Desenvolvimento:

- Importação de Bibliotecas;
- Extração dos dados do arquivo e tratamento (separação de features e classes, e divisão de 25% para os dados de teste usando train_test_split);
- Visualização gráfica dos dados (com matplotlib.pyplot);
- Execução de algoritmo k-nearest neighbors usando a biblioteca sklearn
    - Execução para K=1, 2, 3, 5, 7 e 9, e cálculo de acurácia com accuracy_score para cada execução.
- Desenvolvimento de algoritmo k-nearest neighbors
    - Importação do "most_frequent" para determinar o elemento mais frequente em uma lista.
    - Execução do algoritmo desenvolvido para K=1, 2, 3, 5, 7 e 9, e cálculo de acurácia com accuracy_score para cada execução.

Resultados:

Execução 1

Tamanho do conjunto de testes	K	sklearn.neighbors (%)	Algoritmo sem sklearn (%)
0.25	1	94.73684	94.73684
0.25	3	94.73684	94.73684
0.25	5	94.73684	94.73684
0.25	7	92.10526	92.10526
0.25	9	92.10526	92.10526

Execução 2

Tamanho do conjunto de testes	K	sklearn.neighbors (%)	Algoritmo sem sklearn (%)
0.25	1	97.36842	97.36842
0.25	3	97.36842	97.36842
0.25	5	97.36842	97.36842
0.25	7	100.0	100.0
0.25	9	100.0	100.0

Execução 3

Tamanho do conjunto de testes	K	sklearn.neighbors (%)	Algoritmo sem sklearn (%)
0.25	1	92.10526	92.10526
0.25	3	94.73684	94.73684
0.25	5	94.73684	94.73684
0.25	7	97.36842	97.36842
0.25	9	94.73684	94.73684

Para cada execução é feita uma divisão diferente entre dados de teste e de treinamento, mantendo seus respectivos tamanhos e trazendo resultados diferentes. Não há diferença no resultado do KMN obtido via sklearn.neighbors e o obtido sem o uso da biblioteca.

Com exceção da primeira execução, a taxa de acerto tem sido melhor a cada K maior, podendo, entretanto, levar ao problema do overfitting (exposto mais claramente na segunda execução). O valor de k=5 aparentemente foi o que melhor trouxe equilíbrio na execução, com uma baixa taxa de erros e sem overfitting.