

```
In [7]: ***
Breast Cancer Prediction with KNN
Author: [Your Name]
Description: Predict whether a breast tumor is malignant or benign using KNN
***
```

```
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, confusion_matrix, classification_report

# Load the breast cancer dataset
print("Loading the breast cancer dataset...")
cancer_data = load_breast_cancer()

# Check for missing values
print("Number of rows: ", len(cancer_data['data']))
print("Number of columns: ", len(cancer_data['feature_names']))
print("Number of samples: ", type(cancer_data['data']))
print("Number of features: ", len(cancer_data['feature_names']))
print("Target feature: ", cancer_data['target'].name)

# Convert to DataFrame for easier manipulation
df = pd.DataFrame(cancer_data['data'], columns=cancer_data['feature_names'])
df['target'] = cancer_data['target']

print("First few rows:")
print(df.head())
print("Count info:")
print(df.info())

print("Unique distribution:")
print(df['target'].value_counts())
print("Malignant (1): (df['target'] == 1).sum()")
print("Benign (0): (df['target'] == 0).sum()")

# Basic statistics
print("\n" * 4 + "***")
print("DATA STATISTICS")
print(df.describe())
print(df.describe())

# Check for missing values
print("Number of rows: ", len(df))
print("Number of columns: ", len(df.columns))
print("Missing values: ", df.isnull().sum())
if missing.sum() == 0:
    print("No missing values found!")
else:
    print(missing[missing > 0])

# Visualize feature distributions (select a few key features)
print("\n" * 4 + "***")
print("FEATURE DISTRIBUTIONS")
print(df.describe())

# Select a few representative features to visualize
key_features = ['mean radius', 'mean texture', 'mean perimeter', 'mean area']

fig, axes = plt.subplots(2, 2, figsize=(12, 10))
axes = axes.ravel()

for idx, feature in enumerate(key_features):
    axes[idx].hist(df[df['target'] == 0][feature], alpha=0.5, label='Benign', bins=30)
    axes[idx].hist(df[df['target'] == 1][feature], alpha=0.5, label='Malignant', bins=30)
    axes[idx].set_xlabel(feature)
    axes[idx].set_title(f'Distribution of {feature}')
    axes[idx].legend()

plt.tight_layout()
plt.savefig('feature_distributions.png', dpi=150, bbox_inches='tight')
print("Saved visualization to 'feature_distributions.png'")
plt.show()

# Separate features and target
X = df.drop(['target'], axis=1) # All columns except 'target'
y = df['target'] # Target column

print("Features shape: (X.shape[1])")
print("Target shape: (y.shape[1])")

# Split into training and testing sets
# random_state ensures reproducibility
# stratify ensures both sets have similar class distribution
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42,
    stratify=y
)

print("\n" * 4 + "***")
print("DATA SPLIT")
print("Training set size: (X_train.shape[0]) samples")
print("Test set size: (X_test.shape[0]) samples")
print("Training features: (X_train.shape[1])")
print("Test features: (X_test.shape[1])")

# Verify class distribution in both sets
print("Training set distribution:")
print(y_train.value_counts())
print(" Benign (0): ((y_train == 0).sum()) ((y_train == 0).mean()*100;.1f)%")
print(" Malignant (1): ((y_train == 1).sum()) ((y_train == 1).mean()*100;.1f)%")

print("Test set distribution:")
print(y_test.value_counts())
print(" Benign (0): ((y_test == 0).sum()) ((y_test == 0).mean()*100;.1f)%")
print(" Malignant (1): ((y_test == 1).sum()) ((y_test == 1).mean()*100;.1f)%")

# Create KNN classifier
# n_neighbors=5 means the model will look at the 5 nearest neighbors to make a prediction
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)

print("KNN classifier trained successfully!")
print("Number of neighbors (k): (knn.n_neighbors)")

# Make predictions
y_train_pred = knn.predict(X_train)
y_test_pred = knn.predict(X_test)

print("Training predictions: (len(y_train_pred))")
print("Test predictions: (len(y_test_pred))")

# Compare predictions with actual values
comparison = pd.DataFrame({
    'Actual': y_test.values,
    'Predicted': y_test_pred
})

print("\nFirst 10 prediction comparisons:")
print(comparison.head(10))

# Calculate metrics
train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)
test_precision = precision_score(y_test, y_test_pred)
test_recall = recall_score(y_test, y_test_pred)
test_f1score = f1_score(y_test, y_test_pred)

print("\n--- Model Performance ---")
print("Training Accuracy: (train_accuracy:.4f) ((train_accuracy*100;.2f)%)")
print("Test Accuracy: (test_accuracy:.4f) ((test_accuracy*100;.2f)%)")
print("Test Precision: (test_precision:.4f) ")
print("Test Recall: (test_recall:.4f) ")

print("\n--- Confusion Matrix ---")
print(confusion_matrix(y_test, y_test_pred))

print("\n--- Classification Report ---")
print(classification_report(y_test, y_test_pred, target_names=cancer_data['target'].name))

# Experiment with different K values
print("\n" * 4 + "***")
print("EXPERIMENTING WITH DIFFERENT K VALUES")
print("\n" * 4 + "***")

k_values = [1, 3, 5, 7, 9, 11]
results = []

for k in k_values:
    # Create a train model
    knn_temp = KNeighborsClassifier(n_neighbors=k)
    knn_temp.fit(X_train, y_train)

    # Make predictions
    acc = accuracy_score(y_test, y_pred_temp)
    prec = precision_score(y_test, y_pred_temp)
    rec = recall_score(y_test, y_pred_temp)

    results.append({
        'K': k,
        'Accuracy': acc,
        'Precision': prec,
        'Recall': rec
    })

# Find best K
results_df = pd.DataFrame(results)
best_k = results_df.loc[results_df['Accuracy'].idxmax(), 'K']
print(f"\nBest K value: (best_k) (Accuracy: (results_df['Accuracy'].max()):.4f)*")

# Visualize results
plt.figure(figsize=(10, 6))
plt.plot(k_values, results_df['Accuracy'], marker='o', label='Accuracy')
plt.plot(k_values, results_df['Precision'], marker='s', label='Precision')
plt.plot(k_values, results_df['Recall'], marker='x', label='Recall')
plt.xlabel('K (Number of Neighbors)')
plt.ylabel('Score')
plt.legend()
plt.grid(True, alpha=0.3)
plt.savefig('knn_k_comparison.png', dpi=150, bbox_inches='tight')
print("Saved visualization to 'knn_k_comparison.png'")

print("Loading breast cancer dataset...")

Dataset type: <class 'sklearn.utils._bunch.Bunch'>
Number of samples: 569
Number of features: 30
Target classes: ['malignant' 'benign']

First few rows:
 mean radius  mean texture  mean perimeter  mean area  mean smoothness \
0 10.43  10.38  122.80  1001.0  0.11840
1 20.57  17.77  132.90  1326.0  0.08474
2 19.69  21.25  130.00  1203.0  0.10960
3 11.42  20.38  135.10  1386.1  0.14250
4 20.99  14.34  135.10  1203.0  0.10930

mean concave points  mean concave concavity  mean concave symmetry \
0 0.0771  0.0801  0.0711  0.0711
1 0.0567  ... 0.0899  0.07017  0.14250
2 0.0599  ... 0.0744  0.0790  0.2069
3 0.0890  ... 0.0744  0.0790  0.2439
4 0.0983  ... 0.0744  0.0790  0.2575

mean fractal dimension .. worst texture  worst perimeter  worst area \
0 0.0771  17.33  184.60  2019.0
1 0.0567  ... 23.41  158.80  1956.0
2 0.0599  ... 25.53  152.50  1709.0
3 0.0890  ... 24.50  98.87  1557.0
4 0.0983  ... 16.67  152.20  1575.0

worst smoothness  worst compactness  worst concavity  worst concave points \
0 0.1622  0.0655  0.2416  0.2654
1 0.2750  0.0892  0  0.2439
2 0.1444  0.0645  0.1444  0.2439
3 0.2098  0.0663  0.6969  0.2575
4 0.1374  0.0250  0.4080  0.1625

worst symmetry  worst fractal dimension target
0 0.4601  0.11890  0
1 0.2750  0.0892  0
2 0.1444  0.0645  0
3 0.6638  0.17300  0
4 0.2364  0.07678  0

[5 rows x 31 columns]
```

```
Dataset info:
  -> "pandas.DataFrame"
  RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 # Column          Non-Null Count  Dtype  
--- 
0   mean radius    569 non-null   float64
1   mean texture   569 non-null   float64
2   mean perimeter 569 non-null   float64
3   mean area      569 non-null   float64
4   mean smoothness 569 non-null   float64
5   mean compactness 569 non-null   float64
6   mean concavity  569 non-null   float64
7   mean concave points 569 non-null   float64
8   mean fractal dimension 569 non-null   float64
9   radius error   569 non-null   float64
10  texture error  569 non-null   float64
11  perimeter error 569 non-null   float64
12  area error     569 non-null   float64
13  smoothness error 569 non-null   float64
14  compactness error 569 non-null   float64
15  concavity error 569 non-null   float64
16  concave points error 569 non-null   float64
17  concave concavity error 569 non-null   float64
18  concave symmetry error 569 non-null   float64
19  fractal dimension error 569 non-null   float64
20  worst radius    569 non-null   float64
21  worst texture   569 non-null   float64
22  worst perimeter 569 non-null   float64
23  worst area      569 non-null   float64
24  worst smoothness 569 non-null   float64
25  worst compactness 569 non-null   float64
26  worst concavity  569 non-null   float64
27  worst concave points 569 non-null   float64
28  worst concave concavity 569 non-null   float64
29  worst fractal dimension 569 non-null   float64
30  target         569 non-null   int64
dtypes: float64(30), int64(1)
memory usage: 137.9 KB
```

```
None
```

```
Target distribution:
```

```
target: 35
1 2
Name: count, dtype: int64
```

```
Malignant (1): 357
```

```
Benign (0): 212
```

```
-----
```

```
BASIC STATISTICS
```

```
mean radius  mean texture  mean perimeter  mean area \
count 569.000000 569.000000 569.000000 569.000000
mean 10.43 10.38 122.80 1001.0
std 3.240249 4.302036 24.298891 351.914129
min 6.891000 9.710000 43.790000 143.500000
25% 11.700000 16.170000 75.170000 420.300000
50% 15.900000 19.700000 94.900000 541.000000
75% 15.780000 21.800000 104.100000 782.700000
max 28.110000 39.280000 188.500000 2501.000000
```

```
mean smoothness  mean compactness  mean concavity  mean concave symmetry \
count 569.000000 569.000000 569.000000 569.000000
mean 0.1622 0.0655 0.2416 0.2654
std 0.2750 0.0892 0  0.2439
min 0.1444 0.0645 0.1444 0.2439
25% 0.2098 0.0663 0.6969 0.2575
50% 0.1374 0.0250 0.4080 0.1625
75% 0.0983 0.0744 0.0790 0.14250
max 0.4601 0.11890 0  0.2654
```

```
mean concave points  mean concave concavity  mean concave symmetry \
count 569.000000 569.000000 569.000000
mean 0.0771 0.0801 0.0711
std 0.0567 0.0899 0.07017
min 0.0599 0.0744 0.0790
25% 0.0890 0.0744 0.0790
50% 0.0983 0.0744 0.0790
75% 0.163400 0.045400 0.426800
max 0.4601 0.11890 0.2654
```

```
mean fractal dimension  target
count 569.000000 569.000000
mean 0.1622 0.11890
std 0.2750 0.0892
min 0.1444 0
25% 0.2098 0
50% 0.1374 0
75% 0.0983 0
max 0.4601 0.2654
```

```
-----
```

```
Model Performance
```

```
-----
```

```
Training Accuracy: 0.9473 (94.73%)
```

```
Test Accuracy: 0.9123 (91.23%)
```

```
Test Precision: 0.9429
```

```
Test Recall: 0.9167
```

```
-----
```

```
Confusion Matrix
```

```
-----
```

```
Actual Benign  Predicted Benign
```

```
Actual Benign  Predicted Malignant
```

```
-----
```

```
Actual Malignant  Predicted Benign
```

```
Actual Malignant  Predicted Malignant
```

```
-----
```

```
Classification Report
```

```
-----
```

```
precision  recall  f1-score  support
```

```
-----
```

```
Malignant  0.86  0.88  0.87  42
  Benign    0.94  0.92  0.93  214
```

```
accuracy  0.91  0.91  0.91  256
macro avg  0.90  0.91  0.91  114
weighted avg  0.91  0.91  0.91  114
```

```
-----
```

```
Experimenting with different K values
```

```
-----
```

```
K=1: accuracy=0.9211, Precision=0.9555, Recall=0.9167
```

```
K=3: accuracy=0.9298, Precision=0.9444, Recall=0.9167
```

```
K=5: accuracy=0.9123, Precision=0.9429, Recall=0.9167
```

```
K=7: accuracy=0.9304, Precision=0.9441, Recall=0.9167
```

```
K=9: accuracy=0.9386, Precision=0.9452, Recall=0.9183
```

```
K=11: accuracy=0.9386, Precision=0.9452, Recall=0.9183
```

```
Best K value: 9 (Accuracy: 0.9386)
```

```
Saved visualization to 'knn_k_comparison.png'
```

```
-----
```

```
-----
```

```
-----
```

```
-----
```

```
-----
```

```
-----
```

```
-----
```

```
-----
```

```
-----
```

```
-----
```

```
-----
```

```
-----
```

```
-----
```

```
-----
```

```
-----
```

```
-----
```

```
-----
```

```
-----
```

```
-----
```

```
-----
```

```
-----
```

```
-----
```

```
-----
```

```
-----
```

```
-----
```

```
-----
```

```
-----
```

```
-----
```

```
-----
```

```
-----
```

```
-----
```

```
-----
```

```
-----
```

```
-----
```

```
-----
```

```
-----
```

```
-----
```

```
-----
```

```
-----
```

```
-----
```

```
-----
```

