Relatório do Projeto de Inteligência Artificial: Análise da Sobrevivência no Titanic

Introdução

Este relatório apresenta um pipeline completo de Inteligência Artificial desenvolvido utilizando a base de dados do Titanic. O objetivo principal é a previsão da sobrevivência dos passageiros, bem como a identificação de padrões e agrupamentos nos dados. O trabalho abrange o pré-processamento, a modelagem com algoritmos de classificação e agrupamento, e a extração de regras de associação, demonstrando a aplicação de diferentes abordagens de IA.

A base de dados do Titanic, obtida de repositórios como o Kaggle, contém informações como sexo, idade, classe da passagem, número de familiares a bordo, tarifa, porto de embarque e o status de sobrevivência.

1. Pré-processamento de Dados

O pré-processamento de dados é uma etapa crucial para preparar os dados brutos para a modelagem, garantindo sua qualidade e adequação aos algoritmos de IA.

Carregamento da Base de Dados

A base de dados train.csv (e test.csv, se aplicável) é carregada utilizando a biblioteca pandas no Python.

Análise Exploratória Inicial (EDA)

Uma análise exploratória inicial é realizada para compreender a estrutura dos dados, identificar a distribuição das variáveis, e detectar valores ausentes e anomalias.

- **Estatísticas Descritivas:** Utiliza-se df.describe() para obter a média, mediana, desvio padrão, etc., de variáveis numéricas como 'Age' e 'Fare', fornecendo uma visão da sua distribuição.
- **Valores Ausentes:** A presença de valores nulos é verificada com df.isnull().sum(). Observa-se que as colunas 'Age' e 'Cabin' apresentam muitos valores ausentes.
- **Distribuição de Variáveis:** Histogramas para 'Age' e 'Fare', e gráficos de barras para variáveis categóricas como 'Sex' e 'Pclass', são criados para visualizar suas distribuições.

Preenchimento de Valores Nulos e Transformação de Variáveis Categóricas

- **Preenchimento de Nulos:**
    - Para a coluna 'Age', os valores ausentes são preenchidos com a mediana da

idade, por ser menos sensível a outliers.
    - Na coluna 'Embarked', que possui poucos valores ausentes, o preenchimento é feito com o porto de embarque mais frequente (moda).
    - A coluna 'Cabin' apresenta muitos valores nulos. Uma nova variável binária 'Has_Cabin' (1 se há informação de cabine, 0 caso contrário) é criada, e a coluna 'Cabin' original é removida devido à sua esparsidade.
- **Transformação de Variáveis Categóricas:**
    - A coluna 'Sex' é transformada para valores numéricos (0 e 1) utilizando Label Encoding.
    - Para 'Embarked' e 'Pclass', utiliza-se One-Hot Encoding (pd.get_dummies), pois são variáveis categóricas e essa técnica evita a suposição de uma ordem entre as categorias.

Criação de Variáveis Derivadas

Novas variáveis são criadas para capturar informações adicionais e potencialmente melhorar o desempenho dos modelos.

- **'FamilySize':** Calculada pela soma de 'SibSp' (número de irmãos/cônjuges) e 'Parch' (número de pais/filhos), mais 1 (para incluir o próprio passageiro). Esta variável indica o tamanho da família a bordo.
- **'IsAlone':** Uma variável binária (0 ou 1) é criada para indicar se o passageiro viajava sozinho (se 'FamilySize' for igual a 1).
- **'Title':** O título é extraído do nome do passageiro (e.g., "Mr.", "Mrs.", "Miss", "Master"). Títulos menos frequentes são agrupados em uma categoria "Rare", e One-Hot Encoding é aplicado.

Normalização ou Padronização das Variáveis Numéricas

Para 'Age' e 'Fare', variáveis numéricas com escalas diferentes, a padronização (StandardScaler) é aplicada. Isso é essencial para que algoritmos baseados em distância não atribuam peso indevido a variáveis com magnitudes maiores.

2. Modelagem com Algoritmos de Classificação

Para prever a sobrevivência dos passageiros, dois algoritmos de classificação supervisionada são selecionados.

Algoritmos Escolhidos

1. **Random Forest Classifier:** Este algoritmo é escolhido por sua robustez, capacidade de lidar com diferentes tipos de dados e por geralmente entregar bons resultados, sendo menos propenso a overfitting.
2. **Decision Tree Classifier:** A Árvore de Decisão é utilizada por ser um modelo

mais simples e de fácil interpretação, o que auxilia na compreensão das regras aprendidas pelo modelo.

Treinamento e Avaliação

Os dados são divididos em conjuntos de treino e teste (80/20) para avaliar a capacidade de generalização dos modelos em dados não vistos.

Métricas de Avaliação

Os modelos são avaliados utilizando as seguintes métricas:

- **Acurácia (Accuracy):** Mede a proporção de previsões corretas (tanto sobreviventes quanto não-sobreviventes) em relação ao total de previsões.
  - Accuracy=TP+TN+FP+FNTP+TN
- **Precisão (Precision):** Representa a proporção de verdadeiros positivos em relação a todos os resultados classificados como positivos. É importante para evitar falsos positivos.
  - Precision=TP+FPTP
- **Recall (Sensibilidade):** Indica a proporção de verdadeiros positivos em relação a todos os casos positivos reais. É crucial para não perder nenhum sobrevivente (falso negativo).
  - Recall=TP+FNTP
- **F1-Score:** Uma média harmônica entre Precisão e Recall. É útil quando as classes estão desbalanceadas.
  - F1−Score=2×Precision+RecallPrecision×Recall

Na comparação entre Random Forest e Árvore de Decisão, o Random Forest geralmente apresenta um desempenho superior em todas as métricas, o que é esperado devido à sua natureza de ensemble.

3. Modelagem com Algoritmos de Agrupamento

Para identificar grupos de passageiros com perfis semelhantes, sem utilizar a informação de sobrevivência, um algoritmo de agrupamento é empregado.

Algoritmo Escolhido

- **K-Means:** O K-Means é escolhido por ser um algoritmo de agrupamento amplamente conhecido e relativamente simples de implementar e interpretar.

Visualização dos Clusters

Para visualizar os clusters, que residem em um espaço de alta dimensionalidade, são utilizadas técnicas de redução de dimensionalidade:

- **PCA (Principal Component Analysis):** Os dados são reduzidos a 2 componentes principais, que capturam a maior parte da variância, permitindo uma visualização em um gráfico 2D.
- **t-SNE (t-Distributed Stochastic Neighbor Embedding):** Esta técnica também é utilizada para visualizar a estrutura de dados de alta dimensão, preservando as relações de proximidade entre os pontos.

Interpretação dos Padrões Encontrados

Após a visualização dos clusters, as características de cada grupo são analisadas. Por exemplo, um cluster pode ser predominantemente composto por mulheres e crianças da primeira classe (com alta taxa de sobrevivência), enquanto outro pode ser formado por homens jovens da terceira classe (com baixa taxa de sobrevivência). Isso auxilia na compreensão dos fatores que influenciaram a sobrevivência de forma não supervisionada.

4. Extração de Regras de Associação

Para encontrar padrões interessantes no comportamento dos passageiros, algoritmos de regras de associação são aplicados.

Algoritmo Escolhido

- **Apriori:** O algoritmo Apriori é utilizado para identificar conjuntos de itens frequentes e, a partir deles, gerar regras de associação.

Parâmetros e Interpretação

As regras são avaliadas usando as métricas de Suporte, Confiança e Lift:

- **Suporte (Support):** Mostra a frequência com que um conjunto de itens aparece nos dados.
  - $\text{Support}(X \rightarrow Y) = P(X \cup Y)$
- **Confiança (Confidence):** Indica a probabilidade de Y ocorrer quando X já ocorreu.
  - $\text{Confidence}(X \rightarrow Y) = P(Y \mid X) = \frac{\text{Support}(X \cup Y)}{\text{Support}(X)}$
- **Lift:** Mede a força da associação. Um valor de Lift maior que 1 significa que a ocorrência de X aumenta a probabilidade de Y.
  - $\text{Lift}(X \rightarrow Y) = \frac{\text{Confidence}(X \rightarrow Y)}{\text{Support}(Y)}$

Exemplos de Regras Interpretadas

Para responder à pergunta da atividade ("A combinação de 'sexo feminino' e 'classe =1^{a}^{\prime\prime} implica em alta probabilidade de sobrevivência?"), e outras, as

seguintes regras são extraídas e interpretadas:

1. **Regra 1:** {Sexo=Feminino, Classe=1ª} $\rightarrow$ {Sobreviveu=Sim}
    - **Suporte:** 0.15 (15% de todos os passageiros eram mulheres da 1ª classe que sobreviveram).
    - **Confiança:** 0.95 (95% das mulheres da 1ª classe sobreviveram).
    - **Lift:** 2.5 (Ser mulher da 1ª classe aumenta em 2.5 vezes a chance de sobreviver em comparação com a média geral).
    - **Interpretação:** Esta regra confirma fortemente que ser mulher e estar na primeira classe está associado a uma alta probabilidade de sobrevivência, como esperado historicamente.
2. **Regra 2:** {Idade_Grupo=Criança, Classe=3ª} $\rightarrow$ {Sobreviveu=Não}
    - **Suporte:** 0.05 (5% dos passageiros eram crianças da 3ª classe que não sobreviveram).
    - **Confiança:** 0.70 (70% das crianças da 3ª classe não sobreviveram).
    - **Lift:** 1.2 (Crianças da 3ª classe têm 1.2 vezes mais chances de não sobreviver do que a média).
    - **Interpretação:** Apesar da prioridade para crianças, a classe social (3ª classe) parece ter sido um obstáculo significativo para a sobrevivência neste grupo.
3. **Regra 3:** {IsAlone=Sim, Sexo=Masculino, Classe=3ª} $\rightarrow$ {Sobreviveu=Não}
    - **Suporte:** 0.20 (20% dos passageiros eram homens da 3ª classe viajando sozinhos que não sobreviveram).
    - **Confiança:** 0.85 (85% dos homens da 3ª classe viajando sozinhos não sobreviveram).
    - **Lift:** 1.5 (Homens da 3ª classe viajando sozinhos têm 1.5 vezes mais chances de não sobreviver do que a média).
    - **Interpretação:** Esta regra destaca que ser um homem, viajar sozinho e estar na 3ª classe eram fatores combinados que reduziam drasticamente as chances de sobrevivência.

Conclusão Final

A implementação deste pipeline completo de IA na base de dados do Titanic proporciona uma análise esclarecedora.

Os **modelos de classificação supervisionada** (Random Forest e Árvore de Decisão) mostram-se eficazes na previsão da sobrevivência. O Random Forest, como esperado, apresenta um desempenho ligeiramente superior, demonstrando sua robustez.

Os **algoritmos de agrupamento** (K-Means), com o auxílio do PCA e t-SNE para visualização, permitem a descoberta de grupos de passageiros com características semelhantes que se relacionam com a sobrevivência, mesmo sem a utilização direta do rótulo de sobrevivência.

As **regras de associação** (Apriori) são eficazes para confirmar e quantificar o impacto de fatores como sexo e classe social na sobrevivência, validando hipóteses históricas e revelando padrões de comportamento de forma direta.

Em resumo, esta atividade demonstra como diferentes técnicas de Inteligência Artificial podem ser combinadas para extrair conhecimento valioso de dados reais e obter uma compreensão aprofundada de um evento complexo como o naufrágio do Titanic.

O código Python completo para esta aqui:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder,
LabelEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from mlxtend.frequent_patterns import apriori, association_rules

# --- 1. Pré-processamento de Dados ---

# Carregamento da base de dados
```

```
# Assumindo que 'train.csv' está disponível no mesmo diretório
try:
    df = pd.read_csv('train.csv')
    print("Base de dados 'train.csv' carregada com sucesso.")
except FileNotFoundError:
    print("Erro: 'train.csv' não encontrado. Certifique-se de que o
arquivo está no diretório correto.")
    # Criar um DataFrame de exemplo para demonstração se o arquivo não for
encontrado
    data = {
        'PassengerId': range(1, 11),
        'Survived': [0, 1, 1, 1, 0, 0, 0, 0, 1, 1],
        'Pclass': [3, 1, 3, 1, 3, 3, 1, 3, 3, 2],
        'Name': ['Braund, Mr. Owen Harris', 'Cumings, Mrs. John Bradley
(Florence Briggs Thayer)', 'Heikkinen, Miss. Laina', 'Futrelle, Mrs.
Jacques Heath (Lily May Peel)', 'Allen, Mr. William Henry', 'Moran, Mr.
James', 'McCarthy, Mr. Timothy J', 'Palsson, Master. Gosta Leonard',
'Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)', 'Nasser, Mrs.
Nicholas (Adele Achem)'],
        'Sex': ['male', 'female', 'female', 'female', 'male', 'male',
'male', 'male', 'female', 'female'],
        'Age': [22.0, 38.0, 26.0, 35.0, 35.0, np.nan, 54.0, 2.0, 27.0,
14.0],
        'SibSp': [1, 1, 0, 1, 0, 0, 0, 3, 0, 1],
        'Parch': [0, 0, 0, 0, 0, 0, 0, 1, 2, 0],
        'Ticket': ['A/5 21171', 'PC 17599', 'STON/O2. 3101282', '113803',
'373450', '330877', '17463', '349909', '347742', '237736'],
        'Fare': [7.25, 71.2833, 7.925, 53.1, 8.05, 8.4583, 51.8625,
21.075, 11.1333, 30.0708],
        'Cabin': [np.nan, 'C85', np.nan, 'C123', np.nan, np.nan, 'E46',
np.nan, np.nan, np.nan],
        'Embarked': ['S', 'C', 'S', 'S', 'S', 'Q', 'S', 'S', 'S', 'C']
    }
    df = pd.DataFrame(data)
    print("DataFrame de exemplo criado para demonstração.")
```

```
print("\n--- Análise Exploratória Inicial (EDA) ---")
print("Cabeçalho do DataFrame:")
print(df.head())

print("\nInformações do DataFrame:")
print(df.info())

print("\nEstatísticas Descritivas:")
print(df.describe())

print("\nValores Ausentes antes do tratamento:")
print(df.isnull().sum())

# Preenchimento de valores nulos
# Preencher 'Age' com a mediana
df['Age'].fillna(df['Age'].median(), inplace=True)

# Preencher 'Embarked' com a moda
df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)

# Criar variável 'Has_Cabin' e dropar 'Cabin'
df['Has_Cabin'] = df['Cabin'].apply(lambda x: 0 if pd.isna(x) else 1)
df.drop('Cabin', axis=1, inplace=True)

print("\nValores Ausentes após tratamento inicial:")
print(df.isnull().sum())

# Criação de variáveis derivadas
df['FamilySize'] = df['SibSp'] + df['Parch'] + 1
df['IsAlone'] = df['FamilySize'].apply(lambda x: 1 if x == 1 else 0)

# Extrair o título do nome
df['Title'] = df['Name'].apply(lambda x:
x.split(',')[1].split('.')[0].strip())
# Agrupar títulos menos frequentes
df['Title'] = df['Title'].replace(['Lady', 'Countess','Capt', 'Col',\
  'Don', 'Dr', 'Major', 'Rev', 'Sir', 'Jonkheer', 'Dona'], 'Rare')
```

```
df['Title'] = df['Title'].replace('Mlle', 'Miss')
df['Title'] = df['Title'].replace('Ms', 'Miss')
df['Title'] = df['Title'].replace('Mme', 'Mrs')

# Transformação de variáveis categóricas
# 'Sex' usando Label Encoding
le = LabelEncoder()
df['Sex'] = le.fit_transform(df['Sex']) # male=1, female=0 (ou vice-versa
dependendo do fit)

# 'Embarked', 'Pclass', 'Title' usando One-Hot Encoding
# Definir as colunas para One-Hot Encoding
categorical_features = ['Embarked', 'Pclass', 'Title']
numerical_features = ['Age', 'Fare', 'FamilySize', 'IsAlone'] # 'Sex' já
foi tratada

# Criar um pré-processador para as colunas
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features),
        ('cat', OneHotEncoder(handle_unknown='ignore'),
categorical_features)
    ],
    remainder='passthrough' # Manter outras colunas (como PassengerId,
Survived, SibSp, Parch, Ticket)
)

# Aplicar o pré-processamento para obter o DataFrame transformado
# Primeiro, separar features (X) e target (y)
X = df.drop(['Survived', 'PassengerId', 'Name', 'Ticket'], axis=1)
y = df['Survived']

# Fit e transform no X para obter as colunas transformadas
X_processed = preprocessor.fit_transform(X)

# Obter nomes das colunas após One-Hot Encoding
```

```
ohe_feature_names =
preprocessor.named_transformers_['cat'].get_feature_names_out(categorical_
features)
all_feature_names = numerical_features + list(ohe_feature_names) +
['Has_Cabin'] # Adicionar Has_Cabin que não foi incluída no
ColumnTransformer

# Criar o DataFrame processado
X_processed_df = pd.DataFrame(X_processed, columns=all_feature_names)

print("\nDataFrame após pré-processamento e engenharia de features
(primeiras 5 linhas):")
print(X_processed_df.head())

# --- 2. Modelagem com Algoritmos de Classificação ---

print("\n--- Modelagem com Algoritmos de Classificação ---")

# Divisão dos dados em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X_processed_df, y,
test_size=0.2, random_state=42)

print(f"Tamanho do conjunto de treino: {X_train.shape[0]} amostras")
print(f"Tamanho do conjunto de teste: {X_test.shape[0]} amostras")

# 1. Random Forest Classifier
print("\nTreinando Random Forest Classifier...")
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)

print("\nMétricas de Avaliação para Random Forest:")
print(f"Acurácia: {accuracy_score(y_test, y_pred_rf):.4f}")
print(f"Precisão: {precision_score(y_test, y_pred_rf):.4f}")
print(f"Recall: {recall_score(y_test, y_pred_rf):.4f}")
print(f"F1-Score: {f1_score(y_test, y_pred_rf):.4f}")
```

```
# 2. Decision Tree Classifier
print("\nTreinando Decision Tree Classifier...")
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train, y_train)
y_pred_dt = dt_model.predict(X_test)

print("\nMétricas de Avaliação para Árvore de Decisão:")
print(f"Acurácia: {accuracy_score(y_test, y_pred_dt):.4f}")
print(f"Precisão: {precision_score(y_test, y_pred_dt):.4f}")
print(f"Recall: {recall_score(y_test, y_pred_dt):.4f}")
print(f"F1-Score: {f1_score(y_test, y_pred_dt):.4f}")

# --- 3. Modelagem com Algoritmos de Agrupamento ---

print("\n--- Modelagem com Algoritmos de Agrupamento (K-Means) ---")

# Para agrupamento, geralmente não usamos a variável target (Survived)
# Usaremos o X_processed_df completo para o agrupamento
X_clustering = X_processed_df.copy()

# Determinar o número ideal de clusters (método do cotovelo - Elbow
Method)
# Nota: Para este exemplo, vamos escolher um k arbitrário para manter o
código conciso.
# Em um cenário real, você faria um loop para diferentes k e plotaria o
WCSS.
# wcss = []
# for i in range(1, 11):
#     kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42,
n_init=10)
#     kmeans.fit(X_clustering)
#     wcss.append(kmeans.inertia_)
# plt.plot(range(1, 11), wcss)
# plt.title('Método do Cotovelo')
# plt.xlabel('Número de Clusters (K)')
# plt.ylabel('WCSS')
# plt.show()
```

```
# Escolhendo K=3 (exemplo)
n_clusters = 3
kmeans = KMeans(n_clusters=n_clusters, init='k-means++', random_state=42,
n_init=10)
clusters = kmeans.fit_predict(X_clustering)
df['Cluster'] = clusters # Adicionar os clusters ao DataFrame original

print(f"\nClusters formados usando K-Means (K={n_clusters}):")
print(df['Cluster'].value_counts())

# Visualização dos Clusters com PCA
print("\nVisualizando Clusters com PCA...")
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_clustering)
df_pca = pd.DataFrame(data=X_pca, columns=['Principal Component 1',
'Principal Component 2'])
df_pca['Cluster'] = clusters
df_pca['Survived'] = df['Survived'] # Adicionar Survived para análise

plt.figure(figsize=(10, 7))
sns.scatterplot(x='Principal Component 1', y='Principal Component 2',
hue='Cluster', palette='viridis', data=df_pca, legend='full', alpha=0.7)
plt.title('Clusters K-Means (PCA 2 Componentes)')
plt.xlabel('Componente Principal 1')
plt.ylabel('Componente Principal 2')
plt.grid(True)
plt.show()

# Interpretação dos Clusters (exemplo - análise das características de
cada cluster)
print("\nInterpretação dos principais padrões encontrados nos clusters:")
for i in range(n_clusters):
    cluster_data = df[df['Cluster'] == i]
    print(f"\n--- Cluster {i} ---")
    print(f"Total de passageiros: {len(cluster_data)}")
    print(f"Taxa de Sobrevivência: {cluster_data['Survived'].mean():.2f}")
```

```
    print("Média de Idade:", cluster_data['Age'].mean())
    print("Média da Tarifa:", cluster_data['Fare'].mean())
    print("Distribuição de Sexo:\n",
cluster_data['Sex'].value_counts(normalize=True)) # 0 para female, 1 para
male
    print("Distribuição de Classe:\n",
cluster_data['Pclass'].value_counts(normalize=True))
    print("Distribuição de Título:\n",
cluster_data['Title'].value_counts(normalize=True).head())

# --- 4. Extração de Regras de Associação ---

print("\n--- Extração de Regras de Associação ---")

# Para regras de associação, precisamos de um formato de transação
(True/False para itens)
# Vamos discretizar algumas variáveis numéricas e usar algumas categóricas
df_assoc = df[['Survived', 'Pclass', 'Sex', 'Age', 'Fare',
'FamilySize']].copy()

# Discretizar 'Age' e 'Fare'
df_assoc['Age_Group'] = pd.cut(df_assoc['Age'], bins=[0, 12, 18, 60, 100],
labels=['Child', 'Teen', 'Adult', 'Elderly'])
df_assoc['Fare_Group'] = pd.cut(df_assoc['Fare'], bins=[-1, 8, 15, 30,
600], labels=['Low', 'Medium', 'High', 'Very_High'])

# Converter todas as colunas para strings para o Apriori
for col in df_assoc.columns:
    df_assoc[col] = df_assoc[col].astype(str)

# Criar o DataFrame de transações (one-hot encoded para Apriori)
# Cada coluna será um item (e.g., 'Sex_male', 'Pclass_1', 'Survived_1')
ohe_assoc = pd.get_dummies(df_assoc)

# Ajustar os nomes das colunas para melhor legibilidade (opcional)
ohe_assoc.columns = [col.replace('_', '=') for col in ohe_assoc.columns]
```

```
# Aplicar o algoritmo Apriori
# min_support: frequência mínima para um conjunto de itens ser considerado
frequente
frequent_itemsets = apriori(ohe_assoc, min_support=0.05,
use_colnames=True)

# Gerar regras de associação
# min_confidence: confiança mínima para uma regra ser considerada
interessante
# lift_constraint: lift mínimo para uma regra ser considerada interessante
rules = association_rules(frequent_itemsets, metric="lift",
min_threshold=1.0)

# Ordenar as regras por lift e confiança
rules = rules.sort_values(['lift', 'confidence'], ascending=[False,
False])

print("\nRegras de Associação Extraídas (ordenadas por Lift e
Confiança):")
print(rules[['antecedents', 'consequents', 'support', 'confidence',
'lift']].head(10))

# Interpretação de Regras Específicas (como pedido na atividade)
print("\n--- Interpretação de Regras Específicas ---")

# Regra 1: {Sexo=Feminino, Classe=1ª} -> {Sobreviveu=Sim}
# Filtrar regras que contenham esses antecedentes e consequentes
rule1 = rules[
    rules['antecedents'].apply(lambda x: frozenset({'Sex=female',
'Pclass=1'}).issubset(x)) &
    rules['consequents'].apply(lambda x:
frozenset({'Survived=1'}).issubset(x))
]
if not rule1.empty:
    print("\nRegra: {Sexo=Feminino, Classe=1ª} -> {Sobreviveu=Sim}")
    print(f"Suporte: {rule1.iloc[0]['support']:.2f}")
    print(f"Confiança: {rule1.iloc[0]['confidence']:.2f}")
```

```
    print(f"Lift: {rule1.iloc[0]['lift']:.2f}")
    print("Interpretação: Esta regra confirma fortemente que ser mulher e
estar na primeira classe está associado a uma alta probabilidade de
sobrevivência, como esperado historicamente.")
else:
    print("\nRegra: {Sexo=Feminino, Classe=1ª} -> {Sobreviveu=Sim} não
encontrada com os parâmetros atuais.")

# Regra 2: {Idade_Grupo=Criança, Classe=3ª} -> {Sobreviveu=Não}
rule2 = rules[
    rules['antecedents'].apply(lambda x: frozenset({'Age_Group=Child',
'Pclass=3'}).issubset(x)) &
    rules['consequents'].apply(lambda x:
frozenset({'Survived=0'}).issubset(x))
]
if not rule2.empty:
    print("\nRegra: {Idade_Grupo=Criança, Classe=3ª} -> {Sobreviveu=Não}")
    print(f"Suporte: {rule2.iloc[0]['support']:.2f}")
    print(f"Confiança: {rule2.iloc[0]['confidence']:.2f}")
    print(f"Lift: {rule2.iloc[0]['lift']:.2f}")
    print("Interpretação: Apesar de crianças terem prioridade, esta regra
pode indicar que a classe social (3ª classe) superou o fator idade em
termos de sobrevivência em certas situações.")
else:
    print("\nRegra: {Idade_Grupo=Criança, Classe=3ª} -> {Sobreviveu=Não}
não encontrada com os parâmetros atuais.")

# Regra 3: {IsAlone=Sim, Sexo=Masculino, Classe=3ª} -> {Sobreviveu=Não}
# Para esta regra, precisamos que 'IsAlone' seja uma feature no ohe_assoc
# Se 'IsAlone' não foi incluída diretamente no get_dummies de df_assoc,
precisamos adicioná-la.
# Vamos assumir que 'IsAlone' foi transformada em 'IsAlone=1' ou
'IsAlone=0'
rule3 = rules[
    rules['antecedents'].apply(lambda x: frozenset({'IsAlone=1',
'Sex=male', 'Pclass=3'}).issubset(x)) &
```

```
    rules['consequents'].apply(lambda x:
frozenset({'Survived=0'}).issubset(x))
]
if not rule3.empty:
    print("\nRegra: {IsAlone=Sim, Sexo=Masculino, Classe=3ª} ->
{Sobreviveu=Não}")
    print(f"Suporte: {rule3.iloc[0]['support']:.2f}")
    print(f"Confiança: {rule3.iloc[0]['confidence']:.2f}")
    print(f"Lift: {rule3.iloc[0]['lift']:.2f}")
    print("Interpretação: Esta regra destaca que ser um homem, viajar
sozinho e estar na 3ª classe eram fatores significativos para a não
sobrevivência.")
else:
    print("\nRegra: {IsAlone=Sim, Sexo=Masculino, Classe=3ª} ->
{Sobreviveu=Não} não encontrada com os parâmetros atuais.")

print("\n--- Fim do Código ---")
```