Random Forest Pipeline — `run_model.py` (Tutorial)

This guide explains **what the script does, in what order, and why**, and shows **how to run it from the CLI** for training and evaluation.

---

1) What this script is for

Train and/or evaluate **Random Forest** models on pre-made CSV folds, supporting:

- **Model types:** regression (`reg`), binary classification (`bin`), multi-class classification (`mclass`)
- **Repeated K-fold** cross-validation (e.g., 5 folds × 5 repeats)
- **Multiple datasets** indexed by **scramble fraction** (e.g., `0.00`, `0.25`, `1.00`)

> **Note:** The script **does not create folds**. It expects CSVs already split and named in a specific pattern (see §4).

---

2) Input data expectations

Each CSV must include:

- An **ID column** (default `sequence`, settable via `--ref_id_col`)
- A **label column** (default `label`, settable via `--ref_label_col`)
- **Feature columns**: all remaining columns are treated as numeric features.

The loader casts features to `float32`. Labels are cast to `int` for `bin`/`mclass`, else kept numeric for `reg`.

---

3) Directory & filename conventions

- `--data_dir`: where CSVs live (default: `Data/`)
- `--model_dir`: where models are saved/loaded (default: `Model/`)
- `--prefix`: dataset prefix in filenames (default: `gbsa`)
- `--model_type`: `reg` | `bin` | `mclass`
- `--scramble_fractions`: one or more floats; used only in filenames

A scramble fraction `f` is formatted as `scr{frac_str}` where `frac_str = f"{f:.2f}".replace('.', 'p')`.

Expected CSV names:

- **Training fold:** `{prefix}_{model_type}_scr{frac_str}_trn_{repeat}_{fold}.csv`
- **Validation fold:** `{prefix}_{model_type}_scr{frac_str}_val_{repeat}_{fold}.csv`
- **Final test set:** `{prefix}_{model_type}_scr{frac_str}_tst_final.csv`

Saved models:

- `Model/rf_fold_{repeat}_{fold}_{model_type}_scr{frac_str}.pkl`

Per-fold predictions (training mode):

- `{output_file}_{model_type}_scr{frac_str}_rep{repeat}_fold{fold}.csv`

Aggregated predictions (training mode):

- `{output_file}_{model_type}_final_avg_scr{frac_str}.csv`

Average metrics CSVs:

- Training: `final_metrics_{model_type}_trn_scr{frac_str}.csv`
- Test: `final_metrics_{model_type}_tst_scr{frac_str}.csv`

Prediction CSV schema:

```
Label,Predicted,True
<ID>,<model_output>,<ground_truth>
```

---

4) Script flow — step by step (what & why)

4.1 Parse CLI args

Reads all hyperparameters, file/directory settings, CV layout, and mode selection (`--mode 0` train, `--mode 1` evaluate).

4.2 Mode selection

- **Mode 0: training** → iterate fractions → repeats → folds; for each fold: load train/val, train RF, save model, evaluate on val, log metrics, write fold predictions. Aggregate all folds/repeats and write average metrics + averaged predictions.
- **Mode 1: evaluation** → iterate fractions; for each fraction: load final test CSV; for every saved model (repeat×fold), evaluate on the test set; save all predictions and average metrics.

4.3 Data loading (`load_csv_data`)

- Verifies that the file exists and contains `--ref_id_col` and `--ref_label_col`.
- Extracts **X** (feature matrix), **y** (targets), and **ids** (ID list). Feature dtypes are coerced to `float32` for compactness and speed.

Why: Standardizes inputs for scikit-learn and keeps memory usage reasonable.

4.4 Build the Random Forest (`build_random_forest`)

Creates `or` with:

- `n_estimators`, `max_depth`, `max_features`, `min_samples_split`, `min_samples_leaf`, `random_state`

`max_depth='None'` becomes `None` (unlimited). `max_features` accepts strings (`'auto'`, `'sqrt'`, `'log2'`) or numeric values (fraction or count, depending on sklearn semantics).

Why: Encapsulates hyperparameter handling and model type selection.

4.5 Fit & save (training mode)

For each fold:

1. **Type cast labels** to `int` for `bin`/`mclass`.
2. **Fit** the RF on training features/labels.
3. **Save** model to `model_dir` via `joblib.dump`.

Why: Store per-fold models so they can be reused for test-time evaluation or ensembling.

4.6 Evaluate (`evaluate_model`)

Computes metrics and emits row-level predictions. Evaluation differs by model type:

- **Regression (``)**

- Predict continuous values.

- **Per-ID aggregation**: average predictions and targets across duplicate IDs.
- Metrics: **MSE**, **R^2**, **Pearson r**.

- Also derives binary metrics (**MCC**, **Accuracy**) by thresholding at `0.0` if `--data_scale log`, else `1.0`.

- **Binary (``)**

- Predict class labels (0/1).

- **Per-ID aggregation**: majority vote of predictions and of targets.

- Metrics: **MCC**, **Accuracy**.

- **Multi-class (``)**

- Predict class probabilities.

- **Per-ID aggregation**: sum probabilities across rows per ID, take `argmax` for final class; target by majority vote.
- Metrics: **MCC**, **Accuracy**.

Row-level output is a list of tuples `(ID, predicted, true)` used to write per-fold CSVs.

Why per-ID aggregation? Some datasets provide multiple rows per ID (e.g., multiple windows/features). Aggregation yields a single label/prediction per unique entity.

4.7 Aggregate across folds & repeats (training mode)

- **Metrics**: average available metrics over all folds and repeats (NaNs ignored).
- **Predictions**: group per ID and **average or majority-vote** depending on task; save to `{output_file}_{model_type}_final_avg_scr{frac_str}.csv`.

Why: Produces a stable estimate across CV runs and a single consolidated prediction per ID.

4.8 Evaluate all models on test (evaluation mode)

- For each fraction, load `..._tst_final.csv` once.
- For every saved fold model, predict on the test set and log metrics.
- Save **all** per-model predictions to `{output_file}_test_{model_type}_scr{frac_str}.csv` and write **averaged** test metrics to `final_metrics_{model_type}_tst_scr{frac_str}.csv`.

Why: Ensures test results reflect all trained models, not just one fold.

---

5) CLI usage — common recipes

5.1 Quick start: regression, 5×5 CV, no scrambling

```
python run_model.py \
  --mode 0 \
  --model_type reg \
  --prefix gbsa \
  --data_dir Data \
  --model_dir Model \
  --output_file preds \
  --kfold 5 \
  --num_repeats 5 \
  --scramble_fractions 0.0
```

Then evaluate on test:

```
python run_model.py \
  --mode 1 \
  --model_type reg \
  --prefix gbsa \
  --data_dir Data \
  --model_dir Model \
  --output_file preds \
  --kfold 5 \
```

```
  --num_repeats 5 \
  --scramble_fractions 0.0
```

5.2 Binary classification with custom RF hyperparams

```
python run_model.py \
  --mode 0 \
  --model_type bin \
  --prefix gbsa \
  --data_dir Data \
  --model_dir Model \
  --output_file preds_bin \
  --kfold 5 \
  --num_repeats 3 \
  --scramble_fractions 0.0 0.25 1.0 \
  --n_estimators 500 \
  --max_depth 12 \
  --max_features 0.5 \
  --min_samples_split 4 \
  --min_samples_leaf 2 \
  --random_state 1337
```

5.3 Multi-class with non-log scale (affects reg threshold only)

```
python run_model.py \
  --mode 1 \
  --model_type mclass \
  --prefix gbsa \
  --data_dir Data \
  --model_dir Model \
  --output_file preds_mclass \
  --kfold 5 \
  --num_repeats 5 \
  --scramble_fractions 0.0
```

> **Note:** `--data_scale` only alters the regression binary threshold (0.0 for `log`, 1.0 for `nonlog`). It has no effect for `bin` / `mclass`.

---

6) Preparing your CSVs (naming checklist)

For each **scramble fraction** `f` and for **each** `repeat` and `fold`:

- Training: `{prefix}_{model_type}_scr{f}_trn_{repeat}_{fold}.csv`
- Validation: `{prefix}_{model_type}_scr{f}_val_{repeat}_{fold}.csv`

For each **scramble fraction** `f`:

- Test: `{prefix}_{model_type}_scr{f}_tst_final.csv`

Each CSV must include `--ref_id_col` and `--ref_label_col`. All other columns are features.

> **Scramble fraction** is used **only** to select among multiple dataset variants. The script assumes these files were created upstream (e.g., label shuffling or feature scrambling).

---

7) Outputs you'll get

- **Per-fold prediction CSVs** (training mode) with `Label,Predicted,True`.
- **Aggregated predictions** across all folds/repeats (training mode).
- **Average metrics** CSV summarizing MSE/R^2/Pearson (reg) or MCC/Accuracy (bin/mclass).
- **Test-time predictions & metrics** (evaluation mode) across all saved models.

---

8) Tips, tuning & pitfalls

- **max_features:** Accepts strings (`auto`, `sqrt`, `log2`) or numbers. Numeric $\in (0,1]$ means a fraction of features per split; integer means an absolute count.
- **Class balance:** For `bin`/`mclass`, ensure folds are stratified upstream; RF handles imbalance but metrics can be misleading without good splits.
- **Random state:** Controls tree-wise randomness; for repeated K-fold, the **data splits** are determined by your CSVs, not by this seed.
- **NaNs / Inf:** Ensure no NaNs in features; scikit-learn RF does not support NaNs.
- **Large feature sets:** Consider reducing `max_features` and limiting `max_depth` to avoid overfitting and speed up training.

---

9) Reference: CLI arguments

```
--mode {0,1}                    # 0=train, 1=evaluate
--model_type {reg,bin,mclass}
--model_obj MODEL_OBJ           # descriptor (not used in filenames)
--data_scale {log,nonlog}       # affects reg thresholding of binary metrics
--kfold K                       # folds per repeat
--num_repeats R                 # repeated K-fold count
--model_dir DIR                 # where .pkl models go
--data_dir DIR                  # where CSVs live
--output_file PREFIX            # prefix for prediction/metric outputs
--ref_id_col COL                # ID column in CSV (default: sequence)
--ref_label_col COL             # label column in CSV (default: label)
--n_estimators N
--max_depth DEPTH|None
--max_features AUTO|SQRT|LOG2|FLOAT|INT
--min_samples_split N
--min_samples_leaf N
```

```
--random_state SEED
--prefix DATA_PREFIX            # dataset name stem in CSV filenames
--scramble_fractions F1 [F2 ...]
```

---

10) End-to-end example (train then evaluate)

```
# Train 5x5 CV regression models on three dataset variants
python run_model.py \
  --mode 0 --model_type reg --prefix gbsa \
  --data_dir Data --model_dir Model --output_file preds_reg \
  --kfold 5 --num_repeats 5 --scramble_fractions 0.0 0.25 1.0 \
  --n_estimators 300 --max_depth 20 --max_features 0.7 --random_state 42

# Evaluate all saved models on the corresponding test sets
python run_model.py \
  --mode 1 --model_type reg --prefix gbsa \
  --data_dir Data --model_dir Model --output_file preds_reg \
  --kfold 5 --num_repeats 5 --scramble_fractions 0.0 0.25 1.0
```

TL;DR

- **Mode 0 (train):** loads each train/val fold → trains RF → saves model → evaluates on val → aggregates metrics/predictions.
- **Mode 1 (eval):** loads test set → runs **every saved model** → aggregates predictions/metrics across folds and repeats.

You now have a reproducible RF pipeline with clear inputs/outputs and CLI recipes.