CW1 Regression Report

k21190308

1 Exploratory Data Analysis

The dataset has 10,000 training samples and 1,000 test samples with 30 features and no missing values. The target (`outcome`) ranges from about -45 to $+40$ with a mean around -5. There are 10 diamond features (carat, cut, color, clarity, depth, table, price, x, y, z), and 20 synthetic features split into uniform and normal distributions.

I started by looking at correlations with the outcome. `depth` stood out immediately with $r = -0.41$, much stronger than anything else. After that, `b3` ($r = 0.23$), `b1` ($r = 0.17$), and `a1` ($r = 0.15$) had some signal. Most other features were basically noise with near-zero correlation. I also ran a quick Random Forest to check feature importances, which confirmed depth as the dominant feature (importance = 0.35).

One surprising finding was that the diamond commercial features (carat, price, x, y, z) barely predicted the outcome at all, even though they're highly correlated with each other.

Boxplots of the three categorical features (cut, color, clarity) showed very little variation in outcome across categories, so they don't contribute much.

I found 5 rows with bad values in x, y, z: two diamonds with all zeros, two with $z = 0$, and one where $y = 58.9$ (probably meant to be 5.89). Since I'm using tree-based models which handle outliers fine, I left them in.

2 Model Selection

I compared three models with 5-fold cross-validation:

Model	CV R^2
Linear Regression	0.278 ± 0.032
Random Forest	0.453 ± 0.011
Hist. Gradient Boosting	0.436 ± 0.018

Linear regression only picks up the linear trend with depth and doesn't do well. Both tree models are much better since they can capture non-linear patterns. I decided to tune both RF and HGB and then combine them into an ensemble, since they tend to make different kinds of errors — RF averages many independent trees while boosting builds them sequentially.

I used ordinal encoding for the categoricals instead of one-hot because trees can handle ordinal splits directly, and one-hot would add 15 extra sparse columns for no real benefit.

3 Model Training and Evaluation

Tuning

For **Random Forest**, I searched over `max_features` and `min_samples_leaf`. The best was `max_features`$=$ 0.4, `leaf`$=5$ with $R^2 = 0.460$. Using 40% of features per split worked better than the default $\sqrt{p}$ ($\approx$18%), probably because with so many noise features, trees need to see more columns to find the useful ones.

For **Hist Gradient Boosting**, I tuned iterations, depth, and learning rate. Best was `iter`$= 1000$, `depth`$= 3$, `lr`$= 0.03$, `leaf`$= 20$ with $R^2 = 0.470$. Keeping trees shallow and the learning rate low helped avoid overfitting.

Ensemble

My final model averages 5 sub-models: 3 HGB and 2 RF, each with slightly different hyperparameters and random seeds. Out-of-fold evaluation:

Model	OOF R^2
hgb1 (iter=1000, d=3, lr=0.03)	0.466
hgb2 (iter=1500, d=3, lr=0.02)	0.466
hgb3 (iter=800, d=4, lr=0.03)	0.464
rf1 (feat=0.5, leaf=1)	0.458
rf2 (feat=0.4, leaf=1)	0.459
Ensemble avg	**0.470**

The ensemble beats any single model by a small margin. I used simple averaging rather than learned weights since it's less likely to overfit with this sample size.

4 Code Supplement

Code is at: `https://github.com/abdelrahman-almatrooshi/ML_CW1`

- `notebooks/CW1_notebook.ipynb` — full pipeline
- `src/train_model.py` — standalone script
- `evaluate/CW1_eval_script.py` — provided baseline
- `outputs/CW1_submission_k21190308.csv` — predictions

To reproduce: put the CSVs in `data/` and run `python src/train_model.py`.