House Price Predictor Challenge Report

[**A. Project Overview 2**](#_heading=h.1fob9te)

[**B. Construction of the Alpha Model 2**](#_heading=h.3znysh7)

[B.1. Exploratory Data Analysis (EDA) 2](#_heading=h.2et92p0)

[B.2 Data Preprocessing 2](#_heading=h.tyjcwt)

[B.2.1. Handling Missing Values 2](#_heading=h.3dy6vkm)

[B.2.2. Standardizing Numerical Features 3](#_heading=h.1t3h5sf)

[B.2.3. Encoding Categorical Features 3](#_heading=h.4d34og8)

[B.2.4. Feature Selection and Engineering 3](#_heading=h.2s8eyo1)

[B.2.5. Dimensionality Reduction with PCA 3](#_heading=h.17dp8vu)

[B.3. Data Subgrouping 4](#_heading=h.3rdcrjn)

[B.3.1. K-Means Clustering 4](#_heading=h.26in1rg)

[B.3.2. K-Nearest Neighbors Classifier 4](#_heading=h.lnxbz9)

[B.4. Prediction Model Training and Validation 4](#_heading=h.35nkun2)

[B.4.1. Model Selection 5](#_heading=h.1ksv4uv)

[General Approach 5](#_heading=h.44sinio)

[Models 5](#_heading=h.2jxsxqh)

[B.4.2. Model Training Workflow 6](#_heading=h.z337ya)

[General Workflow Intuition 8](#_heading=h.3j2qqm3)

[Initialization 8](#_heading=h.1y810tw)

[Training and Validation of Individual Models 8](#_heading=h.4i7ojhp)

[Ensemble Model Training 8](#_heading=h.2xcytpi)

[**C. Revisions for the Final Model 9**](#_heading=h.1ci93xb)

[C.1. Train and Test Data Processing 9](#_heading=h.3whwml4)

[C.2. Prediction Model Training 10](#_heading=h.2bn6wsx)

[Functions 10](#_heading=h.qsh70q)

[C.3 Evaluating Models 11](#_heading=h.3as4poj)

[C.2.4. Evaluating Models 11](#_heading=h.1pxezwc)

[Evaluating Training Process 11](#_heading=h.49x2ik5)

[Individual Models 11](#_heading=h.2p2csry)

[Ensemble Models 11](#_heading=h.147n2zr)

[Overall Evaluation: 11](#_heading=h.3o7alnk)

[Evaluating Trained Models Predictions 12](#_heading=h.23ckvvd)

[Functions 12](#_heading=h.ihv636)

[Grouped Dataset Stacking Regressor: Individual Group MSE 12](#_heading=h.32hioqz)

[Entire Dataset Stacking Regressor: Entire Dataset MSE 12](#_heading=h.1hmsyys)

[Grouped Dataset Stacking Regressor: Overall MSE 13](#_heading=h.41mghml)

[Overall Evaluation 13](#_heading=h.2grqrue)

[**D. Conclusion 13**](#_heading=h.vx1227)

[**E. References 13**](#_heading=h.1v1yuxt)

# A. PROJECT OVERVIEW

The goal of this project is to predict the individual house prices in Ames, Iowa given train and test data originally containing 79 features. Data processing includes imputations of missing values, numerical data standardization, categorical data encoding, feature selection and engineering based on the correlation coefficients between features, and dimensionality reduction using primary component analysis (PCA). Regarding subgrouping the data, k-means clustering is used to divide the data, and k-nearest neighbors with grid-search cross-validation is used to predict the clusters for the test data. A variety of ML algorithms including stochastic gradient descent, decision tree regressor, linear regression, lasso and ridge regularization, support vector machine, and neural network models were applied for an overall MSE using the stacking regressor of 707131939.38.

# B. CONSTRUCTION OF THE ALPHA MODEL

## B.1. Exploratory Data Analysis (EDA)

Modeled after the provided examples in Lecture 1 and Homework 1, exploratory data analysis of the train data was performed. The train data set had shape (1022, 82), indicating 1022 observations and 82 features, and contained ‘float64,’ ‘object,’ and ‘int64’ data types. Using the pandas describe() function, a statistical overview of the numerical data was obtained including the mean, standard deviation, and quartiles. Histograms of the overall distribution were also plotted for each numerical feature. To visualize the categorical data, a dictionary was created with the keys being the categorical feature names and the values being the possible values for the categorical feature.

To determine which features were strongly correlated to the target ‘SalePrice,’ the pandas corr() function was applied to the numerical features to construct a correlation matrix, sorted by the strongest correlation to ‘SalePrice’ in ascending order. The most strongly correlated features to ‘SalePrice’ included ‘OverallQual’ (0.790), ‘GrLivArea’ (0.720), ‘GarageCars’ (0.634), ‘GarageArea’ (0.626), and ‘TotalBsmtSF’ (0.613). The function **get\_top\_n\_pairs** was implemented to tabulate the names of feature pairs with the strongest correlation to each other and their correlation coefficient. This revealed that ‘Id’ and ‘Unnamed: 0’ have a correlation coefficient of 1.000, indicating that both features contain the same values to identify individual observations.

## B.2 Data Preprocessing

### B.2.1. Handling Missing Values

In the alpha model, data preprocessing focused on the training data. The column ‘Unnamed: 0’ containing the same values as the ID was dropped to remove redundancy. To determine how to handle features with missing data, two functions were written. The first, **print\_null\_features(X)**, prints each feature in dataframe X with the percent of missing observations. The second function, **null\_columns(X)**, returns a dictionary with features missing data as the keys and the percent of missing data as the values. In a subsequent for loop, **null\_columns** is applied to determine which features are missing more than 80% of their data. These features (‘Alley,’ ‘PoolQC,’ ‘Fence,’ ‘MiscFeature’) are appended to a list and iteratively dropped from the train data.

The training data was split to separate numerical features from categorical features. Regarding the numerical features, applying **print\_null\_features** indicated the missing numerical features were ‘LotFrontage,’ ‘GarbageYrBlt,’ and ‘MasVnrArea.’ As ‘LotFrontage’ was missing a large portion of data (17.6%), K-S tests were performed with imputations using the median and mean. The p-values of the K-S tests were less than 0.05, meaning the null hypotheses, the distributions for the missing values removed vs. values imputed with the median or mean are identical, were rejected. Though this indicated that the missing values should be dropped, dropping the rows with missing ‘LotFrontage’ values would result in losing over 10% of the total observations and could significantly impact the overall model accuracy. For this reason, the missing ‘LotFrontage’ values were imputed with the median; the values for the remaining numerical features with missing observations were also imputed with the median.

### B.2.2. Standardizing Numerical Features

‘SalePrice’ and ‘Id’ were removed from the train numerical data so the target and ID observations would not be standardized. Sklearn’s **MinMaxScaler()** and **StandardScaler()** were both used for fitting and transforming the numerical features dataframe. After evaluating each, standardization was chosen to convert the numerical data in different scales.

### B.2.3. Encoding Categorical Features

The categorical data was separated from the numerical data. For each feature in the categorical features, all the data besides the current feature was stored in a variable. The group names and number of groups in the current feature were also stored in variables. If the number of groups was two, meaning the feature was already binary, the one group in the data was converted to 0 and the other to 1. If the feature was not already binary, sklearn’s **LabelEncoder()** and **LabelBinarizer()** were used to convert each group in the current feature into a new binary feature, increasing the overall dimensionality. The encoded data from **LabelBinarizer**() was converted into a dataframe with a new heading, the current feature name and the group name, and then concatenated to the stored dataframe without the current feature. The steps in this for loop were repeated to encode all categorical features.

### B.2.4. Feature Selection and Engineering

The standardized numerical features, encoded categorical features, and ‘SalePrice’ were concatenated into a dataframe before feature engineering. A correlation matrix was constructed and sorted based on decreasing correlation coefficients. The one of the feature pairs in the first six pairs with a correlation coefficient above 0.95 were dropped.

Features were then analyzed to determine similarity between the information provided in the features. ‘GarageArea’ and ‘GarageCars’ were combined by element-wise division to create a new feature, ‘GarageArea\_per\_Cars.’ Similarly, ‘YearRemodAdd’ and ‘YearBuilt’ were combined by element-wise subtraction to create a new feature ‘NumYears\_BeforeRemodAdd.’ For binary features with correlation coefficients equal or greater than 0.6, the features were combined into new features by element-wise OR operation and renamed with both feature names. After adding these new features to the train data, the original features used for the combinations were dropped.

### B.2.5. Dimensionality Reduction with PCA

To address the increase in dimensionality from applying one hot encoding to the categorical features, principal component analysis (PCA) was applied. PCA determines the principal components in a dataset based on the eigenvectors and eigenvalues of the covariance matrix and creates new features based on the weighted combinations of the principal components. After dropping the ‘Id’ and ‘SalePrice’ columns, PCA was applied to the train data set, with standardized numerical features and encoded categorical features, to reduce the number of features to 35.

## B.3. Data Subgrouping

### B.3.1. K-Means Clustering

To subgroup the train data, k-means clustering was applied. Repeated for a specified number of iterations, k-means clustering randomly assigns k centers for k clusters, assigns each datapoint to its closest center, and recalculates the cluster’s mean to account for the newly assigned datapoints. To find the ideal k value to optimize the clustering, the elbow method is applied; this entails calculating the error associated with a range of k for the number of clusters and plotting the error vs. the number of clusters. The point after which the curve appears approximately linear is selected for k. This was accomplished through testing k values in the range from 1 through 9 with the given function in class **plot\_inertia** and sklearn’s KMeans(). Based on the curve obtained, the ideal k appeared to be three. The assigned clusters were added as a new column.

### B.3.2. K-Nearest Neighbors Classifier

At this point, the train data was divided into train and validation data, with validation being 20%. K-nearest neighbors (KNN), which finds the k closest neighbors to a certain datapoint based on shortest distance and classifies the datapoint’s cluster through a majority vote, was applied. Using sklearn’s KNeighborsClassifier, the model was fitted with the train data with the assigned clusters from k-means clustering as the target. Grid-search cross validation was used to test the number of neighbors in the range of 1 to 30 and to test ‘uniform’ or ‘distance’ weights, which respectively means equal weighting across all points in a cluster or weighting that awards greater influence to the closest neighbors. The mean accuracy scores were above 0.90, and the best accuracy score and parameters were printed. The validated model was then used to predict the clusters for the validation dataset, with an accuracy score above 0.94.

Following the classification, the observations in the train and validation sets were grouped by their clusters, and both dataframes were exported as csv files to be imported into a separate notebook file with ML algorithms and the final predictions. This was revised for the final model to include both the processing and prediction steps in one notebook file. The term cluster and group are used interchangeably as they serve the same purpose at identifying a subgroup of the dataset produced by the Data Subgrouping step. Within the code the data group is represented through the cluster feature in the dataset.

## B.4. Prediction Model Training and Validation

The task required training multiple regression models on each set of grouped data. Initially, the entire dataset was passed as a csv that contained a feature that identified the group in which each observation belonged to. This data was processed and grouped into the k groups for training and validations of the prediction model. This was done through the use of splitting the loaded data with pandas functions and a custom made grouping function. This method of pre-preparation for each of the k groups was used because training each individual model on each individual group of data requires that data group. All models were not trained within the same code block/function, so it seemed redundant to re-group the training data every time a model had to be trained.

The decision to train each model separately was made for the sake of debugging and error checking. While the training process for each model would be very similar, the difference in their parameters could cause a lot of confusion when attempting to debug. Additionally, if they were all trained at the same time, running that single training instance would not only take longer, but make it difficult to isolate and debug that one individual model while still being able to access the other trained models. Since the individual model training was separated from each other, and the training of the stacking method model, a method of storing and referencing each individual model was necessary. This was handled by saving the trained models in a formatted data structure that was referenced by the stacking method model.

As the stacking method for the final predictor within the model required already made models, the individual models were instantiated and trained using GridSearchCV before passing them into the following Stacking Regressors. Using this method, the already optimized individual model for each group could be passed to the Stacking Regressor, speeding up the training of the stacking method predictor.

### B.4.1. Model Selection

#### General Approach

The initial idea behind the model selection for the individual models to be used in the stacking method, was that 4 relatively different models should be used as a basis, and from there additional models could be chosen based upon those individual model’s performances on the training and validation datasets. Since the task was to predict the SalePrice of houses, regression models needed to be selected. The initial 4 models were Stochastic Gradient Descent, Decision Tree Regression, Support Vector Machine Regression, and Neural Network Regression. From there the additional models that performed well in training with similar training times to that of our basis models, excluding Neural Network Regression, were chosen. This training performance was evaluated based upon the reported MSEs in the training process as detailed in section B.4.2. The models chosen that fit the speed and performance criterias were Linear Regression, Lasso Regularization, and Ridge Regularization.

#### Models

Stochastic Gradient Descent (SGD or sgd): Stochastic Gradient Descent utilizes a loss function to impose a penalty on predicted values in an effort to fit a model onto some linearly related data.

Decision Tree Regression (DTR or dtr): Decision Tree Regression uses a set of if then else rules in order to come up with a method of predicting a value.

Linear Regression (LR or lr): Linear Regression uses a specified loss function to predict the value of a variable on a dataset with a linear relationship.

Lasso Regularization (Lasso or lasso): Lasso Regularization is an extension of Linear Regression that places a penalty to increase the generalizability of the trained model, reducing overfit.

Ridge Regularization (Ridge or ridge): Ridge Regularization works similarly to Lasso Regularization, however it does not impose the same penalty that Lasso Regularization does.

Support Vector Machine Regression (SVM or svm): Support Vector Machine Regression extends the classification ability of the Support Vector Machine to predict values

Neural Network Regression (NN or nn): Neural Network Regression utilizes a series of layers of nodes, each having their own weights associated with them, to create a final predicted value.

Stacking Regressor (stregr): Chosen ensemble method, it utilizes the stacking of a series of estimators to be fed to its final\_estimator to create a new model based on the models comprising the estimators.

Random Forest Regression (RFR or rfr): Extends the Decision Tree Regression by incorporating bootstrapping to combine different trained models to create a new model and was used as the final\_estimator for the Stacking Regressor.

### B.4.2. Model Training Workflow

Workflow for Individual Models Training:

1. Initialization
   1. Initialize four datasets:
      1. **X\_train**, **y\_train** (feature and target train data)
      2. **X\_validation** and **y\_validation** (feature and target validation data)
   2. Group the four datasets into k groups, and store in a dictionary where the keys are the k subgroups and values are the four datasets within the k subgroup:
      1. **X\_train\_k**, **y\_train\_k** (feature and target train data identified by k subgroup)
      2. **X\_validation\_k**, **y\_validation\_k** (feature and target validation data identified by k subgroup)
   3. Initialize **fitted\_models**, a list to save individual trained models
2. Training and Validation of Individual Models
   1. Initialize a variable to store the model name that will be used in the stacking regressor
   2. Initialize **cluster\_log**, a dictionary to map each group’s ID (cluster ID) to its trained model
   3. For each subgroup,
      1. Initialize the ML model to be used for training
      2. Create a parameter grid for all possible parameters to be optimized
      3. Initialize the grid to be utilized for GridSearchCV model optimization with a scoring value for Mean Squared Error (MSE)
      4. Fit the grid to the training data for the current data group
      5. Save the best scoring metric score (best\_score\_), utilize it to calculate the MSE (-1 \* best\_score\_) and it’s associated parameters (best\_params\_)
      6. Use the optimized grid to predict the y for the current group validation set
      7. Calculate the validation MSE between the predicted\_y and the actual\_y for the validation dataset using the E function specified below
         1. 
      8. Store the following information in a dictionary (group\_results)
         1. Current group/cluster id
         2. Best parameters from running GridSearchCV
         3. Best MSE and Negative MSE from GridSearchCV
         4. The calculated MSE for the validation data
         5. The fitted model
      9. Store group\_results in cluster\_log with group ID as the key and group\_results as the value
   4. Save the result of step 2.a into fitted\_models as a tuple
      1. Tuple form (<name of model trained>, <cluster\_log for the trained model>)
   5. Repeat Steps 2.a to 2.b for each individual model
   6. Pass fitted\_models to the stacking method to train the stacking model
3. Ensemble Model Training
   1. Grouped Dataset Stacking Regressor
      1. Define the model name for the Stacking Regressors over the group
      2. Read in fitted\_models and the grouped training and validation data
      3. Initialize a dictionary to store the model name for the stacking regressor for the current group
      4. For each training and validation data group
         1. Create the list of estimators for the current data group
            1. This list of estimators is comprised of m, 2 element tuples where the first element is the model\_name of an individual model, and the second element is the saved best model for that individual model’s training on the current training and validation data group

m = the number of individual model types

Each model type has k fitted group models

Both of these are retrievable from fitted\_models

* + - 1. Initialize the final estimator for the Stacking Regressor for the current data group
         1. RandomForestRegressor(criterion=’mse’) was utilized
      2. Initialize the Stacking Regressor for the current datagroup using the results of 3.a.iv.1 and 3.a.iv.2
      3. Fit the Stacking Regressor on the X\_train\_k and y\_train\_k for the current group
      4. Using the fitted Stacking Regressor, predict the y for X\_validation\_k
      5. Calculate the validation MSE between the predicted\_y and the actual\_y, y\_validation\_k, for the current datagroup using the E function specified below
         1. 
      6. Store the following information in a dictionary (group\_results)
         1. Current group/cluster id
         2. Calculated validation MSE
         3. The fitted model
      7. Store group\_results in the dictionary created in step 3.a.ii
    1. Output a tuple
       1. Tuple is of the form (<3.a.i. model name, 3.a.ii dictionary>)
          1. At this point the dictionary from 3.a.ii has all the fitted models for each data group
  1. Entire Dataset Stacking Regressor
     1. Define the model name for the Stacking Regressor on the entire dataset
     2. Read in the output from step 3.a and the training and validation datasets from step 1.a
     3. Create the estimators list using the output from step 3.a
        1. The list of estimator is comprised of k, 2 element tuples where the first element is the model\_name for the fitted stacking regressor associated with the group of group ID = i, and the model is the fitted stacking regressor for group of group ID = i
        2. Initialize the final estimator for the Stacking Regressor
           1. RandomForestRegressor(criterion=’mse’) was utilized
        3. Initialize the Stacking Regressor for the inputted dataset using the results of 3.b.ii.1 and 3.b.ii.2
        4. Fit the Stacking Regressor on the X\_train and y\_train data
        5. Using the fitted Stacking Regressor, predict the y for X\_validation
        6. Calculate the validation MSE between the predicted\_y and the actual\_y, y\_validation, for the validation data using the E function specified below
           1. 
        7. Store the following information in a dictionary (all\_results)
           1. Model name
           2. Validation MSE
           3. Fitted model
        8. Output a tuple
           1. Tuple is of the form (<3.b.i. model name, 3.b.iii.7 dictionary)

##### General Workflow Intuition

This workflow was chosen primarily for the ability to have easy debugging access to each component in the pipeline. Because of this, the standardization of many operations within functions, as well as all of the outputs from those functions was needed. This made adding any additional models, as well as functions, fairly simple as methods were already put in place to handle multiple things from data processing to model training.

##### Initialization

The initialization process for the model training began with creating methods to store each trained best model. The training process for each model should roughly be the same. The training time of each model was reduced by formatting all data into the forms required, and saving for later reference.

##### Training and Validation of Individual Models

As each individual model, as well as the Grouped Dataset Stacking Regressor followed a similar blueprint, the main focus was the construction of a singular model training. From there, a working template was built so that each additional model training and validation method for all models that handle training on grouped data could be made using this template. The only real changes between all of the individual models were the specifying of model names, model initialization, parameter grids, and any adjustments to the final output to be saved to the fitted\_models storage variable.

##### Ensemble Model Training

The ensemble models differed from the individual models, as they did not utilize a GridSearchCV for hyper-parameter tuning/optimization, as the passed in models within the estimators variable had already been through that process. For both the ensemble models the creation of the estimators was the main difference.

For the Grouped Dataset Stacking Regressor, the method still resembled the training of the individual models very closely. The only real addition, aside from the declaration of the final\_estimator, was the need to create a new set of estimators for each individual dataset group.

For the Entire Dataset Stacking Regressor, it no longer required the iteration over every single dataset group, so only one set of estimators was made. This Entire Dataset Stacking Regressor also serves as an additional metric to evaluate the overall performance of the Grouped Dataset Stacking Regressor when predicting in the test data. This is elaborated on in the Evaluating Trained Models subsection in section C.2.4.

# C. Revisions for the Final Model

## C.1. Train and Test Data Processing

The primary difference between the alpha and final models is how the processes previously outlined are structured. The given train and test csv files are imported to the notebook as dataframes and are used as arguments for **test\_data\_prediction** to process both datasets simultaneously. In this function, the ‘Unnamed: 0” column is dropped from both the train and test sets. Features in the train data missing over 80% of observations are identified and dropped from both the train and test data.

The function **imputation**, which is applied with the train or test dataframe as the argument, handles the remaining missing values. **null\_columns** is applied to construct a dictionary with features missing values as keys and the percent of missing data as values. If the feature is numerical, the missing values are imputed with the median. If the feature is categorical and more than 40% of the data is missing, the missing values are imputed with a new category “Missing\_Data.” Otherwise, the missing categorical values are imputed with the mode.

The function **numerical\_feature\_standardize** standardizes the numerical features for both the train and test data. After separating the numerical data from the train and test data, the ‘Id’ and ‘SalePrice’ columns are stored in variables and dropped from the datasets. Using sklearn’s StandardScaler(), the model is fitted with the train numerical data and used to transform the train and test numerical data. The standardized train and test data are concatenated with their respective ‘Id’ and ‘SalePrice’ columns before returning the dataframes.

The function **categorical\_feature\_encoding** uses one hot encoding to convert the categorical features in the train and test data into binary features. For each categorical feature, the remaining features in the train and test data are stored in variables. If the feature is already binary, the groups are converted to either 0 or 1 in both the train and test data. Otherwise, sklearn’s LabelEncoder and LabelBinarizer are fitted with the feature in the train data; the fitted model is then used to transform the train and test feature. The encoded features are then concatenated with the remaining data in their respective train and test data sets before returning the dataframes.

The function **feature\_engineering** drops one feature in strongly correlated feature pairs and combines features that are moderately correlated for train and test data. If the correlation coefficient of two features in the train data is equal to or above 0.95, one of the features is dropped for the train and test data. If two features are binary, likely the encoded categorical features, and have a correlation coefficient equal to or above 0.6, a new feature combining the two using element-wise OR operation is created, and the original features are dropped.

The function **pca** reduces the dimensionality of the train and test data. After storing the ‘Id’ and ‘SalePrice’ columns in variables and dropping both from the datasets, sklearn’s PCA was fitted with the train data and applied to transform the train and test datasets, reducing the number of features to 35. The respective ‘Id’ and ‘SalePrice’ columns were then concatenated to the train and test datasets.

At this stage, the train data was split into train and validation data, respectively 80% and 20% of the given data set. The features were also separated from the target ‘SalePrice,’ denoted as X and y respectively, for both the train and validation sets. The function **plot\_inertia** is applied to graph the error of k-means clustering vs. number of clusters in the range 1 to 10. Observing a reduction in the overall MSE, the number of clusters was chosen to be four for the alpha model, as opposed to the three clusters used for the alpha model. The function **kmeans\_clustering** creates a KMeans model fitted with the train features data and uses the fitted model to predict the clusters for the train and validation data. After predicting the clusters for each datapoint, **KNN\_classifer** builds a KNeighborsClassifier for the number of neighbors in the range 1 to 30, fits the classifier with train data and the target clusters set with **kmeans\_clustering**, and predicts the clusters of the validation data. The accuracy of the validation cluster predictions is plotted against the number of neighbors tested. This model is validated using grid-search cross-validation with the same parameters used in the alpha model including the number of neighbors in the range of 1 to 30 and ‘uniform’ or ‘distance’ weights. The mean accuracy scores were above 0.90, and the best accuracy score and parameters were printed.

## C.2. Prediction Model Training

The prediction modeling workflow changed very little from the initial stages, as the main point of focus was converting the already existing code cells into easier to execute and portable functions. This required the creation of new helper functions for running similar parts of the model training, as well as grouping the data.

With the final model, when training the Grouped Dataset Stacking Regressor, best\_estimator\_ was passed in as opposed to the entire GridSearchCV object. With how GridSearchCV works, unless executing a prediction, the best model is not automatically used.

A helper function was made to calculate MSE as according to the assignment specifications. The alpha-mode used sklearn’s built-in function to calculate the Mean Squared Error divided the sum-squared difference by N, as opposed to dividing by 2N as specified in the assignment.

#### Functions

The training process had functions used to group the data into the appropriate datasets for training. These functions included two helper functions. The first is the **create\_features\_dict()** function, which creates a dictionary to reference the X and y features of the inputted datasets. The second helper function is the **group\_dataset()** function, which does the bulk of the grouping work by making all of the datasets for each of the k groups the data is separated into.

These two helper functions were utilized by the three data grouping functions. The first being **group\_all\_data()**, which takes in the X and y for the training, validation, and test datasets and groups all of them and returns the grouped datasets. The second function is **group\_training\_data()** which does the same process as **group\_all\_data()**, but excludes the test dataset as an input. This is replicated by the last function **group\_test\_data()**, which applies the same process as **group\_all\_data()**, but only on the X and y for the test dataset.

For the functions for the actual training of the models, two helper functions were also made. The first helper function is the **MSE()** function, which calculates the Mean Squared Error with the appropriate division by 2N. The second helper function is the **run\_grid\_search()** function which removes the need to include the grid search code in every single individual model training function.

The functions used for individual model training all follow the same naming convention, **train\_<model\_name>()**. These all follow the model training workflow outlined in section B.4.2. 7 different individual algorithms were trained, resulting in the **train\_sgd** (Stochastic Gradient Descent), **train\_dtr** (Decision Tree Regression), **train\_svr** (Support Vector Machine Regression), **train\_lr** (Linear Regression), **train\_lasso** (Lasso Regularization), **train\_ridge** (Ridge Regularization), and **train\_nn** (Neural Network Regression) functions.

The training of both the Grouped Dataset Stacking Regressor and the Entire Dataset Stacking Regressor were separated into individual functions as well. These functions are named **train\_stregr\_group()** and **train\_stregr\_all()**, with the \_group() and \_all() representing the training of Grouped Dataset Stacking Regressor and Entire Dataset Stacking Regressor respectively.

## C.3 Evaluating Models

### C.2.4. Evaluating Models

#### Evaluating Training Process

In order to evaluate the performance of the individual and ensemble models the individual MSEs for each model were stored and reported during the training process. Two MSEs for each individual model used in training were calculated and reported, the best\_score\_ outputted by GridSearchCV and the validation MSE calculated using the **MSE()** function. For clarification on these MSEs refer back to section B.4.2.

For the ensemble models, a singular MSE for each model was calculated and reported. This MSE was calculated using the **MSE()** function.

##### Individual Models

In the training and optimization of each individual model, GridSearchCV was run with cv=10, and its predicted\_y was evaluated on a separate validation dataset for the GridSearchCVs training dataset. This was done in hopes of minimizing any overfitting.

##### Ensemble Models

The stacking regressors were evaluated based on their MSE for their respective predictions on their validation sets, calculated by **MSE()**.

##### Overall Evaluation:

As it stands the MSEs for the individual and ensemble models can be improved further, it is unlikely that any of the improvements due to changing model design and training will be largely significant. Model selection could aid in improving.

#### Evaluating Trained Models Predictions

The final evaluation of the trained models came in the form of functions to predict the y on both the Grouped Test Dataset and the Entire Test Dataset. Three functions were defined. Two functions for the Grouped Test Dataset, and one function for the Entire Test Dataset. All of these functions utilize the **MSE()** function to calculate their loss.

The two functions were made for the Grouped Test Dataset to output both the MSE for each individual Group Stacking Regressor, as well as an Overall MSE for the Grouped Test Dataset.

The MSE evaluated by the prediction function for the Entire Test Dataset serves as a way to evaluate the Overall MSE of the Grouped Test Dataset predictions. Since this model has been trained with the Group Stacking Regressor and a final estimator directly on the entire test dataset, it can help give additional information on determining how good of a prediction Overall MSE is.

##### Functions

The **predict\_grouped\_data()** function is the first of the two functions to evaluate the Grouped Test Dataset. This function utilized the individual group predictors to predict the y for each of the groups in the Grouped Test Dataset, and reports the MSE for each group. It also returns a dictionary that stores the reported MSE, predicted\_y, actual\_y, model name, and fitted model used for each prediction on each of the k groups of data.

The **predict\_all\_data()** function is the function that predicts y for the Entire Test Dataset, and reports the calculated MSE. This function returns a dictionary containing the calculated MSE, the predicted\_y, the actual\_y, the model name, and the model used in the prediction.

**overall\_group\_mse()** is the last function used for calculating the trained model’s loss. This is the second function that utilizes the Grouped Test Dataset. It goes over each data group and calculates predicted\_y, and stores the predicted\_y and actual\_y for each group in their own arrays. These arrays, which contain all of the predicted\_y and actual\_y for the entire dataset, are then evaluated using the **MSE()** function to get the Overall MSE of the entire trained Grouped Dataset Stacking Regressor. This function outputs the calculated MSE, the arrays containing all the predicted\_y and actual\_y, and all of the trained models used to make the predictions.

##### Grouped Dataset Stacking Regressor: Individual Group MSE

Seeing the individual group model’s loss, as well as the overall loss is useful. Through this, any groups that are performing better than one another are evident, and can provide insights into how that can affect the Overall MSE.

##### Entire Dataset Stacking Regressor: Entire Dataset MSE

The inclusion of this metric serves as more of a secondary measuring stick. It does not directly help us compare the varying loss values reported by **predicted\_grouped\_data()**, but it can give an indication how much room for improvement exists in the predictions when compared to the OverallMSE

##### Grouped Dataset Stacking Regressor: Overall MSE

This is the final overall loss reported across all predictions on the Grouped Test Data. This serves as the final evaluation metric for the training process on the grouped data, and represents the entire loss seen across the entire dataset.

##### Overall Evaluation

The two auxiliary methods of MSE evaluation, Individual Group MSE and Entire Dataset MSE, help with the evaluation of Overall MSE of the entire model training process. By viewing both the Individual Group MSE and the Entire Dataset MSE, individual group performance is displayed, as well as how any possible adjustments of those group performances can aid in lowering Overall MSE.

# D. Conclusion

To summarize the workflow implemented in **test\_data\_prediction**, the features with over 80% missing data are dropped, the missing values are imputed with the median for numerical data or the mode or “Missing\_Data” for categorical data, numerical data is standardized, categorical data is converted to binary data through one hot encoding, feature engineering is performed to construct new features via element-wise operations of two correlated features, and PCA is applied for dimensionality reduction. After splitting the train data into train and validation sets, k-means clustering is used to divide the data into four subgroups, and a KNN classifier is built to predict the clusters for the test data. Regarding the algorithms used to predict the target ‘SalePrice’ for each subgroup, stochastic gradient descent, decision tree regressor, linear regression, lasso and ridge regularization, support vector machine, and neural network models were developed. The overall MSE using the stacking regressor applied to the subgroups was 707131939, and the overall MSE with the final stacking regressor applied to the full dataset was 552354324. For further improvements, the number of clusters and the parameters used for the KNN classifier should be re-evaluated to determine the optimal subgrouping to decrease the MSE associated with the subgroup stacking regressor. The choice of ML algorithms and parameters tested for each algorithm should also be optimized.

# 

# E. References

A collection of links to non-class-material sources referenced in the creation of the project

1. [How to perform one hot encoding on multiple categorical columns - Data Science Stack Exchange](https://datascience.stackexchange.com/questions/71804/how-to-perform-one-hot-encoding-on-multiple-categorical-columns)
2. [Python: Finding Missing Values in a Pandas Data Frame | by Kallepalliravi | Analytics Vidhya | Medium](https://medium.com/analytics-vidhya/python-finding-missing-values-in-a-data-frame-3030aaf0e4fd)
3. [Python: Handling Missing Values in a Data Frame | by Kallepalliravi | Analytics Vidhya | Medium](https://medium.com/analytics-vidhya/python-handling-missing-values-in-a-data-frame-4156dac4399)
4. [k-Neighbors Classifier with GridSearchCV Basics | by Salem O. | Medium](https://medium.com/@salemortega/k-neighbors-classifier-with-gridsearchcv-basics-3c445ddeb657)
5. [How to use Data Scaling Improve Deep Learning Model Stability and Performance - MachineLearningMastery.com](https://machinelearningmastery.com/how-to-improve-neural-network-stability-and-modeling-performance-with-data-scaling/)
6. [List Highest Correlation Pairs from a Large Correlation Matrix in Pandas? - Stack Overflow](https://stackoverflow.com/questions/17778394/list-highest-correlation-pairs-from-a-large-correlation-matrix-in-pandas)
7. [List of Features With Strongest Correlation](https://safjan.com/features-with-strong-correlation/)
8. [Convert pandas Series to DataFrame - Stack Overflow](https://stackoverflow.com/questions/26097916/convert-pandas-series-to-dataframe)
9. [Types of Machine Learning Algorithms You Should Know | by Jose Fumo | Towards Data Science](https://towardsdatascience.com/types-of-machine-learning-algorithms-you-should-know-953a08248861)
10. [How to Choose the Right Machine Learning Algorithm: A Pragmatic Approach](https://labelyourdata.com/articles/how-to-choose-a-machine-learning-algorithm)
11. [1. Supervised learning — scikit-learn 1.3.2 documentation](https://scikit-learn.org/stable/supervised_learning.html)
12. [sklearn.ensemble.StackingRegressor — scikit-learn 1.3.2 documentation](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.StackingRegressor.html)
13. [Pandas vs NumPy in Data Science: Top 15 Differences](https://www.knowledgehut.com/blog/data-science/pandas-vs-numpy)
14. [sklearn.model\_selection.GridSearchCV — scikit-learn 1.3.2 documentation](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html)
15. [Stacking- Ensemble Learning Method | python scikit-learn Demo](https://www.youtube.com/watch?v=r8-Na812VDw&ab_channel=RitheshSreenivasan)
16. [sklearn.metrics.mean\_squared\_error — scikit-learn 1.3.2 documentation](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html)
17. [Python Stacking Regressor Mastery: From Basics to Advanced Tips](https://www.youtube.com/watch?v=gpOBpXZhUrI&ab_channel=RyanNolanData)
18. [Stacking Ensemble Machine Learning With Python - MachineLearningMastery.com](https://machinelearningmastery.com/stacking-ensemble-machine-learning-with-python/)
19. [9. Model persistence — scikit-learn 1.3.2 documentation](https://scikit-learn.org/stable/model_persistence.html)
20. [StackingRegressor: a simple stacking implementation for regression - mlxtend](https://rasbt.github.io/mlxtend/user_guide/regressor/StackingRegressor/)
21. [How to save a trained model by scikit-learn? - Stack Overflow](https://stackoverflow.com/questions/56107259/how-to-save-a-trained-model-by-scikit-learn)
22. [Sort Correlation Matrix in Python - GeeksforGeeks](https://www.geeksforgeeks.org/sort-correlation-matrix-in-python/#)
23. [How to build and fill pandas dataframe from for loop? - Stack Overflow](https://stackoverflow.com/questions/28056171/how-to-build-and-fill-pandas-dataframe-from-for-loop)