COMP4204 Group Project

Feature Engineering and Foundation Model Stacking

WANG Yuqi[1] **DU Haoqian**[1] **DU Haoyang**[1]

[1]Department of Computing, The Hong Kong Polytechnic University

Abstract

Our group scored 0.10939 RMSLE (top 10 excluding leaderboard probe) on Kaggle's House Price Prediction via multi-layer stacking and Out-Of-Fold (OOF) bagging of deep learning based tabular foundation models. EDA reveals long tailed distributions and structured missingness, prompting us to explore Yeo-Johnson transformation, KNN Impute, as well as manual feature engineering that leverages human priors. We further performed comprehensive ablations that isolate each component's marginal contribution. Results reveal: tabular foundation model (e.g., TabPFNv2) are clearly superior to tree ensembles; while hand-crafted impute strategies and features contribute to performance gains in traditional tree-based ensembles, tabular foundation models work best without any preprocessing.

1 Introduction

Advancements in deep learning have transformed the fields of vision and language. Despite this, tree-based gradient boosting (XGBoost, LightGBM, CatBoost, etc.) [1], [2], [3] has continued to dominate tabular data for years. It has also been widely accepted that sample scarcity in many tabular datasets mandates encoding prior knowledge via manual feature engineering. Until recently, deep learning based tabular foundation models (TFMs) emerged and challenged these conventional wisdoms. These models were pretrained on a massive amount of synthetic tabular datasets, allowing them to generalize to any unseen structured data via in-context learning [4], [5], [6], [7]. The Kaggle *House Price Prediction Challenge* provides exactly the valuable opportunity to empirically answer:

- Do foundation models obsolete classical methods?
- How should feature engineering be done in the age of pre-training?
- How should foundation models be integrated with established pipelines?

We approach this through systematic experimentation. We start with establishing our baselines, including linear and gradient boosted models. Next, extensive exploratory data analysis (EDA) of the dataset is performed; through insights gained from these analyses, we perform feature engineering and tuning of these classical methods in a best-effort manner, using techniques such as bagging, stacking, and weighted ensembles. Finally, results are compared to those of deep learning foundation models. Our final model scores 0.10939 Root Mean Square Logarithmic Error (RMSLE) on the Kaggle official leaderboard, which is **ranked top 10 excluding probed solutions** (i.e., solutions that hacked the test set and directly submitted test labels).

To ensure that both sides are being pushed to their absolute limits, we perform a grid search and show through ablation the optimality of our final design choices. These ablations also provide valuable

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

empirical insights for answering the above three questions (see Section 8).[1]

Project Highlights

1. Highly competitive leaderboard RMSLE score of 0.10939 (***Section C***)
2. Diverse methods: TFMs, Zero-shot HPO, Dynamic Stacking, etc (***Section 5***)
3. Comprehensive EDA to guide feature engineering (***Section 4***, ***Section A***)
4. Ablation to show our design process and rationales (***Section 6***)
5. Model interpretability via SHAP values and dependency plots (***Section 7***)
6. Discussions that challenge classical wisdoms in Machine Learning (***Section 8***)

Answers to the two questions *"Which feature engineering is most useful?"* and *"Describe the best submission improvement obtained"* can be found in the Appendix. (***Section D***)

2 Background

Tabular Learning Landscape Structured data has been dominated by gradient boosting decision trees (GBDT). For years, methods like XGBoost, LightGBM, and CatBoost have consistently outperformed carefully tuned deep learning models out-of-the-box. Common belief attributes such divergence to the lack of inductive biases in neural networks, when trees naturally handle mixed feature types, sparse or missing values, and non-smooth decision boundaries common in tabular data. Another conventional wisdom is that a limited sample size in a typical tabular dataset necessitates strong human priors to manually encode raw features into domain-meaningful interactions that capture nonlinear relationships through feature engineering. However, this defeats the advantage of automatic feature extraction in neural networks, as well as the end-to-end training philosophy.

Tabular Foundation Models (TFMs) Recent works in deep learning challenge this classical paradigm through large-scale pretraining. TabPFN [5], short for Tabular Prior-Data Fitted Network, address the previous limitations of deep learning based approaches by pretraining on large synthetic distributions of tabular tasks. They are training-free, meta-learned and perform inference via in-context learning. Later work, such as TabICL [4] and Mitra [6], extends and scales up this method through retrieval-augmented and metric-learning approaches. TABM [7] uses self-supervised pre-training via masked autoencoders (MAE). These models have shown the ability to transfer knowledge across distinct tabular tasks without the need for parameter updates, potentially obsoleting both manual feature engineering and classical methods.

TabPFN TabPFN is a Prior-Data Fitted Network that amortizes the often intractable Bayesian prediction for tabular datasets by learning an explicit one-step transformation from labelled training data to test data class probabilities. Given data $D = \{(x_i, y_i)\}_{i=1}^{n}$, a query x', the Bayesian target is the posterior predictive

$$p(y'|x', D) = \int p(y'|x'\varphi)p(D|\varphi)p(\varphi)\,\mathrm{d}f\,\mathrm{d}\theta \tag{1}$$

Here, φ ranges over **all data-generating mechanisms**. This integral essentially averages prediction over all plausible sets of underlying rules and mechanisms, weighted by how well each explains the observed data D. In practice, TabPFN approximates this intractable integral with a permutation-invariant Transformer trained across many synthetic tabular datasets, too, generated from such mechanisms, so that its one-shot an output approximates the integral above [5]. For regression, the same amortized posterior predictive Transformer is used, but with continuous labels and output head.

Ensemble Methods Stacking and bagging are fundamental methods for achieving competitive predictive power in tabular datasets. Bagging reduces variance through bootstrap aggregation of numerous models of the same type, while stacking learns meta-models over the base models' Out-of-fold (OOF) predictions (e.g., the arithmetic average of the bagged models' output across many shuffles). OOF predictions are crucial when it comes to stacking, as they avoid information

[1] Ablations were performed **post-hoc** to prevent human bias and overfitting to the test dataset; we avoid submitting to the leaderboard during model design, and evaluation is done purely through a OOF validation or holdout datasets.

leakage and prevent learning from trivial, overfitted meta-features. Multi-layer stacking extends this hierarchically, though risks overfitting if not carefully regularized, which we will discuss in detail in upcoming sections.

House Price Prediction The Kaggle House Price Prediction competition provides 1,460 training examples, with 79 feature columns, capturing house characteristics (square footage, quality rating, year built, etc.) and neighborhood attributes. The dataset contains typical challenges: skewness, structured missingness, collinearity, and feature interactions. The scale of the dataset, combined with these characteristics, makes it a perfect toy dataset for empirical comparison between classical methods and the newer foundation models.

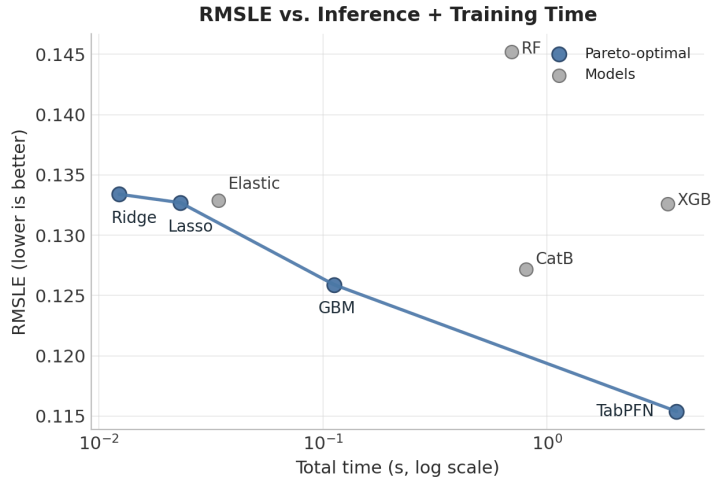
3 Baselines

Experiment Setup Each baseline was tested on the Kaggle dataset using out-of-box implementations with default hyperparameters and minimal preprocessing. Each baselines were experimented under near-identical settings, except for neural network models, for which we scaled and normalized the input features to avoid catastrophic failures. Numerous metrics, including RMSLE, training, and inference time, were adopted. Best performing models should sit at the Pareto frontier of predictive power and computational cost.

Baseline Methods Table 1 shows various models, classical and tree-based models were chosen as baseline. For classical models, we chose three linear models (Ridge, Lasso, ElasticNet) along with Support Vector Regression (**SVR**) [8]. For tree-based models, LightGBM (**GBM**) [3], CatBoost (**CatB**) [2] and XGBoost (**XGB**) [1] were chosen as strong baselines, along with Random Forest (**RF**) [9]. TabPFN was also included in our comparison to demonstrate its exceptional performance.

Table 1: Baseline Comparison Table

Metrics	Lasso	Ridge	Elastic	SVR	RF	GBM	XGB	CatB	MLP	TabPFN
RMSLE	0.1327	0.1334	0.1329	0.4144	0.1452	0.1259	0.1326	0.1272	0.9931	**0.1154**
Training(s)	0.0212	**0.0071**	0.0292	0.1217	0.5724	0.1087	3.4362	0.8043	3.9896	2.3359
Inference(s)	**0.0019**	0.0052	0.0049	0.1877	0.1217	0.0034	0.0170	0.0022	0.0133	1.4493



The baseline comparison in Table 1 highlights the relative predictive power between linear and non-linear models. Linear model such as Lasso, Ridge, and ElasticNet exhibits poor predictive power but are computationally cheap. In contrast, tree-based methods, especially LightGBM, showed competitive predictive performance while maintaining reasonable inference time. Among the traditional machine learning baselines, LightGBM achieved Pareto optimality when it comes to performance-

efficiency tradeoff. TabPFN is the best-performing model amongst all, but suffers from slightly slower speed.

4 Feature Engineering

Motivated by our findings in Section A, we designed various feature engineering stages, applied one after the other, forming a pipeline. Earlier stages focus on basic transformations, such as imputing missing values via k-nearest-neighbor (KNN) [10], normalizing numerical distributions via Yeo-Johnson transformation [11] and zero centring. Later stages implement hand-crafted features and imputation strategies.

4.1 KNN Imputer

As shown in Table 5, the dataset contains a large amount of missingness. Intuitively, we decided to implement a KNN imputer that searches for nearest-neighbor samples and fills in missing values via distance-weighted aggregation. Let the data be a matrix $X \in \mathbb{R}^{n \times p}$ with numerical features $\mathcal{N}$ and categorical features $\mathcal{C}$. The imputer starts by normalizing all numerical columns $k \in \mathcal{N}$ to $[0, 1]$ via min-max scaling. Then, for two samples x_i and x_j, their distance is calculated as the sum of feature-wise distances $d_k(x_i, x_j)$. For any given feature k, the distance is calculated based on its data type:

$$d_k(x_i, x_j) = \begin{cases} |x'_{ik} - x'_{jk}| & \text{if } k \in \mathcal{N} \\ \mathbb{I}(x_{ik} \neq x_{jk}) & \text{if } k \in \mathcal{C} \end{cases} \tag{2}$$

Where $\mathbb{I}(\cdot)$ returns 1 if the condition is true and 0 otherwise. In essence, this formulation guarantees that each feature contributes a value of [0, 1], allowing us to calculate the total distance $D(x_i, x_j)$ as the simple sum of these feature-wise differences: $D(x_i, x_j) = \sum_{k=1}^{p} d_k(x_i, x_j)$. Finally, for each sample x_i that contains a missing value in a feature k', the imputer searches for K nearest neighbors from all samples where k' is observed and calculates their weighted average at k'.

4.2 Yeo-Johnson Transformation

Our EDA shows that many numerical features exhibit non-normality and heteroscedasticity. Yeo-Johnson power transformation is perfect for this, as it not only reduces feature skewness and stablises variance, but is also able to handle features with negative values (which Box-Cox transformation fails at). Through Yeo-Johnson, numerical features can be made to more closely approximate a Gaussian, hopefully benefiting downstream models. For a given feature vector y, the Yeo-Johnson transformation $f^{\lambda}(y)$ is a piecewise function parameterized by λ:

$$f^{\lambda}(y) = \begin{cases} \frac{(y+1)^{\lambda}-1}{\lambda} & \text{if } \lambda \neq 0, y \geq 0 \\ \ln(y+1) & \text{if } y = 0, y \geq 0 \\ -\frac{(-y+1)^{2-\lambda}-1}{2-\lambda} & \text{if } \lambda \neq 2, y < 0 \\ -\ln(-y+1) & \text{if } \lambda = 2, y < 0 \end{cases} \tag{3}$$

The optimal λ is not a set hyperparameter but determined independently for each feature via maximum likelihood estimation (MLE), which finds λ by maximizing the log-likelihood of the data under normality assumption post-transformation [11].

4.3 One-Hot Encoding

For categorical variables, we also tried using one-hot encoding (OHE). The one-hot encoding works by spreading a single categorical feature across many boolean columns. Concretely, given a single categorical feature with K unique categories, the feature is replaced with K new binary features. For a given sample i, if its value for the original feature is the j-th unique category, its j-th binary feature is set to 1, while the other $K - 1$ are set to 0.

OHE benefits machine learning models that operate on continuous vector space: by creating a sparse binary vector representation for each category, it prevents models from falsely inferring ordinal rela-

tionships between categories. However, we noticed in later experiments that OHE is more harmful than beneficial. For datasets like *House Price Prediction* where tree-based models dominate, OHE can cause data fragmentation. With one-hot, a tree can only test "this single category vs. the rest" per split, making **each split inefficient**, forcing deeper and more numerous tree ensembles.

4.4 Manual Imputation

Our manual imputation strategy recognizes that missing values aren't necessarily true data absence, but could rather carry semantic meanings. A type-aware imputation strategy is thus implemented.

For categorical features, we found that missing values often indicate the absence of that particular facility rather than being an unknown quantity. For example, *PoolQC* has a staggering training set and testing set missing rate of 99.52% and 99.79%. However, if we consider the fact that the US national average percentage of homes with pools is $\approx 8\%$, and the fact that the city of Ames is in the Midwest, its percentage is likely much lower than the national average, considering that pools are more common in warmer states like Florida and Arizona. Therefore, it is reasonable to infer that most of the missing values of *PoolQC* actually indicate the absence of pools. Therefore, we impute similar categorical features with a distinct "None" category.

For categorical features with low missing rate and no semantic interpretations (e.g., *MSZoning*, *KitchenQual*, etc.), we apply mode imputation. This is justified by the low missing rates in these features ($< 1\%$), making mode a robust central tendency estimator.

For most numerical features, we found that missing values also indicate facility absence rather than an unknown quantity. For example, a missing *GarageYrBlt* would indicate the absence of a garage in that particular house. Other similar features include *GarageArea*, *GarageCars*, *BsmtFinSF1*, *BsmtCond*, etc. Therefore, for such numerical features, we impute missingness with zero, reflecting the logical interpretation that properties without garages or basements have zero area and no amenities.

One exception, however, is the numerical feature *LotFrontage*, which measures the linear feet of street connected to the property. For this particular value, we utilize neighborhood-stratified median imputation. Concretely, for each entry with missing *LotFrontage*, we impute it with the median values of samples with the same *Neighborhood* (another categorical feature). This essentially utilizes human domain insight that lot characteristics tend to be homogeneous within local subdivisions.

4.5 Hand-Crafted Features

To capture nuanced non-linear relationships between features, we implemented domain-informed features to enhance the predictive power of the model. Our features can be categorized into five: Area Features, Temporal Features, Quality-Area Interactions, Seasonal Context, and Polynomial Features.

Area Features
1. *TotalSF* = *TotalBsmtSF* + *1stFlrSF* + *2ndFlrSF*: represents overall living areas
2. *TotalBathrooms*: combines full and half baths (including basement ones)
3. *TotalPorchSF*: aggregates all porch and deck areas

Temporal Features
1. *Age* = *YrSold* − *YearBuilt*: how many years has the house been built
2. *RemodAge* = *YrSold* − *YearRemodAdd*: how many years since last remodification
3. *IsNew* = $\mathbb{I}$(*YearBuilt* = *YrSold*): whether the house was sold on the same year it was built.

Quality–Area Interactions Interaction terms capture synergy between quality and space:
1. *OverallQual* × *GrLivArea*
2. *OverallQual* × *TotalSF*

Seasonal Context Based on month of sale (*MoSold*), a categorical feature *SeasonSold* is created
1. *SeasonSold* ∈ {*Winter*, *Spring*, *Summer*, *Autumn*}

Polynomial Expansions Squared and cubic terms are added for *OverallQual*, *GrLivArea*, *TotalSF*, *GarageArea*, *Age* to capture potential non-linear effects

5 Methodology

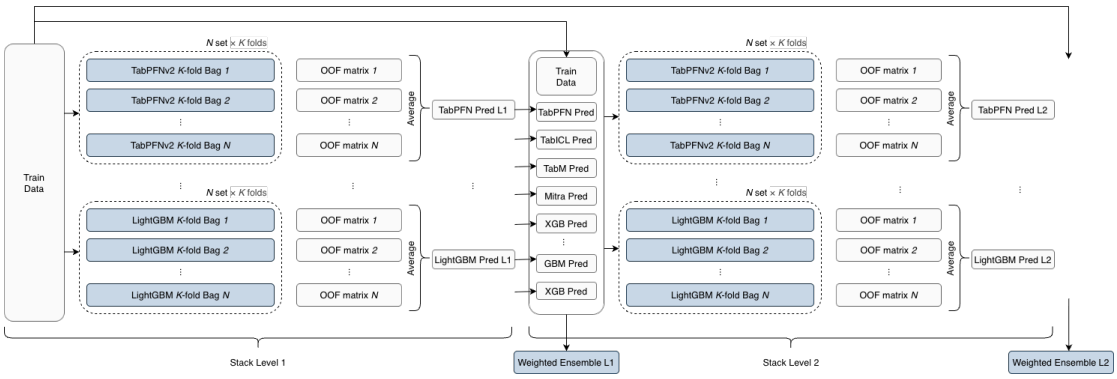


Figure 2: Illustration of our pipeline containing multi-set K-fold bagging, multi-layer stacking, and weighted ensembling of diverse model families at each level.[2]

Bagging Given M model families, for each family (e.g., TabPFN), we first create N different random shuffles of the input data and pass each shuffle into a bag; each bag trains K separate models in a K-fold cross-validation manner, and their fold-wise held-out predictions are concatenated into an out-of-fold (OOF) prediction matrix. The average of the N matrices is treated as the final output of that model family. The above process is repeated across all model families, producing M predictions.

Stacking The M different bagged predictions are concatenated with the training data and used as the input for the next layer. The next layer performs a similar procedure as the previous, where M different model families, each with N sets of K-fold bags, make predictions from the input data plus the previous layer's output. This process is known as stacking.

Weighted Ensembles At the final stage of each layer, a single *meta-model* is trained to combine the predictions of all bagged models from the second to the last layer. During training, the OOF predictions are used; during inference, a weighted average is performed. In practice, this is implemented as a greedy algorithm that attempts to maximize the predictive power.

5.1 Architecture Overview

Our predictive pipeline contains multi-layer stacking of bagged models. As illustrated in Figure 2, at each layer, we train $N \times K$ models of the same type, repeated across M different model families. This gives us a robust ensemble of predictions, which we concatenate and use as the input for the next layer. This process is repeated numerous times, producing multiple layers of "Stack". Notably, at each layer, an optional weighted ensemble can also be trained. It uses a greedy algorithm to assign weights to each of the M predictions, aiming to maximize predictive power.

In practice, we set high values of N, K, and the high stack level S. Then, we evaluate the performance of the weighted ensembles, as well as every bag of every model family in every stack level. By doing so, we gain an extremely nuanced understanding of how each parameter affects the performance of our model, informing hyperparameter choices.

5.2 Dynamic Stacking

Naive multi-layer stacking suffers from a critical limitation: an uninformative evaluation score. As the stacking level increases, the OOF evaluation score tends to become increasingly pessimistic, while the test error actually decreases. We hypothesize that such phenomena might be due to bias accumulation as stacking layers increase. As shown in Figure 3, test performance for both TabPFNv2 and Mitra increases with stacking level, while their evaluation scores decrease. This negative correlation is not ideal, as it undermines our only reliable proxy to test performance. Without an evaluation score as a stable predictor of test performance, we cannot tune our models.

[2]In practice, a different set model family is used (no tree-based models), though the core idea remains unchanged.

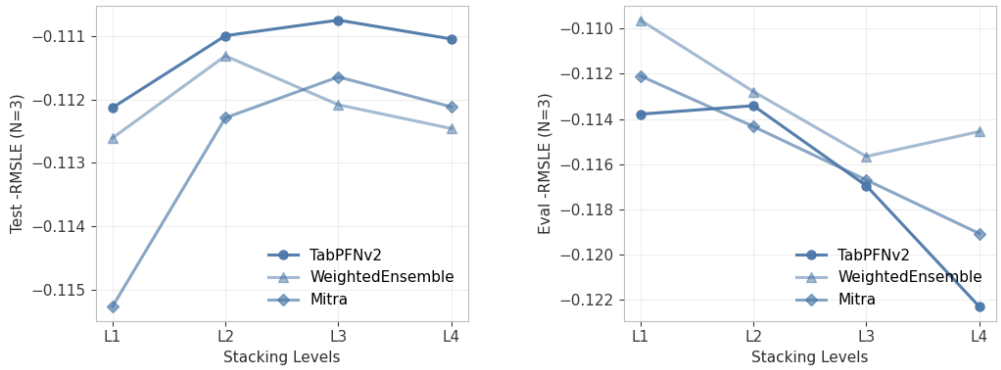


Figure 3: **Left**: test performance plotted against stacking levels; Both Mitra and TabPFNv2′s performances increases up to L3, while the weighted ensemble increase up to L2. **Right**: OOF evaluation score against stacking level shows an opposite result, where deeper stacks worsen validation scores.

To overcome this limitation, we used a technique known as *Dynamic Stacking*. The idea is intuitive. We start by splitting the original training set $\mathcal{D}$ into a training subset $\mathcal{D}_{\text{tr}}$ and a holdout set $\mathcal{D}_{\text{ho}}$, such that $\mathcal{D}_{\text{tr}} \cup \mathcal{D}_{\text{ho}} = \mathcal{D}, \mathcal{D}_{\text{tr}} \cap \mathcal{D}_{\text{ho}} = \varnothing$. Next, we perform a "test run" of our model using the exact same training procedure as before on $\mathcal{D}_{\text{tr}}$, except we measure its generalization performance not using the OOF validation sets within $\mathcal{D}_{\text{tr}}$, but using the separate $\mathcal{D}_{\text{ho}}$. The separate holdout set provides a stable measurement of performance unaffected by stacking, allowing us to determine the best-performing layer in the stack. Finally, we can re-train the entire ensemble on the full dataset $\mathcal{D}$ and take the best-performing layer we found during the earlier test run as our final output layer.

5.3 Zero-shot Hyperparameter Optimization

Hyperparameter optimization (HPO) can be extremely tedious, especially for a large ensemble model like ours. With multiple model families, the hyperparameter combinations explode exponentially. Furthermore, HPO often yield marginal performance gain and is usually our last resort.

We overcome this by leveraging a dataset called *TabRepo* [12], which contains the predictions and metrics of 1310 models evaluated on 200 different tabular datasets, each with 3 folds. Notably, *TabRepo* not only stores test set predictions, but also raw out-of-fold predictions. These precomputed results allow for extremely fast evaluation of the Caruana-style greedy ensemble by simply querying the dataset for the corresponding model's prediction vectors and performing vector arithmetic to find optimal combinations that minimize the loss. The authors took this to their advantage and searched for a fixed set of model configurations (which they called a "portfolio") that are maximally complementary. In simpler terms, the objective is to find a single set of hyperparameters that maximizes the average performance across all 200 datasets. This single fixed portfolio allows them to achieve SOTA results far exceeding those of carefully tuned AutoML systems. The authors named this "zero-shot HPO."

In practice, we utilize the "zero-shot portfolio 2025" in AutoGluon [13], [14], which is meta-learned from an even larger dataset ($\sim 25,000,000$ model instances trained across 1053 tabular datasets) called TabArena [15]. The dataset is ever-growing thanks to an open source collaborative effort.

6 Interpretability

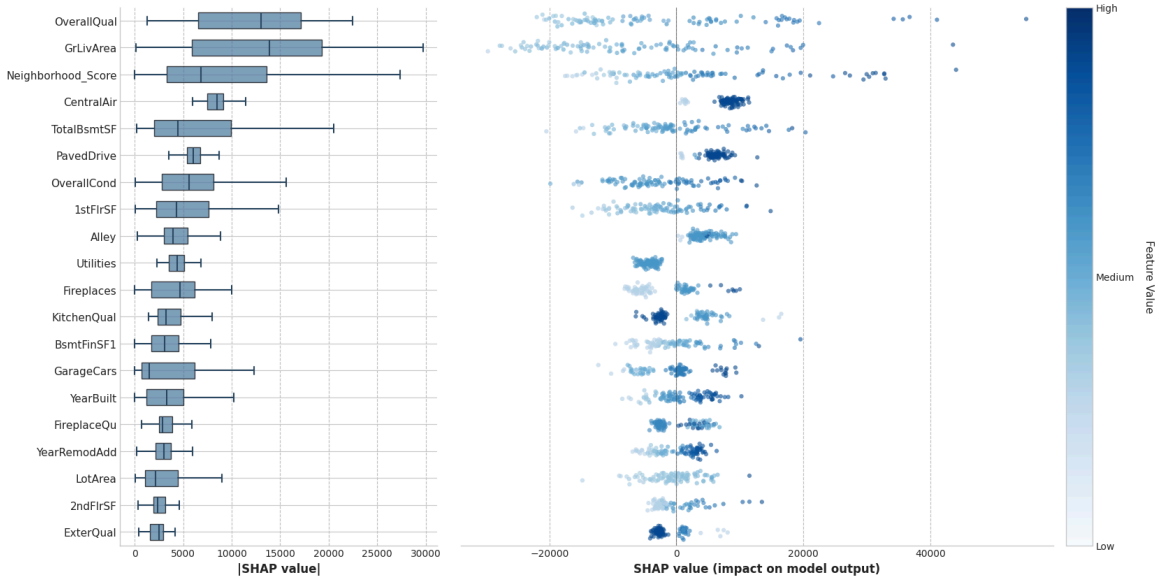


Figure 4: SHAP values of the top twenty most impactful features according to our model. **left**: Feature importance ranked based on mean absolute SHAP values. The box plot showcase the variance of their absolute impact across data samples. **right**: Detailed SHAP summary plot showing relationship between feature values (color scale: low → high) vs. their impact on model output. Each dot on the right plot represents a single prediction; their position along the x-axis indicates feature's contribution to increasing (positive) or decreasing (negative) prediction values.

As shown in Figure 4, *GrLiveArea* (ground living area) and *OverallQual* (overall quality) are the most influential features, contributing on average 15,000 absolute price to the final prediction. The summary plot of their SHAP values reveal a clear pattern: higher feature values (darker blue) consistently push predictions upward (positive SHAP), aligning with our EDA analysis in Figure 10, where larger, higher-quality homes command premium prices.

Though these two features' net effects are strong, both features exhibits large variances. This suggests possible interaction effects. Figure 5 (left) shows precisely these feature interactions. For small houses, SHAP values for *GrLiveArea* tends to be lower when *OverallQual* is high. In simpler terms, small size is more detrimental to price for high-quality houses than it is to low quality houses.

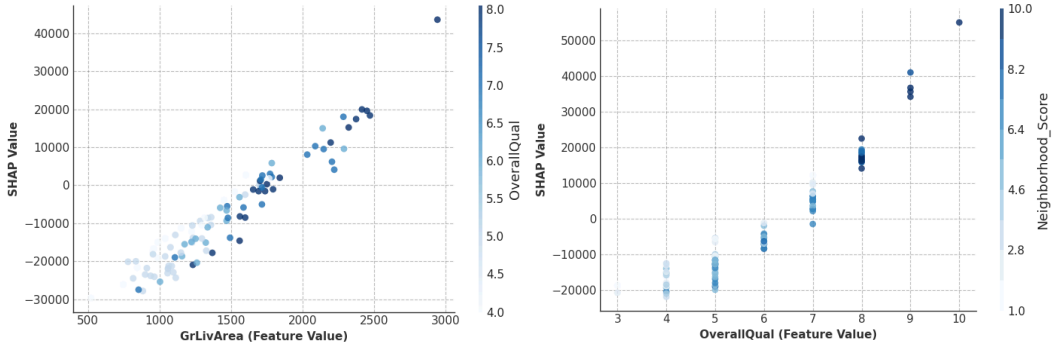


Figure 5: SHAP dependence plots. **left**: dependence plot of *OverallQual* and *GrLivArea*, where vertical axis shows SHAP values of *GrLivArea*; **right**: dependence plot of *OverallQual* vs. *Neighborhood*, where vertical axis shows SHAP vaslues of *OverallQual*. **Note**: the neighborhood is originally a categorical feature, so we prompt GPT-5 to rate neighborhood quality on a scale of 1-10.

7 Ablations

Model Selection Ablation:

Table 2: Model Family Choice Ablation

Configurations			Model Choice (↓)			TFM vs. others (↑)	
F.E.	S	N	Tr+TFMs	Trees	TFMs	ΔTrees	ΔTr+TFMs
Basic	1	1	0.1123	0.1228	**0.1114**	+0.0114	+0.0009
Every	1	1	0.1150	0.1238	**0.1142**	+0.0096	+0.0008
Basic	2	1	0.1120	0.1231	**0.1114**	+0.0117	+0.0006
Every	2	1	**0.1147**	0.1235	0.1153	+0.0082	−0.0006
Basic	1	2	0.1125	0.1225	**0.1120**	+0.0105	+0.0005
Every	1	2	0.1149	0.1234	**0.1144**	+0.0090	+0.0005
Basic	2	2	0.1123	0.1227	**0.1115**	+0.0112	+0.0008
Every	2	2	**0.1146**	0.1234	0.1149	+0.0085	−0.0003
Mean Score			0.1135	0.1231	**0.1131**	+0.0100	+0.0004
Best Score			0.1120	0.1225	**0.1114**	+0.0111	+0.0006

Table 2 answers the question of "which baseline models should be included in our ensemble?" A total of 24 experiments were performed across 8 different sets of configurations. For each configuration, we examine three different model choices: Tree + TFMs, Trees only, and TFMs only. Tree refers to the tree-based models mentioned in Table 1 and a few more, namely, LightGBM, XGBoost, CatBoost, Random Forest, and ExtraTrees. TFMs refers to a set of three different tabular foundation models, namely, TabPFNv2, Mitra, and TabM.

Results shows that the inclusion of tree-based models in our ensemble (bagging-stacking pipeline) consistently hurts performance, worsening RMSLE performance by as much as 0.0114 ($0.1114 \rightarrow 0.1228$). While Tree+TFMs performed better than Tree, it is reasonable to conclude that the TFMs did all the heavy lifting. This justified the complete removal of tree ensembles in our final model.

Stacking and Ensembling Ablation

Table 3: Stacking & Weighted Ensembling Ablation

Stacking	Ensembling	RMSLE
x	√	0.1147
√	√	0.1117
x	x	0.1128
√	x	**0.1101**

Table 3 shows that stacking-only yields the highest performance. Whereas ensembling-only performs worse than even baseline (w/o stacking, w/o ensembling), let alone hybrid (stacking + ensembling). This suggests that ensembling is harmful to our model's performance, irrespective of stacking, while stacking boosts predictive power universally. This is expected, as stacking provides strong generalization capabilities by effectively combining the base learners in a hierarchical manner, whereas the greedy approach used by the weighted ensemble risks overfitting to the validation set. Consequently, pure stacking emerges as our final design.

Feature Engineering Ablation on Foundation Models and Tree-Based Models:

Table 4: Feature Engineering Choice Ablation

F.E.	TFMs	Tree-based	Linear
None	**0.1112**	<u>0.1199</u>	0.1331
Numeric	<u>0.1114</u>	0.1199	0.1330
KNN	0.1127	0.1223	0.1324
One-Hot	0.1144	0.1373	**0.1300**
Manual	0.1141	**0.1198**	<u>0.1319</u>

Tree-based models performed best when combined with our hand-crafted imputer and features described in Section 4.4 and Section 4.5. Whereas techniques such as KNN Imputer and One-Hot encoding (OHE) hurt performance. One-hot encoding can cause inefficient splits in trees, especially for high-cardinality features, forcing deeper and more numerous trees, which can hurt performance.

For linear models, all feature engineering techniques benefit performance, with OHE generating the most pronounced boost, followed by manual features. OHE generally benefits linear models by letting the model learn category-specific intercept shifts, whereas our manual features help improve the linear separability of non-linear features.

As expected, TFMs consistently outperform all other model families. Surprisingly, while tree-based and linear models favor specific feature engineering techniques, our TFMs perform best **without any feature engineering**. This contradicts the stereotypical fragility of neural networks on heteroscedastic, irregular, unnormalized tabular datasets.

8 Discussions

Our final model achieved 0.10939 RMSLE on the Kaggle official leaderboard, ranking top 10 amongst non-probed solutions. This result was obtained through a pure tabular foundation model ensemble ($N = 3$, $K = 8$, $S = 3$, no tree models, no feature engineering, no greedily weighted ensembles) with dynamic stacking and zero-shot HPO. Having reached SOTA performance and conducted comprehensive ablations, we now address the three questions main questions posed in the introduction section:

- Do foundation models obsolete classical methods?
- How should feature engineering be done in the age of pre-training?
- How should foundation models be integrated with established pipelines?

8.1 Do Foundation Models Obsolete Classical Methods?

Our evidence indeed strongly suggests that tabular foundation models have surpassed competitive tree-based models. Despite careful tuning and ensembling, the best performing tree-based ensemble only managed to score 0.1198 RMSLE in our experiments; whereas, TabPFNv2 was able to reach 0.1154 RMSLE right out of the box. And with further tuning, this score further increases to 0.1094. Furthermore, Table 2 demonstrates that across 8 configurations, pure TFM ensembles consistently outperformed tree-based ensembles by an average margin of $\Delta = +0.0100$ RMSLE. More tellingly, when trees were **combined** with TFMs in a hybrid ensemble, performance degraded, suggesting that trees contributed primarily noise rather than a complementary signal.

We hypothesize that such superiority can be attributed to the fundamental difference in inductive biases between TFMs and tree-based ensembles. During training, the information obtainable by classical tree-based methods is purely from the training data itself. The likely only prior knowledge (through inductive bias) encoded into these tree models is the fact that tabular datasets usually contain lots of mixed feature types, sparse or missing values, and non-smooth decision boundaries. Hence, maximizing tree-based models' performance requires strong human priors encoded through hand-crafted features (evident by Table 4).

In contrast, TFMs amortize Bayesian inference across millions of synthetic tabular data during pre-training. TabPFNv2, for example, explicitly learns $p(y'|x', D)$ by approximating the posterior predictive integral over **all plausible data-generating mechanism**, essentially learning a prior over tabular structures, far more nuanced than the tree-based inductive biases.

However, import caveats must be acknowledged. Predictive performance is not always the single considered factor. In Table 1, we showed that TabPFN's inference time is much higher than that of LightGBM. This prohibits its usage for production systems that require real-time processing at scale. Furthermore, TFMs are faced with architectural constraints. For instance, TabPFNv2 has an upper limit of 10,000 samples, due to the quadratic memory cost associated with transformer architectures; whereas tree ensembles handle arbitrarily large datasets.

Therefore, while TFMs clearly dominate in predictive performance for small- to medium-sized tabular competitions like ours, claiming complete obsolescence would be premature. Rather, we observed that TFMs now occupy the Pareto frontier of accuracy-critical applications with medium data scales.

8.2 How Should Feature Engineering Be Done in the Age of Pre-Training?

Our most counterintuitive finding is that TFMs perform best with **zero feature engineering**. Table 4 reveals that every preprocessing technique we tested, be it KNN impute (+0.0015 RMSLE), manual imputation and features (+0.0029), one-hot encoding (+0.0032), and even basic Yeo-Johnson normalization (+0.0002), degrades TFM performance.

This challenges conventional wisdom that noisy, heterogeneous tabular datasets require careful preprocessing. Why do TFMs prefer raw data? We propose three mechanisms:

1. **Prior misalignment on feature transformation** In TabPFNv1′s pretraining stage, it encountered 9,216,000 different tabular datasets. TabPFNv2 scales this up much further and incorporates intentionally messy and heterogeneous synthetic data, commonly seen in the real world—including categorical features, missing values & outliers. The model thus learned to extract predictive signal from raw, unprocessed distributions. When we transform features (e.g., through Yeo-Johnson transformation), we are imposing specific distributional assumptions (Gaussianity) that may conflict model's learned priors. Our hand-crafted features inject human biases, no match for the data-driven patterns learned through an enormous amount of examples.

2. **Information loss on imputation** Consider out KNN imputer, which replaces missing values with distance-weighted neighbors. While this reduces feature variance, it also discards the inherent information in "missingness" itself. As earlier discussed in Section 4.4, missing values like *PoolQC* might semantically indicate pool absence. TFMs might be able to capture even more nuanced semantics than our facility absence assumption. Therefore, any imputation strategies might have inadvertently caused information loss.

3. **Attention handles heterogeneity** Unlike trees which recursively partition feature space, Transformer in TFMs use attention to dynamically weight feature relevance and compute feature interactions. OHE replaces one categorical feature with many binary pseudo-features, breaking the per-feature semantics that the model expects. High sparsity and cardinality introduced by OHE can be both a distributional shift and an inefficiency for the models' attention mechanisms.

Practical implications are clear: **the best feature engineering is NO feature engineering**

8.3 How Should Foundation Models Be Integrated with Established Pipelines?

Our architecture (Figure 2) demonstrates that TFMs benefit from classical ensemble techniques, but with a few nuaunces: **1)** pure stacking dominates weighted ensembling; Table 3 shows that stacking alone yields the best perfomrnace. **2)**, dynamic stacking resolves OOF pessimism. We noticed that naive multi-layer stacking produces increasingly pessimistic validation scores despite test performance increase—a phenomenon not seen in tree-only stacking (which actually suffers from overconfident validation scores). **3)** zero-shot portfolio has proven to be effective even with TFMs, not just gradient boosted models, saving a massive amount of tuning time.

9 Limitations

Our results are competitive and the analysis is encouraging, but several limitations warrants discussion. Limited by the scope of this course project, our conclusions are drawn entirely from a single competition dataset of moderate size (1,460 samples). Rigorously evaluating TFMs' true performance requires aggregating results from hundreds of real-world tabular datasets, as well as on larger datasets (~10,000 samples) where tree ensembles might better exploit data abundance. Therefore, conclusions drawn from this project should be taken with a grain of salt, and serve more as qualitative evidence than a rigorous quantitative one.

References

[1] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco California USA: ACM, Aug. 2016, pp. 785–794. doi: 10.1145/2939672.2939785.

[2] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "CatBoost: unbiased boosting with categorical features," in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2018. Accessed: Nov. 12, 2025. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2018/hash/14491b756b3a51daac41c24863285549-Abstract.html

[3] G. Ke *et al.*, "LightGBM: A Highly Efficient Gradient Boosting Decision Tree," in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2017. Accessed: Nov. 12, 2025. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/hash/6449f44a102fde848669bdd9eb6b76fa-Abstract.html

[4] J. Qu, D. Holzmüller, G. Varoquaux, and M. L. Morvan, "TabICL: A Tabular Foundation Model for In-Context Learning on Large Data." Accessed: Nov. 12, 2025. [Online]. Available: http://arxiv.org/abs/2502.05564

[5] N. Hollmann *et al.*, "Accurate predictions on small data with a tabular foundation model," *Nature*, vol. 637, no. 8045, pp. 319–326, Jan. 2025, doi: 10.1038/s41586-024-08328-6.

[6] X. Zhang *et al.*, "Mitra: Mixed Synthetic Priors for Enhancing Tabular Foundation Models." Accessed: Nov. 12, 2025. [Online]. Available: http://arxiv.org/abs/2510.21204

[7] Y. Gorishniy, A. Kotelnikov, and A. Babenko, "TabM: Advancing Tabular Deep Learning with Parameter-Efficient Ensembling." Accessed: Nov. 12, 2025. [Online]. Available: http://arxiv.org/abs/2410.24210

[8] H. Drucker, C. J. C. Burges, L. Kaufman, A. Smola, and V. Vapnik, "Support Vector Regression Machines," in *Advances in Neural Information Processing Systems*, MIT Press, 1996. Accessed: Nov. 12, 2025. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/1996/hash/d38901788c533e8286cb6400b40b386d-Abstract.html

[9] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct. 2001, doi: 10.1023/A:1010933404324.

[10] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, Jan. 1967, doi: 10.1109/TIT.1967.1053964.

[11] I.-K. Yeo and R. A. Johnson, "A new family of power transformations to improve normality or symmetry," *Biometrika*, vol. 87, no. 4, pp. 954–959, Dec. 2000, doi: 10.1093/biomet/87.4.954.

[12] D. Salinas and N. Erickson, "TabRepo: A Large Scale Repository of Tabular Model Evaluations and its AutoML Applications." Accessed: Nov. 12, 2025. [Online]. Available: http://arxiv.org/abs/2311.02971

[13] N. Erickson, "autogluon/autogluon." Accessed: Nov. 12, 2025. [Online]. Available: https://github.com/autogluon/autogluon

[14] N. Erickson *et al.*, "AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data." Accessed: Nov. 12, 2025. [Online]. Available: http://arxiv.org/abs/2003.06505

[15] N. Erickson *et al.*, "TabArena: A Living Benchmark for Machine Learning on Tabular Data." Accessed: Nov. 12, 2025. [Online]. Available: http://arxiv.org/abs/2506.16791

[16] R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes, "Ensemble selection from libraries of models," in *Twenty-first international conference on Machine learning - ICML '04*, Banff, Alberta, Canada: ACM Press, 2004, p. 18. doi: 10.1145/1015330.1015432.

A Exploratory Data Analysis

Dataset Overview The house price prediction dataset has 79 explanatory variables. Participant are required to train their own model on the 1460 training samples and generate the predicted house prices on 1459 testing instances.

A.1 Label Distribution (*SalePrice*)

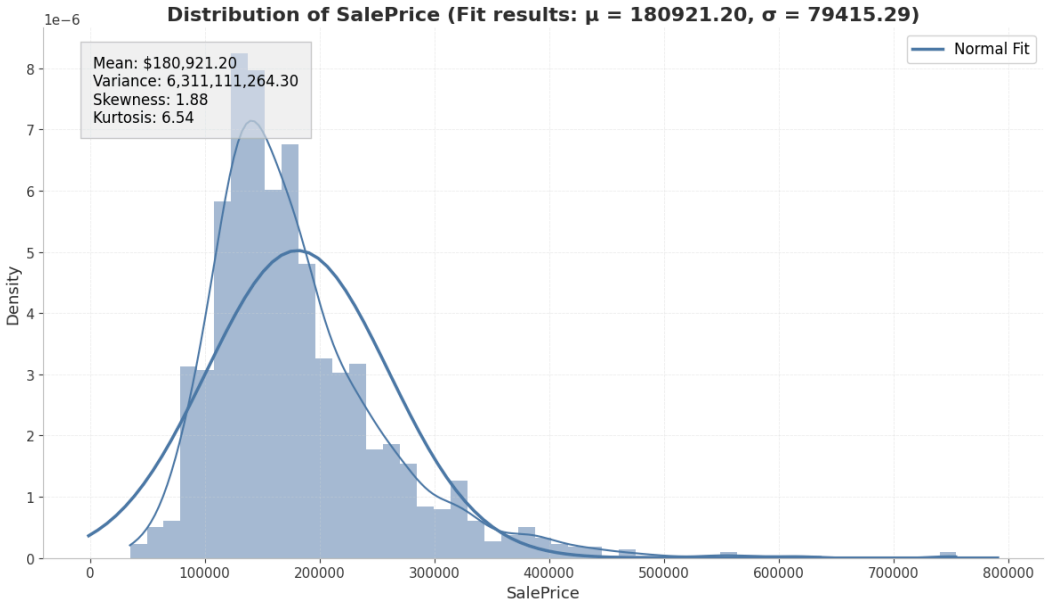


Figure 6: Distribution of Sale Price

The target variable **SalePrice** is **right-skewed**, indicating that most houses are priced below the mean. Such pattern is typical in real-estate markets, where the majority of properties fall within an affordable price range, while a limited number of luxury houses exhibit disproportionately higher prices. The presence of this long tail implies **heteroscedasticity** in the data, which may violate the normality assumption required by certain regression-based models.

A.2 Numerical Feature Distribution

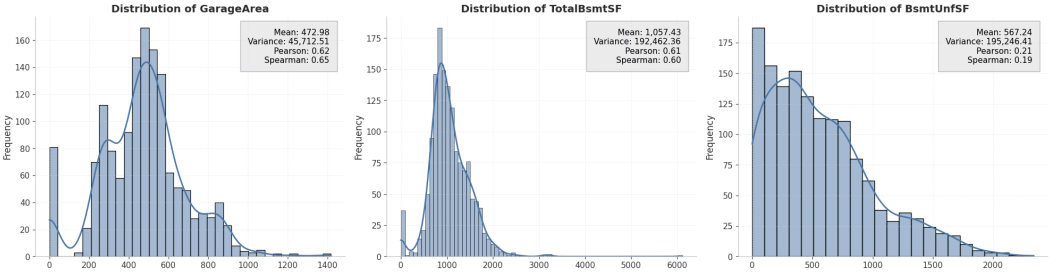


Figure 7: The distribution of *GarageArea1* (**left**) is bimodal, shown by the cluster at zero (indicating properties without a garage) and peak around 473. *TotalBsmtSF* (Total Basement Square Feet) is right-skewed (**middle**), suggesting that while most properties have a basement, few are large. The distribution for *BsmtUnfSF* (Unfinished Basement Square Feet) (**right**) is peaked at zero, meaning many properties either have no basement or a fully finished one.

There are 31 numerical features in total. As shown in Figure 7, these features exhibits strong skewness and sometimes clusters at zero. These characteristics prompts us to perform normalization and skewness adjustments during feature engineering. A full visualization of all numerical features can be found in the Appendix, Figure 11.

A.3 Categorical Data Distribution

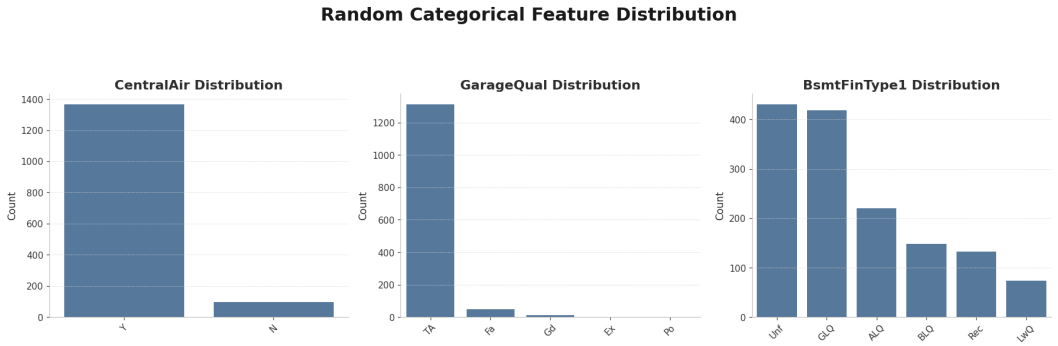
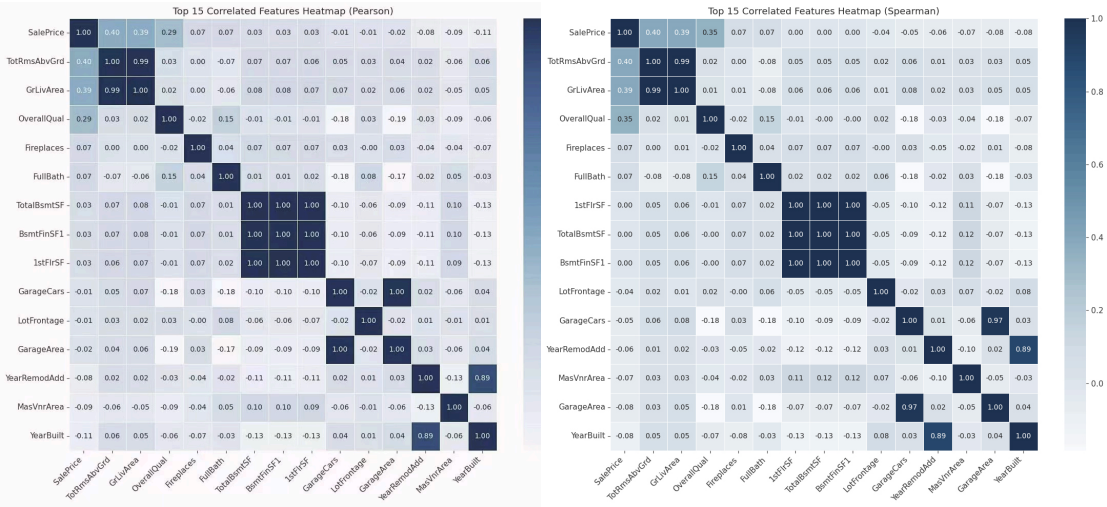


Figure 8: **left**: *CentralAir* has most its values falling under the *Y* (yes) category, meaning houses has a central air conditioner. **middle**: This is a feature that rates the quality of the garage, with *TA*, being most common. **right**: This feature describes the quality of the main finished area of the basement.

The dataset contains 48 categorical features . As shown in Figure 8, the distributions of these features often exhibit a severe imbalance, meaning that one or a few categories account for the vast majority of samples. These distributional characteristics indicate that these variables require careful handling during the feature engineering phase. It is necessary to select appropriate encoding methods (such as one-hot, label, or target encoding) to convert these text-based categories into a numerical format that the model can process. A complete visualization of all categorical feature distributions can be found in the Appendix, Figure 12.

A.4 Correlation Heatmap Visualization



The Pearson correlation coefficient measures the strength and direction of the **linear** relationship between two **continuous** variables. Whereas, the Spearman correlation coefficient assesses the strength and direction of the **monotonic** relationship between two variables. Unlike Pearson, it does not assume a linear relationship and is less sensitive to outliers because it operates on the ranks of the data values rather than the raw values themselves.

A.5 Missing Value Analysis

Table 5 lists the features with significant portions of missing values. A key pattern emerges: for features with extremely high missing rates like *PoolQC*, *Alley*, and *Fence* (>80%), the absence of a value systematically indicates the feature's non-existence (e.g., the property has no pool). Consequently, these will be treated as a distinct "None" category rather than being imputed.

In contrast, the missing values for *LotFrontage* represent genuinely unknown data that will require an appropriate imputation method. Importantly, these missing data patterns are consistent across the training and test sets, allowing for a unified preprocessing strategy, which is critical for building a robust model.

Table 5: Missing Value Table

Feature	PoolQC	MiscFeature	Alley	Fence	MasVnrType	FireplaceQu	LotFrontage
Train Missing (%)	99.52	96.30	93.77	80.75	59.73	47.26	17.74
Test Missing (%)	99.79	96.50	92.67	80.12	61.27	50.03	15.56

A.6 Strong Predictors

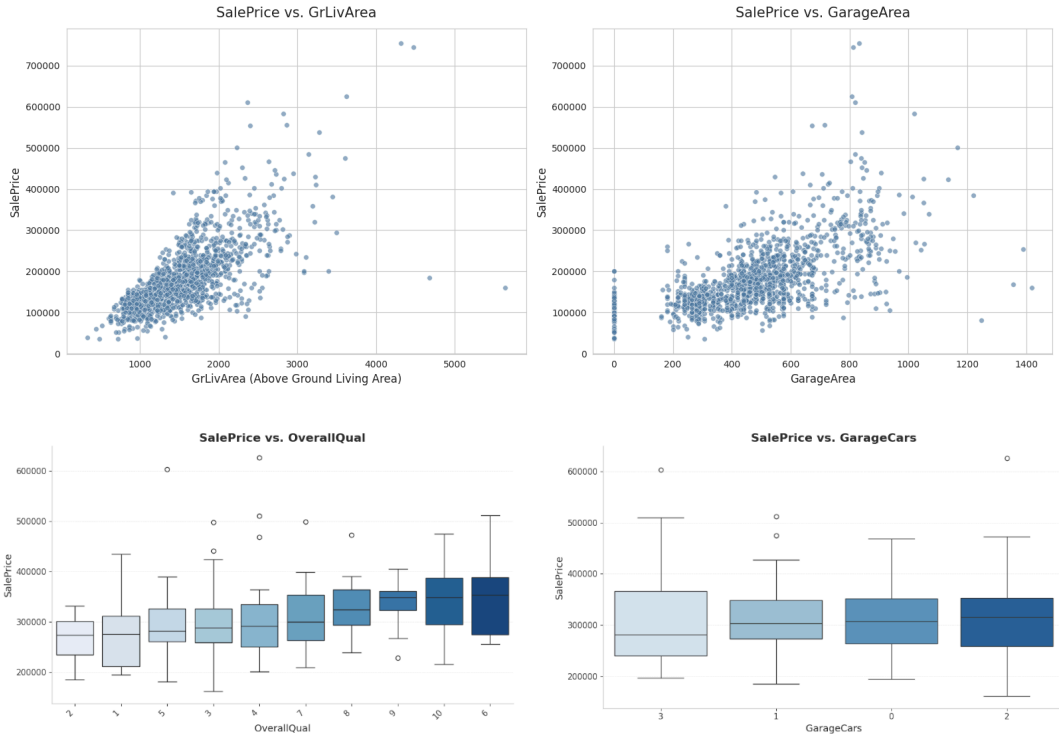
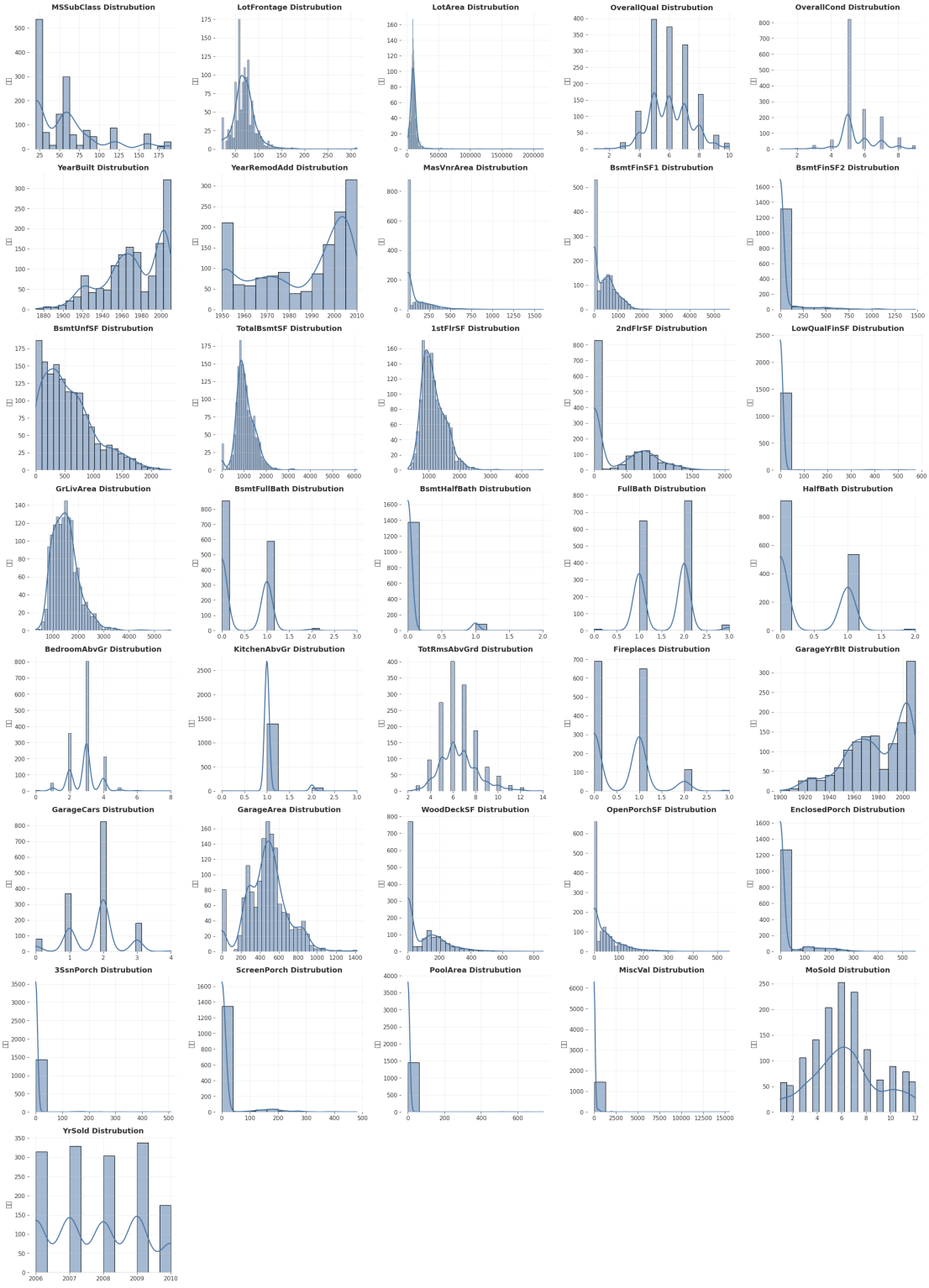
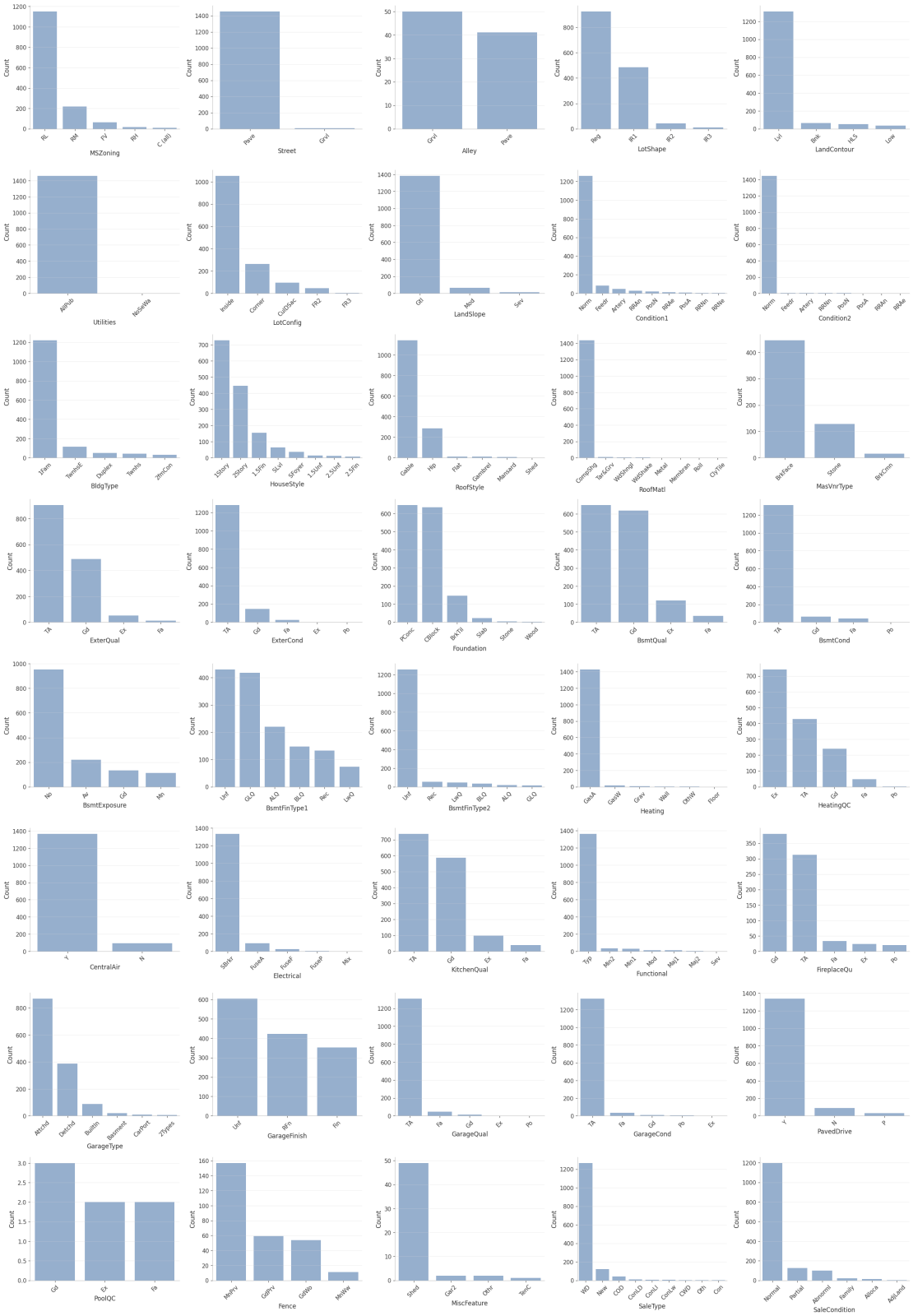


Figure 10: **Top Left**: *GrLivArea* shows a clear positive linear relationship with *SalePrice* (Pearson ≈ 0.39). As living area increases, prices rise accordingly. A few large but low-priced outliers slightly weaken the correlation, but overall, *GrLivArea* is a key factor influencing house value. **Top Right**: *GarageArea* has a low but positive Pearson correlation (due to outliers), indicating some linear relationship with price, though noisy. **Bottom Left**: *OverallQual* correlates strongly and monotonically with *SalePrice* (Spearman ≈0.35). Higher quality ratings correspond to higher median prices in a clear stepwise pattern. *OverallQual* effectively distinguishes property price levels. **Bottom Right**: *GarageCars* exhibits a near-zero Spearman correlation, showing little monotonic relationship. Only homes with three-car garages show slightly higher prices, while others overlap. Garage capacity exerts minimal influence on price.

A.7 Numerical Feature Distribution



A.8 Categorical Feature Distribution



B Hyperparameters

B.1 Ensemble Hyperparameters

Ensembling	
Name	Values
Stacking Levels (S)	5
Bagging Sets (N)	3
Bagging Folds (K)	8
Design Choices	
Dynamic Stacking	True
Feature Pruning	False
Weighted Ensembles	False
Feature Engineering	None
Label Transformation	$\log(1+y)$
Model Families	TabPFNv2, TabM, Mitra

B.2 Model Specific Hyperparameters

TabPFNv2 (from *TabArena*)			
Name	Bagging Set 1	Bagging Set 2	Bagging Set 3
Softmax Temperature (τ)	0.75	0.9	0.95
Avg. before Softmax	False	True	False
N ensemble repeats	4	4	4
Model ID	wyl4o83o	5wof9ojf	default
TabM (from *TabArena*)			
d_{block}	864	848	1024
$d_{\text{embedding}}$	24	28	32
N_{block}	3	4	4
Dropout Rate	0.0	0.4	0.0
Grad. Clipping Norm	1.0	1.0	1.0
Learning rate (α)	0.002	0.001	0.0006
Patience	16	16	16
Weight Decay	0.0	0.069	0.017
Mitra (from *TabArena*)			
$N_{\text{Estimators}}$	1	1	1
Finetune Steps	50	50	50

C Final Scores

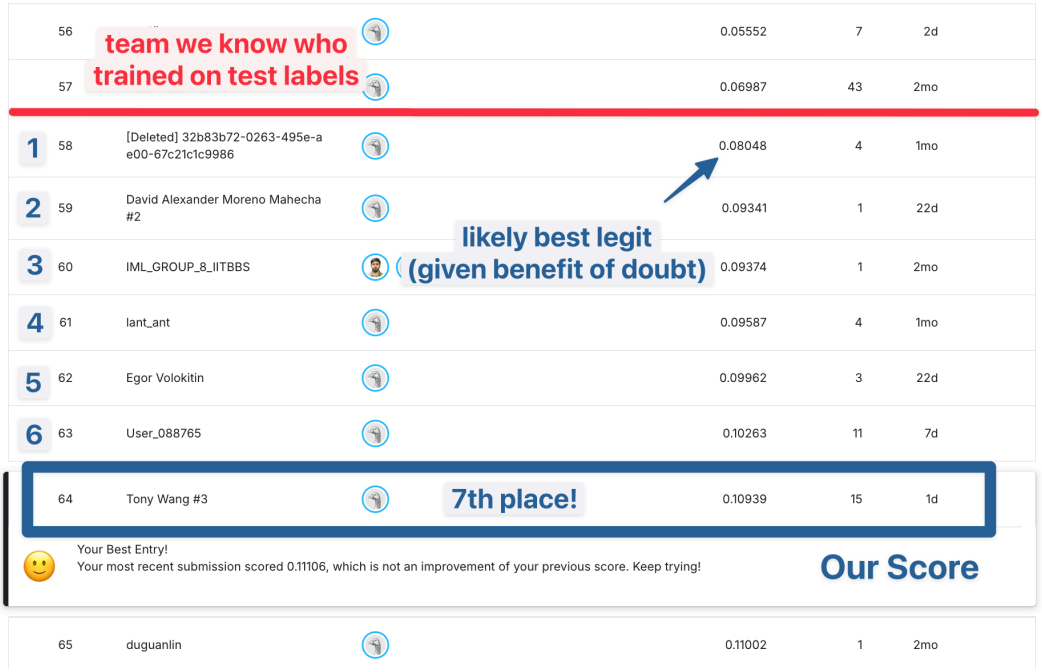


Figure 13: Public leaderboard screenshot with our submission highlighted. We became aware, via first-hand disclosure from a participant, that at least one submission positioned above the **red line** used ground-truth test labels. Thus, entries beyond are non-comparable to rule-compliant results.

D Answers to Project Questions

D.1 *Which feature engineering effort is most useful in your competition work? Any reason behind?*

For final solution:
The single most useful "feature engineering" for our winning system was to **do none at all**.
Every preprocessing we tried on TFMs, be it KNN impute, one-hot encoding, Yeo–Johnson, and our manual imputations/features, hurt RMSLE. We discuss this in detail in Section 8. But, in short, this is likely due to prior misalignment with TFM pretraining, information loss when imputing informative missingness, and OHE-induced sparsity that undermines attention mechanism.

For classical models:
Type-aware manual imputations and domain features helped: adding a "None" category for absent facilities and neighborhood-median imputation for LotFrontage improved trees, and OHE benefited linear models. However, since even an untuned TabPFNv2 outperformed every classical model we tried, the most impactful "feature engineering decision" overall was to **keep raw data**.

D.2 *What is the best improvement across submissions?*

Our largest performance lift came from moving beyond single-shot baselines and hybrid stacks to a pure TFM, stacking-only pipeline with dynamic stacking and zero-shot portfolios. This means:

1. Remove any tree-based models → use TFMs only (i.e., TabPFNv2, Mitra, TabM)
2. Remove all weighted ensembles → use stacking and OOF bagging.

Results:
1. Holdout validation RMSLE from about 0.1128 → 0.1101,
2. Kaggle leaderboard test score from 0.11392 → 0.10939.