```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix

from sklearn.preprocessing import StandardScaler

df = pd.read_csv('heart.csv')

df.head()
```

```
   age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak
slope  \
0   52    1   0       125   212    0        1      168      0      1.0
2
1   53    1   0       140   203    1        0      155      1      3.1
0
2   70    1   0       145   174    0        1      125      1      2.6
0
3   61    1   0       148   203    0        1      161      0      0.0
2
4   62    0   0       138   294    1        1      106      0      1.9
1

   ca  thal  target
0   2     3       0
1   0     3       0
2   0     3       0
3   1     3       0
4   3     2       0
```

```python
df.isnull().sum()
```

```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
```

```
thal          0
target        0
dtype: int64
```

```python
df.replace('?', np.nan, inplace=True)
```

```python
df.isnull().sum()
```

```
age           0
sex           0
cp            0
trestbps      0
chol          0
fbs           0
restecg       0
thalach       0
exang         0
oldpeak       0
slope         0
ca            0
thal          0
target        0
dtype: int64
```

```python
for col in df.columns:
    df[col] = pd.to_numeric(df[col], errors='coerce')
```

```python
df_cleaned = df.dropna()
```

```python
df_cleaned.shape
```

```
(1025, 14)
```

```python
numeric_cols = df_cleaned.select_dtypes(include=[np.number]).columns
df_cleaned = df_cleaned[(df_cleaned[numeric_cols] >= 0).all(axis=1)]
```

```python
df_cleaned.shape
```

```
(1025, 14)
```

```python
unique_values = {col: df_cleaned[col].unique() for col in
df_cleaned.columns}
print(unique_values)
```

```
{'age': array([52, 53, 70, 61, 62, 58, 55, 46, 54, 71, 43, 34, 51, 50,
60, 67, 45,
       63, 42, 44, 56, 57, 59, 64, 65, 41, 66, 38, 49, 48, 29, 37, 47,
68,
       76, 40, 39, 77, 69, 35, 74], dtype=int64), 'sex': array([1, 0],
dtype=int64), 'cp': array([0, 1, 2, 3], dtype=int64), 'trestbps':
array([125, 140, 145, 148, 138, 100, 114, 160, 120, 122, 112, 132,
118,
```

```
       128, 124, 106, 104, 135, 130, 136, 180, 129, 150, 178, 146,
117,
       152, 154, 170, 134, 174, 144, 108, 123, 110, 142, 126, 192,
115,
        94, 200, 165, 102, 105, 155, 172, 164, 156, 101],
dtype=int64), 'chol': array([212, 203, 174, 294, 248, 318, 289, 249,
286, 149, 341, 210, 298,
       204, 308, 266, 244, 211, 185, 223, 208, 252, 209, 307, 233,
319,
       256, 327, 169, 131, 269, 196, 231, 213, 271, 263, 229, 360,
258,
       330, 342, 226, 228, 278, 230, 283, 241, 175, 188, 217, 193,
245,
       232, 299, 288, 197, 315, 215, 164, 326, 207, 177, 257, 255,
187,
       201, 220, 268, 267, 236, 303, 282, 126, 309, 186, 275, 281,
206,
       335, 218, 254, 295, 417, 260, 240, 302, 192, 225, 325, 235,
274,
       234, 182, 167, 172, 321, 300, 199, 564, 157, 304, 222, 184,
354,
       160, 247, 239, 246, 409, 293, 180, 250, 221, 200, 227, 243,
311,
       261, 242, 205, 306, 219, 353, 198, 394, 183, 237, 224, 265,
313,
       340, 259, 270, 216, 264, 276, 322, 214, 273, 253, 176, 284,
305,
       168, 407, 290, 277, 262, 195, 166, 178, 141], dtype=int64),
'fbs': array([0, 1], dtype=int64), 'restecg': array([1, 0, 2],
dtype=int64), 'thalach': array([168, 155, 125, 161, 106, 122, 140,
145, 144, 116, 136, 192, 156,
       142, 109, 162, 165, 148, 172, 173, 146, 179, 152, 117, 115,
112,
       163, 147, 182, 105, 150, 151, 169, 166, 178, 132, 160, 123,
139,
       111, 180, 164, 202, 157, 159, 170, 138, 175, 158, 126, 143,
141,
       167,  95, 190, 118, 103, 181, 108, 177, 134, 120, 171, 149,
154,
       153,  88, 174, 114, 195, 133,  96, 124, 131, 185, 194, 128,
127,
       186, 184, 188, 130,  71, 137,  99, 121, 187,  97,  90, 129,
113],
      dtype=int64), 'exang': array([0, 1], dtype=int64), 'oldpeak':
array([1. , 3.1, 2.6, 0. , 1.9, 4.4, 0.8, 3.2, 1.6, 3. , 0.7, 4.2,
1.5,
       2.2, 1.1, 0.3, 0.4, 0.6, 3.4, 2.8, 1.2, 2.9, 3.6, 1.4, 0.2,
2. ,
       5.6, 0.9, 1.8, 6.2, 4. , 2.5, 0.5, 0.1, 2.1, 2.4, 3.8, 2.3,
```

```
1.3,
       3.5]), 'slope': array([2, 0, 1], dtype=int64), 'ca': array([2,
0, 1, 3, 4], dtype=int64), 'thal': array([3, 2, 1, 0], dtype=int64),
'target': array([0, 1], dtype=int64)}
```

Box Plot

A way to visualize the distribution of one or more groups of numeric data - highlighting its central tendency, spread, potential outliers - all in single graphic

1. The Box   a. Lower Edge(Q1) - 25th percentile - first quartile   b. Upper Edge(Q3) - 75th percentile third quartile   c. Height of Box (IQR) - Interquartile range - IQR = Q3-Q1   Measures spread of middle 50% of data

2. Median Line   Line inside box at 50th percentile (Q2)   Central tendency without being skewed by extreme values

3. Whiskers   Extend from each box edge to the most extreme data point within a certain range   whiskers go out to: [Q1 - 1.5xIQR, Q3 + 1.5xIQR]   Any point beyond these whiskers is potential outlier

4. Outliers   circles or dots

```
def remove_outliers_iqr(df, columns):
    for col in columns:
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1
        lower = Q1 - 1.5 * IQR
        upper = Q3 + 1.5 * IQR
        df = df[(df[col] >= lower) & (df[col] <= upper)]
    return df

df_no_outliers = remove_outliers_iqr(df_cleaned, ['age', 'trestbps',
'chol', 'thalach', 'oldpeak'])

df_no_outliers.shape
```

```
(964, 14)
```

```
# One-hot encode 'cp', 'thal', and 'slope' as they are categorical
df_transformed = pd.get_dummies(df_no_outliers, columns=['cp', 'thal',
'slope'], drop_first=True)
```

One-Hot encoding

Turns signle categorical column(with k distinct values) into k binary(0/1 or True/False) columns - one per category This lets ML algorithms treat categories as separate, unordered features rather than as numeric codes

get_dummies() - creates dummy variables from categorical variables   columns: list of column names to encode   drop_first - default False   True - drops the first category and only creates k-1 dummies - to avoid the dummy-variable-trap(perfect multicolinearity) If you leave a column as text ("typical", "atypical", …) or integer codes (1,2,3,4), the algorithm will either fail or treat those codes as ordinal numbers—which they aren't - problem in knn Logistic Regression finds a weighted sum of inputs. If you leave categories as a single integer, you're forcing a linear ramp across categories

Scale numeric features -   k-Nearest Neighbors uses distance metrics—features on large scales will dominate   Logistic Regression and other gradient-based models converge faster when inputs have mean≈0 and variance≈1 StandardScaler - transformer that standardizes features by removing the mean and scaling to unit variance

KNN

KNN is a supervised machine learning algorithm used for both classification and regression, but it's more commonly used for classification

Choose a value for K → (e.g., K = 3 means we'll look at the 3 nearest neighbors) Calculate distances → Use a distance metric like Euclidean distance to measure how close data points are Find the K nearest neighbors → Based on the distance, pick the K closest training data points to the new input Vote (for classification) → Check the most frequent class among the K neighbors and assign that class to the input point   OR Take the average (for regression) → Predict the value based on the average of K neighbors' outputs

Logistic Regression

Logistic Regression is a supervised machine learning algorithm used for classification problems, especially binary classification (i.e., problems with two possible outcomes like Yes/No, 0/1, True/False) While linear regression predicts continuous values, logistic regression predicts probabilities. These probabilities are then converted into class labels (like 0 or 1)

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report

# Features and target
X = df_transformed.drop('target', axis=1)
y = df_transformed['target']

# Stratified split
X_train, X_test, y_train, y_test = train_test_split(
```

```python
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Logistic Regression
logreg = LogisticRegression(max_iter=2000)
logreg.fit(X_train, y_train)
y_pred_logreg = logreg.predict(X_test)
logreg_acc = accuracy_score(y_test, y_pred_logreg)

# k-NN
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)
knn_acc = accuracy_score(y_test, y_pred_knn)

# Results
print("Logistic Regression Accuracy:", logreg_acc)
print("k-NN Accuracy:", knn_acc)
print("\nClassification Report (Logistic Regression):\n",
classification_report(y_test, y_pred_logreg, zero_division=0))
print("\nClassification Report (k-NN):\n",
classification_report(y_test, y_pred_knn, zero_division=0))
```

```
Logistic Regression Accuracy: 0.8134715025906736
k-NN Accuracy: 0.7046632124352331

Classification Report (Logistic Regression):
               precision    recall  f1-score   support

           0       0.86      0.73      0.79        91
           1       0.78      0.89      0.83       102

    accuracy                           0.81       193
   macro avg       0.82      0.81      0.81       193
weighted avg       0.82      0.81      0.81       193


Classification Report (k-NN):
               precision    recall  f1-score   support

           0       0.68      0.71      0.70        91
           1       0.73      0.70      0.71       102

    accuracy                           0.70       193
   macro avg       0.70      0.71      0.70       193
weighted avg       0.71      0.70      0.70       193
```