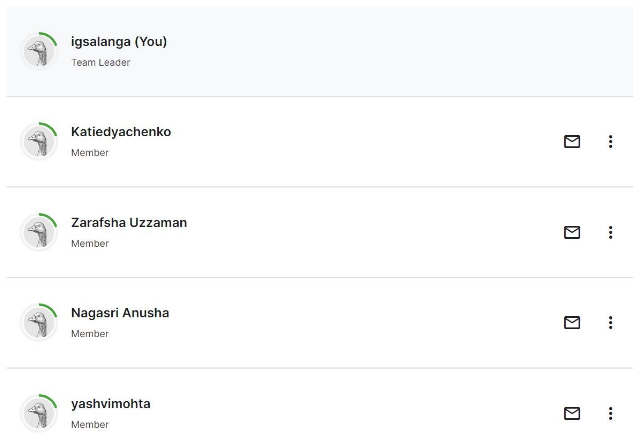
HOUSE PRICES-ADVANCED REGRESSION TECHNIQUES

TEAM NAME:

Team 19

ACCOUNT NAMES:

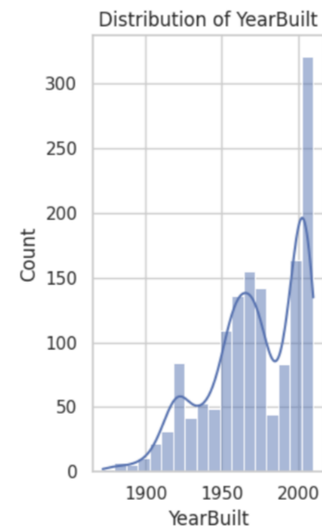
1. igsalanga
2. Katiedyachenko
3. Nagasri Anusha
4. yashvi mohta
5. Zarafsha Uzzaman



METHODS USED:

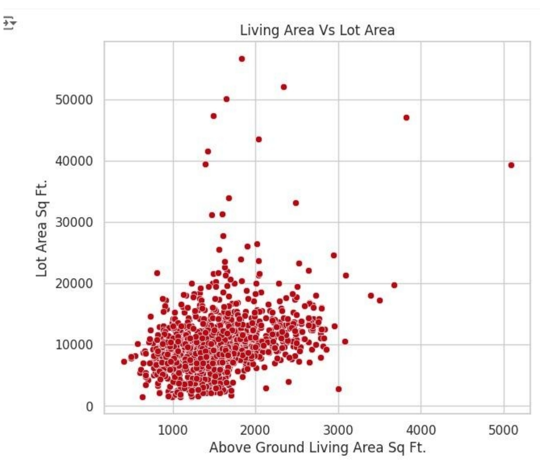
1. **EDA Text and Visual**
    a. *Text:* For initial exploratory train data analysis, the first 5 rows of each category were printed to give an idea of what the data looked like. To better understand the dataset, the data types of all columns were printed. Then summary statistics were calculated for all numerical columns. To help with cleaning the data, each column was filtered for null values, and if any were present the output was equal to the sum of how many per. The exact same steps were taken for the exploratory test data analysis; however, it was revealed that the values were relatively different, so additional cleaning would have to be done to account for the additional types of null values.

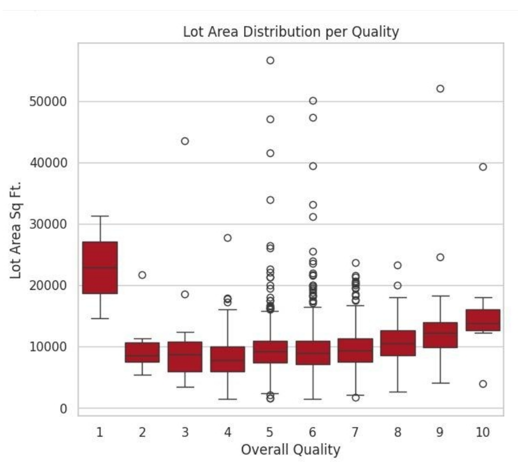
b. *Visualization:* Select numerical variables from train and test data were chosen to be plotted as distribution charts. Each chart represents their respective trends. For example, regarding YearBuilt variable in the train data, the distribution showed how there was an increasing amount of houses being built closer to the 2000's. While there was an overall upward trend, the values spiked significantly from the late 90's to the 2000's. Either this or that this dataset contains more information on houses built in this time period. Outside of distribution graphs, scatter plots and box plots are also good visualization methods for numerical data. As shown in the graphs on the left, they can be used to show the relationship and correlation between two numerical variables. For the test data, regarding Living Area and Lot Area, the information was congested in the lower left area of the graph showing that homes typically had a 1:1000 ratio of living area to lot area. Regarding the relationship in the test data for overall quality and lot area, there was a relatively low median for all quality scores besides homes that scored a one. Similar to numerical variables, select categorical variables were chosen and made into bar graphs giving the counts of their respective values. Bar graphs are an easy and intuitive way to plot information to give a visual on the distribution of data.



2. Cleaning

a. In order to clean our data set, the team's general approach was to replace nulls. For example, within the Pool Quality variable, NAs were replaced with "No Pool". Data types were also converted to ensure each variable corresponded to the correct data type. For variables that had levels, the data type was converted to variabel. For numeric variables, most were left in their original correct data type. Time related variables were dealt with using the pandas pd.datetime method.





3. Feature Engineering And Selection

a. *New Features Made:* Several Interaction terms were created in order to enhance the existing data and to create better informed models that are able to make predictions on the target variable that are as accurate as possible. Seven new features were created.

i. Property Age variable was constructed by subtracting the year that the house was built from the year it was built

ii. Total Square Feet variable was created by adding up the 1st floor square feet, 2nd floor square ft and total basement square feet.

iii. Outdoor living space variable was created by adding up all the outside square footage variables available in the data set (like Wood deck square feet for example)

iv. Neighborhood lot Frontage variable was constructed by grouping neighborhoods and calculating the mean Lot Frontage by neighborhood.

v. Garage variable multiplies Garage Cars by Garage Area.

vi. Quality Area Interaction variable multiplies the overall quality by the lot area.

b. *Correlation Analysis:*

i. This analysis focuses on identifying highly correlated feature pairs in both training and test datasets and evaluating their importances in predicting 'SalePrice'. Initially, correlation matrices were calculated for numerical features, identifying pairs with correlations greater than 0.8. From this, common highly correlated pairs between the training and test datasets were extracted for further analysis. Key pairs identified include ('LotArea', 'QualAreaInteraction'), ('YearBuilt', 'Property_Age'), ('TotalBsmtSF', 'TotalSqFt'), ('GrLivArea', 'TotalSqFt'), ('GarageCars', 'GarageArea'), and others.

ii. A Random Forest Regressor was then used to evaluate the importance of these common pairs, so that we can figure out which features should be removed. The data was split into training and validation sets, with the model being trained on the training set. Feature importances for each pair were computed, sorted to identify the most influential predictors within each pair. This analysis revealed which features are most influential in predicting 'SalePrice' among the highly correlated pairs.

iii. From this we further removed features, to have better predictive abilities.

4. **<u>Data Type Consistency:</u>** A function ensure_consistent_types is defined to convert all categorical features to string type, ensuring consistent data types and avoiding potential issues during one-hot encoding.
5. **<u>Data Preprocessing:</u>** The preprocess_data function is responsible for:
    a. Separating the target variable (SalePrice) and applying a logarithmic transformation to normalize its distribution.
    b. Combining training and test datasets to ensure uniform preprocessing.
    c. Identifying numeric and categorical features for different preprocessing pipelines.
    d. Imputing missing values and scaling numeric features.
    e. Imputing missing values and applying one-hot encoding to categorical features.
    f. The ColumnTransformer is used to apply these transformations in parallel to the appropriate columns
6. **<u>Data Splitting:</u>**
    a. Train_Test_split is a Scikit-learn library function to randomly partition the data into training and validation sections. In the code snippet provided above, the X_train_processed contains the training data and the y_train contains the labels of this train data. The test size is set to 20% to indicate that 20% of the training data will be used for validation. The result of this is 4 subsets of data; X_train_final

contains the training section, y_train_final contains the labels for the training section , X_valid contains the 20% of data for validation and y_valid contains labels for the validation set.

b. The code used :- X_train_final, X_valid, y_train_final, y_valid = train_test_split(X_train_processed, y_train, test_size=0.2, random_state=42)

Predictive Models:

1. **Ridge Regression**

This code block outlines the implementation of a Ridge Regression model for predicting house sale prices. The process is divided into several key steps:

a. Library Importation*:* Essential Python libraries such as pandas, numpy, and various components from scikit-learn are imported to handle data manipulation, mathematical operations, and machine learning tasks.

b. Model Training*:* A Ridge Regression model is instantiated with a regularization parameter alpha set to 1.0. The model is then trained on the preprocessed training data.

c. Model Evaluation: The trained model's performance is evaluated on the training data using the Root Mean Squared Error (RMSE) metric, which provides a measure of the model's prediction accuracy.

i. Predictions are made on the training dataset, and the RMSE is computed to assess the model's training performance.

ii. Then we make predictions on the validation dataset, and see how the RMSE was and realize how to hypertune the model further.

iii. In the end we make predictions on test data for the competition.

2. <u>Support Vector Regression (SVR)</u>

a. Library Importation

i. Besides essential libraries like pandas and numpy, which are used for data manipulation and numerical operations, components from scikit-learn were imported to do data preprocessing tasks and for setting up the machine learning pipeline. From sklearn.svm, SVR parameter was imported to construct the regression model.

b. Data Preprocessing

i. The preprocess_data function is defined to handle training and testing datasets. It:

ii. Separated the target variable from the predictors and applied the logarithmic transformation to normalize its distribution,

iii. Identified categorical variables, which were encoded to convert them into an appropriate form for the machine learning algorithms,

iv. Identified numeric variables, which were scaled and imputed to handle missing values and normalize the data, and lastly,

v. Used a ColumnTransformer to apply mentioned transformations appropriately.

c. Model Training

i. An SVR model was initialized with tuned hyperparameters. kernel = ‘rbf’ is a kernel that is commonly used in SVM for non-linear problems. C = 100 parameter was used for regularization, which helped to control the trade-off between getting a low error on the training data and minimizing the model complexity. gamma = ‘auto’ defines the influence of training on the learning process. After which the model was trained on the preprocessed training data (X_train and y_train)

d. Model Evaluation

i. The SVR model’s performance was evaluated using the Root Mean Squared Error. RMSE provided a measure of SVR model’s prediction accuracy by calculating the square root of the average squared differences between the predicted and actual values.

ii. Predictions are made on the training dataset, and the RMSE is computed to assess the model’s training performance.

iii. Then we make predictions on the validation dataset, and see how the RMSE was and realize how to hypertune the model further.

iv. In the end we make predictions on test data for the competition.

3. **Random Forest**

a. The following is the process for putting a Random Forest Regression model for house sale price prediction into practice and assessing it:

b. Random Forest Model Initialization:Using particular hyperparameters, a RandomForestRegressor from the scikit-learn module is instantiated:

i. n_estimators = 100 There will be one hundred decision trees in the model.

ii. random_state=42: Ensures reproducibility of results by setting a seed for the random number generator.

iii. max_depth=105: Each tree in the forest can grow to a maximum depth of 105 nodes.

iv. min_samples_split=2: A node will split if it contains at least 2 samples.

v. min_samples_leaf=1: A leaf node must have at least 1 sample.

vi. max_features=sqrt_features: The square root of the total number of features is the amount of features that are taken into account for splitting at each node.

c. Model Training:

i. The model is trained on the preprocessed training dataset X_train_final with corresponding target values y_train_final.

d. Model Evaluation:

i. Predictions are made on the training dataset, and the RMSE is computed to assess the model's training performance.

ii. Then we make predictions on the validation dataset, and see how the RMSE was and realize how to hypertune the model further.

iii. In the end we make predictions on test data for the competition.

4. Catboost

a. The methodology for deploying a CatBoost Regression model to predict house sale prices involves several systematic steps, as outlined below:

b. Library Importation:

    i. The CatBoost library is imported to utilize the CatBoostRegressor, a machine learning algorithm based on gradient boosting on decision trees.

c. Model Initialization:

    i. A CatBoostRegressor model is initialized with the following hyperparameters:
    ii. iterations=1500: The model will perform 1500 boosting iterations.
    iii. learning_rate=0.05: Sets the step size for updating the model's weights.
    iv. depth=6: Specifies the maximum depth of the individual trees.
    v. loss_function='RMSE': The model optimizes the Root Mean Squared Error during training.
    vi. eval_metric='RMSE': Performance evaluation during training also uses RMSE.
    vii. random_seed=42: Ensures reproducibility of the model's results.
    viii. verbose=100: Outputs training progress at every 100th iteration.
    ix. l2_leaf_reg=1: Applies L2 regularization to the leaf weights.

d. Model Training:

    i. The model is trained on the preprocessed training dataset X_train_final with the log-transformed target variable y_train_final.

e. Prediction and Evaluation:

    i. Predictions are made on the training dataset, and the RMSE is computed to assess the model's training performance.

ii. Then we make predictions on the validation dataset, and see how the RMSE was and realize how to hypertune the model further.

iii. In the end we make predictions on test data for the competition.

f. This methodology leverages the powerful CatBoost algorithm, known for handling categorical features effectively and providing robust results even with default parameter settings. The chosen hyperparameters aim to balance the model's complexity and learning rate to minimize overfitting while achieving high predictive accuracy. The final predictions are prepared in a format suitable for submission to a competition or for further business analysis.

5. <u>Stacked Models:</u>

a. Library Importation

i. For our stacking ensemble model, we import necessary libraries from scikit-learn, including StackingRegressor, and from CatBoost for the CatBoostRegressor.

b. Model Initialization

i. The stacking model comprises several base models and a meta-regressor:

c. Base Models: A set of diverse models, including Ridge Regression and Random Forest, are initialized with their respective hyperparameters.

d. Meta-Regressor: A CatBoostRegressor is initialized with the following hyperparameters:

i. iterations: 1500, to perform a substantial number of boosting iterations.

ii. learning_rate: 0.05, to control the step size in weight updates.

iii. depth: 6, to set the maximum depth of trees.

iv. loss_function: ‘RMSE’, to optimize the model for the Root Mean Squared Error.

v. eval_metric: ‘RMSE’, to evaluate model performance during training.

vi. random_seed: 42, to ensure reproducibility.

vii. verbose: 100, to output training progress periodically.

viii. l2_leaf_reg: 1, to apply L2 regularization to the leaf weights.

e. Model Training

i. The stacking model is trained on the preprocessed training dataset X_train with the target variable y_train. The base models generate predictions that are used as input for the meta-regressor.

f. Prediction and Evaluation

i. Predictions are made on the training dataset, and the RMSE is computed to assess the model’s training performance.

ii. Then we make predictions on the validation dataset, and see how the RMSE was and release how to hypertune the model further.

iii. In the end we make predictions on test data for the competition.

6. **<u>Neural Networks:</u>**

a. This section describes the methods used to implement a neural network model to predict residential sales prices using the Keras library with a TensorFlow backend..

b. Library Importation

i. The Keras library is imported to construct the neural network architecture. Additionally, NumPy is used for numerical operations, and scikit-learn’s mean_squared_error function is utilized to evaluate the model's performance.

c. Neural Network Architecture

   i. The neural network is defined as a Sequential model with the following layers:

   ii. Input Layer: Consists of 128 neurons and uses the ReLU activation function. The input dimension is set to match the number of features in X_train.

   iii. Hidden Layers: Two hidden layers with 64 and 32 neurons, respectively, both using the ReLU activation function.

   iv. Output Layer: A single neuron with a linear activation function, suitable for regression tasks.

d. Model Compilation

   i. The model is compiled with the Adam optimizer, a popular choice for deep learning models due to its adaptive learning rate capabilities. The learning rate is set to 0.001, and the loss function is mean squared error, aligning with the regression objective.

e. Model Training

   i. The model is trained on the preprocessed training data (X_train, y_train) for 100 epochs with a batch size of 32. A validation split of 10% is used to monitor the model's performance on unseen data during training.

f. Prediction and Evaluation

i. Predictions are made on the training set and RMSE (Root Mean Squared Error) is calculated to assess the accuracy of the model. RMSE measures how well model forecasts match actual sales prices.

ii. Predictions are made on the training dataset, and the RMSE is computed to assess the model's training performance.

iii. Then we make predicitions on the validation dataset, and see how the RMSE was and reliase how to hypertune the model further.

iv. In the end we make predictions on test data for the competition.

g. Inverse transformation

i. Since the target variable is logarithmically transformed for normalization, the forecasts are logarithmically inverse transformed to obtain actual sales prices.

7. **Lasso Regression:**

a. This section describes the methodology for deploying a Lasso regression model to predict house sale prices. Lasso regression is a type of linear regression that includes a penalty term to encourage simpler models with fewer coefficients.

b. Library Importation

i. The Lasso class from the scikit-learn library is used to implement the Lasso regression model. Additionally, the mean_squared_error function is utilized to evaluate the model's performance, and the sqrt function from the math module is used to calculate the root mean squared error (RMSE).

c. Model Initialization and Training

i. alpha: The regularization strength; must be a positive float. Smaller values indicate less regularization.

ii. warm_start: When set to True, reuse the solution of the previous call to fit as initialization.

iii. max_iter: The maximum number of iterations for the optimization algorithm.

iv. selection: The method for choosing the features to update the weights; 'random' means a random coefficient is updated every iteration.

d. Hyperparameter Tuning

i. To obtain the ideal alpha that minimizes the RMSE, the model is retrained while the alpha value is gradually increased. Repeat this procedure until the alpha value approaches one.

e. Evaluation

i. The model predicts on the training set for each alpha value, and the RMSE is computed. The alpha values that correlate to the RMSE values are kept in a dictionary.

8. <u>XGBoost</u>

a. The section describes the methodology regarding the XGBoost model to predict sales prices. XGBoost stands for Extreme Gradient Boosting and uses tree boosting for regression, classification, and ranking. For this project, it will be used for regression.

b. Library Importation

i. To create an XGBoost model, first XGBoost has to be installed into the environment. From XGBoost, then XGBRegressor can be imported. Additionally, mean_squared_error must be imported as well to calculate and determine root mean squared error (RMSE).

c. Model Initialization[1]

i. min_child_weight: minimum sum of instance weight needed in a child; if the sum of instance weight is less than the desired min_child_weight, then the process will give up partitioning

ii. gamma: determined minimum loss required to decide whether to partition on a leaf node in a tree

iii. subsample: ratio of training; XGBoost will randomly sample the desired amount prior to growing trees

iv. colsample_bytree: subsample ratio of columns when constructing each tree

v. reg_alpha: L1 regularization terms on weights

vi. reg_lambda: L2 regularization terms no weights

vii. random_state: used to control randomness and sets the random-generated seed

viii. n_estimators: determines the numbers of boosting rounds or trees to build[2]

ix. max_depth: maximum depth of trees; increasing this value will make the model more complex and susceptible to overfitting

---

[1] https://XGBoost.readthedocs.io/en/stable/parameter.html

[2] https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-XGBoost-with-codes-python/#:~:text=What%20does%20the%20'N_estimators'%20parameter,be%20tuned%20for%20optimal%20performance.

x. learning_rate: the desired step size at each iteration, used to prevent overfitting

xi. objective: goal of the entire model; has multiple objective options to change the purpose and calculation process of the model

d. Model Training

i. The XGBoost model is trained on the prepossed training dataset X_train_final with the log-transformed y_train_final.

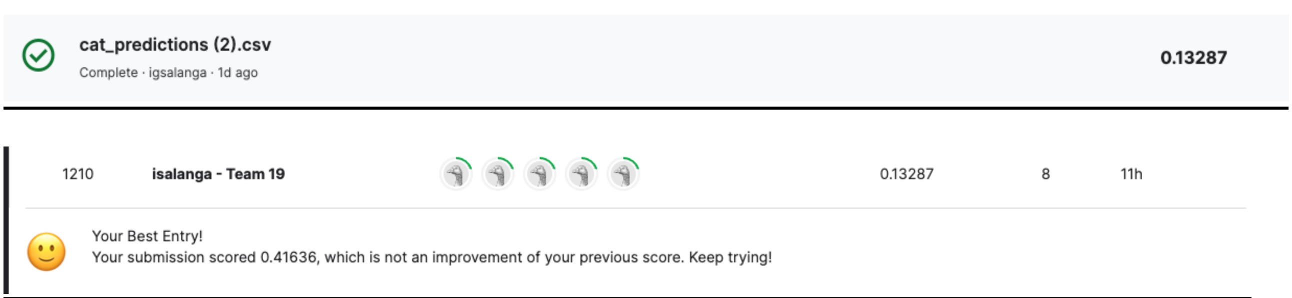
e. Prediction and Evaluation

i. Predictions are made on the training dataset, and RMSE is computed to assess the model's training performance.

ii. Using the same model, it is then applied to the validation data, and RMSE is calculated once again.

iii. These results are either hypertuned to reduce the RMSE further or used to make final predictions that are submitted into the competition.

SCORE FROM KAGGLE: (Best Score)

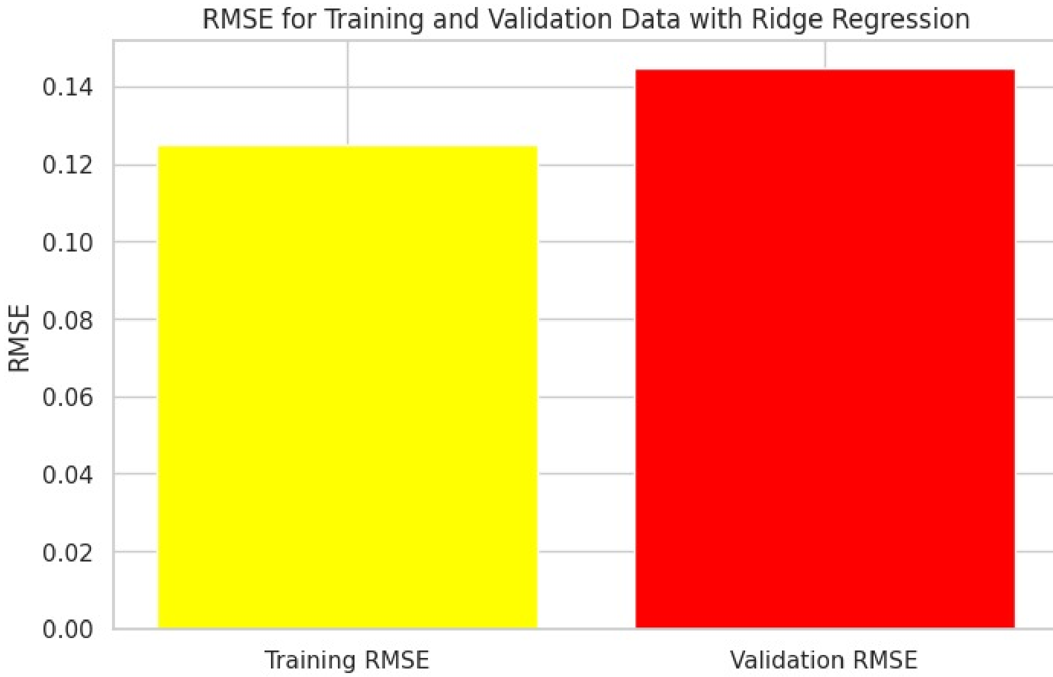


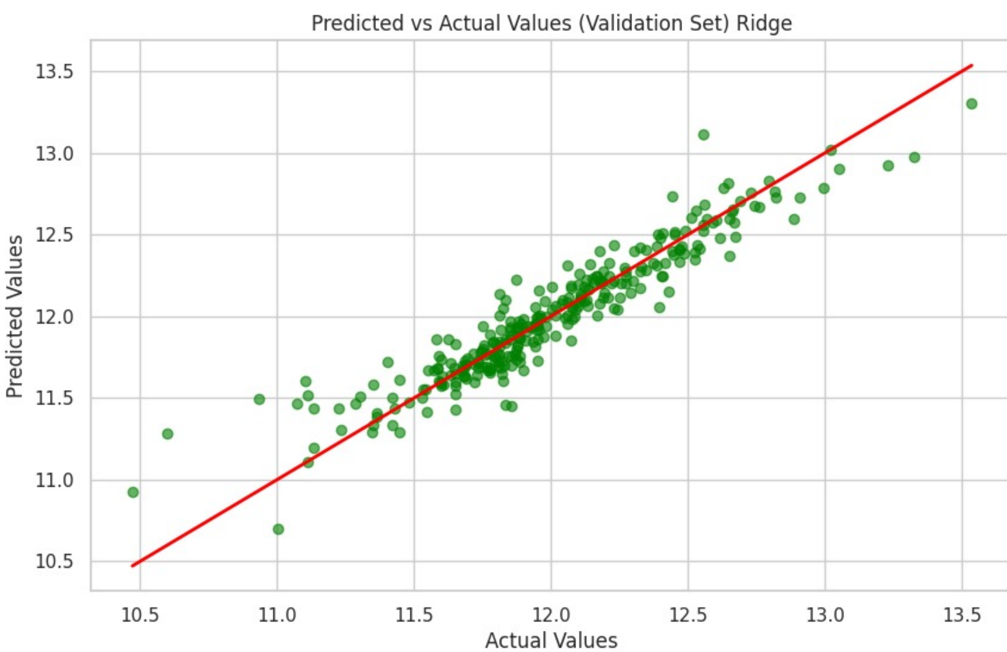
RESULTS AND REPORTS:

- Ridge Regression RMSE:

```
RMSE on training data: 0.12488110280158735
RMSE on valid data: 0.1446240011733382
```

- Ridge Performance Visualization



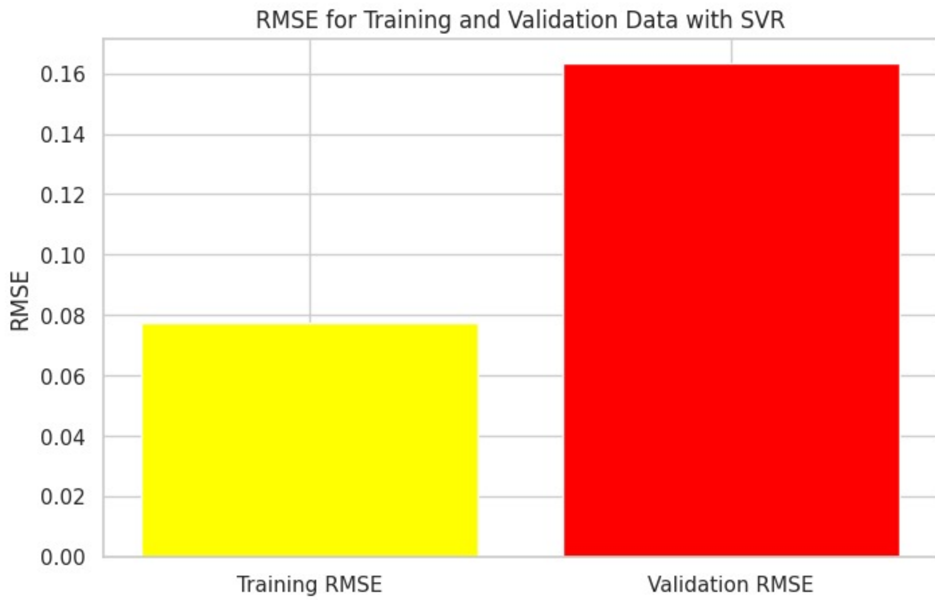


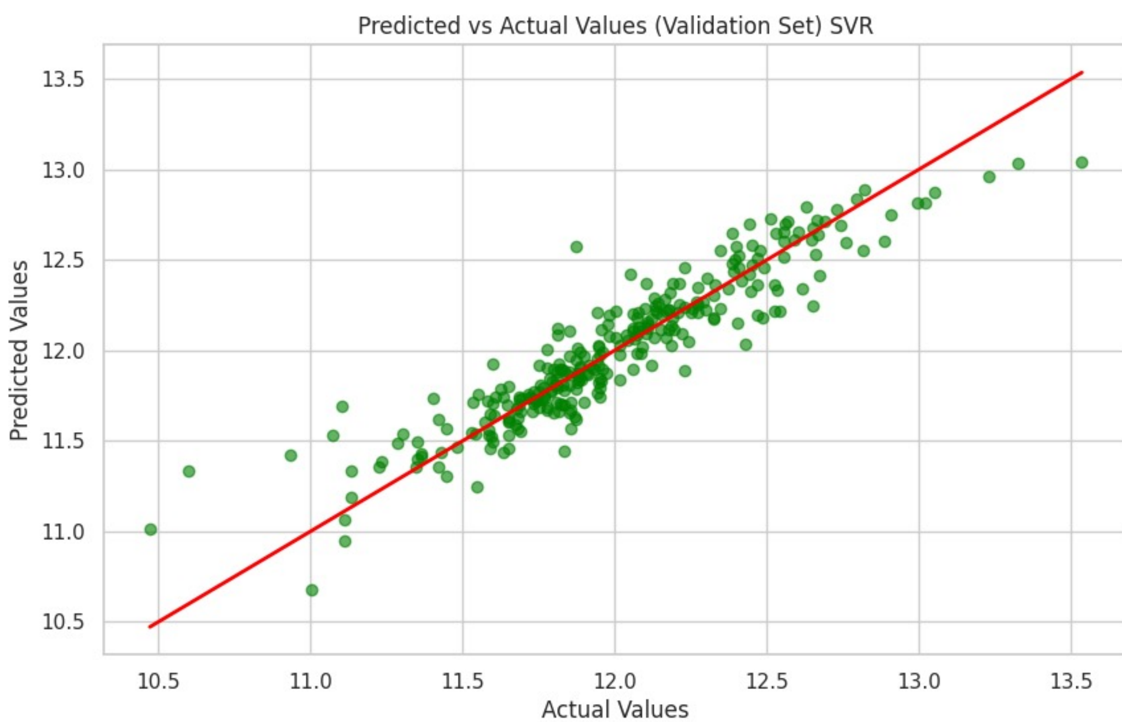
- Support Vector Regression (SVR) RMSE:

```
SVR Training RMSE: 0.07726099396955181
RMSE on valid data: 0.16332514568324108
```

-

- Support Vector Regression (SVR) Performance Visualization





- Random Forest (with and without cross validation) RMSE:

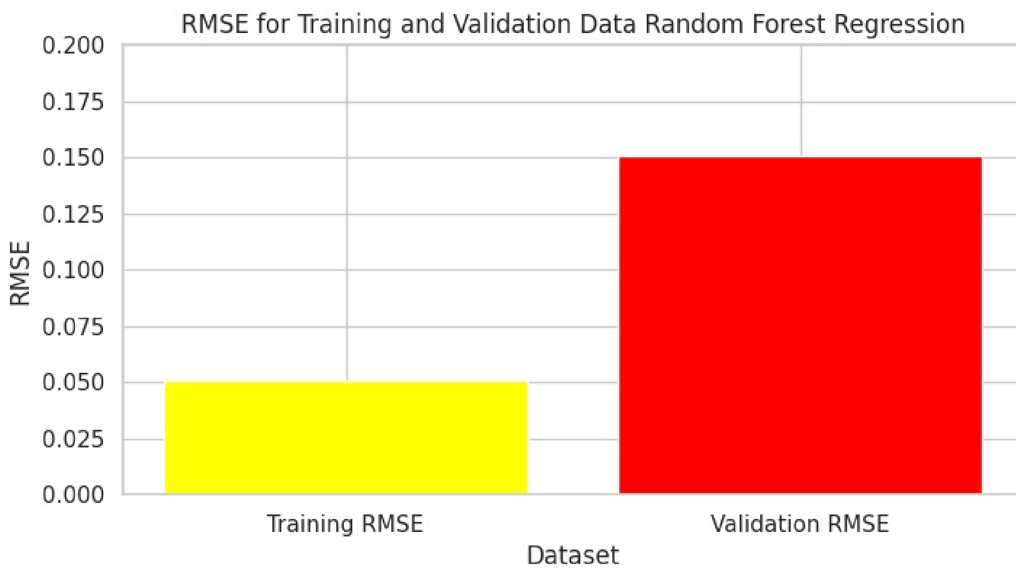
  - Without cross validation

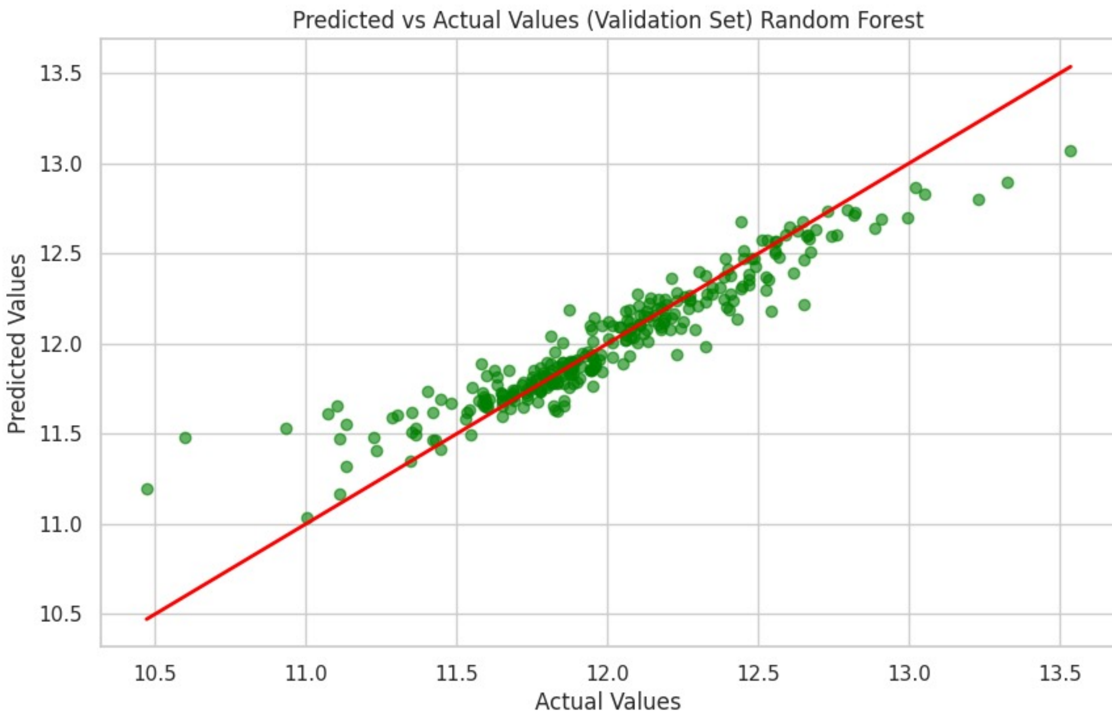
```
RMSE on training data: 0.04947083885761383
RMSE on valid data: 0.1484998875203375
```

  - With cross validation

```
Cross-Validation RMSE Scores: [0.12867426 0.14726364 0.16755438 0.12609939 0.11690885]
Mean RMSE: 0.1373001045310323
Standard Deviation of RMSE: 0.01805689881538686
RMSE on valid data: 0.15168344084146268
```

- Random Forest Visualization

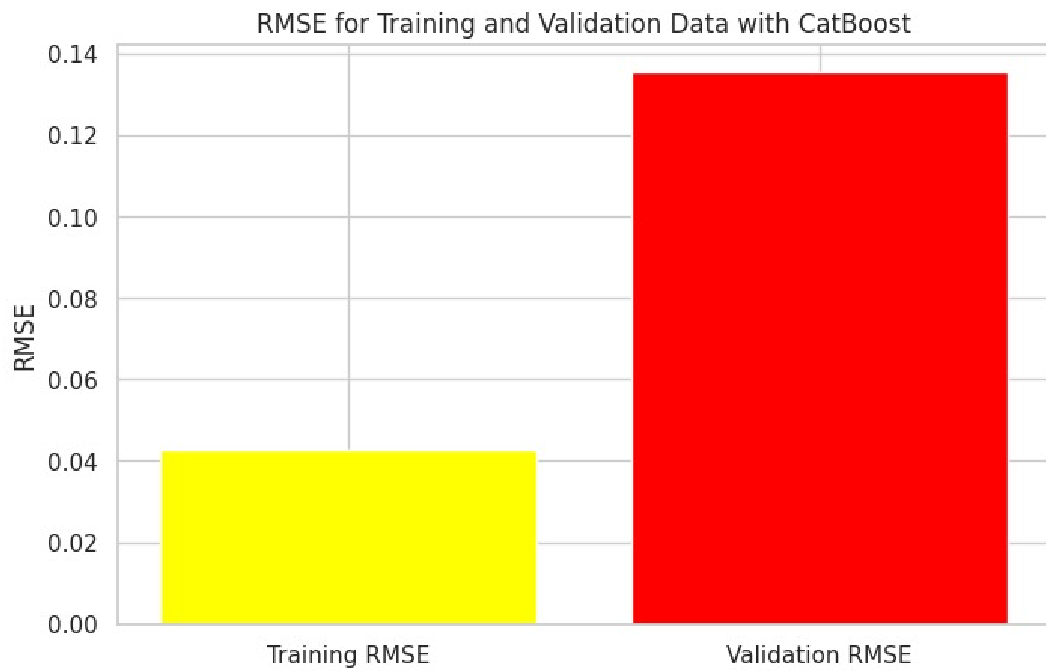


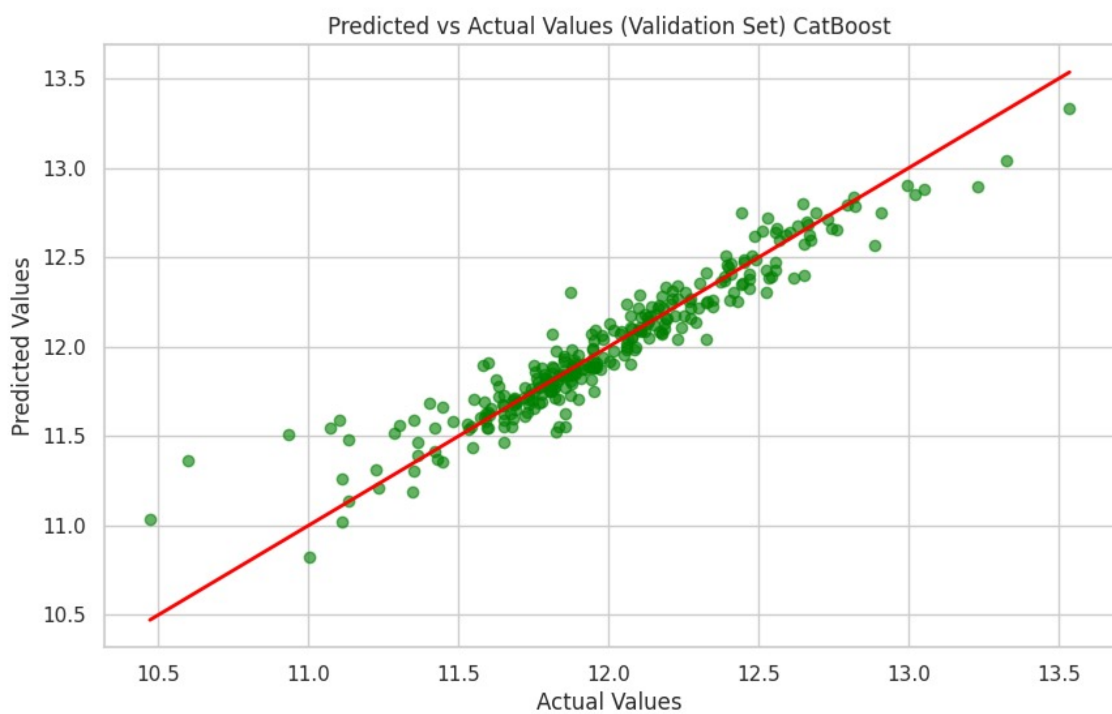


- Catboost RMSE:

```
CatBoost Regression Training RMSE: 0.041573477230806344
RMSE on valid data: 0.13619544929175664
```

- Catboost Performance Visualization

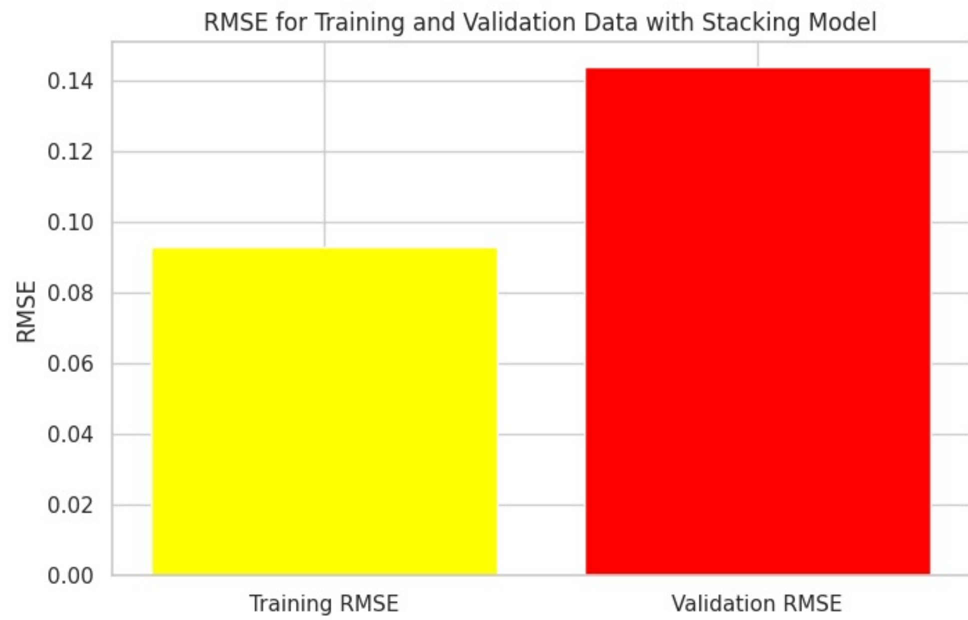


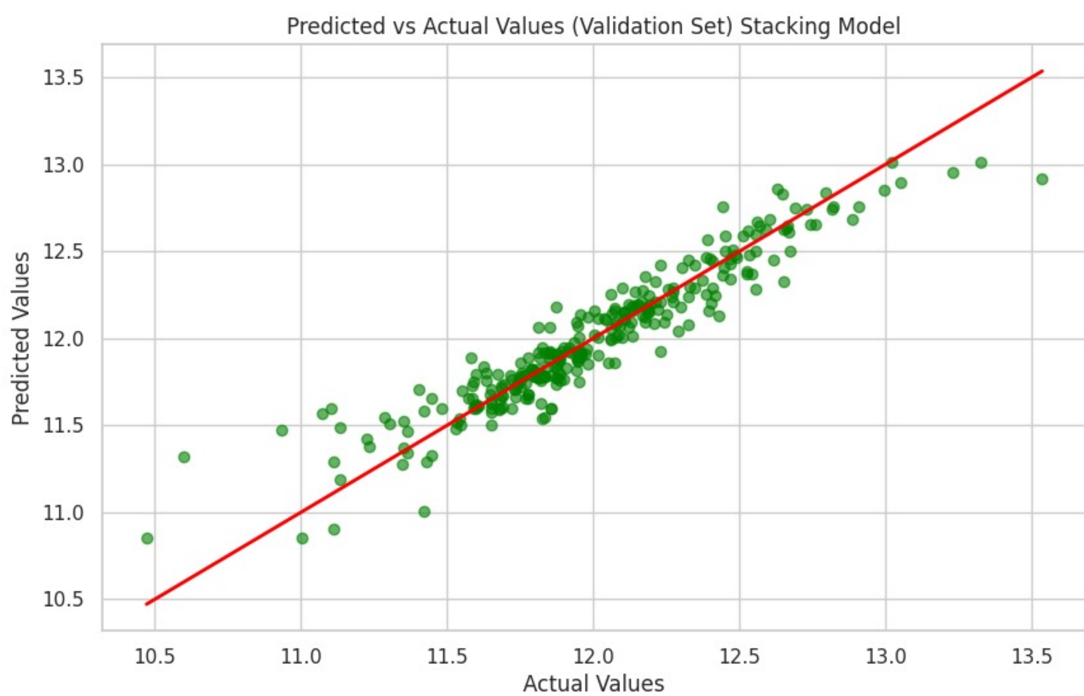


- Stacked Models RMSE:

```
Stacked Regression Training RMSE: 0.0899069894923901
RMSE on valid data: 0.14704525920989686
```

- Stacked Model Visualization

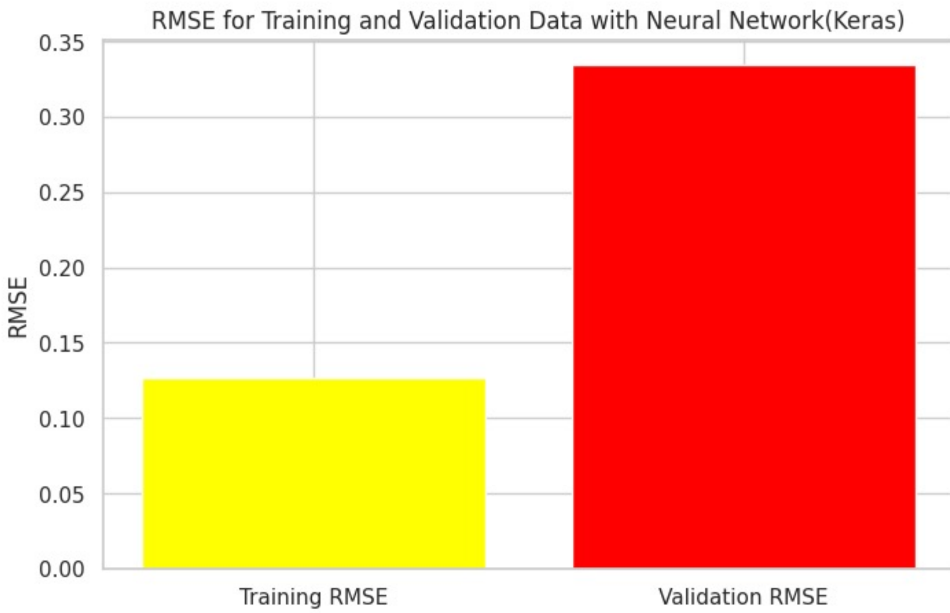


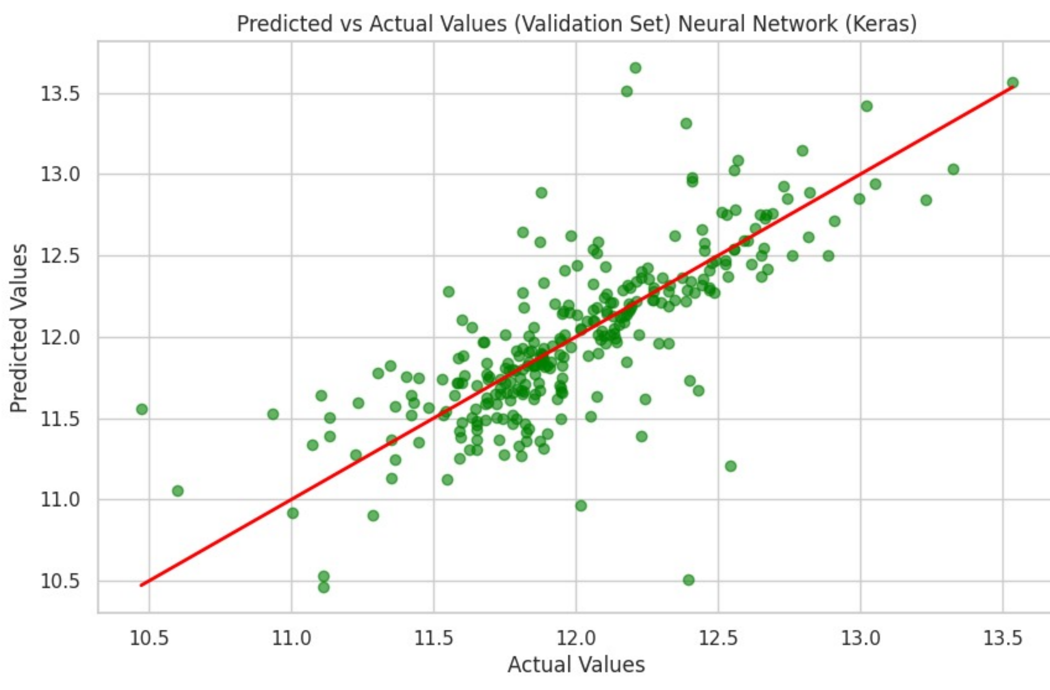


- Neural Networks RMSE:

```
Neural Network Training RMSE: 0.11296644060628722
10/10 [==============================] - 0s 3ms/step
RMSE on valid data: 0.3188546692411229
46/46 [==============================] - 0s 2ms/step
```

- Neural Networks Performance Visualization

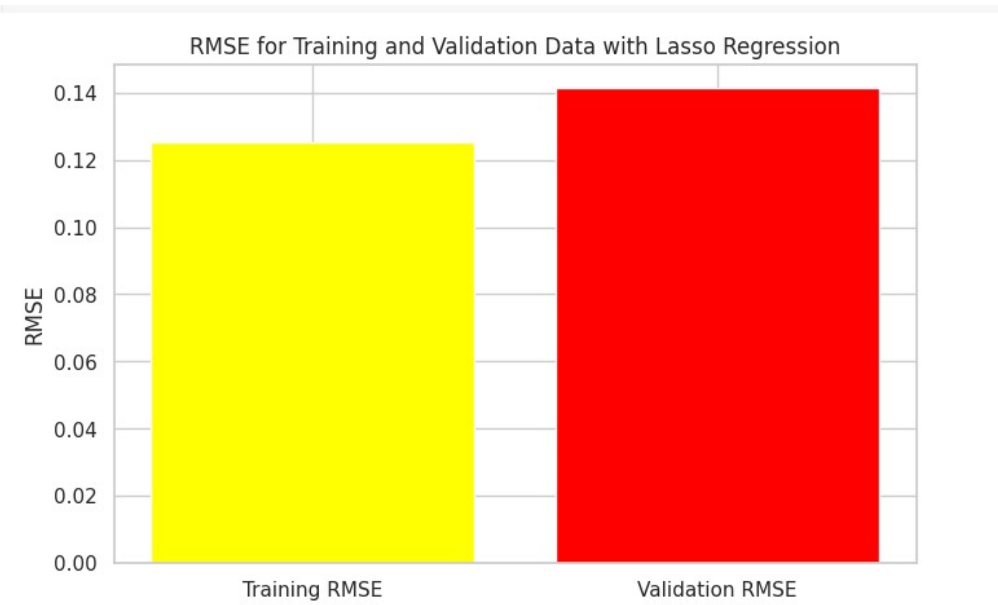


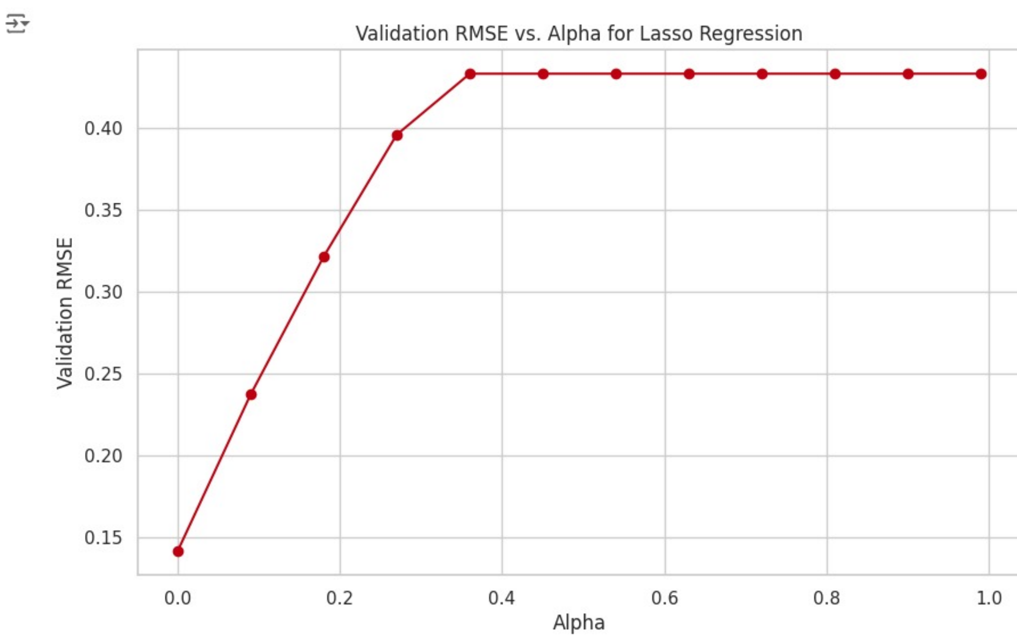


- Lasso Regression RMSE:

```
RMSE on training data: 0.12585344360985992
RMSE on valid data: 0.1423604613238065
```

- Lasso Regression Performance Visualization

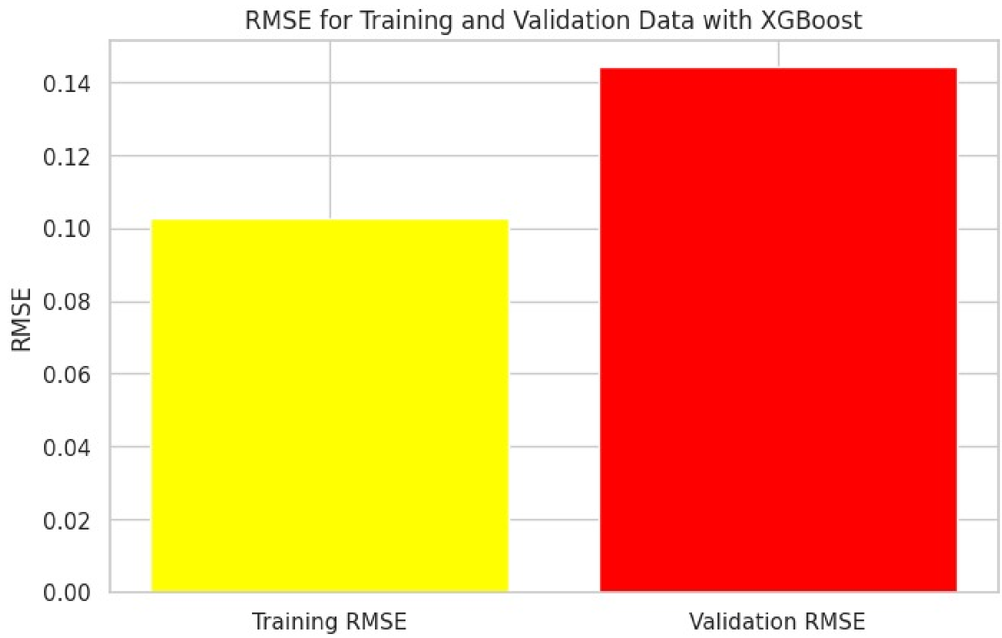


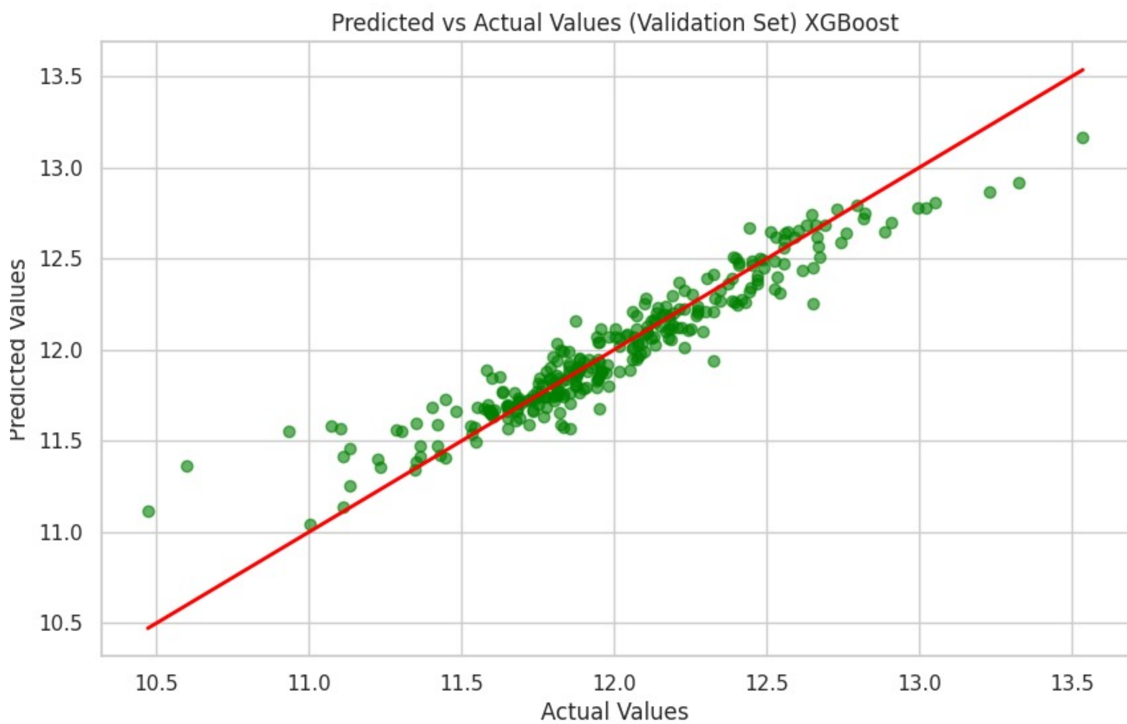


- XGBoost RMSE:

```
XGBoost Regression Training RMSE: 0.10287110868599329
XGBoost Regression VALIDATION RMSE: 0.14442919504597954
```

- XGBoost Performance Visualization





Out of all the models that we ran, Catboost model resulted in the lowest Root Mean Squared Error (RMSE) of 0.136 on the validation data.This outcome highlights Catboost's effectiveness in handling vast datasets having both categorical and numerical variables present. Catboost's ability to deliver predictive accuracy makes it a reliable tool for real estate price prediction tasks. This result not only supports the strength of Catboost models in scenarios with intricate data relationships but also supports its usefulness in providing actionable insights for data-driven decision-making in the real estate domain.

<u>CONTRIBUTION:</u>

- Ekaterina (Katie) Dyachenko: data cleaning, EDA and EDA through visualizations, ridge regression model, SVR model, model and rmse evaluations
- Isabella Gaitan-Salanga: Team leader, data cleaning, EDA, EDA visualization, XGBoost, hypertuning XGBoost
- Yashvi Mohta: data cleaning, EDA, catboost, stacked modeling regression, hypertuning randomforest, XGBoost, lasso hypertuning, model evaluations, model performance visualizations
- Nagasari Peri: data cleaning, EDA, random forest, neural networks, gradient boosting, hypertuning catboost, stacked and XGBoost.
- Zarafsha Uzzaman: data cleaning, feature engineering (interaction terms) , Lasso model Building , hypertuning catboost, catboost with cross validation

<u>DISCUSSION (what each of us have learnt):</u>

- Katie Dyachenko: This project provided me with valuable insights into the application of big data and cloud technologies in predictive modeling. By implementing and comparing the performance of different regression models, I have gained a better understanding of predictive algorithms and machine learning. Additionally, throughout this project, I have been honing my ability not only to code and think critically but also my ability to collaborate with a team when it comes to coding projects. While collaborating on Colab was challenging, overall, it was a great learning experience for me, which definitely shaped my future performances on projects similar to this in the future.

- Isabella Gaitan-Salanga: Through this project, I really learned about the importance of communication and consistency. There is a clear difference in experience with python and coding for the members of our group, and we had to learn to navigate and overcome these differences. We had to be understanding of one another regarding how long it took us to run certain models or learn how to go about any coding. Regarding consistency, we had to do a lot of communication regarding our different methodologies. At one point, we had to break down our code because we were faced with a hurdle and we had to work through it independently to try and see what was wrong. Once we were able to communicate and discern the problem, we were able to carry on more effectively. What also helped us throughout this project was being able to balance one another out throughout the project, showing the importance of well-rounded skill sets and teams. This project would have been much more difficult if we didn't have a team that was willing to support one another and step up to do certain tasks when it was necessary.
- Yashvi Mohta:This project taught me many valuable lessons that have significantly shaped my future performance on similar projects.First and foremost, I realized the critical importance of cleaning data and how essential domain knowledge is for effective feature selection and building. Throughout the model-building process, I learned the significance of hyperparameter tuning. My research into various ensemble models led me to discover CatBoost, which became our top-performing model. Additionally, I gained insights into stacked models, which enhanced our model's robustness. Evaluation curves emerged as a valuable concept for visualizing our progress and understanding model

performance over time. I developed a better understanding of machine learning principles and practical applications. This project also taught me how to work independently and as part of a team, balancing both approaches to maximize productivity and innovation. Overall, this experience has been instrumental and deepened my understanding and interest

- Nagasari Peri: This was a very interesting and fulfilling project as I got to have more exposure to different types of models. I was able to draw comparisons between different model and how each, with its own hyperparameters, could drastically underfit or overfit if caution was not implemented when choosing the hyperparameters.
- Zarafsha Uzzaman: The “freedom” of how to go about this project was an aspect that was challenging in the beginning. However, it provided real world experience in the sense that data analysis and predictive modeling is typically a trial and error process in the real world. The importance of data cleaning, feature engineering (with interaction terms to be specific), and hypertuning parameters of different kinds of models gave insight into how good model building requires a balance between training the model well while avoiding overfitting. For example, in the lasso model in our project, I quickly learned the importance of making sure to look at how different alpha values affect the model performance. Additionally, this project also showed me the importance of effective communication within teams. If a general plan is scheduled ahead of time and well thought out, it streamlines the process and makes for efficient use of everyone's time.

REFERENCES:

- OpenAI. (2024). *ChatGPT* (May 15, 2024 version). https://chat.openai.com

how to i plot rmse of one of my models?

To plot the Root Mean Square Error (RMSE) of a model, you generally follow these steps:

1. **Compute the RMSE** for each prediction you made with your model. If you're plotting RMSE over time or against different models, you'll need RMSE values for each time point or model.
2. **Prepare the data:** Ensure you have your RMSE values in a structure that can be used for plotting. This usually means having them in a list or an array if you're using Python, or in a column if you're working with a data frame.
3. **Choose a plotting library**: In Python, common libraries for plotting include Matplotlib, Seaborn, or libraries specific to the context, like Plotly for interactive plots.
4. **Plot the data**: Create a plot that best represents your data. If you're showing how RMSE changes over time or against various parameters, a line plot might be appropriate. For comparing RMSE between different models, a bar plot could be better.

- Coding references are referenced in the Colab file