ID: cell-a571863327493058 Read-only

Step 1: Select and Explain the Dataset

The famous "Breast Cancer" dataset from scikit-learn. The Breast Cancer dataset contains features computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. The goal is to classify tumors as malignant or benign based on various characteristics.

ID: cell-014575f78d83739e Read-only

Step 2: Read Data (1 point)

Read dataset into a Pandas DataFrame.

In [1]: ID: cell-5ce183277081aa38 Autograded answer

```
import pandas as pd
# from sklearn.datasets import load_breast_cancer

# Load the Breast Cancer dataset
# data = load_breast_cancer()

# breast_cancer_data = pd.DataFrame(data.data, columns=data.feature_names)
# breast_cancer_data['target'] = data.target
breast_cancer_data = pd.read_csv('breast_cancer.csv')
breast_cancer_data.head()
# breast_cancer_data.to_csv('breast_cancer.csv')
```

Out[1]:

	Unnamed: 0	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	...	worst texture	worst perimeter	worst area	worst smoothness	worst compactness	worst concavity	worst concave points	worst symmetry	worst fractal dimension	t
0	0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	...	17.33	184.60	2019.0	0.1622	0.6656	0.7119	0.2654	0.4601	0.11890	
1	1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	...	23.41	158.80	1956.0	0.1238	0.1866	0.2416	0.1860	0.2750	0.08902	
2	2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	...	25.53	152.50	1709.0	0.1444	0.4245	0.4504	0.2430	0.3613	0.08758	
3	3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	...	26.50	98.87	567.7	0.2098	0.8663	0.6869	0.2575	0.6638	0.17300	
4	4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	...	16.67	152.20	1575.0	0.1374	0.2050	0.4000	0.1625	0.2364	0.07678	

5 rows × 32 columns

In [2]: -

```
breast_cancer_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Unnamed: 0               569 non-null    int64
 1   mean radius              569 non-null    float64
 2   mean texture             569 non-null    float64
 3   mean perimeter           569 non-null    float64
 4   mean area                569 non-null    float64
 5   mean smoothness          569 non-null    float64
 6   mean compactness         569 non-null    float64
 7   mean concavity           569 non-null    float64
 8   mean concave points      569 non-null    float64
 9   mean symmetry            569 non-null    float64
 10  mean fractal dimension   569 non-null    float64
 11  radius error             569 non-null    float64
 12  texture error            569 non-null    float64
 13  perimeter error          569 non-null    float64
 14  area error               569 non-null    float64
 15  smoothness error         569 non-null    float64
 16  compactness error        569 non-null    float64
 17  concavity error          569 non-null    float64
 18  concave points error     569 non-null    float64
 19  symmetry error           569 non-null    float64
 20  fractal dimension error  569 non-null    float64
 21  worst radius             569 non-null    float64
 22  worst texture            569 non-null    float64
 23  worst perimeter          569 non-null    float64
 24  worst area               569 non-null    float64
 25  worst smoothness         569 non-null    float64
 26  worst compactness        569 non-null    float64
 27  worst concavity          569 non-null    float64
 28  worst concave points     569 non-null    float64
 29  worst symmetry           569 non-null    float64
 30  worst fractal dimension  569 non-null    float64
 31  target                   569 non-null    int64
dtypes: float64(30), int64(2)
memory usage: 142.4 KB
```

In [3]: Points: 1 ID: cell-0f7bf8d5311ac680 Autograder tests

```
#step 2
assert len(breast_cancer_data) == 569, "Incorrect number of rows"
```

ID: cell-5ede6f97ca578e1a Read-only

Step 3: Exploratory Data Analysis (EDA) and Numerical Results (3 points)

In [4]: ID: cell-985e49e8cfb5ff4e Autograded answer

```
# Calculate summary statistics
mean_radius_mean = breast_cancer_data['mean radius'].mean()
worst_perimeter_max = breast_cancer_data['worst perimeter'].max()
mean_smoothness_sum = breast_cancer_data['mean smoothness'].sum()

mean_radius_mean, worst_perimeter_max, mean_smoothness_sum
```

Out[4]: (14.127291739894552, 251.2, 54.82900000000001)

In [5]: Points: 3 ID: cell-932f975bc5a37635 Autograder tests

```
#step3
assert round(mean_radius_mean, 2) == 14.13, "Incorrect mean mean radius"
assert round(worst_perimeter_max, 2) == 251.2, "Incorrect max worst perimeter"
assert round(mean_smoothness_sum, 2) == 54.83, "Incorrect sum mean smoothness"
```

ID: cell-0709130f89512eb6 Read-only

Step 4: Setup Experiment and Model Training (3 points)

In [6]: ID: cell-c7ace6ade31e8be3 Autograded answer

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

# Split data into features (X) and target (y)
X = breast_cancer_data.drop('target', axis=1)
y = breast_cancer_data['target']

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train the random forest classifier
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)
```

Out[6]:

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

In [7]: Points: 3 ID: cell-8c31d16161235116 Autograder tests

```
#step4
# def mycheck(yp,yt,error_percent=10):
#     p=error_percent/100
#     return yt*(1-p)<yp and yp < yt*(1+p)

assert len(X_train) == 455, "Incorrect number of training samples"
assert len(X_test) == 114, "Incorrect number of test samples"
```

ID: cell-6e6b31589e3436b4 Read-only

Step 5: Evaluation (3 points)

In [8]: ID: cell-6510510e34498eab Autograded answer

```
from sklearn.metrics import accuracy_score, precision_score, recall_score

# Make predictions on the test set
y_pred = model.predict(X_test)

# Calculate accuracy, precision, and recall
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

accuracy, precision, recall
```

Out[8]: (0.9649122807017544, 0.958904109589041, 0.9859154929577465)

In [9]: Points: 3 ID: cell-e13ffa942cad9259 Autograder tests

```
#step5
def mycheck(yp,yt,error_percent=10):
    p=error_percent/100
    return yt*(1-p)<yp and yp < yt*(1+p)

assert mycheck(accuracy,0.965), "Incorrect accuracy"
assert mycheck(precision,0.959), "Incorrect precision"
assert mycheck(recall,0.986), "Incorrect recall"

# assert round(accuracy, 2) == 0.97, "Incorrect accuracy"
# assert round(precision, 2) == 0.98, "Incorrect precision"
# assert round(recall, 2) == 0.96, "Incorrect recall"
```

In [ ]: -