

## comandos



## Explicação Detalhada do Código Streamlit



### SEÇÃO 1: IMPORTAÇÕES E CONFIGURAÇÕES

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

#### Explicação:

- `import streamlit as st`: Importa a biblioteca Streamlit para criar aplicações web interativas
- `import pandas as pd`: Importa Pandas para manipulação e análise de dados
- `import plotly.express as px`: Importa Plotly Express para criar gráficos interativos

```
st.set_page_config(layout="wide")
```

#### Explicação:

- Configura o layout da página Streamlit para usar toda a largura da tela
- Por padrão, Streamlit usa layout centralizado; "wide" expande para usar todo o espaço disponível



### SEÇÃO 2: CARREGAMENTO DE DADOS

```
df = pd.read_csv(r"C:\Laboratório\dashboard\supermarket_sales.csv", sep=";", decimal=",")
```

#### Explicação:

- `pd.read_csv()`: Lê um arquivo CSV e cria um DataFrame
- `r"C:\Laboratório\dashboard\supermarket_sales.csv"`: Caminho do arquivo (o 'r' indica raw string, evitando problemas com barras invertidas)
- `sep=";"`: Define o separador como ponto e vírgula (formato brasileiro comum)
- `decimal=", "`: Define a vírgula como separador decimal (padrão brasileiro)



### SEÇÃO 3: CONVERSÃO E TRATAMENTO DE DATAS

```
df["Date"] = pd.to_datetime(df["Date"])
df = df.sort_values("Date")
```

#### Explicação:

- `pd.to_datetime(df["Date"])`: Converte a coluna "Date" de texto para formato de data
- `df.sort_values("Date")`: Ordena o DataFrame pela coluna de data em ordem crescente
- Isso é essencial para visualizações temporais corretas

---

## SEÇÃO 4: CRIAÇÃO DO FILTRO INTERATIVO

```
df["Month"] = df["Date"].dt.to_period('M').astype(str)
month = st.sidebar.selectbox("Mês", df["Month"].unique())
df_filtered = df[df["Month"] == month]
```

### Explicação:

- `df["Date"].dt.to_period('M')` : Extrai o período mensal das datas (formato YYYY-MM)
- `.astype(str)` : Converte para string para melhor exibição
- `st.sidebar.selectbox()` : Cria uma caixa de seleção na barra lateral do Streamlit
- `df["Month"].unique()` : Obtém todos os meses únicos disponíveis no dataset
- `df_filtered = df[df["Month"] == month]` : Filtra o DataFrame apenas para o mês selecionado

---

## SEÇÃO 5: INTERFACE DO USUÁRIO

```
st.title("Dashboard - Visualização da Informação")
st.markdown("Visualizações das três principais unidades: Temporal, Hierárquica e Geográfica."
```

### Explicação:

- `st.title()` : Cria um título principal na página
- `st.markdown()` : Permite usar formatação Markdown (negrito com **texto**)

---

## SEÇÃO 6: VISUALIZAÇÕES TEMPORAIS

```
st.header("1. Visualização Temporal")
col1, col2, col3 = st.columns(3)
```

### Explicação:

- `st.header()` : Cria um cabeçalho de seção
- `st.columns(3)` : Divide a página em 3 colunas iguais para layout lado a lado

### Gráfico 1: Linha - Faturamento diário por cidade

```
fig_line = px.line(
    df_filtered,
    x="Date",
    y="Total",
    color="City",
    title="Linha: Faturamento diário por cidade"
)
col1.plotly_chart(fig_line, use_container_width=True)
```

### Explicação:

- `px.line()` : Cria um gráfico de linha usando Plotly Express
- `x="Date"` : Define o eixo X como datas
- `y="Total"` : Define o eixo Y como valores totais
- `color="City"` : Cria linhas diferentes para cada cidade (cores diferentes)
- `col1.plotly_chart()` : Exibe o gráfico na primeira coluna
- `use_container_width=True` : Ajusta o gráfico à largura da coluna

## Gráfico 2: Área - Acúmulo de vendas

```
df_filtered['Cumulative Total'] = df_filtered.groupby('City')['Total'].cumsum()
fig_area = px.area(
    df_filtered,
    x="Date",
    y="Cumulative Total",
    color="City",
    title="Área: Acúmulo de vendas no mês"
)
col2.plotly_chart(fig_area, use_container_width=True)
```

### Explicação:

- `df_filtered.groupby('City')['Total'].cumsum()` : Calcula o total cumulativo por cidade
  - `groupby('City')` : Agrupa por cidade
  - `['Total']` : Seleciona a coluna Total
  - `.cumsum()` : Calcula a soma cumulativa
- `px.area()` : Cria um gráfico de área preenchida
- Mostra como as vendas se acumulam ao longo do tempo para cada cidade

## Gráfico 3: Barras - Total diário

```
daily_total = df_filtered.groupby('Date')[['Total']].sum().reset_index()
fig_bar = px.bar(
    daily_total,
    x="Date",
    y="Total",
    title="Barra: Faturamento total diário (todas as cidades)"
)
col3.plotly_chart(fig_bar, use_container_width=True)
```

### Explicação:

- `df_filtered.groupby('Date')[['Total']].sum()` : Soma todos os totais por data
- `.reset_index()` : Transforma o índice em coluna normal
- `px.bar()` : Cria um gráfico de barras
- Mostra o faturamento total consolidado de todas as cidades por dia

## SEÇÃO 7: VISUALIZAÇÃO HIERÁRQUICA (TREEMAP)

```
st.header("2. Visualização Hierárquica (Treemap)")

fig_tree = px.treemap(
    df_filtered,
```

```

    path=['City', 'Product line'],
    values='Total',
    title='Treemap: Faturamento por Filial e Categoria'
)
st.plotly_chart(fig_tree, use_container_width=True)

```

#### Explicação:

- `px.treemap()` : Cria uma visualização em formato de mapa de árvore
- `path=['City', 'Product line']` : Define a hierarquia (Cidade > Linha de Produto)
- `values='Total'` : Tamanho dos retângulos baseado nos valores totais
- Treemap mostra proporções de forma visual - retângulos maiores = valores maiores

## SEÇÃO 8: VISUALIZAÇÃO GEOGRÁFICA

```

st.header("3. Visualização Geográfica")

city_coords = {
    "Yangon": {"lat": 16.8409, "lon": 96.1735},
    "Mandalay": {"lat": 21.9588, "lon": 96.0891},
    "Naypyitaw": {"lat": 19.7633, "lon": 96.0785},
}

```

#### Explicação:

- Cria um dicionário com coordenadas geográficas (latitude e longitude) das cidades
- Necessário para posicionar os pontos no mapa mundial

```

df_filtered["lat"] = df_filtered["City"].map(lambda x: city_coords.get(x, {}).get("lat"))
df_filtered["lon"] = df_filtered["City"].map(lambda x: city_coords.get(x, {}).get("lon"))

```

#### Explicação:

- `.map(lambda x: ...)` : Aplica uma função a cada valor da coluna "City"
- `city_coords.get(x, {}).get("lat")` : Busca a latitude da cidade no dicionário
- Se a cidade não existir, retorna None (evita erros)

```

fig_geo = px.scatter_geo(
    df_filtered,
    lat="lat",
    lon="lon",
    color="City",
    size="Total",
    hover_name="City",
    title="Distribuição Geográfica das Vendas (exemplo conceitual)"
)
st.plotly_chart(fig_geo, use_container_width=True)

```

#### Explicação:

- `px.scatter_geo()` : Cria um gráfico de dispersão em mapa mundial
- `lat="lat", lon="lon"` : Define as coordenadas

- `color="City"` : Cores diferentes para cada cidade
- `size="Total"` : Tamanho dos pontos baseado no valor das vendas
- `hover_name="City"` : Mostra o nome da cidade ao passar o mouse

---

## SEÇÃO 9: NOTAS EXPLICATIVAS

```
st.markdown("""
**Notas:**
- **Gráficos temporais:** mostram tendências e evolução do faturamento.
- **Treemap:** exibe a estrutura hierárquica das vendas por filial e categoria.
- **Mapa geográfico:** exemplifica a distribuição espacial das vendas (ajuste lat/lon se necessário).
""")
```






### Explicação:

- Adiciona explicações em markdown para orientar o usuário
- Uso de `"""` permite texto multilinha
- Formatação com `**texto**` para negrito

---

## RESUMO DO QUE FOI FEITO

### Funcionalidades Implementadas:

1.  **Dashboard Interativo:** Interface web com Streamlit
2.  **Filtro por Mês:** Seleção interativa na barra lateral
3.  **Três Tipos de Visualização Temporal:**
  - Linha: Tendências diárias
  - Área: Acumulação progressiva
  - Barras: Totais consolidados
4.  **Visualização Hierárquica:** Treemap mostrando proporções
5.  **Visualização Geográfica:** Mapa com distribuição espacial

### Tecnologias Utilizadas:

- **Streamlit:** Framework para aplicações web
- **Pandas:** Manipulação de dados
- **Plotly:** Gráficos interativos
- **Python:** Linguagem de programação

### Conceitos de Visualização Aplicados:

- **Temporal:** Como dados mudam ao longo do tempo
- **Hierárquica:** Relações entre categorias e subcategorias
- **Geográfica:** Distribuição espacial dos dados

Este código cria um dashboard completo e interativo que permite explorar dados de vendas de supermercado sob diferentes perspectivas, demonstrando os principais tipos de visualização de dados de forma prática e educativa.