# The Machine Learning Journey

By: Daniel Immediato
Date: 8/16/24

# Preliminary: What is the Ames Dataset?

- Ames, Iowa is a college town of Iowa State University. The Ames dataset consists of the housing sale records between 2006-2010, including features like their area, number of rooms, and other attributes. The goal of this project was to create the most “accurate machine learning model with the best score”.

1. A machine learning model finds patterns from data. In our case, the data has 81 columns and 2580 rows. It would usually, then, make a prediction from said given data set.

2. A model’s “score” is a model evaluation, such as accuracy or R-squared, that explains the strength of the model.

- This can be achieved by:

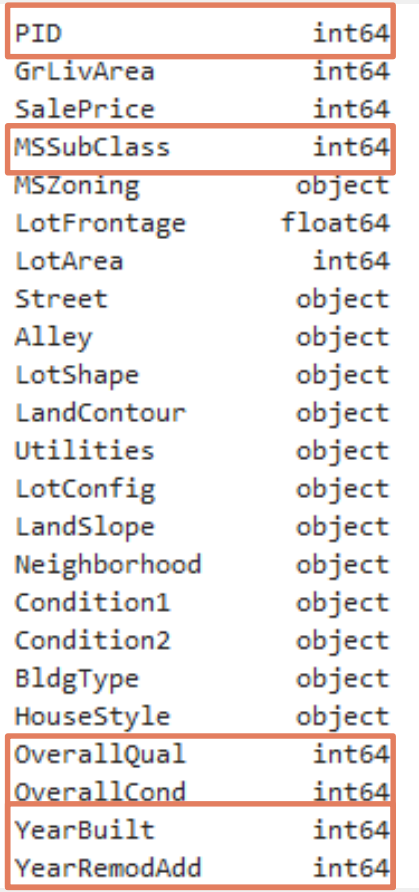
1. Reshaping the data into a workable state

2. Picking and then creating the best models that work for you

3. Tuning said models

# Preprocessing: Data Types

- Before beginning any sort of analysis, one must examine whether or not the data “can” be anyalyzed properly to begin with. The typical dataset is split between numeric, categorical, and ordinal values. When coding, one has to ensure values are consistently classified throughout.
    - As a few examples, ‘MSSubClass’ is actually a category, not an integer, and dates, like ‘YearBuilt’ and ‘YearRemodAdd’, are also categories. PID is ultimately pointless as a feature when it comes to what we initially want to look at.
    - It is necessary to convert these to their proper data types, otherwise errors may occur or results may appear different than expected.
    - Variables like ‘OverallQual’ and ‘OverallCond’ must be manually encoded to a ‘least-to-greatest’ directory.



| | |
|---|---|
| PID | int64 |
| GrLivArea | int64 |
| SalePrice | int64 |
| MSSubClass | int64 |
| MSZoning | object |
| LotFrontage | float64 |
| LotArea | int64 |
| Street | object |
| Alley | object |
| LotShape | object |
| LandContour | object |
| Utilities | object |
| LotConfig | object |
| LandSlope | object |
| Neighborhood | object |
| Condition1 | object |
| Condition2 | object |
| BldgType | object |
| HouseStyle | object |
| OverallQual | int64 |
| OverallCond | int64 |
| YearBuilt | int64 |
| YearRemodAdd | int64 |

# Preprocessing: Nulls and NA

- An NA value is likely to cause the most issues when dealing with data. When running functions, depending on the program, many can't process them and may produce an error or show results incorrectly. To deal with them you have several options, such as:

    1. Assume what the value was going to be based on related values.

    2. Replace values as a "0" or "none" if the value is legitimately lacking, or if there is no value.

    3. Replace values with most common value or median.

- Shown is an untransformed version of the "count" of all missing values in the dataset.

| | |
|---|---|
| LotFrontage | 462 |
| Alley | 2412 |
| MasVnrType | 1573 |
| MasVnrArea | 14 |
| BsmtQual | 69 |
| BsmtCond | 69 |
| BsmtExposure | 71 |
| BsmtFinType1 | 69 |
| BsmtFinSF1 | 1 |
| BsmtFinType2 | 70 |
| BsmtFinSF2 | 1 |
| BsmtUnfSF | 1 |
| TotalBsmtSF | 1 |
| Electrical | 1 |
| BsmtFullBath | 2 |
| BsmtHalfBath | 2 |
| FireplaceQu | 1241 |
| GarageType | 127 |
| GarageYrBlt | 129 |
| GarageFinish | 129 |
| GarageCars | 1 |
| GarageArea | 1 |
| GarageQual | 129 |
| GarageCond | 129 |
| PoolQC | 2571 |
| Fence | 2055 |
| MiscFeature | 2483 |

# Preprocessing: The Pipeline

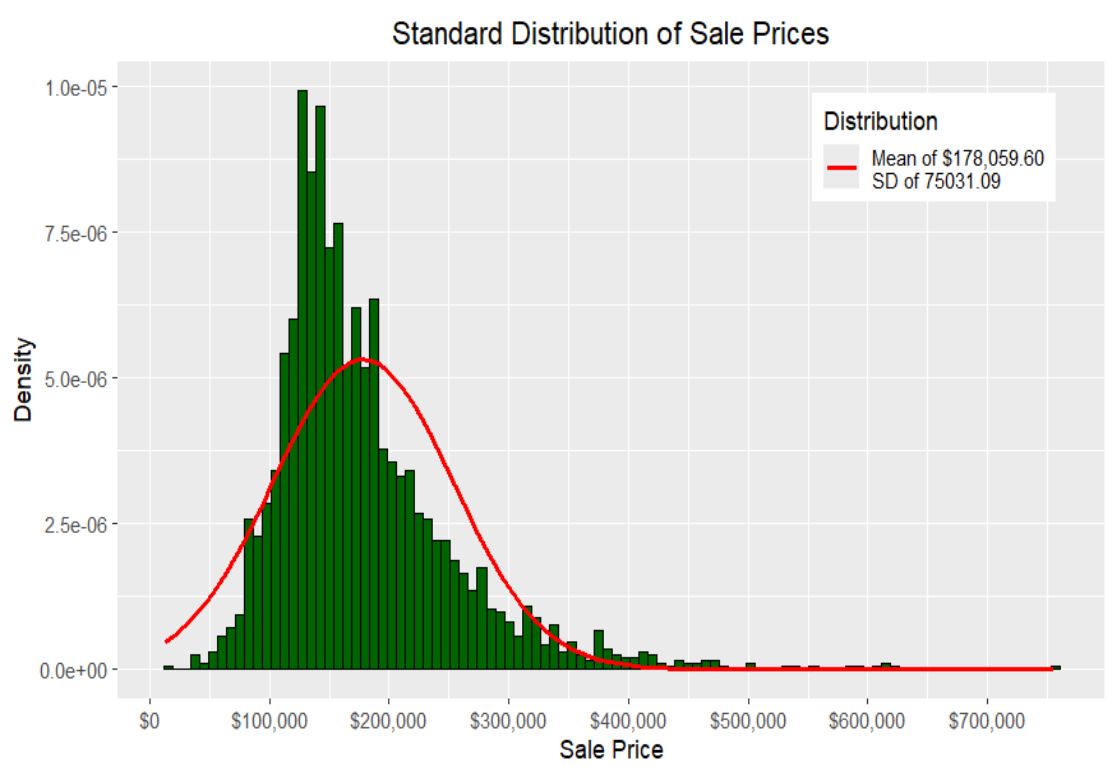
- The problem with preprocessing independently of each other is that it can lead not only to data leakage, but as the code becomes increasingly complex, so does the requirement for automation. Pipelines can be used to condense “heavier” data processing into more easily digestible, and, more importantly, recallable steps.
    - For our purposes, the basic pipeline consisted of: separating the data types (and classifying them appropriately), deal with the “NA” values using a SimpleImputer, scaling the numeric values, and then encoding the categorical and ordinal values. These can then be combined into a preprocessor using ColumnTransformer.

```
electrical_transformer = Pipeline(steps=[
    ('impute_electrical', SimpleImputer(strategy='most_frequent')),
    ('ordinal_electrical', OrdinalEncoder(categories=[ordinal_order['Electrical']]))
])
numeric_transformer = Pipeline(steps=[
    ('impute_mean', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())
])
ordinal_transformer = Pipeline(steps=[
    ('impute_none', SimpleImputer(strategy='constant', fill_value='None')),  # Using SimpleImputer
    ('ordinal', OrdinalEncoder(categories=[ordinal_order[feature] for feature in ordinal_except_electrical]))
])
nominal_features = [feature for feature in categorical_features if feature not in ordinal_features]
categorical_transformer = Pipeline(steps=[
    ('impute_none', SimpleImputer(strategy='constant', fill_value='None')),  # Using SimpleImputer
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])
```

# Overview: The Target

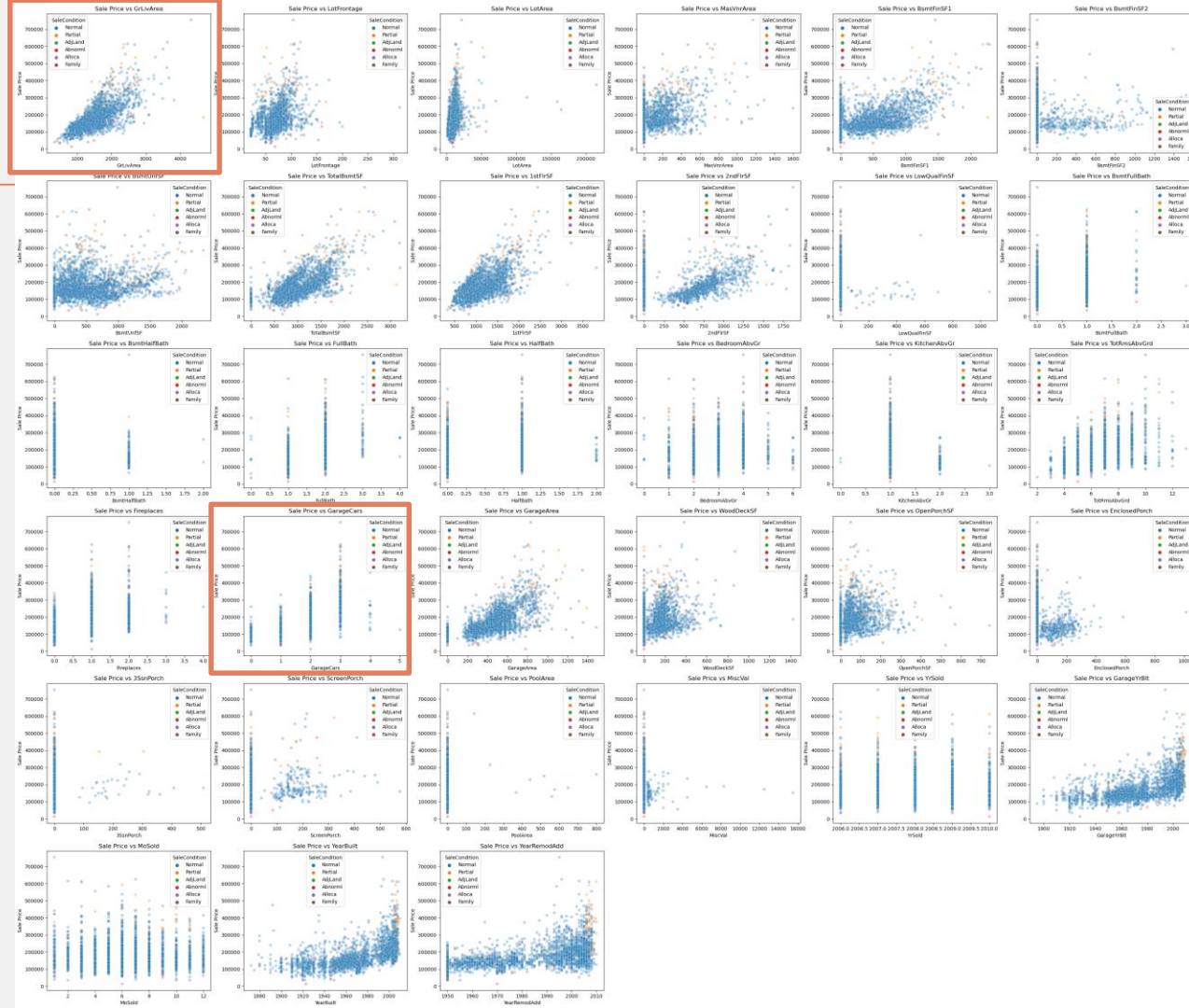
- Our target, the ‘Y’ value, is Sale Price. This is the value we are measuring, and eventually predicting. Using a standard distribution histogram, we can see that most houses have been sold at around $180,000. This is fairly typical, as most people cannot afford more expensive houses.

- However, it should also be kept in mind that there are numerous outliers in the data because of this. In fact, one can even be seen here, with a house being sold at over $700,000.

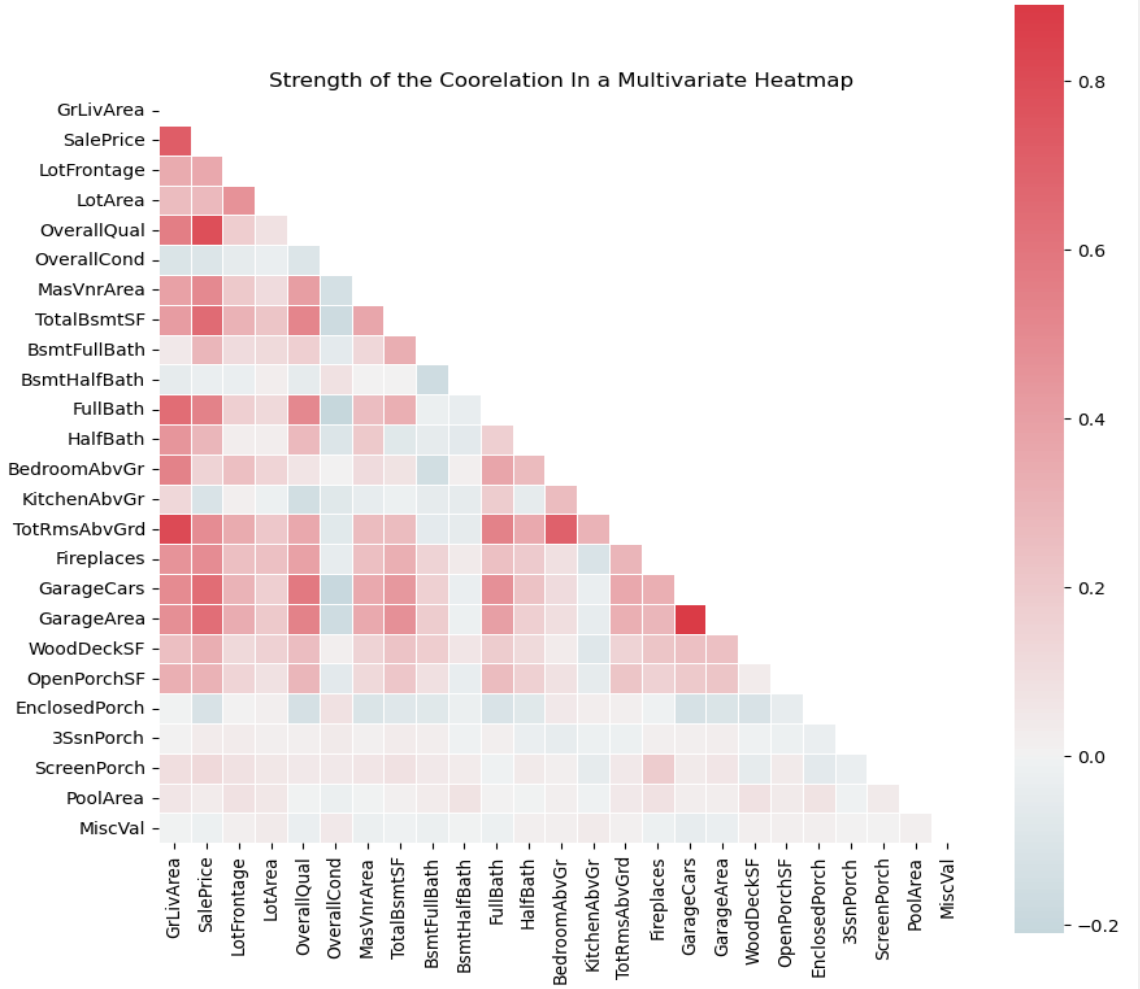


# Overview: Numeric

- Here is every numeric variable tested against the target, which is SalePrice. We can see many of the variables have some sort of correlation just from eyeing them, and we can also tell which might satisfy the features of linear regression.
- We can also see which variables are ordinal judging by the spacing of the plot points.
- Correlations can be extracted from this.



# Overview: Summarizing Correlations




- To the left, we see the numeric and ordinal values (assuming it is ranked).
- We can see that overall quality, ground living area, garage area, and others have higher correlation, while values like pool area and the “MiscVal” feature are of weaker strength.
- This can be used to compare feature importance, as well as collinearity for a multiple linear regression model.

# Multiple Linear Regression: As a Model

(Image Created Via Stable Diffusion)

- A MLR is used to determine the strength of relationships, as well as statistical significance. When used in machine learning, it is using the more than one x value (independent variable) to predict the y value (dependent variable)

- This allows for the inclusion of several predictors, which an OLS (Ordinary Least Squares) model cannot do. This would, therefore, determine the significance of each predictor as well.

- One must ensure features are linear to the target, there is constant variance, normality of errors, there is independence of errors, and that there is as little multi-collinearity as possible.

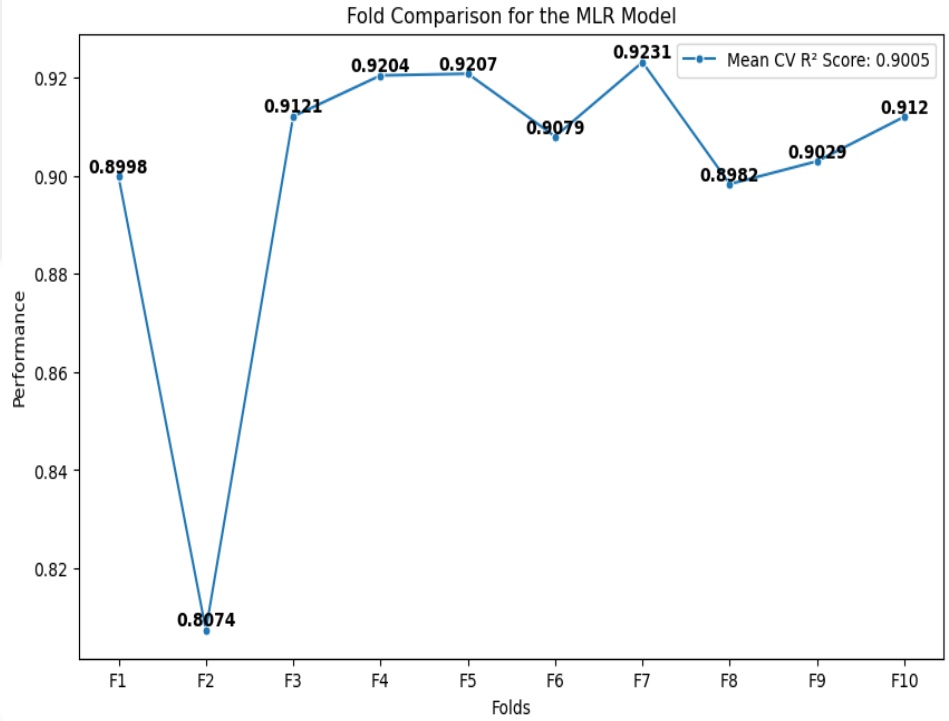
# MLR: The Transition From a Stats Model

- The OLS model, notably, has a higher R-squared, but it doesn't support cross-validation. This means: inconsistent coefficients, or an inflation of performance through overfitting.

- Cross-validation score is the performance of the model on unseen data. The data is split into groups, called folds, of hopefully equal size so that different models are being trained in order to improve accuracy and performance.

OLS Regression Results

| | | | |
|---|---|---|---|
| Dep. Variable: | y | R-squared: | 0.943 |
| Model: | OLS | Adj. R-squared: | 0.928 |
| Method: | Least Squares | F-statistic: | 61.90 |
| Date: | Sun, 11 Aug 2024 | Prob (F-statistic): | 0.00 |
| Time: | 18:04:14 | Log-Likelihood: | -28921. |
| No. Observations: | 2580 | AIC: | 5.894e+04 |
| Df Residuals: | 2033 | BIC: | 6.214e+04 |
| Df Model: | 546 | | |
| Covariance Type: | nonrobust | | |



# Regression Models: Lasso, Ridge, and Elastic Net

- Multiple-regression models are used as regularization techniques to deal with the problem of "overfitting", which occurs when the model is too closely adapted to the training data. When one or more coefficients are too high, the model's output becomes sensitive to minor alterations in the input data.

- Therefore, one has to use regularization techniques that work better with more features, and can deal with the issue of multicollinearity.

| Lasso | Ridge | Elastic Net |
|---|---|---|
| • More effective in situations where subset of features contribute more to the output.<br><br>• Ensures features remain small, at the cost of high bias and weak predictive power.<br><br>• Lasso selects one variable from a group of highly correlated variables; this can lead to suboptimal performance. | • Ridge's added penalty term (squared magnitude instead of Lasso's absolute value) to the least squares cost function shrinks the coefficients towards zero, but never at exactly zero like in Lasso.<br><br>• Ridge's even distribution of correlated variables makes it more suited to deal with multicollinearity, but it does not perform feature selection | • Combination of the Lasso and Ridge, with the addition of an alpha. Selects a subset of predictors that are correlated, but not redundant.<br><br>• Computationally expensive due to parameter tuning, and may fail to select relevant features if there are too many predictors. |

# Regressions: “Default” and “Tuned” Models

- The default model is the version of the model that has not been affected by parameter tuning, or anything related to it. It is the most basic cross-validation score one can get. In an ideal scenario, the default model is kept if it is superior in score to a tuned model.

- The tuned model is subject to parameter tuning, done through a grid search to find the best combination of "hyper-parameters" to find the highest accuracy. This is done manually with “ranges” of parameters, although for the regressors only the alphas were used.

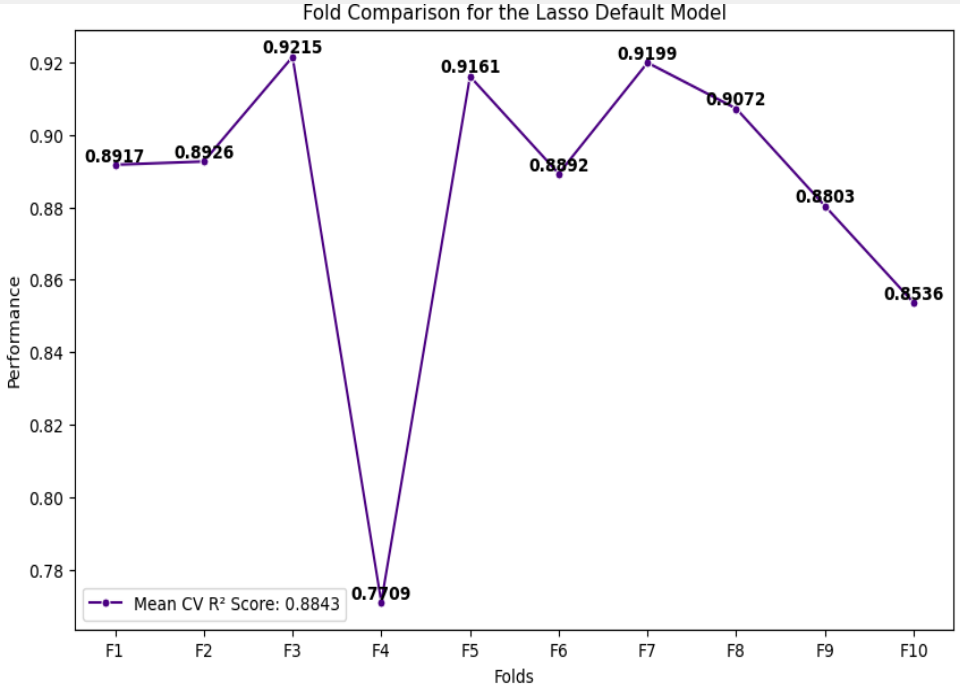
- Below is an example of parameter tuning. Note the difference for elastic net, which corresponds directly to the lasso at 1, but then shrinks to the ridge as it approaches 0. Python will find the “best parameters” that lead to the “best score” out of these. In our case, it was 40 (and nothing below that), .1, and .9, respectively.

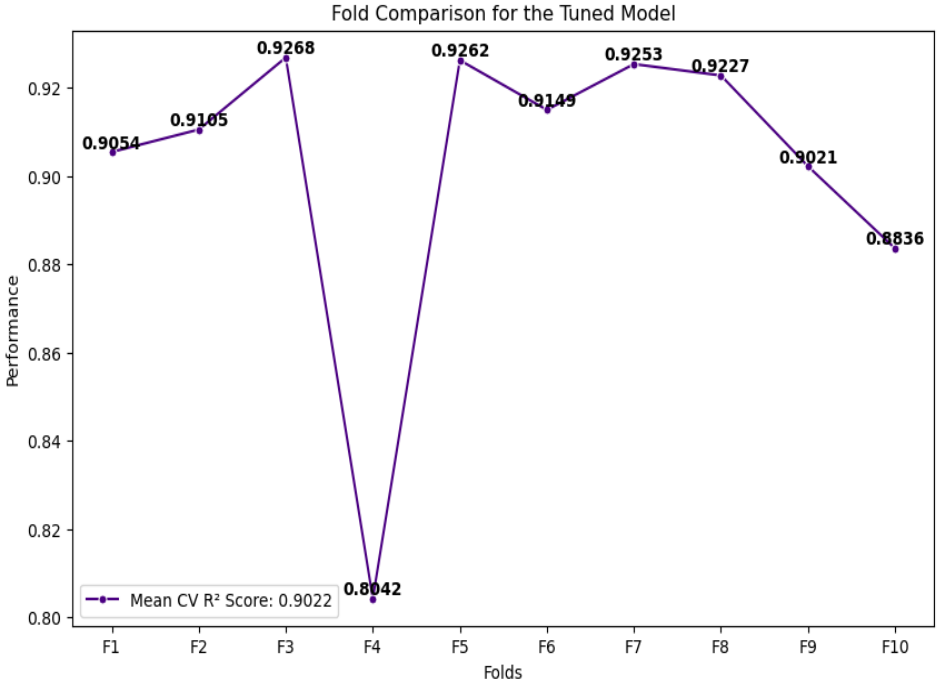
```
param_grids = {
    'Lasso': {'regressor__alpha': [40, 50, 60]},
    'Ridge': {'regressor__alpha': [40, 50, 60]},
    'ElasticNet': {'regressor__alpha': [0.01, 0.1, 1, 10], 'regressor__l1_ratio': [0.8, .9, .95]}
}
```

# Regressions: The Lasso

- The Lasso regression selects only the important features in a data set, while also reducing complexity. It is one of the fastest models out of the three, if not the fastest, and can handle larger number of features.

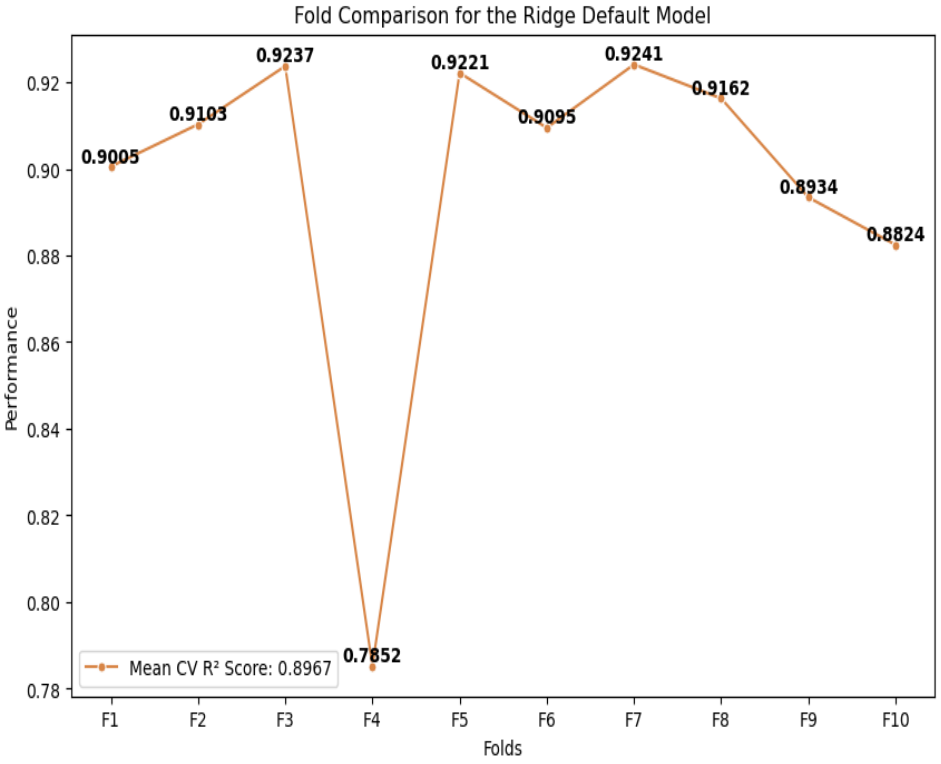
- The problem with the Lasso regression is that it is attempting to shrink the coefficient estimates to zero. This means that, because it will only retain one variable, a highly correlated model will end up having lower accuracy and a loss of information.

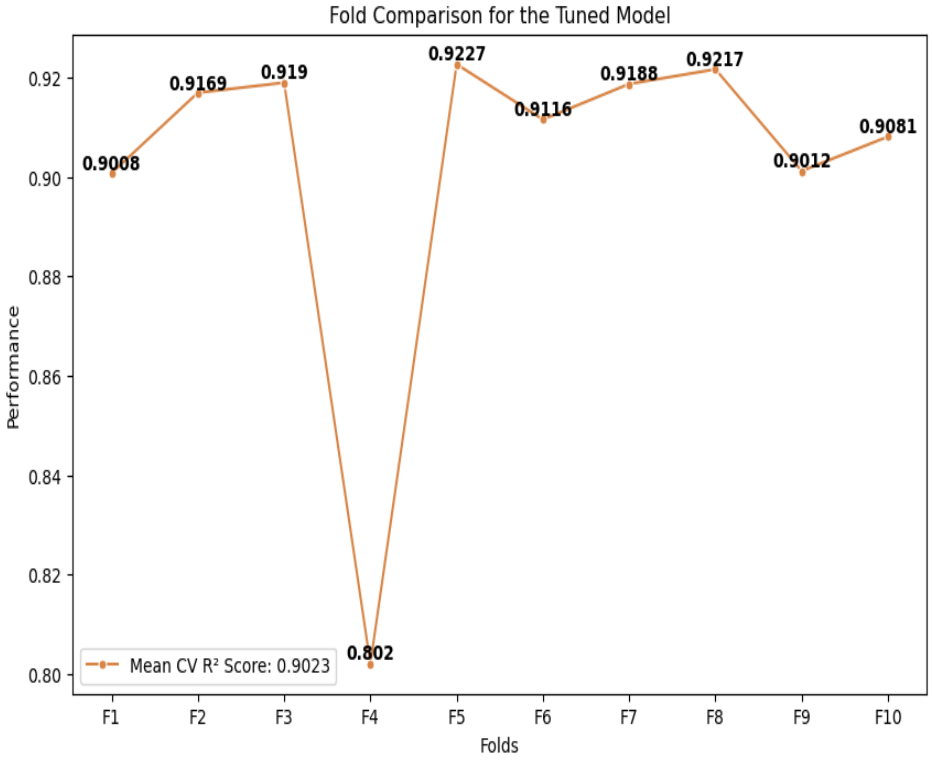


# Regressions: The Ridge

- Ridge regressions retain all the features in the model by only shrinking the coefficients, instead of setting them to zero like in the Lasso. This means there is no feature selection and less bias than the Lasso regression, at the cost of making the model more complex.

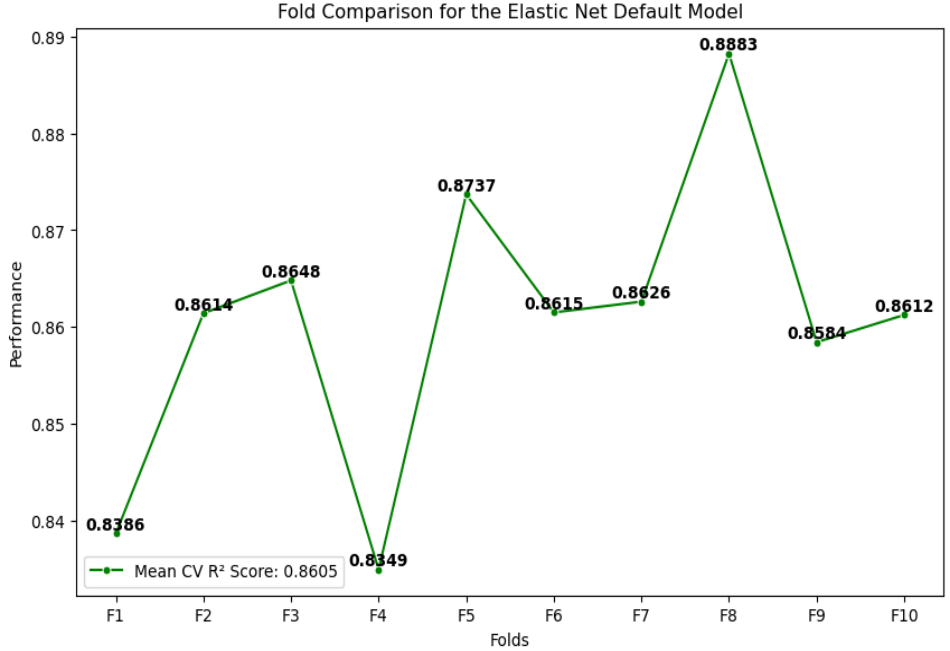


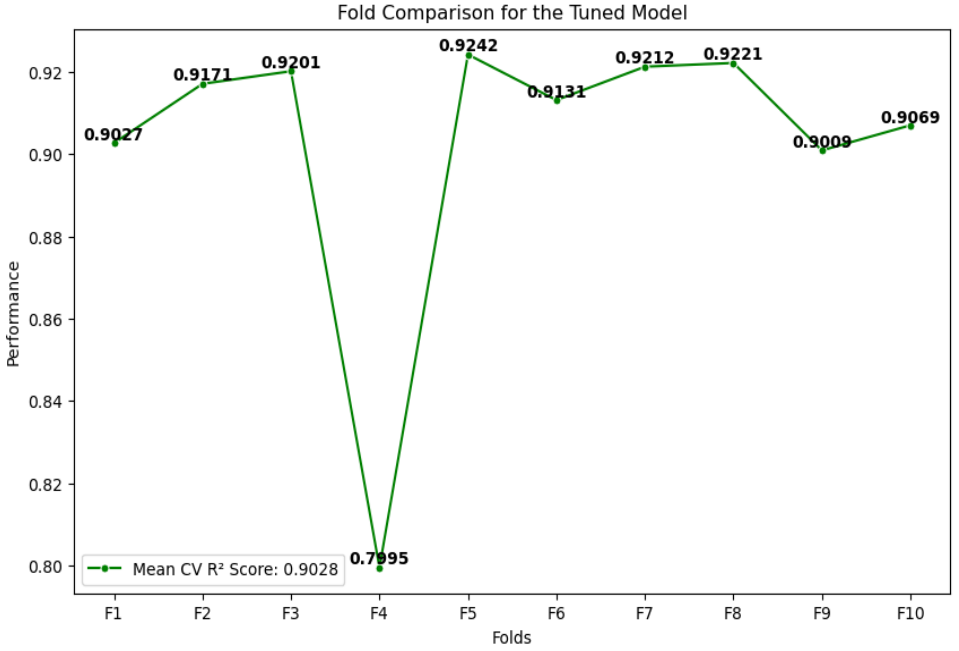



# Regressions: Elastic Net

- Elastic Net is a hybrid model, adding a penalty equivalent to the sum of the absolute values (L1-norm) of the coefficients and the squares (L2-norm) of the coefficients. It is used when the Lasso becomes too dependent on a particular variable.

- The problem, being that it is a hybrid, is that the Elastic Net is extremely computationally intensive.

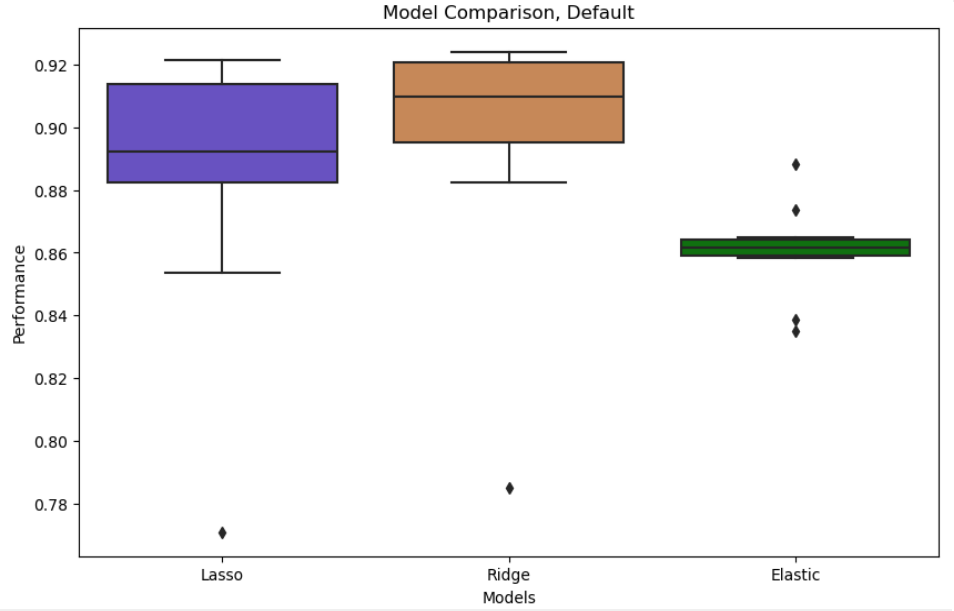


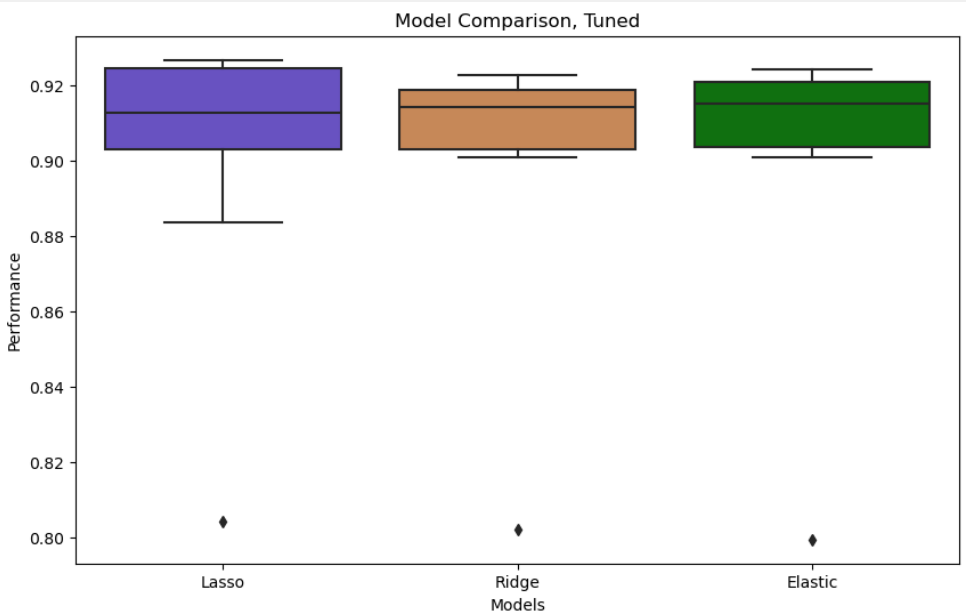



# Regressions: Model Summary

- When it comes to the default, the Elastic model had the lowest mean CV score, while the regression model had the highest. The tuned model, on the other hand, has the elastic having the highest mean CV score, but the lasso, overall, has some of the highest fold scores out of the three.

- It should also be noted that the  tuned Elastic model has one of the lowest fold score of the three, while the Lasso seems to have the greatest variation.



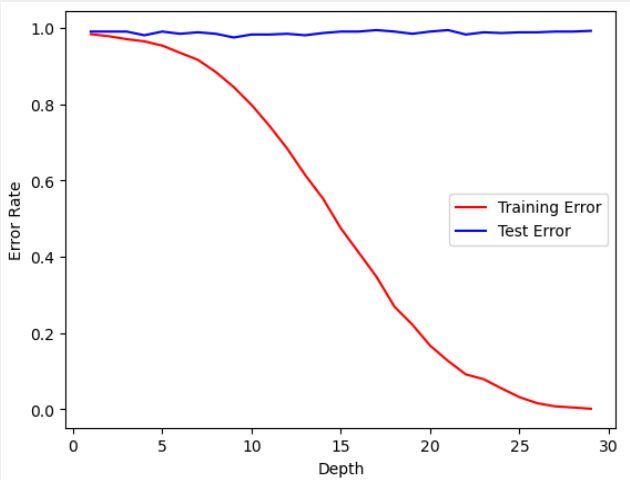



# Random Forest: As a Model

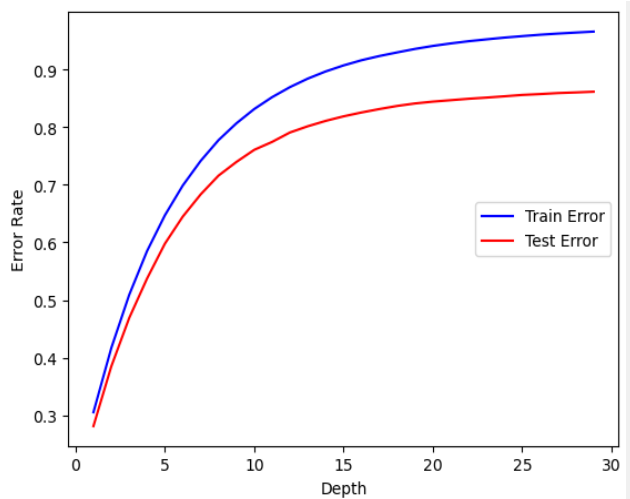
- A Random Forest model combines multiple Decision Trees into a single model. In this case, it would determine feature importance rather than, for example, a MLR's correlations. Feature importance is a degree of dependency on the variables being compared. The higher feature importance, the larger the effect it has on the model.

- A Decision Tree is a hierarchical structure where each "node" represents branches based on different conditions being satisfied. Decision trees are subject to "overfitting", which can lead to "poor predictive performance".

- Example of overfitting in a Decision Tree: High test error, but low training error. We want to prevent this.




- Example of when the train error is higher than the test error. This is caused by a sampling bias and substantial differences in the properties of the testing set.



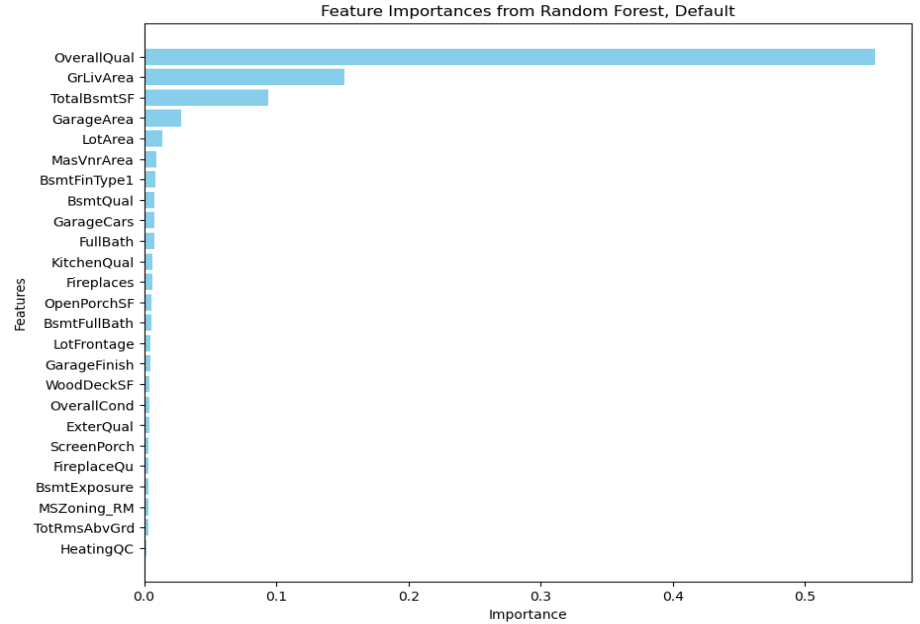
# RF: The Feature Importance

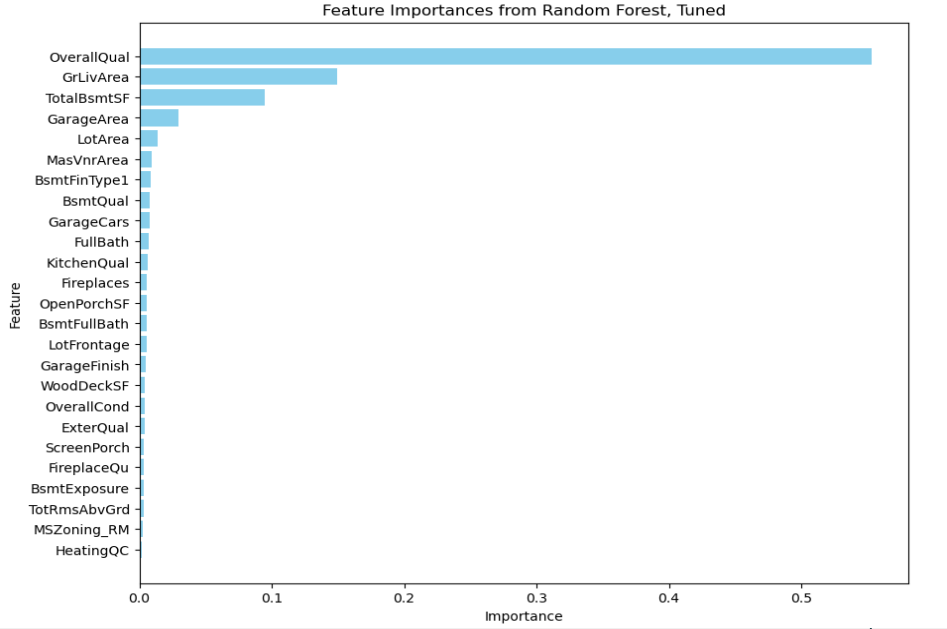
- With visually indistinct feature importances, the random forest models tell us not only a ranking of the features, but a model score. However, the model score, which is .886, is so close between both models it is almost not worth adjusting the default parameters.

- To the right, one can see just how close the importances are.

| | Feature | Importance |
|---|---|---|
| 4 | OverallQual | 5.526790e-01 |
| 1 | GrLivArea | 1.514606e-01 |
| 7 | TotalBsmtSF | 9.358243e-02 |
| 17 | GarageArea | 2.779386e-02 |
| 3 | LotArea | 1.365511e-02 |

| | Feature | Importance |
|---|---|---|
| 4 | OverallQual | 5.528583e-01 |
| 1 | GrLivArea | 1.492397e-01 |
| 7 | TotalBsmtSF | 9.456192e-02 |
| 17 | GarageArea | 2.934008e-02 |
| 3 | LotArea | 1.409694e-02 |



# Gradient Boosting

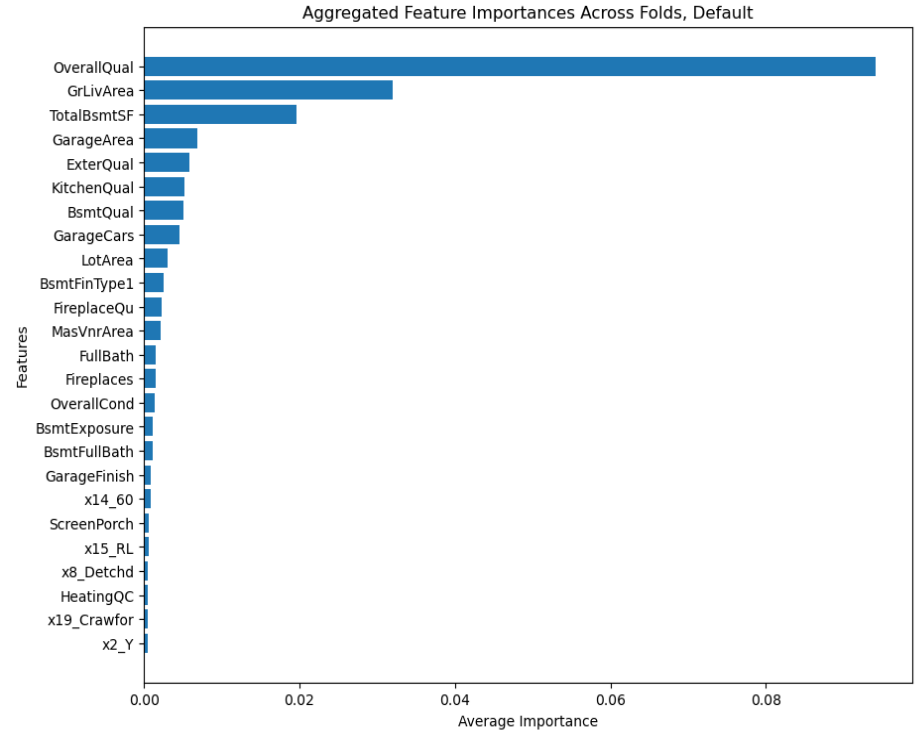
- Gradient boosting combines weaker models (usually decision trees), and adds said models sequentially, where each "addition" tries to make up for the errors made by the previous ones. The final prediction is an aggregate sum of the individual predictions.

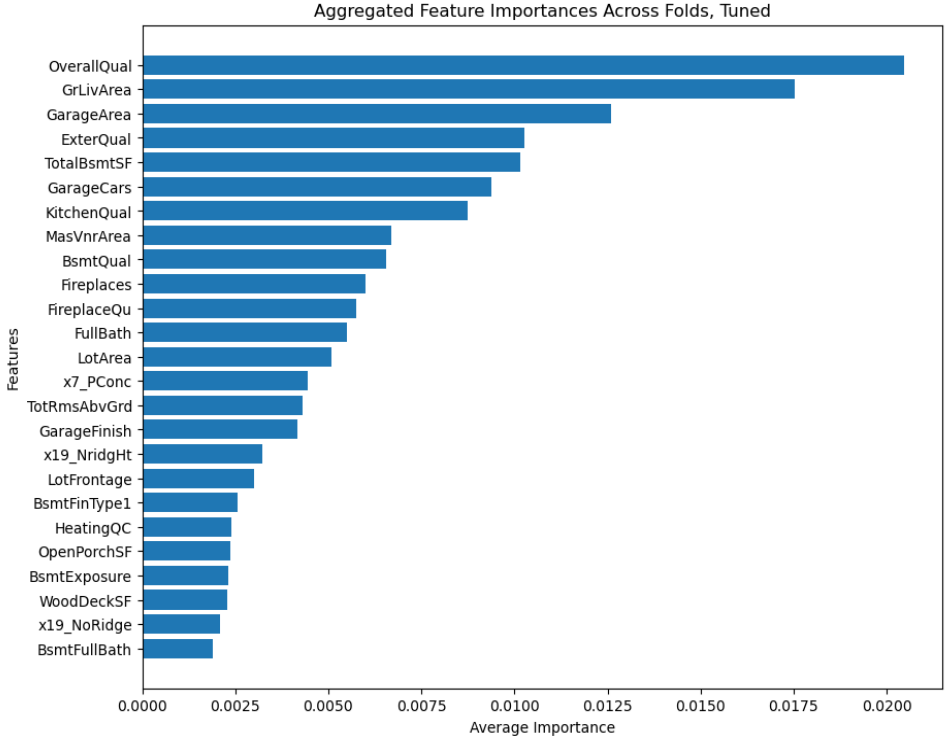
- Because of the sequencing, the gradient boosting model reduces bias with the addition of each additional leaner. This further allows for greater accuracy, especially on larger data sets.

(Image Created Via Stable Diffusion)

# GB: The Booster

- The Gradient Boosting model is an interest one as its tuned version is had the best model performance out of all models, but the feature importances had a stark contrasts to each other. “Rank” should be looked at instead of the “strength”, as the GB will start taking into account One Hot Encoded values, which were a part of those weaker models.



# Summary: Results

- The models themselves all have their strengths and weaknesses, but the results bring out which models may be preferred over the other. Of the regression models, the Elastic Net has the highest average CV score, although not by much of a margin.

- On the other hand, the Gradient Boosting model is the strongest model of the six, while the Random Forest model is the weakest. The MLR model was the only model that actually performed worse when subject to a tuning, so it was left untouched.

| MLR Default Score | Lasso Default Score | Ridge Default Score | EN Default Score | RF Default Score | GB Default Score |
|---|---|---|---|---|---|
| 0.9005 | 0.8844 | 0.8968 | 0.8605 | 0.8863 | 0.8947 |

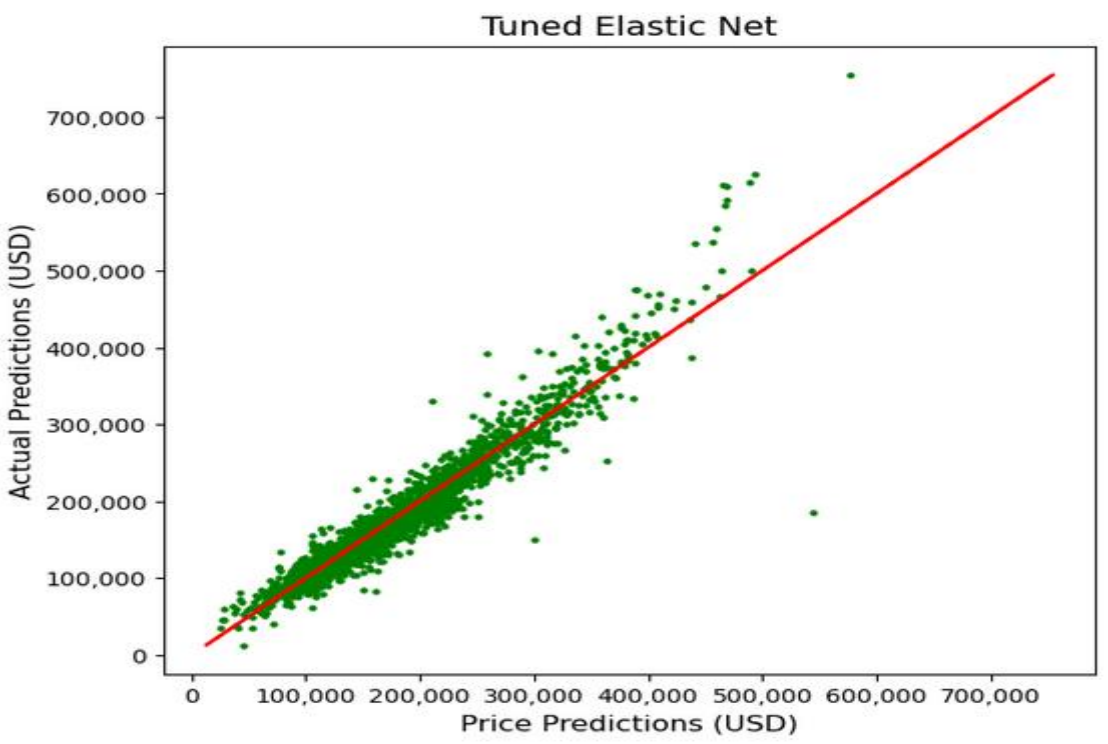
| MLR Tuned Score | Lasso Tuned Score | Ridge Tuned Score | EN Tuned Score | RF Tune Score | GB Tuned Score |
|---|---|---|---|---|---|
| N/A | 0.9022 | 0.9023 | 0.9028 | 0.8864 | 0.9056 |

# Final Thoughts: Linear Models

- The regression models are an interesting case in that they all not only are varied, but all have their advantages and disadvantages. Some of them are specifically used in certain situations, but after using four of them, there's no one that actually stands out well-above the rest.

- Therefore, it ends up turning into a "pick-and-choose" situation, where one would select regression models that suites the situation to compare them alongside other regression models. This is fine because, in our experience, these models are: compact, easy to use, and similar enough to each other that they can be compared with ease.

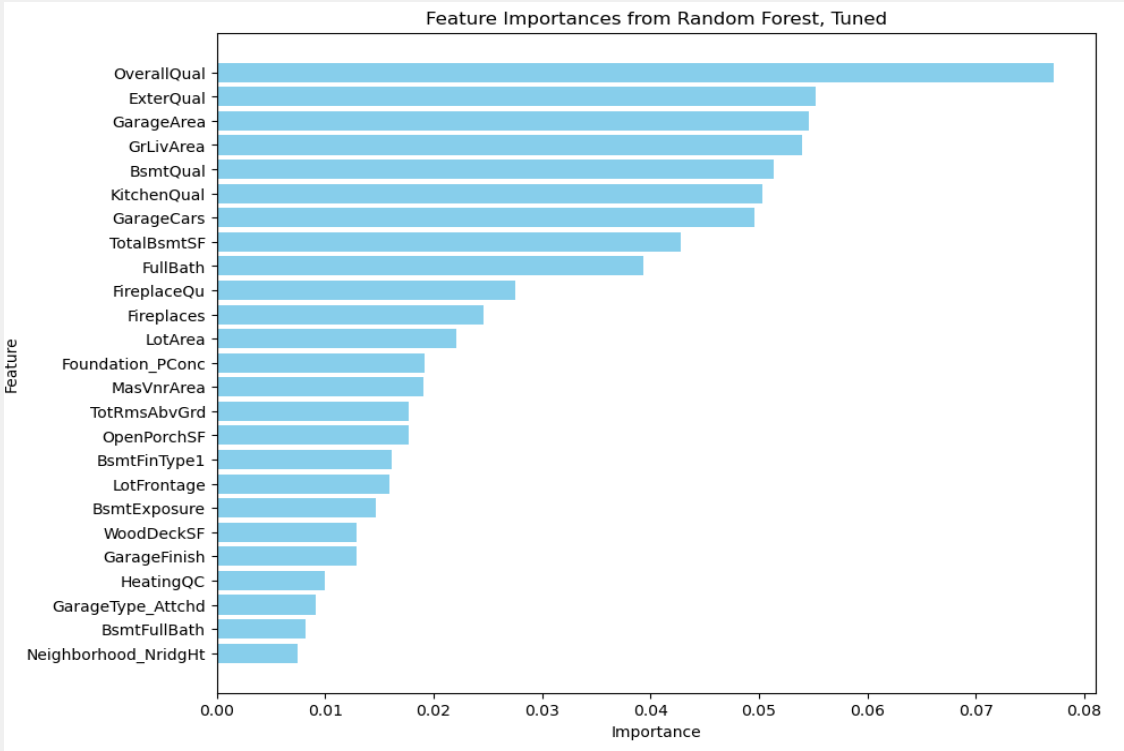
- Take this prediction for example from the Elastic Net. This is nearly identical visually across the regressions, so it's pointless to show a different one.



# Final Thoughts: Random Forest

- The Random Forest is an interesting case as we found it to be the most unique out of the models presented. Although not shown, its construction method is visually distinct and that allows for greater parameter tuning.

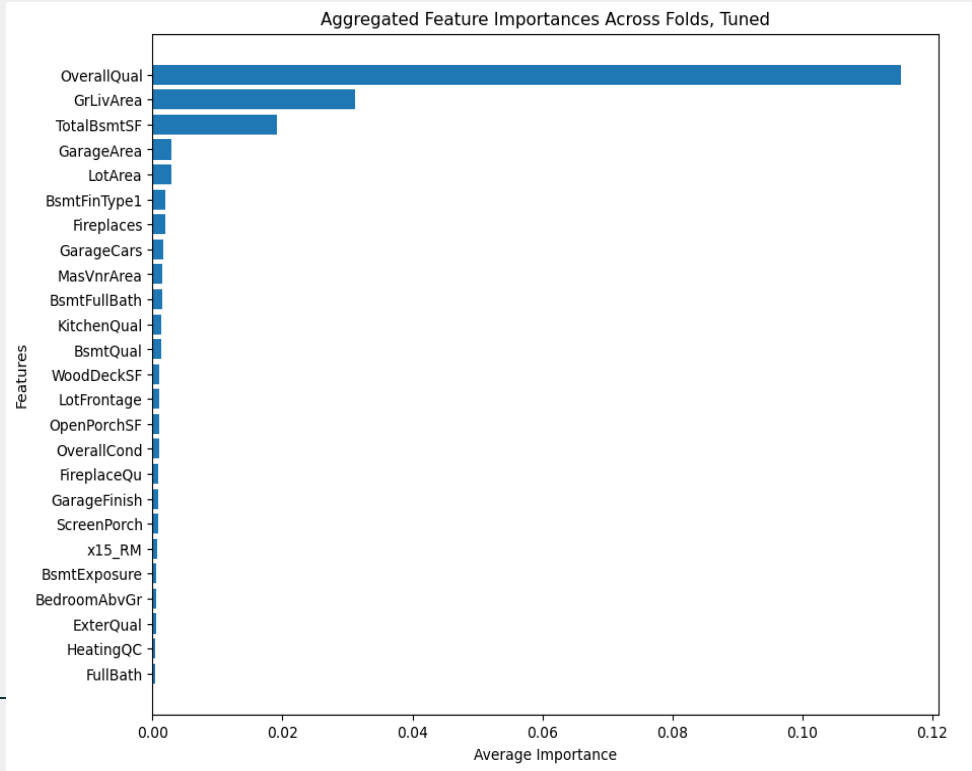
- Yet even with that ability, it happened to be the weakest of all models. Worst of all, it was the most computationally expansive of all models. This is due to the parameter of “feature”, a parameter that has the chance to completely change the results. For example, to the right are the RF results if “feature” was square root instead of none. It would have a lower score, but everything would have higher importance. Note its similarity in appearance to the GB model.



# Final Thoughts: Gradient Boosting

- A similar model to the Random Forest, the Gradient Boosting model was the best model in performance out of the six. It was computationally impressive (even with a “none” parameter, it was still faster than the RF), and allows one to extract similar results from a RF model.

- Included to the right is if square root was not used for the feature parameter, which shows just how similar the importance would be to a RF model. Its average importance is still weaker, but the spread is almost identical. However, this version of the model had a much weaker score than the version that was chosen.



# Areas For Improvement

## Modeling

1. **Predictive Reliability:** The score for all models was lower than desired. Ideally, they should all be above .90.
2. **Beyond Score:** The goal of the project was to find the model with the best score. While we did dabble in some other results, ideally one would use the models to truly find out the limits of what they're capable of.
3. **Other Models:** Other models can be used to address various issues. This includes classification modeling (regressors were used), so-called XGboosting and AdaBoosting, and bootstrap aggregating. Additionally the parameters can be adjusted.

## Data

1. **Feature Selection:** We never actually used the MLR model for feature selection, which can be used to further tune the model.
2. **Pipeline Changes:** The data was edited in pipelines, done in different ways throughout the models to match their specifications. Obviously, the feature engineering was not perfect and can be changed to better adapt to certain models. Just for an example, the mean was used to fill null values for the numeric features. Sometimes this isn't ideal.
3. **Feature Engineering :** Some features were changed to account for their uselessness in the grand scheme of the data. This can be done for other features, such as with total baths.

# Questions? Comments? Suggestions?

- Massive thank you to my mentor Vinod, who pushed me well beyond what I knew to complete this project.