) == 4:
                    registros.append({
                        'nome': campos[0],
```

```

        'idade': int(campos[1]),
        'departamento': campos[2],
        'salario': float(campos[3])
    })
    df_txt = pd.DataFrame(registros)
    resultados['txt'] = df_txt
    print(f"Dados lidos de TXT: {len(df_txt)} registros")

return resultados

# Listar arquivos em um diretório
def listar_arquivos(diretorio=None, padrao='*'):
    """Lista arquivos em um diretório com um padrão específico."""
    if diretorio is None:
        diretorio = diretorio_base

    caminho_padrao = os.path.join(diretorio, padrao)
    arquivos = glob.glob(caminho_padrao)

    for arquivo in arquivos:
        tamanho = os.path.getsize(arquivo) / 1024 # KB
        data_mod = datetime.fromtimestamp(os.path.getmtime(arquivo))
        print(f"{os.path.basename(arquivo):<30} {tamanho:.2f} KB {data_mod.strftime('%Y-%m-%d %H:%M:%S')}")

    return arquivos

# Operações de arquivo
def demonstrar_operacoes_arquivo():
    """Demonstra operações básicas com arquivos."""
    # Criar arquivo de texto
    caminho_log = os.path.join(diretorio_base, 'logs/emplo.log')

    with open(caminho_log, 'w') as f:
        f.write(f"Log iniciado em {datetime.now()}\n")
        f.write("Operação 1: Criação de diretórios\n")
        f.write("Operação 2: Salvamento de dados\n")
        f.write("Operação 3: Leitura de dados\n")

    print(f"Arquivo de log criado: {caminho_log}")

    # Ler arquivo de texto
    print("\nConteúdo do arquivo de log:")
    with open(caminho_log, 'r') as f:
        conteudo = f.read()
        print(conteudo)

    # Adicionar ao arquivo
    with open(caminho_log, 'a') as f:
        f.write(f"Log atualizado em {datetime.now()}\n")
        f.write("Operação 4: Atualização de log\n")

    print("\nConteúdo do arquivo de log após atualização:")
    with open(caminho_log, 'r') as f:
        conteudo = f.read()
        print(conteudo)

    # Copiar arquivo
    caminho_backup = os.path.join(diretorio_base, 'logs/emplo_backup.log')
    shutil.copy2(caminho_log, caminho_backup)
    print(f"Arquivo copiado para: {caminho_backup}")

    # Renomear arquivo
    caminho_renomeado = os.path.join(diretorio_base, 'logs/emplo_renomeado.log')
    os.rename(caminho_backup, caminho_renomeado)

```

continue

 Monica Sonnet

## Ambiente de Código e Programação Disponível (Continuação)

### 10. Integração com Ferramentas Externas (Continuação)

#### Manipulação de Arquivos (Continuação)

- Operações com Arquivos (Continuação):

python

 Copiar

```

# Renomear arquivo
caminho_renomeado = os.path.join(diretorio_base, 'logs/emplo_renomeado.log')
os.rename(caminho_backup, caminho_renomeado)
print(f"Arquivo renomeado para: {caminho_renomeado}")

# Verificar existência
existe = os.path.exists(caminho_renomeado)
print(f"Arquivo renomeado existe? {existe}")

# Obter informações do arquivo
tamanho = os.path.getsize(caminho_renomeado) / 1024 # KB
data_criacao = datetime.fromtimestamp(os.path.getctime(caminho_renomeado))
data_modificacao = datetime.fromtimestamp(os.path.getmtime(caminho_renomeado))

print(f"Informações do arquivo:")
print(f"Tamanho: {tamanho:.2f} KB")
print(f>Data de criação: {data_criacao}")
print(f>Data de modificação: {data_modificacao}")

# Excluir arquivo
os.remove(caminho_renomeado)
print(f"Arquivo excluído: {caminho_renomeado}")
existe = os.path.exists(caminho_renomeado)
print(f"Arquivo ainda existe? {existe}")

# Executar demonstração
def executar_demonstracao_arquivos():
    """Executa demonstração completa de manipulação de arquivos."""
    print("Iniciando demonstração de manipulação de arquivos...\n")

    # Criar estrutura de diretórios
    print("Criando estrutura de diretórios...")
    criar_estrutura_diretorios()

    # Salvar dados em diferentes formatos
    print("\nSalvando dados em diferentes formatos...")
    df = salvar_dados_exemplo()

    # Listar arquivos
    print("\nListando arquivos no diretório de dados processados:")
    listar_arquivos(os.path.join(diretorio_base, 'dados/processados'))

    # Ler dados
    print("\nLendo dados de diferentes formatos...")

```

```

resultados = ler_dados_exemplo()

# Demonstrar operações de arquivo
print("\nDemonstrando operações básicas com arquivos...")
demonstrar_operacoes_arquivo()

print("\nDemonstração de manipulação de arquivos concluída!")

return {
    'dataframe': df,
    'resultados_leitura': resultados
}

# resultados_arquivos = executar_demonstracao_arquivos()

```

## Exportação de Dados

- Exportação para Formatos Diversos:

python

 Copiar

```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
import json
from datetime import datetime

# Configurações
diretorio_base = '/home/user'
diretorio_resultados = os.path.join(diretorio_base, 'resultados')
os.makedirs(diretorio_resultados, exist_ok=True)

# Gerar dados de exemplo
def gerar_dados_vendas():
    """Gera dados de vendas para exportação."""
    # Dados de vendas por região e categoria
    regioes = ['Norte', 'Sul', 'Leste', 'Oeste', 'Centro']
    categorias = ['Eletrônicos', 'Roupas', 'Alimentos', 'Móveis', 'Livros']

    np.random.seed(42)

    # Criar DataFrame
    dados = []
    for ano in range(2020, 2026):
        for mes in range(1, 13):
            for regioao in regioes:
                for categoria in categorias:
                    # Adicionar sazonalidade e tendência
                    base_vendas = 1000 + (ano - 2020) * 100 # Tendência de crescimento

                    # Sazonalidade por mês
                    if mes in [11, 12]: # Fim de ano
                        fator_mes = 1.5
                    elif mes in [1, 2]: # Início de ano
                        fator_mes = 0.7
                    else:
                        fator_mes = 1.0

                    # Variação por região
                    fator_regiao = {
                        'Norte': 0.8,
                        'Sul': 1.2,
                        'Leste': 1.1,
                        'Oeste': 0.9,
                        'Centro': 1.0
                    }[regiao]

                    # Variação por categoria
                    fator_categoria = {
                        'Eletrônicos': 1.3,
                        'Roupas': 1.1,
                        'Alimentos': 0.9,
                        'Móveis': 0.7,
                        'Livros': 0.5
                    }[categoria]

                    # Calcular vendas
                    vendas = base_vendas * fator_mes * fator_regiao * fator_categoria

                    # Adicionar ruído
                    vendas *= np.random.normal(1, 0.1)

                    # Adicionar ao DataFrame
                    dados.append({
                        'ano': ano,
                        'mes': mes,
                        'regiao': regioao,
                        'categoria': categoria,
                        'vendas': round(vendas, 2),
                        'unidades': int(vendas / 50) # Aproximadamente 50 por unidade
                    })

    df = pd.DataFrame(dados)

    # Adicionar data completa
    df['data'] = pd.to_datetime(df[['ano', 'mes']].assign(day=1))

    return df

# Exportar para CSV
def exportar_csv(df, nome_arquivo='vendas.csv'):
    """Exporta DataFrame para CSV."""
    caminho = os.path.join(diretorio_resultados, nome_arquivo)
    df.to_csv(caminho, index=False)
    print(f"Dados exportados para CSV: {caminho}")
    return caminho

# Exportar para Excel
def exportar_excel(df, nome_arquivo='vendas.xlsx'):
    """Exporta DataFrame para Excel com formatação."""
    caminho = os.path.join(diretorio_resultados, nome_arquivo)

    # Criar writer
    writer = pd.ExcelWriter(caminho, engine='xlsxwriter')

    # Exportar dados principais
    df.to_excel(writer, sheet_name='Dados', index=False)

    # Criar pivot table por região
    pivot_regiao = df.pivot_table(
        values='vendas',
        index=['ano', 'mes'],
        columns='regiao',

```

```

        aggfunc='sum'
    ).reset_index()

    pivot_regiao.to_excel(writer, sheet_name='Por Região', index=False)

    # Criar pivot table por categoria
    pivot_categoria = df.pivot_table(
        values='vendas',
        index=['ano', 'mes'],
        columns='categoria',
        aggfunc='sum'
    ).reset_index()

    pivot_categoria.to_excel(writer, sheet_name='Por Categoria', index=False)

    # Resumo estatístico
    df.describe().to_excel(writer, sheet_name='Estatísticas')

    # Salvar
    writer.close()

    print(f"Dados exportados para Excel: {caminho}")
    return caminho

# Exportar para JSON
def exportar_json(df, nome_arquivo='vendas.json'):
    """Exporta DataFrame para JSON."""
    caminho = os.path.join(diretorio_resultados, nome_arquivo)

    # Converter para formato de registros
    registros = df.to_dict(orient='records')

    # Adicionar metadados
    dados_json = {
        'metadados': {
            'titulo': 'Dados de Vendas',
            'descricao': 'Dados de vendas por região, categoria, ano e mês',
            'data_geracao': datetime.now().isoformat(),
            'num_registros': len(df),
            'colunas': list(df.columns)
        },
        'dados': registros
    }

    # Salvar como JSON
    with open(caminho, 'w', encoding='utf-8') as f:
        json.dump(dados_json, f, ensure_ascii=False, indent=4)

    print(f"Dados exportados para JSON: {caminho}")
    return caminho

# Exportar gráficos
def exportar_graficos(df):
    """Exporta gráficos baseados nos dados."""
    diretorio_graficos = os.path.join(diretorio_resultados, 'graficos')
    os.makedirs(diretorio_graficos, exist_ok=True)

    # Configurar estilo
    plt.style.use('seaborn-v0_8-whitegrid')

    # Gráfico 1: Vendas por Região
    plt.figure(figsize=(12, 6))
    vendas_regiao = df.groupby('regiao')['vendas'].sum().sort_values(ascending=False)
    sns.barplot(x=vendas_regiao.index, y=vendas_regiao.values)
    plt.title('Vendas Totais por Região', fontsize=14)
    plt.xlabel('Região')
    plt.ylabel('Vendas (R$)')
    plt.tight_layout()

    caminho_grafico1 = os.path.join(diretorio_graficos, 'vendas_por_regiao.png')
    plt.savefig(caminho_grafico1, dpi=300)
    plt.close()

    # Gráfico 2: Vendas por Categoria
    plt.figure(figsize=(12, 6))
    vendas_categoria = df.groupby('categoria')['vendas'].sum().sort_values(ascending=False)
    sns.barplot(x=vendas_categoria.index, y=vendas_categoria.values)
    plt.title('Vendas Totais por Categoria', fontsize=14)
    plt.xlabel('Categoria')
    plt.ylabel('Vendas (R$)')
    plt.xticks(rotation=45)
    plt.tight_layout()

    caminho_grafico2 = os.path.join(diretorio_graficos, 'vendas_por_categoria.png')
    plt.savefig(caminho_grafico2, dpi=300)
    plt.close()

    # Gráfico 3: Tendência de Vendas ao Longo do Tempo
    plt.figure(figsize=(14, 7))
    vendas_tempo = df.groupby('data')['vendas'].sum()
    plt.plot(vendas_tempo.index, vendas_tempo.values, marker='o', linestyle='-', alpha=0.7)
    plt.title('Tendência de Vendas ao Longo do Tempo', fontsize=14)
    plt.xlabel('Data')
    plt.ylabel('Vendas (R$)')
    plt.grid(True, alpha=0.3)
    plt.tight_layout()

    caminho_grafico3 = os.path.join(diretorio_graficos, 'tendencia_vendas.png')
    plt.savefig(caminho_grafico3, dpi=300)
    plt.close()

    # Gráfico 4: Heatmap de Vendas por Mês e Ano
    plt.figure(figsize=(12, 8))
    vendas_mes_ano = df.pivot_table(values='vendas', index='mes', columns='ano', aggfunc='sum')
    sns.heatmap(vendas_mes_ano, annot=True, fmt='.0f', cmap='YlGnBu')
    plt.title('Vendas por Mês e Ano', fontsize=14)
    plt.xlabel('Ano')
    plt.ylabel('Mês')
    plt.tight_layout()

    caminho_grafico4 = os.path.join(diretorio_graficos, 'heatmap_vendas.png')
    plt.savefig(caminho_grafico4, dpi=300)
    plt.close()

    # Gráfico 5: Boxplot de Vendas por Categoria
    plt.figure(figsize=(14, 8))
    sns.boxplot(x='categoria', y='vendas', data=df)
    plt.title('Distribuição de Vendas por Categoria', fontsize=14)
    plt.xlabel('Categoria')
    plt.ylabel('Vendas (R$)')
    plt.xticks(rotation=45)
    plt.tight_layout()

    caminho_grafico5 = os.path.join(diretorio_graficos, 'boxplot_categorias.png')
    plt.savefig(caminho_grafico5, dpi=300)
    plt.close()

```

```

print(f"Gráficos exportados para: {diretorio_graficos}")

return [
    caminho_grafico1,
    caminho_grafico2,
    caminho_grafico3,
    caminho_grafico4,
    caminho_grafico5
]

# Exportar relatório HTML
def exportar_html(df, caminhos_graficos, nome_arquivo='relatorio_vendas.html'):
    """Exporta relatório em formato HTML."""
    caminho = os.path.join(diretorio_resultados, nome_arquivo)

    # Calcular estatísticas
    total_vendas = df['vendas'].sum()
    total_unidades = df['unidades'].sum()
    media_vendas_mes = df.groupby(['ano', 'mes'])['vendas'].sum().mean()

    top_regioes = df.groupby('regiao')['vendas'].sum().sort_values(ascending=False).head(3)
    top_categorias = df.groupby('categoria')['vendas'].sum().sort_values(ascending=False).head(3)

    # Gerar tabelas HTML
    tabela_vendas_regiao = df.pivot_table(
        values='vendas',
        index='regiao',
        columns='ano',
        aggfunc='sum'
    ).to_html(classes='table table-striped')

    tabela_vendas_categoria = df.pivot_table(
        values='vendas',
        index='categoria',
        columns='ano',
        aggfunc='sum'
    ).to_html(classes='table table-striped')

    # Construir HTML
    html_content = f"""
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Relatório de Vendas</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet">
    <style>
        body {{ font-family: Arial, sans-serif; margin: 20px; }}
        .header {{ background-color: #f8f9fa; padding: 20px; margin-bottom: 30px; border-bottom: 1px solid #ddd; }}
        .section {{ margin-bottom: 30px; }}
        .stat-card {{ background-color: #f8f9fa; padding: 20px; border-radius: 5px; margin-bottom: 20px; }}
        .chart-container {{ margin-bottom: 30px; }}
        .table-container {{ margin-bottom: 30px; overflow-x: auto; }}
    </style>
</head>
<body>
    <div class="container">
        <div class="header">
            <h1>Relatório de Vendas</h1>
            <p class="text-muted">Gerado em {datetime.now().strftime('%d/%m/%Y %H:%M:%S')}</p>
        </div>

        <div class="section">
            <h2>Resumo Executivo</h2>
            <div class="row">
                <div class="col-md-4">
                    <div class="stat-card">
                        <h3>R$ {total_vendas:,.2f}</h3>
                        <p>Total de Vendas</p>
                    </div>
                </div>
                <div class="col-md-4">
                    <div class="stat-card">
                        <h3>{total_unidades:,}</h3>
                        <p>Total de Unidades Vendidas</p>
                    </div>
                </div>
                <div class="col-md-4">
                    <div class="stat-card">
                        <h3>R$ {media_vendas_mes:,.2f}</h3>
                        <p>Média de Vendas Mensais</p>
                    </div>
                </div>
            </div>
        </div>

        <div class="section">
            <h2>Desempenho por Região</h2>
            <div class="chart-container">
                
            </div>
            <div class="table-container">
                <h4>Vendas por Região e Ano</h4>
                {tabela_vendas_regiao}
            </div>
        </div>

        <div class="section">
            <h2>Desempenho por Categoria</h2>
            <div class="chart-container">
                
            </div>
            <div class="table-container">
                <h4>Vendas por Categoria e Ano</h4>
                {tabela_vendas_categoria}
            </div>
        </div>

        <div class="section">
            <h2>Análise Temporal</h2>
            <div class="chart-container">
                
            </div>
            <div class="chart-container">
                
            </div>
        </div>

        <div class="section">
            <h2>Distribuição de Vendas</h2>
            <div class="chart-container">
                
            </div>
        </div>
    </div>
    """

```

```

        <div class="section">
            <h2>Top Performers</h2>
            <div class="row">
                <div class="col-md-6">
                    <h4>Top 3 Regiões</h4>
                    <ul class="list-group">

=====

# Adicionar top regiões
for regioao, valor in top_regioes.items():
    html_content += f'<li class="list-group-item d-flex justify-content-between align-items-center">{regiao} <span class="badge bg-primary rounded-pill">R$
{valor:,.2f}</span></li>\n'

    html_content += """
        </ul>
    </div>
    <div class="col-md-6">
        <h4>Top 3 Categorias</h4>
        <ul class="list-group">

=====

# Adicionar top categorias
for categoria, valor in top_categorias.items():
    html_content += f'<li class="list-group-item d-flex justify-content-between align-items-center">{categoria} <span class="badge bg-primary rounded-pill">R$
{valor:,.2f}</span></li>\n'

    html_content += """
        </ul>
    </div>
</div>
</div>

<footer class="mt-5 pt-3 text-muted border-top">
    &copy; 2025 Relatório de Vendas
</footer>
</div>
</body>
</html>
"""

# Salvar HTML
with open(caminho, 'w', encoding='utf-8') as f:
    f.write(html_content)

print(f"Relatório HTML exportado para: {caminho}")
return caminho

# Pipeline de exportação completo
def executar_pipeline_exportacao():
    """Executa pipeline completo de exportação de dados."""
    print("Iniciando pipeline de exportação de dados...\n")

    # Gerar dados
    print("Gerando dados de vendas...")
    df_vendas = gerar_dados_vendas()
    print(f"Gerados {len(df_vendas)} registros de vendas.\n")

    # Exportar para CSV
    print("Exportando para CSV...")
    caminho_csv = exportar_csv(df_vendas)

    # Exportar para Excel
    print("\nExportando para Excel...")
    caminho_excel = exportar_excel(df_vendas)

    # Exportar para JSON
    print("\nExportando para JSON...")
    caminho_json = exportar_json(df_vendas)

    # Exportar gráficos
    print("\nGerando e exportando gráficos...")
    caminhos_graficos = exportar_graficos(df_vendas)

    # Exportar relatório HTML
    print("\nGerando relatório HTML...")
    caminho_html = exportar_html(df_vendas, caminhos_graficos)

    print("\nPipeline de exportação concluído com sucesso!")

    return {
        'dataframe': df_vendas,
        'caminhos': {
            'csv': caminho_csv,
            'excel': caminho_excel,
            'json': caminho_json,
            'graficos': caminhos_graficos,
            'html': caminho_html
        }
    }

# resultados_exportacao = executar_pipeline_exportacao()

```

## 11. Considerações Finais

### Boas Práticas de Código

- **Estilo de Código:**
  - Seguir PEP 8 para Python
  - Indentação consistente
  - Nomes descritivos para variáveis e funções
  - Comentários úteis e docstrings
  - Modularização adequada
- **Organização de Projetos:**
  - Estrutura de diretórios lógica
  - Separação de código em módulos
  - Uso de ambientes virtuais
  - Gerenciamento de dependências
  - Documentação clara
- **Otimização:**
  - Evitar operações redundantes
  - Usar estruturas de dados apropriadas

- Vetorização em vez de loops quando possível
- Gerenciar uso de memória
- Perfis de desempenho quando necessário

## Recursos de Aprendizado

- **Documentação Oficial:**
  - Python: [python.org/doc](https://python.org/doc)
  - Pandas: [pandas.pydata.org/docs](https://pandas.pydata.org/docs)
  - NumPy: [numpy.org/doc](https://numpy.org/doc)
  - Matplotlib: [matplotlib.org/stable/contents.html](https://matplotlib.org/stable/contents.html)
  - Scikit-learn: [scikit-learn.org/stable/documentation.html](https://scikit-learn.org/stable/documentation.html)
- **Tutoriais e Cursos:**
  - Real Python: [realpython.com](https://realpython.com)
  - DataCamp: [datacamp.com](https://datacamp.com)
  - Kaggle: [kaggle.com/learn](https://kaggle.com/learn)
  - Coursera: [coursera.org](https://coursera.org)
  - edX: [edx.org](https://edx.org)
- **Comunidades:**
  - Stack Overflow: [stackoverflow.com](https://stackoverflow.com)
  - GitHub: [github.com](https://github.com)
  - Reddit (r/Python, r/datascience): [reddit.com](https://reddit.com)
  - PyData: [pydata.org](https://pydata.org)
  - Kaggle Forums: [kaggle.com/discussion](https://kaggle.com/discussion)

## Limitações e Alternativas

- **Limitações do Ambiente:**
  - Tempo de execução limitado (120 segundos)
  - Sem acesso à internet durante execução
  - Bibliotecas pré-instaladas apenas
  - Memória e processamento limitados
  - Sem persistência entre sessões
- **Alternativas para Casos Complexos:**
  - Ambientes de desenvolvimento locais
  - Serviços de notebooks na nuvem (Google Colab, Kaggle Notebooks)
  - Plataformas de ciência de dados (Databricks, AWS SageMaker)
  - Ambientes de contêiner (Docker)
  - Serviços de computação em nuvem (AWS, GCP, Azure)

## Conclusão

Este guia abrangente apresentou as capacidades e ferramentas disponíveis no ambiente de programação, com foco em Python e suas bibliotecas para análise de dados, visualização, machine learning e desenvolvimento web. As seções cobrem desde conceitos básicos até técnicas avançadas, com exemplos práticos e código funcional.

O ambiente oferece um conjunto robusto de ferramentas para resolver uma ampla variedade de problemas, desde análise exploratória de dados até modelagem preditiva e criação de relatórios interativos. Embora existam limitações inerentes ao ambiente, as capacidades disponíveis são suficientes para a maioria das tarefas de análise de dados e prototipagem.

Para aproveitar ao máximo o ambiente, recomenda-se:

1. **Planejar o fluxo de trabalho** antes de começar a codificar
2. **Modularizar o código** para facilitar a manutenção e reutilização
3. **Otimizar operações** para trabalhar dentro dos limites de tempo e memória
4. **Documentar adequadamente** o código e os resultados
5. **Explorar as capacidades de visualização** para comunicar resultados efetivamente

Com as ferramentas e técnicas apresentadas neste guia, você está equipado para abordar uma ampla gama de problemas de dados e desenvolvimento no ambiente de programação