

```
***
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 matplotlib.pyplot as plt

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 breast cancer dataset...")
cancer_data = load_breast_cancer()

# The dataset is a Bunch object with 'data', 'target', 'feature_names', etc.
print(f"Dataset type: {type(cancer_data)}")
print(f"Number of samples: {len(cancer_data.data)}")
print(f"Number of features: {len(cancer_data.feature_names)}")
print(f"Target classes: {cancer_data.target_names}")

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

print(f"First few rows:")
print(df.head())

print(f"Dataset info:")
print(df.info())

print(f"Target distribution:")
print(df['target'].value_counts())
print(f"Benign: {df['target'] == 0}.sum()")
print(f"Malignant: {df['target'] == 1}.sum()")

# Basic statistics
print(f"n * *")
print("BASIC STATISTICS")
print(df.describe())

# Check for missing values
print(f"n * *")
print("MISSING VALUES")
print(df.isnull().sum())
if missing_sum == 0:
    print("✓ No missing values found!")
else:
    print(missing_sum)

# Visualize feature distributions (select a few key features)
print(f"n * *")
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_ylabel('Frequency')
    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(f"Features shape: {X.shape}")
print(f"Target shape: {y.shape}")

# Split into training and testing sets
# random_state ensures reproducibility
# stratify=y 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(f"n * *")
print("DATA SPLIT")
print(f"n *")
print(f"Training set size: {X_train.shape[0]} samples")
print(f"Test set size: {X_test.shape[0]} samples")
print(f"Training features: {X_train.shape[1]}")
print(f"Test features: {X_test.shape[1]}")

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

print(f"Test set target distribution:")
print(y_test.value_counts())
print(f"Benign: {(y_test == 0).sum()} ({(y_test == 0).mean()*100:.1f}%)")
print(f"Malignant: {(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(f"Number of neighbors (k): {knn.n_neighbors}")

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

print(f"Training predictions: {len(y_train_pred)}")
print(f"Test predictions: {len(y_test_pred)}")

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

print(f"First 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_confusion = confusion_matrix(y_test, y_test_pred)

print(f"Model Performance ==")
print(f"Training Accuracy: {train_accuracy:.4f} (Training Accuracy: 100.2f) %")
print(f"Test Accuracy: {test_accuracy:.4f} (Test Accuracy: 100.2f) %")
print(f"Test Precision: {test_precision:.4f}")
print(f"Test Recall: {test_recall:.4f}")

print(f"Confusion Matrix ==")
print("Predicted")
print("Actual Benign (test_confusion[0,0]:4d) (test_confusion[0,1]:4d) %")
print("Actual Malignant (test_confusion[1,0]:4d) (test_confusion[1,1]:4d) %")

print(f"Classification Report ==")
print(classification_report(y_test, y_test_pred, target_names=cancer_data.target_names))

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

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

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

    # Make predictions
    y_pred_temp = knn_temp.predict(X_test)

    # Calculate accuracy
    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
    })

print(f"Model Performance ==")
print(f"Accuracy: {acc:.4f}, Precision: {prec:.4f}, Recall: {rec:.4f}")

# Find best K
results_df = pd.DataFrame(results)
best_k = results_df.loc[results_df['Accuracy'].idxmax(), 'k']
print(f"Best K value: {best_k} (Accuracy: {results_df['Accuracy'].max():.4f})")

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

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      17.99      10.38      122.80      1001.0      0.11840
1      20.57      17.77      132.90      1326.0      0.08474
2      13.69      21.25      130.00      1021.0      0.10960
3      11.42      20.38      77.58      386.1      0.18250
4      20.29      14.34      135.10      1297.0      0.10030

   mean compactness  mean concavity  mean concave points  mean symmetry \
0      0.27760      0.3001      0.14710      0.2435
1      0.07864      0.0869      0.07017      0.1812
2      0.13990      0.1374      0.12790      0.2089
3      0.28390      0.2416      0.10520      0.2597
4      0.13280      0.1980      0.10430      0.1809

   mean fractal dimension  ...  worst texture  worst perimeter  worst area \
0      0.07871  ...      17.33      184.60      2019.0
1      0.05867  ...      23.41      158.80      1956.0
2      0.05999  ...      25.53      152.50      1789.0
3      0.09744  ...      26.50      99.87      567.7
4      0.05983  ...      16.07      152.20      1575.0

   worst smoothness  worst compactness  worst concavity  worst concave points \
0      0.1622      0.6656      0.7119      0.2634
1      0.1238      0.1886      0.2416      0.1880
2      0.1444      0.4245      0.4504      0.2430
3      0.2098      0.8663      0.6869      0.2575
4      0.1374      0.2050      0.4600      0.1625

   worst symmetry  worst fractal dimension  target
0      0.4601      0.11890      0
1      0.2750      0.08902      0
2      0.1613      0.00758      0
3      0.4638      0.17300      0
4      0.2364      0.07678      0

[5 rows x 31 columns]

<class '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 symmetry          569 non-null    float64
9   mean fractal dimension  569 non-null    float64
10  radius error           569 non-null    float64
11  texture error          569 non-null    float64
12  perimeter error        569 non-null    float64
13  area error             569 non-null    float64
14  smoothness error       569 non-null    float64
15  compactness error      569 non-null    float64
16  concavity error        569 non-null    float64
17  concave points error   569 non-null    float64
18  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 symmetry         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
0    357
1    212
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    14.127282    19.286449     91.869033    654.881014
std      3.524049     4.301036    24.289381    351.914129
min      6.981000     9.710000    43.790000    143.500000
25%     11.700000    16.170000    75.170000    420.300000
50%     13.370000    18.840000    86.240000    551.100000
75%     15.780000    21.800000    104.100000    762.700000
max     28.110000    39.280000    188.500000    2501.000000

   mean smoothness  mean compactness  mean concavity  mean concave points \
count    569.000000    569.000000    569.000000    569.000000
mean     0.206360      0.124311     0.089789     0.248919
std      0.101064      0.052813     0.079720     0.038803
min      0.026230      0.013980     0.000000     0.000000
25%      0.086370      0.045400     0.023660     0.020110
50%      0.095870      0.092630     0.061540     0.033500
75%      0.165300      0.120400     0.107000     0.074000
max      0.183800      0.345400     0.428800     0.201200

   mean symmetry  mean fractal dimension  ...  worst texture \
count    569.000000    569.000000  ...    569.000000
mean     0.181162      0.062758  ...    25.677223
std      0.027114      0.007060  ...     4.146258
min      0.106000      0.049960  ...    12.020000
25%      0.161900      0.039700  ...    21.080000
50%      0.179200      0.061540  ...    25.410000
75%      0.195700      0.066120  ...    29.720000
max      0.304000      0.097440  ...    49.540000

   worst perimeter  worst area  worst smoothness  worst compactness \
count    569.000000    569.000000    569.000000    569.000000
mean    107.261213    880.583128     0.132369     0.254265
std     31.602542    569.355993     0.022832     0.157316
min      50.410000    185.200000     0.071170     0.027290
25%      84.110000    515.300000     0.116600     0.147200
50%      97.660000    686.500000     0.131300     0.211900
75%     125.400000    1084.000000     0.146000     0.339100
max     291.200000    4294.000000     0.222600     1.058000

   worst concavity  worst concave points  worst symmetry \
count    569.000000    569.000000    569.000000
mean     0.272188      0.114006     0.290076
std      0.288624      0.065732     0.061861
min      0.000000      0.000000     0.155000
25%      0.114500      0.064930     0.250400
50%      0.224700      0.039930     0.282200
75%      0.382900      0.161400     0.317900
max      1.252000      0.291000     0.663800

   worst fractal dimension  target
count    569.000000    569.000000
mean     0.083946     0.627417
std      0.018061     0.483918
min      0.055040     0.000000
25%      0.071460     0.000000
50%      0.080040     1.000000
75%      0.092080     1.000000
max      0.207500     1.000000

[8 rows x 31 columns]

=====
MISSING VALUES
=====
✓ No missing values found!

=====
FEATURE DISTRIBUTIONS
=====
Saved visualization to 'feature_distributions.png'

Distribution of mean radius
Distribution of mean texture
Distribution of mean perimeter
Distribution of mean area
```

