Buckling Strength ML – Implementation Plan (copy/paste for VSCode AI coding agent)

> Goal: build a **standalone notebook** (VSCode → later runnable in **Google Colab**) that cleans the Excel dataset, defines a reliable target, trains **two model families** (regularized linear + tree-based), selects the best per family using **grouped-by-geometry evaluation**, and produces publication-quality **visualizations + metrics** aligned to a Chapter-6 style write-up.

---

0) Locked requirements

Target

- Primary target column: `Ultimate  Buckiling Strength (Mtest)`
- Missing target fill rule: `Mtest_filled = Mtest` then fill missing from `Ultimate Buckiling Strength (MFEM)`
- `MFEM` is **never** used as an input feature.
- After fill, coerce target to numeric; **drop** any rows still non-numeric/missing (e.g., `NC`).

Columns to drop / exclude

- Exclude columns **A–F** and **AD–AG** from modeling features (keep `JOURNAL TITLE` and `SPECIMEN` for diagnostics only).
- Drop column `q` (mostly missing).
- Drop the "units row" where `SL. No == "Units"`.

Model families to compare

1. **Regularized linear**: Ridge, Lasso, ElasticNet
2. **Tree-based**: RandomForest, HistGradientBoosting (and optionally XGBoost if available)

Evaluation strategy

- Primary grouping to prevent leakage: **geometry_signature** (hash of cleaned feature values)
- Model selection: **GroupKFold CV** using `geometry_signature`
- Final "exam" evaluation: **GroupShuffleSplit holdout test set** (20% of geometry signatures)

Selection rule

- Primary metric to minimize: **MAE**
- Constraint: **Pearson correlation ≥ 0.90** (preferably for best model in each family)

Duplicate handling

- Aggregate duplicates **within a journal**:
- Group key: `(JOURNAL TITLE, geometry_signature)`
- Aggregations: `target_mean`, `target_std`, `replicate_count`
- Use `target_mean` for modeling

Reviewer/validator mindset (must-follow)

The agent must treat every data-cleaning rule as a **hypothesis** and verify it against the dataset:

- After each cleaning step, print a **before/after summary** (row count, column count, missingness, unique categories).
- Produce **conditional missingness tables** for key hole-related columns by `HOLE SHAPE` and by `Number Of Holes`.
- Flag inconsistencies (e.g., `no hole` rows that contain hole-location values, `Number Of Holes==1` rows with non-null `Sx`).
- Prefer adding **flags** + logging over silently changing values.

Notebook-only constraint

- Keep helper functions **inside the notebook** (no mandatory external `.py` files) so the notebook is **Colab-standalone**.

---

1) Project directory template (simple + notebook-first)

```
buckling-ml/
  data/
    raw/
      data.xlsx
    interim/
      cleaned.parquet
      aggregated.parquet
  notebooks/
    00_end_to_end_modeling.ipynb
  outputs/
    figures/
    tables/
    models/
  README.md
  requirements.txt
```

Notes:

- You can do scratch work inside `00_end_to_end_modeling.ipynb` initially.
- If the notebook grows too large later, we can refactor—but **not required now**.

---

2) Notebook outline mapped to Chapter 6 write-up

Use markdown headings exactly like this (agent should generate these sections):

- **6.1 Introduction**
- **6.2 Motivation for Machine Learning-Based Prediction**
- **6.3 Input Features and Output Parameters**

- **6.4 Data Preprocessing and Feature Selection**
- **6.5 Machine Learning Model Development**
- **6.6 Model Training, Testing, and Performance Metrics**
- **6.7 Summary**

Each section includes code + plots + brief interpretation.

---

3) Step-by-step implementation plan (the agent should implement in this order)

Cell group A — Setup (Colab-friendly)

1. Imports: pandas, numpy, matplotlib, sklearn (pip install optional if missing).
2. Set `RANDOM_SEED`.
3. Define `DATA_PATH`:
4. VSCode: `../data/raw/data.xlsx`
5. Colab: assume user uploads to `/content/data.xlsx` and sets `DATA_PATH` accordingly.

Deliverable: a single cell that runs in both environments with minimal edits.

Cell group B — Load Excel + remove units row

1. Read Excel sheet: `Swati (Beam with hole)`.
2. Drop units row:
3. Identify row where `SL. No` equals `"Units"` (string match) and drop it.
4. **Normalize column names** (important; the raw file contains leading/trailing spaces):
5. `df.columns = df.columns.str.strip()`
6. Optionally rename to stable snake_case equivalents (keep a mapping table in the notebook).
7. Normalize categorical text columns:
8. `.astype(str).str.strip()`
9. `.str.lower()`
10. Apply explicit mappings for `HOLE SHAPE` to ensure consistent labels: `circle`, `square`, `rectangle`, `elongated`, `no_hole`.
11. Create a **data quality snapshot**:
12. shape, dtypes, missingness%, unique counts per categorical

Diagnostics to print:

- shape before/after
- list of columns after stripping
- head(3)
- missingness top-10 columns

Cell group C — Define the modeling target

1. Extract `y_mtest`, `y_mfem`.
2. Create `y = y_mtest.fillna(y_mfem)`.

3. Create flag `y_from_fem = y_mtest.isna() & y_mfem.notna()`.
4. Coerce `y` to numeric with `errors='coerce'`.
5. Drop rows where `y` is NA.

Diagnostics to show:

- Missing rates of Mtest, MFEM, filled target
- Count of targets filled from MFEM
- Count of rows dropped due to non-numeric target

Cell group D — Drop/exclude columns and define feature set

1. Keep metadata columns for diagnostics/splitting only:
2. `JOURNAL TITLE`, `SPECIMEN`
3. Drop non-feature columns (use names; do not rely on Excel letters):
4. identifiers/metadata/constants: `SL. No`, `SHAPE CODE`, `Boundary condition`, `Loading condition`
5. guideline/outcome fields: `Failure Mode`, `AISI`, `Eurocode 3`, `AS/NZS`
6. drop `q`
7. do **not** include targets: `Ultimate  Buckiling Strength (Mtest)`, `Ultimate  Buckiling Strength (MFEM)`
8. Define:
9. `X_full` = remaining feature columns
10. `meta` = journal/specimen

Diagnostics to show:

- list of remaining feature columns
- numeric vs categorical split counts
- check for any constant columns (nunique==1) and drop them from features

---

Cell group E — Geometry signature (group key) + structural-missingness handling

Key point: hole-related columns have **structural NA** (not applicable). We need stable grouping + sensible modeling.

E1) Verify hole-shape patterns (do not assume)

The agent must compute and display a table of missingness and min/max by `HOLE SHAPE` for:

- `l hole`, `h hole`, `r hole`
- `Number Of Holes in Channel`
- `Distance from top to centre of first hole Y`
- `Distance from left  to centre of first hole X`
- `Centre to centre distance between adjesent holes in X direction  Sx`

Expected patterns to confirm (from dataset inspection):

- **circle**: `l hole`, `h hole` are NA; `r hole` present

- **rectangle/square**: `l hole`, `h hole` present; `r hole` NA
- **elongated**: `l hole`, `h hole` present **and** `r hole` present (non-zero)
- **no_hole**: all hole dims/locations should be NA (if not, flag as inconsistency)

E2) Consistency cleaning rules (apply only after reporting)

- Create `has_hole = (HOLE SHAPE != 'no_hole')`.
- If `has_hole==0` and any hole-dimension/location fields are non-null:
- create flag `inconsistent_no_hole_metadata = 1`
- set those hole fields to NA **only after logging counts**
- If `Number Of Holes in Channel <= 1` and `Sx` is non-null:
- create flag `inconsistent_sx_for_single_hole = 1`
- set `Sx` to NA (recommended) after logging

Rationale: these values are structurally not applicable; leaving them can confuse learning and signature grouping.

E3) Geometry signature creation

1. Identify feature columns (`feature_cols`) excluding target + metadata.
2. For **signature creation only**:
3. Create a temporary copy of X
4. Fill NaNs with a stable sentinel (e.g., `"__NA__"`) so identical patterns hash identically.
5. Build `geometry_signature` as a stable hash of row values (e.g., using `pd.util.hash_pandas_object`).
6. Define aggregation key:
7. `(JOURNAL TITLE, geometry_signature)`

Diagnostics to show:

- number of unique geometry signatures
- count of duplicate groups (size>1)
- counts of inconsistency flags created above

Cell group F — Aggregate duplicates within journal

1. Group by `(JOURNAL TITLE, geometry_signature)`.
2. Aggregate:
3. `target_mean = mean(y)`
4. `target_std = std(y)`
5. `replicate_count = size`
6. for feature columns: take `first`
7. Create a **data-quality summary table** with counts of:
8. rows dropped (units row, non-numeric targets)
9. `inconsistent_no_hole_metadata`
10. `inconsistent_sx_for_single_hole`
11. duplicate groups and their sizes
12. Define final modeling dataframe:
13. `df_model` with columns: meta + features + `geometry_signature` + `target_mean` + `replicate_count` (+ flags)

14. Use `target_mean` as the final `y_model`.

Diagnostics to show:

- how many rows before vs after aggregation
- distribution of replicate_count
- head() of the quality summary table

---

4) Visualizations to include (pre-model)

Preprocessing visuals (must-have)

1. **Missingness bar chart**: % missing per feature (before imputation).
2. **Conditional missingness heatmap/table**:
3. missingness of hole-related fields grouped by `HOLE SHAPE`
4. missingness of `Sx` grouped by `Number Of Holes in Channel`
5. **Target distribution**:
6. histogram + boxplot for `target_mean`
7. compare `y_from_fem` vs not (two histograms or overlay)
8. **Target vs key categoricals** (box/violin):
9. `HOLE SHAPE` vs target
10. **Scatter plots**:
11. choose top 3–5 numeric features (by absolute correlation with target) and plot vs target
12. **Replicate noise**:
13. histogram of `replicate_count`
14. scatter of `target_std` vs `target_mean` (only where replicate_count>1)

Metadata-only diagnostics (recommended; not used as features)

These help verify distribution shift and whether the dataset is dominated by one source. 7. **Rows per journal** bar chart (`JOURNAL TITLE` count). 8. **Target by journal** boxplot (warn if some journals have tiny sample sizes). 9. **Hole-shape mix by journal** stacked bar chart. 10. Optional: **Specimen length distribution** (e.g., `L`) by journal (boxplot) to see source differences.

Counter-argument (why we keep it limited):

- We avoid turning this into a full “source bias” study; the goal is actionable modeling. The above metadata plots are lightweight and high-value for sanity checking.

All plots must have:

- title, axis labels, readable font size
- saved to `outputs/figures/`

---

5) Train/test split and CV

Grouped holdout

- Use `GroupShuffleSplit(test_size=0.20, random_state=RANDOM_SEED)`
- Groups = `geometry_signature`
- Split `df_model` into train_holdout and test_holdout

Grouped CV on training portion

- Use `GroupKFold(n_splits=5)`
- Groups = `geometry_signature`

Why both:

- CV selects models/hyperparams.
- Holdout provides a final unbiased comparison number.

---

6) Preprocessing pipelines (sklearn)

Column typing

- Identify:
- numeric columns: dtype number (after coercion)
- categorical columns: object/string

Missingness handling (important)

- Many hole-related fields are **structurally missing** depending on `HOLE SHAPE` / `Number Of Holes`.
- Use either:
- explicit flags created earlier (`inconsistent_*`, `has_hole`), and/or
- `SimpleImputer(..., add_indicator=True)` to add missingness indicators automatically.

Transformers

- Numeric: `SimpleImputer(strategy='median', add_indicator=True)`
- Categorical: `SimpleImputer(strategy='most_frequent')` then `OneHotEncoder(handle_unknown='ignore')`

Pipelines

- Linear models:
- add `StandardScaler(with_mean=False)` after encoding (or scale numeric before OHE; whichever is cleaner)
- Tree models:
- no scaling needed

(Agent should implement both pipelines cleanly with `ColumnTransformer`.)

---

7) Model families + tuning grids

Linear family (optimize MAE)

- Ridge: `alpha` grid (log-spaced, e.g., 1e-4 … 1e3)
- Lasso: `alpha` grid
- ElasticNet: `alpha` grid + `l1_ratio` grid

Tree family

- RandomForestRegressor:
- n_estimators, max_depth, min_samples_leaf
- HistGradientBoostingRegressor:
- max_depth / max_leaf_nodes, learning_rate
- Optional: XGBoost
- only if available; otherwise skip to keep notebook dependency-light

Search strategy:

- Start with **small grids** first (fast), then expand around best region.

---

8) Metrics + selection logic

Compute for both:

- Out-of-fold (OOF) predictions from GroupKFold
- Holdout test predictions

Metrics to report:

- MAE (primary)
- RMSE
- R^2
- Pearson correlation

Selection per family:

1. Filter candidates with `Pearson_OOF >= 0.90`.
2. Pick the one with lowest `MAE_OOF`.
3. If none pass, pick lowest `MAE_OOF` and explicitly flag correlation shortfall.

Create a single `results_df` table with:

- model_family, model_name, params
- OOF metrics (mean ± std across folds)
- holdout metrics

Save to `outputs/tables/metrics.csv`.

---

9) Visualizations to include (post-model)

For **best linear** and **best tree** models, generate:

1. **Observed vs Predicted** (OOF)
2. **Observed vs Predicted** (holdout)
3. **Residuals vs Predicted** (holdout)
4. **Residual histogram** (holdout)
5. **Error by HOLE SHAPE** (holdout): MAE per category
6. **Error by journal** (holdout): MAE per `JOURNAL TITLE` (include sample sizes; skip journals with extremely tiny n if plot becomes unreadable)
7. Optional: **Observed vs Predicted colored by journal** (holdout) to visually spot source-specific bias
8. Interpretability:
9. Linear: top 20 absolute coefficients
10. Tree: top 20 feature importances (note: one-hot expands features; handle names cleanly)

Save all figures to `outputs/figures/`.

---

10) Final comparison + refit on full data

Model comparison

- Compare best linear vs best tree using **holdout metrics**.
- Declare the winner based on:
- lowest MAE (primary)
- Pearson $\geq$ 0.90 requirement

Refit for deployment

- Refit the **chosen final model** on the **entire aggregated dataset** (`df_model`).
- Save final model artifact to `outputs/models/final_model.joblib`.

Important reporting rule:

- Final refit is for deployment.
- Performance claims must come from OOF + holdout.

---

11) Final notebook narrative (Chapter 6 write-up guidance)

The agent should add short markdown interpretations:

- 6.3: list features retained; what is categorical vs numeric; what is target.
- 6.4: explain missingness patterns (structural NA for hole dims) and how imputation + one-hot handles it.
- 6.5: briefly describe the two model families and why they're appropriate.
- 6.6: present results table + plots, explain MAE vs Pearson tradeoff.
- 6.7: summarize winner + limitations (e.g., replicate noise, journal bias).

12) Phased execution protocol (must-follow)

You (the coding agent) must **not** implement the whole notebook end-to-end in one pass. Instead, work in **phases**. For each phase:

1. Output the **exact code cells** to add/run.
2. Specify the **expected outputs** (shapes, key tables, plots).
3. Provide a **validation checklist** (“if X is not true, stop and debug”).
4. **STOP** and wait for the user to confirm (“continue”) before proceeding.

Phase 0 — Environment + file load sanity

- Create folders (`outputs/figures`, `outputs/tables`, `outputs/models`).
- Read Excel sheet, strip column names, drop units row.
- Print: shape, columns, dtypes summary, top-10 missingness. **Gate:** user confirms columns/shape look correct.

Phase 1 — Target construction and target QC

- Build filled target `y = Mtest.fillna(MFEM)` + `y_from_fem` flag.
- Coerce numeric; drop non-numeric target rows.
- Print: missingness rates, fill counts, rows dropped. **Gate:** user confirms target definition counts.

Phase 2 — Feature set definition + column pruning

- Drop excluded columns by name (A–F & AD–AG equivalents) + drop `q`.
- Identify numeric vs categorical; drop constants.
- Print: final feature column list + counts. **Gate:** user confirms features look reasonable.

Phase 3 — Structural missingness validation + inconsistency flags

- Produce conditional missingness tables by `HOLE SHAPE` and by `Number Of Holes`.
- Create flags and (only after reporting) apply recommended consistency NA fixes:
- `no_hole` rows with hole metadata
- single-hole rows with `Sx` present
- Print: counts of flagged inconsistencies before/after. **Gate:** user confirms we should apply the fixes.

Phase 4 — Geometry signature + duplicates + within-journal aggregation

- Create `geometry_signature` with stable hashing (using NA sentinel for signature only).
- Quantify duplicates (group sizes).
- Aggregate by `(JOURNAL TITLE, geometry_signature)` into `df_model`.
- Print: rows before/after aggregation, replicate_count distribution. **Gate:** user confirms aggregation impact is acceptable.

Phase 5 — Pre-model EDA visualizations

- Generate and save the required EDA plots (missingness, conditional missingness, target dist, target vs hole-shape, scatter vs top numeric, replicate noise, rows per journal, target by journal, hole-shape mix by journal).

- Display key plots inline. **Gate:** user confirms plots look sensible.

Phase 6 — Grouped holdout + GroupKFold setup (leakage checks)

- Create grouped holdout (20%) by `geometry_signature`.
- Create GroupKFold on training.
- Print: overlap checks (no group overlap), counts of groups per split. **Gate:** user confirms split logic.

Phase 7 — Preprocessing pipelines

- Build sklearn `ColumnTransformer` pipelines for linear and tree models.
- Run a tiny smoke-test fit/predict on one fold.
- Print: pipeline feature counts, successful run. **Gate:** user confirms preprocessing works.

Phase 8 — Model training (small grids first)

- Train/evaluate linear family with small grids (fast).
- Train/evaluate tree family with small grids.
- Produce `results_df` with OOF + holdout metrics. **Gate:** user chooses whether to expand grids around best candidates.

Phase 9 — Final per-family selection + post-model visuals

- Apply selection rule (min MAE subject to Pearson$\geq$0.90).
- Generate post-model plots (OOF + holdout obs-vs-pred, residual plots, MAE by hole-shape, MAE by journal, interpretability plots). **Gate:** user confirms final winners.

Phase 10 — Final refit + artifact export

- Refit chosen final model on full aggregated dataset.
- Save model + metrics + cleaned datasets.
- End with a short “how to run in Colab” note.

---

13) “Agent prompt” footer (operational instructions)

When generating the notebook:

- Prefer clear, modular code cells.
- Use functions inside the notebook for repeated tasks:
- `make_geometry_signature()`
- `build_preprocessor()`
- `evaluate_model_cv()`
- `plot_diagnostics()`
- Print intermediate checks after each major step.
- Save outputs (figures/tables/models) to the specified folders.
- **STOP after each phase** and ask the user to confirm before moving on.

End state:

- One notebook (`00_end_to_end_modeling.ipynb`) runs end-to-end and produces:

- cleaned+aggregated dataset files
- metrics table
- all required plots
- final refit model artifact