Projeto

Definição:

Nosso projeto visa facilitar a tomada de decisões de uma loja, partindo da criação de modelos de IA baseados no banco de dados da própria empresa.

Ferramentas utilizadas:

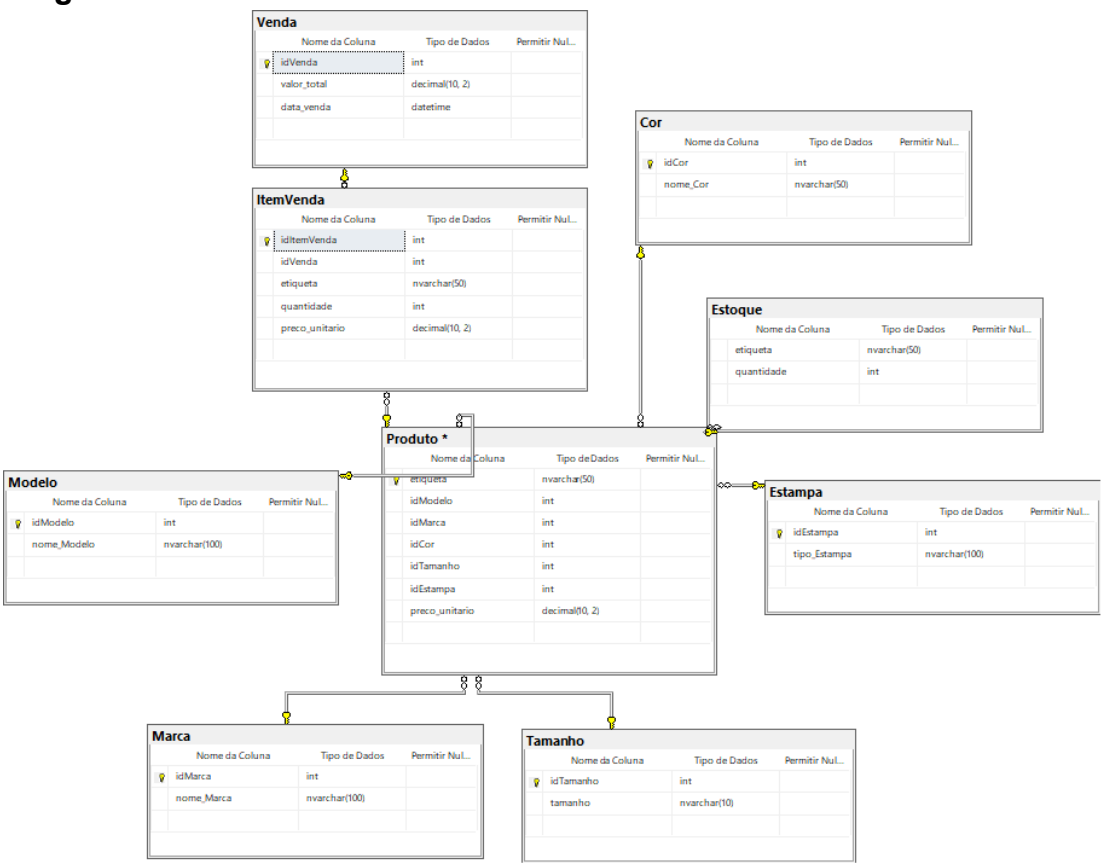
SQL Server, Canvas, Word, Visual Studio Code;

- **Linguagens:**
    - SQL para desenvolvimento e utilização do banco de dados
    - Python para desenvolvimento, utilização do modelo de IA, integração com o banco de dados e sua interface.
    - HTML/CSS para a criação e manutenção da interface

Modelagem e criação do banco:

Definição do tipo de loja: Uma pequena loja de roupas

Diagrama do banco



Integração do banco com o código em Python:

```
def connect_db():
conn = pyodbc.connect('DRIVER={ODBC Driver 17 for SQL Server};'
                      'SERVER=localhost;'
                      'DATABASE=BANCOPI;'
                      'Trusted_Connection=yes;'
                      'Encrypt=yes;'
                      'TrustServerCertificate=yes;')
return conn
```

connect_db(): Função que conecta ao banco de dados BANCOPI no servidor local usando o driver ODBC. Ela retorna uma conexão ativa para ser usada nas operações de banco de dados.

Criação do modelo IA:

Sugestão do chat:

No documento fornecido, a abordagem de criação de modelos de IA é descrita como baseada nos dados do banco de dados da loja, utilizando ferramentas como Python e SQL Server(Escopo do PI).

No entanto, o documento não menciona explicitamente quais modelos de IA específicos seriam mais adequados ou foram utilizados. No contexto de problemas comuns de tomada de decisão em uma loja, os melhores modelos de IA geralmente dependem do tipo de análise e previsão que se pretende realizar. Abaixo estão algumas sugestões de modelos de IA que podem ser úteis com base em cenários comuns de lojas**:**

1. Modelos de Recomendação

- **Modelo sugerido:** Sistemas de recomendação colaborativa (ou filtragem colaborativa).
- **Finalidade:** Recomendação de produtos aos clientes com base em suas preferências ou compras anteriores. Modelos como SVD (Singular Value Decomposition) ou KNN (K-nearest neighbors) são comumente usados nesse caso.

2. Análise de Previsão de Demanda

- **Modelos sugeridos:**
    - **Regressão Linear:** Para prever vendas futuras com base em dados históricos de vendas.
    - **Modelos de Séries Temporais (ARIMA, Prophet):** Para prever vendas diárias, semanais ou mensais.
- **Finalidade:** Previsão de demanda futura para otimização de estoque.

3. Classificação de Produtos

- **Modelo sugerido:** Classificadores de Machine Learning (SVM, Decision Trees, Random Forest).
- **Finalidade:** Classificar os produtos em diferentes categorias com base em suas características (modelo, cor, marca, etc.).

4. Análise de Churn de Clientes

- **Modelo sugerido:** Regressão Logística ou Redes Neurais.
- **Finalidade:** Identificar padrões que indicam que um cliente está prestes a abandonar o serviço da loja, permitindo a loja adotar ações de retenção.

5. Agrupamento de Clientes

- **Modelo sugerido:** K-Means.
- **Finalidade:** Segmentação de clientes em grupos para melhor personalização de ofertas e marketing.

Esses são os tipos mais comuns de modelos de IA aplicáveis ao cenário descrito no documento, onde a loja utiliza o banco de dados para melhorar sua tomada de decisões(Escopo do PI)

Criação da interface: index.html

1. Estrutura HTML e Metadados:

```
<!DOCTYPE html>

<html lang="pt-BR">
```

<!DOCTYPE html> Declara que o documento é do tipo HTML5.

<html lang="pt-BR"> Define o idioma da página como português do Brasil (lang="pt-BR").

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Gerenciamento de Loja</title>

O **<head>** contém metadados da página.

- **meta charset="UTF-8"** garante que a codificação de caracteres seja UTF-8, suportando caracteres especiais.
- **meta name="viewport"** ajusta a escala da página para ser responsiva em dispositivos móveis.
- **<title>** define o título que será exibido na aba do navegador, "Gerenciamento de Loja".

2. **Estilos e CSS:**

<!-- Bootstrap CSS -->

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css" rel="stylesheet">

Aqui o framework **Bootstrap** é carregado diretamente de um CDN, fornecendo estilos prontos para botões, formulários, etc.

<!-- Custom CSS -->

<style>

body {

background-color: #f8f9fa;

}

.container {

margin-top: 50px;

}

.card {

margin-bottom: 20px;

}

</style>

</head>

O bloco **<style>** contém o CSS customizado.

- Define um fundo claro (#f8f9fa) para a página.
- Define uma margem superior de 50 pixels para .container, para dar espaçamento entre o topo da página e o conteúdo.
- Adiciona um espaçamento inferior de 20 pixels para os elementos .card, separando os cartões de conteúdo.

3. **Corpo da Página:**

```
<body>
<div class="container">
    <h1 class="text-center">Sistema de Gerenciamento de Loja</h1>
    <div class="row justify-content-center">
        <div class="col-md-6">
```

1º A estrutura do corpo começa com a tag <body>.

2º O conteúdo da página está dentro de uma , que utiliza o layout fluido do Bootstrap.

3º <h1 class="text-center"> define o título principal da página, centralizado.

4. **Cards de Navegação:**

Os cards fornecem botões para navegar entre diferentes funcionalidades do sistema.

```
            <div class="card">
                <div class="card-body">
                    <h5 class="card-title">Cadastrar Produto</h5>
                    <p class="card-text">Cadastre novos produtos no sistema.</p>
                    <a href="/cadastrar-produto" class="btn btn-primary w-100">Ir para Cadastro</a>
```

Um cartão **( )** contendo:

- Um título **(<h5 class="card-title">Cadastrar Produto</h5>).**
- Texto explicativo do card **(Cadastre novos produtos no sistema.).**
- Um botão para redirecionar o usuário à rota **/cadastrar-produto**, estilizado com **Bootstrap** como um botão azul (**btn btn-primary w-100**), que ocupa toda a largura do card.

Esses mesmos elementos se repetem para outras funcionalidades:

1. **Atualizar Estoque:** Link para /atualizar-estoque.
2. **Registrar Venda:** Link para /registrar-venda.
3. **Cadastrar Auxiliares:** Link para /cadastrar-auxiliares.
4. **Consultar Vendas:** Link para /consultar-vendas.

5. **Scripts:**

   <!-- Bootstrap JS -->

   <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/js/bootstrap.bundle.min.js"></script>

   O arquivo de **JavaScript** do Bootstrap é carregado no final do corpo, permitindo a funcionalidade de componentes como modais, dropdowns, etc.

6. **Fechamento:**

   </body>

   </html>

   Finaliza o conteúdo do corpo e fecha o documento HTML.

Resumo:

Este HTML cria uma página inicial para o sistema de gerenciamento de loja, com navegação para diferentes seções (Cadastro de Produto, Atualização de Estoque, Registro de Vendas, etc.). A página é responsiva e usa Bootstrap para layout e design simplificado, com um esquema de cores claro e uma interface amigável através de cartões interativos.

Integração da interface com o banco

1. **Interface Web**

O **Flask** renderiza as páginas HTML (como index.html, cadastrar_produto.html, etc.) e utiliza templates para exibir os dados dinâmicos que são carregados ou enviados para o banco de dados.

2. **Pontos de Integração Banco de Dados e Interface Web:**

   - **Função connect_db():** Essa função é usada em todas as rotas que precisam se comunicar com o banco de dados. A conexão com o SQL Server é estabelecida e usada para realizar operações de inserção, consulta e atualização. Esta função não cria a interface, mas possibilita que as rotas Flask comuniquem-se com o banco.

```
def connect_db():
    conn = pyodbc.connect('DRIVER={ODBC Driver 17 for SQL Server};'
                          'SERVER=localhost;'
                          'DATABASE=BANCOPI;'
                          'Trusted_Connection=yes;'
                          'Encrypt=yes;'
                          'TrustServerCertificate=yes;')
    return conn
```

3. **Renderização de Templates com Dados do Banco de Dados:** Quando a função render_template() é usada, o Flask gera a página HTML e pode passar dados dinâmicos (consultados do

banco de dados) para essa página. Isso é o que conecta a interface (página web) com o banco de dados.

Exemplos:

4. **Renderização da página de cadastro de produtos:** A rota /cadastrar-produto consulta as tabelas de Modelo, Marca, Cor, Tamanho e Estampa para carregar as opções dinâmicas no formulário. O resultado da consulta é passado para o template cadastrar_produto.html, onde os dados são exibidos em um formulário HTML.

```
@app.route('/cadastrar-produto', methods=['GET', 'POST'])
def cadastrar_produto():
if request.method == 'GET':
conn = connect_db()
cursor = conn.cursor()

cursor.execute("SELECT nome_Modelo FROM Modelo")
modelos = [row[0] for row in cursor.fetchall()]

cursor.execute("SELECT nome_Marca FROM Marca")
marcas = [row[0] for row in cursor.fetchall()]

cursor.execute("SELECT nome_Cor FROM Cor")
cores = [row[0] for row in cursor.fetchall()]

cursor.execute("SELECT tamanho FROM Tamanho")
tamanhos = [row[0] for row in cursor.fetchall()]

cursor.execute("SELECT tipo_Estampa FROM Estampa")
estampas = [row[0] for row in cursor.fetchall()]
```

```
    conn.close()

    # Passa os dados para o template
    return render_template('cadastrar_produto.html',
        modelos=modelos,
        marcas=marcas,
        cores=cores,
        tamanhos=tamanhos,
        estampas=estampas)
```

Neste exemplo, os dados consultados (como modelos, marcas, etc.) são extraídos do banco e passados ao template, integrando a interface web com o banco de dados.

Explicação passo a passo do detalhamento do código:

1. **Importação das bibliotecas**

```
from flask import Flask, render_template, request, redirect, url_for, flash
import pyodbc
import webbrowser
import threading
```

1.1 from flask import Flask, render_template, request, redirect, url_for, flash: Importa as funções e classes necessárias do Flask:

- **Flask**: Para criar a aplicação web.
- **render_template**: Renderiza arquivos HTML.
- **request**: Acessa dados de uma requisição HTTP (por exemplo, dados de formulários).
- **redirect** e **url_for**: Redirecionam o usuário para outras rotas dentro da aplicação.

- **flash:** Exibe mensagens temporárias (notificações de sucesso ou erro).

1.2 import **pyodbc:** Biblioteca usada para conectar-se a bancos de dados via ODBC (neste caso, SQL Server).

1.3 import **webbrowser:** Permite abrir URLs no navegador web.

1.4 import **threading:** Importa a biblioteca de threads para executar ações em paralelo (aqui usada para abrir o navegador ao iniciar o servidor Flask).

2. Inicializando a Aplicação Flask

```
app = Flask(__name__)
app.secret_key = "chave_secreta_para_flash_messages"
```

1º app = Flask(__name__): Cria uma instância da aplicação Flask, onde __name__ é o nome do módulo atual.

2º app.secret_key = "chave_secreta_para_flash_messages": Define uma chave secreta usada pelo Flask para proteger as mensagens flash e outras funcionalidades que exigem segurança (como sessões).

3. Variável de Controle para o Navegador

```
browser_opened = False
```

browser_opened = False: Variável usada para garantir que o navegador só seja aberto uma vez.

4. Função de Conexão ao Banco de Dados

```
def connect_db():
    conn = pyodbc.connect('DRIVER={ODBC Driver 17 for SQL Server};'
                          'SERVER=localhost;'
                          'DATABASE=BANCOPI;'
                          'Trusted_Connection=yes;'
```

```
                'Encrypt=yes;'
                'TrustServerCertificate=yes;')
    return conn
```

connect_db(): Função que conecta ao banco de dados BANCOPI no servidor local usando o driver ODBC. Ela retorna uma conexão ativa para ser usada nas operações de banco de dados.

5. **Rota Principal**

```
@app.route('/')
def index():
    return render_template('index.html')
```

- **@app.route('/')**: Define a rota principal do site (quando acessamos /).
- **def index()**: Função associada à rota. Retorna o template index.html, que será renderizado no navegador.

6. **Rota de Cadastro de Produtos**

```
@app.route('/cadastrar-produto', methods=['GET', 'POST'])
def cadastrar_produto():
    if request.method == 'POST':
```

- **@app.route('/cadastrar-produto', methods=['GET', 'POST'])**: Define a rota para cadastrar produtos. Essa rota aceita os métodos HTTP GET (para exibir o formulário) e POST (para enviar dados).
- **if request.method == 'POST':**: Verifica se o método da requisição é POST, ou seja, se o formulário foi enviado.

7. **Processamento do Formulário**

```
etiqueta = request.form['etiqueta']

modelo = request.form['modelo']

marca = request.form['marca']

cor = request.form['cor']

tamanho = request.form['tamanho']

estampa = request.form['estampa']

preco_unitario = request.form['preco_unitario']
```

request.form: Extrai os dados enviados no formulário via POST, como etiqueta, modelo, marca, etc

8. **Conexão e Inserção no Banco**.

```
conn = connect_db()

cursor = conn.cursor()
```

1º conn = connect_db(): Estabelece uma conexão com o banco de dados.

2º cursor = conn.cursor(): Cria um cursor, que é usado para executar comandos SQL no banco.

9. Inserção de Produto

```
cursor.execute("""

   INSERT INTO Produto (etiqueta, idModelo, idMarca, idCor, idTamanho, idEstampa, preco_unitario)

   VALUES (?, ?, ?, ?, ?, ?, ?)

""", (etiqueta, idModelo, idMarca, idCor, idTamanho, idEstampa, preco_unitario))

conn.commit()
```

- **cursor.execute(...)**: Insere o novo produto na tabela Produto com os dados fornecidos.
- **conn.commit()**: Confirma as alterações no banco de dados.

10. Mensagem de Sucesso e Redirecionamento

```
flash("Produto cadastrado com sucesso!", "success")

return redirect(url_for('cadastrar_produto'))
```

- **flash()**: Exibe uma mensagem de sucesso usando o mecanismo de mensagens temporárias.
- **redirect(url_for(...))**: Redireciona o usuário de volta para a página de cadastro de produtos.

11. O Resto do Código

As outras rotas (/atualizar-estoque, /registrar-venda, /cadastrar-auxiliares, etc.) seguem uma lógica semelhante, com:

1. **Recepção de dados de formulários**.
2. **Consultas ao banco de dados** (para buscar ou atualizar dados).
3. **Renderização de templates HTML**.
4. **Mensagens de feedback usando flash()**.

12. Abrir o Navegador

```
def open_browser():
    global browser_opened
    if not browser_opened:
        webbrowser.open("http://127.0.0.1:5000/")
        browser_opened = True
```

open_browser(): Função que abre o navegador na URL http://127.0.0.1:5000/ ao iniciar o servidor.

13. Iniciar o Servidor

```
if __name__ == '__main__':
```

```
    threading.Thread(target=open_browser).start()

    app.run(debug=True, use_reloader=False)
```

- **if __name__ == '__main__':**: Verifica se o script está sendo executado diretamente e não importado como módulo.
- **threading.Thread(...).start()**: Inicia uma thread paralela para abrir o navegador automaticamente.
- **app.run(debug=True, use_reloader=False)**: Inicia o servidor Flask com o modo de depuração ativado, mas sem recarregar o servidor automaticamente (para evitar múltiplas aberturas de navegador).