Python ↔ R Machine Learning Cheat Sheet: scikit-learn ↔ caret / tidymodels

Focus: typical supervised ML workflow (split → preprocess → train → tune → evaluate). R shows both caret and tidymodels patterns.

Setup (classification example data)

R (caret / tidymodels)	Python (scikit-learn)
`# caret library(caret) set.seed(1) data(iris) df <- iris # tidymodels library(tidymodels) set.seed(1)`	`import numpy as np import pandas as pd from sklearn.datasets import load_iris iris = load_iris(as_frame=True) df = iris.frame # target column name: 'target'`

Train/test split

R (caret / tidymodels)	Python (scikit-learn)
`# caret set.seed(1) idx <- createDataPartition(df$Species, p=0.8, list=FALSE) train <- df[idx, ] test <- df[-idx, ] # tidymodels set.seed(1) split <- initial_split(df, prop=0.8, strata=Species) train <- training(split) test <- testing(split)`	`from sklearn.model_selection import train_test_split X = df.drop(columns=['target']) y = df['target'] X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_state=1, stratify=y )`

Preprocessing pipeline (scale + one-hot)

R (caret / tidymodels)	Python (scikit-learn)
`# caret: preProcess (numeric only) + dummyVars for categoricals pp <- preProcess(train[, -5], method=c('center','scale')) X_train <- predict(pp, train[, -5]) X_test <- predict(pp, test[, -5]) # tidymodels: recipe rec <- recipe(Species ~ ., data=train) %>% step_normalize(all_numeric_predictors())`	`from sklearn.compose import ColumnTransformer from sklearn.preprocessing import StandardScaler, OneHotEncoder from sklearn.pipeline import Pipeline num_cols = X_train.select_dtypes(include='number').columns cat_cols = X_train.select_dtypes(exclude='number').columns preprocess = ColumnTransformer(transformers=[('num', StandardScaler(), num_cols), ('cat', OneHotEncoder(handle_unknown='ignore'), cat_cols)])`

Baseline classifier (logistic regression)

R (caret / tidymodels)	Python (scikit-learn)

`# caret set.seed(1) ctrl <- trainControl(method='cv', number=5) fit <- train(Species ~ ., data=train, method='multinom', trControl=ctrl, preProcess=c('center','scale')) fit # tidymodels lr_spec <- multinom_reg() %>% set_engine('nnet') %>% set_mode('classification') wf <- workflow() %>% add_recipe(rec) %>% add_model(lr_spec) set.seed(1) res <- fit_resamples(wf, vfold_cv(train, v=5, strata=Species)) collect_metrics(res)`	`from sklearn.linear_model import LogisticRegression from sklearn.model_selection import cross_val_score clf = Pipeline([ ('prep', preprocess), ('model', LogisticRegression(max_iter=1000)) ]) scores = cross_val_score(clf, X_train, y_train, cv=5, scoring='accuracy') scores.mean()`

Fit final model + evaluate (accuracy, confusion matrix)

R (caret / tidymodels)	Python (scikit-learn)
`# caret pred <- predict(fit, newdata=test) confusionMatrix(pred, test$Species) # tidymodels final_fit <- fit(wf, data=train) pred <- predict(final_fit, test) %>% bind_cols(test %>% select(Species)) pred %>% conf_mat(truth=Species, estimate=.pred_class) pred %>% accuracy(truth=Species, estimate=.pred_class)`	`from sklearn.metrics import accuracy_score, confusion_matrix, classification_report clf.fit(X_train, y_train) y_pred = clf.predict(X_test) accuracy_score(y_test, y_pred) confusion_matrix(y_test, y_pred) print(classification_report(y_test, y_pred))`

Random forest

R (caret / tidymodels)	Python (scikit-learn)
`# caret set.seed(1) rf_fit <- train(Species ~ ., data=train, method='rf', trControl=trainControl(method='cv', number=5)) rf_pred <- predict(rf_fit, newdata=test) confusionMatrix(rf_pred, test$Species) # tidymodels rf_spec <- rand_forest(trees=500) %>% set_engine('ranger', importance='impurity') %>% set_mode('classification') rf_wf <- workflow() %>% add_recipe(rec) %>% add_model(rf_spec) set.seed(1) rf_res <- fit_resamples(rf_wf, vfold_cv(train, v=5, strata=Species)) collect_metrics(rf_res)`	`from sklearn.ensemble import RandomForestClassifier rf = Pipeline([ ('prep', preprocess), ('model', RandomForestClassifier(n_estimators=500, random_state=1)) ]) scores = cross_val_score(rf, X_train, y_train, cv=5, scoring='accuracy') scores.mean()`

Hyperparameter tuning (grid search)

R (caret / tidymodels)	Python (scikit-learn)
`# caret tuning example (rf mtry) set.seed(1) grid <- expand.grid(mtry = c(1,2,3,4)) rf_tuned <- train(Species ~ ., data=train, method='rf', trControl=trainControl(method='cv', number=5), tuneGrid=grid) rf_tuned$bestTune # tidymodels tuning example rf_spec <- rand_forest(mtry=tune(), trees=500) %>% set_engine('ranger') %>% set_mode('classification') rf_wf <- workflow() %>% add_recipe(rec) %>% add_model(rf_spec) grid <- grid_regular(mtry(range=c(1,4)), levels=4) set.seed(1) tuned <- tune_grid(rf_wf, resamples=vfold_cv(train, v=5, strata=Species), grid=grid) show_best(tuned, metric='accuracy')`	`from sklearn.model_selection import GridSearchCV rf = Pipeline([('prep', preprocess), ('model', RandomForestClassifier(n_estimators=500, random_state=1))]) param_grid = { 'model__max_depth': [None, 3, 5], 'model__max_features': ['sqrt', 'log2', None] } gs = GridSearchCV(rf, param_grid, cv=5, scoring='accuracy') gs.fit(X_train, y_train) gs.best_params_, gs.best_score_`

Pipelines / workflows (preprocess + model together)

R (caret / tidymodels)	Python (scikit-learn)
`# tidymodels workflow is the direct analogue wf <- workflow() %>% add_recipe(rec) %>% add_model(rf_spec) wf # caret: use preProcess=... inside train(), or preprocess externally rf_fit <- train(Species ~ ., data=train, method='rf', preProcess=c('center','scale'))`	`# scikit-learn Pipeline is the direct analogue pipe = Pipeline([ ('prep', preprocess), ('model', RandomForestClassifier(random_state=1)) ]) pipe`