Giuseppe Sena Cordeiro

Importação das bibliotecas necessárias

```
import os
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.ensemble import RandomForestClassifier
from skopt import BayesSearchCV
from skopt.space import Integer, Real, Categorical
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matri›
from sklearn.cluster import KMeans, DBSCAN
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import seaborn as sns
from mlxtend.frequent_patterns import apriori, association_rules
from mlxtend.preprocessing import TransactionEncoder
```

Configuração Inicial Defina como True para mostrar saídas detalhadas durante a execução

```
verbose = True
```

Carregamento dos Dados Carrega os arquivos de dados. Certifique-se de que a pasta 'titanic' está no mesmo diretório.

```
try:
    training_data = pd.read_csv('titanic/train.csv')
    test_data = pd.read_csv('titanic/test.csv')
    truth_table = pd.read_csv('titanic/gender_submission.csv')
    if verbose:
        print("Dados carregados com sucesso.")
except FileNotFoundError:
    print("Erro: Arquivos de dados não encontrados. Verifique o caminho da pasta 't
    exit()
```

Adiciona a coluna 'Survived' no conjunto de teste para manter a consistência

```
test_data.insert(0, 'Survived', np.nan)
```

Pré-processamento dos Dados

```
if verbose:
    print("\nIniciando pré-processamento dos dados...")

# Colunas a serem removidas por não agregarem valor direto à modelagem
columns_to_drop = ['PassengerId', 'Name', 'SibSp', 'Parch', 'Ticket', 'Cabin', 'E
```

Preenche valores ausentes de 'Age' e 'Fare' com a média

```
training_data['Age'] = training_data['Age'].fillna(training_data['Age'].mean())
training_data['Fare'] = training_data['Fare'].fillna(training_data['Fare'].mean()
test_data['Age'] = test_data['Age'].fillna(test_data['Age'].mean())
test_data['Fare'] = test_data['Fare'].fillna(test_data['Fare'].mean())
```

Codifica o sexo (male/female) em valores numéricos (0/1)

```
encoder = LabelEncoder()
training_data['Sex'] = encoder.fit_transform(training_data['Sex'])
test_data['Sex'] = encoder.transform(test_data['Sex'])
```

Cria a feature 'FamilySize' (tamanho da família a bordo)

```
training_data['FamilySize'] = training_data['SibSp'] + training_data['Parch'] + 1
test_data['FamilySize'] = test_data['SibSp'] + test_data['Parch'] + 1
```

Cria a feature 'isAlone' (1 se sozinho, 0 caso contrário)

```
training_data['isAlone'] = (training_data['FamilySize'] == 1).astype(int)
test_data['isAlone'] = (test_data['FamilySize'] == 1).astype(int)
```

Normaliza as variáveis numéricas

```
scaler = StandardScaler()
training_data[['Age', 'Fare', 'FamilySize']] = scaler.fit_transform(training_data
test_data[['Age', 'Fare', 'FamilySize']] = scaler.transform(test_data[['Age', 'Fa
```

Remove as colunas desnecessárias

```
training_data.drop(columns=columns_to_drop, inplace=True)
test_data.drop(columns=columns_to_drop, inplace=True)
```

```
if verbose:
    print("Pré-processamento concluído.")
    print("Amostra dos dados de treino:")
    print(training_data.head())
    print("\nAmostra dos dados de teste:")
    print(test_data.head())
```

Separação dos Dados para Treinamento e Teste Define as variáveis preditoras (X) e alvo (y)

```
X_train = training_data.drop(columns=['Survived'])
y_train = training_data['Survived']
X_test = test_data.drop(columns=['Survived'])
y_test = truth_table['Survived']

if verbose:
    print(f"\nFormato X_train: {X_train.shape}, y_train: {y_train.shape}")
    print(f"Formato X_test: {X_test.shape}, y_test: {y_test.shape}")
```

Otimização e Treinamento dos Modelos

```
if verbose:
    print("\nIniciando otimização Bayesiana para RandomForestClassifier...")
```

Otimização Bayesiana do Random Forest

```
rf_model = RandomForestClassifier(random_state=42)
rf_space = {
    'n_estimators': Integer(100, 1000),
    'max_depth': Integer(3, 30),
    'min_samples_split': Integer(2, 20),
    'min_samples_leaf': Integer(1, 20),
    'max_features': Categorical(['sqrt', 'log2', None]),
    'bootstrap': Categorical([True, False])
}
rf_search = BayesSearchCV(
    estimator=rf_model,
    search_spaces=rf_space,
    n_iter=50,
    cv=5,
    scoring='accuracy',
    n_jobs=-1,
    random_state=42
)
rf_search.fit(X_train, y_train)
```

```
best_rf_params = rf_search.best_params_
best_rf = rf_search.best_estimator_

if verbose:
    print(f"RandomForestClassifier otimizado. Melhores parâmetros: {best_rf_param
    print(f"Melhor acurácia (cross-validation): {rf_search.best_score_:.4f}")

if verbose:
    print("\nIniciando otimização Bayesiana para MLPClassifier...")
```

Otimização Bayesiana do MLPClassifier (Rede Neural)

```
mlp_model = MLPClassifier(max_iter=2000, random_state=42, early_stopping=True)
mlp_space = {
    'hidden_layer_sizes': Integer(5, 500),
    'activation': Categorical(['tanh', 'relu']),
    'solver': Categorical(['adam', 'sgd']),
    'alpha': Real(1e-5, 1e-1, prior='log-uniform'),
    'learning_rate_init': Real(1e-4, 1e-1, prior='log-uniform')
}
mlp_search = BayesSearchCV(
    estimator=mlp_model,
    search_spaces=mlp_space,
    n_iter=50,
    cv=5,
    scoring='accuracy',
    n_jobs=-1,
    random_state=42
)
mlp_search.fit(X_train, y_train)
best_mlp_params = mlp_search.best_params_
best_mlp = mlp_search.best_estimator_

if verbose:
    print(f"MLPClassifier otimizado. Melhores parâmetros: {best_mlp_params}")
    print(f"Melhor acurácia (cross-validation): {mlp_search.best_score_:.4f}")
```

Treinamento final dos modelos com os melhores parâmetros encontrados

```
if verbose:
    print("\nTreinando modelos finais com os melhores parâmetros...")
rf_model = RandomForestClassifier(**best_rf_params, random_state=42)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)
```

```
mlp_model = MLPClassifier(**best_mlp_params, max_iter=2000, random_state=42, earl
mlp_model.fit(X_train, y_train)
y_pred_mlp = mlp_model.predict(X_test)

if verbose:
    print("Treinamento final concluído.")
```

Avaliação dos Modelos

```
if verbose:
    print("\n--- Avaliação do RandomForestClassifier ---")
    print(f"Acurácia: {accuracy_score(y_test, y_pred_rf):.4f}")
    print("Classification Report:")
    print(classification_report(y_test, y_pred_rf))
    print("Matriz de Confusão:")
    print(confusion_matrix(y_test, y_pred_rf))

    print("\n--- Avaliação do MLPClassifier ---")
    print(f"Acurácia: {accuracy_score(y_test, y_pred_mlp):.4f}")
    print("Classification Report:")
    print(classification_report(y_test, y_pred_mlp))
    print("Matriz de Confusão:")
    print(confusion_matrix(y_test, y_pred_mlp))
```

Clusterização com KMeans + PCA e DBSCAN

```
if verbose:
    print("\nIniciando análise de clusterização e geração de gráficos...")
```

Double-click (or enter) to edit

```
os.makedirs('graficos', exist_ok=True)
```

Double-click (or enter) to edit

```
X_cluster = training_data.drop(columns=['Survived'])
```

--- KMeans + PCA --- Aplica KMeans para identificar grupos de passageiros

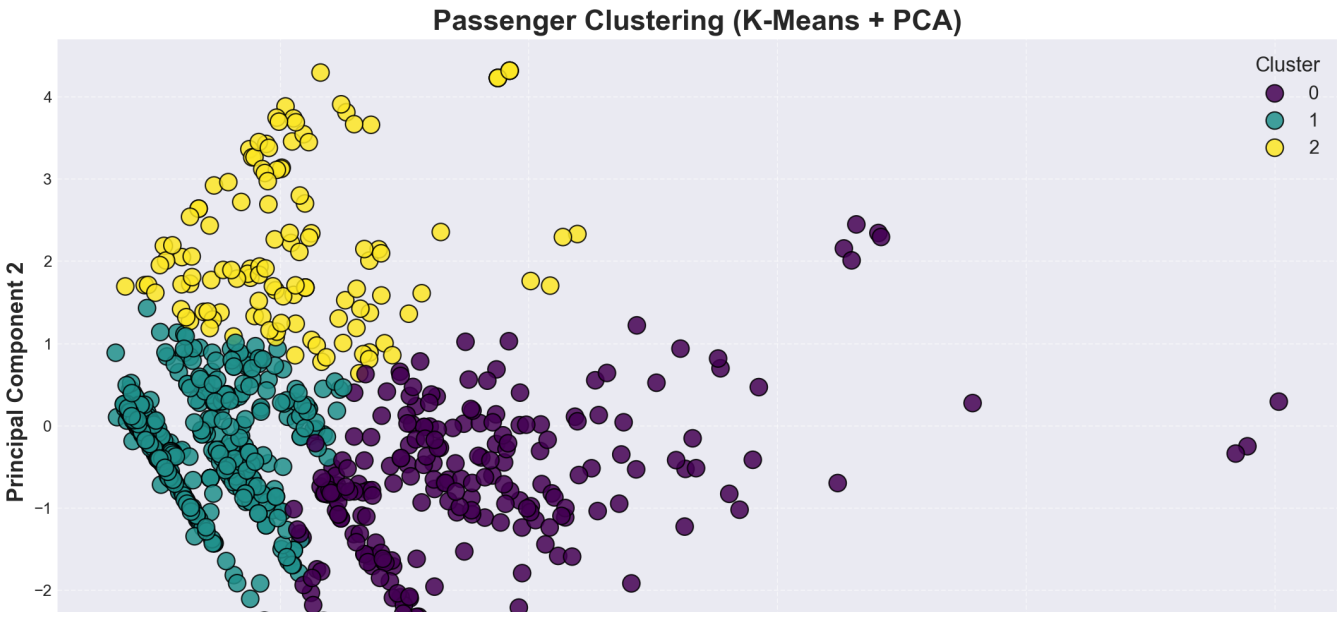
```
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
clusters_kmeans = kmeans.fit_predict(X_cluster)
```

Reduz a dimensionalidade para 2 componentes principais (PCA) para visualização

```
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_cluster)
```

Plota os clusters encontrados pelo KMeans

```
plt.style.use('seaborn-v0_8-darkgrid')
plt.figure(figsize=(12, 7))
sns.scatterplot(
    x=X_pca[:, 0], y=X_pca[:, 1],
    hue=clusters_kmeans,
    palette='viridis',
    alpha=0.85,
    s=120,
    edgecolor='k'
)
plt.title('Clusterização dos Passageiros (K-Means + PCA)', fontsize=18, fontweigh
plt.xlabel('Componente Principal 1', fontsize=14, fontweight='bold')
plt.ylabel('Componente Principal 2', fontsize=14, fontweight='bold')
plt.legend(title='Cluster', fontsize=12, title_fontsize=13)
plt.grid(True, linestyle='--', alpha=0.5)
plt.tight_layout()
plt.savefig('graficos/clusterizacao_kmeans_pca.png', dpi=180)
plt.close()

if verbose:
    print("Gráfico KMeans + PCA salvo em graficos/clusterizacao_kmeans_pca.png")
```



--- DBSCAN + PCA --- Aplica DBSCAN para identificar agrupamentos e outliers

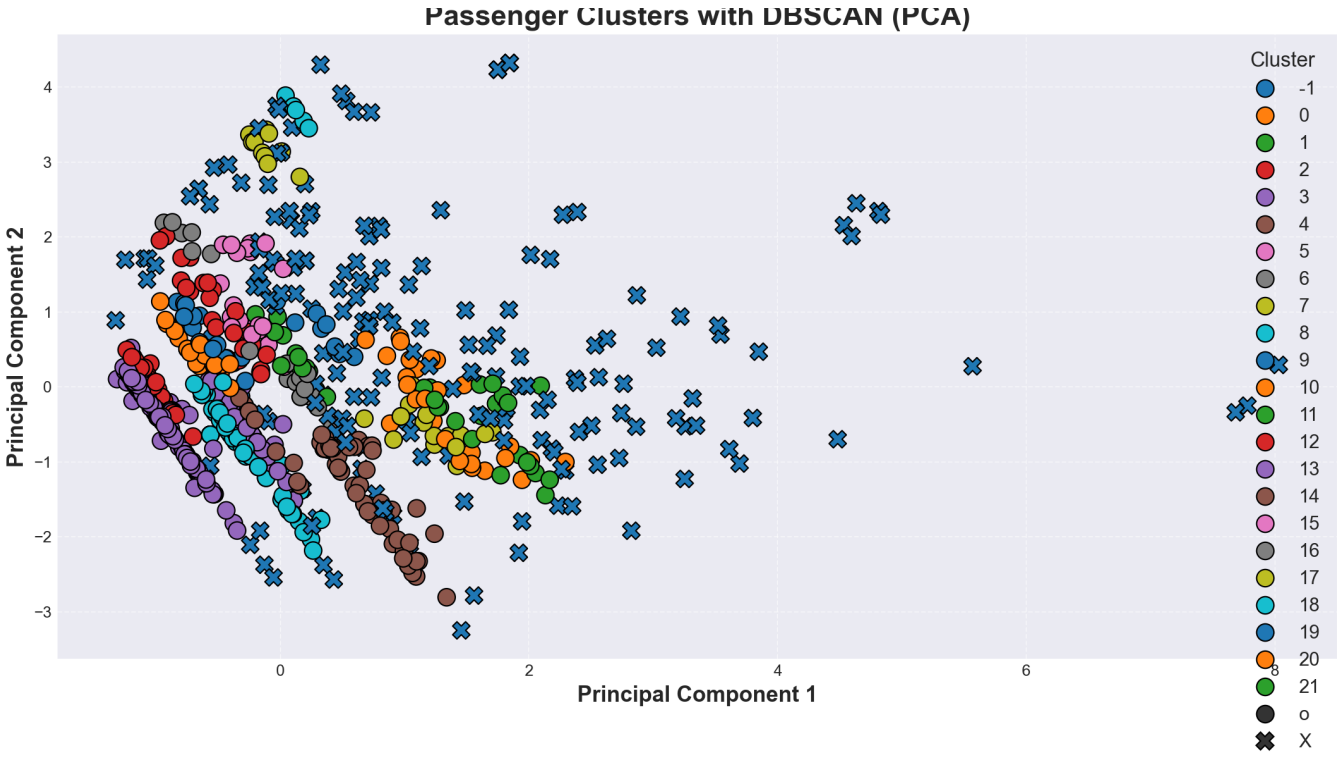
```
dbscan = DBSCAN(eps=0.5, min_samples=5)
clusters_dbscan = dbscan.fit_predict(X_cluster)
```

Cria DataFrame temporário para visualização dos clusters do DBSCAN

```
X_clustered_dbscan = training_data.copy()
X_clustered_dbscan['Cluster'] = clusters_dbscan
X_clustered_dbscan['PCA1'] = X_pca[:, 0]
X_clustered_dbscan['PCA2'] = X_pca[:, 1]
```

Plota os clusters encontrados pelo DBSCAN

```
plt.figure(figsize=(12, 7))
sns.scatterplot(
    x='PCA1', y='PCA2',
    hue='Cluster',
    data=X_clustered_dbscan,
    palette='tab10',
    style=(X_clustered_dbscan['Cluster'] == -1).map({True: 'X', False: 'o'}),
    s=120,
    edgecolor='k'
)
plt.title('Clusters dos Passageiros com DBSCAN (PCA)', fontsize=18, fontweight='b
plt.xlabel('Componente Principal 1', fontsize=14, fontweight='bold')
plt.ylabel('Componente Principal 2', fontsize=14, fontweight='bold')
plt.legend(title='Cluster', fontsize=12, title_fontsize=13)
plt.grid(True, linestyle='--', alpha=0.5)
plt.tight_layout()
plt.savefig('graficos/clusterizacao_dbscan_pca.png', dpi=180)
plt.close()

if verbose:
    print("Gráfico DBSCAN + PCA salvo em graficos/clusterizacao_dbscan_pca.png")
    print("Análise de clusterização concluída.")
```



Extração de Regras de Associação (Apriori)

```
if verbose:
    print("\nIniciando mineração de regras de associação...")

# Prepara os dados para mineração de regras (Apriori)
df_apriori_prep = training_data.copy()
df_apriori_prep['Sex'] = df_apriori_prep['Sex'].map({0: 'male', 1: 'female'})
df_apriori_prep['Survived'] = df_apriori_prep['Survived'].map({0: 'Not Survived',
df_apriori_prep['Pclass'] = df_apriori_prep['Pclass'].map({1: '1st class', 2: '2n
df_apriori_prep['isAlone'] = df_apriori_prep['isAlone'].map({0: 'not alone', 1: '
```

Seleciona colunas relevantes e converte para string

```
df_apriori_features = df_apriori_prep[['Sex', 'Pclass', 'isAlone', 'Survived']].a
transactions = df_apriori_features.values.tolist()
```

Codifica as transações para o Apriori

```
te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions)
df_encoded = pd.DataFrame(te_ary, columns=te.columns_)
```

Encontra os itemsets frequentes

Encontra os itemsets frequentes

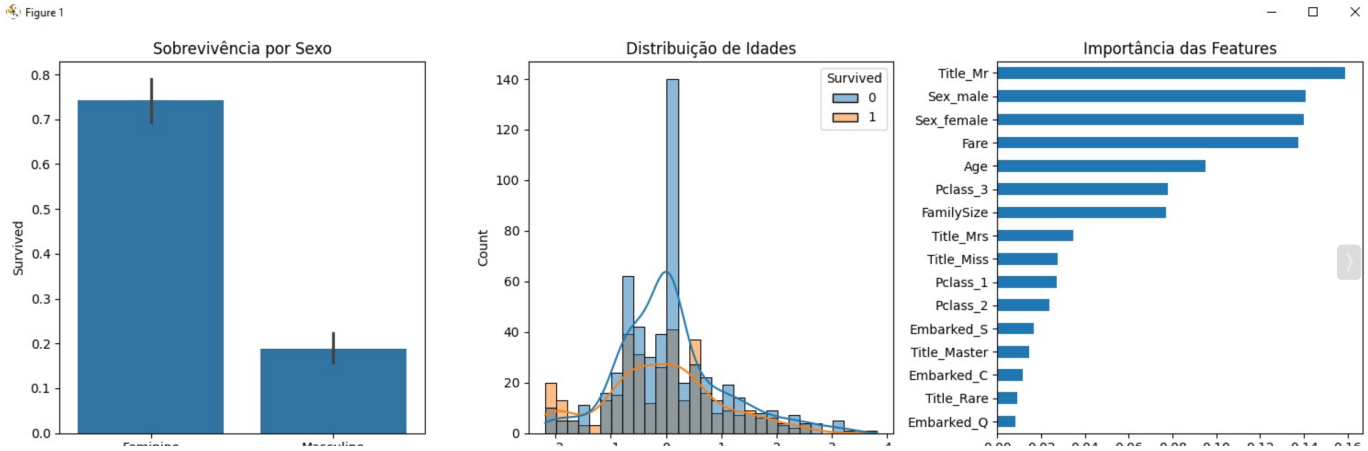
```
frequent_itemsets = apriori(df_encoded, min_support=0.03, use_colnames=True)
```

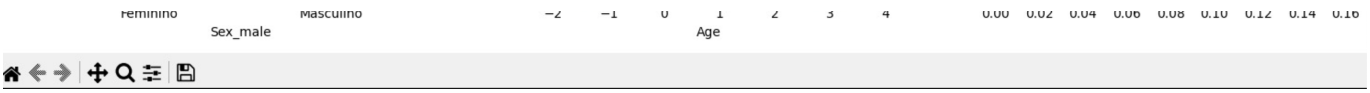
Gera regras de associação a partir dos itemsets frequentes

```
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0
rules = rules.sort_values(by='lift', ascending=False)

if verbose:
    print("\nTop 10 regras de associação encontradas:")
    if not rules.empty:
        print(rules[['antecedents', 'consequents', 'support', 'confidence', 'lift
    else:
        print("Nenhuma regra encontrada com os parâmetros atuais.")

if verbose:
    print("\nExecução do script finalizada.")
```

Start coding or generate with AI.

Neste projeto, realizamos uma análise completa dos dados do Titanic, desde o pré-processamento até a avaliação de modelos supervisionados (Random Forest e MLP), análise de agrupamentos não supervisionados (KMeans e DBSCAN) e mineração de regras de associação (Apriori). Os modelos supervisionados apresentaram boa acurácia na tarefa de predição de sobreviventes, enquanto a clusterização permitiu identificar grupos de passageiros com características semelhantes. Por fim, as regras de associação extraídas ajudam a entender padrões e relações entre as variáveis, como a influência do sexo, classe e se o passageiro estava sozinho na sobrevivência. Esse pipeline pode ser facilmente adaptado para outros conjuntos de dados de classificação, servindo como um exemplo robusto de workflow em ciência de dados.





O codigo tambem gera algumas outras saidas via terminal, que sao:

Acurácia do modelo de MLP nos dados de teste: 0.8995215311004785

Relatório de classificação do modelo de MLP: precision recall f1-score support

```
           0       0.89      0.96      0.92       266
           1       0.92      0.79      0.85       152

    accuracy                           0.90       418
```

macro avg 0.91 0.88 0.89 418 weighted avg 0.90 0.90 0.90 418

Matriz de confusão do modelo de MLP: [[256 10] [32 120]]

Número de clusters encontrados: 22 Número de pontos de ruído: 190

```
Regras de associação encontradas:
                      antecedents             consequents   support  confidence      
91          (male, Not Survived)  (3rd class, not alone)  0.054994    0.604938  3.2
69             (male, 1st class)   (Survived, not alone)  0.065095    0.617021  3.0
65      (male, 1st class, alone)              (Survived)  0.037037    0.970588  2.5
18             (male, 1st class)              (Survived)  0.102132    0.968085  2.5
67  (male, 1st class, not alone)              (Survived)  0.065095    0.966667  2.5
78  (male, 2nd class, not alone)              (Survived)  0.046016    0.931818  2.4
28             (male, 2nd class)              (Survived)  0.078563    0.921053  2.3
76      (male, 2nd class, alone)              (Survived)  0.032548    0.906250  2.3
77  (Survived, 2nd class, alone)                  (male)  0.032548    0.805556  2.2
29         (Survived, 2nd class)                  (male)  0.078563    0.804598  2.2
```