📊 Análise da Curva ABC - Documentação Completa

Índice

Visão Geral

O aplicativo **Análise da Curva ABC** é uma ferramenta interativa desenvolvida em Streamlit que auxilia empresas na gestão de seus estoques através da análise da Curva ABC e da clusterização de produtos utilizando o algoritmo K-Means. Além disso, o aplicativo integra-se com o **Google Gemini** para gerar análises detalhadas e relatórios em PDF que proporcionam insights estratégicos para a otimização do estoque e aumento do faturamento.

Características Principais

- **Upload de Planilhas:** Carregue dados de produtos em formatos CSV ou XLSX.
- **Adição Manual de Produtos:** Insira produtos diretamente pela interface intuitiva.
- **Análise Avançada:** Utilize K-Means para identificar padrões e otimizar o estoque.
- **Visualizações Dinâmicas:** Interaja com gráficos detalhados para melhor compreensão.
- **Relatórios Personalizados:** Gere PDFs profissionais com insights acionáveis.
- **Segurança de Dados:** Utilize segredos do Streamlit para proteger chaves de API.

Pré-requisitos

Antes de iniciar, certifique-se de que você possui os seguintes itens:

- **Python 3.7 ou superior** instalado em sua máquina.
- **Chave de API do Google Gemini**.
- **Pacotes Python necessários** instalados (listados na seção de instalação).
- **Arquivo de logo** (`eseg.png`) no diretório raiz do projeto para exibição na interface.

Instalação

Siga os passos abaixo para configurar o ambiente e instalar as dependências necessárias:

1. **Clone o Repositório:**

    ```
    git clone https://github.com/seu-usuario/seu-repositorio.git
    cd seu-repositorio
    ```

2. **Crie e Ative um Ambiente Virtual (Opcional, mas recomendado):**

```
python -m venv venv
source venv/bin/activate  # No Windows: venv\Scripts\activate
```

3. **Instale as Dependências:**

   Certifique-se de que você possui o arquivo `requirements.txt` com as seguintes linhas:

```
pip install pandas numpy matplotlib seaborn streamlit scikit-learn google-generativeai python-dotenv reportlab scipy streamlit
```

   Ou instale diretamente via pip:

```
pip install pandas numpy matplotlib seaborn streamlit scikit-learn google-generativeai python-dotenv reportlab scipy streamlit
```

Configuração de Segredos

Para garantir a segurança da sua chave de API do Google Gemini, utilize o sistema de **Secrets** do Streamlit.

1. Configuração Local

1. **Crie a Pasta `.streamlit`:**

   No diretório raiz do seu projeto, crie uma pasta chamada `.streamlit` se ela ainda não existir.

```
mkdir .streamlit
```

2. **Crie o Arquivo `secrets.toml`:**

   Dentro da pasta `.streamlit`, crie um arquivo chamado `secrets.toml`.

```
touch .streamlit/secrets.toml
```

3. **Adicione o Token do Google Gemini ao `secrets.toml`:**

   Opção 1: Chave de Nível Superior

```
GOOGLE_API_KEY = "SEU_GOOGLE_API_KEY_AQUI"
```

   Opção 2: Estrutura Aninhada

```
[google]
api_key = "SEU_GOOGLE_API_KEY_AQUI"
```

4. **Proteja o Arquivo `secrets.toml`:**

   Adicione o arquivo ao `.gitignore` para evitar que seja versionado.

```
echo ".streamlit/secrets.toml" >> .gitignore
```

2. Configuração no Streamlit Cloud

Ao implantar seu aplicativo no Streamlit Cloud, configure os segredos diretamente na interface da plataforma:

1. **Acesse o Painel do Seu Aplicativo:**
2. **Navegue até a Seção de Segredos:**
   - Clique em **"Manage app"** (Gerenciar aplicativo).
   - Selecione a aba **"Secrets"**.
3. **Adicione os Segredos:**

   Exemplo para Opção 1:

```
GOOGLE_API_KEY = "SEU_GOOGLE_API_KEY_AQUI"
```

   Exemplo para Opção 2:

```
[google]
api_key = "SEU_GOOGLE_API_KEY_AQUI"
```

4. **Salve as Alterações:**

- Clique em **"Save"** (Salvar).
- Aguarde cerca de um minuto para que as alterações se propaguem.

Estrutura do Projeto

A estrutura do seu projeto deve ser organizada da seguinte forma:

```
seu-repositorio/
│
├── .streamlit/
│   └── secrets.toml
│
├── eseg.png
│
├── app.py
│
├── requirements.txt
│
└── README.md
```

- **`.streamlit/secrets.toml`** : Armazena os segredos e chaves de API.
- **`eseg.png`** : Logo da empresa exibido na interface.
- **`app.py`** : Código principal do aplicativo Streamlit.
- **`requirements.txt`** : Lista de dependências do projeto.
- **`README.md`** : Documentação do projeto.

Descrição do Código

A seguir, uma descrição detalhada das principais seções e funções do código.

Importações e Configurações Iniciais

O código começa com a importação das bibliotecas necessárias e a configuração inicial do aplicativo Streamlit.

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import streamlit as st
from streamlit_option_menu import option_menu
from sklearn.cluster import KMeans
import google.generativeai as genai
import time
from io import BytesIO
from reportlab.lib.pagesizes import A4
from reportlab.lib.styles import getSampleStyleSheet, ParagraphStyle
from reportlab.lib.enums import TA_CENTER, TA_LEFT
from reportlab.platypus import SimpleDocTemplate, Paragraph, Spacer, Image, Table, TableStyle, PageBreak
from reportlab.lib import colors
from reportlab.lib.units import inch
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score
from scipy import stats
import re
from streamlit_modal import Modal
import markdown  # Import para conversão de Markdown
```

- **Bibliotecas de Manipulação de Dados e Visualização:**
  - `pandas` , `numpy` : Manipulação de dados.
  - `matplotlib` , `seaborn` : Criação de gráficos.
- **Streamlit e Componentes Adicionais:**
  - `streamlit` : Framework para criação do aplicativo.
  - `streamlit_option_menu` : Criação de menus personalizados.
  - `streamlit_modal` : Criação de modais/pop-ups.
- **Clusterizacão e Análise:**

- `sklearn.cluster.KMeans` : Algoritmo de clusterização.
- `StandardScaler`, `silhouette_score` : Pré-processamento e avaliação de clusters.

- **Geração de Relatórios em PDF:**
  - `reportlab` : Biblioteca para criação de PDFs.
- **Integração com Google Gemini:**
  - `google.generativeai` : API para geração de conteúdo com Google Gemini.

Estilização Personalizada

A função `custom_css` adiciona estilos personalizados ao aplicativo para melhorar a aparência e a usabilidade.

```
def custom_css():
    st.markdown("""
    <style>
    /* Estilos personalizados aqui */
    /* ... */
    </style>
    """, unsafe_allow_html=True)

custom_css()
```

- **Personalizações Incluem:**
  - Estilização da barra lateral.
  - Fundo da página.
  - Rodapé com ícone e tooltip.
  - Estilização de botões.
  - Cabeçalhos e tabelas.
  - Pop-up/modal.

Funções Principais

O aplicativo é organizado em várias funções que encapsulam diferentes funcionalidades. A seguir, são descritas cada uma dessas funções.

Exibição do Logo

Exibe o logo da empresa na barra lateral. Se o arquivo `eseg.png` não for encontrado, exibe um aviso.

```
def exibir_logo():
    img_path = "eseg.png"  # Substitua pelo caminho da sua imagem
    if os.path.exists(img_path):
        st.sidebar.image(img_path, caption="ESEG Corp", use_column_width=True)
    else:
        st.sidebar.warning("Logo não encontrado!")
```

Carregamento da Planilha

Carrega os dados dos produtos a partir de um arquivo CSV ou XLSX.

```
def carregar_planilha(uploaded_file):
    try:
        if uploaded_file.name.endswith('.csv'):
            return pd.read_csv(uploaded_file)
        elif uploaded_file.name.endswith('.xlsx'):
            return pd.read_excel(uploaded_file)
    except Exception as e:
        st.sidebar.error(f"Erro ao carregar a planilha: {e}")
    st.sidebar.error("Formato não suportado. Use .csv ou .xlsx.")
    return None
```

Conversão de Markdown para ReportLab

Converte texto em Markdown para um formato compatível com ReportLab para inclusão no PDF.

```
def markdown_para_reportlab(texto):
    # Converte Markdown para HTML
    html = markdown.markdown(texto)

    # Substituir listas não ordenadas e ordenadas por tags compatíveis
    html = html.replace('<ul>', '<bullet>').replace('</ul>', '</bullet>')
    html = html.replace('<ol>', '<ordered>').replace('</ol>', '</ordered>')
    html = html.replace('<li>', '<bullet>&bull; ').replace('</li>', '</bullet>')

    # Remover outras tags HTML
    clean_text = re.sub('<[^<]+?>', '', html)

    return clean_text
```

Cálculo da Curva ABC

Calcula a Curva ABC dos produtos com base no valor total (Preço * Quantidade).

```
def calcular_curva_abc(df):
    df = df.copy()
    df['Valor Total'] = df['Preço'] * df['Quantidade']
    df_sorted = df.sort_values(by='Valor Total', ascending=False).reset_index(drop=True)
    df_sorted['% Acumulado'] = df_sorted['Valor Total'].cumsum() / df_sorted['Valor Total'].sum() * 100
    df_sorted['Classe'] = 'C'
    df_sorted.loc[df_sorted['% Acumulado'] <= 80, 'Classe'] = 'A'
    df_sorted.loc[(df_sorted['% Acumulado'] > 80) & (df_sorted['% Acumulado'] <= 95), 'Classe'] = 'B'
    return df_sorted
```

Pré-processamento dos Dados

Remove linhas com valores ausentes e outliers utilizando o z-score.

```
def preprocessar_dados(df):
    df = df.dropna(subset=["Preço", "Quantidade"])
    z_scores = np.abs(stats.zscore(df[["Preço", "Quantidade"]]))
    df = df[(z_scores < 3).all(axis=1)]
    return df
```

Determinação do Número Ótimo de Clusters

Determina o número ideal de clusters utilizando o Método do Cotovelo e o Silhouette Score.

```
def determinar_n_clusters(df, max_clusters=10):
    scaler = StandardScaler()
    features = ["Preço", "Quantidade"]
    scaled_features = scaler.fit_transform(df[features])

    sse = []
    silhouette_scores = []
    for k in range(2, max_clusters + 1):
        kmeans = KMeans(n_clusters=k, random_state=42)
        kmeans.fit(scaled_features)
        sse.append(kmeans.inertia_)
        score = silhouette_score(scaled_features, kmeans.labels_)
        silhouette_scores.append(score)

    # Sugestão de Número de Clusters
    best_k = silhouette_scores.index(max(silhouette_scores)) + 2
    st.success(f"Número ótimo de clusters sugerido: *{best_k}*")

    # Plot Método do Cotovelo e Silhouette Score
    fig, ax1 = plt.subplots(figsize=(8, 4))  # Reduzido o tamanho para melhor ajuste
    color = '#27AE60'
    ax1.set_xlabel('Número de Clusters')
    ax1.set_ylabel('SSE', color=color)
    ax1.plot(range(2, max_clusters + 1), sse, marker='o', color=color, label='SSE')
    ax1.tick_params(axis='y', labelcolor=color)
    ax1.legend(loc='upper left')

    ax2 = ax1.twinx()
    color = '#E67E22'
    ax2.set_ylabel('Silhouette Score', color=color)
    ax2.plot(range(2, max_clusters + 1), silhouette_scores, marker='x', color=color, label='Silhouette Score')
    ax2.tick_params(axis='y', labelcolor=color)
    ax2.legend(loc='upper right')

    plt.title('Método do Cotovelo e Silhouette Score', fontsize=14)
    plt.tight_layout()
    st.pyplot(fig)

    return best_k
```

Aplicação do K-Means

Aplica o algoritmo K-Means para clusterizar os produtos e retorna os centróides e o DataFrame com os clusters.

```
def aplicar_kmeans(df, n_clusters=3):
    scaler = StandardScaler()
    features = ["Preço", "Quantidade"]
    scaled_features = scaler.fit_transform(df[features])

    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    df["Cluster"] = kmeans.fit_predict(scaled_features)

    # Centróides na escala original
    centroids = scaler.inverse_transform(kmeans.cluster_centers_)
    centroids_df = pd.DataFrame(centroids, columns=features)
    centroids_df['Cluster'] = range(n_clusters)

    return df, centroids_df
```

Visualização da Distribuição das Classes ABC nos Clusters

Cria uma tabela de contingência (crosstab) para visualizar como as classes ABC estão distribuídas nos clusters.

```
def visualizar_abc_clusters(df):
    crosstab = pd.crosstab(df['Classe'], df['Cluster'])
    return crosstab
```

Geração da Análise com Google Gemini

Integra-se com o Google Gemini para gerar uma análise detalhada dos produtos com base na Curva ABC e nos clusters identificados.

```
def gerar_analise_gemini(df):
    produtos_str = df[['Nome', 'Quantidade', 'Preço', 'Classe', 'Cluster']].to_string(index=False)
    data_atual = time.strftime('%d/%m/%Y às %H:%M')  # Captura a data e hora atual

    # Prompt fixo para análise
    prompt = (
        f"Data e Hora do Relatório: {data_atual}\n\n"
        f"A partir de agora você é um especialista em Supply Chain. "
        f"Faça uma análise detalhada desses produtos com base na curva ABC e nos clusters identificados. "
        f"Explique o que os clusters indicam sobre o comportamento dos produtos e forneça insights "
        f"sobre estoque, faturamento e possíveis sinergias entre produtos do mesmo cluster. "
        f"Seja breve, porém seja extremamente específico com os produtos a seguir:\n{produtos_str}"
    )

    model = genai.GenerativeModel("gemini-1.5-flash")

    # Exibir o spinner antes de gerar a resposta
    with st.spinner("Gerando análise..."):
        response = model.generate_content(prompt)

        # Simulação de digitação da resposta
        resposta_texto = response.text
        resposta_container = st.empty()  # Cria um espaço para a resposta

        # Simulação do efeito de digitação
        for i in range(len(resposta_texto) + 1):
            resposta_container.markdown(resposta_texto[:i], unsafe_allow_html=True)
            time.sleep(0.005)  # Ajuste o tempo para controlar a velocidade da digitação

    # Salvar a análise gerada no estado da sessão
    st.session_state.analise_gemini = resposta_texto
```

Adição Manual de Produtos

Permite que o usuário adicione produtos manualmente através da interface da barra lateral. Utiliza uma função de callback para gerenciar o estado da sessão sem causar erros.

```
def adicionar_produto():
    nome = st.session_state.nome_produto
    preco = st.session_state.preco_produto
    quantidade = st.session_state.quantidade_produto

    if nome and preco > 0 and quantidade > 0:
        if 'produtos' not in st.session_state:
            st.session_state.produtos = []
        st.session_state.produtos.append(
            {"Nome": nome, "Preço": preco, "Quantidade": quantidade}
        )
        st.sidebar.success(f"Produto *{nome}* adicionado com sucesso!")
        # Limpar os campos após adicionar
        st.session_state.nome_produto = ""
        st.session_state.preco_produto = 0.0
        st.session_state.quantidade_produto = 0
    else:
        st.sidebar.error("Preencha todos os campos corretamente.")
```

Salvamento de Gráficos como Imagens

Funções para salvar gráficos em buffers de memória como imagens PNG, que serão incluídas no relatório PDF.

```
def salvar_grafico_pizza(data, titulo):
    buffer = BytesIO()
    fig, ax = plt.subplots(figsize=(3, 3))  # Tamanho reduzido para melhor ajuste
    ax.pie(
        data,
        labels=data.index,
        autopct='%1.1f%%',
        startangle=90,
        colors=sns.color_palette("pastel")[:len(data)],
        wedgeprops={'edgecolor': 'white'}
    )
    ax.set_title(titulo, fontsize=12, color="#27AE60")
    ax.axis('equal')
    plt.tight_layout()
    plt.savefig(buffer, format="PNG")
    buffer.seek(0)
    plt.close(fig)
    return buffer

def salvar_grafico_dispersao(df):
    buffer = BytesIO()
    fig, ax = plt.subplots(figsize=(4, 3))  # Tamanho reduzido para melhor ajuste
    sns.scatterplot(
        x='Quantidade', y='Preço', data=df, hue='Cluster', palette='deep', s=60, edgecolor='white', alpha=0.7, ax=ax
    )
    plt.title('Dispersão de Preço x Quantidade com Clusters', fontsize=12, color="#27AE60")
    plt.xlabel('Quantidade', fontsize=10)
    plt.ylabel('Preço (R$)', fontsize=10)
    plt.legend(title='Cluster', fontsize=8, title_fontsize=10)
    plt.grid(True, linestyle='--', alpha=0.5)
    plt.tight_layout()
    plt.savefig(buffer, format="PNG")
    buffer.seek(0)
    plt.close(fig)
    return buffer
```

Geração do Relatório em PDF

Gera um relatório em PDF que inclui todas as análises, tabelas e gráficos gerados.

```
def gerar_pdf(df, class_counts, class_counts2, analise_texto):
    buffer = BytesIO()
    doc = SimpleDocTemplate(
        buffer,
        pagesize=A4,
        rightMargin=30, leftMargin=30,
        topMargin=30, bottomMargin=18
    )
    elements = []

    # Define os estilos
    styles = getSampleStyleSheet()
    styles.add(ParagraphStyle(
        name='CenterTitle',
        parent=styles['Title'],
        alignment=TA_CENTER,
        fontSize=24,
        spaceAfter=20,
        spaceBefore=20
    ))
    styles.add(ParagraphStyle(
        name='SectionHeader',
```

```
    fontSize=18,
    spaceAfter=10,
    textColor=colors.HexColor("#27AE60"),
    leading=22,
    alignment=TA_LEFT
))
# Renomear 'Heading1' e 'Heading2' para evitar conflito
styles.add(ParagraphStyle(
    name='CustomHeading1',
    parent=styles['Heading1'],
    fontSize=18,
    leading=22,
    spaceAfter=12,
    textColor=colors.HexColor("#27AE60")
))
styles.add(ParagraphStyle(
    name='CustomHeading2',
    parent=styles['Heading2'],
    fontSize=16,
    leading=20,
    spaceAfter=10,
    textColor=colors.HexColor("#27AE60")
))
styles.add(ParagraphStyle(
    name='NormalLeft',
    alignment=TA_LEFT,
    fontSize=12,
    spaceAfter=10,
    leading=15
))
# Renomear o estilo para 'CustomBullet' para evitar conflito
styles.add(ParagraphStyle(
    name='CustomBullet',
    parent=styles['Normal'],
    leftIndent=20,
    bulletIndent=10,
    bulletFontName='Helvetica',
    bulletFontSize=12,
))

# Capa
if os.path.exists("eseg.png"):
    logo = "eseg.png"
    im = Image(logo, 3 * inch, 0.8 * inch)
    im.hAlign = 'CENTER'
    elements.append(im)
    elements.append(Spacer(1, 12))

# Título na capa
elements.append(Paragraph("Relatório de Análise da Curva ABC", styles['CenterTitle']))
elements.append(Spacer(1, 50))
elements.append(PageBreak())

# Sumário Manual
elements.append(Paragraph("Sumário", styles['SectionHeader']))
elements.append(Spacer(1, 12))
sumario = """
1. Introdução ..... 1
2. Metodologia ..... 2
3. Estatísticas Descritivas ..... 3
4. Tabela de Produtos ..... 4
5. Gráficos ..... 5
6. Detalhamento dos Clusters ..... 6
7. Análise Detalhada com Google Gemini ..... 7
8. Recomendações ..... 8
```

```
    9. Referências ..... 9
    """
    for linha in sumario.strip().split('\n'):
        elements.append(Paragraph(linha.strip(), styles['NormalLeft']))
    elements.append(PageBreak())

    # Sumário Executivo
    elements.append(Paragraph("Sumário Executivo", styles['SectionHeader']))
    elements.append(Paragraph(
        "Este relatório apresenta uma análise detalhada da Curva ABC dos produtos, incluindo a clusterização utilizando o método K-M
        styles['NormalLeft']
    ))
    elements.append(PageBreak())

    # **Introdução**
    elements.append(Paragraph("Introdução", styles['SectionHeader']))
    elements.append(Paragraph(
        "A Curva ABC é uma técnica amplamente utilizada na gestão de estoques e processos, permitindo a classificação dos itens com
        styles['NormalLeft']
    ))
    elements.append(PageBreak())

    # **Metodologia**
    elements.append(Paragraph("Metodologia", styles['SectionHeader']))
    elements.append(Paragraph(
        "Para a elaboração deste relatório, adotamos a seguinte metodologia:",
        styles['NormalLeft']
    ))
    metodologia = [
        "1. **Coleta de Dados:** Reunimos informações sobre os produtos, incluindo nome, preço unitário e quantidade em estoque.",
        "2. **Cálculo do Valor Total:** Calculamos o valor total de cada produto multiplicando o preço unitário pela quantidade.",
        "3. **Ordenação Decrescente:** Organizamos os produtos em ordem decrescente de valor total para identificar os itens de maio
        "4. **Cálculo do Percentual Acumulado:** Determinamos o percentual acumulado de cada produto em relação ao valor total acumu
        "5. **Classificação ABC:** Classificamos os produtos em classes A, B ou C com base nos seguintes critérios:",
        "   - **Classe A:** Itens que representam até 80% do valor acumulado.",
        "   - **Classe B:** Itens que representam entre 80% e 95% do valor acumulado.",
        "   - **Classe C:** Itens que representam os 5% restantes do valor acumulado.",
        "6. **Análise de Clusterização (K-Means):** Aplicamos o algoritmo K-Means para identificar agrupamentos naturais entre os pr
        "7. **Interpretação dos Resultados:** Analisamos os clusters e as classes ABC para extrair insights sobre o comportamento do
    ]
    # Texto detalhado para cada tópico
    metodologia_detalhada = {
        "1": """
Reunimos informações detalhadas sobre os produtos comercializados pela empresa. Essa etapa é crucial para garantir que a análise sej

- **Nome do Produto:** Identificação única de cada item, permitindo a distinção clara entre os diferentes produtos no portfólio.
- **Preço Unitário:** Valor monetário pelo qual cada unidade do produto é vendida. Este dado é essencial para o cálculo do faturamen
- **Quantidade em Estoque:** Número de unidades disponíveis de cada produto. Essa informação é fundamental para avaliar a disponibil
        """,
        "2": """
Nesta etapa, calculamos o valor total que cada produto representa no estoque, multiplicando o preço unitário pela quantidade em esto

Este cálculo permite identificar o peso financeiro de cada produto no inventário total da empresa. Produtos com alto valor total pod
        """,
        "3": """
Após calcular o valor total de cada produto, organizamos os itens em ordem decrescente com base nesse valor. Esta ordenação facilita

- Identificar rapidamente os produtos de maior impacto financeiro.
- Priorizar a análise e gestão dos itens mais relevantes.
- Estabelecer uma base para a classificação ABC.

A ordenação é feita utilizando ferramentas de análise de dados que permitem a manipulação eficiente de grandes volumes de informaçõe
        """,
        "4": """
Com os produtos ordenados, calculamos o percentual acumulado do valor total para cada produto em relação ao valor total acumulado de
```

```
Este percentual acumulado ajuda a compreender como cada produto contribui para o valor total do estoque e é fundamental para a defin
        """,
        "5": """
Com base nos percentuais acumulados, classificamos os produtos em três categorias principais, seguindo a metodologia da Curva ABC:

- **Classe A:** Produtos que representam aproximadamente os primeiros 80% do valor acumulado. Geralmente, constituem cerca de 20% do
- **Classe B:** Produtos que contribuem com os próximos 15% do valor acumulado, totalizando até 95% quando somados aos da Classe A.
- **Classe C:** Produtos que representam os últimos 5% do valor acumulado. Apesar de serem numerosos (podendo chegar a 50% dos itens

A classificação é aplicada conforme os seguintes critérios:

- **Classe A:** Percentual acumulado de 0% a 80%.
- **Classe B:** Percentual acumulado acima de 80% até 95%.
- **Classe C:** Percentual acumulado acima de 95% até 100%.

Essa categorização permite alocar recursos de forma eficiente, focando nos produtos que mais influenciam o desempenho financeiro da
        """,
        "6": """
Para aprofundar a análise e identificar padrões ocultos nos dados, aplicamos o algoritmo de clusterização K-Means. Este método agrup

- **Seleção das Variáveis de Análise:** Utilizamos as variáveis 'Preço Unitário' e 'Quantidade em Estoque' para capturar tanto o val
- **Normalização dos Dados:** Padronizamos as variáveis para eliminar diferenças de escala que possam influenciar os resultados. A n
- **Determinação do Número Ótimo de Clusters:** Utilizamos métodos estatísticos, como o método do cotovelo (Elbow Method) e o coefic
- **Aplicação do Algoritmo K-Means:** Com o número de clusters definido, aplicamos o K-Means para segmentar os produtos. O algoritmo
- **Análise dos Clusters Formados:** Avaliamos as características de cada cluster, como médias e dispersões, para interpretar os gru

A clusterização complementa a análise ABC, oferecendo uma visão multidimensional dos produtos e auxiliando na elaboração de estratég
        """,
        "7": """
Com os produtos classificados e agrupados, procedemos à interpretação detalhada dos resultados:

- **Análise da Distribuição das Classes ABC nos Clusters:** Verificamos como os produtos das classes A, B e C estão distribuídos ent
- **Identificação de Padrões e Tendências:** Avaliamos se existem tendências, como produtos de baixo preço e alta quantidade em um c
- **Insights para Gestão de Estoque:** Compreendemos quais clusters requerem maior atenção em termos de reposição de estoque, negoci
- **Oportunidades de Sinergia e Otimização:** Identificamos possibilidades de agrupar produtos para promoções conjuntas, otimizar lo
- **Avaliação de Riscos:** Reconhecemos produtos ou clusters que possam representar riscos, como excesso de estoque em itens de baix

A interpretação dos resultados é fundamental para transformar a análise em ações estratégicas. Envolve a colaboração entre diferente
        """
    }

    for passo in metodologia:
        match = re.match(r'^(\d+)\.\s(.*)', passo.strip())
        if match:
            num = match.group(1)
            title = match.group(2)
            elements.append(Paragraph(f"<b>{num}. {title}</b>", styles['NormalLeft']))
            elements.append(Spacer(1, 6))
            if num in metodologia_detalhada:
                detalhe = metodologia_detalhada[num]
                # Processar formatações dentro do texto
                detalhe = re.sub(r'\*\*(.*?)\*\*', r'<b>\1</b>', detalhe)
                detalhe = re.sub(r'\*(.*?)\*', r'<i>\1</i>', detalhe)
                # Adicionar quebras de linha
                detalhe = detalhe.replace('\n', '<br/>')
                elements.append(Paragraph(detalhe, styles['NormalLeft']))
                elements.append(Spacer(1, 12))
        else:
            # Processar subitens (por exemplo, itens com '   - ')
            subitem_match = re.match(r'^\s+-\s(.*)', passo.strip())
            if subitem_match:
                subtext = subitem_match.group(1)
                subtext = re.sub(r'\*\*(.*?)\*\*', r'<b>\1</b>', subtext)
                subtext = re.sub(r'\*(.*?)\*', r'<i>\1</i>', subtext)
                elements.append(Paragraph(subtext, styles['NormalLeft'], bulletText='•'))
```

```
            elements.append(Spacer(1, 6))
        else:
            elements.append(Paragraph(passo.strip(), styles['NormalLeft']))
            elements.append(Spacer(1, 6))
elements.append(PageBreak())

# Estatísticas Descritivas
elements.append(Paragraph("Estatísticas Descritivas", styles['SectionHeader']))
descr = df[['Preço', 'Quantidade']].describe().round(2)
data = [['Métrica', 'Preço (R$)', 'Quantidade']]
for index, row in descr.iterrows():
    data.append([index, row['Preço'], row['Quantidade']])
table = Table(data, hAlign='LEFT')
table_style = TableStyle([
    ('BACKGROUND', (0, 0), (-1, 0), colors.HexColor("#27AE60")),
    ('TEXTCOLOR', (0, 0), (-1, 0), colors.white),
    ('ALIGN', (1, 1), (-1, -1), 'CENTER'),
    ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
    ('FONTSIZE', (0, 0), (-1, 0), 12),
    ('BOTTOMPADDING', (0, 0), (-1, 0), 12),
    ('GRID', (0, 0), (-1, -1), 0.5, colors.grey),
])
table.setStyle(table_style)

# Alternar cores das linhas
for i in range(1, len(data)):
    bg_color = colors.lightgrey if i % 2 == 0 else colors.whitesmoke
    table_style.add('BACKGROUND', (0, i), (-1, i), bg_color)
table.setStyle(table_style)

elements.append(table)
elements.append(PageBreak())

# Tabela de Produtos Detalhada
elements.append(Paragraph("Tabela de Produtos", styles['SectionHeader']))
data = [['Nome', 'Preço (R$)', 'Quantidade', 'Valor Total (R$)', '% Acumulado', 'Classe', 'Cluster']]
for index, row in df.iterrows():
    data.append([
        row['Nome'],
        f"{row['Preço']:.2f}",
        row['Quantidade'],
        f"{row['Valor Total']:.2f}",
        f"{row['% Acumulado']:.2f}%",
        row['Classe'],
        row['Cluster']
    ])

table = Table(data, hAlign='LEFT', repeatRows=1)
table_style = TableStyle([
    ('BACKGROUND', (0, 0), (-1, 0), colors.HexColor("#27AE60")),
    ('TEXTCOLOR', (0, 0), (-1, 0), colors.white),
    ('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),
    ('GRID', (0, 0), (-1, -1), 0.5, colors.grey),
])
table.setStyle(table_style)

# Alternar cores das linhas
for i in range(1, len(data)):
    bg_color = colors.lightgrey if i % 2 == 0 else colors.whitesmoke
    table_style.add('BACKGROUND', (0, i), (-1, i), bg_color)
table.setStyle(table_style)

elements.append(table)
```

```
elements.append(table)
elements.append(PageBreak())

# Gráficos
elements.append(Paragraph("Gráficos", styles['SectionHeader']))

# Distribuição das Classes
elements.append(Paragraph("Distribuição das Classes (A, B, C)", styles['SectionHeader']))
grafico_pizza_classe = salvar_grafico_pizza(class_counts, "Distribuição das Classes (A, B, C)")
im_pizza_classe = Image(grafico_pizza_classe, 3 * inch, 3 * inch)  # Tamanho reduzido
im_pizza_classe.hAlign = 'CENTER'
elements.append(im_pizza_classe)
elements.append(Spacer(1, 12))

# Distribuição dos Clusters
elements.append(Paragraph("Distribuição dos Clusters", styles['SectionHeader']))
grafico_pizza_cluster = salvar_grafico_pizza(class_counts2, "Distribuição dos Clusters")
im_pizza_cluster = Image(grafico_pizza_cluster, 3 * inch, 3 * inch)  # Tamanho reduzido
im_pizza_cluster.hAlign = 'CENTER'
elements.append(im_pizza_cluster)
elements.append(PageBreak())

# Gráfico de Dispersão
elements.append(Paragraph("Gráfico de Dispersão de Clusters", styles['SectionHeader']))
grafico_dispersao = salvar_grafico_dispersao(df)
im_dispersao = Image(grafico_dispersao, 4 * inch, 3 * inch)  # Tamanho reduzido
im_dispersao.hAlign = 'CENTER'
elements.append(im_dispersao)
elements.append(PageBreak())

# Detalhamento dos Clusters
elements.append(Paragraph("Detalhamento dos Clusters", styles['SectionHeader']))
centroids = df.groupby('Cluster').agg({
    'Preço': 'mean',
    'Quantidade': 'mean',
    'Valor Total': 'mean'
}).round(2)
centroids.reset_index(inplace=True)
data = [['Cluster', 'Preço Médio (R$)', 'Quantidade Média', 'Valor Total Médio (R$)']]
for index, row in centroids.iterrows():
    data.append([
        row['Cluster'],
        f"{row['Preço']:.2f}",
        row['Quantidade'],
        f"{row['Valor Total']:.2f}"
    ])
table = Table(data, hAlign='LEFT')
table_style = TableStyle([
    ('BACKGROUND', (0, 0), (-1, 0), colors.HexColor("#27AE60")),
    ('TEXTCOLOR', (0, 0), (-1, 0), colors.white),
    ('ALIGN', (1, 1), (-1, -1), 'CENTER'),
    ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
    ('FONTSIZE', (0, 0), (-1, 0), 12),
    ('BOTTOMPADDING', (0, 0), (-1, 0), 12),
    ('GRID', (0, 0), (-1, -1), 0.5, colors.grey),
])
table.setStyle(table_style)

# Alternar cores das linhas
for i in range(1, len(data)):
    bg_color = colors.lightgrey if i % 2 == 0 else colors.whitesmoke
    table_style.add('BACKGROUND', (0, i), (-1, i), bg_color)
table.setStyle(table_style)

elements.append(table)
elements.append(PageBreak())
```

```
# Análise Detalhada
elements.append(Paragraph("Análise Detalhada com Google Gemini", styles['SectionHeader']))

# Processar 'analise_texto' para aplicar formatação
lines = analise_texto.strip().split('\n')

for line in lines:
    line = line.strip()
    if not line:
        elements.append(Spacer(1, 12))
        continue

    # Processar cabeçalhos
    if line.startswith('## '):
        header_text = line[3:].strip()
        elements.append(Paragraph(header_text, styles['CustomHeading2']))
    elif line.startswith('# '):
        header_text = line[2:].strip()
        elements.append(Paragraph(header_text, styles['CustomHeading1']))
    else:
        # Processar negrito '**texto**' e itálico '*texto*'
        line = re.sub(r'\*\*(.*?)\*\*', r'<b>\1</b>', line)
        line = re.sub(r'\*(.*?)\*', r'<i>\1</i>', line)
        # Processar listas não ordenadas
        if line.startswith('- '):
            bullet_text = line[2:].strip()
            elements.append(Paragraph(bullet_text, styles['NormalLeft'], bulletText='•'))
            elements.append(Spacer(1, 6))
        # Processar listas ordenadas
        elif re.match(r'^\d+\.\s', line):
            match = re.match(r'^(\d+)\.\s(.*)', line)
            num = match.group(1)
            text = match.group(2)
            elements.append(Paragraph(text.strip(), styles['NormalLeft'], bulletText=f'{num}.'))
            elements.append(Spacer(1, 6))
        else:
            # Parágrafo normal
            elements.append(Paragraph(line, styles['NormalLeft']))
            elements.append(Spacer(1, 6))
elements.append(PageBreak())

# Recomendações
elements.append(Paragraph("Recomendações", styles['SectionHeader']))
elements.append(Paragraph(
    "Com base na análise realizada, recomendamos as seguintes ações para otimizar o gerenciamento de estoque e aumentar o fatura
    styles['NormalLeft']
))
elements.append(Spacer(1, 12))
recomendacoes = [
    "1. *Foco nos Produtos Classe A:* Priorizar o gerenciamento e controle de estoque dos produtos classificados como Classe A,
    "2. *Promoções para Produtos Classe B:* Implementar estratégias de marketing e promoções para os produtos Classe B a fim de
    "3. *Redução de Estoque de Classe C:* Considerar a redução do estoque ou descontinuação dos produtos Classe C que não contri
    "4. *Sinergias entre Clusters:* Identificar produtos dentro dos mesmos clusters que possam ser vendidos em conjunto para aum
    "5. *Revisão Periódica:* Realizar análises periódicas da Curva ABC e dos clusters para ajustar as estratégias conforme as mu
]

for rec in recomendacoes:
    match = re.match(r'^(\d+)\.\s(.*)', rec.strip())
    if match:
        num = match.group(1)
        text = match.group(2)
        # Processar itálico dentro do texto
        text = re.sub(r'\*(.*?)\*', r'<i>\1</i>', text)
        elements.append(Paragraph(text, styles['NormalLeft'], bulletText=f'{num}.'))
```

```
            elements.append(Spacer(1, 6))
        else:
            elements.append(Paragraph(rec.strip(), styles['NormalLeft']))
            elements.append(Spacer(1, 6))
    elements.append(PageBreak())

    # Referências
    elements.append(Paragraph("Referências", styles['SectionHeader']))
    referencias = [
        "- Metodologia ABC: https://pt.wikipedia.org/wiki/An%C3%A1lise_ABC",
        "- K-Means Clustering: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html",
        "- Ballou, R. H. (2006). Gerenciamento da Cadeia de Suprimentos/Logística Empresarial. Bookman.",
        "- Slack, N., Chambers, S., & Johnston, R. (2009). Administração da Produção. Atlas.",
        "- Chopra, S., & Meindl, P. (2016). Gerenciamento da Cadeia de Suprimentos: Estratégia, Planejamento e Operação. Pearson.",
        "- Análise ABC: https://pt.wikipedia.org/wiki/An%C3%A1lise_ABC",
        "- Supply Chain Management: https://pt.wikipedia.org/wiki/Gest%C3%A3o_da_cadeia_de_suprimentos"
    ]

    for ref in referencias:
        elements.append(Paragraph(ref.strip('- '), styles['NormalLeft'], bulletText='•'))
        elements.append(Spacer(1, 6))
    elements.append(PageBreak())

    # Rodapé com data e número de página
    def add_footer(canvas_obj, doc_obj):
        page_num = canvas_obj.getPageNumber()
        page_text = f"Página {page_num}"
        date_text = f"Data: {time.strftime('%d/%m/%Y às %H:%M')}"

        canvas_obj.setFont('Helvetica', 10)
        canvas_obj.setFillColor(colors.grey)

        # Posiciona a data no centro, na parte inferior da página
        canvas_obj.drawCentredString(A4[0] / 2.0, 0.5 * inch, date_text)
        # Posiciona o número da página abaixo da data
        canvas_obj.drawCentredString(A4[0] / 2.0, 0.35 * inch, page_text)

    # Construção do documento com o novo rodapé
    doc.build(elements, onFirstPage=add_footer, onLaterPages=add_footer)
    buffer.seek(0)
    return buffer.getvalue()
```

Download do PDF na Sidebar

Permite que o usuário baixe o relatório em PDF gerado através de um botão na barra lateral.

```
def baixar_pdf_sidebar():
    if 'pdf' in st.session_state:
        st.sidebar.download_button(
            label="⬇️ Baixar Relatório em PDF",
            data=st.session_state['pdf'],
            file_name="relatorio_curva_abc.pdf",
            mime="application/pdf",
            key='download_pdf_sidebar'
        )
    else:
        st.sidebar.info("Gerar o relatório na aba 'Análise Gemini' para disponibilizar o download.")
```

Adição do Rodapé com Ícone e Tooltip

Adiciona um rodapé fixo na aplicação com um ícone que exibe um tooltip ao ser passado o mouse.

```
def adicionar_footer():
    footer_html = """
    <div class="footer">
        <div class="tooltip">
            ⚡
            <span class="tooltiptext">Powered by ALJ Corp</span>
        </div>
    </div>
    """
    st.markdown(footer_html, unsafe_allow_html=True)
```

Exibição do Pop-up de Boas-vindas

Exibe um modal de boas-vindas quando o usuário acessa o aplicativo pela primeira vez.

```
def exibir_pop_up():
    if 'show_modal' not in st.session_state:
        st.session_state.show_modal = True

    modal = Modal(title="Bem-vindo ao Análise Curva ABC! ", key="welcome_modal")
    if st.session_state.show_modal:
        with modal.container():
            # Corpo do Modal
            st.markdown("""
                <div class='modal-body'>
                    <p>
                        Este aplicativo foi desenvolvido para auxiliar na análise da Curva ABC dos seus produtos, proporcionando ins
                    </p>
                    <p>
                        <strong>Funcionalidades Principais:</strong>
                    </p>
                    <ul>
                        <li> <strong>Upload de Planilhas:</strong> Carregue seus dados em formato CSV ou XLSX.</li>
                        <li> <strong>Adição Manual:</strong> Insira produtos diretamente pela interface intuitiva.</li>
                        <li> <strong>Análise Avançada:</strong> Utilize K-Means para identificar padrões e otimizar seu estoque.</l
                        <li> <strong>Visualizações Dinâmicas:</strong> Interaja com gráficos detalhados para melhor compreensão.</l
                        <li> <strong>Relatórios Personalizados:</strong> Gere PDFs profissionais com insights acionáveis.</li>
                </div>
            """, unsafe_allow_html=True)

            # Rodapé do Modal com botão
            if st.button(" Começar"):
                st.session_state.show_modal = False
                modal.close()
```

Exibição do Menu Principal

Cria um menu horizontal para navegar entre diferentes abas do aplicativo.

```
def menu_principal():
    selected = option_menu(
        menu_title=None,
        options=["Análise Clusters", "Visualizações", "Análise Gemini"],
        icons=["bar-chart", "graph-up", "file-earmark-text"],
        menu_icon="cast",
        default_index=0,
        orientation="horizontal",
        styles={
            "container": {"padding": "0!important", "background-color": "#ffffff"},
            "icon": {"color": "#27AE60", "font-size": "18px"},
            "nav-link": {"font-size": "16px", "text-align": "center", "margin": "0px", "--hover-color": "#27AE60"},
            "nav-link-selected": {"background-color": "#d1e7dd"},
        }
    )
    return selected
```

Inicialização dos Campos de Entrada no `session_state`

Garante que os campos de entrada estejam inicializados no `session_state` para evitar erros.

```
# Inicializar os campos de entrada no session_state se não existirem
if 'nome_produto' not in st.session_state:
    st.session_state.nome_produto = ""
if 'preco_produto' not in st.session_state:
    st.session_state.preco_produto = 0.0
if 'quantidade_produto' not in st.session_state:
    st.session_state.quantidade_produto = 0
```

Interface Principal

Organiza a interface principal do aplicativo, chamando as funções de exibição do logo e do pop-up, e definindo o menu principal.

```
# Interface Principal
exibir_logo()
exibir_pop_up()
selected = menu_principal()
```

Conteúdo das Abas

Define o conteúdo de cada aba selecionada no menu principal.

```
# Conteúdo das abas
if selected == "Análise Clusters":
    st.markdown("<div class='header-title'> Análise da Curva ABC</div>", unsafe_allow_html=True)
    # Adicionando o botão dentro da aba
    if st.button(" Determinar Número Ótimo de Clusters"):
        if 'produtos' in st.session_state and st.session_state.produtos:
            df = pd.DataFrame(st.session_state.produtos)
            df_abc = calcular_curva_abc(df)
            df_abc = preprocessar_dados(df_abc)
            best_k = determinar_n_clusters(df_abc)
            st.session_state.best_k = best_k
        else:
            st.error("Nenhum produto foi adicionado.")

elif selected == "Visualizações":
    st.markdown("<div class='header-title'> Visualizações</div>", unsafe_allow_html=True)
    if 'produtos' in st.session_state and st.session_state.produtos:
        df = pd.DataFrame(st.session_state.produtos)
        df_abc = calcular_curva_abc(df)
        df_abc = preprocessar_dados(df_abc)
        n_clusters = st.session_state.get('best_k', 3)
        df_clusterizado, centroids_df = aplicar_kmeans(df_abc, n_clusters=n_clusters)
```

```
# Atualizar o estado com o DataFrame clusterizado
st.session_state.df_clusterizado = df_clusterizado

# Verificar se 'Valor Total' existe
if 'Valor Total' not in df_clusterizado.columns:
    st.error("A coluna 'Valor Total' está faltando no DataFrame clusterizado.")
    st.stop()

# Definir crosstab antes de usar
crosstab = pd.crosstab(df_clusterizado['Classe'], df_clusterizado['Cluster'])

# Distribuição das Classes e Centróides em colunas lado a lado
col1, col2 = st.columns(2)

with col1:
    # Tabela de Produtos
    st.markdown("### □ Tabela de Produtos")
    st.dataframe(
        df_clusterizado[['Nome', 'Quantidade', 'Preço', 'Valor Total', '% Acumulado', 'Classe', 'Cluster']]
    )

with col2:
    st.subheader('□ Centróides dos Clusters')
    st.dataframe(centroids_df)

# Gráficos adicionais
st.markdown("### □ Gráficos Adicionais")

# Criar duas linhas de gráficos com duas colunas cada para acomodar quatro gráficos
col3, col4 = st.columns(2)
col5, col6 = st.columns(2)

# Contar a distribuição de classes
class_counts = df_clusterizado['Classe'].value_counts()

# Contar a distribuição de Clusters
class_counts2 = df_clusterizado['Cluster'].value_counts()

with col3:
    st.subheader('□ Classes ABC')
    fig, ax = plt.subplots(figsize=(3, 3))  # Tamanho reduzido
    ax.pie(
        class_counts,
        labels=class_counts.index,
        autopct='%1.1f%%',
        startangle=90,
        colors=sns.color_palette("pastel")[:len(class_counts)],
        wedgeprops={'edgecolor': 'white'}
    )
    ax.axis('equal')
    st.pyplot(fig)

with col4:
    st.subheader('□ Clusters')
    fig, ax = plt.subplots(figsize=(3, 3))  # Tamanho reduzido
    ax.pie(
        class_counts2,
        labels=class_counts2.index,
        autopct='%1.1f%%',
        startangle=90,
        colors=sns.color_palette("Set2")[:len(class_counts2)],
        wedgeprops={'edgecolor': 'white'}
    )
    ax.axis('equal')
    st.pyplot(fig)
```

```
            st.pyplot(fig)

        with col5:
            st.subheader('📉 Dispersão')
            fig2, ax2 = plt.subplots(figsize=(3, 3))  # Tamanho reduzido
            sns.scatterplot(
                x='Quantidade',
                y='Preço',
                data=df_clusterizado,
                hue='Cluster',
                palette='deep',
                s=60,  # Tamanho reduzido
                edgecolor='white',
                alpha=0.7,
                ax=ax2
            )
            ax2.set_title("Dispersão de Preço x Quantidade", fontsize=10)
            ax2.grid(True, linestyle='--', alpha=0.5)
            st.pyplot(fig2)

        with col6:
            st.subheader('🔥 Heatmap Classes ABC nos Clusters')
            fig, ax = plt.subplots(figsize=(6, 4))
            sns.heatmap(crosstab, annot=True, fmt='d', cmap='Greens', ax=ax)
            ax.set_title("Distribuição das Classes ABC nos Clusters", fontsize=14)
            st.pyplot(fig)

    else:
        st.warning("Adicione ou faça upload de produtos para visualizar as análises.")

elif selected == "Análise Gemini":
    st.markdown("<div class='header-title'>🤖 Análise Gemini</div>", unsafe_allow_html=True)
    # Adicionando o botão dentro da aba
    if st.button("📄 Gerar Relatório"):
        if 'produtos' in st.session_state and st.session_state.produtos:
            df = pd.DataFrame(st.session_state.produtos)
            df_abc = calcular_curva_abc(df)
            df_abc = preprocessar_dados(df_abc)

            # Determinar o número de clusters
            n_clusters = st.session_state.get('best_k', 3)
            df_clusterizado, centroids_df = aplicar_kmeans(df_abc, n_clusters=n_clusters)

            # Verificar se 'Valor Total' existe
            if 'Valor Total' not in df_clusterizado.columns:
                st.error("A coluna 'Valor Total' está faltando no DataFrame clusterizado.")
                st.stop()

            # Contar a distribuição de classes
            class_counts = df_clusterizado['Classe'].value_counts()

            # Contar a distribuição de Clusters
            class_counts2 = df_clusterizado['Cluster'].value_counts()

            # Visualizações (já incluídas no relatório)
            crosstab = visualizar_abc_clusters(df_clusterizado)

            # Gerar análise com Gemini
            gerar_analise_gemini(df_clusterizado)

            # Gerar o PDF
            pdf = gerar_pdf(df_clusterizado, class_counts, class_counts2, st.session_state.analise_gemini)

            # Armazenar no session_state
            st.session_state['pdf'] = pdf
```

```
            st.success("Relatório gerado com sucesso! Você pode baixá-lo na barra lateral.")
        else:
            st.error("Adicione ou faça upload de produtos antes de gerar o relatório.")
    else:
        st.info("Clique no botão para gerar o relatório em PDF.")
```

Como Usar

A seguir, uma orientação passo a passo sobre como utilizar o aplicativo.

Upload de Planilha

1. **Acesse a Barra Lateral:**
    - Na parte esquerda da interface, localize a seção "**Upload de Planilha**".
2. **Escolha o Arquivo:**
    - Clique no botão de upload e selecione um arquivo no formato **CSV** ou **XLSX** contendo os dados dos produtos.
    - O arquivo deve conter as seguintes colunas:
        - **Nome:** Nome do produto.
        - **Preço:** Preço unitário do produto.
        - **Quantidade:** Quantidade em estoque do produto.
3. **Confirmação:**
    - Após o upload bem-sucedido, uma mensagem de sucesso será exibida.

Adição Manual de Produtos

1. **Expandir o Gerenciamento de Produtos:**
    - Na barra lateral, expanda a seção "**Gerenciamento de Produtos**" clicando no cabeçalho.
2. **Preencher os Campos:**
    - **Nome do Produto:** Insira o nome do produto.
    - **Preço (R$):** Insira o preço unitário do produto.
    - **Quantidade:** Insira a quantidade em estoque.
3. **Adicionar o Produto:**
    - Clique no botão "**Adicionar Produto**".
    - Se os campos forem preenchidos corretamente, uma mensagem de sucesso será exibida e os campos serão limpos automaticamente.

Análise de Clusters

1. **Selecionar a Aba "Análise Clusters":**
    - No menu principal horizontal, clique em "**Análise Clusters**".
2. **Determinar o Número Ótimo de Clusters:**
    - Clique no botão "**Determinar Número Ótimo de Clusters**".
    - O aplicativo calculará o número ideal de clusters utilizando o Método do Cotovelo e o Silhouette Score.
    - O número sugerido será exibido e os gráficos correspondentes serão gerados.

Visualizações

1. **Selecionar a Aba "Visualizações":**
    - No menu principal horizontal, clique em "**Visualizações**".
2. **Visualizar Tabelas e Gráficos:**
    - **Tabela de Produtos:** Exibe uma tabela detalhada com todos os produtos, suas quantidades, preços, valores totais, percentuais acumulados, classes ABC e clusters.
    - **Centróides dos Clusters:** Exibe uma tabela com os centróides de cada cluster.
    - **Gráficos Adicionais:**
        - **Classes ABC:** Pie chart mostrando a distribuição das classes A, B e C.
        - **Clusters:** Pie chart mostrando a distribuição dos clusters.
        - **Dispersão:** Scatter plot mostrando a relação entre preço e quantidade, colorido por cluster.
        - **Heatmap:** Heatmap mostrando a distribuição das classes ABC nos clusters.

Geração do Relatório em PDF

1. **Selecionar a Aba "Análise Gemini":**
    - No menu principal horizontal, clique em "**Análise Gemini**".
2. **Gerar o Relatório:**

- Clique no botão "**Gerar Relatório**".
- O aplicativo irá gerar uma análise detalhada com o auxílio do Google Gemini e compilar todas as informações, tabelas e gráficos em um relatório PDF.
- Após a geração, uma mensagem de sucesso será exibida.

3. **Baixar o Relatório:**

   - Navegue até a barra lateral.
   - Clique no botão "**Baixar Relatório em PDF**" para baixar o relatório gerado.

Manutenção e Atualizações

Para manter o aplicativo funcionando corretamente e atualizado, siga estas orientações:

1. **Atualização das Dependências:**

   - Periodicamente, verifique se há atualizações para as bibliotecas utilizadas.
   - Atualize o arquivo `requirements.txt` conforme necessário.

2. **Gerenciamento de Segredos:**

   - Nunca compartilhe ou versione o arquivo `secrets.toml`.
   - Atualize os segredos através da interface do Streamlit Cloud ao implantar o aplicativo.

3. **Backups:**

   - Mantenha backups regulares dos dados dos produtos para evitar perda de informações.

4. **Monitoramento de Performance:**

   - Utilize ferramentas de monitoramento para acompanhar o desempenho do aplicativo e identificar possíveis gargalos.

5. **Feedback dos Usuários:**

   - Colete feedback dos usuários para aprimorar as funcionalidades e a usabilidade do aplicativo.

Dicas de Depuração

Caso encontre problemas durante o uso ou desenvolvimento do aplicativo, considere as seguintes dicas:

1. **Erros de Carregamento da Planilha:**

   - Verifique se o arquivo está no formato correto (CSV ou XLSX).
   - Assegure-se de que as colunas necessárias estão presentes e nomeadas corretamente.

2. **Problemas com Clusterização:**

   - Certifique-se de que há dados suficientes para a clusterização.
   - Verifique se o número de clusters sugerido é adequado para os dados.

3. **Geração de PDF:**

   - Assegure-se de que todas as dependências da `reportlab` estão instaladas corretamente.
   - Verifique se o arquivo de logo (`eseg.png`) está presente no diretório raiz.

4. **Erros de API do Google Gemini:**

   - Verifique se a chave de API está correta e ativa.
   - Consulte a documentação do Google Gemini para garantir que a integração está correta.

5. **Logs do Streamlit:**

   - Utilize os logs do Streamlit para identificar e resolver erros específicos.
   - Execute o aplicativo localmente para facilitar a depuração.

Considerações de Segurança

A segurança dos dados e das credenciais é primordial. Siga estas práticas para garantir a proteção das informações:

1. **Segredos Protegidos:**

   - Utilize o sistema de **Secrets** do Streamlit para armazenar chaves de API e outras informações sensíveis.
   - Nunca exponha segredos no código-fonte ou em repositórios públicos.

2. **Validação de Entrada:**

   - Implemente validações rigorosas para os dados de entrada dos usuários para evitar injeções e outros tipos de ataques.

3. **Atualizações de Segurança:**

   - Mantenha todas as bibliotecas e dependências atualizadas para proteger contra vulnerabilidades conhecidas.

4. **Controle de Acesso:**

   - Limite o acesso ao aplicativo e aos dados a usuários autorizados.

Licença

Este projeto está licenciado sob a MIT License.

Referências

- **Streamlit Documentation:** https://docs.streamlit.io
- **Google Gemini API:** https://developers.google.com/gemini
- **ReportLab Documentation:** https://www.reportlab.com/docs/reportlab-userguide.pdf
- **Scikit-learn K-Means:** https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
- **Análise ABC:** https://pt.wikipedia.org/wiki/An%C3%A1lise_ABC
- **Gestão da Cadeia de Suprimentos:** https://pt.wikipedia.org/wiki/Gest%C3%A3o_da_cadeia_de_suprimentos

---

Contribuição

Contribuições são bem-vindas! Sinta-se à vontade para abrir issues ou pull requests no repositório.