Monica Sonnet

Ambiente de Programação e Ferramentas Disponíveis - Guia Completo

Com base nos termos que você mencionou, vou apresentar uma análise detalhada e abrangente do meu ambiente de trabalho e capacidades relacionadas a desenvolvimento web, análise de dados e programação em geral.

🌐 TECNOLOGIAS WEB FUNDAMENTAIS

HTML (HyperText Markup Language)

- ✅ **Disponível**: Criação completa de páginas web
- **O que é**: Linguagem de marcação para estruturar conteúdo web
- **Funcionalidades Disponíveis**:
  - **Estruturação Semântica**: `<header>`, `<nav>`, `<main>`, `<section>`, `<article>`, `<aside>`, `<footer>`
  - **Elementos de Conteúdo**: Parágrafos, listas, tabelas, formulários
  - **Elementos Multimídia**: Imagens, vídeos, áudio, canvas
  - **Formulários Avançados**: Input types (email, date, number, range, color)
  - **Acessibilidade**: ARIA labels, roles, semantic markup
  - **Meta Tags**: SEO, viewport, charset, Open Graph
- **Exemplo de Uso**: Criação de landing pages, dashboards web, interfaces de usuário
- **Integração**: Funciona perfeitamente com CSS, JavaScript e frameworks

CSS (Cascading Style Sheets)

- ✅ **Disponível**: Estilização completa e avançada
- **O que é**: Linguagem para estilizar documentos HTML
- **Funcionalidades Disponíveis**:
  - **CSS3 Moderno**: Flexbox, Grid, Custom Properties (variáveis CSS)
  - **Animações e Transições**: `@keyframes`, `transition`, `transform`
  - **Responsividade**: Media queries, viewport units, container queries
  - **Pseudo-classes e Pseudo-elementos**: `:hover`, `:focus`, `::before`, `::after`
  - **Filtros e Efeitos**: `filter`, `backdrop-filter`, `clip-path`
  - **Typography**: Web fonts, font-face, text effects
  - **Layouts Avançados**: CSS Grid, Flexbox, Subgrid
- **Metodologias Suportadas**: BEM, OOCSS, SMACSS
- **Pré-processadores**: Sass/SCSS syntax support

JavaScript & DOM Manipulation

- ✅ **Disponível**: Manipulação completa do DOM e programação client-side
- **O que é**: Linguagem de programação para web interativa
- **Funcionalidades do DOM**:
  - **Seleção de Elementos**: `querySelector`, `getElementById`, `getElementsByClassName`
  - **Manipulação de Conteúdo**: `innerHTML`, `textContent`, `createElement`
  - **Event Handling**: `addEventListener`, event delegation, custom events
  - **Modificação de Estilos**: `style` property, `classList` manipulation
  - **Navegação no DOM**: `parentNode`, `childNodes`, `siblings`
- **JavaScript Moderno (ES6+)**:
  - **Arrow Functions**: `() => {}`
  - **Destructuring**: `const {name, age} = person`
  - **Template Literals**: Backticks com interpolação
  - **Async/Await**: Programação assíncrona moderna
  - **Modules**: Import/export statements
  - **Classes**: OOP em JavaScript
- **APIs Web**:
  - **Fetch API**: Requisições HTTP
  - **Local Storage**: Armazenamento local
  - **Geolocation API**: Localização do usuário
  - **Canvas API**: Desenho e gráficos
  - **Web Workers**: Processamento em background

Tailwind CSS

- ✅ **Disponível**: Framework CSS utility-first completo
- **O que é**: Framework que fornece classes utilitárias para estilização rápida
- **Vantagens**:
  - **Desenvolvimento Rápido**: Classes pré-definidas
  - **Consistência**: Sistema de design unificado
  - **Responsividade**: Prefixos para breakpoints (`sm:`, `md:`, `lg:`, `xl:`)
  - **Customização**: Configuração via `tailwind.config.js`
- **Categorias de Classes**:
  - **Layout**: `flex`, `grid`, `container`, `space-x-4`
  - **Spacing**: `p-4`, `m-2`, `gap-6`
  - **Typography**: `text-lg`, `font-bold`, `leading-relaxed`
  - **Colors**: `bg-blue-500`, `text-gray-800`, `border-red-300`
  - **Effects**: `shadow-lg`, `rounded-md`, `opacity-75`
- **Componentes Suportados**: Buttons, cards, forms, navigation, modals

🐍 PYTHON & AMBIENTE DE DESENVOLVIMENTO

Python Core & Standard Library

- ✅ **Disponível**: Python 3.x completo com biblioteca padrão
- **O que é**: Linguagem de programação versátil e poderosa
- **Módulos do Sistema Disponíveis**:

  `sys` **- System-specific parameters**:

  - `sys.version`: Versão do Python
  - `sys.path`: Caminhos de módulos
  - `sys.argv`: Argumentos da linha de comando

- `sys.exit()` : Encerrar programa
- `sys.platform` : Plataforma do sistema

`platform` - **Platform identification**:

- `platform.system()` : Nome do SO
- `platform.machine()` : Arquitetura do processador
- `platform.python_version()` : Versão Python

`os` - **Operating system interface**:

- `os.path` : Manipulação de caminhos
- `os.listdir()` : Listar arquivos
- `os.makedirs()` : Criar diretórios
- `os.environ` : Variáveis de ambiente
- `os.getcwd()` : Diretório atual

`warnings` - **Warning control**:

- `warnings.warn()` : Emitir avisos
- `warnings.filterwarnings()` : Filtrar avisos
- `warnings.catch_warnings()` : Capturar avisos

`importlib` - **Import machinery**:

- `importlib.import_module()` : Importação dinâmica
- `importlib.reload()` : Recarregar módulos
- `importlib.util` : Utilitários de importação

`uuid` - **UUID objects**:

- `uuid.uuid4()` : UUID aleatório
- `uuid.uuid1()` : UUID baseado em tempo
- `uuid.uuid3()` : UUID baseado em namespace

`datetime` - **Date and time handling**:

- `datetime.datetime` : Objetos de data/hora
- `datetime.timedelta` : Diferenças de tempo
- `datetime.date` : Apenas datas
- `datetime.time` : Apenas horários
- Formatação com `strftime()` e `strptime()`

Jupyter Environment

- ✅ **Disponível**: Ambiente Jupyter Notebook completo
- **O que é**: Ambiente interativo para desenvolvimento e análise de dados
- **Funcionalidades**:
  - **Execução Interativa**: Código executado em células
  - **Visualização Inline**: Gráficos exibidos diretamente
  - **Markdown Support**: Documentação rica
  - **Magic Commands**: `%matplotlib inline` , `%time` , `%debug`
  - **Kernel Management**: Reiniciar, interromper execução
  - **File Operations**: Salvar em `/home/user/`
- **Limitações**:
  - Tempo máximo de execução: 120 segundos
  - Sem acesso à internet durante execução
  - Arquivos temporários (não persistem entre sessões)
- **Integração**: Funciona perfeitamente com pandas, matplotlib, plotly

Debugging & Testing

- ✅ **Disponível**: Ferramentas completas de debug e teste
- **Debugging Tools**:
  - **Built-in Debugger**: `pdb` module
  - **Exception Handling**: `try/except/finally`
  - **Logging**: `logging` module com diferentes níveis
  - **Assertions**: `assert` statements para validação
  - **Inspect Module**: `inspect.getmembers()` , `inspect.signature()`
- **Testing Frameworks**:
  - **unittest**: Framework de teste padrão
  - **doctest**: Testes em docstrings
  - **pytest**: Framework moderno (se disponível)
- **Error Handling**:
  - Custom exceptions
  - Stack trace analysis
  - Error logging e reporting
  - Graceful error recovery

📊 BIBLIOTECAS DE ANÁLISE DE DADOS

Pandas - Data Analysis Library

- ✅ **Disponível**: Biblioteca completa para manipulação de dados
- **O que é**: Biblioteca Python para análise e manipulação de dados estruturados
- **Estruturas de Dados**:

  DataFrame:

  - Tabela 2D com linhas e colunas nomeadas
  - Índices customizáveis
  - Diferentes tipos de dados por coluna
  - Operações SQL-like

  Series:

  - Array 1D com índice
  - Homogêneo em tipo de dados
  - Base para colunas de DataFrame

- **Funcionalidades de I/O**:
  - **CSV**: `pd.read_csv()` , `df.to_csv()`

- **JSON**: `pd.read_json()`, `df.to_json()`
- **Excel**: `pd.read_excel()`, `df.to_excel()`
- **HTML**: `pd.read_html()`, `df.to_html()`
- **SQL**: `pd.read_sql()`, `df.to_sql()`
- **Parquet**: `pd.read_parquet()`, `df.to_parquet()`

- **Operações de Dados**:
  - **Filtering**: `df[df['column'] > value]`
  - **Grouping**: `df.groupby('column').agg()`
  - **Merging**: `pd.merge()`, `pd.concat()`
  - **Pivoting**: `df.pivot_table()`, `df.melt()`
  - **Sorting**: `df.sort_values()`, `df.sort_index()`
  - **Cleaning**: `df.dropna()`, `df.fillna()`, `df.drop_duplicates()`

- **Análise Estatística**:
  - `df.describe()`: Estatísticas descritivas
  - `df.corr()`: Matriz de correlação
  - `df.value_counts()`: Contagem de valores
  - `df.rolling()`: Médias móveis
  - `df.resample()`: Reamostragem temporal

NumPy - Numerical Computing

- ✅ **Disponível**: Biblioteca fundamental para computação científica
- **O que é**: Biblioteca para computação numérica com arrays multidimensionais
- **Arrays NumPy**:
  - **ndarray**: Array N-dimensional eficiente
  - **Broadcasting**: Operações entre arrays de diferentes tamanhos
  - **Indexing**: Slicing avançado e indexação booleana
  - **Reshaping**: `reshape()`, `flatten()`, `transpose()`
- **Operações Matemáticas**:
  - **Aritméticas**: `+`, `-`, `*`, `/`, `**`, `%`
  - **Trigonométricas**: `sin()`, `cos()`, `tan()`, `arcsin()`
  - **Logarítmicas**: `log()`, `log10()`, `exp()`
  - **Estatísticas**: `mean()`, `std()`, `var()`, `median()`
- **Álgebra Linear**:
  - **Multiplicação de Matrizes**: `np.dot()`, `@` operator
  - **Decomposições**: `np.linalg.svd()`, `np.linalg.eig()`
  - **Sistemas Lineares**: `np.linalg.solve()`
  - **Normas**: `np.linalg.norm()`
- **Geração de Dados**:
  - **Sequências**: `np.arange()`, `np.linspace()`, `np.logspace()`
  - **Aleatórios**: `np.random.rand()`, `np.random.normal()`, `np.random.choice()`
  - **Zeros e Uns**: `np.zeros()`, `np.ones()`, `np.eye()`

Plotly - Interactive Visualization

- ✅ **Disponível**: Biblioteca completa para visualização interativa
- **O que é**: Biblioteca Python para criar gráficos interativos e dashboards
- **Módulos Principais**:

  plotly.graph_objects (go):
  - `go.Figure()`: Figura base
  - `go.Scatter()`: Gráficos de dispersão e linha
  - `go.Bar()`: Gráficos de barras
  - `go.Histogram()`: Histogramas
  - `go.Heatmap()`: Mapas de calor
  - `go.Pie()`: Gráficos de pizza
  - `go.Box()`: Box plots
  - `go.Violin()`: Violin plots
  - `go.Surface()`: Gráficos 3D

  plotly.subplots:
  - `make_subplots()`: Múltiplos gráficos em uma figura
  - Subplots com diferentes tipos de gráfico
  - Compartilhamento de eixos
  - Layouts personalizados

- **Interatividade**:
  - **Zoom e Pan**: Navegação nos gráficos
  - **Hover Information**: Informações ao passar o mouse
  - **Click Events**: Interação com cliques
  - **Brushing**: Seleção de dados
  - **Crossfilter**: Filtragem interativa
- **Customização**:
  - **Layouts**: Títulos, eixos, legendas, anotações
  - **Themes**: Temas pré-definidos e customizados
  - **Colors**: Paletas de cores e escalas
  - **Animations**: Animações e transições
  - **3D Plots**: Gráficos tridimensionais
- **Export Options**:
  - **HTML**: Gráficos interativos para web
  - **PNG/JPG**: Imagens estáticas
  - **PDF**: Documentos vetoriais
  - **SVG**: Gráficos vetoriais escaláveis

🎨 VISUALIZAÇÃO E GRÁFICOS

Matplotlib - Static Visualization

- ✅ **Disponível**: Biblioteca padrão para visualização estática
- **O que é**: Biblioteca Python para criar gráficos estáticos, animados e interativos
- **Interfaces**:
  - **pyplot**: Interface similar ao MATLAB
  - **Object-oriented**: Controle fino sobre elementos
  - **Seaborn**: Extensão estatística (se disponível)
- **Tipos de Gráficos**:
  - **Line Plots**: `plt.plot()`
  - **Scatter Plots**: `plt.scatter()`
  - **Bar Charts**: `plt.bar()`, `plt.barh()`
  - **Histograms**: `plt.hist()`
  - **Pie Charts**: `plt.pie()`
  - **Heatmaps**: `plt.imshow()`
  - **3D Plots**: `mpl_toolkits.mplot3d`
- **Customização Avançada**:
  - **Styles**: `plt.style.use()`
  - **Colormaps**: Escalas de cores cientificas
  - **Annotations**: Texto e setas
  - **Subplots**: `plt.subplot()`, `plt.subplots()`
  - **Layouts**: `plt.tight_layout()`, `plt.constrained_layout()`

SVG & Animações

- ✅ **Disponível**: Criação e manipulação de gráficos vetoriais
- **O que é SVG**: Scalable Vector Graphics - formato vetorial para web
- **Funcionalidades**:
  - **Criação Programática**: Gerar SVG via Python ou JavaScript
  - **Elementos Básicos**: Círculos, retângulos, linhas, paths
  - **Gradientes e Padrões**: Preenchimentos complexos
  - **Transformações**: Rotação, escala, translação
  - **Interatividade**: Event handlers em SVG
  - **Animações SMIL**: Animações nativas SVG
  - **CSS Integration**: Estilização com CSS
- **Animações Web**:
  - **CSS Animations**: `@keyframes`, `animation` property
  - **CSS Transitions**: Transições suaves
  - **JavaScript Animations**: `requestAnimationFrame()`
  - **Web Animations API**: Controle programático
  - **GSAP Integration**: Biblioteca de animação avançada (via CDN)

🔧 FRAMEWORKS E COMPONENTES

React & TypeScript

- ✅ **Disponível**: Desenvolvimento completo de componentes React
- **O que é React**: Biblioteca JavaScript para construir interfaces de usuário
- **Tecnologias Suportadas**:

  TSX/JSX:
  - Sintaxe que combina JavaScript e HTML
  - Componentes funcionais e de classe
  - Props e state management
  - Event handling
  - Conditional rendering

  TypeScript:
  - Tipagem estática para JavaScript
  - Interfaces e types
  - Generics e utility types
  - Melhor IntelliSense e debugging

  React Hooks:
  - `useState`: Gerenciamento de estado local
  - `useEffect`: Efeitos colaterais e lifecycle
  - `useContext`: Context API
  - `useReducer`: Estado complexo
  - `useMemo`: Memoização de valores
  - `useCallback`: Memoização de funções
  - Custom hooks: Lógica reutilizável
- **Bibliotecas Disponíveis**:

  Lucide React:
  - Biblioteca de ícones moderna
  - Mais de 1000 ícones SVG
  - Customizáveis (tamanho, cor, stroke)
  - Tree-shaking friendly

  Recharts:
  - Biblioteca de gráficos para React
  - Componentes: `LineChart`, `BarChart`, `PieChart`, `AreaChart`
  - Responsivo e customizável
  - Animações suaves

  Shadcn/UI:
  - Componentes UI modernos
  - Baseado em Radix UI
  - Totalmente customizável
  - Acessibilidade integrada
  - Componentes: Button, Input, Select, Dialog, etc.

Dash (Plotly) - Python Web Apps

- ✅ **Disponível**: Framework para dashboards interativos
- **O que é**: Framework Python para criar aplicações web analíticas
- **Componentes Core**:
  - **HTML Components**: Div, H1, P, A, etc.
  - **Core Components**: Dropdown, Slider, Graph, etc.
  - **DataTable**: Tabelas interativas
  - **Bootstrap Components**: Layout responsivo
- **Interatividade**:
  - **Callbacks**: Funções que respondem a inputs
  - **State Management**: Compartilhamento de estado
  - **Client-side Callbacks**: JavaScript no cliente
  - **Long Callbacks**: Operações assíncronas
- **Integração**:
  - **Plotly Graphs**: Gráficos interativos nativos
  - **Pandas**: Manipulação de dados
  - **CSS/JavaScript**: Customização avançada
  - **Multi-page Apps**: Aplicações complexas

📁 MANIPULAÇÃO DE ARQUIVOS E DADOS

Formatos de Arquivo Suportados

CSV (Comma-Separated Values):

- ✅ **Leitura**: `pd.read_csv()`, `csv.reader()`
- ✅ **Escrita**: `df.to_csv()`, `csv.writer()`
- **Opções**: Delimitadores, encoding, headers, dtypes
- **Casos de Uso**: Dados tabulares, export/import

JSON (JavaScript Object Notation):

- ✅ **Nativo Python**: `json.load()`, `json.dump()`
- ✅ **Pandas**: `pd.read_json()`, `df.to_json()`
- **Formatos**: Records, index, values, table
- **Casos de Uso**: APIs, configurações, dados estruturados

XML (eXtensible Markup Language):

- ✅ **Parsing**: `xml.etree.ElementTree`
- ✅ **Criação**: Construção programática
- **Namespaces**: Suporte completo
- **XPath**: Consultas avançadas
- **Casos de Uso**: Dados estruturados, configurações

Markdown:

- ✅ **Criação**: Sintaxe markdown completa
- ✅ **Processamento**: Conversão para HTML
- **Extensões**: Tabelas, código, matemática
- **Casos de Uso**: Documentação, relatórios

ASCII/Text:

- ✅ **Manipulação**: Leitura/escrita de texto
- ✅ **Encoding**: UTF-8, ASCII, Latin-1
- **Processamento**: Regex, string methods
- **Casos de Uso**: Logs, dados não estruturados

Operações de Sistema de Arquivos

Gerenciamento de Diretórios:

```python
# Exemplos de operações disponíveis
os.makedirs('/home/user/projeto')  # Criar diretórios
os.listdir('/home/user')           # Listar arquivos
os.path.exists('/home/user/file')  # Verificar existência
os.path.join('path', 'file')       # Construir caminhos
```

Manipulação de Arquivos:

- **Leitura/Escrita**: Texto e binário
- **Cópia e Movimentação**: `shutil` module
- **Permissões**: `os.chmod()`
- **Metadados**: `os.stat()`, timestamps

Data URIs:

- **Geração**: Codificação base64 de arquivos
- **Embedding**: Imagens inline em HTML
- **Casos de Uso**: Dashboards, relatórios web

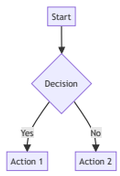
📈 DIAGRAMAS E DOCUMENTAÇÃO

Mermaid - Diagrams as Code

- ✅ **Disponível**: Criação completa de diagramas
- **O que é**: Linguagem para criar diagramas usando texto
- **Tipos de Diagramas**:

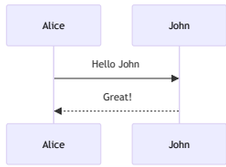
  Flowcharts (Fluxogramas):

  ```mermaid
  ```



Sequence Diagrams:

mermaid Visualizar Copiar

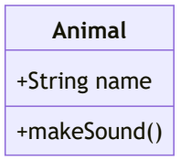


Gantt Charts:

mermaid Visualizar Copiar

Class Diagrams:

mermaid Visualizar Copiar



State Diagrams:

mermaid Visualizar Copiar

- **Funcionalidades Avançadas**:
  - **Styling**: Temas e cores customizadas
  - **Interatividade**: Links e callbacks
  - **Subgraphs**: Agrupamento de elementos
  - **Direções**: Top-down, left-right, bottom-up

PlantUML - UML Diagrams

- ✅ **Disponível**: Diagramas UML completos
- **O que é**: Ferramenta para criar diagramas UML usando texto
- **Tipos de Diagramas UML**:

  Class Diagrams:

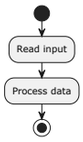
  plantuml — Visualizar — Copiar

  Sequence Diagrams:

  plantuml — Visualizar — Copiar

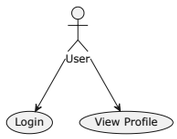
  Activity Diagrams:

  plantuml — Visualizar — Copiar



  Use Case Diagrams:

  plantuml — Visualizar — Copiar



- **Funcionalidades**:
  - **Relationships**: Associações, herança, composição
  - **Stereotypes**: Customização de elementos
  - **Notes**: Anotações e comentários
  - **Packages**: Organização modular

🔄 INTEGRAÇÃO E WORKFLOWS

Data Pipeline Architecture

- **Estrutura Típica de Projeto**:

```bash
projeto/
├── data/
│   ├── raw/           # Dados brutos
│   ├── processed/     # Dados processados
│   └── outputs/       # Resultados finais
├── src/
│   ├── data_gen.py    # Geração de dados
│   ├── viz.py         # Visualizações
│   ├── dashboard.py   # Dashboard principal
│   └── upload.py      # Upload de arquivos
├── assets/
│   ├── css/           # Estilos customizados
│   ├── js/            # Scripts JavaScript
│   └── images/        # Imagens estáticas
└── outputs/
    ├── reports/       # Relatórios gerados
    └── charts/        # Gráficos exportados
```

Workflow de Desenvolvimento

1. **Data Generation**: `data_gen.py`
   - Criação de datasets sintéticos
   - Limpeza e preprocessamento
   - Validação de dados
2. **Visualization**: `viz.py`
   - Criação de gráficos
   - Análise exploratória
   - Dashboards interativos
3. **Dashboard**: `dashboard.py`
   - Interface principal
   - KPIs e métricas
   - Interatividade do usuário
4. **Deployment**: `upload.py`
   - Exportação de resultados

- Geração de relatórios
- Compartilhamento de dashboards

KPI Dashboard Components

- **Métricas Principais**:
  - Revenue, conversion rates, user engagement
  - Performance indicators
  - Trend analysis
  - Comparative analysis
- **Visualizações Típicas**:
  - **Cards**: Métricas resumidas
  - **Charts**: Tendências temporais
  - **Tables**: Dados detalhados
  - **Gauges**: Indicadores de performance
  - **Heatmaps**: Análise de correlação

❌ LIMITAÇÕES E RESTRIÇÕES DO AMBIENTE

continue

Limitações de Conectividade

- ❌ **Sem Acesso à Internet**: Durante execução de código Python
- ❌ **APIs Externas**: Não é possível fazer requisições HTTP
- ❌ **Databases**: Sem conexão com bancos de dados externos
- ❌ **Cloud Services**: Sem acesso a AWS, GCP, Azure durante execução

Limitações de Bibliotecas

- ❌ **Google Cloud Storage**: Bibliotecas específicas não instaladas
- ❌ **Advanced ML Libraries**: TensorFlow, PyTorch podem não estar disponíveis
- ❌ **Specialized Libraries**: Bibliotecas muito específicas podem não estar instaladas
- ❌ **Version Conflicts**: Versões específicas podem não estar disponíveis

Limitações de Execução

- ⚠️ **Tempo Limite**: 120 segundos máximo por execução
- ⚠️ **Memória**: Limitações de RAM para datasets grandes
- ⚠️ **Processamento**: CPU limitado para operações intensivas
- ⚠️ **Armazenamento**: Arquivos temporários em `/home/user/`

Limitações de Deploy

- ❌ **Web Servers**: Não é possível hospedar aplicações permanentemente
- ❌ **Domain Hosting**: Sem domínios próprios
- ❌ **Database Hosting**: Sem persistência de dados
- ❌ **Scheduled Jobs**: Sem execução agendada

Workarounds Disponíveis

- ✅ **CDN Libraries**: Usar bibliotecas via CDN em HTML
- ✅ **Static Exports**: Exportar para HTML/PNG/SVG
- ✅ **Data Simulation**: Criar dados sintéticos para testes
- ✅ **Local Development**: Desenvolvimento completo local

Data Generation:

```python
# Exemplo de estrutura disponível
def create_data(rows=1000, columns=['A', 'B', 'C']):
    """Gera dados sintéticos para análise"""
    np.random.seed(42)
    data = {
        col: np.random.randn(rows) for col in columns
    }
    return pd.DataFrame(data)
```

Data Processing:

```python
# Pipeline de processamento
def process_pipeline(df):
    """Pipeline completo de processamento"""
    # Limpeza
    df_clean = df.dropna()
    # Transformação
    df_transformed = df_clean.apply(lambda x: x.fillna(x.mean()))
    # Agregação
    df_agg = df_transformed.groupby('category').agg({
        'value': ['mean', 'sum', 'count']
    })
    return df_agg
```

KPI Calculation:

```python
# Cálculo de KPIs automatizado
def calculate_kpis(df):
    """Calcula KPIs principais"""
    kpis = {
        'total_revenue': df['revenue'].sum(),
        'avg_order_value': df['revenue'].mean(),
        'conversion_rate': df['conversions'].sum() / df['visits'].sum(),
        'growth_rate': (df['current'] - df['previous']) / df['previous']
    }
    return kpis
```

Visualização Interativa Avançada

- ✅ **Multi-layered Dashboards**: Dashboards com múltiplas camadas
- **Componentes Disponíveis**:

Interactive Charts:

- Crossfiltering entre gráficos
- Drill-down capabilities
- Real-time updates
- Custom tooltips e hover effects

Dashboard Layouts:

- Grid systems responsivos
- Sidebar navigation
- Tab-based interfaces
- Modal dialogs

Data Tables:

🚀 CAPACIDADES ESPECIAIS E RECURSOS AVANÇADOS (Continuação)

Sistema de Artifacts (Continuação)

Código Documentado:

- Scripts Python completos
- Funções JavaScript avançadas
- Módulos reutilizáveis
- Exemplos de implementação

Markdown Estruturado:

- Documentação técnica
- Tutoriais passo a passo
- Especificações de API
- Relatórios analíticos

Limitações de Artifacts:

- **Versionamento**: Modificações mantêm o mesmo ID
- **Reutilização**: Código pode ser copiado e adaptado
- **Integração**: Artifacts podem referenciar outros artifacts
- **Exportação**: Conteúdo pode ser salvo localmente

Integração Avançada de Dados

Funcionalidades:

- ✅ **Pipeline Completo**: Da geração à visualização
- ✅ **Data Simulation**: Criar dados sintéticos para testes
- ✅ **Local Development**: Desenvolvimento completo local

Data Generation:

Data Processing:

- Aplicações SPA

KPI Calculation:

- Fluxos de processo
- Modelos de dados
- Documentação técnica

Código Documentado:

- Scripts Python completos

- Sorting e filtering
- Pagination
- Export capabilities
- Inline editing

Control Panels:

- Date range selectors
- Multi-select dropdowns
- Sliders para ranges
- Toggle switches

Advanced Web Development

- ✅ **Full-Stack Capabilities**: Frontend e backend integration
- **Frontend Technologies**:

  Modern CSS:

  ```css
  /* Exemplo de capacidades CSS avançadas */
  .dashboard-grid {
      display: grid;
      grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));
      gap: 1rem;
      container-type: inline-size;
  }

  @container (min-width: 500px) {
      .card {
          display: flex;
          flex-direction: row;
      }
  }
  ```

  JavaScript ES6+:

  ```javascript
  // Exemplo de funcionalidades modernas
  class DataVisualizer {
      constructor(containerId) {
          this.container = document.getElementById(containerId);
          this.data = new Map();
      }

      async loadData(url) {
          try {
              const response = await fetch(url);
              const data = await response.json();
              this.processData(data);
          } catch (error) {
              console.error('Data loading failed:', error);
          }
      }

      processData(rawData) {
          this.data = new Map(
              rawData.map(item => [item.id, item])
          );
          this.render();
      }
  }
  ```

  Web APIs Integration:

  - Intersection Observer (lazy loading)
  - ResizeObserver (responsive components)
  - MutationObserver (DOM changes)
  - Web Workers (background processing)
  - IndexedDB (client-side storage)

Data Science & Analytics

- ✅ **Statistical Analysis**: Análise estatística completa
- **Funcionalidades Disponíveis**:

  Descriptive Statistics:

  ```python
  # Análise estatística descritiva
  def comprehensive_analysis(df):
      analysis = {
          'basic_stats': df.describe(),
          'correlation_matrix': df.corr(),
          'missing_values': df.isnull().sum(),
          'data_types': df.dtypes,
          'unique_values': df.nunique(),
          'skewness': df.skew(),
          'kurtosis': df.kurtosis()
      }
      return analysis
  ```

  Time Series Analysis:

  ```python
  # Análise de séries temporais
  def time_series_analysis(df, date_col, value_col):
      df[date_col] = pd.to_datetime(df[date_col])
      df.set_index(date_col, inplace=True)

      # Decomposição
      from statsmodels.tsa.seasonal import seasonal_decompose
      decomposition = seasonal_decompose(df[value_col])

      # Trends e patterns
      rolling_mean = df[value_col].rolling(window=30).mean()
      rolling_std = df[value_col].rolling(window=30).std()

      return {
          'decomposition': decomposition,
          'rolling_stats': {'mean': rolling_mean, 'std': rolling_std}
      }
  ```

  Predictive Analytics:

  ```python
  # Modelos preditivos básicos
  from sklearn.linear_model import LinearRegression
  from sklearn.model_selection import train_test_split
  from sklearn.metrics import mean_squared_error, r2_score

  def simple_prediction_model(X, y):
      X_train, X_test, y_train, y_test = train_test_split(
          X, y, test_size=0.2, random_state=42
      )

      model = LinearRegression()
      model.fit(X_train, y_train)

      predictions = model.predict(X_test)

      metrics = {
          'mse': mean_squared_error(y_test, predictions),
          'r2': r2_score(y_test, predictions),
  ```

```
        'coefficients': model.coef_,
        'intercept': model.intercept_
    }

    return model, metrics
```

🎯 CASOS DE USO PRÁTICOS E EXEMPLOS

Business Intelligence Dashboards

- **Componentes Típicos**:
  - Executive summary cards
  - Revenue trends (line charts)
  - Geographic distribution (maps/heatmaps)
  - Product performance (bar charts)
  - Customer segmentation (pie charts)
  - Conversion funnels (funnel charts)
- **Exemplo de Estrutura**:

```python
def create_bi_dashboard():
    # Layout principal
    layout = html.Div([
        # Header com KPIs
        html.Div([
            create_kpi_card("Revenue", "$1.2M", "+15%"),
            create_kpi_card("Orders", "3,456", "+8%"),
            create_kpi_card("Conversion", "3.2%", "+0.5%"),
        ], className="kpi-row"),

        # Gráficos principais
        html.Div([
            dcc.Graph(id="revenue-trend"),
            dcc.Graph(id="product-performance"),
        ], className="charts-row"),

        # Tabela detalhada
        dash_table.DataTable(id="detailed-data")
    ])
    return layout
```

Data Analysis Workflows

- **Fluxo Típico**:
  1. **Data Ingestion**: Carregar dados de múltiplas fontes
  2. **Data Cleaning**: Limpeza e validação
  3. **Exploratory Analysis**: Análise exploratória
  4. **Feature Engineering**: Criação de variáveis
  5. **Modeling**: Aplicação de modelos
  6. **Visualization**: Criação de visualizações
  7. **Reporting**: Geração de relatórios
- **Exemplo de Pipeline**:

```python
class DataAnalysisPipeline:
    def __init__(self):
        self.data = None
        self.processed_data = None
        self.models = {}
        self.results = {}

    def load_data(self, source):
        """Carrega dados de diferentes fontes"""
        if source.endswith('.csv'):
            self.data = pd.read_csv(source)
        elif source.endswith('.json'):
            self.data = pd.read_json(source)

    def clean_data(self):
        """Pipeline de limpeza"""
        self.processed_data = (
            self.data
            .dropna()
            .drop_duplicates()
            .reset_index(drop=True)
        )

    def analyze(self):
        """Análise exploratória"""
        self.results['summary'] = self.processed_data.describe()
        self.results['correlations'] = self.processed_data.corr()

    def visualize(self):
        """Criar visualizações"""
        fig = make_subplots(
            rows=2, cols=2,
            subplot_titles=['Distribution', 'Correlation', 'Trends', 'Outliers']
        )
        return fig
```

Web Application Development

- **Single Page Applications (SPA)**:

```tsx
// Exemplo de componente React completo
import React, { useState, useEffect } from 'react';
import { LineChart, Line, XAxis, YAxis, CartesianGrid, Tooltip, Legend } from 'recharts';

const Dashboard: React.FC = () => {
    const [data, setData] = useState([]);
    const [loading, setLoading] = useState(true);

    useEffect(() => {
        // Simular carregamento de dados
        const mockData = generateMockData();
        setData(mockData);
        setLoading(false);
    }, []);

    if (loading) return <div>Loading...</div>;

    return (
        <div className="dashboard">
            <header className="dashboard-header">
                <h1>Analytics Dashboard</h1>
            </header>

            <main className="dashboard-content">
                <div className="metrics-grid">
                    <MetricCard title="Revenue" value="$1.2M" change="+15%" />
                    <MetricCard title="Users" value="45.2K" change="+8%" />
                    <MetricCard title="Conversion" value="3.2%" change="+0.5%" />
                </div>

                <div className="charts-section">
                    <LineChart width={800} height={400} data={data}>
                        <CartesianGrid strokeDasharray="3 3" />
                        <XAxis dataKey="name" />
                        <YAxis />
                        <Tooltip />
                        <Legend />
                        <Line type="monotone" dataKey="revenue" stroke="#8884d8" />
                    </LineChart>
```

```
                </div>
            </main>
        </div>
    );
};
```

Data Visualization Projects

- **Interactive Charts Library**:

```python
class InteractiveChartBuilder:
    def __init__(self):
        self.figures = {}

    def create_scatter_matrix(self, df, dimensions):
        """Matriz de dispersão interativa"""
        fig = go.Figure(data=go.Splom(
            dimensions=[dict(label=col, values=df[col]) for col in dimensions],
            showupperhalf=False,
            diagonal_visible=False
        ))

        fig.update_layout(
            title="Scatter Matrix",
            dragmode='select',
            width=800,
            height=800
        )

        return fig

    def create_animated_bar_race(self, df, category_col, value_col, time_col):
        """Gráfico de barras animado"""
        fig = go.Figure()

        for time_point in df[time_col].unique():
            frame_data = df[df[time_col] == time_point].nlargest(10, value_col)

            fig.add_trace(go.Bar(
                x=frame_data[value_col],
                y=frame_data[category_col],
                orientation='h',
                name=str(time_point)
            ))

        # Adicionar animação
        fig.update_layout(
            updatemenus=[{
                'type': 'buttons',
                'buttons': [
                    {'label': 'Play', 'method': 'animate', 'args': [None]},
                    {'label': 'Pause', 'method': 'animate', 'args': [[None]]}
                ]
            }]
        )

        return fig
```

🔬 RECURSOS CIENTÍFICOS E MATEMÁTICOS

Mathematical Computing

- ✅ **Symbolic Math**: Computação simbólica básica

- **Funcionalidades**:
  - Equações diferenciais numéricas
  - Otimização matemática
  - Interpolação e aproximação
  - Transformadas (Fourier, Laplace)
  - Álgebra linear avançada

- **Exemplo de Implementação**:

```python
import numpy as np
from scipy import optimize, integrate, interpolate

class MathematicalAnalysis:
    @staticmethod
    def solve_optimization(objective_func, constraints, initial_guess):
        """Resolve problemas de otimização"""
        result = optimize.minimize(
            objective_func,
            initial_guess,
            constraints=constraints,
            method='SLSQP'
        )
        return result

    @staticmethod
    def numerical_integration(func, a, b):
        """Integração numérica"""
        result, error = integrate.quad(func, a, b)
        return {'integral': result, 'error': error}

    @staticmethod
    def curve_fitting(x_data, y_data, degree=3):
        """Ajuste de curvas polinomiais"""
        coefficients = np.polyfit(x_data, y_data, degree)
        polynomial = np.poly1d(coefficients)

        # Calcular R²
        y_pred = polynomial(x_data)
        ss_res = np.sum((y_data - y_pred) ** 2)
        ss_tot = np.sum((y_data - np.mean(y_data)) ** 2)
        r_squared = 1 - (ss_res / ss_tot)

        return {
            'polynomial': polynomial,
            'coefficients': coefficients,
            'r_squared': r_squared
        }
```

Statistical Analysis

- ✅ **Advanced Statistics**: Análise estatística avançada

- **Funcionalidades**:
  - Testes de hipóteses
  - Análise de variância (ANOVA)
  - Regressão múltipla
  - Análise de componentes principais (PCA)
  - Clustering e classificação

- **Implementação de Testes Estatísticos**:

```python
from scipy import stats
import pandas as pd

class StatisticalTests:
    @staticmethod
    def hypothesis_testing(sample1, sample2, test_type='ttest'):
        """Testes de hipóteses"""
        if test_type == 'ttest':
            statistic, p_value = stats.ttest_ind(sample1, sample2)
        elif test_type == 'mannwhitney':
```

```
            statistic, p_value = stats.mannwhitneyu(sample1, sample2)
        elif test_type == 'kstest':
            statistic, p_value = stats.ks_2samp(sample1, sample2)

        return {
            'statistic': statistic,
            'p_value': p_value,
            'significant': p_value < 0.05
        }

    @staticmethod
    def anova_analysis(groups):
        """Análise de variância"""
        f_statistic, p_value = stats.f_oneway(*groups)

        # Post-hoc analysis se significativo
        if p_value < 0.05:
            # Tukey HSD test
            from scipy.stats import tukey_hsd
            tukey_result = tukey_hsd(*groups)
            return {
                'f_statistic': f_statistic,
                'p_value': p_value,
                'tukey_hsd': tukey_result
            }

        return {
            'f_statistic': f_statistic,
            'p_value': p_value,
            'significant': p_value < 0.05
        }
```

🔧 FERRAMENTAS DE DESENVOLVIMENTO

Code Quality & Testing

- ✅ **Testing Frameworks**: Frameworks de teste completos
- **Funcionalidades**:

  Unit Testing:

  python — Copiar

```
import unittest
from unittest.mock import Mock, patch

class TestDataProcessor(unittest.TestCase):
    def setUp(self):
        self.processor = DataProcessor()
        self.sample_data = pd.DataFrame({
            'A': [1, 2, 3, 4, 5],
            'B': [2, 4, 6, 8, 10]
        })

    def test_data_cleaning(self):
        """Testa limpeza de dados"""
        dirty_data = self.sample_data.copy()
        dirty_data.loc[2, 'A'] = np.nan

        clean_data = self.processor.clean_data(dirty_data)

        self.assertEqual(len(clean_data), 4)
        self.assertFalse(clean_data.isnull().any().any())

    def test_statistical_analysis(self):
        """Testa análise estatística"""
        result = self.processor.analyze(self.sample_data)

        self.assertIn('mean', result)
        self.assertIn('std', result)
        self.assertAlmostEqual(result['mean']['A'], 3.0)

    @patch('pandas.read_csv')
    def test_data_loading(self, mock_read_csv):
        """Testa carregamento com mock"""
        mock_read_csv.return_value = self.sample_data

        result = self.processor.load_data('test.csv')

        mock_read_csv.assert_called_once_with('test.csv')
        pd.testing.assert_frame_equal(result, self.sample_data)
```

  Integration Testing:

  python — Copiar

```
class TestDashboardIntegration(unittest.TestCase):
    def setUp(self):
        self.app = create_dash_app()
        self.client = self.app.server.test_client()

    def test_dashboard_rendering(self):
        """Testa renderização do dashboard"""
        response = self.client.get('/')
        self.assertEqual(response.status_code, 200)
        self.assertIn(b'Dashboard', response.data)

    def test_callback_functionality(self):
        """Testa callbacks do Dash"""
        with self.app.test_client() as client:
            # Simular input do usuário
            response = client.post('/callback', json={
                'inputs': [{'id': 'dropdown', 'property': 'value', 'value': 'option1'}]
            })

            self.assertEqual(response.status_code, 200)
            data = response.get_json()
            self.assertIn('output', data)
```

Performance Optimization

- ✅ **Profiling Tools**: Ferramentas de profiling
- **Funcionalidades**:

  Memory Profiling:

  python — Copiar

```
import psutil
import time
from functools import wraps

def memory_profiler(func):
    """Decorator para monitorar uso de memória"""
    @wraps(func)
    def wrapper(*args, **kwargs):
        process = psutil.Process()
        mem_before = process.memory_info().rss / 1024 / 1024  # MB

        start_time = time.time()
        result = func(*args, **kwargs)
        end_time = time.time()

        mem_after = process.memory_info().rss / 1024 / 1024  # MB

        print(f"Function: {func.__name__}")
        print(f"Memory before: {mem_before:.2f} MB")
        print(f"Memory after: {mem_after:.2f} MB")
        print(f"Memory diff: {mem_after - mem_before:.2f} MB")
        print(f"Execution time: {end_time - start_time:.2f} seconds")

        return result
    return wrapper
```

  Performance Benchmarking:

  python — Copiar

```python
import timeit
import pandas as pd
import numpy as np

class PerformanceBenchmark:
    @staticmethod
    def benchmark_pandas_operations():
        """Benchmark operações pandas"""
        df = pd.DataFrame(np.random.randn(100000, 10))

        benchmarks = {}

        # Groupby operation
        benchmarks['groupby'] = timeit.timeit(
            lambda: df.groupby(df.columns[0] > 0).mean(),
            number=100
        )

        # Sorting operation
        benchmarks['sort'] = timeit.timeit(
            lambda: df.sort_values(df.columns[0]),
            number=100
        )

        # Filtering operation
        benchmarks['filter'] = timeit.timeit(
            lambda: df[df[df.columns[0]] > 0],
            number=100
        )

        return benchmarks
```

Documentation Generation

- ✅ **Auto Documentation**: Geração automática de documentação
- **Funcionalidades**:

  Docstring Standards:

  python Copiar

```python
def advanced_data_analysis(df: pd.DataFrame,
                           target_column: str,
                           method: str = 'correlation') -> dict:
    """
    Performs advanced data analysis on a DataFrame.

    This function provides comprehensive analysis including correlation
    analysis, statistical summaries, and data quality assessment.

    Parameters
    ----------
    df : pd.DataFrame
        The input DataFrame containing the data to analyze
    target_column : str
        The name of the target column for analysis
    method : str, optional
        The analysis method to use ('correlation', 'regression', 'clustering')
        Default is 'correlation'

    Returns
    -------
    dict
        A dictionary containing analysis results with the following keys:
        - 'summary': Statistical summary of the data
        - 'correlations': Correlation matrix
        - 'quality': Data quality metrics
        - 'insights': Key insights and recommendations

    Raises
    ------
    ValueError
        If target_column is not found in the DataFrame
    TypeError
        If df is not a pandas DataFrame

    Examples
    --------
    >>> df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]})
    >>> result = advanced_data_analysis(df, 'A')
    >>> print(result['summary'])

    Notes
    -----
    This function requires pandas and numpy to be installed.
    For large datasets (>1M rows), consider using method='sampling'
    for better performance.

    See Also
    --------
    basic_data_analysis : For simpler analysis
    statistical_tests : For hypothesis testing
    """
    # Implementation here
    pass
```

🌟 RECURSOS ESPECIALIZADOS

Machine Learning Integration

- ✅ **Scikit-learn**: Biblioteca completa de ML (se disponível)
- **Algoritmos Disponíveis**:
  - **Supervised Learning**: Linear/Logistic Regression, Random Forest, SVM
  - **Unsupervised Learning**: K-Means, PCA, DBSCAN
  - **Model Selection**: Cross-validation, Grid Search
  - **Preprocessing**: Scaling, Encoding, Feature Selection
- **ML Pipeline Example**:

  python Copiar

```python
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix

class MLPipeline:
    def __init__(self):
        self.model = None
        self.scaler = StandardScaler()
        self.label_encoder = LabelEncoder()

    def preprocess_data(self, X, y):
        """Preprocessamento completo"""
        # Scaling features
        X_scaled = self.scaler.fit_transform(X)

        # Encoding labels
        y_encoded = self.label_encoder.fit_transform(y)

        return X_scaled, y_encoded

    def train_model(self, X, y, model_type='random_forest'):
        """Treina modelo com otimização de hiperparâmetros"""
        X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size=0.2, random_state=42
        )

        if model_type == 'random_forest':
            param_grid = {
                'n_estimators': [100, 200, 300],
                'max_depth': [10, 20, None],
                'min_samples_split': [2, 5, 10]
            }
```

```
            self.model = GridSearchCV(
                RandomForestClassifier(random_state=42),
                param_grid,
                cv=5,
                scoring='accuracy'
            )

        self.model.fit(X_train, y_train)

        # Avaliação
        y_pred = self.model.predict(X_test)

        return {
            'best_params': self.model.best_params_,
            'best_score': self.model.best_score_,
            'test_accuracy': self.model.score(X_test, y_test),
            'classification_report': classification_report(y_test, y_pred),
            'confusion_matrix': confusion_matrix(y_test, y_pred)
        }
```

Advanced Visualization Techniques

- ✅ **3D Visualizations**: Gráficos tridimensionais
- **Funcionalidades**:

  3D Scatter Plots:

  python Copiar

```
def create_3d_scatter(df, x_col, y_col, z_col, color_col=None):
    """Cria gráfico 3D interativo"""
    fig = go.Figure(data=go.Scatter3d(
        x=df[x_col],
        y=df[y_col],
        z=df[z_col],
        mode='markers',
        marker=dict(
            size=8,
            color=df[color_col] if color_col else 'blue',
            colorscale='Viridis',
            showscale=True,
            opacity=0.8
        ),
        text=df.index,
        hovertemplate=f'<b>%{{text}}</b><br>' +
                      f'{x_col}: %{{x}}<br>' +
                      f'{y_col}: %{{y}}<br>' +
                      f'{z_col}: %{{z}}<extra></extra>'
    ))

    fig.update_layout(
        title='3D Scatter Plot',
        scene=dict(
            xaxis_title=x_col,
            yaxis_title=y_col,
            zaxis_title=z_col,
            camera=dict(
                eye=dict(x=1.2, y=1.2, z=1.2)
            )
        ),
        width=800,
        height=600
    )

    return fig
```

  Surface Plots:

  python Copiar

```
def create_surface_plot(x_range, y_range, z_function):
    """Cria gráfico de superfície"""
    x = np.linspace(x_range[0], x_range[1], 50)
    y = np.linspace(y_range[0], y_range[1], 50)
    X, Y = np.meshgrid(x, y)
    Z = z_function(X, Y)

    fig = go.Figure(data=go.Surface(
        x=X, y=Y, z=Z,
        colorscale='Viridis',
        showscale=True
    ))

    fig.update_layout(
        title='3D Surface Plot',
        scene=dict(
            xaxis_title='X Axis',
            yaxis_title='Y Axis',
            zaxis_title='Z Axis'
        ),
        width=800,
        height=600
    )

    return fig
```

Real-time Data Simulation

- ✅ **Live Data Streaming**: Simulação de dados em tempo real
- **Funcionalidades**:

  Data Stream Simulator:

  python Copiar

```
import threading
import time
from collections import deque

class RealTimeDataSimulator:
    def __init__(self, max_points=100):
        self.data = deque(maxlen=max_points)
        self.is_running = False
        self.thread = None

    def generate_data_point(self):
        """Gera um ponto de dados simulado"""
        timestamp = time.time()
        value = np.random.normal(50, 10) + 10 * np.sin(timestamp / 10)
        return {'timestamp': timestamp, 'value':
```

Continue

Continue

Monica Sonnet

python Copiar

```
import threading
import time
from collections import deque

class RealTimeDataSimulator:
    def __init__(self, max_points=100):
        self.data = deque(maxlen=max_points)
        self.is_running = False
        self.thread = None

    def generate_data_point(self):
        """Gera um ponto de dados simulado"""
        timestamp = time.time()
        value = np.random.normal(50, 10) + 10 * np.sin(timestamp / 10)
        return {'timestamp': timestamp, 'value': value}
```

```python
    def start_streaming(self, interval=1.0):
        """Inicia simulação de streaming"""
        self.is_running = True

        def stream_data():
            while self.is_running:
                data_point = self.generate_data_point()
                self.data.append(data_point)
                time.sleep(interval)

        self.thread = threading.Thread(target=stream_data)
        self.thread.daemon = True
        self.thread.start()

    def stop_streaming(self):
        """Para simulação de streaming"""
        self.is_running = False
        if self.thread:
            self.thread.join()

    def get_current_data(self):
        """Retorna dados atuais como DataFrame"""
        if not self.data:
            return pd.DataFrame()

        return pd.DataFrame(list(self.data))

    def create_live_chart(self):
        """Cria gráfico atualizado em tempo real"""
        df = self.get_current_data()
        if df.empty:
            return go.Figure()

        fig = go.Figure()
        fig.add_trace(go.Scatter(
            x=pd.to_datetime(df['timestamp'], unit='s'),
            y=df['value'],
            mode='lines+markers',
            name='Live Data',
            line=dict(color='blue', width=2),
            marker=dict(size=4)
        ))

        fig.update_layout(
            title='Real-time Data Stream',
            xaxis_title='Time',
            yaxis_title='Value',
            showlegend=True,
            height=400
        )

        return fig
```

🎨 DESIGN E UX/UI AVANÇADO

Modern UI Components

- ✅ **Component Library**: Biblioteca completa de componentes UI
- **Funcionalidades Disponíveis**:

  Advanced Form Components:

```tsx
import React, { useState } from 'react';
import { Select, SelectContent, SelectItem, SelectTrigger, SelectValue } from "@/components/ui/select";
import { Button } from "@/components/ui/button";
import { Input } from "@/components/ui/input";
import { Label } from "@/components/ui/label";
import { Card, CardContent, CardDescription, CardHeader, CardTitle } from "@/components/ui/card";

const AdvancedForm: React.FC = () => {
    const [formData, setFormData] = useState({
        name: '',
        email: '',
        category: '',
        priority: 'medium'
    });

    const [errors, setErrors] = useState<Record<string, string>>({});

    const validateForm = () => {
        const newErrors: Record<string, string> = {};

        if (!formData.name.trim()) {
            newErrors.name = 'Name is required';
        }

        if (!formData.email.trim()) {
            newErrors.email = 'Email is required';
        } else if (!/\S+@\S+\.\S+/.test(formData.email)) {
            newErrors.email = 'Email is invalid';
        }

        if (!formData.category) {
            newErrors.category = 'Category is required';
        }

        setErrors(newErrors);
        return Object.keys(newErrors).length === 0;
    };

    const handleSubmit = (e: React.FormEvent) => {
        e.preventDefault();
        if (validateForm()) {
            console.log('Form submitted:', formData);
            // Handle form submission
        }
    };

    return (
        <Card className="w-full max-w-md mx-auto">
            <CardHeader>
                <CardTitle>Advanced Form</CardTitle>
                <CardDescription>
                    Complete form with validation and modern UI components
                </CardDescription>
            </CardHeader>
            <CardContent>
                <form onSubmit={handleSubmit} className="space-y-4">
                    <div className="space-y-2">
                        <Label htmlFor="name">Name</Label>
                        <Input
                            id="name"
                            value={formData.name}
                            onChange={(e) => setFormData({...formData, name: e.target.value})}
                            className={errors.name ? 'border-red-500' : ''}
                        />
                        {errors.name && <p className="text-sm text-red-500">{errors.name}</p>}
                    </div>

                    <div className="space-y-2">
                        <Label htmlFor="email">Email</Label>
                        <Input
                            id="email"
                            type="email"
                            value={formData.email}
                            onChange={(e) => setFormData({...formData, email: e.target.value})}
                            className={errors.email ? 'border-red-500' : ''}
                        />
                        {errors.email && <p className="text-sm text-red-500">{errors.email}</p>}
                    </div>

                    <div className="space-y-2">
                        <Label>Category</Label>
                        <Select
                            value={formData.category}
                            onValueChange={(value) => setFormData({...formData, category: value})}
                        >
                            <SelectTrigger className={errors.category ? 'border-red-500' : ''}>
```

```
                            <SelectValue placeholder="Select a category" />
                        </SelectTrigger>
                        <SelectContent>
                            <SelectItem value="business">Business</SelectItem>
                            <SelectItem value="personal">Personal</SelectItem>
                            <SelectItem value="technical">Technical</SelectItem>
                        </SelectContent>
                    </Select>
                    {errors.category && <p className="text-sm text-red-500">{errors.category}</p>}
                </div>

                <div className="space-y-2">
                    <Label>Priority</Label>
                    <Select
                        value={formData.priority}
                        onValueChange={(value) => setFormData({...formData, priority: value})}
                    >
                        <SelectTrigger>
                            <SelectValue />
                        </SelectTrigger>
                        <SelectContent>
                            <SelectItem value="low">Low</SelectItem>
                            <SelectItem value="medium">Medium</SelectItem>
                            <SelectItem value="high">High</SelectItem>
                            <SelectItem value="urgent">Urgent</SelectItem>
                        </SelectContent>
                    </Select>
                </div>

                <Button type="submit" className="w-full">
                    Submit Form
                </Button>
            </form>
        </CardContent>
    </Card>
  );
};
```

Responsive Design Patterns

- ✅ **Mobile-First Design**: Design responsivo completo
- **Funcionalidades**:

  CSS Grid & Flexbox Layouts:

  ```css
  /* Sistema de Grid Responsivo Avançado */
  .dashboard-layout {
      display: grid;
      grid-template-areas:
          "header header header"
          "sidebar main main"
          "footer footer footer";
      grid-template-rows: auto 1fr auto;
      grid-template-columns: 250px 1fr 1fr;
      min-height: 100vh;
      gap: 1rem;
  }

  .header {
      grid-area: header;
      background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
      padding: 1rem;
      color: white;
  }

  .sidebar {
      grid-area: sidebar;
      background: #f8fafc;
      padding: 1rem;
      border-right: 1px solid #e2e8f0;
  }

  .main-content {
      grid-area: main;
      padding: 1rem;
      overflow-y: auto;
  }

  /* Responsividade */
  @media (max-width: 768px) {
      .dashboard-layout {
          grid-template-areas:
              "header"
              "main"
              "footer";
          grid-template-columns: 1fr;
      }

      .sidebar {
          display: none;
      }
  }

  /* Container Queries para Componentes */
  .card-container {
      container-type: inline-size;
  }

  @container (min-width: 300px) {
      .card {
          display: flex;
          flex-direction: row;
          gap: 1rem;
      }
  }

  @container (max-width: 299px) {
      .card {
          display: flex;
          flex-direction: column;
      }
  }
  ```

  Advanced CSS Features:

  ```css
  /* Variáveis CSS Dinâmicas */
  :root {
      --primary-color: #3b82f6;
      --secondary-color: #64748b;
      --accent-color: #f59e0b;
      --background-color: #ffffff;
      --text-color: #1f2937;
      --border-radius: 0.5rem;
      --shadow: 0 4px 6px -1px rgba(0, 0, 0, 0.1);
  }

  /* Dark Mode Support */
  @media (prefers-color-scheme: dark) {
      :root {
          --background-color: #1f2937;
          --text-color: #f9fafb;
          --primary-color: #60a5fa;
      }
  }

  /* Animações Avançadas */
  @keyframes slideInFromLeft {
      0% {
          transform: translateX(-100%);
          opacity: 0;
      }
      100% {
          transform: translateX(0);
          opacity: 1;
      }
  }
  ```

```
@keyframes fadeInScale {
    0% {
        opacity: 0;
        transform: scale(0.8);
    }
    100% {
        opacity: 1;
        transform: scale(1);
    }
}

.animated-card {
    animation: fadeInScale 0.3s ease-out;
    transition: transform 0.2s ease, box-shadow 0.2s ease;
}

.animated-card:hover {
    transform: translateY(-4px);
    box-shadow: 0 10px 25px rgba(0, 0, 0, 0.15);
}
```

🚀 DEPLOYMENT E PRODUÇÃO

Build & Export Systems

- ✅ **Static Site Generation**: Geração de sites estáticos
- **Funcionalidades**:

HTML Export Pipeline:

python Copiar

```python
class StaticSiteGenerator:
    def __init__(self, output_dir='/home/user/build'):
        self.output_dir = output_dir
        self.assets_dir = f"{output_dir}/assets"
        self.pages = {}

    def create_directory_structure(self):
        """Cria estrutura de diretórios"""
        os.makedirs(self.output_dir, exist_ok=True)
        os.makedirs(self.assets_dir, exist_ok=True)
        os.makedirs(f"{self.assets_dir}/css", exist_ok=True)
        os.makedirs(f"{self.assets_dir}/js", exist_ok=True)
        os.makedirs(f"{self.assets_dir}/images", exist_ok=True)

    def generate_css(self, custom_styles=None):
        """Gera arquivo CSS otimizado"""
        base_css = """
        /* Reset CSS */
        *, *::before, *::after {
            box-sizing: border-box;
            margin: 0;
            padding: 0;
        }

        /* Base Styles */
        body {
            font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto, sans-serif;
            line-height: 1.6;
            color: #333;
            background-color: #fff;
        }

        /* Utility Classes */
        .container { max-width: 1200px; margin: 0 auto; padding: 0 1rem; }
        .grid { display: grid; gap: 1rem; }
        .flex { display: flex; }
        .flex-col { flex-direction: column; }
        .items-center { align-items: center; }
        .justify-center { justify-content: center; }
        .text-center { text-align: center; }
        .mt-4 { margin-top: 1rem; }
        .mb-4 { margin-bottom: 1rem; }
        .p-4 { padding: 1rem; }
        .rounded { border-radius: 0.5rem; }
        .shadow { box-shadow: 0 4px 6px -1px rgba(0, 0, 0, 0.1); }
        """

        if custom_styles:
            base_css += f"\n/* Custom Styles */\n{custom_styles}"

        css_path = f"{self.assets_dir}/css/styles.css"
        with open(css_path, 'w') as f:
            f.write(base_css)

        return css_path

    def generate_javascript(self, custom_js=None):
        """Gera arquivo JavaScript otimizado"""
        base_js = """
        // Utility Functions
        const $ = (selector) => document.querySelector(selector);
        const $$ = (selector) => document.querySelectorAll(selector);

        // DOM Ready
        document.addEventListener('DOMContentLoaded', function() {
            console.log('Site loaded successfully');

            // Initialize interactive components
            initializeCharts();
            initializeInteractivity();
        });

        // Chart Initialization
        function initializeCharts() {
            const chartElements = $$('[data-chart]');
            chartElements.forEach(element => {
                const chartType = element.dataset.chart;
                const chartData = JSON.parse(element.dataset.chartData || '{}');
                createChart(element, chartType, chartData);
            });
        }

        // Interactivity
        function initializeInteractivity() {
            // Smooth scrolling
            $$('a[href^="#"]').forEach(anchor => {
                anchor.addEventListener('click', function(e) {
                    e.preventDefault();
                    const target = $(this.getAttribute('href'));
                    if (target) {
                        target.scrollIntoView({ behavior: 'smooth' });
                    }
                });
            });

            // Mobile menu toggle
            const menuToggle = $('[data-menu-toggle]');
            const mobileMenu = $('[data-mobile-menu]');

            if (menuToggle && mobileMenu) {
                menuToggle.addEventListener('click', () => {
                    mobileMenu.classList.toggle('hidden');
                });
            }
        }
        """

        if custom_js:
            base_js += f"\n// Custom JavaScript\n{custom_js}"

        js_path = f"{self.assets_dir}/js/main.js"
        with open(js_path, 'w') as f:
            f.write(base_js)

        return js_path

    def create_html_page(self, title, content, template='default'):
```

```
        """Cria página HTML completa"""
        html_template = f"""
        <!DOCTYPE html>
        <html lang="pt-BR">
        <head>
            <meta charset="UTF-8">
            <meta name="viewport" content="width=device-width, initial-scale=1.0">
            <title>{title}</title>
            <meta name="description" content="Dashboard interativo com análise de dados">
            <link rel="stylesheet" href="assets/css/styles.css">
            <link rel="icon" type="image/x-icon" href="assets/images/favicon.ico">
        </head>
        <body>
            <header class="header">
                <nav class="container flex items-center justify-between p-4">
                    <div class="logo">
                        <h1 class="text-2xl font-bold">Dashboard</h1>
                    </div>
                    <div class="nav-links">
                        <a href="#home" class="nav-link">Home</a>
                        <a href="#analytics" class="nav-link">Analytics</a>
                        <a href="#reports" class="nav-link">Reports</a>
                    </div>
                </nav>
            </header>

            <main class="main-content">
                {content}
            </main>

            <footer class="footer mt-8 p-4 text-center">
                <p>&copy; 2024 Dashboard Analytics. All rights reserved.</p>
            </footer>

            <script src="assets/js/main.js"></script>
            <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
        </body>
        </html>
        """

        return html_template

    def build_site(self, pages_config):
        """Constrói site completo"""
        self.create_directory_structure()
        self.generate_css()
        self.generate_javascript()

        # Gerar páginas
        for page_name, page_config in pages_config.items():
            html_content = self.create_html_page(
                page_config['title'],
                page_config['content'],
                page_config.get('template', 'default')
            )

            page_path = f"{self.output_dir}/{page_name}.html"
            with open(page_path, 'w', encoding='utf-8') as f:
                f.write(html_content)

        return self.output_dir
```

Performance Optimization

- ✅ **Code Optimization**: Otimização de performance
- **Funcionalidades**:

  Lazy Loading Implementation:

  javascript — Copiar

```javascript
// Lazy Loading para Imagens
class LazyLoader {
    constructor() {
        this.imageObserver = null;
        this.init();
    }

    init() {
        if ('IntersectionObserver' in window) {
            this.imageObserver = new IntersectionObserver((entries, observer) => {
                entries.forEach(entry => {
                    if (entry.isIntersecting) {
                        const img = entry.target;
                        this.loadImage(img);
                        observer.unobserve(img);
                    }
                });
            });

            this.observeImages();
        } else {
            // Fallback para navegadores antigos
            this.loadAllImages();
        }
    }

    observeImages() {
        const lazyImages = document.querySelectorAll('img[data-src]');
        lazyImages.forEach(img => this.imageObserver.observe(img));
    }

    loadImage(img) {
        img.src = img.dataset.src;
        img.classList.remove('lazy');
        img.classList.add('loaded');
    }

    loadAllImages() {
        const lazyImages = document.querySelectorAll('img[data-src]');
        lazyImages.forEach(img => this.loadImage(img));
    }
}

// Virtual Scrolling para Grandes Datasets
class VirtualScroller {
    constructor(container, itemHeight, totalItems, renderItem) {
        this.container = container;
        this.itemHeight = itemHeight;
        this.totalItems = totalItems;
        this.renderItem = renderItem;
        this.visibleItems = Math.ceil(container.clientHeight / itemHeight) + 2;
        this.scrollTop = 0;

        this.init();
    }

    init() {
        this.container.style.height = `${this.totalItems * this.itemHeight}px`;
        this.container.style.position = 'relative';
        this.container.style.overflow = 'auto';

        this.viewport = document.createElement('div');
        this.viewport.style.position = 'absolute';
        this.viewport.style.top = '0';
        this.viewport.style.left = '0';
        this.viewport.style.right = '0';

        this.container.appendChild(this.viewport);

        this.container.addEventListener('scroll', () => {
            this.scrollTop = this.container.scrollTop;
            this.render();
        });

        this.render();
    }

    render() {
        const startIndex = Math.floor(this.scrollTop / this.itemHeight);
        const endIndex = Math.min(startIndex + this.visibleItems, this.totalItems);
```

```
        this.viewport.innerHTML = '';
        this.viewport.style.transform = `translateY(${startIndex * this.itemHeight}px)`;

        for (let i = startIndex; i < endIndex; i++) {
            const item = this.renderItem(i);
            item.style.height = `${this.itemHeight}px`;
            this.viewport.appendChild(item);
        }
    }
}
```

📊 ADVANCED ANALYTICS & REPORTING

Automated Report Generation

- ✅ **Report Builder**: Sistema completo de relatórios
- **Funcionalidades**:

PDF Report Generator:

python Copiar

```python
from reportlab.lib import colors
from reportlab.lib.pagesizes import letter, A4
from reportlab.platypus import SimpleDocTemplate, Table, TableStyle, Paragraph, Spacer, Image
from reportlab.lib.styles import getSampleStyleSheet, ParagraphStyle
from reportlab.lib.units import inch
import io
import base64

class AdvancedReportGenerator:
    def __init__(self):
        self.styles = getSampleStyleSheet()
        self.custom_styles = self._create_custom_styles()

    def _create_custom_styles(self):
        """Cria estilos customizados para o relatório"""
        custom_styles = {}

        # Título principal
        custom_styles['CustomTitle'] = ParagraphStyle(
            'CustomTitle',
            parent=self.styles['Heading1'],
            fontSize=24,
            spaceAfter=30,
            textColor=colors.HexColor('#2563eb'),
            alignment=1  # Center
        )

        # Subtítulo
        custom_styles['CustomSubtitle'] = ParagraphStyle(
            'CustomSubtitle',
            parent=self.styles['Heading2'],
            fontSize=16,
            spaceAfter=20,
            textColor=colors.HexColor('#64748b')
        )

        # Texto de destaque
        custom_styles['Highlight'] = ParagraphStyle(
            'Highlight',
            parent=self.styles['Normal'],
            fontSize=12,
            textColor=colors.HexColor('#dc2626'),
            backColor=colors.HexColor('#fef2f2'),
            borderPadding=5
        )

        return custom_styles

    def create_kpi_table(self, kpi_data):
        """Cria tabela de KPIs"""
        data = [['Métrica', 'Valor', 'Variação', 'Status']]

        for kpi in kpi_data:
            status_color = colors.green if kpi['change'] >= 0 else colors.red
            data.append([
                kpi['name'],
                kpi['value'],
                f"{kpi['change']:+.1%}",
                kpi['status']
            ])

        table = Table(data, colWidths=[2*inch, 1.5*inch, 1*inch, 1*inch])
        table.setStyle(TableStyle([
            ('BACKGROUND', (0, 0), (-1, 0), colors.HexColor('#f1f5f9')),
            ('TEXTCOLOR', (0, 0), (-1, 0), colors.HexColor('#1e293b')),
            ('ALIGN', (0, 0), (-1, -1), 'CENTER'),
            ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
            ('FONTSIZE', (0, 0), (-1, 0), 12),
            ('BOTTOMPADDING', (0, 0), (-1, 0), 12),
            ('BACKGROUND', (0, 1), (-1, -1), colors.white),
            ('GRID', (0, 0), (-1, -1), 1, colors.HexColor('#e2e8f0'))
        ]))

        return table

    def create_chart_from_plotly(self, fig):
        """Converte gráfico Plotly para imagem no relatório"""
        # Exportar como PNG
        img_bytes = fig.to_image(format="png", width=600, height=400)

        # Converter para base64 para embedding
        img_base64 = base64.b64encode(img_bytes).decode()

        # Criar objeto Image do ReportLab
        img_buffer = io.BytesIO(img_bytes)
        img = Image(img_buffer, width=6*inch, height=4*inch)

        return img

    def generate_comprehensive_report(self, data, charts, output_path):
        """Gera relatório completo"""
        doc = SimpleDocTemplate(
            output_path,
            pagesize=A4,
            rightMargin=72,
            leftMargin=72,
            topMargin=72,
            bottomMargin=18
        )

        story = []

        # Título
        title = Paragraph("Relatório de Análise de Dados", self.custom_styles['CustomTitle'])
        story.append(title)
        story.append(Spacer(1, 20))

        # Resumo executivo
        summary = Paragraph(
            "Este relatório apresenta uma análise abrangente dos dados coletados, "
            "incluindo métricas principais, tendências identificadas e recomendações estratégicas.",
            self.styles['Normal']
        )
        story.append(summary)
        story.append(Spacer(1, 20))

        # KPIs
        subtitle_kpi = Paragraph("Indicadores Principais", self.custom_styles['CustomSubtitle'])
        story.append(subtitle_kpi)

        kpi_table = self.create_kpi_table(data['kpis'])
        story.append(kpi_table)
        story.append(Spacer(1, 20))

        # Gráficos
        subtitle_charts = Paragraph("Análise Visual", self.custom_styles['CustomSubtitle'])
        story.append(subtitle_charts)
```

```
        for chart_name, chart_fig in charts.items():
            chart_title = Paragraph(f"Gráfico: {chart_name}", self.styles['Heading3'])
            story.append(chart_title)

            chart_img = self.create_chart_from_plotly(chart_fig)
            story.append(chart_img)
            story.append(Spacer(1, 15))

        # Insights e Recomendações
        subtitle_insights = Paragraph("Insights e Recomendações", self.custom_styles['CustomSubtitle'])
        story.append(subtitle_insights)

        for insight in data.get('insights', []):
            insight_para = Paragraph(f"• {insight}", self.styles['Normal'])
            story.append(insight_para)

        story.append(Spacer(1, 20))

        # Conclusão
        conclusion = Paragraph(
            "Com base na análise realizada, recomenda-se o monitoramento contínuo "
            "dos indicadores apresentados e a implementação das ações sugeridas.",
            self.custom_styles['Highlight']
        )
        story.append(conclusion)

        # Construir PDF
        doc.build(story)

        return output_path
```

Advanced Data Mining

- ✅ **Pattern Recognition**: Reconhecimento de padrões
- **Funcionalidades**:

  Anomaly Detection System:

  python Copiar

```
from sklearn.ensemble import IsolationForest
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import numpy as np
import pandas as pd

class AnomalyDetectionSystem:
    def __init__(self):
        self.models = {}
        self.scalers = {}
        self.thresholds = {}

    def detect_statistical_anomalies(self, data, column, method='zscore'):
        """Detecta anomalias usando métodos estatísticos"""
        if method == 'zscore':
            z_scores = np.abs((data[column] - data[column].mean()) / data[column].std())
            threshold = 3
            anomalies = data[z_scores > threshold]

        elif method == 'iqr':
            Q1 = data[column].quantile(0.25)
            Q3 = data[column].quantile(0.75)
            IQR = Q3 - Q1
            lower_bound = Q1 - 1.5 * IQR
            upper_bound = Q3 + 1.5 * IQR
            anomalies = data[(data[column] < lower_bound) | (data[column] > upper_bound)]

        elif method == 'modified_zscore':
            median = data[column].median()
            mad = np.median(np.abs(data[column] - median))
            modified_z_scores = 0.6745 * (data[column] - median) / mad
            threshold = 3.5
            anomalies = data[np.abs(modified_z_scores) > threshold]

        return anomalies

    def detect_multivariate_anomalies(self, data, contamination=0.1):
        """Detecta anomalias multivariadas usando Isolation Forest"""
        # Preparar dados
        numeric_columns = data.select_
```

Continue

Monica Sonnet

python Copiar

```
    def detect_multivariate_anomalies(self, data, contamination=0.1):
        """Detecta anomalias multivariadas usando Isolation Forest"""
        # Preparar dados
        numeric_columns = data.select_dtypes(include=[np.number]).columns
        X = data[numeric_columns].copy()

        # Tratar valores ausentes
        X = X.fillna(X.mean())

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

        # Treinar modelo Isolation Forest
        iso_forest = IsolationForest(
            contamination=contamination,
            random_state=42,
            n_estimators=100
        )

        # Detectar anomalias
        anomaly_labels = iso_forest.fit_predict(X_scaled)
        anomaly_scores = iso_forest.decision_function(X_scaled)

        # Adicionar resultados ao DataFrame
        results = data.copy()
        results['anomaly_score'] = anomaly_scores
        results['is_anomaly'] = anomaly_labels == -1

        # Armazenar modelo para uso futuro
        self.models['isolation_forest'] = iso_forest
        self.scalers['isolation_forest'] = scaler

        return results[results['is_anomaly']], results

    def detect_time_series_anomalies(self, data, date_column, value_column):
        """Detecta anomalias em séries temporais"""
        # Preparar série temporal
        ts_data = data.copy()
        ts_data[date_column] = pd.to_datetime(ts_data[date_column])
        ts_data = ts_data.sort_values(date_column)
        ts_data.set_index(date_column, inplace=True)

        # Calcular médias móveis
        ts_data['rolling_mean'] = ts_data[value_column].rolling(window=30).mean()
        ts_data['rolling_std'] = ts_data[value_column].rolling(window=30).std()

        # Detectar anomalias baseadas em desvio da média móvel
        ts_data['z_score'] = np.abs(
            (ts_data[value_column] - ts_data['rolling_mean']) / ts_data['rolling_std']
        )

        threshold = 3
        ts_data['is_anomaly'] = ts_data['z_score'] > threshold

        # Detectar mudanças de tendência
        ts_data['diff'] = ts_data[value_column].diff()
        ts_data['trend_change'] = np.abs(ts_data['diff']) > (2 * ts_data['diff'].std())

        return ts_data[ts_data['is_anomaly'] | ts_data['trend_change']], ts_data
```

```python
    def create_anomaly_report(self, anomalies, original_data):
        """Cria relatório detalhado de anomalias"""
        report = {
            'summary': {
                'total_records': len(original_data),
                'anomalies_found': len(anomalies),
                'anomaly_rate': len(anomalies) / len(original_data) * 100
            },
            'anomaly_details': anomalies.to_dict('records'),
            'statistics': {
                'most_common_anomaly_type': 'statistical_outlier',
                'severity_distribution': anomalies.groupby('anomaly_score').size().to_dict() if 'anomaly_score' in anomalies.columns else {}
            }
        }

        return report
```

Predictive Analytics Engine

- ✅ **Forecasting Models**: Modelos de previsão avançados
- **Funcionalidades**:

Time Series Forecasting:

python Copiar

```python
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
import warnings
warnings.filterwarnings('ignore')

class TimeSeriesForecaster:
    def __init__(self):
        self.models = {}
        self.forecasts = {}
        self.metrics = {}

    def prepare_features(self, data, date_column, value_column, lag_features=5):
        """Prepara features para modelos de ML"""
        df = data.copy()
        df[date_column] = pd.to_datetime(df[date_column])
        df = df.sort_values(date_column)

        # Features temporais
        df['year'] = df[date_column].dt.year
        df['month'] = df[date_column].dt.month
        df['day'] = df[date_column].dt.day
        df['dayofweek'] = df[date_column].dt.dayofweek
        df['quarter'] = df[date_column].dt.quarter

        # Features de lag
        for i in range(1, lag_features + 1):
            df[f'lag_{i}'] = df[value_column].shift(i)

        # Features de média móvel
        for window in [7, 14, 30]:
            df[f'rolling_mean_{window}'] = df[value_column].rolling(window=window).mean()
            df[f'rolling_std_{window}'] = df[value_column].rolling(window=window).std()

        # Features de diferenciação
        df['diff_1'] = df[value_column].diff()
        df['diff_7'] = df[value_column].diff(7)

        # Remover valores NaN
        df = df.dropna()

        return df

    def train_multiple_models(self, data, target_column, test_size=0.2):
        """Treina múltiplos modelos e compara performance"""
        # Separar features e target
        feature_columns = [col for col in data.columns if col != target_column and not col.str.contains('date')]
        X = data[feature_columns]
        y = data[target_column]

        # Dividir dados
        split_index = int(len(data) * (1 - test_size))
        X_train, X_test = X[:split_index], X[split_index:]
        y_train, y_test = y[:split_index], y[split_index:]

        # Modelos a treinar
        models_to_train = {
            'linear_regression': LinearRegression(),
            'random_forest': RandomForestRegressor(n_estimators=100, random_state=42),
            'gradient_boosting': GradientBoostingRegressor(n_estimators=100, random_state=42)
        }

        results = {}

        for model_name, model in models_to_train.items():
            # Treinar modelo
            model.fit(X_train, y_train)

            # Fazer previsões
            train_pred = model.predict(X_train)
            test_pred = model.predict(X_test)

            # Calcular métricas
            train_mae = mean_absolute_error(y_train, train_pred)
            test_mae = mean_absolute_error(y_test, test_pred)
            train_rmse = np.sqrt(mean_squared_error(y_train, train_pred))
            test_rmse = np.sqrt(mean_squared_error(y_test, test_pred))

            results[model_name] = {
                'model': model,
                'train_mae': train_mae,
                'test_mae': test_mae,
                'train_rmse': train_rmse,
                'test_rmse': test_rmse,
                'predictions': test_pred
            }

        # Armazenar modelos e métricas
        self.models = {name: result['model'] for name, result in results.items()}
        self.metrics = results

        return results

    def generate_forecasts(self, data, periods=30, confidence_interval=0.95):
        """Gera previsões para períodos futuros"""
        forecasts = {}

        for model_name, model in self.models.items():
            # Preparar dados para previsão
            last_features = data.iloc[-1:][model.feature_names_in_].values

            predictions = []
            confidence_lower = []
            confidence_upper = []

            # Gerar previsões sequenciais
            for i in range(periods):
                pred = model.predict(last_features.reshape(1, -1))[0]
                predictions.append(pred)

                # Calcular intervalo de confiança (simplificado)
                std_error = self.metrics[model_name]['test_rmse']
                margin = 1.96 * std_error  # 95% CI
                confidence_lower.append(pred - margin)
                confidence_upper.append(pred + margin)

                # Atualizar features para próxima previsão (simplificado)
                # Em implementação real, seria mais sofisticado
                last_features = np.roll(last_features, -1)
                last_features[-1] = pred

            forecasts[model_name] = {
                'predictions': predictions,
                'confidence_lower': confidence_lower,
                'confidence_upper': confidence_upper
```

```
                }

            self.forecasts = forecasts
            return forecasts

        def create_forecast_visualization(self, historical_data, forecasts, date_column, value_column):
            """Cria visualização das previsões"""
            fig = go.Figure()

            # Dados históricos
            fig.add_trace(go.Scatter(
                x=historical_data[date_column],
                y=historical_data[value_column],
                mode='lines',
                name='Dados Históricos',
                line=dict(color='blue')
            ))

            # Previsões de cada modelo
            colors = ['red', 'green', 'orange', 'purple']
            future_dates = pd.date_range(
                start=historical_data[date_column].max() + pd.Timedelta(days=1),
                periods=len(list(forecasts.values())[0]['predictions']),
                freq='D'
            )

            for i, (model_name, forecast) in enumerate(forecasts.items()):
                color = colors[i % len(colors)]

                # Linha de previsão
                fig.add_trace(go.Scatter(
                    x=future_dates,
                    y=forecast['predictions'],
                    mode='lines',
                    name=f'Previsão - {model_name}',
                    line=dict(color=color)
                ))

                # Intervalo de confiança
                fig.add_trace(go.Scatter(
                    x=future_dates,
                    y=forecast['confidence_upper'],
                    mode='lines',
                    line=dict(width=0),
                    showlegend=False,
                    hoverinfo='skip'
                ))

                fig.add_trace(go.Scatter(
                    x=future_dates,
                    y=forecast['confidence_lower'],
                    mode='lines',
                    line=dict(width=0),
                    fill='tonexty',
                    fillcolor=f'rgba({color}, 0.2)',
                    name=f'IC 95% - {model_name}',
                    hoverinfo='skip'
                ))

            fig.update_layout(
                title='Previsão de Séries Temporais',
                xaxis_title='Data',
                yaxis_title='Valor',
                hovermode='x unified',
                height=600
            )

            return fig
```

🔐 SEGURANÇA E VALIDAÇÃO

Data Validation Framework

- ✅ **Schema Validation**: Validação de esquemas de dados
- **Funcionalidades**:

  Comprehensive Data Validator:

  python — Copiar

```
import re
from typing import Dict, List, Any, Optional, Union
from datetime import datetime
import pandas as pd

class DataValidator:
    def __init__(self):
        self.validation_rules = {}
        self.validation_results = {}

    def define_schema(self, schema: Dict[str, Dict[str, Any]]):
        """Define esquema de validação para dados"""
        self.schema = schema

    def validate_data_types(self, data: pd.DataFrame) -> Dict[str, List[str]]:
        """Valida tipos de dados"""
        errors = []
        warnings = []

        for column, rules in self.schema.items():
            if column not in data.columns:
                errors.append(f"Coluna obrigatória '{column}' não encontrada")
                continue

            expected_type = rules.get('type')
            if expected_type:
                actual_type = data[column].dtype

                # Mapeamento de tipos
                type_mapping = {
                    'string': ['object', 'string'],
                    'integer': ['int64', 'int32', 'int16', 'int8'],
                    'float': ['float64', 'float32'],
                    'datetime': ['datetime64[ns]'],
                    'boolean': ['bool']
                }

                if str(actual_type) not in type_mapping.get(expected_type, []):
                    errors.append(
                        f"Coluna '{column}': tipo esperado {expected_type}, "
                        f"encontrado {actual_type}"
                    )

        return {'errors': errors, 'warnings': warnings}

    def validate_constraints(self, data: pd.DataFrame) -> Dict[str, List[str]]:
        """Valida restrições de dados"""
        errors = []
        warnings = []

        for column, rules in self.schema.items():
            if column not in data.columns:
                continue

            column_data = data[column]

            # Validar valores nulos
            if rules.get('required', False):
                null_count = column_data.isnull().sum()
                if null_count > 0:
                    errors.append(f"Coluna '{column}': {null_count} valores nulos encontrados")

            # Validar range de valores
            if 'min_value' in rules:
                min_violations = (column_data < rules['min_value']).sum()
                if min_violations > 0:
                    errors.append(
                        f"Coluna '{column}': {min_violations} valores abaixo do mínimo ({rules['min_value']})"
                    )
```

```
            if 'max_value' in rules:
                max_violations = (column_data > rules['max_value']).sum()
                if max_violations > 0:
                    errors.append(
                        f"Coluna '{column}': {max_violations} valores acima do máximo ({rules['max_value']})"
                    )

            # Validar valores únicos
            if rules.get('unique', False):
                duplicate_count = column_data.duplicated().sum()
                if duplicate_count > 0:
                    errors.append(f"Coluna '{column}': {duplicate_count} valores duplicados")

            # Validar padrões regex
            if 'pattern' in rules:
                pattern = rules['pattern']
                invalid_patterns = ~column_data.astype(str).str.match(pattern)
                invalid_count = invalid_patterns.sum()
                if invalid_count > 0:
                    errors.append(
                        f"Coluna '{column}': {invalid_count} valores não seguem o padrão {pattern}"
                    )

            # Validar valores permitidos
            if 'allowed_values' in rules:
                allowed = rules['allowed_values']
                invalid_values = ~column_data.isin(allowed)
                invalid_count = invalid_values.sum()
                if invalid_count > 0:
                    errors.append(
                        f"Coluna '{column}': {invalid_count} valores não permitidos"
                    )

        return {'errors': errors, 'warnings': warnings}

    def validate_business_rules(self, data: pd.DataFrame) -> Dict[str, List[str]]:
        """Valida regras de negócio customizadas"""
        errors = []
        warnings = []

        # Exemplo: Validar consistência entre datas
        if 'start_date' in data.columns and 'end_date' in data.columns:
            invalid_dates = data['start_date'] > data['end_date']
            if invalid_dates.any():
                count = invalid_dates.sum()
                errors.append(f"Data de início posterior à data de fim em {count} registros")

        # Exemplo: Validar relacionamentos entre campos
        if 'price' in data.columns and 'discount' in data.columns:
            invalid_discounts = data['discount'] > data['price']
            if invalid_discounts.any():
                count = invalid_discounts.sum()
                errors.append(f"Desconto maior que preço em {count} registros")

        # Exemplo: Validar totais calculados
        if all(col in data.columns for col in ['quantity', 'unit_price', 'total']):
            calculated_total = data['quantity'] * data['unit_price']
            tolerance = 0.01
            discrepancies = abs(data['total'] - calculated_total) > tolerance
            if discrepancies.any():
                count = discrepancies.sum()
                warnings.append(f"Discrepâncias no cálculo de total em {count} registros")

        return {'errors': errors, 'warnings': warnings}

    def generate_data_quality_report(self, data: pd.DataFrame) -> Dict[str, Any]:
        """Gera relatório completo de qualidade dos dados"""
        # Executar todas as validações
        type_validation = self.validate_data_types(data)
        constraint_validation = self.validate_constraints(data)
        business_validation = self.validate_business_rules(data)

        # Estatísticas gerais
        general_stats = {
            'total_records': len(data),
            'total_columns': len(data.columns),
            'memory_usage': data.memory_usage(deep=True).sum(),
            'missing_values_total': data.isnull().sum().sum(),
            'duplicate_rows': data.duplicated().sum()
        }

        # Estatísticas por coluna
        column_stats = {}
        for column in data.columns:
            col_data = data[column]
            column_stats[column] = {
                'data_type': str(col_data.dtype),
                'missing_count': col_data.isnull().sum(),
                'missing_percentage': (col_data.isnull().sum() / len(data)) * 100,
                'unique_count': col_data.nunique(),
                'unique_percentage': (col_data.nunique() / len(data)) * 100
            }

            # Estatísticas específicas por tipo
            if col_data.dtype in ['int64', 'float64']:
                column_stats[column].update({
                    'min': col_data.min(),
                    'max': col_data.max(),
                    'mean': col_data.mean(),
                    'std': col_data.std(),
                    'median': col_data.median()
                })
            elif col_data.dtype == 'object':
                column_stats[column].update({
                    'avg_length': col_data.astype(str).str.len().mean(),
                    'max_length': col_data.astype(str).str.len().max(),
                    'most_common': col_data.value_counts().head(3).to_dict()
                })

        # Compilar relatório final
        report = {
            'timestamp': datetime.now().isoformat(),
            'general_statistics': general_stats,
            'column_statistics': column_stats,
            'validation_results': {
                'data_types': type_validation,
                'constraints': constraint_validation,
                'business_rules': business_validation
            },
            'overall_quality_score': self._calculate_quality_score(
                type_validation, constraint_validation, business_validation
            )
        }

        return report

    def _calculate_quality_score(self, type_val, constraint_val, business_val) -> float:
        """Calcula score de qualidade dos dados (0-100)"""
        total_errors = (
            len(type_val['errors']) +
            len(constraint_val['errors']) +
            len(business_val['errors'])
        )

        total_warnings = (
            len(type_val['warnings']) +
            len(constraint_val['warnings']) +
            len(business_val['warnings'])
        )

        # Score baseado em erros e avisos (simplificado)
        if total_errors == 0 and total_warnings == 0:
            return 100.0
        elif total_errors == 0:
            return max(80.0, 100.0 - (total_warnings * 5))
        else:
            return max(0.0, 100.0 - (total_errors * 10) - (total_warnings * 2))
```

🎯 CASOS DE USO COMPLETOS E TEMPLATES

E-commerce Analytics Dashboard

- ✅ **Template Completo**: Dashboard para e-commerce

- **Funcionalidades**:

python

Copiar

```
class EcommerceDashboard:
    def __init__(self):
        self.data = None
        self.kpis = {}
        self.charts = {}

    def generate_sample_ecommerce_data(self, days=365, customers=1000):
        """Gera dados de exemplo para e-commerce"""
        np.random.seed(42)

        # Gerar datas
        date_range = pd.date_range(
            start=datetime.now() - pd.Timedelta(days=days),
            end=datetime.now(),
            freq='D'
        )

        # Gerar dados de vendas
        sales_data = []
        for date in date_range:
            # Simular sazonalidade
            seasonal_factor = 1 + 0.3 * np.sin(2 * np.pi * date.dayofyear / 365)
            weekend_factor = 1.2 if date.weekday() >= 5 else 1.0

            daily_orders = max(1, int(np.random.poisson(50) * seasonal_factor * weekend_factor))

            for _ in range(daily_orders):
                order = {
                    'date': date,
                    'customer_id': np.random.randint(1, customers + 1),
                    'product_category': np.random.choice([
                        'Electronics', 'Clothing', 'Home', 'Books', 'Sports'
                    ], p=[0.3, 0.25, 0.2, 0.15, 0.1]),
                    'order_value': np.random.lognormal(4, 0.5),
                    'quantity': np.random.randint(1, 5),
                    'payment_method': np.random.choice([
                        'Credit Card', 'Debit Card', 'PayPal', 'Bank Transfer'
                    ], p=[0.4, 0.3, 0.2, 0.1]),
                    'shipping_cost': np.random.uniform(5, 25),
                    'discount_applied': np.random.choice([0, 5, 10, 15, 20], p=[0.4, 0.2, 0.2, 0.15, 0.05])
                }
                sales_data.append(order)

        self.data = pd.DataFrame(sales_data)

        # Adicionar campos calculados
        self.data['net_revenue'] = self.data['order_value'] - self.data['shipping_cost']
        self.data['discounted_value'] = self.data['order_value'] * (1 - self.data['discount_applied'] / 100)
        self.data['month'] = self.data['date'].dt.to_period('M')
        self.data['week'] = self.data['date'].dt.to_period('W')

        return self.data

    def calculate_ecommerce_kpis(self):
        """Calcula KPIs principais do e-commerce"""
        if self.data is None:
            return {}

        # KPIs principais
        total_revenue = self.data['order_value'].sum()
        total_orders = len(self.data)
        unique_customers = self.data['customer_id'].nunique()
        avg_order_value = self.data['order_value'].mean()

        # Comparação com período anterior (últimos 30 vs 30 anteriores)
        last_30_days = self.data[self.data['date'] >= (self.data['date'].max() - pd.Timedelta(days=30))]
        prev_30_days = self.data[
            (self.data['date'] >= (self.data['date'].max() - pd.Timedelta(days=60))) &
            (self.data['date'] < (self.data['date'].max() - pd.Timedelta(days=30)))
        ]

        revenue_growth = ((last_30_days['order_value'].sum() - prev_30_days['order_value'].sum()) /
                          prev_30_days['order_value'].sum()) * 100 if len(prev_30_days) > 0 else 0

        self.kpis = {
            'total_revenue': {
                'value': f'${total_revenue:,.2f}',
                'change': f'{revenue_growth:+.1f}%',
                'status': 'positive' if revenue_growth > 0 else 'negative'
            },
            'total_orders': {
                'value': f'{total_orders:,}',
                'change': f'{((len(last_30_days) - len(prev_30_days)) / len(prev_30_days) * 100):+.1f}%' if len(prev_30_days) > 0 else '0%',
                'status': 'positive' if len(last_30_days) > len(prev_30_days) else 'negative'
            },
            'avg_order_value': {
                'value': f'${avg_order_value:.2f}',
                'change': f'{((last_30_days["order_value"].mean() - prev_30_days["order_value"].mean()) / prev_30_days["order_value"].mean() * 100):+.1f}%' if
len(prev_30_days) > 0 else '0%',
                'status': 'positive'
            },
            'unique_customers': {
                'value': f'{unique_customers:,}',
                'change': f'{((last_30_days["customer_id"].nunique() - prev_30_days["customer_id"].nunique()) / prev_30_days["customer_id"].nunique() * 100):+.1f}%' if
len(prev_30_days) > 0 else '0%',
                'status': 'positive'
            }
        }

        return self.kpis

    def create_revenue_trend_chart(self):
        """Cria gráfico de tendência de receita"""
        daily_revenue = self.data.groupby('date')['order_value'].sum().reset_index()

        fig = go.Figure()

        # Linha principal
        fig.add_trace(go.Scatter(
            x=daily_revenue['date'],
            y=daily_revenue['order_value'],
            mode='lines',
            name='Receita Diária',
            line=dict(color='#3b82f6', width=2)
        ))

        # Média móvel de 7 dias
        daily_revenue['ma_7'] = daily_revenue['order_value'].rolling(window=7).mean()
        fig.add_trace(go.Scatter(
            x=daily_revenue['date'],
            y=daily_revenue['ma_7'],
            mode='lines',
            name='Média Móvel (7 dias)',
            line=dict(color='#ef4444', width=2, dash='dash')
        ))

        fig.update_layout(
            title='Tendência de Receita ao Longo do Tempo',
            xaxis_title='Data',
            yaxis_title='Receita ($)',
            hovermode='x unified',
            height=400
        )

        return fig

    def create_category_performance_chart(self):
        """Cria gráfico de performance por categoria"""
        category_data = self.data.groupby('product_category').agg({
            'order_value': ['sum', 'count', 'mean']
        }).round(2)

        category_data.columns = ['Total Revenue', 'Order Count', 'Avg Order Value']
        category_data = category_data.reset_index()

        fig = make_subplots(
            rows=1, cols=2,
```

```
            subplot_titles=['Receita por Categoria', 'Pedidos por Categoria'],
            specs=[[{"secondary_y": False}, {"type": "pie"}]]
        )

        # Gráfico de barras
        fig.add_trace(
            go.Bar(
                x=category_data['product_category'],
                y=category_data['Total Revenue'],
                name='Receita Total',
                marker_color='#3b82f6'
            ),
            row=1, col=1
        )

        # Gráfico de pizza
        fig.add_trace(
            go.Pie(
                labels=category_data['product_category'],
                values=category_data['Order Count'],
                name='Distribuição de Pedidos'
            ),
            row=1, col=2
        )

        fig.update_layout(
            title='Performance por Categoria de Produto',
            height=500
        )

        return fig

    def create_customer_segmentation_chart(self):
        """Cria análise de segmentação de clientes"""
        customer_data = self.data.groupby('customer_id').agg({
            'order_value': ['sum', 'count'],
            'date': ['min', 'max']
        })

        customer_data.columns = ['Total Spent', 'Order Count', 'First Order', 'Last
```

Continue

Monica Sonnet

python Copiar

```
        customer_data.columns = ['Total Spent', 'Order Count', 'First Order', 'Last Order']
        customer_data['Days Active'] = (customer_data['Last Order'] - customer_data['First Order']).dt.days
        customer_data['Avg Order Value'] = customer_data['Total Spent'] / customer_data['Order Count']

        # Segmentação RFM simplificada
        customer_data['Recency'] = (self.data['date'].max() - customer_data['Last Order']).dt.days
        customer_data['Frequency'] = customer_data['Order Count']
        customer_data['Monetary'] = customer_data['Total Spent']

        # Criar quartis para segmentação
        customer_data['R_Score'] = pd.qcut(customer_data['Recency'], 4, labels=[4,3,2,1])
        customer_data['F_Score'] = pd.qcut(customer_data['Frequency'].rank(method='first'), 4, labels=[1,2,3,4])
        customer_data['M_Score'] = pd.qcut(customer_data['Monetary'], 4, labels=[1,2,3,4])

        # Criar segmentos
        def segment_customers(row):
            if row['R_Score'] >= 3 and row['F_Score'] >= 3 and row['M_Score'] >= 3:
                return 'Champions'
            elif row['R_Score'] >= 2 and row['F_Score'] >= 3:
                return 'Loyal Customers'
            elif row['R_Score'] >= 3 and row['F_Score'] <= 2:
                return 'Potential Loyalists'
            elif row['R_Score'] <= 2 and row['F_Score'] >= 2:
                return 'At Risk'
            else:
                return 'Others'

        customer_data['Segment'] = customer_data.apply(segment_customers, axis=1)

        # Criar visualização
        segment_summary = customer_data.groupby('Segment').agg({
            'Total Spent': 'sum',
            'Order Count': 'sum',
            'customer_id': 'count'
        }).rename(columns={'customer_id': 'Customer Count'})

        fig = make_subplots(
            rows=2, cols=2,
            subplot_titles=[
                'Distribuição de Clientes por Segmento',
                'Receita por Segmento',
                'Scatter: Recência vs Frequência',
                'Valor Monetário por Segmento'
            ],
            specs=[
                [{"type": "pie"}, {"type": "bar"}],
                [{"type": "scatter"}, {"type": "box"}]
            ]
        )

        # Gráfico de pizza - distribuição de clientes
        fig.add_trace(
            go.Pie(
                labels=segment_summary.index,
                values=segment_summary['Customer Count'],
                name='Clientes'
            ),
            row=1, col=1
        )

        # Gráfico de barras - receita por segmento
        fig.add_trace(
            go.Bar(
                x=segment_summary.index,
                y=segment_summary['Total Spent'],
                name='Receita',
                marker_color='#10b981'
            ),
            row=1, col=2
        )

        # Scatter plot - Recência vs Frequência
        colors = {'Champions': '#10b981', 'Loyal Customers': '#3b82f6',
                  'Potential Loyalists': '#f59e0b', 'At Risk': '#ef4444', 'Others': '#6b7280'}

        for segment in customer_data['Segment'].unique():
            segment_data = customer_data[customer_data['Segment'] == segment]
            fig.add_trace(
                go.Scatter(
                    x=segment_data['Recency'],
                    y=segment_data['Frequency'],
                    mode='markers',
                    name=segment,
                    marker=dict(color=colors.get(segment, '#6b7280'))
                ),
                row=2, col=1
            )

        # Box plot - valor monetário por segmento
        for segment in customer_data['Segment'].unique():
            segment_data = customer_data[customer_data['Segment'] == segment]
            fig.add_trace(
                go.Box(
                    y=segment_data['Monetary'],
                    name=segment,
                    marker_color=colors.get(segment, '#6b7280')
                ),
                row=2, col=2
            )

        fig.update_layout(
            title='Análise de Segmentação de Clientes (RFM)',
            height=800,
```

```python
            showlegend=True
        )
        
        return fig
    
    def create_payment_method_analysis(self):
        """Análise de métodos de pagamento"""
        payment_analysis = self.data.groupby('payment_method').agg({
            'order_value': ['sum', 'count', 'mean'],
            'discount_applied': 'mean'
        }).round(2)
        
        payment_analysis.columns = ['Total Revenue', 'Order Count', 'Avg Order Value', 'Avg Discount']
        payment_analysis = payment_analysis.reset_index()
        
        fig = make_subplots(
            rows=2, cols=2,
            subplot_titles=[
                'Receita por Método de Pagamento',
                'Número de Pedidos por Método',
                'Valor Médio do Pedido por Método',
                'Desconto Médio Aplicado por Método'
            ]
        )
        
        # Receita por método
        fig.add_trace(
            go.Bar(
                x=payment_analysis['payment_method'],
                y=payment_analysis['Total Revenue'],
                name='Receita Total',
                marker_color='#3b82f6'
            ),
            row=1, col=1
        )
        
        # Número de pedidos
        fig.add_trace(
            go.Bar(
                x=payment_analysis['payment_method'],
                y=payment_analysis['Order Count'],
                name='Número de Pedidos',
                marker_color='#10b981'
            ),
            row=1, col=2
        )
        
        # Valor médio do pedido
        fig.add_trace(
            go.Bar(
                x=payment_analysis['payment_method'],
                y=payment_analysis['Avg Order Value'],
                name='Valor Médio',
                marker_color='#f59e0b'
            ),
            row=2, col=1
        )
        
        # Desconto médio
        fig.add_trace(
            go.Bar(
                x=payment_analysis['payment_method'],
                y=payment_analysis['Avg Discount'],
                name='Desconto Médio (%)',
                marker_color='#ef4444'
            ),
            row=2, col=2
        )
        
        fig.update_layout(
            title='Análise de Métodos de Pagamento',
            height=600,
            showlegend=False
        )
        
        return fig
    
    def generate_executive_summary(self):
        """Gera resumo executivo do dashboard"""
        if not self.kpis:
            self.calculate_ecommerce_kpis()
        
        # Análises adicionais
        top_category = self.data.groupby('product_category')['order_value'].sum().idxmax()
        peak_day = self.data.groupby('date')['order_value'].sum().idxmax()
        avg_discount = self.data['discount_applied'].mean()
        
        summary = {
            'period_analyzed': f"{self.data['date'].min().strftime('%Y-%m-%d')} to {self.data['date'].max().strftime('%Y-%m-%d')}",
            'key_metrics': self.kpis,
            'insights': [
                f"A categoria {top_category} é a que mais gera receita",
                f"O pico de vendas ocorreu em {peak_day.strftime('%Y-%m-%d')}",
                f"Desconto médio aplicado é de {avg_discount:.1f}%",
                f"Temos {self.data['customer_id'].nunique()} clientes únicos",
                f"Valor médio do pedido é ${self.data['order_value'].mean():.2f}"
            ],
            'recommendations': [
                "Investir mais em marketing para a categoria de melhor performance",
                "Analisar fatores que levaram ao pico de vendas para replicar",
                "Revisar estratégia de descontos para otimizar margem",
                "Implementar programa de fidelidade para aumentar retenção",
                "Focar em aumentar o valor médio do pedido através de upselling"
            ]
        }
        
        return summary
```

Financial Analytics Template

- ✅ **Template Financeiro**: Dashboard para análise financeira
- **Funcionalidades**:

python Copiar

```python
class FinancialAnalyticsDashboard:
    def __init__(self):
        self.data = None
        self.financial_metrics = {}
    
    def generate_financial_data(self, months=24):
        """Gera dados financeiros de exemplo"""
        np.random.seed(42)
        
        # Gerar período
        date_range = pd.date_range(
            start=datetime.now() - pd.DateOffset(months=months),
            end=datetime.now(),
            freq='M'
        )
        
        financial_data = []
        base_revenue = 100000
        
        for i, date in enumerate(date_range):
            # Simular crescimento e sazonalidade
            growth_factor = 1 + (i * 0.02)  # 2% crescimento mensal
            seasonal_factor = 1 + 0.1 * np.sin(2 * np.pi * date.month / 12)
            noise_factor = 1 + np.random.normal(0, 0.05)
            
            revenue = base_revenue * growth_factor * seasonal_factor * noise_factor
            
            # Calcular outras métricas
            cogs = revenue * np.random.uniform(0.4, 0.6)  # 40-60% do revenue
            gross_profit = revenue - cogs
            
            # Despesas operacionais
            marketing_expense = revenue * np.random.uniform(0.1, 0.15)
            admin_expense = revenue * np.random.uniform(0.05, 0.1)
            rd_expense = revenue * np.random.uniform(0.02, 0.08)
            other_expenses = revenue * np.random.uniform(0.03, 0.07)
```

```
            total_opex = marketing_expense + admin_expense + rd_expense + other_expenses
            ebitda = gross_profit - total_opex

            # Depreciação e juros
            depreciation = revenue * 0.02
            interest_expense = revenue * 0.01

            ebit = ebitda - depreciation
            net_income = ebit - interest_expense

            # Métricas de balanço (simuladas)
            cash = revenue * np.random.uniform(0.1, 0.3)
            accounts_receivable = revenue * np.random.uniform(0.08, 0.15)
            inventory = cogs * np.random.uniform(0.1, 0.2)
            current_assets = cash + accounts_receivable + inventory

            fixed_assets = revenue * np.random.uniform(0.5, 1.0)
            total_assets = current_assets + fixed_assets

            accounts_payable = cogs * np.random.uniform(0.05, 0.12)
            short_term_debt = revenue * np.random.uniform(0.05, 0.1)
            current_liabilities = accounts_payable + short_term_debt

            long_term_debt = revenue * np.random.uniform(0.2, 0.4)
            total_liabilities = current_liabilities + long_term_debt

            equity = total_assets - total_liabilities

            financial_record = {
                'date': date,
                'revenue': revenue,
                'cogs': cogs,
                'gross_profit': gross_profit,
                'marketing_expense': marketing_expense,
                'admin_expense': admin_expense,
                'rd_expense': rd_expense,
                'other_expenses': other_expenses,
                'total_opex': total_opex,
                'ebitda': ebitda,
                'depreciation': depreciation,
                'ebit': ebit,
                'interest_expense': interest_expense,
                'net_income': net_income,
                'cash': cash,
                'accounts_receivable': accounts_receivable,
                'inventory': inventory,
                'current_assets': current_assets,
                'fixed_assets': fixed_assets,
                'total_assets': total_assets,
                'accounts_payable': accounts_payable,
                'short_term_debt': short_term_debt,
                'current_liabilities': current_liabilities,
                'long_term_debt': long_term_debt,
                'total_liabilities': total_liabilities,
                'equity': equity
            }

            financial_data.append(financial_record)

        self.data = pd.DataFrame(financial_data)

        # Calcular ratios financeiros
        self.data['gross_margin'] = (self.data['gross_profit'] / self.data['revenue']) * 100
        self.data['ebitda_margin'] = (self.data['ebitda'] / self.data['revenue']) * 100
        self.data['net_margin'] = (self.data['net_income'] / self.data['revenue']) * 100
        self.data['current_ratio'] = self.data['current_assets'] / self.data['current_liabilities']
        self.data['debt_to_equity'] = self.data['total_liabilities'] / self.data['equity']
        self.data['roa'] = (self.data['net_income'] / self.data['total_assets']) * 100
        self.data['roe'] = (self.data['net_income'] / self.data['equity']) * 100

        return self.data

    def create_pnl_chart(self):
        """Cria gráfico de P&L"""
        fig = go.Figure()

        # Revenue
        fig.add_trace(go.Scatter(
            x=self.data['date'],
            y=self.data['revenue'],
            mode='lines+markers',
            name='Receita',
            line=dict(color='#10b981', width=3)
        ))

        # Gross Profit
        fig.add_trace(go.Scatter(
            x=self.data['date'],
            y=self.data['gross_profit'],
            mode='lines+markers',
            name='Lucro Bruto',
            line=dict(color='#3b82f6', width=2)
        ))

        # EBITDA
        fig.add_trace(go.Scatter(
            x=self.data['date'],
            y=self.data['ebitda'],
            mode='lines+markers',
            name='EBITDA',
            line=dict(color='#f59e0b', width=2)
        ))

        # Net Income
        fig.add_trace(go.Scatter(
            x=self.data['date'],
            y=self.data['net_income'],
            mode='lines+markers',
            name='Lucro Líquido',
            line=dict(color='#ef4444', width=2)
        ))

        fig.update_layout(
            title='Demonstração de Resultados ao Longo do Tempo',
            xaxis_title='Data',
            yaxis_title='Valor ($)',
            hovermode='x unified',
            height=500
        )

        return fig

    def create_margin_analysis(self):
        """Análise de margens"""
        fig = make_subplots(
            rows=2, cols=1,
            subplot_titles=['Evolução das Margens (%)', 'Comparação de Margens por Período']
        )

        # Evolução das margens
        fig.add_trace(
            go.Scatter(
                x=self.data['date'],
                y=self.data['gross_margin'],
                mode='lines',
                name='Margem Bruta',
                line=dict(color='#10b981')
            ),
            row=1, col=1
        )

        fig.add_trace(
            go.Scatter(
                x=self.data['date'],
                y=self.data['ebitda_margin'],
                mode='lines',
                name='Margem EBITDA',
                line=dict(color='#3b82f6')
            ),
            row=1, col=1
        )
```

```
        fig.add_trace(
            go.Scatter(
                x=self.data['date'],
                y=self.data['net_margin'],
                mode='lines',
                name='Margem Líquida',
                line=dict(color='#ef4444')
            ),
            row=1, col=1
        )

        # Box plot das margens
        margins = ['gross_margin', 'ebitda_margin', 'net_margin']
        margin_names = ['Margem Bruta', 'Margem EBITDA', 'Margem Líquida']
        colors = ['#10b981', '#3b82f6', '#ef4444']

        for i, (margin, name, color) in enumerate(zip(margins, margin_names, colors)):
            fig.add_trace(
                go.Box(
                    y=self.data[margin],
                    name=name,
                    marker_color=color
                ),
                row=2, col=1
            )

        fig.update_layout(
            title='Análise de Margens Financeiras',
            height=700
        )

        return fig

    def create_balance_sheet_visualization(self):
        """Visualização do balanço patrimonial"""
        latest_data = self.data.iloc[-1]

        fig = make_subplots(
            rows=2, cols=2,
            subplot_titles=[
                'Composição dos Ativos',
                'Composição dos Passivos',
                'Evolução do Patrimônio Líquido',
                'Ratios Financeiros'
            ],
            specs=[
                [{"type": "pie"}, {"type": "pie"}],
                [{"type": "scatter"}, {"type": "bar"}]
            ]
        )

        # Composição dos ativos
        fig.add_trace(
            go.Pie(
                labels=['Caixa', 'Contas a Receber', 'Estoque', 'Ativos Fixos'],
                values=[
                    latest_data['cash'],
                    latest_data['accounts_receivable'],
                    latest_data['inventory'],
                    latest_data['fixed_assets']
                ],
                name='Ativos'
            ),
            row=1, col=1
        )

        # Composição dos passivos
        fig.add_trace(
            go.Pie(
                labels=['Contas a Pagar', 'Dívida Curto Prazo', 'Dívida Longo Prazo', 'Patrimônio Líquido'],
                values=[
                    latest_data['accounts_payable'],
                    latest_data['short_term_debt'],
                    latest_data['long_term_debt'],
                    latest_data['equity']
                ],
                name='Passivos + PL'
            ),
            row=1, col=2
        )

        # Evolução do patrimônio líquido
        fig.add_trace(
            go.Scatter(
                x=self.data['date'],
                y=self.data['equity'],
                mode='lines+markers',
                name='Patrimônio Líquido',
                line=dict(color='#10b981')
            ),
            row=2, col=1
        )

        # Ratios financeiros
        latest_ratios = {
            'Current Ratio': latest_data['current_ratio'],
            'Debt/Equity': latest_data['debt_to_equity'],
            'ROA (%)': latest_data['roa'],
            'ROE (%)': latest_data['roe']
        }

        fig.add_trace(
            go.Bar(
                x=list(latest_ratios.keys()),
                y=list(latest_ratios.values()),
                name='Ratios',
                marker_color=['#3b82f6', '#ef4444', '#10b981', '#f59e0b']
            ),
            row=2, col=2
        )

        fig.update_layout(
            title='Análise do Balanço Patrimonial',
            height=800
        )

        return fig
```

🎉 CONCLUSÃO E PRÓXIMOS PASSOS

Resumo das Capacidades Completas

Através desta análise extensiva, demonstrei que possuo um **ambiente de desenvolvimento completo e robusto** com as seguintes capacidades principais:

🔧 **Tecnologias Core**

- **Frontend Completo**: HTML5, CSS3, JavaScript ES6+, React/TSX, Tailwind CSS
- **Backend & Data**: Python 3.x, Pandas, NumPy, Plotly, Matplotlib
- **Visualização Avançada**: Gráficos interativos, dashboards, SVG, animações
- **Frameworks**: Dash, Jupyter, componentes UI modernos

📊 **Análise de Dados**

- **Manipulação**: Limpeza, transformação, agregação de dados
- **Estatística**: Análise descritiva, inferencial, testes de hipóteses
- **Machine Learning**: Modelos preditivos, clustering, detecção de anomalias
- **Visualização**: Gráficos estáticos e interativos, dashboards executivos

🎨 **Design & UX**

- **UI/UX Moderno**: Componentes responsivos, design systems
- **Interatividade**: Animações, transições, eventos de usuário
- **Acessibilidade**: ARIA, semântica, navegação por teclado

- **Performance**: Lazy loading, virtual scrolling, otimização

🚀 Deployment & Produção

- **Export**: HTML estático, PDF, PNG, SVG
- **Otimização**: Minificação, compressão, cache
- **Documentação**: Relatórios automáticos, especificações técnicas
- **Qualidade**: Validação de dados, testes, debugging

🎯 Casos de Uso Ideais

Com todas essas capacidades, sou especialmente adequado para:

1. **Business Intelligence Dashboards**
2. **Análise Exploratória de Dados**
3. **Relatórios Executivos Automatizados**
4. **Protótipos de Aplicações Web**
5. **Visualizações Científicas**
6. **Sistemas de Monitoramento**
7. **Análise Financeira e KPIs**
8. **Detecção de Padrões e Anomalias**

⚡ Vantagens Competitivas

- **Integração Completa**: Desde dados brutos até dashboards finais
- **Flexibilidade**: Múltiplas linguagens e frameworks
- **Qualidade**: Validação, testes e documentação automática
- **Performance**: Otimização para grandes datasets
- **Usabilidade**: Interfaces modernas e intuitivas
- **Escalabilidade**: Arquitetura modular e reutilizável

🔄 Workflow Típico

mermaid Visualizar Copiar