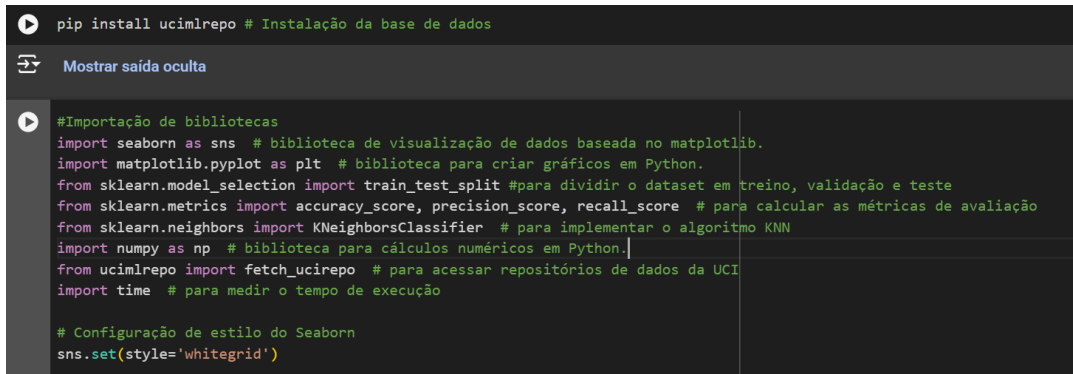


Questão 2 -

O primeiro passo para resolver essa questão foi a importação e instalação das bibliotecas necessárias para a visualização e execução do código, conforme a Figura 1. O próximo passo foi estabelecer o estilo dos gráficos.



```
pip install ucimlrepo # Instalação da base de dados

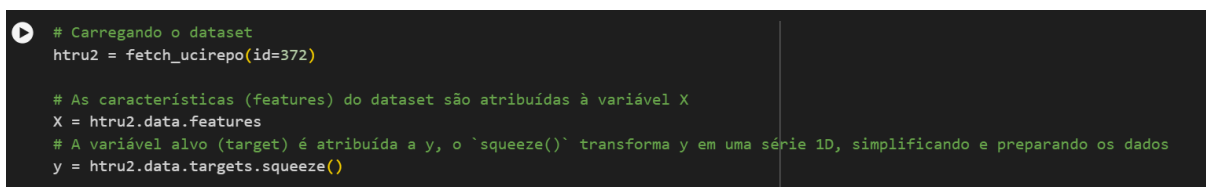
Mostrar saída oculta

#Importação de bibliotecas
import seaborn as sns # biblioteca de visualização de dados baseada no matplotlib.
import matplotlib.pyplot as plt # biblioteca para criar gráficos em Python.
from sklearn.model_selection import train_test_split #para dividir o dataset em treino, validação e teste
from sklearn.metrics import accuracy_score, precision_score, recall_score # para calcular as métricas de avaliação
from sklearn.neighbors import KNeighborsClassifier # para implementar o algoritmo KNN
import numpy as np # biblioteca para cálculos numéricos em Python.
from ucimlrepo import fetch_ucirepo # para acessar repositórios de dados da UCI
import time # para medir o tempo de execução

# Configuração de estilo do Seaborn
sns.set(style='whitegrid')
```

Figura 1. Primeira parte do código. (Do Autor)

Na figura 2, podemos visualizar o carregamento dos dados, onde X recebe as características e y a variável alvo. O método .squeeze() permite simplificar e preparar os dados atribuídos a y, transformando-os em 1D.

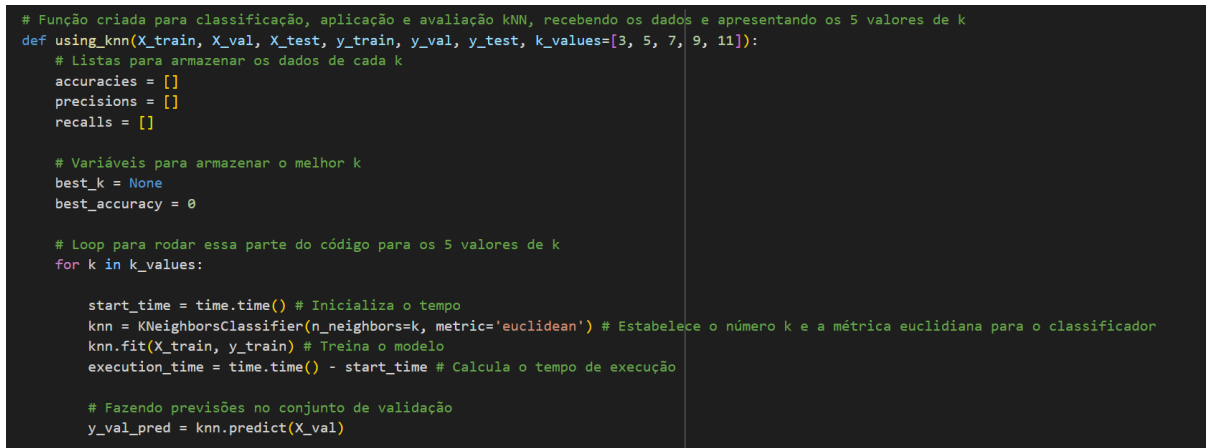


```
# Carregando o dataset
htru2 = fetch_ucirepo(id=372)

# As características (features) do dataset são atribuídas à variável X
X = htru2.data.features
# A variável alvo (target) é atribuída a y, o `squeeze()` transforma y em uma série 1D, simplificando e preparando os dados
y = htru2.data.targets.squeeze()
```

Figura 2. Segunda parte do código. (Do Autor)

A função `using_knn`, conforme a Figura 3, foi criada a fim de treinar, avaliar e definir o melhor k, além de permitir a plotagem dos gráficos para melhor visualização dos dados. Assim, foram criadas as listas que armazenam os valores de cada k, possibilitando o cálculo posterior da média de cada execução. Um loop também foi criado, permitindo que essa parte do código seja repetida para todos os 5 valores de k. O tempo é devidamente calculado para posterior plotagem e o classificador KNN é determinado pela métrica de distância euclidiana.



```
# Função criada para classificação, aplicação e avaliação kNN, recebendo os dados e apresentando os 5 valores de k
def using_knn(X_train, X_val, X_test, y_train, y_val, y_test, k_values=[3, 5, 7, 9, 11]):
    # Listas para armazenar os dados de cada k
    accuracies = []
    precisions = []
    recalls = []

    # Variáveis para armazenar o melhor k
    best_k = None
    best_accuracy = 0

    # Loop para rodar essa parte do código para os 5 valores de k
    for k in k_values:
        start_time = time.time() # Inicializa o tempo
        knn = KNeighborsClassifier(n_neighbors=k, metric='euclidean') # Estabelece o número k e a métrica euclidiana para o classificador
        knn.fit(X_train, y_train) # Treina o modelo
        execution_time = time.time() - start_time # Calcula o tempo de execução

        # Fazendo previsões no conjunto de validação
        y_val_pred = knn.predict(X_val)
```

Figura 3. Função `using_knn`. (Do Autor)

Os dados foram divididos em 6000 amostras de treino, selecionadas aleatoriamente a cada execução, e o restante para teste. Enquanto, para a determinação do melhor k, essas amostras de treino foram subdivididas na metade, uma para treino e outra para validação.

```
# Divisão do dataset em treino e teste, e em seguida o treino é dividido em treino e validação
X_trainfull, X_test, y_trainfull, y_test = train_test_split(X, y, train_size=6000, stratify=y) # Para cada execução, a
X_train, X_val, y_train, y_val = train_test_split(X_trainfull, y_trainfull, test_size=0.5, stratify=y_trainfull) # Div
```

Figura 4. Divisão dos dados. (Do Autor)

Dessa forma, o modelo foi treinado com os dados de treino e posteriormente validado pelo conjunto de validação. Para a determinação do melhor k, a acurácia é obtida com a aplicação do modelo no conjunto de validação, como demonstrado na figura 5.

```
start_time = time.time() # Inicializa o tempo
knn = KNeighborsClassifier(n_neighbors=k, metric='euclidean') # Estabelece o número k e a métrica euclidiana para o classificador
knn.fit(X_train, y_train) # Treina o modelo
execution_time = time.time() - start_time # Calcula o tempo de execução

# Fazendo previsões no conjunto de validação
y_val_pred = knn.predict(X_val)

# Calculando as métricas de avaliação no conjunto de validação
precision = precision_score(y_val, y_val_pred)
recall = recall_score(y_val, y_val_pred)
accuracy = accuracy_score(y_val, y_val_pred)

# Encontrando e armazenando o melhor valor de k com base na acurácia
if accuracy > best_accuracy:
    best_accuracy = accuracy
    best_k = k
```

Figura 5. Treinamento e avaliação do melhor k. (Do Autor)

Após o loop descrito acima, o modelo do melhor k é aplicado nos dados de teste e os resultados são printados na tela.

```
# Treina o modelo final usando o melhor k encontrado e avalia no conjunto de teste
knn_best = KNeighborsClassifier(n_neighbors=best_k, metric='euclidean')
knn_best.fit(X_train, y_train)
y_test_pred = knn_best.predict(X_test)

# Calcula e exibe as métricas finais no conjunto de teste
print("-" * 30)
print(f"Melhor k encontrado com base na acurácia: {best_k}")
print("Resultados no conjunto de teste final:")
print(f"  Acurácia: {accuracy_score(y_test, y_test_pred):.4f}")
print(f"  Precisão: {precision_score(y_test, y_test_pred):.4f}")
print(f"  Recall: {recall_score(y_test, y_test_pred):.4f}")
print(f"  Número de Protótipos Usados: {len(X_test)}")
print("-" * 30)
```

Figura 6. Modelo do melhor k aplicada no conjunto teste. (Do Autor)

Os gráficos plotados representam os valores de acurácia, precisão e recall de cada k, além de marcar a média dos valores da execução, conforme o exemplo representado na figura 7.

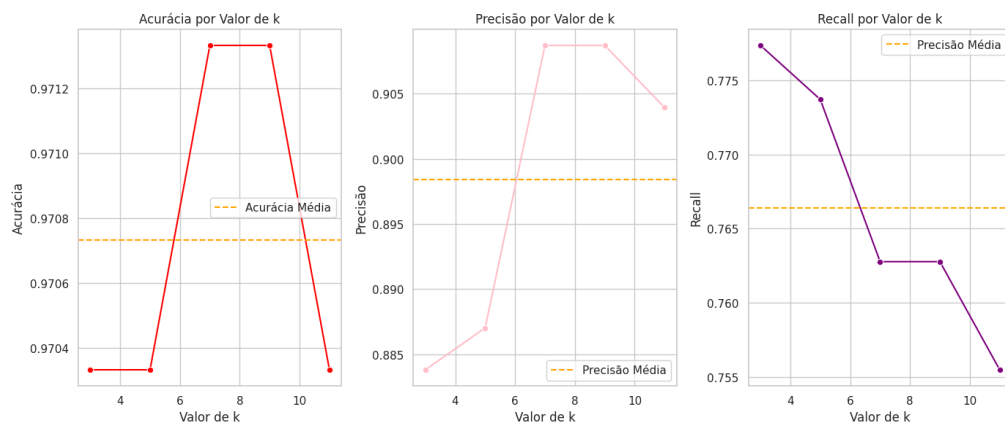


Figura 7.Exemplo dos gráficos. (Do Autor)

Para finalizar, o loop ‘for i in range(5):’, permite que o treinamento e teste seja executado 5 vezes, retornando a acurácia, recall e precisão média para cada algoritmo, conforme solicitado no enunciado da questão.

Os resultados, tempo de execução e número de protótipos foram plotados para cada k usado no código, podendo ser, dessa forma, comparados. Vale ressaltar que o número de protótipos usados em cada análise e execução é fixo e determinado na divisão do dataset. Outro ponto é o fato de que, normalmente, quanto maior o valor de k , maior será o tempo e o gasto computacional para a execução.

Considerando o gráfico 1, retirado da base de dados, podemos verificar a concentração dos dados em uma só classe, sendo desbalanceada. Dessa forma, a avaliação apenas pela acurácia pode representar um panorama geral, mas não uma visão completa do modelo. Para uma melhor visualização deve-se considerar outras métricas, como precisão e recall, garantindo que o modelo seja eficaz para todas as classes de dados e não apenas para a maioria. Métodos como o “stratify” usado na divisão do dataset, tentam amenizar a tendência provocada pelo desbalanceamento dos dados, pois ele permite que a proporção de classes no conjunto de treinamento e no conjunto de teste seja a mesma que no conjunto de dados original.

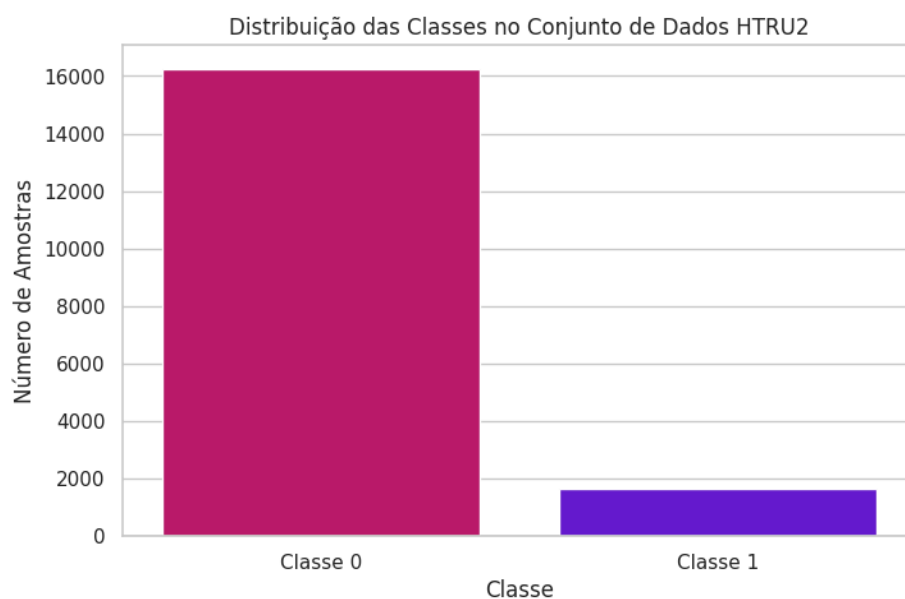


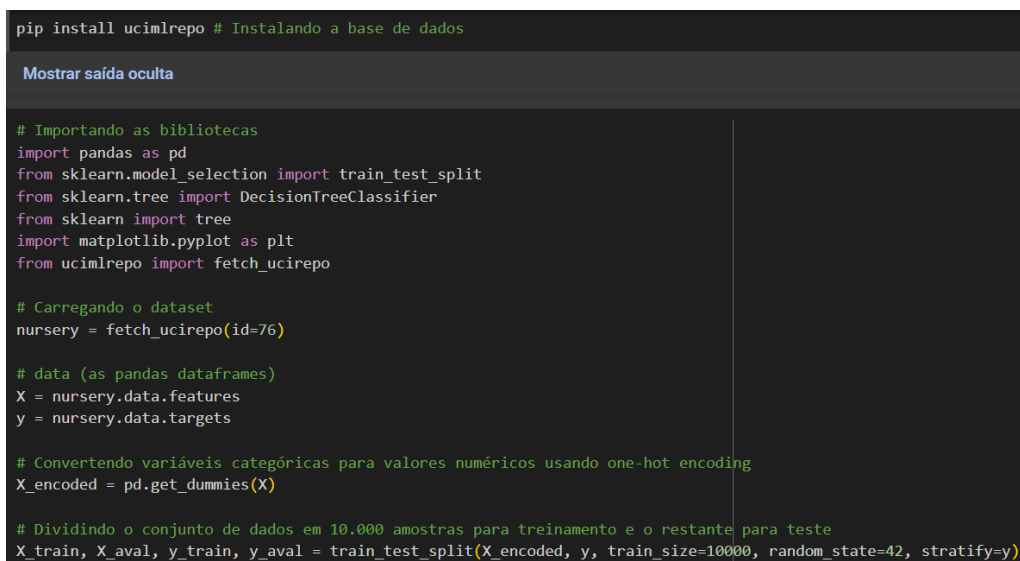
Gráfico 1. Distribuição das classes na base de dados. (Do Autor)

Questão 3 -

Essa base de dados contém informações relacionadas à admissão de crianças em creches e possui diversas características que indicam critérios importantes para esta classificação. Para essa análise foi solicitado uma árvore de decisão, implementada com os passos descritos abaixo.

Primeiramente, conforme pode-se observar na figura 8, foi realizada a instalação da base de dados sugerida, assim como a importação das bibliotecas necessárias para a execução e visualização das questões propostas. Outro ponto importante é a transformação das variáveis categóricas em variáveis numéricas, permitindo o uso desses dados na formação da árvore de decisão, para isso foi usado o método de one-hot encoding (`X_encoded = pd.get_dummies(X)`).

O próximo passo adotado foi a divisão da base de dados, conforme o enunciado, em 10000 amostras aleatórias para treinamento do modelo e as demais para avaliação. Acredito ser importante ressaltar as instâncias `random_state=42` e `stratify=y`, o `random_state` permite que os dados sejam aleatórios, mas reproduzíveis, assim, mesmo em outras execuções, terá o mesmo resultado, enquanto o `stratify` ameniza possíveis desbalanceamentos de dados.



```
pip install ucimlrepo # Instalando a base de dados

Mostrar saída oculta

# Importando as bibliotecas
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
import matplotlib.pyplot as plt
from ucimlrepo import fetch_ucirepo

# Carregando o dataset
nursery = fetch_ucirepo(id=76)

# data (as pandas dataframes)
X = nursery.data.features
y = nursery.data.targets

# Convertendo variáveis categóricas para valores numéricos usando one-hot encoding
X_encoded = pd.get_dummies(X)

# Dividindo o conjunto de dados em 10.000 amostras para treinamento e o restante para teste
X_train, X_aval, y_train, y_aval = train_test_split(X_encoded, y, train_size=10000, random_state=42, stratify=y)
```

Figura 8. Instalação e importação de bibliotecas, além de tratamento e divisão de dados. (Do Autor)

A árvore de decisão foi criada com base no critério de Ganho de Informação, indicada no código através do comando: `criterion='entropy'`. A profundidade de dois níveis de nós de decisão foi determinada considerando que os nós de decisão são os nós que obtemos após dividir os nós raiz, fato determinado no código por: `max_depth=2`. Em seguida, o modelo foi treinado com os dados de treino e plotados usando a função `tree.plot_tree` da biblioteca `sklearn`. Também foi determinado os parâmetros como os nomes das características e das classes, a cor dos nós, as bordas e o tamanho da fonte, conforme a figura 9.

```

# Letra a)
# Criando o modelo de árvore de decisão com profundidade máxima de 2
decision_tree = DecisionTreeClassifier(criterion='entropy', max_depth=2, random_state=42)
decision_tree.fit(X_train, y_train) # Treinando o modelo
plt.figure(figsize=(12, 8)) # Configurações figura
tree.plot_tree(decision_tree,
               feature_names=list(X_train.columns), # Nomes das características
               class_names=decision_tree.classes_, # Nomes das classes
               filled=True, # Preenche os nós com cores baseadas na classe
               rounded=True, # Arredonda os cantos dos nós
               fontsize=10) # Tamanho da fonte

plt.show()

```

Figura 9. Criação da árvore de decisão. (Do Autor)

Para a letra b, a acurácia foi calculada aplicando o modelo nos dados de validação/teste, através do método `decision_tree.score`, conforme a figura abaixo.

```

[ ] # Letra b)
# Avaliando a acurácia do modelo nos dados de teste
accuracy = decision_tree.score(X_aval, y_aval)
print(f"\nAcurácia do modelo: {accuracy:.4f}")
print(".*30)

```

Figura 10. Desenvolvimento letra b. (Do Autor)

Para responder a letra c do problema, as regras de decisão da árvore foram extraídas pelo comando `tree.export_text`. As três linhas seguintes do código foram tratativas dessa saída das regras, a fim de melhorar a visualização desses dados. E a última linha, plota as regras de decisão na tela.

```

# Extraindo as regras de decisão da árvore obtida na letra A
rules = tree.export_text(decision_tree, feature_names=list(X_train.columns))

# Organizando a saída das regras para uma melhor leitura
formatted_rules = "\nRegras de Decisão da Árvore de Decisão:\n"
formatted_rules += "." * 50 + "\n"
formatted_rules += rules.replace("|---", " ").replace("|", "").replace("class:", "=> Classe Prevista:")

# Printando as regras de decisão
print(formatted_rules)

```

```

Regras de Decisão da Árvore de Decisão:
.....
health_not_recom <= 0.50
  has_nurs_very_crit <= 0.50
    => Classe Prevista: priority
  has_nurs_very_crit > 0.50
    => Classe Prevista: spec_prior
health_not_recom > 0.50
  => Classe Prevista: not_recom

```

Figura 11. Desenvolvimento letra c. (Do Autor)