**Final Project House Prices Prediction**

**Hint:**

Some imports haven`t been used.

**Imports:**

Make sure to download `Xgboost`, `lightgbm`, `sklearn`, `keras`.

We are going to use a lot of libraries for this task.

**Step:**

* **Load the datasets:**
* Start by loading the datasets you’ll interact with. Make sure you also open the “data\_description.txt” file to get a better understanding of the data.
* After opening the “data\_description.txt” file we will see that both our test and training set have the same sizes. In total, we have 1460 observations with the relevant “SalePrice” we can use for training our models. There are 80 variables we can use.
* **Data Cleaning:**
* We look for missing values in both the training and test set.
* First, after looking at the data description file, we change the data type of two variables: “MSSubClass” and “MoSold” as these are categorical variables and not integers.
* Cleaning categorical variables: We impute the mode of the “Neighborhood” and “MSSubClass” when we can do so. Else, we impute the column’s own mode. The logic behind this is that houses in the same neighborhood and from the same class might have the same characteristics as most houses from the same class in their area.
* One last categorical variable needs to be cleaner: “PoolQC,” here we just add a new entry called “Other” as most values are missing.
* Cleaning continuous variables: In the case of continuous variables, we impute the values with the mean of the group that makes the most sense.
* E.g., for the “LotFrontage,” we group the “LotFrontage” of the non-missing values by “LotConfig”, compute the mean there on and then apply this mean to the matching groups of missing values.
* For the “GarageYrBlt” variable, we impute the “YearBuilt” of the house as it may be that the garage and home were built at the same time.
* Finally, we fill all leftovers with the column’s own mean. And we check which columns still have missing values.
* Only, “SalePrice” is missing which is exactly what we want as we are trying to predict the home’s sale price.
* **Feature engineering:**
* We will add some features that could improve our models’ performance. First, we will add some “age” features as the age might by more relevant than the “year”. It is easier to see the relative difference between a 10 and a 20-year-old house than between a house that was built in 2010 and one built in 2020.
* **Plotting:**
* Now that we have cleaned our data, we can plot it in order to visually look for correlations, outliers, or any interesting point.
* Categorical variables versus “SalePrice”: ("MSSubClass", "MSZoning", "HouseStyle","CentralAir", "PoolQC", "SaleType")
* Continuous variables versus “SalePrice”: ("1stFlrSF", "LotArea", "OverallQual", "OverallCond", "YearBuilt","ExterQual", "YrSold")
* **Data transformation:**
* The last step in building our models is transforming the data into a digestible format.
* Generate dummies: An algorithm has no clues what a “SaleType” of “WD” means. Instead, it understands if there is a 1 or 0 in the “SaleType\_WD column.
* Split data set into test and train: We combined the train and test set into a big data set. Now we are going to split it.
* Set “Id” column as index: We do not want to use the “Id” column in our model. We still want to keep it for our predictions later.
* Scale the data: Most algorithms rely on the distance between the predicted value and real value in order to improve their predictions. It is thus best practice to scale our data so as to make sure that these distances are not impacted by the scale of a variable. E.g., errors from the surface of a house and its “OverallCond” don’t have the same scale at all.
* Feature selection: Sometimes, you want to reduce the numbers of features your model uses. One simple method is to use the variance threshold and remove all the variables that do not change enough. `**We commented this feature selection section as it decreased our model’s performance**. `
* **Modelling:**
* Finally, we start by splitting our training set into a train and test set. So that we can measure our results instantly. We also set the random\_state to have reproducible results.
* In the next part, we will run the models listed here under and stack the best ones into a stacked regression. Most of the comments are included in the code chunks.
* Lasso (LassoRegressor RMSE: 0.14840)
* XGBoost (XGBoostRegressor RMSE: 0.14684)
* ANN with Keras (ANNRegressor RMSE: 0.16085)
* LightGBM (LGBMRegressor RMSE: 0.14034)
* SVM (SVRRegressor RMSE: 0.18398)
* Stacked Regression: We take our best models and stack them into a stacked regression. We then submit this model’s CSV file. (StackedRegressor RMSE: 0.13925)