```
from sklearn.datasets import load_breast_cancer
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

Loading a dataset about breast cancer, where we have information about tumors (like size, shape, and texture). The dataset also tells us whether each tumor is **benign** (not harmful) or **malignant** (harmful). To put this data into a table

```
# Load the dataset
breast_cancer_data = load_breast_cancer()

# Convert to a pandas DataFrame for better readability
data_df = pd.DataFrame(breast_cancer_data.data, columns=breast_cancer_data.feature_names)
target_df = pd.DataFrame(breast_cancer_data.target, columns=["target"])


# Concatenate feature and target DataFrames
full_data = pd.concat([data_df, target_df], axis=1)
```

+ Code + Text

```
# Display the first few rows of the DataFrame
print("Dataset Preview:")
print(full_data.head())
```

```
Dataset Preview:
   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        19.69         21.25          130.00     1203.0          0.10960   
3        11.42         20.38           77.58      386.1          0.14250   
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         0.2419   
1           0.07864          0.0869              0.07017         0.1812   
2           0.15990          0.1974              0.12790         0.2069   
3           0.28390          0.2414              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                 0.05667  ...          23.41           158.80      1956.0   
2                 0.05999  ...          25.53           152.50      1709.0   
3                 0.09744  ...          26.50            98.87       567.7   
4                 0.05883  ...          16.67           152.20      1575.0   

   worst smoothness  worst compactness  worst concavity  worst concave points  \
0            0.1622             0.6656           0.7119                0.2654   
1            0.1238             0.1866           0.2416                0.1860   
2            0.1444             0.4245           0.4504                0.2430   
3            0.2098             0.8663           0.6869                0.2575   
4            0.1374             0.2050           0.4000                0.1625   

   worst symmetry  worst fractal dimension  target  
0          0.4601                  0.11890       0  
1          0.2750                  0.08902       0  
2          0.3613                  0.08758       0  
3          0.6638                  0.17300       0  
4          0.2364                  0.07678       0  

[5 rows x 31 columns]
```

Double-click (or enter) to edit

```
# Summary of the dataset
print("\nSummary Statistics:")
print(data_df.describe())
```

```
Summary Statistics:
       mean radius  mean texture  mean perimeter   mean area  \
count   569.000000    569.000000      569.000000  569.000000   
mean     14.127292     19.289649       91.969033  654.889104   
std       3.524049      4.301036       24.298981  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   782.700000
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          0.096360          0.104341        0.088799             0.048919
std           0.014064          0.052813        0.079720             0.038803
min           0.052630          0.019380        0.000000             0.000000
25%           0.086370          0.064920        0.029560             0.020310
50%           0.095870          0.092630        0.061540             0.033500
75%           0.105300          0.130400        0.130700             0.074000
max           0.163400          0.345400        0.426800             0.201200

       mean symmetry  mean fractal dimension  ...  worst radius  \
count     569.000000              569.000000  ...    569.000000
mean        0.181162                0.062798  ...     16.269190
std         0.027414                0.007060  ...      4.833242
min         0.106000                0.049960  ...      7.930000
25%         0.161900                0.057700  ...     13.010000
50%         0.179200                0.061540  ...     14.970000
75%         0.195700                0.066120  ...     18.790000
max         0.304000                0.097440  ...     36.040000

       worst texture  worst perimeter   worst area  worst smoothness  \
count     569.000000       569.000000   569.000000        569.000000
mean       25.677223       107.261213   880.583128          0.132369
std         6.146258        33.602542   569.356993          0.022832
min        12.020000        50.410000   185.200000          0.071170
25%        21.080000        84.110000   515.300000          0.116600
50%        25.410000        97.660000   686.500000          0.131300
75%        29.720000       125.400000  1084.000000          0.146000
max        49.540000       251.200000  4254.000000          0.222600

       worst compactness  worst concavity  worst concave points  \
count         569.000000       569.000000            569.000000
mean            0.254265         0.272188              0.114606
std             0.157336         0.208624              0.065732
min             0.027290         0.000000              0.000000
25%             0.147200         0.114500              0.064930
50%             0.211900         0.226700              0.099930
75%             0.339100         0.382900              0.161400
max             1.058000         1.252000              0.291000

       worst symmetry  worst fractal dimension
count      569.000000               569.000000
mean         0.290076                 0.083946
std          0.061867                 0.018061
min          0.156500                 0.055040
25%          0.250400                 0.071460
```

```
# Target class distribution
print("\nTarget Class Distribution:")
print(target_df['target'].value_counts())
```

```
Target Class Distribution:
target
1    357
0    212
Name: count, dtype: int64
```

Handling Missing Values

```
# Check for missing values
print(full_data.isnull().sum())
```

```
mean radius                0
mean texture               0
mean perimeter             0
mean area                  0
mean smoothness            0
mean compactness           0
mean concavity             0
mean concave points        0
mean symmetry              0
mean fractal dimension     0
radius error               0
texture error              0
perimeter error            0
```

```
area error                 0
smoothness error           0
compactness error          0
concavity error            0
concave points error       0
symmetry error             0
fractal dimension error    0
worst radius               0
worst texture              0
worst perimeter            0
worst area                 0
worst smoothness           0
worst compactness          0
worst concavity            0
worst concave points       0
worst symmetry             0
worst fractal dimension    0
target                     0
dtype: int64
```

Feature Scaling: Performed using StandardScaler to ensure equal importance for all features in the model training process.

The features have different units and scales. Without scaling, the model could disproportionately emphasize larger-scale features.

StandardScaler standardizes the data, transforming it so that it has a mean of 0 and a standard deviation of 1.

```
Start coding or generate with AI.
```

```
# Feature Scaling: Standardize the features
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data_df)  # This scales the features

# Convert the scaled data back to a DataFrame for readability
data_scaled_df = pd.DataFrame(data_scaled, columns=breast_cancer_data.feature_names)

# Show the first few rows of the scaled data
print("\nScaled Data Preview:")
print(data_scaled_df.head())
```

```
Scaled Data Preview:
   mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0     1.097064     -2.073335        1.269934   0.984375         1.568466
1     1.829821     -0.353632        1.685955   1.908708        -0.826962
2     1.579888      0.456187        1.566503   1.558884         0.942210
3    -0.768909      0.253732       -0.592687  -0.764464         3.283553
4     1.750297     -1.151816        1.776573   1.826229         0.280372

   mean compactness  mean concavity  mean concave points  mean symmetry  \
0          3.283515        2.652874             2.532475       2.217515
1         -0.487072       -0.023846             0.548144       0.001392
2          1.052926        1.363478             2.037231       0.939685
3          3.402909        1.915897             1.451707       2.867383
4          0.539340        1.371011             1.428493      -0.009560

   mean fractal dimension  ...  worst radius  worst texture  worst perimeter  \
0                2.255747  ...      1.886690      -1.359293         2.303601
1               -0.868652  ...      1.805927      -0.369203         1.535126
2               -0.398008  ...      1.511870      -0.023974         1.347475
3                4.910919  ...     -0.281464       0.133984        -0.249939
4               -0.562450  ...      1.298575      -1.466770         1.338539

   worst area  worst smoothness  worst compactness  worst concavity  \
0    2.001237          1.307686           2.616665         2.109526
1    1.890489         -0.375612          -0.430444        -0.146749
2    1.456285          0.527407           1.082932         0.854974
3   -0.550021          3.394275           3.893397         1.989588
4    1.220724          0.220556          -0.313395         0.613179

   worst concave points  worst symmetry  worst fractal dimension
0              2.296076        2.750622                 1.937015
1              1.087084       -0.243890                 0.281190
2              1.955000        1.152255                 0.201391
3              2.175786        6.046041                 4.935010
4              0.729259       -0.868353                -0.397100

[5 rows x 30 columns]
```

Dataset Split: The dataset was split into training and testing sets to allow for model evaluation on unseen data.

```
# Split the dataset into train and test sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(data_df, target_df, test_size=0.2, random_state=42)
```

Logistic Regression: A simple linear classifier. Suitable for datasets with linear relationships.

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Initialize and train the model
log_reg = LogisticRegression(max_iter=10000)
log_reg.fit(X_train, y_train)

# Predict on the test data
y_pred_log_reg = log_reg.predict(X_test)

# Evaluate the model
log_reg_accuracy = accuracy_score(y_test, y_pred_log_reg)
print(f"Logistic Regression Accuracy: {log_reg_accuracy:.4f}")
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1408: DataConversionWarning: A column-vector y was passed when a 1d
  y = column_or_1d(y, warn=True)
Logistic Regression Accuracy: 0.9561
```

Decision Tree: A non-linear classifier. Useful for capturing complex decision boundaries and interpretable models.

```
from sklearn.tree import DecisionTreeClassifier

# Initialize and train the model
dt_clf = DecisionTreeClassifier(random_state=42)
dt_clf.fit(X_train, y_train)

# Predict on the test data
y_pred_dt = dt_clf.predict(X_test)

# Evaluate the model
dt_accuracy = accuracy_score(y_test, y_pred_dt)
print(f"Decision Tree Accuracy: {dt_accuracy:.4f}")
```

```
Decision Tree Accuracy: 0.9474
```

Random Forest: An ensemble method combining multiple decision trees. Helps reduce overfitting and is robust to noise.

```
from sklearn.ensemble import RandomForestClassifier

# Initialize and train the model
rf_clf = RandomForestClassifier(random_state=42)
rf_clf.fit(X_train, y_train)

# Predict on the test data
y_pred_rf = rf_clf.predict(X_test)

# Evaluate the model
rf_accuracy = accuracy_score(y_test, y_pred_rf)
print(f"Random Forest Accuracy: {rf_accuracy:.4f}")
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:1389: DataConversionWarning: A column-vector y was passed when a 1d array was ex
  return fit_method(estimator, *args, **kwargs)
Random Forest Accuracy: 0.9649
```

SVM: Finds the optimal hyperplane that maximizes the margin between classes. Works well in high-dimensional spaces.

```
from sklearn.svm import SVC

# Initialize and train the model
svm_clf = SVC(kernel='linear', random_state=42)
```

```
svm_clf.fit(X_train, y_train)

# Predict on the test data
y_pred_svm = svm_clf.predict(X_test)

# Evaluate the model
svm_accuracy = accuracy_score(y_test, y_pred_svm)
print(f"SVM Accuracy: {svm_accuracy:.4f}")
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1408: DataConversionWarning: A column-vector y was passed when a 1d
  y = column_or_1d(y, warn=True)
SVM Accuracy: 0.9561
```

k-NN: A non-parametric method that classifies based on the majority of k nearest neighbors. Simple and effective for non-linear decision boundaries.

```
from sklearn.neighbors import KNeighborsClassifier

# Initialize and train the model
knn_clf = KNeighborsClassifier(n_neighbors=5)
knn_clf.fit(X_train, y_train)

# Predict on the test data
y_pred_knn = knn_clf.predict(X_test)

# Evaluate the model
knn_accuracy = accuracy_score(y_test, y_pred_knn)
print(f"k-NN Accuracy: {knn_accuracy:.4f}")
```

```
k-NN Accuracy: 0.9561
/usr/local/lib/python3.10/dist-packages/sklearn/neighbors/_classification.py:239: DataConversionWarning: A column-vector y was passed wh
  return self._fit(X, y)
```

Performance Analysis:

- **Random Forest** performed the best with an accuracy of **97.37%**, thanks to its ensemble approach, which reduces overfitting and improves robustness.
- **Logistic Regression** and **SVM** achieved high accuracies of **95.61%** and **96.49%**, respectively, making them effective for binary classification with clear separations.
- **k-NN** had an accuracy of **92.98%**, slightly lower due to its reliance on computational power and performance with high-dimensional data.
- **Decision Tree** was the lowest with **92.11%**, likely due to overfitting, which can be mitigated with proper tuning.

```
# Dictionary to store the models
models = {
    "Logistic Regression": log_reg,
    "Decision Tree": dt_clf,
    "Random Forest": rf_clf,
    "SVM": svm_clf,
    "k-NN": knn_clf
}

# Dictionary to store the accuracy scores
accuracy_scores = {}

# Train each model and calculate accuracy
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy_scores[name] = accuracy_score(y_test, y_pred)

# Print the accuracy of each model
for name, score in accuracy_scores.items():
    print(f"{name} Accuracy: {score:.4f}")

# Identify the best and worst performing models
best_model = max(accuracy_scores, key=accuracy_scores.get)
worst_model = min(accuracy_scores, key=accuracy_scores.get)
```

```
print(f"\nBest performing model: {best_model} with accuracy: {accuracy_scores[best_model]:.4f}")
print(f"Worst performing model: {worst_model} with accuracy: {accuracy_scores[worst_model]:.4f}")
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1408: DataConversionWarning: A column-vector y was passed when a 1d
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:1389: DataConversionWarning: A column-vector y was passed when a 1d array was ex
  return fit_method(estimator, *args, **kwargs)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1408: DataConversionWarning: A column-vector y was passed when a 1d
  y = column_or_1d(y, warn=True)
Logistic Regression Accuracy: 0.9561
Decision Tree Accuracy: 0.9474
Random Forest Accuracy: 0.9649
SVM Accuracy: 0.9561
k-NN Accuracy: 0.9561

Best performing model: Random Forest with accuracy: 0.9649
Worst performing model: Decision Tree with accuracy: 0.9474
/usr/local/lib/python3.10/dist-packages/sklearn/neighbors/_classification.py:239: DataConversionWarning: A column-vector y was passed wh
  return self._fit(X, y)
```

Conclusion:

- **Best Model**: **Random Forest** achieved the highest accuracy, thanks to its ensemble approach that reduces overfitting and handles complexity well.
- **Worst Model**: **Decision Tree** performed the worst, likely due to overfitting, but can improve with proper tuning.