Project 1 Report: Predict the Housing Prices in Ames

1. Technical Details

1.1 Data Preprocessing

I began data preprocessing by removing inconsistent rows. Specifically, I eliminated cases where the year the house was built was later than the year it was remodeled, as well as instances where the Masonry veneer area was 0 but the Masonry veneer type was not listed as NA. These rows were removed only from the training data, as the test data needed to remain intact for accurate predictions.

Next, I dropped low-information and unbalanced columns, including *‘PID’, ‘Street’, ‘Utilities’, ‘Condition_2’, ‘Roof_Matl’, ‘Heating’, ‘Pool_QC’, ‘Misc_Feature’, ‘Low_Qual_Fin_SF’, ‘Pool_Area’, ‘Longitude’*, and *‘Latitude’*. To handle missing data, I replaced missing values in the *‘Garage_Yr_Blt’* column with zero and in the *‘Mas_Vnr_Type’* column with *‘No_MasVnr’*. For outliers, I applied the winsorization technique to several columns, including *‘Lot_Frontage’, ‘Lot_Area’, ‘Mas_Vnr_Area’, ‘BsmtFin_SF_2’, ‘Bsmt_Unf_SF’, ‘Total_Bsmt_SF’, ‘Second_Flr_SF’, ‘First_Flr_SF’, ‘Gr_Liv_Area’, ‘Garage_Area’, ‘Wood_Deck_SF’, ‘Open_Porch_SF’, ‘Enclosed_Porch’, ‘Three_season_porch’*, *‘Screen_Porch’*, and *‘Misc_Val’*. Winsorization was applied to the test data using the 95th percentile from the training data.

For the Elastic Net model, I removed highly correlated columns (Pearson correlation > 0.7), which included *‘First_Flr_SF’, ‘TotRms_AbvGrd’*, and *‘Garage_Cars’*. I also standardized numerical variables such as *‘Lot_Frontage’, ‘Lot_Area’, ‘Mas_Vnr_Area’, ‘BsmtFin_SF_1’, ‘BsmtFin_SF_2’, ‘Bsmt_Unf_SF’, ‘Total_Bsmt_SF’, ‘Second_Flr_SF’, ‘Gr_Liv_Area’, ‘Bsmt_Full_Bath’, ‘Bsmt_Half_Bath’, ‘Full_Bath’, ‘Half_Bath’, ‘Bedroom_AbvGr’, ‘Kitchen_AbvGr’, ‘Fireplaces’, ‘Garage_Area’, ‘Wood_Deck_SF’, ‘Open_Porch_SF’, ‘Enclosed_Porch’, ‘Three_season_porch’, ‘Screen_Porch’,* and *‘Misc_Val’*. Certain variables, like *‘Sale_Price’, ‘Year_Built’, ‘Year_Remod_Add’, ‘Garage_Yr_Blt’, ‘Year_Sold’*, and *‘Mo_Sold’*, Ire not standardized.

The last step was to handle categorical variables. I used label encoding for variables with K=2 categories, such as *'Central_Air'*. For variables with K>2 categories, I used one-hot encoding technique with K dummy variables for both ElasticNet and CatBoost. These variables included *'MS_SubClass', 'MS_Zoning', 'Alley', 'Lot_Shape', 'Land_Contour', 'Lot_Config', 'Land_Slope', 'Neighborhood', 'Condition_1', 'Bldg_Type', 'House_Style', 'Overall_Qual', 'Overall_Cond', 'Roof_Style', 'Exterior_1st', 'Exterior_2nd', 'Mas_Vnr_Type', 'Exter_Qual', 'Exter_Cond', 'Foundation', 'Bsmt_Qual', 'Bsmt_Cond', 'Bsmt_Exposure', 'BsmtFin_Type_1', 'BsmtFin_Type_2', 'Heating_QC', 'Electrical', 'Kitchen_Qual', 'Functional', 'Fireplace_Qu', 'Garage_Type', 'Garage_Finish', 'Garage_Qual', 'Garage_Cond', 'Paved_Drive', 'Fence', 'Sale_Type',* and *'Sale_Condition'*.

1.2 Model Implementation

For the Linear Regression model, I utilized the scikit-learn library to get the implementations for Lasso, Ridge & ElasticNet. I tried fitting a Lasso & Ridge model by themselves with default parameters as well as fitting a Lasso to select variables and then a Ridge using those variables for the predictions but both these methods failed to get us below the threshold. Our final Linear Regression model was an ElasticNet model with parameters: (alpha=0.0001, l1_ratio=0.5, max_iter=10000) and using standardized data. This model was sufficient to get our RMSE values under the threshold.

For the tree-based model, I used the scikit-learn for Random Forest, as well as the XGBoost & CatBoost libraries. First, I fit a Random Forest model with 1000 estimators, but I couldn't get below the threshold. I found that an XGBoost model with parameters: (n_estimators=5000, max_depth=6, eta=0.05, subsample=0.5), was able to get below the threshold. I also fitted a CatBoost and allowed it to choose its own parameters and I found that this model performed better and also trained & predicted faster than the XGBboost model. The parameters chosen by the CatBoost model Ire: (iterations: 1000, bootstrap_type: MVS, max_leaves: 64, learning_rate: 0.0458, depth: 6).

2. Performance Metrics

The models were trained & tested on the Google Colab Python3 runtime using the CPU Hardware accelerator/default setting. Based on a Ib search it is an Intel Xeon CPU @ 2.3 Ghz with 13 GB of RAM.
These are the performance metrics for the **ElasticNet Model**, with parameters (alpha=0.0001, l1_ratio=0.5, max_iter=10000):

```
Starting Fold 1...
Fold 1 RMSE: 0.1212
Fold 1 Runtime: 3.761 seconds
----------------------------------
Starting Fold 2...
Fold 2 RMSE: 0.1187
Fold 2 Runtime: 6.326 seconds
----------------------------------
Starting Fold 3...
Fold 3 RMSE: 0.1170
Fold 3 Runtime: 2.412 seconds
----------------------------------
Starting Fold 4...
Fold 4 RMSE: 0.1206
Fold 4 Runtime: 2.788 seconds
----------------------------------
Starting Fold 5...
Fold 5 RMSE: 0.1120
Fold 5 Runtime: 5.995 seconds
----------------------------------
```

```
Starting Fold 6...
Fold 6 RMSE: 0.1332
Fold 6 Runtime: 6.718 seconds
----------------------------------
Starting Fold 7...
Fold 7 RMSE: 0.1296
Fold 7 Runtime: 3.508 seconds
----------------------------------
Starting Fold 8...
Fold 8 RMSE: 0.1205
Fold 8 Runtime: 2.918 seconds
----------------------------------
Starting Fold 9...
Fold 9 RMSE: 0.1309
Fold 9 Runtime: 4.120 seconds
----------------------------------
Starting Fold 10...
Fold 10 RMSE: 0.1251
Fold 10 Runtime: 2.597 seconds
----------------------------------
```

These are the performance metrics for the **CatBoost Model**, with all default parameters(e.g. Letting the model decide them by itself):

```
Starting Fold 1...
Fold 1 RMSE: 0.1117
Fold 1 Runtime: 14.707 seconds
----------------------------------
Starting Fold 2...
Fold 2 RMSE: 0.1156
Fold 2 Runtime: 16.947 seconds
----------------------------------
Starting Fold 3...
Fold 3 RMSE: 0.1133
Fold 3 Runtime: 17.522 seconds
----------------------------------
Starting Fold 4...
Fold 4 RMSE: 0.1147
Fold 4 Runtime: 16.048 seconds
----------------------------------
Starting Fold 5...
Fold 5 RMSE: 0.1062
Fold 5 Runtime: 15.854 seconds
----------------------------------
```

```
Starting Fold 6...
Fold 6 RMSE: 0.1270
Fold 6 Runtime: 14.614 seconds
----------------------------------
Starting Fold 7...
Fold 7 RMSE: 0.1300
Fold 7 Runtime: 17.464 seconds
----------------------------------
Starting Fold 8...
Fold 8 RMSE: 0.1231
Fold 8 Runtime: 16.487 seconds
----------------------------------
Starting Fold 9...
Fold 9 RMSE: 0.1303
Fold 9 Runtime: 15.038 seconds
----------------------------------
Starting Fold 10...
Fold 10 RMSE: 0.1191
Fold 10 Runtime: 10.152 seconds
----------------------------------
```