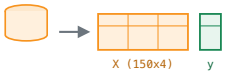
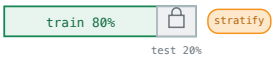
scikit-learn

The scikit-learn workflow, from splitting data and building pipelines to tuning, scoring, and saving a model.

Data: Load & Split

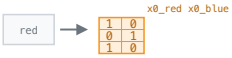
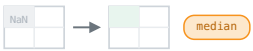
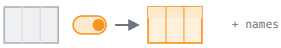
Get arrays in, hold out a test set you never touch while modeling.

PURPOSE	COMMAND	RESULT
Load a toy dataset as a DataFrame.	`X, y = load_iris(return_X_y=True, as_frame=True)`	
Fetch a real dataset by name.	`X, y = fetch_openml("titanic", version=1, return_X_y=True, as_frame=True)`	
Make synthetic data for demos.	`X, y = make_classification(n_samples=1000, n_classes=3, n_informative=3)`	
Hold out a test set (stratified).	`train_test_split(X, y, test_size=0.2, stratify=y, random_state=0)`	

X is (n_samples, n_features); split off the test set and do not look at it until final evaluation

Preprocess: Scale, Encode, Impute

Transformers learn statistics with .fit, then reshape any data with .transform.

PURPOSE	COMMAND	RESULT
Standardize numeric columns.	`StandardScaler().fit_transform(X_tr)`	
One-hot encode categoricals.	`OneHotEncoder(handle_unknown="ignore", sparse_output=False)`	
Fill in missing values.	`SimpleImputer(strategy="median")`	
Get output column names back.	`enc.get_feature_names_out()`	
Return a DataFrame, not an array.	`StandardScaler().set_output("pandas")`	

fit learns statistics from training data only; that is what keeps test data from leaking in

Estimator: Fit, Predict, Score

.fit learns, .predict applies, .score reports. Same three calls for every algorithm.

PURPOSE	COMMAND	RESULT
Pick & train a classifier.	`clf =RandomForestClassifier( n_estimators=300).fit(X_tr, y_tr)`	
Predict labels on new data.	`y_pred = clf.predict(X_te)`	
Predict class probabilities.	`proba = clf.predict_proba(X_te)`	
Default score on a test set.	`clf.score(X_te, y_te)`	
Swap in another algorithm.	`from sklearn.svm import SVC`	

the interface is uniform, so swapping algorithms is a one-line change

Pipeline: Chain Steps Safely

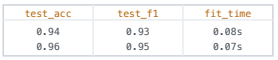
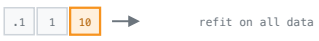
Wrap preprocessing and a model in one estimator that travels together.

PURPOSE	COMMAND	RESULT
Chain a scaler + model.	`pipe =make_pipeline(StandardScaler(), LogisticRegression(max_iter=1000))`	
Route numeric vs categorical cols.	`ColumnTransformer([("num", StandardScaler(), num_cols), ("cat", OneHotEncoder(), cat_cols)])`	
Auto-select columns by dtype.	`make_column_selector( dtype_include="number")`	
Fit the whole chain at once.	`pipe.fit(X_tr, y_tr)`	
Reach into a named step.	`pipe.named_steps["logisticregression"].coef_`	

transforms are fit only on training folds, which prevents the most common data leakage

Validate & Tune

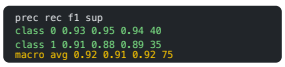
Rotate the held-out fold for honest scores, then search and refit the best config.

PURPOSE	COMMAND	RESULT
K-fold cross-validated score.	`scores =cross_val_score(pipe, X, y, cv=5)`	
Multiple metrics + train scores.	`cross_validate(pipe, X, y, cv=5, scoring=["accuracy","f1_macro"])`	
Grid search over a pipeline.	`gs =GridSearchCV(pipe, {"logisticregression__C": [0.1, 1, 10]}, cv=5)`	
Read the winner.	`gs.best_params_gs.best_score_`	
Randomized search (big spaces).	`RandomizedSearchCV(pipe, ..., n_iter=20)`	

pass the whole pipeline so preprocessing is re-fit inside every fold; tune with step__param naming

Metrics & Evaluation

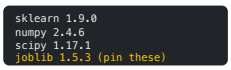
Pick the metric for the task; accuracy hides class imbalance.

PURPOSE	COMMAND	RESULT
Per-class precision/recall/F1.	`print(classification_report(...))`	
Confusion matrix (plotted).	`ConfusionMatrixDisplay.from_estimator(...)`	
Ranking quality (AUC).	`roc_auc_score(y_te, proba[:, 1])`	
Regression error.	`root_mean_squared_errorr2_score`	
Custom scorer for search.	`make_scorer(f1_score, average="macro")`	

classification_report, confusion matrix, and ROC AUC for classes; RMSE / R² for regression

Persist: Save & Load

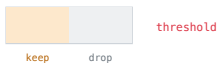
Train once, serve many times. Persist the whole fitted pipeline with joblib.

PURPOSE	COMMAND	RESULT
Save a fitted pipeline.	`joblib.dump(pipe, "model.joblib")`	
Load it back later.	`pipe = joblib.load("model.joblib")`	
Predict in a fresh process.	`joblib.load("model.joblib").predict(X_new)`	
Check installed versions.	`sklearn.show_versions()`	

the serialized format is not guaranteed across versions: pin and record them; only unpickle models you trust

Inspect & Select

Understand what the model uses and trim the inputs.

PURPOSE	COMMAND	RESULT
Tree feature importances.	`clf.feature_importances_`	
Model-agnostic importance.	`permutation_importance(clf, X_te, y_te)`	
Keep the best K features.	`SelectKBest(f_classif, k=10)` `  .fit_transform(X, y)`	
Select via a model's weights.	`SelectFromModel(estimator,` `  threshold="median")`	
Compress with PCA.	`PCA(n_components=2).fit_transform(X)`	
Visualize the estimator.	`set_config(display="diagram")`	

feature_importances_ is fast; permutation_importance works for any fitted estimator