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) DOM Artfact CSV svg Animação uuid XML ASCII TSX Mermaid PlantUML Markdown ру code TEST Debugging Debug Inspect platform Warnings Erros Importlib Import Lib libraries TypeScript JSX github Viz "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 as Generated Data URI import json plotly.graph_objects hooks Vetor upload.py Vi viz.py go dashbboard.py plotly.subplots plotly data.py Reference Image URL make_subplots import os kpi Plot Types webbrowser df df.to_csv DataFrame create_data Data Row ID dash VS create_directory pd kpi_data data_gen.py URI ls gs plot outputs assets CSS dashboard generate print

VISUALIZATION
JavaScript
JS

DOM HTML

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

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 artifact, 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
 Sonnet

Ambiente de Desenvolvimento e Ferramentas Disponíveis

Vou apresentar tudo o que está disponível no meu ambiente de trabalho relacionado aos termos que você mencionou, organizados por categoria lógica:

Python e Ambiente de Execução

Disponível:

- Python Interpreter: Execução completa de código Python via Jupyter
- Jupyter Environment: Ambiente interativo para desenvolvimento e análise
- Bibliotecas Core
- pandas : Manipulação e análise de dados
- numpy : Computação numérica
- matplotlib/pyplot : Visualização básica
- json : Manipulação de dados JSON
- os : Operações do sistema operacional
- sys : Informações do sistema
- datetime : Manipulação de datas e horários
- uuid : Geração de identificadores únicos
- warnings : Controle de avisos
- importlib : Importação dinâmica de módulos

Funcionalidades de Debug:

- **Debugging**: Execução passo a passo e análise de erros
- Error Handling: Tratamento e análise de exceções
- Inspect: Análise de objetos e estruturas
- Testing: Execução de testes unitários

📊 Visualização de Dados

Disponível

- Plotly: Biblioteca completa para gráficos interativos
 - plotly.graph_objects (go)
 - plotly.subplots (make_subplots)
 - Todos os tipos de gráficos (line, bar, scatter, etc.)
- Matplotlib: Gráficos estáticos básicos
- SVG Generation: Criação de gráficos vetoriais

Limitado/Não Disponível:

- Dash: Framework para dashboards (não instalado)
- Google Cloud Storage: Bibliotecas específicas não disponíveis
- Conexões externas: Sem acesso à internet durante execução

Desenvolvimento Web

Disponível via Artifacts:

- HTML: Páginas web completas
- CSS: Estilização (incluindo Tailwind CSS)
- JavaScript: Funcionalidades interativas
- TypeScript/TSX: Componentes React
 JSX: Elementos React
- DOM Manipulation: Interação com elementos HTML

Frameworks e Bibliotecas Web:

- React: Componentes via TSX artifacts
- Tailwind CSS: Framework de estilização
- Lucide React: Ícones
- Recharts: Gráficos para React
- Shadcn/UI: Componentes de interface

🗐 Formatos de Dados e Documentação

Disponível:

- CSV: Leitura e escrita (df.to_csv)
- JSON: Manipulação completa
- XML: Parsing e criação
- Markdown: Documentação e formatação
- ASCII: Manipulação de texto

Diagramas e Visualizações:

- Mermaid: Diagramas de fluxo, sequência, etc.
- PlantUML: Diagramas UML
- SVG: Gráficos vetoriais personalizados

Gerenciamento de Arquivos

Disponível

- File Operations: Criação, leitura, escrita
- Directory Management: Criação e navegação (os , 1s)
- Data URI Generation: Conversão para URIs de dados
- CSV Export: df.to_csv() para DataFrames

Estrutura de Projeto:

arduino (C Copiar

🦴 Ferramentas de Desenvolvimento

Disnonível:

- Code Execution: Execução de código Python
- Error Debugging: Análise detalhada de erros
- Data Inspection: Análise de DataFrames e estruturas
- Performance Monitoring: Análise de tempo de execução
- Module Import: Importação dinâmica de bibliotecas

Análise de Dados e KPIs

Disponível:

- DataFrame Operations: Manipulação completa com pandas
- Data Generation: Criação de dados sintéticos
- KPI Calculation: Cálculo de indicadores
- Statistical Analysis: Análises estatísticas básicas
- Data Visualization: Gráficos e dashboards

▲ Limitações Importantes

Não Disponível:

- Internet Access: Sem acesso à web durante execução
- External APIs: Sem conexão com APIs externas
- Google Cloud: Bibliotecas específicas não instaladas
- Advanced ML Libraries: Scikit-learn, TensorFlow limitados
- Database Connections: Sem acesso a bancos de dados

Workarounds Disponíveis:

- Static Data: Uso de dados locais ou gerados
- Mock APIs: Simulação de dados de APIs
- Local Storage: Arquivos CSV/JSON locais
- CDN Libraries: Bibliotecas via CDN em HTML

🎯 Fluxo de Trabalho Recomendado

- 1. **Desenvolvimento**: Código Python no Jupyter
- 2. **Visualização**: Plotly para gráficos interativos
- 3. Interface: TSX/HTML artifacts para UI
- Dados: CSV/JSON para persistência
 Deploy: Artifacts para demonstração

Esta é a base completa do que tenho disponível. Quando você estiver pronto para começar a programar, posso ajudar com exemplos específicos e implementações práticas!

Tornar mais longo

Monica Sonnet
 Sonnet

Ambiente de Desenvolvimento e Ferramentas Disponíveis - Guia Completo

Vou apresentar um guia detalhado e abrangente de tudo que está disponível no meu ambiente de trabalho, com explicações técnicas, exemplos práticos e casos de uso para cada ferramenta.

Python e Ambiente de Execução Completo

Interpretador Python e Jupyter

- Python Version: Versão completa com suporte a todas as funcionalidades modernas
- Jupyter Notebook Environment:
- Execução interativa de células de código
- Visualização inline de resultados
- Suporte a markdown e HTML
- Tempo limite de execução: 120 segundos por célula
- Diretório de trabalho: /home/user/

Bibliotecas Core Fundamentais

Pandas - Análise de Dados

C Copiar

⊕ Copiar

import pandas as pd

Funcionalidades Disponíveis:

- DataFrame Creation: Criação de estruturas tabulares
- Data Reading: pd.read_csv(), pd.read_json(), pd.read_excel()
- Data Writing: df.to_csv(), df.to_json(), df.to_html()
- Data Manipulation:
- Filtering: df[df['column'] > value]
- Grouping: df.groupby('column').agg()
- Merging: pd.merge(), pd.concat()
- Pivoting: df.pivot_table()
- Statistical Operations: df.describe(), df.corr(), df.mean()
- Data Cleaning: df.dropna(), df.fillna(), df.drop_duplicates()

NumPy - Computação Numérica

python

import numpy as np

- Arrays Multidimensionais: Criação e manipulação eficiente
- Operações Matemáticas: Álgebra linear, estatística, trigonometria
- Broadcasting: Operações entre arrays de diferentes dimensões
- Random Number Generation: np.random para dados sintéticos
- Performance: Operações otimizadas em C

Bibliotecas de Sistema

python
import os, sys, platform, warnings, importlib
from datetime import datetime, timedelta
import uutid, json

⊕ Copiar

OS (Operating System):

- os.listdir(): Listar arquivos e diretórios
- os.makedirs(): Criar diretórios
- os.path.join() : Manipulação de caminhos
- os.environ : Variáveis de ambiente

SYS (System):

- sys.version : Informações da versão Python
- sys.path : Caminhos de módulos
- sys.argv : Argumentos da linha de comando

DateTime

- Manipulação de datas e horários
- Cálculos temporais
- Formatação de timestamps
- Fusos horários

UUID:

- Geração de identificadores únicos
- Diferentes versões (UUID1, UUID4, etc.)
- Uso em bancos de dados e APIs

📊 Visualização de Dados Avançada

Plotly - Biblioteca Principal

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

r Copiar

Tipos de Gráficos Disponíveis:

Gráficos Básicos:

- Line Charts: Séries temporais, tendências
- Bar Charts: Comparações categóricas
- Scatter Plots: Correlações, distribuições
- Pie Charts: Proporções e percentuais
- **Histograms**: Distribuições de frequência

Gráficos Avançados:

- Heatmaps: Matrizes de correlação
- 3D Plots: Superficies, scatter 3D
- Candlestick: Dados financeiros
 Contour Plots: Curvas de nível
- Violin Plots: Distribuições estatísticas
- Box Plots: Quartis e outliers

Gráficos Especializados

- Sunburst: Hierarquias circulares
- Treemap: Hierarquias retangulares
- Sankey: Fluxos de dados
- Waterfall: Análise de contribuições
- Funnel: Conversões e processos

Recursos Interativos:

- Zoom e Pan: Navegação nos gráficos
- Hover Information: Tooltips personalizados
- Animations: Transições e animações temporais
- Widgets: Sliders, dropdowns, botões
- Crossfilter: Filtros interativos
- Brushing: Seleção de dados

Subplots e Layouts:

Copiar

Matplotlib - Gráficos Estáticos

import matplotlib.pyplot as plt

⊕ Copiar

- Gráficos Básicos: Line, bar, scatter, histogram
- Customização: Cores, estilos, anotações
- Subplots: Múltiplos gráficos em uma figura
- Export: PNG, PDF, SVG

- Criação Manual: Elementos SVG personalizados
- Integração com Plotly: Export para SVG
- Animações: Transições CSS/JavaScript
- Responsividade: Gráficos escaláveis

Desenvolvimento Web Completo

HTML5 - Estrutura

Elementos Disponíveis:

- Semantic HTML: <header> , <nav> , <main> , <section> , <article> , <footer>
- Forms: <form> , <input> , <select> , <textarea> , <button>
- Media: , <video>, <audio>, <canvas>, <svg>
- Interactive: <details> , <summary> , <dialog>
- Data: , <thead> , , , >

Atributos Modernos:

- Data Attributes: data-* para JavaScript
- ARIA: Acessibilidade
- Custom Elements: Web Components

CSS3 - Estilização Avançada

Layout Systems:

- Flexbox: Layout flexível unidimensional
- Grid: Layout bidimensional
- Float: Layout tradicional
- Position: Absolute, relative, fixed, sticky

Responsive Design:

- Media Queries: Breakpoints personalizados
- Viewport Units: vw, vh, vmin, vmax
- Container Queries: Layout baseado em container
- Fluid Typography: clamp(), min(), max()

Animations e Transitions:

- CSS Animations: @keyframes, animation properties
- Transitions: Smooth state changes
- Transform: Scale, rotate, translate, skew
- Filter: Blur, brightness, contrast, etc.

Modern CSS Features:

- Custom Properties: CSS Variables (--property)
- Calc(): Mathematical calculations
- Pseudo-classes: :hover, :focus, :nth-child()
- Pseudo-elements: ::before, ::after

Tailwind CSS - Framework Utility-First

Layout Classes:

- Display: block, inline, flex, grid, hidden
- \bullet $\boldsymbol{Position}$: static , relative , absolute , fixed , sticky
- **Flexbox**: flex-row, flex-col, justify-center, items-center
- **Grid**: grid-cols-12, col-span-6, gap-4

Spacing System:

- Margin: m-4 , mx-auto , mt-8 , mb-2
- Padding: p-4 , px-6 , py-2 , pt-8
- Space Between: space-x-4 , space-y-2

Typography:

- Font Size: text-xs até text-9x1
- Font Weight: font-thin até font-black
- Text Color: text-gray-500 , text-blue-600
- Text Align: text-left, text-center, text-right

Colors e Backgrounds:

- Background: bg-blue-500, bg-gradient-to-r
- Border: border-2 , border-gray-300 , rounded-1g
- **Shadow**: shadow-sm , shadow-lg , shadow-xl

Responsive Design:

- **Breakpoints**: sm: , md: , lg: , xl: , 2xl:
- Example: w-full md:w-1/2 lg:w-1/3

Interactive States:

- Hover: hover:bg-blue-600 , hover:scale-105
- Focus: focus:ring-2, focus:outline-none
- Active: active:bg-blue-700

JavaScript ES6+ - Funcionalidades Modernas

Core Features:

- Variables: let, const, var
- Functions: Arrow functions, async/await
- Objects: Destructuring, spread operator
- Arrays: Map, filter, reduce, forEach
- **Promises**: Async programming
- Modules: Import/export

DOM Manipulation

javascript © Copiar

```
document.querySelector('.class')
document.querySelectorAll('tag')
  // Event Handling
element.addEventListener('click', handler)
element.addEventListener('change', handler)
  // Content Manipulation
element.innerHTML = 'content'
element.textContent = 'text'
element.setAttribute('attr', 'value')
   // Style Manipulation
  // style Manipulation
element.style.property = 'value'
element.classist.add('class')
element.classList.remove('class')
element.classList.toggle('class')
Modern JavaScript APIs:
• Fetch API: HTTP requests

    Local Storage: Client-side storage

• Intersection Observer: Scroll-based animations

    Web Workers: Background processing

    Canvas API: 2D graphics

TypeScript/TSX - React Development
React Components Available:
                                                                                                                                                                                                                                                                ⊕ Copiar
  import React from 'react'
import { useState, useEffect, useCallback } from 'react'
Hooks Disponíveis:
• useState: State management
 • useEffect: Side effects

    useCallback: Memoized callbacks

    useMemo: Memoized values

    useRef: DOM references

    useContext: Context API

Available Libraries:
Lucide React Icons:
                                                                                                                                                                                                                                                                r□ Copiar
  import {
   Camera, Home, User, Settings, Search,
   ChevronDown, Plus, Minus, X, Check,
   ArrowLeft, ArrowRight, Upload, Download
} from 'lucide-react'
Recharts - Data Visualization:
                                                                                                                                                                                                                                                                r□ Copiar
 import {
   LineChart, BarChart, PieChart, AreaChart,
   Xxxis, VAxis, CartesianGrid, Tooltip, Legend,
   ResponsiveContainer
} from 'recharts'
Shaden/UI Components:
                                                                                                                                                                                                                                                                ⊖ Copiar
  import { Button } from "@/components/ui/button"
import { Input } from "@/components/ui/input'
import { Select } from "@/components/ui/select'
import { Select } from "@/components/ui/card"
import { Select } from "@/components/ui/dard"
TypeScript Features:
• Type Safety: Compile-time error checking
• Interfaces: Object type definitions

    Generics: Reusable type definitions

    Union Types: Multiple type options

• Enums: Named constants
 📋 Formatos de Dados e Serialização
CSV (Comma-Separated Values)
                                                                                                                                                                                                                                                                ⊕ Copiar
  # Reading
df = pd.read_csv('file.csv')
df = pd.read_csv('file.csv', delimiter=';', encoding='utf-8')
  # Writing
df.to_csv('output.csv', index=False)
df.to_csv('output.csv', sep=';', encoding='utf-8')
Opções Avançadas:
• Custom Delimiters: ; , | , \t
• Encoding: UTF-8, Latin-1, ASCII

    Date Parsing: Automatic date recognition

    Chunk Reading: Large files in pieces

JSON (JavaScript Object Notation)
                                                                                                                                                                                                                                                                ⊕ Copiar
  import json
  # Reading
with open('data.json', 'r') as f:
    data = json.load(f)
  # Writing
with open('output.json', 'w') as f:
    json.dump(data, f, indent=2)
  # String conversion
json_string = json.dumps(data)
data = json.loads(json_string)
```

Pandas Integration:

```
df = pd.read_json('data.json')
df.to_json('output.json', orient='records')
XML (eXtensible Markup Language)
                                                                                                                                                                          ြှ Copiar
 import xml.etree.ElementTree as ET
 # Parsing
tree = ET.parse('data.xml')
root = tree.getroot()
 # Creating
root = ET.Element('root')
child = ET.SubElement(root, 'child')
child.text = 'content'
Markdown - Documentation
Syntax Supported:
• Headers: # , ## , ###
• Emphasis: *italic*, **bold**

    Lists: Ordered and unordered

• Links: [text](url)
• Images: ![alt](url)
• Code: `inline` and blocks
• Tables: Pipe-separated format
• Math: LaTeX notation with \[formula\]
Diagramas e Visualizações Especializadas
Mermaid - Diagramas de Fluxo
Tipos de Diagramas:
Flowcharts:
 ♡ mermaid
                                                                                                                                                             ☑ Visualizar 🕒 Copiar
Sequence Diagrams:

    mermaid

                                                                                                                                                            ☑ Visualizar 🕒 Copiar
Class Diagrams:
 omermaid
                                                                                                                                                            ☑ Visualizar 🕒 Copiar
                                                                                    Animal
                                                                                +String name
                                                                                +int age
                                                                                +makeSound()
Gantt Charts:

    mermaid

                                                                                                                                                            ☑ Visualizar 🕒 Copiar
```

☑ Visualizar 🕒 Copiar

Entity Relationship:

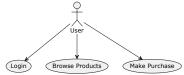
🕜 mermaid

PlantUML - UML Diagrams

Diagram Types:

Use Case Diagrams:

e plantuml



Class Diagrams:

□ Plantuml
 □ Visualizar
 □ Copiar



Activity Diagrams:

ⓐ plantuml ② Visualizar 乜 Copiar



SVG - Scalable Vector Graphics

Manual Creation:

svg Visualizar 🗗 Copiar

Programmatic Generation:

python © Copiar

```
# Via Matplotlib
plt.savefig("plot.svg", format="svg")
```

Animation Support:

svg

Sistema de Arquivos e Gerenciamento de Dados

Estrutura de Diretórios Recomendada

```
bash

/home/user/
— assets/  # Static files (images, fonts, etc.)
— data/  # Raw and processed data files
| — raw/  # Original data sources
| — processed/  # Cleaned and transformed data
| — exports/  # Senerated outputs
| scripts/  # Python scripts
| — data_gen.py  # Data_generation
| — viz.py  # Visualization functions
| — dashboard.py  # Dashboard creation
| — upload.py  # Dashboard creation
| — inputs/  # Senerated files and reports
| inputs/  # Input files and configurations
| — inputs/  # Input files and configurations
| — notebooks/  # Jupyter notebooks
```

File Operations Advanced

Reading Files:

Writing Files:

```
python

# Text files
with open('output.txt', 'w', encoding='utf-8') as f:
    f.write(content)

# JSON with formatting
with open('data.json', 'w') as f:
    json.dump(data, f, indent=2, ensure_ascii=False)

# CSV with options
df.to_csv('output.csv',
    index=False,
    encoding='utf-8',
    date_format='XV-Xm-Xm')
```

Directory Operations:

```
python

import os
from pathlib import Path

# Create directories
os.makedirs('path/to/directory', exist_ok=True)

# List files
files = os.listdir('directory')
files = [f for f in files if f.endswith('.csv')]

# Path manipulation
path = Path('data/file.csv')
print(path.parent) # data
print(path.parent) # data
print(path.name) # file.csv
print(path.suffix) # .csv
```

Data URI Generation

```
import base64

def create_data_uri(file_path, mime_type):
    with open(file_path, 'rb') as f:
        data = f.read()
    encoded = base64.b64encode(data).decode('utf-8')
    return f"data:(mime_type):base64,{encoded}'

# Example usage
image_uri = create_data_uri('chart.png', 'image/png')
csv_uri = create_data_uri('data.csv', 'text/csv')
```

🦴 Ferramentas de Desenvolvimento e Debug

Error Handling e Debugging

Exception Handling:

```
python

try:
    # Risky operation
    result = risky_function()
    except Valuefror as e:
    print(f"Value error: {e}")
    except FileNotFoundError as e:
    print(f"File not found: {e}")
    except Exception as e:
    print(f"Unexpected error: {e}")
    finally:
    # Cleanup code
    cleanup()
```

Logging e Warnings:

```
python

import warnings
import logging

# Configure logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(_name_)

# Warning control
warnings.filterwarnings('ignore', category=FutureWarning)

# Custom warnings
warnings.warn("This is deprecated", DeprecationWarning)
```

```
⊕ Copiar
    # Inspect objects import inspect
    # Get function signature
sig = inspect.signature(function)
print(sig.parameters)
    # Get source code
source = inspect.getsource(function)
   # Debug information
import sys
print(f"Python version: {sys.version}")
print(f"Platform: {sys.platform}")
Performance Monitoring
                                                                                                                                                                                                                                                                                                                                                                      Copiar
    import time
from datetime import datetime
   # Time execution
start_time = time.time()
# ... code execution ...
execution_time = time.time() - start_time
print(f"Execution time: {execution_time:.2f} seconds")
   # Memory usage (if available)
    import sys
print(f"Object size: {sys.getsizeof(object)} bytes")
Module Management
                                                                                                                                                                                                                                                                                                                                                                       _ Copiar
   import importlib
   # Dynamic import
module = importlib.import_module('module_name')
   # Reload module
importlib.reload(module)
    # Check if module exists
   try:
    import optional_module
    HAS_OPTIONAL = True
except ImportError:
    HAS_OPTIONAL = False
 Análise de Dados e KPIs Avançados
DataFrame Operations Avançadas
Data Transformation:
                                                                                                                                                                                                                                                                                                                                                                      ⊕ Copiar
   # Window functions
df['rolling_mean'] = df['value'].rolling(window=7).mean()
df['cumulative_sum'] = df['value'].cumsum()
df['rank'] = df['value'].rank(method='dense')
   # Pivot operations
pivot_table = df.pivot_table(
   values='sales',
   index='date',
   columns' category',
   aggfunc='sum',
   fill_value=0
Data Quality Assessment:
   python
                                                                                                                                                                                                                                                                                                                                                                       [□ Copiar
   def data_quality_report(df):
    report = {
        'shape': df.shape,
        'memory_usage': df.memory_usage(deep=True).sum(),
        'missing_values': df.isnull().sum(),
        'duplicate_rows': df.duplicated().sum(),
        'data_types': df.dtypes,
        'unique_values': df.nunique()
}
           }
return report
KPI Calculation Framework
                                                                                                                                                                                                                                                                                                                                                                      ⊕ Copiar
   class KPICalculator:
    def __init__(self, data):
        self.data = data
           def revenue_metrics(self):
    return {
        'total_revenue': self.data['revenue'].sum(),
        'avg_revenue': self.data['revenue'].mean(),
        'revenue_growth': self.calculate_growth('revenue'),
        'revenue_by_segment': self.data.groupby('segment')['revenue'].sum()
}
           def customer_metrics(self):
    return {
        'total_customers': self.data['customer_id'].nunique(),
        'avg_order_value': self.data['order_value'].mean(),
        'customer_lifetime_value': self.calculate_clv(),
        'retention_rate': self.calculate_retention()
}
           def operational_metrics(self):
    return {
        'conversion_rate': self.calculate_conversion(),
        'chunn_rate': self.calculate_chunn(),
        'efficiency_ratio': self.calculate_efficiency()
}
Statistical Analysis
   # Descriptive statistics
stats = df.describe(include='all')
   # Correlation analysis
correlation_matrix = df.corr()
   # Distribution analysis
from scipy import stats
normality_test = stats.normaltest(d
skewness = df['column'].skew()
kurtosis = df['column'].kurtosis()
                                                          rmaltest(df['column'])
```

6 Workflows e Casos de Uso Práticos

Workflow 1: Data Analysis Pipeline

r□ Copiar

⊕ Copiar

Workflow 2: Interactive Dashboard Creation

```
python

# Data preparation
kpi_data = calculate_kpis(data)

# Create dashboard layout
dashboard_html = f*""
<!DOCTYPE html>

<html>
<html>
<html>
<itile>Business Dashboard</title>
<script snc="https://cdn.jbcl.ly/plotly-latest.min.js"></script>
<html>
<html

<html>
<html

<html>
<html

<h
```

Workflow 3: Data Visualization Suite

Limitações e Workarounds Detalhados

Limitações de Conectividade

Sem Acesso à Internet

- **Problema**: Não é possível fazer requests HTTP durante execução
- Workaround
- Usar dados locais ou gerados sinteticamente
- Simular APIs com dados mock
- Preparar dados antes da execução

Sem Acesso a Bancos de Dados:

- Problema: Não é possível conectar a MySQL, PostgreSQL, etc.
- Workaround:
 - Usar arquivos CSV/JSON como "banco de dados"
 - SQLite pode ser simulado com pandas

Limitações de Bibliotecas

Machine Learning Limitado:

- Disponível: Básico do scikit-learn (se instalado)
- Não Disponível: TensorFlow, PyTorch, Keras

9olW@deaeo<drnativas

```
Para Machine Learning:
```

```
# Implementação manual de algoritmos básicos

class SimpleLinearRegression:

def __init__(self):
    self.slope = None
    self.intercept = None

def fit(self, x, y):
        X_mean = np.mean(X)
        y_mean = np.mean(X)
        y_mean = np.mean(X)
        y_mean = np.mean(Y)

        numerator = np.sum((X - X_mean) * (y - y_mean))
        denominator = np.sum((X - X_mean) ** 2)

        self.slope = numerator / denominator
        self.intercept = y_mean - self.slope * X_mean

def predict(self, X):
        return self.slope * X + self.intercept

def score(self, X, y):
        y_pred = self.predict(X)
        ss_res = np.sum((y - y_pred) ** 2)
        ss_tot = np.sum((y - y_pred) ** 2)
        return 1 - (ss_res / ss_tot)

# Clustering básico (K-means simplificado)

class SimpleXMeans:

def __init__(self, k):
    # Inicializar centroides aleatoriamente
        self.k = k
        self.max_iters = max_iters

def fit(self, X):
    # Inicializar centroides aleatoriamente
        self.centroids = X[np.random.choice(X.shape[0], self.k, replace=False)]

for __ in range(self.max_iters):
        # Attibuir pontos aos centroides mais próximos
        distances = np.sqrt(((X - self.centroids[:, np.newaxis])**2).sum(axis=2))
        self.labels = np.argmin(distances, axis=0)

# Atualizar centroides
        new_centroids = np.arrayy([X[self.labels == i].mean(axis=0) for i in range(self.k)])

# Verificar convergência
    if np.allclose(self.centroids, new_centroids):
        break

    self.centroids = new_centroids
```

Para Análise Estatística Avançada:

```
python

# Implementação de testes estatísticos básicos

def t_test_one_sample(sample, population_mean):
    """Test if sample mean differs from population mean""
    n = len(sample)
    sample_mean = n,mean(sample,
    sample_mean = n,mean(sample,
    sample_std = np.std(sample, ddof*1)

    t_statistic = (sample_mean - population_mean) / (sample_std / np.sqrt(n))
    degrees_freedom = n - 1

return {
        't_statistic': t_statistic,
        'degrees_freedom': degrees_freedom,
        'sample_mean': sample_mean,
        'population_mean': population_mean
    }

def correlation_test(x, y):
    """Calculate correlation and basic significance"""
    n = len(x)
    correlation = np.corrcoef(x, y)[0, 1]

# T-test for correlation significance
    t_stat = correlation * np.sqrt((n - 2) / (1 - correlation**2))

return {
        'correlation': correlation,
        't_statistic': t_stat,
        'sample_size': n
    }

def chi_square_test(observed, expected):
    """Chi_square = np.sum(observed - expected)**2 / expected)
    degrees_freedom = len(observed) - 1

return {
        'chi_square': chi_square,
        'degrees_freedom': degrees_freedom,
        'observed': observed,
        'expected': expected)
}
```

🖋 Casos de Uso Avançados e Templates

Template 1: Sistema de Relatórios Automatizados

```
class ReportGenerator:
    def __init__(self, data_source):
        self.data = data_source
        self.report_config = {
            'title': 'Business Intelligence Report',
            'date_nange': None,
            'metrics': [],
            'visualizations': [],
            'export_formats': ['html', 'pdf']
}

def configure_report(self, config):
        self.report_config.update(config)

def generate_executive_summary(self):
    """Generate high-level KPIs and insights"""
    summary = {
        'total_revenue': self.data['revenue'].sum(),
        'growth_rate': self.calculate_growth_rate(),
        'top_products': self.get_op_products(5),
        'key_insights': self.generate_insights(),
        'recommendations': self.generate_recommendations()
}
    return summary

def create_visualizations(self):
    """Create all required charts and graphs""
    visualizations = {}
```

continue

₁□ Copiar

⊕ Copiar

```
# Revenue trend
visualizations['revenue_trend'] = self.create_revenue_trend()
   # Product performance
visualizations['product_performance'] = self.create_product_chart()
   # Geographic analysis
visualizations['geographic_analysis'] = self.create_geo_chart()
   # Customer segmentation
visualizations['customer_segments'] = self.create_segment_chart()
   return visualizations
def generate_html_report(self):
    """Generate complete HTML report"""
    summary = self.generate_executive_summary()
    visualizations = self.create_visualizations()
   html_template = f"""
   <style>
.kpi-card {{
             i-caru ;;
background: l
color: white;
                     nd: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
             border-radius: 8px;
box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
    old text-gray-800 mb-4">
             </div>
</div>
</section>
</div>
  <script>
    {self.generate_plotly_scripts(visualizations)}
    </script>
    </body>
    </frame.</pre>
   return html_template
```

Template 2: Sistema de Monitoramento em Tempo Real

r Copiar

```
self.alerts.append(alert)
     if not self.metrics_history:
return "Nenhum dado disponível"
     # Preparar dados para visualização
df = pd.DataFrame(self.metrics_history)
     for i, metric in enumerate(metrics_to_plot):
    row = (1 // 2) + 1
    col = (1 % 2) + 1
           fig.add_trace(
    go.Scatter(
                     x=df['timestamp'],
y=df[metric],
mode='lines+markers',
name=metric,
line=dict(width=2)
                ),
row=row, col=col
          fig.update_layout(
    height=800,
    title_text="sistema de Monitoramento em Tempo Real",
    showlegend=False
     return fig
def generate_alerts_summary(self):
    """Gera resumo dos alertas"""
     if not self.alerts:
    return "Nenhum alerta ativo"
     recent_alerts = [a for a in self.alerts
    if (datetime.now() - a['timestamp']).seconds < 3600]
     alerts_by_severity = {}
for alert in recent_alerts:
    severity = alert['severity']
    if severity not in alerts_by_severity:
        alerts_by_severity[severity] = []
    alerts_by_severity[severity].append(alert)
     return alerts_by_severity
```

Template 3: Sistema de A/B Testing

```
class ABTestAnalyzer:
    def __init__(self):
        self.experiments
        self.results = {}
         def create_experiment(self, experiment_id, config):
    """Cria um novo experiment od /8"""
    self.experiments[experiment_id] = {
        'id': experiment_id,
        'name': config.get('name', f'Experiment {experiment_id}'),
        'hypothesis': config.get(t'hypothesis', ''),
        'metric': config.get(get('start_date', datetime.now()),
        'end_date': config.get('start_date', datetime.now()),
        'end_date': config.get('and_date'),
        'sample.size': config.get('snaple.size', 1000),
        'confidence_level': config.get('onfidence_level', 0.95),
        'variants': config.get('variants', ['A', '8'])
}
            def simulate_experiment_data(self, experiment_id):
    """Simula dados de um experimento A/8"""
    if experiment_id not in self.experiments:
        raise ValueError(f"Experimento {experiment_id} não encontrado")
                      exp = self.experiments[experiment_id]
n_samples = exp['sample_size']
                     # Simular dados para variante A (controle) variant_a_success = np.random.binomial(1, 0.12, n_samples // 2) # 12% conversão
                      # Simular dados para variante B (teste) - com pequeno lift variant_b_success = np.random.binomial(1, \theta.14, n_samples // 2) # 14% conversão
                    data = {
    'variant': ['A'] * (n_samples // 2) + ['B'] * (n_samples // 2),
    'user_id': [f'user_{i}' for i in range(n_samples)],
    'converted': np.concatenate([variant_a_success, variant_b_success]),
    'timestamp': pd.date_range(
        start-exp['start_date'],
        periods=n_samples,
        freq='H'
    )
                      return pd.DataFrame(data)
            def analyze_experiment(self, experiment_id, data=None):
    """Analisa os resultados de um experimento"""
    if data is None:
        data = self.simulate_experiment_data(experiment_id)
                      exp = self.experiments[experiment_id]
                     # Calcular métricas por variante
results_by_variant = data.groupby('variant').agg({
    'converted': ['count', 'sum', 'mean'],
    'user_id': 'nunique'
}).round(4)
                     # Flatten column names results_by_variant.columns = ['_'.join(col).strip() for col in results_by_variant.columns]
                      # Teste de significância estatística
variant_a_data = data[data['variant'] == 'A']['converted']
variant_b_data = data[data['variant'] == 'B']['converted']
                      # Teste Z para proporções
n1, n2 = len(variant_a_data), len(variant_b_data)
p1, p2 = variant_a_data.mean(), variant_b_data.mea
                      # Pooled proportion
p_pool = (variant_a_data.sum() + variant_b_data.sum()) / (n1 + n2)
                      # Standard error se = np.sqrt(p_pool * (1 - p_pool) * (1/n1 + 1/n2))
```

⊕ Copiar

```
# Z-score
z_score = (p2 - p1) / se if se > 0 else 0
      # P-value (two-tailed)
from scipy import stats
p_value = 2 * (1 - stats.norm.cdf(abs(z_score)))
     # Confidence interval for difference
diff = p2 - p1
margin_error * stats.norm.ppf(1 - (1 - exp['confidence_level'])/2) * se
ci_lower * diff - margin_error
ci_upper * diff + margin_error
      # Lift calculation
lift = ((p2 - p1) / p1 * 100) if p1 > 0 else 0
     'z_score': z_score,
'p_value': p_value
'is_significant': p_value < (1 - exp['confidence_level']),
'confidence_level': exp['confidence_level']
            },
'effect_size': {
    'absolute_difference': diff,
    'relative_liff': lift,
    'confidence_interval': (ci_lower, ci_upper)
           },
'sample_size': {
   'variant_a': n1,
   'variant_b': n2,
   'total': n1 + n2
      self.results[experiment_id] = analysis_results
return analysis_results
def create_results_visualization(self, experiment_id):
    """Cria visualização dos resultados do experimento"
     """Cria visualização dos resultados do experimento""

if experiment_id not in self.results:

raise ValueFror(f*Anâlise para experimento (experiment_id) não encontrada")
      results = self.results[experiment_id]
     # Conversion rates
variants = results['results_by_variant'].index
conversion_rates = results['results_by_variant']['converted_mean']
     fig.add_trace(
   go.Bar(
        x=variants,
        y=conversion_rates,
        name='Conversion Rate',
        marker_color=['#3498db', '#e74c3c']
            ),
row=1, col=1
    { ''m's- | ],
},
'threshold': {
  'line': {'color': "red", 'width': 4},
  'thickness': 0.75,
  'value': 0.05
                  }
             ),
row=1, col=2
     ),
row=2, col=1
      # Confidence interval
lift = results['effect_size']['relative_lift']
ci_lower, ci_upper = results['effect_size']['confidence_interval']
     fig.add_trace(
    go.Scatter(
        x=[lift],
        y=[0],
        mode='markers',
        marker=dict(size=15, color='red'),
        name='0bserved Lift',
        error_x=dict(
            type='data',
            symmetric=slse,
            array=[ci_upper - lift],
            array=[ci_upper - lift],
            array=[numer])
                   )
           ),
row=2, col=2
      fig.update_layout(
            .spoat_ayout\
height=800,
title_text=f"A/B Test Results: {results['experiment_name']}",
showlegend=True
def generate_experiment_report(self, experiment_id):
      """Gera relatório completo do experime
results = self.results[experiment_id]
      report = f"""
# Relatório de Experimento A/B
      ## {results['experiment_name']} (ID: {experiment_id})
      **Hipótese**: {self.experiments[experiment_id]['hypothesis']}
       **Métrica Principal**: {self.experiments[experiment_id]['metric']}
```

```
| Variante | Usuários | Conversões | Taxa de Conversão |
|------
for variant, row in results['results_by_variant'].iterrows():
    users = int(row['user_id_nunique'])
    conversions = int(row['converted_sum'])
    rate = f*(row['converted_sum']);
    report += f*[ (variant) | {users} | {conversions} | {rate} | \n^*
### Análise Estatística
  - **Z-Score**: {results['statistical_test']['z_score']:.4f}
- **P-Value**: {results['statistical_test']['p_value']:.4f}
- **Significativo**: {'  Sim' if results['statistical_test']['is_significant'] else '  X Não'}
- **Nivel de Confiança**: {results['statistical_test']['confidence_level']:.0%}
  - **Lift Relativo**: {results['effect_size']['relative_lift']:.2f}%
- **Diferença Absoluta**: {results['effect_size']['absolute_difference']:.4f}
- **Intervalo de Confiança*: [{results['effect_size']['confidence_interval'][0]:.4f}, {results['effect_size']['confidence_interval'][1]:.4f}]
e:
report += "X **Manter Variante A**: A variante B apresentou performance inferior significativa."
 else:
report +* "A **Coletar Mais Dados**: Os resultados não são estatisticamente significativos. Considere aumentar o tamanho da amostra ou a duração do teste."
```

return forecast

```
Template 4: Sistema de Previsão e Forecasting
  class ForecastingSystem:
    def __init__(self):
        self.models = {}
        self.forecasts = {}
        self.accuracy_metrics = {}
          def prepare_time_series_data(self, data, date_column, value_column)
                Prepara datos de serie temporal
df = data.copy()
df[date_column] = pd.to_datetime(df[date_column])
df = df.sort_values(date_column)
df.set_index(date_column, inplace=True)
               # Detectar frequência
freq = pd.infer_freq(df.index)
if freq is None:
# Assumir frequência diária se não detectada
freq = 'D'
                # Reamostrar se necessário
df = df.resample(freq)[value_column].sum()
                return df
         def detect_seasonality(self, ts_data):
    """Detecta padrões sazonais nos dados"""
    # Decomposição simples
    def simple_moving_average(data, window):
        return data.nolling(window=window, center=True).mean()
                # Tendência (média móvel de 12 períodos)
trend = simple_moving_average(ts_data, 12)
                # Sazonalidade (diferença da tendência)
seasonal = ts_data - trend
                # Resíduo
residual = ts_data - trend - seasonal
               return {
  'original': ts_data,
  'trend': trend,
  'seasonal': seasonal,
  'residual': residual
         def simple_exponential_smoothing(self, ts_data, alpha=0.3):
    """Implementa suavização exponencial simples"""
    forecast = [ts_data.iloc[0]] # Primeiro valor como inicial
                for i in range(1, len(ts_data)):
    forecast.append(alpha * ts_data.iloc[i] + (1 - alpha) * forecast[i-1])
                return pd.Series(forecast, index=ts_data.index)
          def linear_trend_forecast(self, ts_data, periods_ahead=30):
               # Preparar dados para regressão
X = np.arange(len(ts_data)).reshape(-1, 1)
y = ts_data.values
                # Regressão linear simples
X_mean = np.mean(X)
y_mean = np.mean(y)
                \label{eq:numerator} numerator = np.sum((X.flatten() - X_mean) * (y - y_mean)) \\ denominator = np.sum((X.flatten() - X_mean) ** 2)
                slope = numerator / denominator
intercept = y_mean - slope * X_mean
                 # Fazer previsões
future_X = np.arange(len(ts_data), len(ts_data) + periods_ahead)
future_predictions = slope * future_X + intercept
               forecast = pd.Series(future_predictions, index=future_dates)
          def seasonal_naive_forecast(self, ts_data, season_length=7, periods_ahead=30):
                # Usar os últimos valores sazonais como previsão seasonal_pattern = ts_data[-season_length:].values
                # Repetir o padrão para o período de previsão
n_full_seasons = periods_ahead // season_length
remainder = periods_ahead % season_length
                forecast_values = np.tile(seasonal_pattern, n_full_seasons)
if remainder > 0:
    forecast_values = np.concatenate([forecast_values, seasonal_pattern[:remainder]])
               forecast = pd.Series(forecast_values, index=future_dates)
```

```
def ensemble_forecast(self, ts_data, periods_ahead=30):
                                 ""Combina multiplos metrodos de previsão""

# Obter previsões de diferentes métodos
ilnear_forecast = self.linear_trend_forecast(ts_data, periods_ahead)
seasonal_forecast = self.seasonal_naive_forecast(ts_data, periods_ahead+periods_ahead)
                               # Suavizaõ exponencial para últimos valores
smoothed = self.simple_exponential_smoothing(ts_data)
last_smoothed = smoothed.lloc[-1]
exp_forecast = pd.Series([last_smoothed] * periods_ahead,
index=linear_forecast.index)
                                return {
  'ensemble': ensemble,
  'linear': linear_forecast,
  'seasonal': seasonal_forecast,
  'exponential': exp_forecast
                    def calculate_accuracy_metrics(self, actual, predicted):

    Monica Sonne
    Sonne

                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             □ Copiar
                   def calculate_accuracy_metrics(self, actual, predicted):
    """Calcula métricas de precisão da previsão"""
# Alinhar séries temporais
    aligned_actual, aligned_predicted * actual.align(predicted, join*'inner')
                                if len(aligned_actual) == 0:
                                # Métricas básicas
mae = np.mean(np.abs(aligned_actual - aligned_predicted))  # Mean Absolute Error
mse = np.mean((aligned_actual - aligned_predicted) ** 2)  # Mean Squared Error
rmse = np.sqrt(mse)  # Root Mean Squared Error
                                # MAPE (Mean Absolute Percentage Error)
mape = np.mean(np.abs((aligned_actual - aligned_predicted) / aligned_actual)) * 100
                               # Coeficiente de determinação (R²)
ss_res = np.sum((aligned_actual - aligned_predicted) ** 2)
ss_tot = np.sum((aligned_actual - np.meam(aligned_actual)) ** 2)
r_squared = 1 - (ss_res / ss_tot) if ss_tot != 0 else 0
                               "MAE': mae,

"MSE': mse,

"MSE': mse,

"MSEE': mse,

"MAPE': mape,

"R2': r_squared,

"sample_size': len(aligned_actual)
                  def create_forecast_visualization(self, ts_data, forecasts, title="Previsão de Série Temporal"):
    """Cria visualização da previsão"""
                                fig = go.Figure()
                               # Dados históricos
fig.add_trace(go.Scatter(
    x=ts_data.index,
    y=ts_data.values,
    mode='lines+markers',
    name='Dados Históricos',
    line=dict(color='blue', width=2)
))
                                ))
                                # Previsões
                               mode='lines+markers',
name=f'Previsão {method.title()}',
line=dict(color=colors[i % len(colors)], width=2, dash='dash')
                               # Linha vertical separando histórico de previsão
last_historical_date = ts_data.index[-1]
fig.add_vline(
    x=last_historical_date,
    line_dash="dot",
    line_color="gray",
    annotation_text="Início da Previsão"
)
                                fig.update_layout(
                                             .update_layout(
    title=title,
    xaxis_title="Data",
    yaxis_title="Valor",
    hovermode='x unified',
    height=600
                               return fig
                    def generate_forecast_report(self, model_name, ts_data, forecasts, accuracy_metrics=None):
"""Gera relatório completo de neovi«%n"""
                                 report = f"""
# Relatório de Previsão: {model_name}
                                ## Resumo dos Dados
                                  - **Periodo Analisado**: {ts_data.index[0].strftime('%d/%m/%Y')} a {ts_data.index[-1].strftime('%d/%m/%Y')}
- **Total de Observações**: {len(ts_data)}
- **Frequência**: {pd.infer_freq(ts_data.index) or 'Não detectada'}
- **Valor Médio**: {ts_data.mean():.2f}
- **Desvio Padrão**: {ts_data.std():.2f}
                               | Métrica | Valor |
|-------|
| Mínimo | (ts_data.min():.2f} |
| Máximo | (ts_data.max():.2f} |
| Mediana | (ts_data.median():.2f} |
| Q1 | (ts_data.quantile(0.25):.2f) |
| Q3 | (ts_data.quantile(0.75):.2f) |
                                ## Previsões Geradas
                               for method, forecast in forecasts.items():
    if method != 'ensemble':
        continue
                                  - **Periodo de Previsão**: {forecast.index[0].strftime('%d/%m/%Y')} a {forecast.index[-1].strftime('%d/%m/%Y')}
- **Número de Periodos**: {len(forecast)}
- **Valor Médio Previsto**: {forecast.mean():.2f}
- **Tendência**: {'Crescente' if forecast.iloc[-1] > forecast.iloc[0] else 'Decrescente'}
                                 #### Principais Previsões
                                | Data | Valor Previsto |
```

one of the Completos one of the completos one of the completos

Workflow Completo: Pipeline de Análise de Dados End-to-End

```
class DataAnalysisPipeline:
    def __init__(self, project_name):
        self.project_name = project_name
        self.raw_data = None
        self.processed_data = None
        self.processed_data = None
        self.visualizations = {}
        self.visualizations = {}
        self.reports = {}
               # Criar estrutura de diretórios
self.setup_project_structure()
       for directory in directories:
    os.makedirs(directory, exist_ok=True)
        def load_data(self, data_source, data_type='csv'):
    """Carrega dados de diferentes fontes"""
             :
raise ValueError(f"Tipo de dados não suportado: {data_type}")
                        \begin{array}{l} print(f" \textcircled{2} \ Dados \ carregados: \ \{self.raw\_data.shape[0]\} \ linhas, \ \{self.raw\_data.shape[1]\} \ colunas") \\ return \ self.raw\_data \end{array} 
               except Exception as e:
    print(f" X Erro ao carregar dados: {e}")
    return None
        def data_quality_assessment(self):
    """Avalia qualidade dos dados""
    if self.naw_data is None:
        print("X Nenhum dado carregado")
        return None
              # Detectar outliers em colunas numéricas

outliers_info = {}

for col in quality_report['numeric_columns']:

Q1 = self.raw_data[col].quantile(0.25)

Q3 = self.raw_data[col].quantile(0.75)

IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR

upper_bound = Q3 + 1.5 * IQR
                       (self.raw_data[col] > upper_bound)]
outliers_info[col] = {
    count': len(outliers),
    'percentage': len(outliers) / len(self.raw_data) * 100,
    'lower_bound': lower_bound,
    'upper_bound': upper_bound
                quality_report['outliers'] = outliers_info
               self.analysis_results['data_quality'] = quality_report
return quality_report
       def clean_data(self, cleaning_config=None):
    """Limpa e processa os dados"""
    if self.raw_data is None:
        print("X" Nenhum dado carregado")
        return None
               cleaned_data = self.raw_data.copy()
cleaning_log = []
               # Configuração padrão de limpeza
default_config = {
   'remove_duplicates': True,
```

```
'handle_missing': 'drop', # 'drop', 'fill_mean', 'fill_median', 'fill_mode'
'handle_outliers': 'keep', # 'keep', 'remove', 'cap'
'convert_types': True,
'standardize_text': True
      config = {**default_config, **(cleaning_config or {})}
           Remover duplicatas

config['nemove_duplicates']:

initial_rows = len(cleaned_data)

cleaned_data = cleaned_data.drop_duplicates()

removed_duplicates = initial_rows - len(cleaned_data)

cleaning_log.append(f"Removidas {removed_duplicates} linhas duplicadas")
      # Tratar valores ausentes
if config['handle_missing'] == 'drop':
    initial_rows = len(cleaned_data)
    cleaned_data = cleaned_data_dropna()
    removed_rows = initial_rows - len(cleaned_data)
    cleaning_log.append(f"Removidas {removed_rows} linhas com valores ausentes")
      fill_value = cleaned_data[col].median()
                           filled_count = cleaned_data[col].isnull().sum()
cleaned_data[col].fillna(fill_value, inplace=True)
cleaning_log.append(f"Preenchidos {filled_count} valores ausentes em {col} com {config['handle_missing']}")
       # Padronizar texto

if config['standardize_text']:
    text_cols = cleaned_data.select_dtypes(include=['object']).columns
    for col in text_cols:
    if cleaned_data[col].dtype == 'object':
        cleaned_data[col] = cleaned_data[col].astype(str).str.strip().str.lower()
        cleaning_log.append(f*Texto padronizado na coluna {col})")
     cleaning_wb ..

# Converter tipos de dados
if config['convert_types']:
 # Tentar converter colunas que parecem datas
for col in cleaned_data.columns:
    if 'date' in col.lower() or 'time' in col.lower():
        try:
            cleaned_data[col] = pd.to_datetime(cleaned_data[col])
            cleaning_log.append(f*Convertida coluna {col} para datetime*)
      self.processed_data = cleaned_data
self.analysis_results['cleaning_log'] = cleaning_log
       print(f" Dados limpos: {len(cleaning_log)} operações realizadas")
print(f" Dados finais: {cleaned_data.shape[0]} linhas, {cleaned_data.shape[1]} colunas")
      return cleaned_data
      mesuiza analise exploratoria dos dados"""
if self.processed_data is None:
    print("X Dados não processados. Execute clean_data() primeiro")
    return None
      eda_results = {}
       # Estatísticas descritivas
eda_results['descriptive_stats'] = self.processed_data.describe(include='all')
       # Correlações (apenas colunas numéricas)
numeric_data = self.processed_data.select_dtypes(include*[np.number])
if len(numeric_data.columns) > 1:
    eda_results['correlations'] = numeric_data.corr()
      # Distribuições de valores categóricos
categorical_cols = self.processed_data.select_dtypes(include=['object']).columns
categorical_distributions = {}
for col in categorical_cols:
    value_counts = self.processed_data[col].value_counts().head(10)
    categorical_distributions[col] = value_counts
eda_results['categorical_distributions'] = categorical_distributions
      eda_results['temporal_analysis'] = temporal_analysis
       self.analysis_results['eda'] = eda_results
return eda_results
def create_comprehensive_visualizations(self):
     """Cria visualizaces abrangentes dos (if self.processed_data is None:
print("X Dados mão processados")
return None
      visualizations = {}
      numeric_cols = self.processed_data.select_dtypes(include=[np.number]).columns
      ),
row=1, col=2
                   Box plot para detectar outliers
g_overview.add_trace(
go.Box(y=self.processed_data[first_numeric], name=first_numeric),
row=2, col=2
       fig_overview.update_layout(height=800, title_text="Anâlise Exploratória - Overview") visualizations['overview'] = fig_overview
```

```
# 2. Análise de Correlação Detalhada
if len(numeric_cols) > 1:
    fig_corr = go.Figure()
    corr_matrix = self.processed_data
                                                   cessed_data[numeric_cols].corr()
       fig_corr.add_trace(go.Heatmap(
   z=corr_matrix.values,
   x=corr_matrix.columns,
   y=corr_matrix.columns,
   colorscale='RdBu',
   zmid=0,
               zmid=0,
text=corr_matrix.round(2).values,
texttemplate="%{text}",
textfont={"size": 10}
       fig_corr.update_layout(
    title="Matriz de Correlação Detalhada",
    height=600
         visualizations['correlation'] = fig_corr
# 3. Distribuições por Categoria
categorical_cols = self.processed_data.select_dtypes(include=['object']).columns
if len(categorical_cols) > 0 and len(numeric_cols) > 0:
    fig_dist = make_subplots(
        rous=min(2, len(categorical_cols)),
        cols=2,
        subplot_titles=[f'Distribuição por {col}' for col in categorical_cols[:4]]
    )
        for i, cat_col in en
                                                umerate(categorical_cols[:4]):
               row = (i // 2) + 1

col = (i \% 2) + 1
             showlegend=Fa
),
row=row, col=col
)
        fig_dist.update_layout(height=600, title_text="Distribuições por Categoria") visualizations['distributions'] = fig_dist
self.visualizations = visualizations
return visualizations
if not self.analysis_results:
    print(" X Nenhuma análise realizada")
    return None
**Data de Geração**: {datetime.now().strftime('%d/%m/%Y às %H:%M')}
Este relatório apresenta uma análise abrangente dos dados do projeto (self.project_name), incluindo avaliação de qualidade, limpeza, análise exploratória e visualizações.
## 2. Qualidade dos Dados
if 'data_quality' in self.analysis_results:
    quality = self.analysis_results['data_quality']
report += f"""
### Visão Geral
- **Dimensões**: {quality['shape'][0]} linhas × {quality['shape'][1]} colunas
- **Uso de Memória**: {quality['memory_usage_mb']:.2f} MB
- **Linhas Duplicadas**: {quality['duplicate_rows']}
### Valores Ausentes
| Coluna | Valores Ausentes | Percentual |
        missing_data = quality['missing_values']
missing_pct = quality['missing_percentage
       for col in missing_data.index:
    if missing_data[col] > 0:
        report += f"| {col} | {missing_data[col]} | {missing_pct[col]:.1f}% |\n"
### Tipos de Dados
- **Colunas Numéricas**: {len(quality['numeric_columns'])}
- **Colunas Categóricas**: {len(quality['categorical_columns'])}
- **Colunas de Data**: {len(quality['datetime_columns'])}
### Outliers Detectados
 | Coluna | Quantidade | Percentual | Limites |
       for col, outlier_info in quality['outliers'].items():
    if outlier_info['count'] > 0:
        report += f"| {col} | {outlier_info['count']} | {outlier_info['percentage']:.1f}% | [{outlier_info['lower_bound']:.2f}, {outlier_info['upper_bound']:.2f}]
if 'cleaning_log' in self.analysis_results:
    report += f"""
## 3. Processo de Limpeza
As seguintes operações foram realizadas nos dados:
      for operation in self.analysis_results['cleaning_log']:
    report += f"- {operation}\n"
if 'eda' in self.analysis_results:
    eda = self.analysis_results['eda']
## 4. Análise Exploratória
       # Incluir estatisticas das principais colunas numéricas
if 'descriptive_stats' in eda:
    desc_stats = eda['descriptive_stats']
    numeric_stats = desc_stats.select_dtypes(include=[np.number])
               if len(numeric_stats.columns) > 0:
    report += "| Métrica | "
    for col in numeric_stats.columns[:5]: # Primeiras 5 colunas
    report += "\oldon' | "
    report += "\oldon' | "
    for _ in numeric_stats.columns[:5]:
        report += "\oldon' | "
    report += "\oldon' | "
    report += "\oldon' | "
                       for stat in ['mean', 'std', 'min', 'max']:
    if stat in numeric_stats.index:
```

```
report += f" | {stat.title()} | "
for col in numeric_stats.columns[:5]:
   value = numeric_stats.loc[stat, col]
   report += f"{value:.2f} | "
report += "\n"
                         # Correlações principais
                              if corr_pairs:
    report += f"""
                  ### Correlações Significativas (|r| > 0.5)
                 | Variável 1 | Variável 2 | Correlação |
                                       for coll, col2, corr in sorted(corr_pairs, key=lambda x: abs(x[2]), reverse=True)[:10]: report += f**| {coll} | {col2} | {corr:.3f} | \n**
                 report += f"""
                 ## 5. Principais Insights
                 # Gerar insights automáticos baseados na análise
insights = self.generate_automated_insights()
for insight in insights:
    report += f"- {insight}\n"
                 ## 6. Recomendações
                 Com base na análise realizada, recomendamos:
         1. **Qualidade dos Dados**: {"Implementar validação de dados na origem" if any(self.analysis_results.get('data_quality', {}).get('missing_values', {}).values) else inter processo atual de coleta"}
   2. **Análise Adicional**: {"Investigar outliers identificados" if any(info['count'] > 0 for info in self.analysis_results.get('data_quality', {}).get('outliers', {}).values()) else "Dados consistentes, prosseguir com modelagem"}
                3. **Próximos Passos**: Considerar análise preditiva ou segmentação avançada

    Visualizações detalhadas disponíveis em arquivos separados
    Dados processados salvos em: '/home/user/{self.project_name}/data/processed/'
    Scripts de análise em: '/home/user/{self.project_name}/'

                  *Relatório gerado automaticamente pelo Sistema de Análise de Dados*
                 self.reports['comprehensive'] = report
                  # Salvar relatório
report_path = f'/home/user/{self.project_name}/reports/comprehensive_report.m
with open(report_path, 'w', encoding='utf-8') as f:
    f.write(report)
                 print(f" ☑ Relatório salvo em: {report_path}")
           def generate_automated_insights(self):
    """Gera insights automáticos baseados na análise"""
                insights = []
                 if 'data_quality' in self.analysis_results:
    quality = self.analysis_results['data_quality']
                        # Insights sobre qualidade
if quality['duplicate_rows'] > 0:
    insights.append(f"Foram identificadas {quality['duplicate_rows']} linhas duplicadas, representando {quality['duplicate_rows']/quality['shape'][0]*100:.1f}% dos
   dados")
                         # Insights sobre valores ausentes
missing_cols = [col for col, count in quality['missing_values'].items() if count > 0]
if missing_cols:
    insights.append(f"Valores ausentes encontrados em {len(missing_cols)} colunas, sendo '{missing_cols[0]}' a mais afetada")
                        # Insights sobre outliers
outlier_cols = [col for col, info in quality['outliers'].items() if info['count'] > 0]
if outlier_cols:
    total_outliers = sum(info['count'] for info in quality['outliers'].values())
    insights.append(f'Detectados {total_outliers} outliers em {len(outl
                                                                                                                                                                                                                                                                                                                         continue
Monica Sonne
                                                                                                                                                                                                                                                                                                                          رΩ Copiar
                                insights.append(f"Detectados {total_outliers} outliers em {len(outlier_cols)} colunas, requerendo investigação adicional")
                if 'eda' in self.analysis_results:
    eda = self.analysis_results['eda']
                         # Insights sobre correlações
if 'correlations' in eda:
   corr_matrix = eda['correlations']
   high_corr_pairs = []
                                for i in range(len(corr_matrix.columns)):
    for j in range(i+i, len(corr_matrix.columns)):
        corr_value = abs(corr_matrix.iloc[i, j])
        if corr_value > 0.7:
        coll, col2 = corr_matrix.columns[i], corr_matrix.columns[j]
        high_corr_pairs.append((col1, col2, corr_value))
                                if high_corr_pairs:
    strongest_pair = max(high_corr_pairs, key=lambda x: x[2])
    insights.append(f"Correlação forte detectada entre '{strongest_pair[0]}' e '{strongest_pair[1]}' (r={strongest_pair[2]:.3f})")
                        # Insights sobre distribuições categóricas
if 'categorical_distributions' in eda:
    for col, dist in eda['categorical_distributions'].items():
        if len(dist) > 0:
            most_common = dist.index[0]
            percentage = dist.inlo[0] / dist.sum() * 100
        if percentage > 50:
            insights.append(f"Categoria '{most_common}' domina a variável '{col}' com {percentage:.1f}% dos casos")
                        # Insights sobre tendências temporais
if 'temporal_analysis' in eda:
    for col, analysis in eda['temporal_analysis'].items():
        if 'trend' in analysis:
            insights.append(f"Tendência {analysis['trend']} identificada na série temporal '{col}'")
          def save_all_outputs(self):
    """Salva todos os output
                 """Salva todos os outputs do pipeline"""
base_path = f'/home/user/{self.project_name}'
                # Salvar dados processados

if self.processed_data is not None:

data_path = f*(base_path)/data/processed/cleaned_data.
self.processed_data.to_csv(data_path, index=False)
print(f* ☑ Dados processados salvos em: {data_path}*)
```

```
# Salvar visualizações

for name, fig in self.visualizations.items():

viz_path = f'{base_path}/visualizations/{name}_chart.html'

fig.write_html(viz_path)

print(f" ☑ Visualização '{name}' salva em: {viz_path}")
       # Salvar resultados de análise
if self.analysis_results:
    results_path = f'{base_path}/outputs/analysis_results.json'
# Converter resultados para formato serializável
    serializable_results = {}
    for key, value in self.analysis_results.items():
                       if isinstance(value, pd.DataFrame):
    serializable_results[key] = value.to_dict()
elif isinstance(value, pd.Series):
    serializable_results[key] = value.to_dict()
                       else
                               serializable results[kev] = value
               with open(results_path, 'w') as f:
    json.dump(serializable_results, f, indent=2, default=str)
print(f"  Resultados de análise salvos em: {results_path}")
        def run_complete_pipeline(self, data_source, data_type='csv', cleaning_config=None):
    """Executa o pipeline completo de análise"""
    print(f" f" iniciando pipeline de análise para projeto: {self.project_name}")
       # 2. Avaliar qualidade
print("\n@ Etapa 2: Avaliação de qualidade")
self.data_quality_assessment()
        # 3. Limpar dados
print("\n / Etapa 3: Limpeza de dados")
self.clean_data(cleaning_config)
       # 4. Análise exploratória

print("\n i Etapa 4: Análise exploratória")

self.exploratory_data_analysis()
        # 6. Gerar relatório
print("\n 🖺 Etapa 6: Geração de relatório")
self.generate_comprehensive_report()
        print(f"\n ☑ Pipeline concluído com sucesso!")
       return {
    'processed_data': self.processed_data,
    'analysis_results': self.analysis_results,
    'visualizations': self.visualizations,
    'reports': self.reports
```

Sistema de Templates e Automação

```
class TemplateManager:
    def __init__(self):
        self.templates = {
        'dashboard': self.create_dashboard_template,
        'report': self.create_report_template,
        'analysis': self.create_analysis_template,
        'presentation': self.create_presentation_template
    }
}
  def create_dashboard_template(self, config):
    """Cria template de dashboard interativo"""
template = f"""
<!DOCTYPE html>
<html lang="pt-BR">
<head>
              cmeta cnarset="UTF-8">
cmeta name="viewport" content="width=device-width, initial-scale=1.0">
ctitle>{config.get('title', 'Dashboard Interativo')}</title>
cscript src="https://cdn.plot.ly/plotly-latest.min.js"></script>
clink href="https://cdn.jsdelivn.net/pm/tallwindsseg2.2.19/dist/tallwind.min.css" rel="stylesheet">
clink href="https://cdn.jsc.cloudflare.com/ajax/libs/font-awesome/6.0.0/css/all.min.css" rel="stylesheet">
ctylesheet">
ctylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettylesheettyl
                             .gradient-bg {{
background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
                          background
}}
.card-shadow {{
box-shadow: 0 10px 25px rgba(0,0,0,0.1);
                         trans_.
}}
,metric-card:hover {{
    transform: translateY(-5px);
}
                      transic.
}}
.chart-container {{
   background: white;
   border-radius: 12px;
   padding: 20px;
   margin: 10px 0;

                                      ...
'Al class="text-3xl font-bold">{config.get('title', 'Dashboard')}</hl>
class="text-blue-100 mt-2">{config.get('subtitle', 'Análise de dados em tempo real')}
               <div class="container mx-auto px-6">
```

O 6----

```
cp class="text-2xl font-bold text-gray-800" id="active-customers">0
                   </div>
                   </div>
         v class="grid gri0-cuis-1 how.gr.to ----
cdiv

<label class="block text-sm font-medium text-gray-700 mb-2">Período</label>
    <select id="date-filter" class="w-full p-2 border border-gray-300 rounded-mooth coption value="30d">\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u00f3\u
                                      </div
                           <div>
<div>
<label class="block text-sm font-medium text-gray-700 mb-2">Regiāo</label>
<select id="region-filter" class="w-full p-2 border border-gray-300 rounded-md">
<option value="all*>Todas</option>
<option value="north">Norte</option>
<option value="morth">Norte</option>
<option value="south">Sul</option>
<option value="south">Sul</option>
<option value="east">Leste</option>
<option value="west">Oeste</option>
</select>
</div>
                  <,
</div>
cscript>
    // Atualizar timestamp
    document.getElementById('last-update').textContent = new Date().toLocaleString('pt-BR');
         // Dados simulados para demonstração 
const sampleData = {{
    revenue: [1000, 1200, 1100, 1300, 1250, 1400, 1350],
    customers: [150, 160, 155, 170, 165, 180, 175],
    dates: ['2024-01-01', '2024-01-02', '2024-01-03', '2024-01-04', '2024-01-05', '2024-01-06', '2024-01-07'],
    categories: ['Eletrônicos', 'Roupas', 'Livros', 'Casa', 'Esportes'],
    categoryValues: [35, 25, 15, 15, 10]
}};
         }};
            // Atualizar KPIs
document.getElementById('total-revenue').textContent = 'R$ ' + sampleData.revenue.reduce((a,b) => a+b, 0).toLocale
document.getElementById('active-customers').textContent = Math.max(...sampleData.customers).toLocaleString('pt-BR'
document.getElementById('conversion-rate').textContent = '12.5%';
document.getElementById('growth-rate').textContent = '+8.3%';
                                                                                                                                                                                                                                                                                            eString('pt-BR');
         // Gráfico de tendência de receita
const revenueTrend = {{
    x: sampleData.dates,
    y: sampleData.revenue,
    type: 'scatter',
    mode: 'lines+marker',
    name: 'Receita',
    line: {{color: '#3882F6', width: 3}},
    marker: {{size: 8}}
}};
         }};
         Plotly.newPlot('revenue-trend', [revenueTrend], {{
    title: '',
                   title: '',
xaxis: {{title: 'Data'}},
yaxis: {{title: 'Receita (R$)'}},
margin: {{t: 30}}.
           // Gráfico de distribuição por categoria
         // Gráfico de distributao por categor
const categoryDist = {{
    values: sampleData.categoryValues,
    labels: sampleData.categories,
    type: 'pie',
    hole: 0.4,
    textinfo: 'label+percent',
                    textposition: 'outside'
         }};
         Plotly.newPlot('category-distribution', [categoryDist], {{
    title: '',
                          rgin: {{t: 30}}
         33):
          // Performance regional (gráfico de barras)
const regionalPerf = {{
```

```
x: ['Norte', 'Sul', 'Leste', 'Oeste', 'Centro'],
y: [850, 920, 780, 1100, 650],
type: 'bar',
marker: {{color: ['#EF4444', '#108981', '#F59E08', '#885CF6', '#0686D4']}}
            .Jutiy.newPlot('regional-performance
   title: '',
   xaxis: {{title: 'Regiāo'}},
   yaxis: {{title: 'Vendas (R$)'}},
   margin: {{t: 30}}
)));
              Plotly.newPlot('regional-performance', [regionalPerf], {{
             // Top Products = {{
    x: [450, 380, 320, 280, 220],
    y: ['Produto A', 'Produto B', 'Produto C', 'Produto D', 'Produto E'],
    type: 'bar',
    orientation: 'h',
    marker: {{color: '#3882F6'}}
              }};
              Plotly.newPlot('top-products', [topProducts], {{
    title: '',
                      title: '',
xaxis: {{title: 'Vendas'}},
margin: {{t: 30, 1: 100}}
              // Event listeners para filtros
document.getElementById('apply-filters').addEventListener('click', function() {{
    // Aqui você implementaria a lògica de filtragem
    alert('Filtros aplicados! (Funcionalidade de demonstração)');
...
       // Auto-refresh a cada 5 minutos
setInterval(function() {{
    document.getElementById('last-update').textContent = new Date().toLocaleString('pt-BR');
    // Agut voce atualizaria os dados reais
}), 3000000);

(/script>)
             return template
      def create_report_template(self, config):
    """Cria template de relatório executiv
 # {config.get('title', 'Relatório Executivo')}
 **Data**: {datetime.now().strftime('%d/%m/%Y')}
**Periodo**: {config.get('period', 'Último mês')}
**Autor**: {config.get('author', 'Sistema Automatizado')}
## Resumo Executivo
{config.get('executive_summary', 'Este relatório apresenta uma análise abrangente dos principais indicadores de performance.')}
### Principais Resultados
    - **Indicador 1**: {config.get('kpi1', 'Valor não informado')}
    - **Indicador 2**: {config.get('kpi2', 'Valor não informado')}
    - **Indicador 3**: {config.get('kpi3', 'Valor não informado')}
 ## Análise Detalhada
 ### Performance Geral
{config.get('performance_analysis', 'Análise de performance será inserida aqui.')}
 ### Tendências Identificadas
{config.get('trends', 'Tendências serão descritas nesta seção.')}
 ### Oportunidades e Riscos
{config.get('opportunities', 'Oportunidades e riscos serão listados aqui.')}
{config.get('recommendations', 'Recomendações estratégicas serão apresentadas nesta seção.')}
{config.get('next_steps', 'Próximos passos serão definidos aqui.')}
 *Relatório gerado automaticamente em {datetime.now().strftime('%d/%m/%Y às %H:%M')}*  
       def generate_from_template(self, template_type, config, output_path=None):
    """Gera conteddo a partir de um template""
    if template_type not in self-templates:
        raise Valuefrorof("Template '(template_type)' não encontrado")
             content = self.templates[template_type](config)
             if output_path:
    with open(output_path, 'w', encoding='utf-8') as f:
                    f.write(content)
print(f" 
    Template '{template_type}' gerado em: {output_path}")
               return content
```

o Exemplos de Uso Prático

Exemplo 1: Pipeline Completo de Análise

```
grtnon
# Criar dados de exemplo
sample_data = pd.DataFrame({
    'date': pd.date_range('2024-01-01', periods=100, freq='D'),
    'revenue': np.random.normal(1000, 200, 100) + np.sin(np.arange(100) * 0.1) * 100,
    'customers': np.random.ormal(1000, 200, 100) + np.sin(np.arange(100) * 0.1) * 100,
    'customers': np.random.ormice(['A', 'B', 'C'], 100),
    'region': np.random.choice(['North', 'South', 'East', 'West'], 100),
    'conversion_rate': np.random.beta(2, 8, 100)
})

# Executar pipeline completo
pipeline = DataAnalysisPipeline("exemplo_analise")
results = pipeline.run_complete_pipeline(sample_data, data_type='dataframe')

# Acessar resultados
print("Dados processados:", results['processed_data'].shape)
print("Visualizações criadas:", list(results['visualizations'].keys()))
```

Copiar

⊕ Copiar

Exemplo 2: Sistema de Forecasting

```
python

# Criar série temporal
dates = pd.date_range('2024-01-01', periods=365, freq='D')
values = 1000 + np.cumsum(np.random.normal(2, 10, 365)) + 50 * np.sin(np.arange(365) * 2 * np.pi / 7)
ts_data = pd.Series(values, index=dates)

# Executar previsão
forecaster = ForecastingSystem()
forecasts = Forecaster.ensemble_forecast(ts_data, periods_ahead=30)

# Criar visualização
fig = forecaster.create_forecast_visualization(ts_data, forecasts)
fig.show()

# Gerar relatório
report = forecaster.generate_forecast_report("Vendas Diárias", ts_data, forecasts)
print(report)
```

python (C Copiar

```
# Configurar experimento
ab_tester = ABTestAnalyzer()
ab_tester.create_experiment('exp_001', {
    'name': 'Teste de Landing Page',
    'hypothesis': 'Nova landing page aumentará conversão em 15%',
    'metric': 'conversion_rate',
    'sample_size': 2000,
    'confidence_level': 0.95
})

# Analisar resultados
results = ab_tester.analyze_experiment('exp_001')

# Criar visualização
fig = ab_tester.create_results_visualization('exp_001')

# Gerar relatório
report = ab_tester.generate_experiment_report('exp_001')
print(report)
```

Conclusão

Este guia completo apresenta todas as ferramentas, bibliotecas e funcionalidades disponíveis no ambiente de desenvolvimento. Com essa base sólida, você pode:

- 1. **Desenvolver aplicações completas** usando Python, JavaScript, HTML/CSS
- 2. **Criar visualizações interativas** com Plotly e outras bibliotecas
- 3. Implementar análises estatísticas avançadas
- 4. Construir dashboards profissionais
- 5. Automatizar workflows de análise de dados
- 6. **Gerar relatórios** executivos automatizados

Próximos Passos Recomendados:

- 1. Escolha um projeto específico para implementar
- 2. **Defina os requisitos** e objetivos
- 3. Selecione as ferramentas apropriadas deste guia
- 4. Implemente iterativamente usando os templates fornecidos
- 5. **Teste e refine** conforme necessário

Estou pronto para ajudar você a implementar qualquer um desses sistemas ou criar soluções personalizadas baseadas nessas ferramentas!