# Checkpoint 4 Guide: Modeling

## Modeling Principles

* **Start simple before going complex**
  + Baselines (dummy, linear, simple trees) should be your reference point.
  + Complex models are only justified if they provide clear improvements in cross-validated KPIs.
* **Reproducibility**
  + All modeling must be encapsulated in reproducible pipelines (no loose cells with one-off preprocessing).
* **Avoid soup models**
  + Don’t throw in every algorithm. Each model type should be chosen for a defensible reason (interpretability, non-linear interactions, temporal handling, etc.).

## Model Families to Consider

* **Linear Models**
  + Regression: [LinearRegression](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html)
  + Classification: [LogisticRegression](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html)
* **Regularized Linear Models**
  + [Ridge](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html), [Lasso](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html), [ElasticNet](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ElasticNet.html)
* **Tree-Based Models**
  + [DecisionTree\*](https://scikit-learn.org/stable/modules/tree.html), [RandomForest\*](https://scikit-learn.org/stable/modules/ensemble.html#forest), [GradientBoosting\*](https://scikit-learn.org/stable/modules/ensemble.html#gradient-boosting)
  + Consider [XGBoost](https://xgboost.readthedocs.io/en/stable/), [LightGBM](https://lightgbm.readthedocs.io/), [CatBoost](https://catboost.ai/) for stronger ensembles.
* **Support Vector Machines**
  + [SVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html), [SVR](https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html)
* **Neural Networks (optional, advanced)**
  + [MLPClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html), [MLPRegressor](https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html)
  + Only pursue if you can justify the added complexity.

## Hyperparameter Tuning

* **Always tie to CV strategy**
  + Grid/Random search or Bayesian optimization must respect your split logic (temporal, group, etc.).
  + Hyperparameter tuning should usually be done in a nested cross-validation step to avoid "over-fitting the split"
* **Tools**
  + [GridSearchCV](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html)
  + [RandomizedSearchCV](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html)
  + [Optuna](https://optuna.org/) for more efficient search.

## Model Comparison and Selection

* **Use the KPIs defined earlier**
  + Compare models on *primary KPI* first, then look at secondary KPIs (precision/recall trade-offs, fairness, calibration, cost).
* **Fair comparison**
  + All models must be trained and evaluated on the same splits.
  + No cherry-picking folds or metrics.
* **Interpretability**
  + When possible, include interpretability diagnostics (coefficients, feature importance, partial dependence, SHAP).
  + Tools: [eli5](https://eli5.readthedocs.io/), [shap](https://shap.readthedocs.io/), [sklearn.inspection](https://scikit-learn.org/stable/modules/inspection.html).

## Iteration and Feedback Loops

* **Modeling is iterative**
  + Expect to fit, evaluate, discard, and revise. Modeling is rarely “one and done.”
* **Feedback into feature work**
  + Poor performance or odd patterns often point back to feature issues:
    - If all models perform equally badly → revisit EDA (maybe the target isn’t learnable with current features).
    - If tree-based models outperform linear by a wide margin → investigate non-linear relationships in features.
    - If models are unstable across folds → revisit feature selection, leakage checks, or group definitions.
* **Workflow**
  1. Fit a model on current pipeline features.
  2. Examine errors and diagnostic plots.
  3. Ask: are errors random, or do they cluster by subgroup/time/etc.?
  4. If clustering is visible, return to **Checkpoint 2 (EDA & Feature Engineering)** to design better features or prune irrelevant ones.
  5. Refit and evaluate again.
* **Mindset**
  + Treat modeling and feature design as a loop. The “final model” emerges after several rounds of refinement, not the first training run.

## Deliverables

* **Written / Conceptual**
  + A short document (modeling\_plan.md) describing:
    - Which model families were tried and why.
    - How they align with project goals.
    - Justification for the final chosen model, tied explicitly to KPIs and evaluation strategy.
    - Notes on what didn’t work and why those approaches were discarded.
* **Code / Repo Artifacts**
  + **notebooks/modeling\_baselines.ipynb**:
    - Implements trivial and simple baselines (Dummy, Linear, Logistic, basic trees).
    - Reports CV performance with chosen KPIs.
  + **notebooks/modeling\_experiments.ipynb**:
    - Trains more complex models (ensembles, regularized, etc.).
    - Includes performance tables/plots across folds.
    - Documents failed experiments and iteration decisions.
  + **src/models/**:
    - Python scripts defining reusable training functions and model wrappers.
  + **src/models/tune.py**:
    - Hyperparameter search logic, tied to the correct CV splitter.
  + **results/model\_comparison.csv**:
    - Tabular record of model family, hyperparameters, and KPI scores.
  + **results/interpretability/**:
    - Feature importance plots, coefficient tables, or SHAP visualizations.
  + **Serialized models** in artifacts/ (via joblib.dump), with matching environment spec so they can be reloaded.
  + **Tests in tests/test\_models.py**:
    - Ensure training scripts run end-to-end, models fit without errors, and CV returns consistent shapes.