*##Load Dataset & Explore Structure*

**import** pandas **as** pd

**from** sklearn.datasets **import** load\_iris

*# Example using built-in dataset; replace with pd.read\_csv("your\_data.csv") for custom data*

iris **=** load\_iris(as\_frame**=True**) df **=** iris**.**frame

*# Quick look at first five rows*

print(df**.**head())

*# Column data types* print(df**.**info()) print(df**.**describe())

sepal length (cm) sepal width (cm) petal length (cm) petal width (cm) \

|  |  |  |
| --- | --- | --- |
| 0 5.1 3.5 | 1.4 | 0.2 |
| 1 4.9 3.0 | 1.4 | 0.2 |
| 2 4.7 3.2 | 1.3 | 0.2 |
| 3 4.6 3.1 | 1.5 | 0.2 |
| 4 5.0 3.6 | 1.4 | 0.2 |
| target  0 0 |  | |
| 1 0 |
| 2 0 |
| 3 0 |
| 4 0 |
| <class 'pandas.core.frame.DataFrame'> |

RangeIndex: 150 entries, 0 to 149 Data columns (total 5 columns):

# Column Non-Null Count Dtype

1. sepal length (cm) 150 non-null float64
2. sepal width (cm) 150 non-null float64
3. petal length (cm) 150 non-null float64
4. petal width (cm) 150 non-null float64
5. target 150 non-null int64 dtypes: float64(4), int64(1)

memory usage: 6.0 KB None

sepal length (cm) sepal width (cm) petal length (cm) \

|  |  |  |  |
| --- | --- | --- | --- |
| count | 150.000000 | 150.000000 | 150.000000 |
| mean | 5.843333 | 3.057333 | 3.758000 |
| std | 0.828066 | 0.435866 | 1.765298 |
| min | 4.300000 | 2.000000 | 1.000000 |
| 25% | 5.100000 | 2.800000 | 1.600000 |
| 50% | 5.800000 | 3.000000 | 4.350000 |
| 75% | 6.400000 | 3.300000 | 5.100000 |

max 7.900000 4.400000 6.900000

|  |  |  |  |
| --- | --- | --- | --- |
|  | petal | width (cm) | target |
| count |  | 150.000000 | 150.000000 |
| mean |  | 1.199333 | 1.000000 |
| std |  | 0.762238 | 0.819232 |
| min |  | 0.100000 | 0.000000 |
| 25% |  | 0.300000 | 0.000000 |
| 50% |  | 1.300000 | 1.000000 |
| 75% |  | 1.800000 | 2.000000 |
| max |  | 2.500000 | 2.000000 |

*##Handle Missing Values*

*# Check missing values*

print(df**.**isnull()**.**sum())

*# Impute numerical features with median (robust against outliers)*

**from** sklearn.impute **import** SimpleImputer

imputer **=** SimpleImputer(strategy**=**'median')

df[df**.**columns[:**-**1]] **=** imputer**.**fit\_transform(df[df**.**columns[:**-**1]])

|  |  |  |
| --- | --- | --- |
| sepal | length (cm) | 0 |
| sepal | width (cm) | 0 |
| petal | length (cm) | 0 |
| petal | width (cm) | 0 |
| target |  | 0 |

dtype: int64

*##Encode Categorical Variables*

**from** sklearn.preprocessing **import** OneHotEncoder

*# Example: Suppose 'species' is categorical, otherwise skip for all-numeric datasets*

encoder **=** OneHotEncoder(sparse\_output**=False**, drop**=**'first')

*# drop for reference avoidance*

species\_encoded **=** encoder**.**fit\_transform(df[['target']])

*##Normalize / Scale Numerical Features*

**from** sklearn.preprocessing **import** StandardScaler

scaler **=** StandardScaler()

df[df**.**columns[:**-**1]] **=** scaler**.**fit\_transform(df[df**.**columns[:**-**1]])

*##Split Data*

**from** sklearn.model\_selection **import** train\_test\_split

X **=** df[df**.**columns[:**-**1]] y **=** df['target']

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split( X, y, test\_size**=**0.2, random\_state**=**42, stratify**=**y)

*##Choose and Train a Model*

**from** sklearn.ensemble **import** RandomForestClassifier

clf **=** RandomForestClassifier(random\_state**=**42) clf**.**fit(X\_train, y\_train)

RandomForestClassifier

[?](https://scikit-learn.org/1.6/modules/generated/sklearn.ensemble.RandomForestClassifier.html)i RandomForestClassifier(random\_state=42) *##Cross-Validation*

**from** sklearn.model\_selection **import** cross\_val\_score

scores **=** cross\_val\_score(clf, X\_train, y\_train, cv**=**5, scoring**=**'accuracy')

print("Cross-validation Accuracy: %.2f%% +/- %.2f" **%** (scores**.**mean() **\*** 100, scores**.**std() **\***

100))

Cross-validation Accuracy: 95.00% +/- 1.67

*##Hyperparameter Tuning*

**from** sklearn.model\_selection **import** GridSearchCV

param\_grid **=** {

'n\_estimators': [50, 100, 200],

'max\_depth': [**None**, 5, 10],

'min\_samples\_split': [2, 5, 10]

}

grid **=** GridSearchCV(clf, param\_grid, cv**=**3, scoring**=**'accuracy') grid**.**fit(X\_train, y\_train)

print("Best Parameters:", grid**.**best\_params\_)

print("Best Cross-Validation Accuracy: %.2f%%" **%** (grid**.**best\_score\_ **\*** 100))

Best Parameters: {'max\_depth': None, 'min\_samples\_split': 5, 'n\_estimators': 100} Best Cross-Validation Accuracy: 95.83%

*##Evaluation Metrics*

**from** sklearn.metrics **import** accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_auc\_score, confusion\_matrix, roc\_curve

**import** matplotlib.pyplot **as** plt

y\_pred **=** grid**.**predict(X\_test)

y\_proba **=** grid**.**predict\_proba(X\_test)[:, 1] **if** hasattr(grid, "predict\_proba") **else None**

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print("Precision:", precision\_score(y\_test, y\_pred, average**=**'weighted')) print("Recall:", recall\_score(y\_test, y\_pred, average**=**'weighted')) print("F1 Score:", f1\_score(y\_test, y\_pred, average**=**'weighted'))

*# If binary classification or appropriate multi-class handling*

**if** y\_proba **is not None and** len(set(y\_test)) **==** 2: print("ROC AUC:", roc\_auc\_score(y\_test, y\_proba))

*# ROC Curve*

fpr, tpr, \_ **=** roc\_curve(y\_test, y\_proba) plt**.**plot(fpr, tpr)

plt**.**title('ROC Curve') plt**.**xlabel('False Positive Rate') plt**.**ylabel('True Positive Rate') plt**.**show()

*# Confusion Matrix*

plt**.**matshow(confusion\_matrix(y\_test, y\_pred), cmap**=**plt**.**cm**.**Blues) plt**.**title('Confusion Matrix')

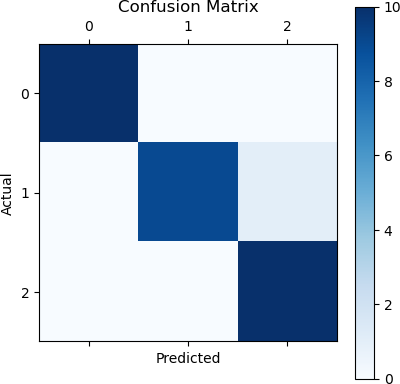
plt**.**colorbar() plt**.**xlabel('Predicted') plt**.**ylabel('Actual') plt**.**show()

Accuracy: 0.9666666666666667

Precision: 0.9696969696969696

Recall: 0.9666666666666667

F1 Score: 0.9665831244778613



*##Pipeline Integration*

**from** sklearn.pipeline **import** Pipeline

**from** sklearn.compose **import** ColumnTransformer

*# If you have categorical columns, setup transformers accordingly*

preprocessor **=** ColumnTransformer( transformers**=**[

*# Example: numeric and categorical*

('num', StandardScaler(), X**.**columns), *# List only numeric columns here # ('cat', OneHotEncoder(handle\_unknown='ignore'), categorical\_columns) # uncomment if categorical*

]

)

pipeline **=** Pipeline([ ('preprocessing', preprocessor),

('classifier', RandomForestClassifier(**\*\***grid**.**best\_params\_))

])

pipeline**.**fit(X\_train, y\_train)

# Pipeline

[**?**](https://scikit-learn.org/1.6/modules/generated/sklearn.pipeline.Pipeline.html)**i**

# preprocessing: ColumnTransformer

[**?**](https://scikit-learn.org/1.6/modules/generated/sklearn.compose.ColumnTransformer.html)

**num**

StandardScaler

[?](https://scikit-learn.org/1.6/modules/generated/sklearn.preprocessing.StandardScaler.html)

RandomForestClassifier

[?](https://scikit-learn.org/1.6/modules/generated/sklearn.ensemble.RandomForestClassifier.html)