



MODELO DE CLASSIFICAÇÃO

MÉTODO DE CLASSIFICAÇÃO PARA PREVISÃO DE SATISFAÇÃO COM PRODUTOS

Objetivo: prever se um cliente gostará ou não de um produto com base em suas características específicas.

Aplicação: e-commerce, marketing, melhoria de produtos, e otimização de estoque, onde a antecipação de comportamentos e preferências de clientes pode resultar em melhores decisões de negócio e maior lucratividade.



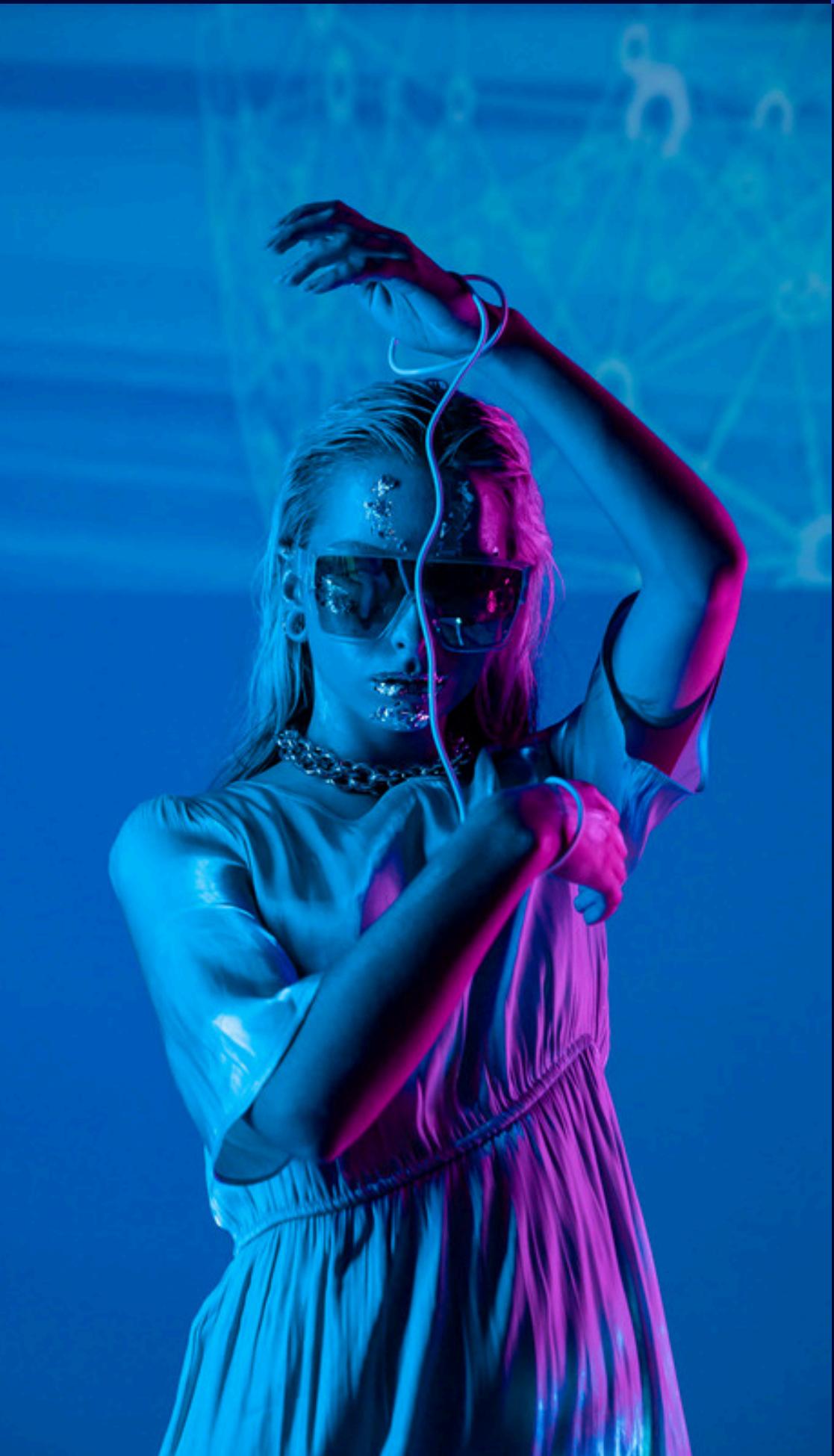
BASE DE DADOS

KAGGLE

LINK: <https://www.kaggle.com/datasets/cameronseamons/electronic-sales-sep2023-sep2024/data>

O conjunto de dados contém informações sobre vendas eletrônicas de setembro de 2023 a setembro de 2024.

- **Customer ID, Age, Gender:** Identificadores e características do cliente.
- Loyalty Member: Status de fidelidade (Sim ou Não).
- **Product Type, SKU:** Tipo de produto e código do produto.
- **Rating:** Avaliação do produto.
- Order Status: Estado do pedido (ex.: Completado, Cancelado).
- Payment Method: Método de pagamento.
- **Total Price, Unit Price, Quantity:** Informações sobre preço total, unitário e quantidade comprada.



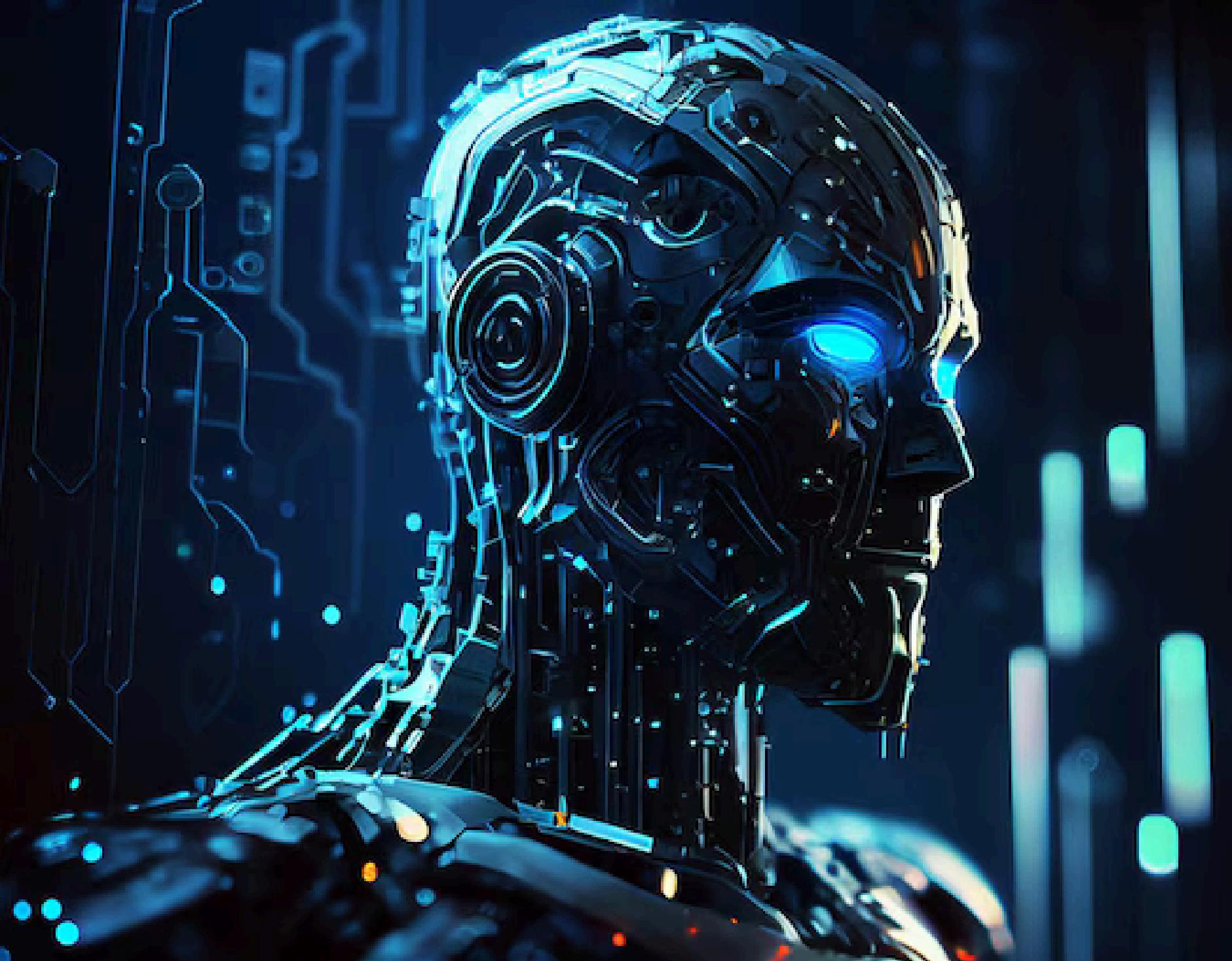
TARGET

- O código irá criar uma nova coluna chamada Liked no conjunto de dados.
- Ele converte a coluna de avaliações (Rating) em uma variável binária.
- Se a avaliação (Rating) for maior ou igual a 4, o produto é considerado "gostado" e a variável recebe o valor 1.
- Se a avaliação for menor que 4, o produto é considerado "não gostado" e a variável recebe o valor 0.

Se a avaliação (Rating) for maior ou igual a 4, o produto é considerado "gostado" e a variável recebe o valor 1.

```
# Preparar a variável alvo  
data['Liked'] = (data['Rating'] >= 4).astype(int) # Cria uma coluna 'Liked' como alvo
```

PROCESSO



VARIAVÉIS CATEGÓRICAS

- O LabelEncoder será usado para converter dados categóricos em números inteiros.
- As classes presente na coluna **Gender** são: Male e Female.
- As classes presente na coluna Product Type são: Laptop, Headphones, Tablet, Smartphone, Smartwatch.

```
[110] from sklearn.preprocessing import LabelEncoder  
  
# Criar encoders separados  
gender_encoder = LabelEncoder()  
product_type_encoder = LabelEncoder()  
  
# Ajustar os encoders  
data['Gender'] = gender_encoder.fit_transform(data['Gender'])  
data['Product Type'] = product_type_encoder.fit_transform(data['Product Type'])
```

PROCESSO



CARACTERÍSTICAS

- Criação da variável X, que contém as características que o modelo usará para fazer previsões.
- As características escolhidas são:
 - **Age (Idade):** A idade do cliente (numérica).
 - **Gender (Gênero):** O gênero do cliente, que já foi convertido para números usando LabelEncoder (numérica codificada).
 - **Total Price (Preço Total):** O preço total do(s) produto(s) comprados pelo cliente (numérica).
 - **Product Type (Tipo de Produto):** O tipo de produto que o cliente comprou, também já codificado para números com o LabelEncoder (numérica codificada).
- A **variável y (target)** é o que o modelo tenta prever. Durante o treinamento, o modelo aprende a partir de X para prever y, ajustando-se conforme observa os padrões dos dados.

```
▶ # Selecionar características e a variável alvo  
X = data[['Age', 'Gender', 'Total Price', 'Unit Price', 'Quantity', 'Product Type']]  
y = data['Liked']
```

PROCESSO



CONJUNTO DE TREINAMENTO

- O código irá dividir os dados em conjuntos de treinamento e teste.
- O modelo de machine learning é treinado usando o conjunto X_train e y_train. Durante o treinamento, o modelo aprende a associar as características (X) às avaliações (y).
- Após a execução, serão quatro conjuntos de dados:
 - X_train: Contém 75% das características de entrada que serão usadas para treinar o modelo.
 - X_test: Contém 25% das características de entrada que serão usadas para testar o modelo.
 - y_train: Contém 75% da variável alvo correspondente ao conjunto de treinamento.
 - y_test: Contém 25% da variável alvo correspondente ao conjunto de teste.

```
[37] # Dividir os dados em treino e teste  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=1)
```

PROCESSO

RandomForestClassifier

- Definição de classificação Random Forest, utilizando 100 árvores e garantindo que o processo seja reproduzível.

```
# Treinar um modelo de classificação  
model = RandomForestClassifier(n_estimators=100, random_state=1)  
model.fit(X_train, y_train)
```



PROCESSO

Matriz de Confusão



- `confusion_matrix(y_test, predictions)`: Essa função calcula a matriz de confusão comparando os valores reais (`y_test`) com as previsões feitas pelo modelo (`predictions`).
- `y_test`: É a variável que contém os rótulos verdadeiros (o que realmente aconteceu) para os dados de teste.
- `predictions`: É a variável que contém as previsões feitas pelo modelo sobre os dados de teste.

```
[44] import matplotlib.pyplot as plt
     import seaborn as sns
     from sklearn.metrics import confusion_matrix

     # Calcular a matriz de confusão
     cm = confusion_matrix(y_test, predictions)

     # Criar um gráfico de calor da matriz de confusão
     plt.figure(figsize=(8, 6))
     sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Não Gostou', 'Gostou'], yticklabels=['Não Gostou', 'Gostou'])
     plt.ylabel('Classe Verdadeira')
     plt.xlabel('Classe Prevista')
     plt.title('Matriz de Confusão')
     plt.show()
```

PROCESSO

RESULTADO

Total de Registros 5.000

REALMENTE GOSTARAM 1.544

REALMENTE NÃO GOSTARAM 3.456

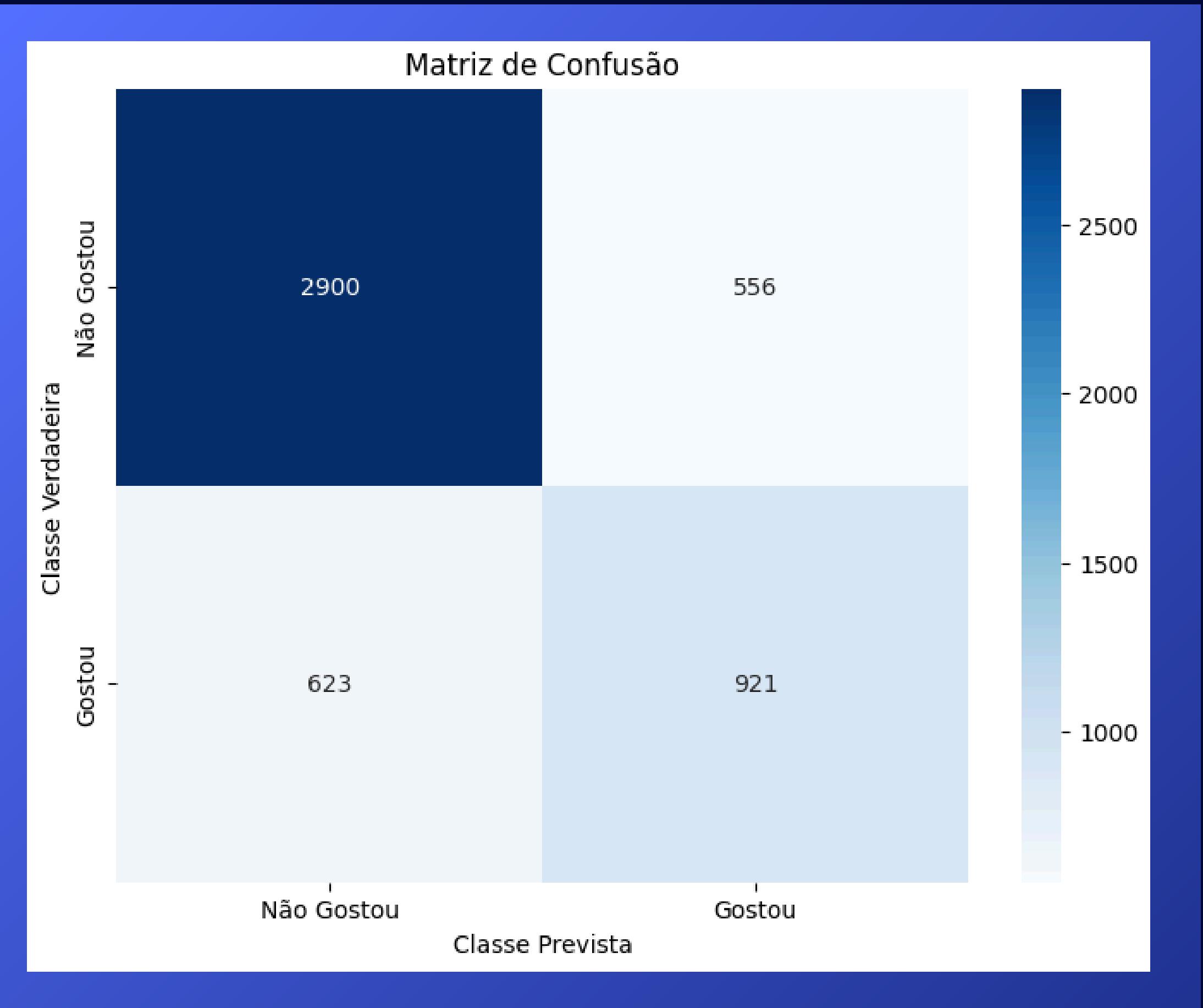
IA PREVISTO (TP) 921

IA PREVISTO (FP) 2.900

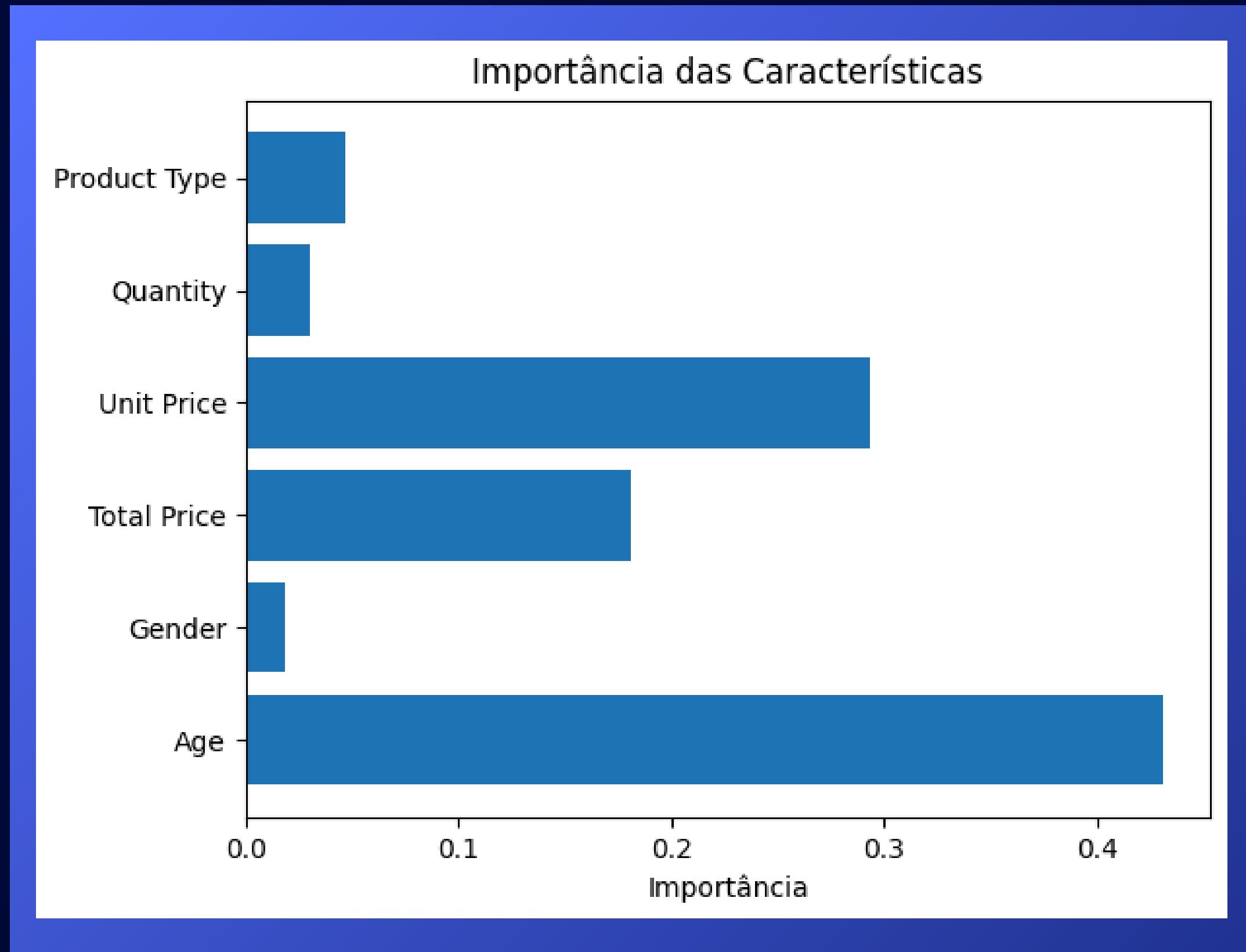
IA PREVISTO (FP) 556

IA PREVISTO (FN) 623

Matriz de Confusão



IMPORTÂNCIA DE DECISÃO POR CARACTERÍSTICA



RELATÓRIO

Accuracy: 0.7642

Classification Report:

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.82 | 0.84 | 0.83 | 3456 |
| 1 | 0.62 | 0.60 | 0.61 | 1544 |
| accuracy | | | 0.76 | 5000 |
| macro avg | 0.72 | 0.72 | 0.72 | 5000 |
| weighted avg | 0.76 | 0.76 | 0.76 | 5000 |

01

O modelo tem um desempenho melhor em prever a classe "Não Gostou" (classe 0) em comparação com a classe "Gostou" (classe 1).

02

Possível solução é melhorar as características do modelo, ou ajustar hiperparâmetros para tentar melhorar a previsão da classe "Gostou".



CASO DE TESTE

```
def predict_liked(age, gender, total_price, unit_price, quantity, product_type):
    try:
        # Codificar o tipo de produto e o gênero usando seus respectivos encoders
        product_type_encoded = product_type_encoder.transform([product_type])[0]
        gender_encoded = gender_encoder.transform([gender])[0]

        # Criar um DataFrame com as características, incluindo todos os recursos
        input_features = pd.DataFrame({
            'Age': [age],
            'Gender': [gender_encoded],
            'Total Price': [total_price],
            'Unit Price': [unit_price],
            'Quantity': [quantity],
            'Product Type': [product_type_encoded]
        })

        # Fazer a previsão
        prediction = model.predict(input_features)

        # Retornar o resultado
        return "Gostou" if prediction[0] == 1 else "Não Gostou"

    except ValueError as e:
        return str(e) # Retornar a mensagem de erro se uma categoria inválida for inserida
```



RESULTADO DO TESTE

```
[127] result = predict_liked(35, 'Female', 150, 50, 2, 'Smartwatch')
      print(f"A previsão é: {result}")
```

```
▶ result = predict_liked(35, 'Female', 150, 50, 2, 'Smartwatch')
  print(f"A previsão é: {result}")

→ A previsão é: Gostou
```

```
...od ~ modifier_ob.modifiers.new("...  
ur object to mirror_ob  
mod.mirror_object = mirror_ob  
ion == "MIRROR_X":  
mod.use_x = True  
mod.use_y = False  
mod.use_z = False  
ction == "MIRROR_Y":  
mod.use_x = False  
mod.use_y = True  
mod.use_z = False  
ction == "MIRROR_Z":  
mod.use_x = False  
mod.use_y = False  
mod.use_z = True  
tion at the end -add back the deselected  
select= 1
b.select=1
t.scene.objects.active = modifier_ob
selected" + str(modifier_ob)) # modifier ob
r_ob.select = 0
context.selected_objects[0]
objects[one.name].select = 1
"please select exactly two objects.
OPERATOR CLASSES - - - - -  
...Operator):
  "mirror to the selected object"
  t.mirror_mirror_x"
  "X"
next):
  "active_object is not None"
  active_object
```

RESULTADO DO TESTE

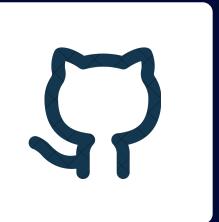
```
[127] result = predict_liked(35, 'Female', 150, 50, 2, 'Smartwatch')
      print(f"A previsão é: {result}")
```

```
▶ result = predict_liked(35, 'Female', 150, 50, 2, 'Smartwatch')
  print(f"A previsão é: {result}")
```

```
→ A previsão é: Gostou
```

```
..._mod ~ modifier_ob.modifiers.new("...")  
... object to mirror_ob  
... mod.mirror_object = mirror_ob  
... ion == "MIRROR_X":  
... mod.use_x = True  
... mod.use_y = False  
... mod.use_z = False  
... ution == "MIRROR_Y":  
... mod.use_x = False  
... mod.use_y = True  
... mod.use_z = False  
... ution == "MIRROR_Z":  
... mod.use_x = False  
... mod.use_y = False  
... mod.use_z = True  
... tion at the end -add back the deselected objects  
... select= 1  
... &.select=1  
... &.scene.objects.active = modifier_ob  
... selected" + str(modifier_ob)) # modifier ob  
... &_ob.select = 0  
... &.context.selected_objects[0]  
... objects[one.name].select = 1  
... "please select exactly two objects.  
... OPERATOR CLASSES ...  
... .operator):  
... "mirror to the selected object"**  
... &.mirror_mirror_x"  
... "X"  
... ext):  
... "active_object is not None"  
... &.active_object
```

ACESSE O PROJETO



[GITHUB](#)



[LINKEDIN](#)





MUITO OBRIGADO!