

K-Nearest Neighbors (KNN): Practical Tutorial on UCI Heart Disease Data

Name: Namra Islam

Github Repo: [namra-123/Machine-Learning-and-Neural-Networks](#)

Table of Contents

1. Introduction.....	3
2. Dataset Overview	3
3. Loading and Preparing the Data	4
4. Why Scaling Matters for KNN	6
Classification report:	8
Confusion matrix:	8
5. Exploring Different Values of k	8
6. Final Model Evaluation.....	9
Confusion matrix:	10
7. The Interpretation of KNN in Terms of Bias-Variance Trade-off.....	11
8. Why the Heart Disease Dataset is a Good Case Study	11
9. Pictorial Presentation of the Results	12
10. Ethical and Accessibility Issues	13
Ethical aspects include:.....	13
Risk of misclassification:	13
Data representativeness:.....	13
Transparency:.....	13
11. Relating the Findings to Machine Learning Concepts:.....	13
Importance of preprocessing:.....	14
Model complexity vs generalization:.....	14
Reproducibility:	14
12. Guidelines on the Practical Use of KNN	14
13. Conclusion	15

1. Introduction

K-Nearest Neighbors (KNN) is among the most intuitive and least complex supervised learning algorithms despite it being an important baseline method in a broad spectrum of applications of machine learning in the real world (Guo et al., 2003). Originally introduced by Cover and Hart (1967), KNN is an instance-based technique, unlike parametric models which learn explicit decision boundaries: it saves the training data and predicts new observations by analysing the nearest previously observed points. Due to this local behaviour, KNN enables us to learn some of the basic concepts of machine-learning, including the impact of distance measures (Prasatha et al., 2017), scaling of the data, or the bias-variance trade-off that is formed by the hyperparameter k —the number of neighbours (Hastie, Tibshirani and Friedman, 2009). The following properties are what render KNN to be suitable in an educational tutorial where the goal is to illustrate machine-learning reasoning in a clear and reproducible manner.

In this tutorial, we concentrate on the variation of model performance with the variation of the values of k and the importance of using appropriate preprocessing to enhance the generalisation of the models. We use the UCI Heart Disease dataset (Dua and Graff, 2019) as it has a medical problem of interest (presence or absence of disease) and contains both continuous variables and is not too large to enable quick experimentation—a key trait of a teaching and learning task. We do not simply intend to demonstrate the implementation of KNN but to give the reader sufficient knowledge that he or she can use the technique in his or her work.

There is both a runnable code and generated figures as well as a coherent description of results in this tutorial. Each of the code excerpts presented below is also present in the Jupyter notebook accompanying the assignment, which meets the requirement of the assignment to be reproducible. The implementation utilises the scikit-learn library (Pedregosa et al., 2011), which provides efficient tools for machine learning in Python.

2. Dataset Overview

The UCI Heart Disease data is a collection of patient-level clinical values, including age, resting blood pressure, cholesterol level, maximal heart rate, and angina caused by exercising (Dua

and Graff, 2019). The target variable shows the diagnosis of the patient having heart disease or not. This reduces this problem to a binary classification task, which fits KNN well (Mohan, Thirumalai and Srivastava, 2019). The first step in modelling before the data is processed is to clean it by dropping rows where a target is missing and transforming the target into a binary (0/1) label. The dataset is further divided into 80% training (237 samples) and 20% testing (60 samples) with the help of stratified sampling to maintain the class proportions. This will keep evaluation outcomes free of aspects of imbalanced distributions (Uddin et al., 2019).

Feature shape: (303, 13)

Target shape: (303, 1)

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal
0	63	1	1	145	233	1	2	150	0	2.3	3	0.0	6.0
1	67	1	4	160	286	0	2	108	1	1.5	2	3.0	3.0
2	67	1	4	120	229	0	2	129	1	2.6	2	2.0	7.0
3	37	1	3	130	250	0	0	187	0	3.5	3	0.0	3.0
4	41	0	2	130	204	0	2	172	0	1.4	1	0.0	3.0

3. Loading and Preparing the Data

Python

```
from ucimlrepo import fetch_ucirepo
```

```
# load dataset
```

```
heart_disease = fetch_ucirepo(id=45)
```

```
# features and target
```

```
X_raw = heart_disease.data.features.copy()
```

```
y_raw = heart_disease.data.targets.copy()
```

```
print("Feature shape:", X_raw.shape)
```

```
print("Target shape:", y_raw.shape)
```

```
X_raw.head()

# clean data
df = pd.concat([X_raw, y_raw], axis=1)

# drop missing rows
df = df.dropna()

# target column name
target_col = df.columns[-1]
print("Target column:", target_col)

# binary target
y = (df[target_col] > 0).astype(int)

# final features
X = df.drop(columns=[target_col])

print("Clean features shape:", X.shape)
print("Clean target shape:", y.shape)

# split data
X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.2,
    random_state=42,
```

```

        stratify=y
    )

print("Train samples:", X_train.shape[0])
print("Test samples:", X_test.shape[0])

```

4. Why Scaling Matters for KNN

Since KNN is sensitive to distance measure (usually the Euclidean distance), the calculation may be controlled by more numerically large features (Prasatha et al., 2017; Ali, Neagu and Trundle, 2019). As an example, a binary indicator variable compared to cholesterol (which can vary between 120-500) would distort neighbourhood structure unless it is normalised. As Zhang (2021) emphasises, appropriate feature scaling is essential for addressing challenges in KNN classification. Thus, we use StandardScaler in a scikit-learn Pipeline (Pedregosa et al., 2011), which means that the cleaning of the data, scaling, and classification run at the same time in a workflow that can be repeated.

Python

```

from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.neighbors import KNeighborsClassifier

```

Baseline KNN Model ($k = 5$)

The default k which we choose as the first model is $k=5$, following common practice in the literature (Guo et al., 2003):

Python

```

# baseline model
baseline_knn = Pipeline([
    ("scaler", StandardScaler()),
    ("knn", KNeighborsClassifier(n_neighbors=5))
])

base_train_acc, base_test_acc = evaluate_model(

```

```

baseline_knn,
X_train,
y_train,
X_test,
y_test,
label="KNN k = 5"
)

print("\nClassification report:")
print(classification_report(y_test, baseline_knn.predict(X_test)))

```

```

print("Confusion matrix:")
print(confusion_matrix(y_test, baseline_knn.predict(X_test)))

```

The evaluation function prints both training and test accuracy:

Python

```

def evaluate_model(clf, X_train, y_train, X_test, y_test, label=None):
    clf.fit(X_train, y_train)
    y_train_pred = clf.predict(X_train)
    y_test_pred = clf.predict(X_test)

    train_acc = accuracy_score(y_train, y_train_pred)
    test_acc = accuracy_score(y_test, y_test_pred)

    if label is not None:
        print(f'{label:25s} | Train: {train_acc:.4f} | Test: {test_acc:.4f}')

    return train_acc, test_acc

```

Running this model yields:

The classification report shows balanced performance across both classes:

text

KNN k = 5 | Train: 0.8692 | Test: 0.8833

Classification report:

	precision	recall	f1-score	support
0	0.86	0.94	0.90	32
1	0.92	0.82	0.87	28
accuracy				0.88 60
macro avg	0.89	0.88	0.88	60
weighted avg	0.89	0.88	0.88	60

Confusion matrix:

[[30 2]

[5 23]]

Such a solid foundation proves that the dataset is favourable to KNN and that scaling works, consistent with findings in similar healthcare classification studies (Mohan, Thirumalai and Srivastava, 2019; Anjaneyulu et al., 2023).

5. Exploring Different Values of k

The hyperparameter of KNN that matters the most is k, which regulates the smoothness of the decision boundaries (Zhang et al., 2017). As noted by Cover and Hart (1967) in their foundational work:

- When k is Low (i.e. $k = 1$), then the variance is high, and the results are overfit.
- Large k (e.g. $k = 25$) results in overfitting and large bias.

We systematically test odd values of k from 1 to 25:

Python

```
# hyperparameter loop
```

```
k_values = list(range(1, 26, 2))
```

The results show a clear pattern:

K values summary

text

k = 1	Train: 1.0000 Test: 0.7833
k = 3	Train: 0.8903 Test: 0.9000
k = 5	Train: 0.8692 Test: 0.8833
k = 7	Train: 0.8565 Test: 0.8333
k = 9	Train: 0.8397 Test: 0.8667
k = 11	Train: 0.8523 Test: 0.8833
k = 13	Train: 0.8523 Test: 0.8667
k = 15	Train: 0.8523 Test: 0.8500
k = 17	Train: 0.8481 Test: 0.8500
k = 19	Train: 0.8608 Test: 0.8667
k = 21	Train: 0.8565 Test: 0.8500
k = 23	Train: 0.8565 Test: 0.8667
k = 25	Train: 0.8439 Test: 0.8667

The optimal generalisation will be at k=3 and the test accuracy will be 0.90.

The plot illustrates a standard bias-variance curve: training accuracy declines slowly with the increase of k but test accuracy reaches its maximum at k=3 and does not change any further. This pattern aligns with the theoretical framework described by Hastie, Tibshirani and Friedman (2009).

6. Final Model Evaluation

Using the optimal value:

Python

```
# final model
```

```
best_knn = Pipeline([
    ("scaler", StandardScaler()),
    ("knn", KNeighborsClassifier(n_neighbors=best_k))
])
```

```
best_knn.fit(X_train, y_train)
y_pred_best = best_knn.predict(X_test)
```

Final performance:

text

Final model k 3

Classification report:

	precision	recall	f1-score	support
0	0.91	0.91	0.91	32
1	0.89	0.89	0.89	28
accuracy			0.90	60
macro avg	0.90	0.90	0.90	60
weighted avg	0.90	0.90	0.90	60

Confusion matrix:

```
[[29 3]
```

```
[ 3 25]]
```

This last model exhibits stability in generalisation, equal classification and an intelligible pattern of performance—perfect characteristics in education. The results are comparable to those reported by Mohan, Thirumalai and Srivastava (2019) and Anjaneyulu et al. (2023) in their heart disease prediction studies.

7. The Interpretation of KNN in Terms of Bias-Variance Trade-off

The primary way that the use of KNN benefits pedagogical practice is that its behaviour directly reflects the bias-variance trade-off, which is a fundamental principle in machine learning (Hastie, Tibshirani and Friedman, 2009). In the case when k is too small (particularly when $k = 1$), the classifier tries to memorise the training data perfectly. This forms very localised decision boundaries which adjust to noise and outliers, which provide high training accuracy but poor generalisation (Zhang, 2021). This is in agreement with our results: when $k = 1$, the training accuracy is 1.00, and the test accuracy falls to 0.7833, which is extremely overfit.

On the other hand, a large k causes the decision boundary to be over-smooth, and separate regions of classes are combined, and smaller-scale structures are disregarded (Cover and Hart, 1967). This heightens prejudice, resulting in underfitting. Even in our experiment, such values as $k = 21$ or $k = 25$ work fairly well, but no longer achieve the highest accuracy of $k = 3$. The model starts factoring in hundreds of points as part of the majority trend, which minimises its capability to detect subtle differences in the patient attributes. The highest value at $k = 3$, where the accuracy of tests is the highest (0.90), proves the sweet spot—the place at which noise is removed without any crucial structure in the data.

The graphical representation of this is the inverted-U shape of test accuracy and a line of steadily diminishing training accuracy, which are seen in the accuracy curve. This behaviour is also typical of most non-parametric models (Hastie, Tibshirani and Friedman, 2009), and therefore explains why KNN is a good model to study the sensitivity of hyperparameters and tuning.

8. Why the Heart Disease Dataset is a Good Case Study

The choice of dataset is the important point of a tutorial, which is highlighted in the assignment brief. The UCI Heart Disease data pack (Dua and Graff, 2019) has a number of advantages, both technical and pedagogical, in prospect.

Originally, it has clinically meaningful, heterogeneous data (continuous measures, including maximum heart rate and cholesterol, and more categorical measures, such as chest pain type and exercise-induced angina) that can be found. The KNN is not very efficient in high-dimensional spaces (Zhang, 2021), but since the number of features in this dataset is medium, it is the best choice of data to illustrate the KNN without excessive computation. Ali, Neagu and Trundle (2019) have demonstrated that KNN performs well on such heterogeneous datasets when proper preprocessing is applied.

Second, the size of the dataset is small enough to enable quick experimentation, and this implies that the students will be able to see the influence of varying model parameters within a very short time. Such responsiveness supports the learning aspect by ensuring that experimentation

goes easily. Moreover, since the dataset has been used extensively in the academic community (Mohan, Thirumalai and Srivastava, 2019; Anjaneyulu et al., 2023), the readers can compare the findings with the existing research, but still can come to original conclusions by exploring k-values and scaling.

Third, the medical nature of the data implies an ethical aspect. In healthcare applications, predictive performance is not only adequate but also interpretability, data quality, and possible bias should be taken into account (Uddin et al., 2019). Through a simple and understandable model such as KNN, students will be able to consider the bigger picture in terms of what mis-categorisation means, and how various decision-making about modelling can impact sensitive areas.

9. Pictorial Presentation of the Results

Another important feature of teaching machine learning is not just demonstrating the numbers but also making clear how numbers display model behaviour (Hastie, Tibshirani and Friedman, 2009). This tutorial is very informative, especially in the plot of accuracy versus k.

This figure reveals three clear regions:

Low-k Region (1–5):

There is overfitting as the curve is not stable. The accuracy peaks at $k = 3$, whereas $k = 1$ is too sensitive to noise. The abrupt decrease at $k = 1$ depicts the effect of memorisation that causes poor validation data even when training accuracy is perfect. This phenomenon is well-documented in the literature on instance-based learning (Guo et al., 2003; Zhang et al., 2017).

Middle-k Region (7–15):

At this medium level, the accuracy levels off at 0.83 to 0.88. The model is not too complicated or complex. It is anticipated to have small oscillations because of the structure of the dataset (Ali, Neagu and Trundle, 2019).

High-k Region (17–25):

The neighbourhoods become excessive in size as k increases. The decision boundaries become smoother, and the classifier is unable to draw fine lines between subtle variations. Reasonable accuracy is obtained, but no longer optimal as had been previously at $k = 3$. In terms of visual teaching, this shows that simple plots can be used to explain to learners otherwise abstract theoretical concepts.

10. Ethical and Accessibility Issues

The brief focuses on AI-ethical and accessibility. Even though KNN is not an opaque model per se, it is nonetheless not unjustified to raise ethical considerations, particularly when predicting health-related outcomes (Uddin et al., 2019).

Ethical aspects include:

Risk of misclassification:

False negative (negative prediction when the disease is present) can cause the failure to receive medical care in time. A false positive would result in unwarranted stress or treatments. Thus, it should be evaluated based on recall and precision, and not accuracy (Mohan, Thirumalai and Srivastava, 2019).

Data representativeness:

It is common to find that clinical datasets are a representation of a particular population (Dua and Graff, 2019). In case the training data is not diverse, the predictions of the model might fail to extend to the wider groups of patients and give unfair results.

Transparency:

The local behavior of KNN is not very difficult to understand: a decision is made according to the closeness to similar cases. This reinstates its educational applicability in sensitive areas (Guo et al., 2003).

The considerations with regard to accessibility are:

- Plots in high contrast color schemes.
- Offering titles and axes of descriptive figures.
- Creating the tutorial in a plain, readable form that will accommodate the screen-readers.
- Making the code runnable on non-specialized hardware and non-standard libraries.
- Correlating the Findings with the Concepts of Machine Learning.

11. Relating the Findings to Machine Learning Concepts:

The experiment with KNN on this dataset confirms a number of more general machine-learning concepts:

Importance of preprocessing:

The distance-based algorithms are highly affected by feature scaling (Prasatha et al., 2017). In the absence of a scaling, cholesterol among other features would dominate proximity calculations and give distorted boundaries (Ali, Neagu and Trundle, 2019).

Model complexity vs generalization:

KNN lacks parameters such as weight matrices used in neural networks yet the k hyperparameter is an effective measure of control of complexity (Zhang et al., 2017). The findings demonstrate that the simplest, but the most suitability calibrated model is the best performing (Hastie, Tibshirani and Friedman, 2009).

Reproducibility:

The organisation of the notebook in terms of data-loading, evaluation, and plotting steps makes the tutorial compliant with best practises highlighted in the assignment brief. The use of scikit-learn's Pipeline functionality (Pedregosa et al., 2011) ensures consistent preprocessing across training and testing phases.

12. Guidelines on the Practical Use of KNN

The findings of this tutorial enable us to draw a number of applicable principles that readers can use in their respective machine-learning processes. Despite the simplicity of KNN, its application largely relies on the nature of data and pre-processing decisions (Zhang, 2021). Thus, its successful application must be paid to several of its basic considerations.

1. Always scale your features.

Distance-based approaches are very much susceptible to scale differences (Prasatha et al., 2017). The distance measure will be dominated by a feature that has numerically large range even though it is not the most informative. This is normally addressed by StandardScaler or MinMaxScaler (Pedregosa et al., 2011). Scaling is required to ensure that KNN does not give incorrect neighbourhood structures and also haphazard classification boundaries.

2. Tune k systematically rather than relying on defaults.

Most introductory algorithms typically work with $k = 5$, though our results indicate that $k = 3$ worked much better on the heart disease data, with 90 percent test accuracy. As Zhang et al. (2017) demonstrate, learning the optimal k value is crucial for KNN classification performance. KNN is

computationally cheap to perform hyperparameter searches over a space of k values and, therefore, systematic exploration is not only possible but also necessary.

3. Use odd values of k for binary classification.

Odd values will ensure that there is no tie among the neighbours (Cover and Hart, 1967). This does not create ambiguity and does not make decisions more complex but adds no additional computational cost.

4. Consider class balance and evaluation metrics.

Being accurate does not necessarily indicate a good model performance. KNN projections are prone to give bias towards the majority class especially when the k is high (Uddin et al., 2019). Precision, recall, F1-score, and confusion matrices are additional metrics that are more insightful. In the case of medical data, recall can be particularly valuable in order to minimise the false negatives (Mohan, Thirumalai and Srivastava, 2019).

5. Be aware of computational cost at prediction time.

KNN does not fit a compact representation as parametric models do (Guo et al., 2003). The whole training set is stored and referred to in prediction, which makes inference of large data slower. This did not form a restriction in our tutorial, but in large scale deployments, practitioners need to apply approximate nearest neighbour methods or dimensionality reduction.

6. Know when KNN is not suitable.

KNN works poorly in high-dimensional space, where there is the curse of dimensionality, and distances cease to be meaningful (Zhang, 2021). On datasets that have millions of samples, it becomes infeasible, too, unless specialised data structures (such as KD-trees, ball trees) are employed (Pedregosa et al., 2011). However, in the case of moderate size datasets, such as UCI Heart Disease, KNN can still be used effectively (Ali, Neagu and Trundle, 2019).

13. Conclusion

K-Nearest Neighbors is still a conceptually strong machine-learning learning tool (Cover and Hart, 1967; Guo et al., 2003). This tutorial has brought to light the importance of considering the preprocessing, hyperparameter tuning, and evaluation metrics in the process of achieving good performance of a model, as demonstrated through the use of KNN on the UCI Heart Disease data (Dua and Graff, 2019). We have seen that the best generalization results were obtained with a small neighborhood size ($k = 3$), which is a clear illustration of the interplay between bias and variance (Hastie, Tibshirani and Friedman, 2009).

In addition to the quantitative findings, this tutorial demonstrates a larger point that can be made in the assignment brief: successful machine learning is not only about how algorithms work, but an explanation of how they work. Analyzing performance curves, confusion matrices, and

underlying data structure enables learners to have a better basis for applying machine-learning models in a responsible and effective way in real-world scenarios (Uddin et al., 2019).

KNN is still a very useful model to teach or experiment with and use as a standard model in working processes due to its transparency, simplicity, and empirical effectiveness (Zhang et al., 2017). The techniques here can be readily applied to other datasets, feature engineering pipelines and cross-validation strategies, which provides the readers with a usable and flexible template to use in the future.

References

Ali, N., Neagu, D. and Trundle, P., 2019. Evaluation of k-nearest neighbour classifier performance for heterogeneous data sets. *SN Applied Sciences*, 1(12), pp.1-15.

Anjaneyulu, M., Degala, D.P., Devika, P. and Hema, V., 2023. Effective heart disease prediction using hybrid machine learning techniques. *ADVANCEMENTS IN AEROMECHANICAL MATERIALS FOR MANUFACTURING: ICAAMM-2021*, 2492(1), p.030070.

Cover, T. and Hart, P., 1967. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1), pp.21-27.

Dua, D. and Graff, C., 2019. *UCI Machine Learning Repository*. Irvine, CA: University of California, School of Information and Computer Science. Available at: <http://archive.ics.uci.edu/ml> [Accessed 2024].

Guo, G., Wang, H., Bell, D., Bi, Y. and Greer, K., 2003, November. KNN model-based approach in classification. In *OTM Confederated International Conferences "On the Move to Meaningful Internet Systems"* (pp. 986-996). Berlin, Heidelberg: Springer Berlin Heidelberg.

Hastie, T., Tibshirani, R. and Friedman, J., 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2nd ed. New York: Springer.

Mohan, S., Thirumalai, C. and Srivastava, G., 2019. Effective heart disease prediction using hybrid machine learning techniques. *IEEE Access*, 7, pp.81542-81554.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, E., 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, pp.2825-2830.

Prasatha, V.S., Alfeilate, H.A.A., Hassanate, A.B., Lasassmehe, O., Tarawnehf, A.S., Alhasanatg, M.B. and Salmane, H.S.E., 2017. Effects of distance measure choice on KNN classifier performance—a review. *arXiv preprint arXiv:1708.04321*, 56, p.24.

Uddin, S., Khan, A., Hossain, M.E. and Moni, M.A., 2019. Comparing different supervised machine learning algorithms for disease prediction. *BMC Medical Informatics and Decision Making*, 19(1), pp.1-16.

Zhang, S., 2021. Challenges in KNN classification. *IEEE Transactions on Knowledge and Data Engineering*, 34(10), pp.4663-4675.

Zhang, S., Li, X., Zong, M., Zhu, X. and Cheng, D., 2017. Learning k for KNN classification. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(3), pp.1-19.

Appendix:

Code

```
# -*- coding: utf-8 -*-
```

```
"""KNNtutorial
```

Automatically generated by Colab.

Original file is located at

https://colab.research.google.com/drive/1JAWkr2ZMsARwVWrP_ezgiqN8mTmdS3xJ

"""

install package

!pip install ucimlrepo -q

1. Setup & Imports

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import fetch_openml

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.pipeline import Pipeline

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

from ucimlrepo import fetch_ucirepo

load dataset

heart_disease = fetch_ucirepo(id=45)

```
# features and target

X_raw = heart_disease.data.features.copy()
y_raw = heart_disease.data.targets.copy()


print("Feature shape:", X_raw.shape)
print("Target shape:", y_raw.shape)
X_raw.head()


# inspect target
y_raw.head()


# clean data
df = pd.concat([X_raw, y_raw], axis=1)


# drop missing rows
df = df.dropna()


# target column name
target_col = df.columns[-1]
print("Target column:", target_col)


# binary target
y = (df[target_col] > 0).astype(int)


# final features
X = df.drop(columns=[target_col])
```

```

print("Clean features shape:", X.shape)
print("Clean target shape:", y.shape)

# split data
X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.2,
    random_state=42,
    stratify=y
)

print("Train samples:", X_train.shape[0])
print("Test samples:", X_test.shape[0])

# helper function
def evaluate_model(clf, X_train, y_train, X_test, y_test, label=None):
    clf.fit(X_train, y_train)
    y_train_pred = clf.predict(X_train)
    y_test_pred = clf.predict(X_test)

    train_acc = accuracy_score(y_train, y_train_pred)
    test_acc = accuracy_score(y_test, y_test_pred)

    if label is not None:
        print(f'{label:25s} | Train: {train_acc:.4f} | Test: {test_acc:.4f}')

```

```
return train_acc, test_acc
```

```
# baseline model
```

```
baseline_knn = Pipeline([  
    ("scaler", StandardScaler()),  
    ("knn", KNeighborsClassifier(n_neighbors=5))  
])
```

```
base_train_acc, base_test_acc = evaluate_model(  
    baseline_knn,  
    X_train,  
    y_train,  
    X_test,  
    y_test,  
    label="KNN k = 5"  
)
```

```
print("\nClassification report:")
```

```
print(classification_report(y_test, baseline_knn.predict(X_test)))
```

```
print("Confusion matrix:")
```

```
print(confusion_matrix(y_test, baseline_knn.predict(X_test)))
```

```
# hyperparameter loop
```

```
k_values = list(range(1, 26, 2))
```

```
train_scores = []
```

```

test_scores = []

print("K values summary\n")

for k in k_values:
    clf = Pipeline([
        ("scaler", StandardScaler()),
        ("knn", KNeighborsClassifier(n_neighbors=k))
    ])

    label = f"k = {k}"
    tr_acc, te_acc = evaluate_model(
        clf,
        X_train,
        y_train,
        X_test,
        y_test,
        label=label
    )

    train_scores.append(tr_acc)
    test_scores.append(te_acc)

# accuracy plot
plt.figure(figsize=(8, 5))
plt.plot(k_values, train_scores, marker="o", label="Train")
plt.plot(k_values, test_scores, marker="s", label="Test")

```

```
plt.xlabel("k neighbours")
plt.ylabel("Accuracy")
plt.title("KNN accuracy on heart disease data")
plt.grid(alpha=0.3)
plt.legend()
plt.show()

# best k
results = pd.DataFrame({
    "k": k_values,
    "train_accuracy": train_scores,
    "test_accuracy": test_scores
})

best_row = results.iloc[results["test_accuracy"].idxmax()]
best_k = int(best_row["k"])
best_acc = best_row["test_accuracy"]

print("Best k:", best_k)
print("Best test accuracy:", round(best_acc, 4))
results

# final model
best_knn = Pipeline([
    ("scaler", StandardScaler()),
    ("knn", KNeighborsClassifier(n_neighbors=best_k))
])
```

```
best_knn.fit(X_train, y_train)

y_pred_best = best_knn.predict(X_test)


print(f'Final model k {best_k}\n')
print("Classification report:")
print(classification_report(y_test, y_pred_best))


print("Confusion matrix:")
print(confusion_matrix(y_test, y_pred_best))


# Drop missing values as used in your modelling pipeline
df_clean = df.dropna()
y_clean = (df_clean[target_col] > 0).astype(int)


print("=== Cleaned Dataset Shape ===")
print(df_clean.shape)
print()


print("=== Cleaned Target Distribution ===")
display(y_clean.value_counts(normalize=True).rename("proportion"))


plt.figure(figsize=(8, 5))
plt.plot(k_values, train_scores, marker="o", label="Train")
plt.plot(k_values, test_scores, marker="s", label="Test")
plt.xlabel("k neighbours")
plt.ylabel("Accuracy")
```

```

plt.title("KNN accuracy on heart disease data")
plt.grid(alpha=0.3)
plt.legend()
plt.show()

plt.figure(figsize=(8,5))
plt.plot(k_values, train_scores, marker="o", label="Train Accuracy")
plt.plot(k_values, test_scores, marker="s", label="Test Accuracy")

plt.xlabel("k (number of neighbours)")
plt.ylabel("Accuracy")
plt.title("Bias–Variance Illustration using KNN")

# Annotate high variance region
plt.annotate("Overfitting (High Variance)",
            xy=(1, train_scores[0]),
            xytext=(3, 0.98),
            arrowprops=dict(arrowstyle="->"))

# Annotate optimal region
best_k = 3
plt.annotate("Optimal k",
            xy=(best_k, test_scores[1]),
            xytext=(best_k+4, test_scores[1]+0.05),
            arrowprops=dict(arrowstyle="->"))

# Annotate high bias region

```

```
plt.annotate("Underfitting (High Bias)",  
            xy=(23, test_scores[-2]),  
            xytext=(15, 0.82),  
            arrowprops=dict(arrowstyle="->"))
```

```
plt.grid(alpha=0.3)
```

```
plt.legend()
```

```
plt.show()
```