***Factors Influencing Residential Properties Sales Price Pre- Covid and Post-Covid and A Machine Learning Approach on Predicting Market Value in 2023 for Residential Properties***

MANSI MAHAJAN, PASCAL NTAGANDA, and JIBOK PARK

BUAN 5510

Dr. Ben Kim

Aug 11, 2023

[Abstract 2](#_Toc178799086)

[1. Introduction 3](#_Toc1817900049)

[a. Problem Statement 4](#_Toc199328655)

[b. Contribution 4](#_Toc413104755)

[c. Background Information: 4](#_Toc1777084702)

[2. Description of Datasets and Data Exploration 5](#_Toc145017155)

[a. Description of Datasets 5](#_Toc433350341)

[b. Data Exploration 5](#_Toc793591463)

[3. Literature Reviews 7](#_Toc1332987964)

[4. Data Pre-Processing 9](#_Toc1052786545)

[a. Processing of NULL values 10](#_Toc1600115099)

[c. Merging external datasets 11](#_Toc1001429045)

[e. Use clustering for preprocessing 13](#_Toc2102872074)

[5. Data Mining Models and Evaluations 14](#_Toc1455998733)

[a. Evaluation Metrics: 14](#_Toc1659579752)

[b. Statement1 15](#_Toc335764117)

[c. Statement2 19](#_Toc1300500177)

[6. Discussion 22](#_Toc796907834)

[7. Conclusion 23](#_Toc505941315)

[References 25](#_Toc2127237033)

[Appendices 27](#_Toc519784736)

[Data Dictionary: 34](#_Toc1222009595)

[Code 34](#_Toc2122843073)

# **Abstract**

This study investigates the significant features influencing the sales price of residential properties both pre and post Covid-19, aiming to identify potential variations in factors affecting property values during these periods. In addition, we aim to build a machine learning model capable of predicting the total market value of residential properties.

The impact of Covid-19 on various industries, including real estate, has been substantial, and understanding the changes in property sales dynamics is crucial for making informed decisions in the current market. By comparing the influential features for residential property sales price pre and post Covid-19, we seek to uncover any shifts in market demand and preferences that may have occurred due to the pandemic.

To ensure a homogeneous subset of properties, the dataset is narrowed down to a specific time range, reducing the impact of outliers, and ensuring that the properties considered share similar characteristics and market conditions. This approach enhances the reliability and accuracy of our predictions.

Overall, this study seeks to contribute valuable insights into the factors influencing residential property sales prices both pre and post Covid-19, and to develop an effective machine learning model for predicting the total market value in 2023 for residential properties built in the last ten years. The findings may provide valuable guidance for real estate professionals, investors, and policymakers in navigating the dynamic real estate market and to make informed decisions.

# **Introduction**

1. **Problem Statement:**
2. What are the significant features impacting the Sales Price of Residential Properties pre Covid and post Covid?
3. To build a machine learning model for predicting the Total Market Value in 2023 for Residential Properties built in the last 10 years.

## **Contributions**

This study makes several noteworthy contributions to the field of real estate analysis and data-driven decision-making. Firstly, by analyzing the factors influencing residential property sales prices both before and after the pandemic, our research provides valuable insights into the shifting landscape of buyer preferences and market trends brought about by the COVID-19 global health crisis. This enhanced understanding of the pandemic's impact on the real estate market aids in predicting and adapting to long-term changes in housing demand and pricing. Secondly, our implementation of a machine learning model for predicting market value of residential properties in 2023 highlights the application of advanced data analysis techniques to the real estate domain. This novel model facilitates accurate market value predictions based on historical data and property attributes, empowering stakeholders such as property owners, investors, and policymakers to make informed decisions for the upcoming tax year. Additionally, our focus on properties built within the last ten years provides specific insights into the behavior of newer properties in response to evolving market dynamics and buyer preferences. The practical implications of our research extend to real estate agents, developers, and investors, guiding them on how to adapt their strategies and offerings to cater to the changing needs and expectations of buyers in a post-pandemic environment. Overall, our study contributes significantly to the growing body of knowledge concerning the real estate market's response to significant global events and demonstrates the importance of employing machine learning techniques for data-driven predictions in the housing sector.

## **Background Information:**

The real estate market has always been influenced by a wide variety of factors that influence property values. However, the COVID-19 pandemic's emergence changed the dynamics of the housing market. This research paper builds upon prior research and develops several machine learning models to understand the significant factors influencing Sales Price of Residential Houses in Pierce County, WA based on the properties characteristics as well as quality of local schools. In addition, the study aims to develop a predictive model to estimate the total market value in 2023 for Residential properties built in the Pierce County over the last ten years. The results of this study can offer valuable insights for homebuyers, sellers, real estate agents, and policymakers to the broader understanding of the real estate market dynamics and provide valuable insights for various stakeholders in the industry.

# **2.** **Description of Datasets and Data Exploration**

## **Dataset Description**

We sourced property information data1) from the Pierce County Assessor-Treasurer’s Data website to facilitate our study. The dataset encompasses a collection of nine distinct tables, each offering a unique perspective on property-related attributes. For our research objectives, we have chosen to focus on five key datasets: Appraisal Account, Improvement, Improvement Built-as, Sale, and Tax Account. The Appraisal Account table had 6.0 MB of data, distributed across 24 attributes, and 338,546 individual data records. The Improvement table contributes an additional 3.8 MB of data, with 356,693 records and 25 attributes that show property improvements.

The Improvement Built-as table has a size of 4.3 MB, and 359,476 records intricately woven into 26 attributes. Subsequently, the Sale table, recognized as the biggest dataset, with 18.2 MB in size. This table has 598,670 rows of sale data spanning from 1997 to 2022, with 13 attributes. The Tax Account table, our final focal dataset had 8.5 MB in size and comprised of 350,472 rows of data. This table had 28 attributes. Address Point data sourced from Pierce County was also used. This integration enabled us to effectively merge property-related information with the school scores, thereby fostering a holistic understanding of the underlying relationships and effects.

## **Data Exploration**

Initially, we conducted a correlation analysis due to the potential adverse effects of multicollinearity, including unstable model coefficients, challenges in interpretation, and decreased model generalization. As a result, we identified pairs of attributes with correlations exceeding 0.7 and retained only one attribute from each pair. This process led to the removal of 6 attributes from the Appraisal, 2 from the Improvement, 4 from ImprovementBuiltAs, and 1 from Sale, as well as 22 attributes from the Tax Account. Additionally, non-essential string-type attributes like ID or **code** were excluded before dataset merging.

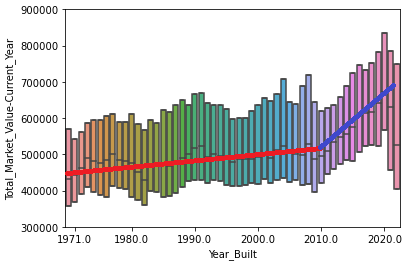
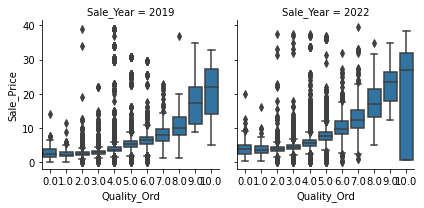
Upon merging the datasets, we observed that dataset for statement 1 contained 11,295 records for 2019 and 10247 for 2022 with 44 attributes related to sale price data, while statement 2 featured 17,246 and 24 attributes associated with tax data. Due to varying record counts in the initial tables, employing an outer join resulted in null values, prompting us to investigate further.

We tried to understand the trends within the dependent attributes, namely Sale Price and Market Value, and their correlations with other attributes. Notably, the mean Sale Price in 2022 exceeded that of 2019, accompanied by a wider price range. (Fig1) An intriguing insight emerged as house quality exhibited an exponential relationship with price, indicating that superior quality correlated with significantly higher prices. (Fig2)

Comparing transaction patterns between 2019 and 2022, we observed a shift in peak activity from the summer months to an earlier onset in March for the latter year, followed by a subdued summer. Despite a higher transaction volume in 2019, price fluctuations were more pronounced in 2022. Notably, the appreciation in house prices associated with increased square footage was more rapid in 2022 than in 2019, implying a preference for larger homes in the former year. Driven by these findings, our focus turns to a deeper exploration of how individual attributes exert distinct influences on house prices during the years 2019 and 2022.

Shifting our gaze to the taxable market value based on the year of construction, a conspicuous pattern emerges; houses built in recent years exhibit a distinct trend compared to those constructed before the early 2010s. This distinction prompts us to consider employing separate Machine Learning models for new and old houses, with our immediate attention centered on the dynamics of new houses.

Figure1. Sale Price by Quality in 2019 vs 2022 Figure2. Market Value by Year Built



# **Literature Reviews**

1. **Predictive analytics using Big Data for the real estate market during the COVID-19 pandemic 2)**

The article aimed to identify apartment attributes influencing price revisions during the pandemic using SHAP values and exploring the impact of Time on the Market (TOM) on apartment prices. The three-step methodology involved data mining through web scraping, resulting in 18,992 data points with 16 variables. They then conducted data cleaning and preparation, rearranging variables for analysis and adding dummy variables. Employing 15 machine learning algorithms, the study found that apartment prices remained resilient during the pandemic, with TOM and initial price setup emerging as the most dominant and consistent variables influencing price revisions.

1. **Housing Price Prediction Using Machine Learning Algorithms in COVID-19 Times3)**

This study pursued two main objectives: to identify optimal machine learning algorithms for predicting housing prices and to assess the pandemic's impact on housing prices in Alicante, Spain. Approximately 40,000 property samples from 2019 to 2021 were collected and analyzed using web scraping to understand COVID-19's influence on property prices. The data cleaning process involved adjusting variables causing multicollinearity and removing extreme data points. Boosting-based algorithms outperformed basic linear methods. Key predictors for housing prices were floor area, number of bathrooms, and elevator presence. Despite using data from different periods, regional characteristics of coastal cities played a decisive role. Compared to other regions, housing prices were lower, and net household income significantly influenced purchasing power and predicted housing prices. The pandemic's impact was localized and temporary, displaying housing market resilience.

1. **Real estate price forecasting based on SVM optimized by PSO (Particle Swarm Optimization) 4)**

The paper explores the application of Support Vector Machines (SVM) optimized by Particle Swarm Optimization (PSO) in predicting real estate prices. The study aims to improve the accuracy of real estate price forecasts by leveraging the combined strengths of SVM and PSO. The authors demonstrate the effectiveness of their proposed approach through experiments and analysis. The findings suggest that the SVM-PSO model can provide reliable predictions for real estate prices, offering potential benefits for real estate market analysis and decision-making.

**d.** **Price forecast in the competitive electricity market by support vector machine5)**

The paper focuses on the use of Support Vector Machines (SVM) for predicting prices in the competitive electricity market. The study aims to enhance price forecasting accuracy in this dynamic market environment. The authors propose an SVM-based model and evaluate its performance using historical electricity price data. The findings highlight the effectiveness of the SVM approach in providing reliable price forecasts, thereby supporting decision-making in the competitive electricity market.

1. **Housing value forecasting based on machine learning methods6)**

In this paper the researchers used housing data from various Boston suburbs with 13 features, including per capita crime rate. This paper argues that ANN’s are not an optimal method to use for predicting housing prices, because its accuracy is low and convergence speed is far from ideal. Instead, this paper uses different models including SVM, LSSVM and PLS to analyze housing values.

The best performing model for this sample group was the Support Vector Machine, which had a mean square error estimate of 10.7373, while the other two took longer to run and had a larger mean square of error estimate. SVM performed better than LSSVM, due to LSSVM being a slightly more simplified mathematical version of SVM. The PLS algorithm's forecast results for the home values in the Boston suburb dataset were not satisfactory due to the significant presence of nonlinearity. Additionally, the computation efficiency of the PLS algorithm was relatively low. Several machine learning models should be constructed and analyzed, as this is a very important part of any analysis. The models should also be combined with corresponding characteristics of testing data to predict the housing values.

1. **A Comparison of Regression and Artificial Intelligence Methods in a Mass Appraisal Context7)**

The paper presents a comparison analysis that assesses property prices in Louisville, Kentucky using several regression and machine learning methods. The dataset for this study consisted of 309,000 properties with 143 descriptive variables. In addition to this assessment data the researchers were also provided with a data set containing over 16,000 sales transactions. These large data sets gave them the ability to conduct a more comprehensive comparative study than had been conducted in the past. For the regression methods, the researchers used one traditional multiple regression and three non-traditional regression methods including a support vector machine (SVM). The models utilized in the AI (Artificial Intelligence) section of the study included neural networks (NN), radial basis function neural networks (RBFNN), and memory-based reasoning (MBR).

The above models were all tested using five distinct error measures: MAE (Mean Absolute Error), RMSE (Root Mean Squared Error), RAE, RRSE, and R2. The study found that AI methods, particularly Artificial Neural Networks, outperformed traditional regression techniques in terms of prediction accuracy. The neural network models captured complex correlations between property attributes and market values, resulting in more accurate and reliable predictions.

# **Data Pre-Processing**

Our research focused on analyzing the impact of factors on Residential Sales Prices before and after the Covid pandemic. To achieve this, we utilized specific datasets, including ***Sales Data, Improvement Data, Improvement BuiltAs Data, and Appraisal Account Data****.* We filtered the Property type to include only Residential properties, and then narrowed down the data further by selecting residential property sales that occurred in 2019 and 2022.

For predicting Market Value, we utilized datasets such as **A*ppraisal Account Type, Improvement, Improvement Built As, and Tax Account*.** We only considered the dataset for properties built on or after 2014 to 2023. To address any issues with outliers in the Market Value the final market value of properties was limited to sales between $100,000 and $30,000,000.

## **Processing of NULL values**

Before merging the property data tables, every data table was independently analyzed, and most variables with Null ratios of more than 30% were eliminated while the few relevant ones like *Waterfront Type, View Quality are* replaced with Null Values for ‘Not Applicable’. Also, the null values for variables such as ‘*Attic Finished Square Feet,’ ‘Basement Square Feet,’ ‘Carport Square Feet’, ‘Porch Square Feet’, ‘Attached Garage Square Feet’, and ‘Detached Garage Square Feet’* were replaced with zero. In addition, the attributes with 10% to 30% of missing values such as *Exterior, Interior, Roof Cover* are replaced with the Null Value by assigning an additional category ‘Not Applicable’. Finally, we eliminated the rows with nulls in the columns less than 10% of Null Ratio columns to make sure that variables did not contain any null values,

1. **Converting the nominal, ordinal, date/time data to numbers**
2. **Nominal**

Nominal variables are variables that have distinct categories or labels without any inherent order or ranking. The nominal variables *HVAC Description, Utility Water, Utility Electric* etc. were converted to dummy variables using one hot encoding. One-hot encoding creates separate binary features for each category, which allows the algorithm to treat each category independently.

1. **Ordinal**

An ordinal attribute is a specific type of categorical data characterized by meaningful order or ranking among its categories. Unlike nominal attributes, which lack any inherent order and consist of distinct labels, ordinal attributes exhibit a clear relative ordering among their categories. In our dataset, the *‘view quality’, 'condition' and 'quality'* attributes are ordinal variables and changed to numeric to capture the inherent order among categories.

Table1: Ordinal variables

|  |  |
| --- | --- |
| Table | Ordinal attributes |
| Improvement | Condition: 0'Uninhabitable ~ 8 Excellent  Quality: 0 Low ~10 Excellent |

1. **Dates**

In the Pierce County dataset, we encountered date or year information represented as string types, necessitating their conversion to the appropriate date data type. This transformation was crucial to ensure accurate utilization of these columns in our analysis.

In addition to removing variables with high Null Ratios, variables with no correlation to Sale Price and Market Value or variables with extremely high correlation to other variables were also removed. Once the initial set of important variables was identified the property data tables were joined with ‘Parcel Number’ into one master table.

## **Merging external datasets**

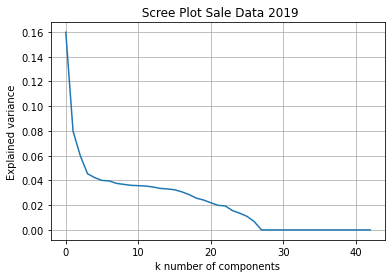
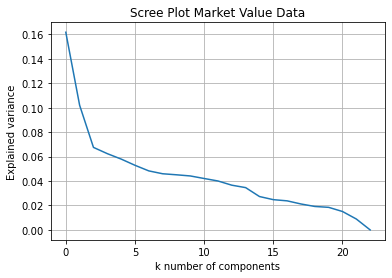
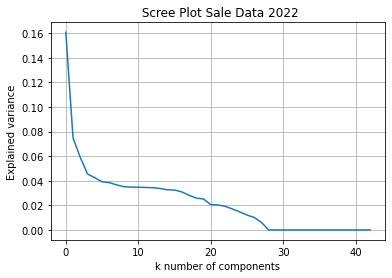
Incorporating school scores into our dataset can significantly enhance our analysis, as the quality of schools in the neighborhood directly influences house prices8). Given that our research is focused on Sales Price and Market Value for Residential properties, school scores serve as a crucial attribute to consider. Obtaining this valuable information required the integration of three datasets. Firstly, we sourced school scores from the School Digger 9) webpage, which provides a comprehensive list of elementary schools, their corresponding zip codes, and school scores ranging from 0 to 10. To merge the school dataset with master dataset, we utilized another dataset Address Points10) that contains ‘Tax Parcel Number’ and contains the relevant location (X, Y coordinate, Zip Code) information. The ‘Tax Parcel Number’ of the Address points was joined with the master property table ‘Parcel Number’. For each property, we then computed the average school score by considering all schools located within a 2-mile radius of the property and are in the same zip code.

Once all of the cleaning was done and all relevant variables were selected and the master property data set for 2019 Sales data consisted of 11,295 observations and 44 attributes while for 2022, Sales Data consisted of 10,247 with 44 attributes. For Market Value prediction, the final dataset consisted of 17,246 and 24 attributes.

1. **Reduction of features using PCA (Primary Component Analysis) and 'Importances'.**

***PCA*** (Principal Component Analysis) is used for dimensionality reduction, which transforms the original features into uncorrelated principal components. In Principal Component Analysis (PCA), we have used the Scree Plots which is an effective way to understand and decide how many principal components we should retain for our analysis. For instance, from the scree plot in *Figure3*, we can observe that the variance is decreasing with increase of components, however after k=28 variance value is becoming constant. So, k= 28 seems to be the appropriate component (feature) value, which we can use for further analysis.

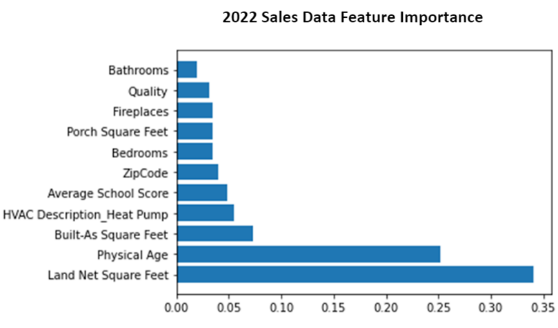
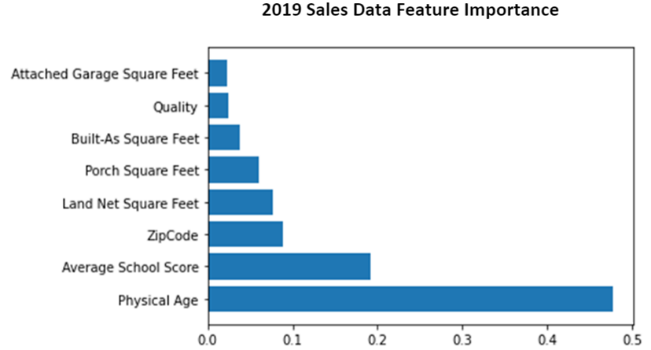
*Figure3:Scree Plots(2019)* *Figure4: PCA Scree Plots(2022)*  *Figure5: Scree Plot(Market Value)*

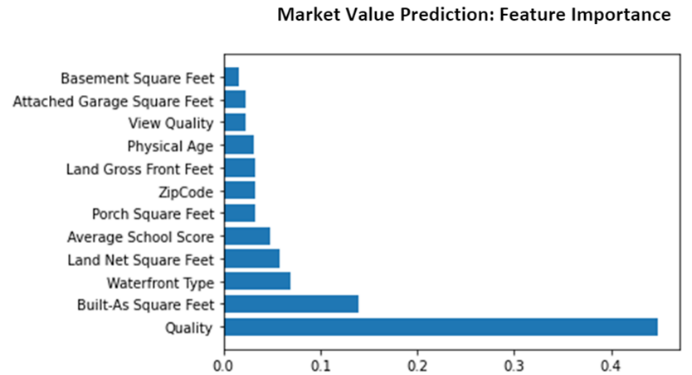
In our first problem statement, we aim to determine the most significant features that influence Residential property Sales Price both pre and post Covid. Feature selection will help us identify the most relevant features and their importance values, providing insights into any changes in feature importance and features between the Pre-Covid and Post-Covid periods. Another thing to note is that the more accurate our model is, the more we can trust feature important measures and other interpretations. For ***2019*** ***Sales*** data the important features *(Figure 6) identified* by Random Forest Regressor are *'Land Net Square Feet', 'Quality', 'Porch Square Feet', 'Attached Garage Square Feet', 'Built-As Square Feet', 'Physical Age', 'ZipCode', 'Average School Score'.* For ***2022 Sales***data the selected important features *(Figure 7)* are: *'Land Net Square Feet', 'Quality', 'Porch Square Feet', 'Attached Garage Square Feet', 'Fireplaces', 'Built-As Square Feet', 'HVAC Description', 'Bedrooms', 'Physical Age', 'ZipCode','Average School Score'*

For ***Market Value Prediction*** Data, the most important features *(Figure 8*) are: *'Land Net Square Feet', 'Land Gross Front Feet', 'View Quality', 'Waterfront Type', 'Quality', 'Basement Square Feet', 'Porch Square Feet', 'Built-As Square Feet', 'Physical Age', 'ZipCode', 'Average School Score'*

*Figure 6: Feature Importance (2019)*  *Figure 7: Feature Importance (2022)*



*Figure 8: Feature Importance (Market Value)*

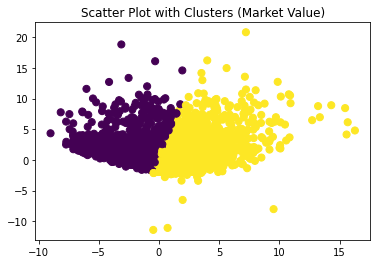
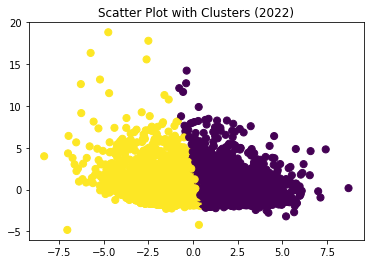
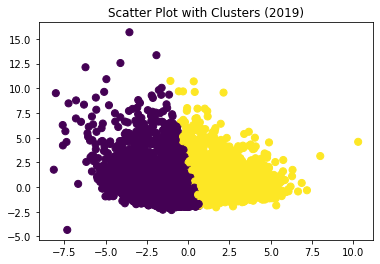


In conclusion, PCA and Feature Selection are two different techniques used for feature reduction and feature importance, respectively. Both techniques can be used to simplify the dataset and play a vital role in data preprocessing and model development.

## **Use clustering for preprocessing**

K-Means clustering is a popular unsupervised machine learning algorithm used for classification on unlabeled data. However, another application of clustering is in data preprocessing, where it is used to segment the data into more manageable and easier-to-predict subsets. In our case study, the data will be segmented into two distinct clusters that will be used for further analysis due to containing the most data.

*Figure 9: Scatter Plot (2019)* *Figure 10: Scatter Plot (2022)*   *Figure 11: Scatter Plot (Market Value)*



The *Silhouette Score* is a metric we used to evaluate how well-separated the clusters are. It ranges from -1 to 1, with higher scores indicating better-defined and more distinct clusters. The ***Sales 2019 & 2022*** dataset *(Table 2)* has a Silhouette score is 0.44 & 0.45, while the ***Market Value*** Dataset has a Silhouette score is 0.48, which suggests that the clusters in this dataset are relatively well-separated. From (*Figure 9, 10, 11)* we can see the distribution of Clusters for each dataset.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Table 2: K-means Clustering* | | | | |
| *Data* | | ***Silhouette Score*** | ***Numbers of Records*** | |
|  |  | ***Cluster 1*** | | ***Cluster 2*** |
| *Sales Data 2019* | | 0.44 | 5077 | 6108 |
| *Sales Data 2022* | | 0.45 | 5399 | 4729 |
| *Market Value Data* | | 0.48 | 8278 | 9009 |

# **Data Mining Models and Evaluations**

## **Evaluation Metrics:**

In this project, various machine learning algorithms are used in order predict the value of property; random forests, gradient boosting, and neural networks (Mora-Garcia, Cespedes-Lopez, & Perez-Sanchez, 2022). The metrics used for evaluation are R squared, Weighted Average

R-squared (R²) is a commonly used evaluation metric in machine learning for regression tasks. It measures the goodness of fit of a regression model, indicating how well the model explains the variance in the target variable (dependent variable) based on the input features (independent variables).

***R-squared = 1 - (SSR (Sum of Squared Residuals) / SST)***

where:

SSR (Sum of Squared Residuals) is the sum of the squared differences between the predicted values and the actual target values.

SST (Total Sum of Squares) is the sum of the squared differences between the actual target values and the mean of the target values.

To evaluate the performance of the models across various clustered data the weighted average of cluster results is calculated using the formula below where m = metric and r = # of rows for each cluster.

***Weighted Average = (m1\*r1 + m2\*r2 + m3 \* r3) / ∑ ri***

***k-fold Cross Validation:*** k-fold cross-validation is a model evaluation technique used to assess the performance of machine learning models.

For both Sales and Market Value datasets, k-fold cross-validation with k=5 is applied. The data is divided into 5 folds, and the model is iteratively trained on 4 folds while testing on the remaining fold. This process is repeated 5 times, ensuring each fold is used for testing once. K-fold cross-validation helps prevent overfitting and provides a more reliable evaluation of the model's performance by avoiding dependency on a specific test and training dataset.

1. **Model Evaluations:**

## **Statement 1:**

Data mining and model evaluation is done to identify the best fitting machine learning models for sales data in ***2019*** and ***2022***. Initially, four machine learning models, namely Random Forest Regression, Gradient Boosting, Simple Vector Machines, and Neural Network Models, were deployed for each year's sales data. After hyperparameter tuning, the top-performing models were determined as the Random Forest Model (RFM) (*n\_estimators = 1000, min\_samples\_split=15*) and Gradient Boosting (GBR) models(*n\_estimators = 750, min\_samples\_split=10*) and Gradient Boosting (GBR) models, based on the evaluation metric R-squared, which calculates the percentage fit of the model to the data.

For ***2019 Sales*** ***data***, the RFR and GBR models exhibited high R-squared values of 0.90 and 0.89. With three hidden layers *(50, 40, 20)* Neural Network has 0.80 R-squared which is less than RFR (Random Forest Regression) and GBR models.

|  |  |  |  |
| --- | --- | --- | --- |
| ***Table 3: Results for Sales Dataset 2019*** | | | |
| ***Evaluation Metric: R Squared*** | | | |
| ***Model*** | ***With All Features*** | ***Feature Importance*** | ***PCA*** |
| ***Random Forest Regressor*** | 0.90 | 0.90 | 0.82 |
| ***Gradient Boosting Regressor*** | 0.89 | 0.89 | 0.80 |
| ***Neural Network*** | 0.80 |  | 0.72 |

In addition, the Feature importance *(Table 3)* for RFM and GBR resulted in similar R-squared value which is 0.90 and 0.89 respectively. By identifying and selecting the most prominent features, the feature selection process provided valuable insights into which variables have the most significant impact on the Sales Price of Residential Properties pre-Covid.

Subsequently, Principal Component Analysis (PCA) was applied to reduce the dimensionality of the data. This led to a decrease in the R-squared values for the RFM and GBR models in ***2019*** to 0.82 and 0.80, respectively. However, despite the reduction, the models still retained respectable predictive capabilities, indicating that PCA successfully preserved essential information while reducing the feature space.

|  |  |  |  |
| --- | --- | --- | --- |
| *Table 4: Clustering Results: Sales Dataset 2019* | | | |
| *Evaluation Metric: R Squared* | | | |
| *Model* | ***Cluster 1*** | ***Cluster 2*** | ***Weighted Average*** |
| *Random Forest Regressor* | 0.87 | 0.91 | **0.89** |
| *Gradient Boosting Regressor* | 0.93 | 0.87 | **0.90** |
| *Neural Network* | 0.75 | 0.80 | **0.77** |

|  |  |  |  |
| --- | --- | --- | --- |
| ***Table 5: Clustering Results (Feature Importance/PCA): Sales Dataset 2019*** | | | |
| ***Feature Importance: Evaluation Metric: R Squared*** | | | |
| ***Model*** | ***Cluster 1*** | ***Cluster 2*** | ***Weighted Average*** |
| ***Random Forest Regressor*** | 0.93 | 0.78 | **0.88** |
| ***Gradient Boosting Regressor*** | 0.87 | 0.75 | **0.82** |
| ***Principle Component Analysis: Evaluation Metric: R Squared*** | | | |
| ***Random Forest Regressor*** | 0.75 | 0.82 | **0.78** |
| ***Gradient Boosting Regressor*** | 0.74 | 0.78 | **0.76** |
| ***Neural Network*** | 0.60 | 0.68 | **0.77** |

Furthermore, for 2019 data, after applying K-Nearest Neighbor clustering into two groups, achieved a silhouette coefficient of 0.45, indicating moderately good clusters with some cohesion within the groups. However, some points showed proximity to other clusters or overlaps. These clusters performed well, with R-squared values of 0.89 for RFM and 0.90 for GBR *(Table 5)*. However, after dimensional reduction techniques such as PCA were applied to these clusters, the R-squared values decreased to 0.78 for RFM and 0.76 for GBR. In addition, the models with important set of features still maintained 0.88 and 0.82 R squared values.

In the ***2022 Sales data*** *(Table 6)*, the RFM and GBR models first showed strong performance with R-squared values of 0.87 and 0.88, respectively. The three-layer *(50, 40, 20)* Neural Network model has an R-squared value of 0.82 which is slightly is slightly less than RFR and GBR. In addition, the Feature selection for RFM and GBR resulted in similar R squared value which is 0.88 and 0.89 respectively. By identifying and selecting the most prominent features, the feature selection process provided valuable insights into which variables have the most significant impact on the Sales Price of Residential Properties post Covid.

|  |  |  |  |
| --- | --- | --- | --- |
| *Table 6: Results for Sales Dataset 2022* | | | |
| *Evaluation Metric: R Squared* | | | |
| *Model* | ***With All Features*** | ***Feature Importance*** | ***PCA*** |
| *Random Forest Regressor* | 0.87 | 0.88 | 0.80 |
| *Gradient Boosting Regressor* | 0.88 | 0.89 | 0.83 |
| *Neural Network* | 0.82 |  | 0.70 |

However, after applying PCA, the R-squared values decreased slightly to 0.80 for RFM and 0.83 for GBR. Nevertheless, both models retained considerable predictive power, suggesting that PCA effectively captured essential information while reducing the feature space. Overall, these results indicate that both models remain reliable for predicting sales trends in 2022, though with a slightly decreased level of accuracy compared to the original models without PCA. The Gradient Boosting model slightly outperformed the Random Forest model.

|  |  |  |  |
| --- | --- | --- | --- |
| *Table 7: Clustering Results: Sales Dataset 2022* | | | |
| *Evaluation Metrics: R Squared* | | | |
| *Model* | ***Cluster 1*** | ***Cluster 2*** | ***Weighted Average*** |
| *Random Forest Regressor* | 0.75 | 0.82 | **0.77** |
| *Gradient Boosting Regressor* | 0.77 | 0.81 | **0.76** |
| *Neural Network* | 0.70 | 0.81 | 0.75 |

Moreover, when creating two clusters for the 2022 data, the silhouette coefficient was 0.45, but these clusters exhibited the lowest R-squared values (*Table 7*), with a weighted average of 0.77 for RFM and 0.76 for GBR. After applying PCA to these clusters (*Table 8*), the R-squared values further decreased to 0.75 for RFM and 0.72 for GBR. In addition, the models with selected features have shown decreased R squared performance for both RFR and GBR.

|  |  |  |  |
| --- | --- | --- | --- |
| *Table 8: Clustering Results (Feature Importance/PCA): Sales Dataset 2022* | | | |
| *Feature Importance: Evaluation Metric: R Squared* | | | |
| *Model* | ***Cluster 1*** | ***Cluster 2*** | ***Weighted Average*** |
| *Random Forest Regressor* | 0.76 | 0.70 | **0.72** |
| *Gradient Boosting Regressor* | 0.70 | 0.65 | **0.67** |
| *PCA: Evaluation Metric: R Squared* | | | |
| *Random Forest Regressor* | 0.72 | 0.77 | **0.75** |
| *Gradient Boosting Regressor* | 0.75 | 0.73 | **0.72** |
| *Neural Network* | 0.61 | 0.62 | **0.60** |

In conclusion, the data mining and model evaluation process provided valuable insights into the performance of machine learning models for sales data in both 2019 and 2022 (Pre and Post Covid). The results demonstrate that both RFM and GBR models are reliable predictors for factors influencing Sales Price, and the application of PCA and clustering techniques influenced the models' accuracy, preserving essential information while reducing feature space. These findings contribute to a deeper understanding of the models' capabilities and aid in making data-driven decisions for sales forecasting and strategic planning.

## **Statement 2:**

For Market value prediction model, initially, four machine learning models, namely Random Forest Regression, Gradient Boosting, Simple Vector Machines, and Neural Network Models, were built. The Random Forest Model (RFM) (*n\_estimators = 750, min\_samples\_split=10*) and Gradient Boosting (GBR) models (*n\_estimators = 500, min\_samples\_split=10*) were determined as the top-performing models, based on the evaluation metric R-squared which shows how well the data fit the regression model. The RFR and GBR models exhited high R-squared values of 0.86 and 0.85 respectively. In addition, with hyperparameter tuning, the three-hidden layer *(50, 50, 20)* Neural Network model performed well with an R square value of 0.81 which also makes it one of the reliable predictors for market values.

|  |  |  |  |
| --- | --- | --- | --- |
| *Table 9: Results for Market Value Dataset* | | | |
| *Evaluation Metrics: R Squared* | | | |
| *Model* | ***With All Features*** | ***Feature Importance*** | ***PCA*** |
| *Random Forest Regressor* | **0.86** | **0.85** | 0.80 |
| *Gradient Boosting Regressor* | **0.85** | **0.83** | 0.75 |
| *Neural Network* | **0.81** |  | 0.78 |

Furthermore, with feature importance *(Table 9)*, the RFR and GBR were built for selected features resulting in similar R squared value, which is 0.85 and 0.83, respectively. By identifying and selecting the most prominent features, the feature selection process will provide valuable insights into the most crucial features for predicting the market value for the houses built in the last ten years.

Subsequently, Principal Component Analysis (PCA) was applied to reduce the dimensionality of the data. This led to a decrease in the R-squared values for both the RFM and GBR models to 0.80 and 0.75, respectively. Interestingly, the Neural Network model surpassed the other toe models after PCA with an R-squared value of 0.78, indicating its enhanced performance in this context.

|  |  |  |  |
| --- | --- | --- | --- |
| *Table 10: Clustering Results: Market Value Dataset* | | | |
| *Evaluation Metrics: R Squared* | | | |
| *Model* | ***Cluster 1*** | ***Cluster 2*** | ***Weighted Average*** |
| *Random Forest* | 0.90 | 0.82 | **0.87** |
| *Gradient Boosting* | 0.87 | 0.79 | **0.84** |
| *Neural Network* | 0.81 | 0.51 | **0.71** |

For clustering, we have assigned weights to the clusters and added their weighted evaluations to further decide if clusters helped us in improving the performance of the model or not. From the weighted average evaluations *(Table 10)*, it can be observed that RFR and GBR performed well on the clustered dataset. However, for clustered dataset Neural Network didn’t perform well compared to RFR and GBR. However, feature importance *(Table 11)* resulted in lowered R squared for RFR i.e., 0.68, while GBR exhibited strong predictive capability with an R squared value of 0.86. Additionally, after PCA, the decrease in R squared values were observed as 0.82 for RFR, 0.79 for GBR. After PCA, Neural Network achieved the highest R squared value of 0.79.

|  |  |  |  |
| --- | --- | --- | --- |
| *Table 11: Clustering Results (Feature Importance/PCA): Market Value* | | | |
| *Feature Importance: Evaluation Metric: R Squared* | | | |
| *Model* | ***Cluster 1*** | ***Cluster 2*** | ***Weighted Average*** |
| *Random Forest Regressor* | 0.70 | 0.67 | **0.68** |
| *Gradient Boosting Regressor* | 0.87 | 0.86 | **0.86** |
| *Principle Component Analysis: Evaluation Metric: R Squared* | | | |
| *Random Forest Regressor* | 0.87 | 0.72 | **0.82** |
| *Gradient Boosting Regressor* | 0.82 | 0.73 | **0.79** |
| *Neural Network* | 0.84 | 0.68 | **0.79** |

In conclusion, we have observed that our Total Market Value prediction models in 2023, such as RFM and GBR, are performing well, demonstrating their reliability in accurately estimating the market value of properties. It can be assumed that the model is equally effective in predicting the Total Market Value for both clustered and non-clustered datasets with all the features. The reliable predictive capabilities of these models provide valuable insights for stakeholders, empowering data-driven decision-making in the real estate market.

# **Discussion**

In sales dataset, certain models stood out in performance, displaying their ability to fit the data. Notably, both the Random Forest Model (RFM) and Gradient Boosting (GBR) displayed the best fitting to the data. Additionally, the Neural Network model demonstrated commendable predictive capabilities. Through feature selection, the relevance of specific attributes in shaping sales prices was emphasized, encompassing periods both prior to and after the Covid-19 pandemic.

The use of Principal Component Analysis (PCA) for data dimensionality reduction resulted in marginal adjustments in model fitting for select models. While some information was inevitably lost, the models maintained significant predictive strength. Furthermore, the influence of PCA on the 2022 dataset became evident, particularly when considering the weighted average evaluation. This insight into post-dimensionality reduction performance contributed to a deeper understanding of the models' behavior.

Predicting total market value yielded valuable insights whereby several models consistently demonstrated good fitting to the data, indicating their capability to capture underlying trends (Jingyi Mu, Wu, & Zhang, 2014). Feature selection highlighted the contributions of specific attributes, notably those associated with properties constructed within the past decade.

The impact of PCA on dimensionality reduction was evident. Despite modest adjustments in model fitting for certain instances, the Neural Network model showed its adaptability after PCA, harnessing the reduced feature space. Additionally, the analysis of clustered data further unveiled patterns, affirming the models' capacity to predict total market value across various dataset characteristics.

***Hyperparameter Tuning***played a crucial role in refining model performance. It involved systematically adjusting parameters for the Random Forest Model (RFM), Gradient Boosting (GBR), and Neural Network models. This process optimized their ability to fit the data and enhance predictive accuracy. Successful hyperparameter tuning ensured consistent model fitting and improved reliability in capturing property value dynamics across the different periods.

In sum, this research underscores the dependability of specific models in predicting property’s marketvalues, bridging the gap between pre- and post-Covid-19 periods (Grybauskas, Pilinkienė, & Stundžienė, 2021). Their consistent fitting to the data provides valuable insights, guiding informed decision-making within the evolving real estate landscape. Understanding how preprocessing techniques, such as PCA and clustering, impact model performance empowers businesses to make well-informed choices, enhancing their overall efficacy and adaptability within the real estate industry. Lastly, the models' ability to consistently capture property value changes across diverse times reinforces their significance within the evolving post-pandemic real estate context. This significance extends not only to model dependability but also to securing storage and computation capabilities in understanding economic shocks like Covid-19 to an industry with such a large data availability (Roy, Roy, & Mahmood, 2021).

# **Conclusion**

This study aims at using applied machine learning techniques to identify the significant factors influencing residential property sales prices before and after Covid, as well as predicting the market value of residential houses in 2023. Discoveries and new insights were made at each phase, observing that both Random Forest and Gradient Boosting Regressor outperformed other models. The feature importance helps us understand significant factors for both post and pre Covid, and how different they are.

Conclusively, several factors that influence residential property Sale Prices remained consistent between 2019 and 2022, hence their enduring importance in the real estate market. The factors include land size, quality, porch and attached garage square footage, built-as square footage, physical age, zip code, and average school score. However, the post-Covid period brought about changes in buyer preferences leading to the inclusion of new attributes. Forinstance, the significance of energy-efficient HVAC systems like heat pumps highlighted preference for environmentally friendly energy options. The number of bedrooms gained importance, reflecting the need for home offices, or changing family requirements.

The emergence of fireplaces as an influential attribute suggested increased emphasis on home comfort, highlighting evolving buyer priorities post-Covid. Additionally, it was observed that proximity to schools with higher average school scores impacted residential property prices, reflecting families' preferences for quality education. Understanding these similarities and differences is essential for real estate professionals and stakeholders in navigating the evolving market and making informed decisions during an economic shock like Covid-19.

Our predictive market value machine learning model has yielded an R-squared value of 0.87. This indicates that approximately 87% of the variability in market values is effectively captured by our model's selected variables, highlighting its strong predictive capabilities and potential for making accurate market value forecasts. In addition, from feature importance revealed the most important feature as the *Quality* of the property. School ratings are still a prominent feature. The study discovered that regardless of the dataset, the random forest and Gradient Boosting models consistently demonstrated superior accuracy. This finding contradicted previous research which had suggested that Neural Networks outperformed these models [Jian Guan, Alan S. Levitan, Jozef Zurada. (2011)].

An intriguing observation emerged when using clustering as a preprocessing technique during data re-structuring. Notably, the R-squared values for 2019 Sales Data and Market Value prediction datasets remain the same, reflecting the effectiveness of this approach. However, for the 2022 Sales Data, clustering did not yield favorable results. The study also suggests a potential avenue for further exploration: investigating the efficacy of density-based clustering through the DBSCAN algorithm.

Additionally, there is a significant amount of data on the Pierce County open data portal, although we were constrained by time, with more time one could include many additional community variables. Also, our research did not consider demographic information about the homeowner, that information could also be used as features in future studies. The findings in this paper are both interesting and impactful. For researchers, it provides a roadmap for constructing effective prediction algorithms that can be used in future research.

Constraints such as computation power was a major issue since the dataset size required more memory and time, especially when machine learning models were being deployed and developed. It is important to consider a wider scope of variables and a wider temporal resolution of dataset to be able to find more insights and patterns. The more data used to train the models, the better the predictive ability.

# References

1. *Property Information (2023, June 23).* | Pierce County, WA - Official Website. (n.d.). https://www.piercecountywa.gov/736/Data-Downloads
2. Grybauskas, A., Pilinkien Vaida& Stundien Alina. (2021, August 3). *Predictive analytics using big data for the real estate market during the COVID-19 pandemic - journal of big data*. SpringerOpen. <https://journalofbigdata.springeropen.com/articles/10.1186/s40537-021-00476-0>
3. Mora-Garcia, R.-T., Cespedes-Lopez, M.-F., & Perez-Sanchez, V. R. (2022, November 21). *Housing price prediction using machine learning algorithms in COVID-19 times*. MDPI. <https://www.mdpi.com/2073-445X/11/11/2100>
4. Wang, X., Wen, J., Zhang, Y., & Wang, Y. (2014). *Real estate price forecasting based on SVM optimized by PSO*. Optik, 125(3), 1439-1443.
5. Gao, C., Bompard, E., Napoli, R., & Cheng, H. (2007). *Price forecast in the competitive electricity market by support vector machine. Physica A: Statistical Mechanics and its Applications*, 382(1), 98-113.
6. Jingyi Mu, Fang Wu, Aihua Zhang. (2014). *Housing value forecasting based on machine learning methods. In Abstract and Applied Analysis (Vol. 2014)*. Hindawi. <https://www.hindawi.com/journals/aaa/2014/648047/>
7. Jian Guan, Alan S. Levitan, Jozef Zurada. (2011). *A Comparison of Regression and Artificial Intelligence Methods in a Mass Appraisal Context. Journal of Real Estate Research, 33(3)*, 349-387 <https://www.jstor.org/stable/24888380>
8. Ward, J. (2022, August 24). *How school ratings can affect home values and prices. Home Sweet Homes*. <https://www.homes.com/blog/2022/08/how-school-ratings-can-affect-home-values-and-prices/>
9. *School score (2023, July 05)*. SchoolDigger. (n.d.). <https://www.schooldigger.com/go/WA/schools/0870001496/school.aspx>
10. *Address points(2023, July 05)*. Pierce County WA Open GeoSpatial Data Portal (v2.1). (n.d.-a). https://gisdata-piercecowa.opendata.arcgis.com/datasets/address-points/expl

# Appendices

## **Code:**

**Statement 1 Code:**

# Import relevant libraries

import time

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import collections

from sklearn import metrics

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor

from sklearn.ensemble import GradientBoostingRegressor

from sklearn.svm import SVR

from sklearn.neural\_network import MLPRegressor

from sklearn.metrics import r2\_score, make\_scorer

from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler

from sklearn.feature\_selection import SelectFromModel

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette\_score

from sklearn.model\_selection import GridSearchCV

from sklearn.model\_selection import cross\_val\_score

####################################################################

# Utility Functions

####################################################################

def DecisionTreeRegressorModel(X\_test, y\_test, X\_train, y\_train):

t\_start = time.time()

dtr = DecisionTreeRegressor(max\_leaf\_nodes = 500, max\_features='auto', splitter='best', random\_state = 0)

dtr\_mean\_score = np.mean(cross\_val\_score(dtr, X\_train, y\_train, cv=5))

print(f'Mean Score for Decision Tree: {dtr\_mean\_score:.4f}')

dtr.fit(X\_train, y\_train)

y\_pred\_dtr = dtr.predict(X\_test)

mae = metrics.mean\_absolute\_error(y\_test, y\_pred\_dtr)

mse = metrics.mean\_squared\_error(y\_test,y\_pred\_dtr)

rmse = np.sqrt(mse)

print('\*\*Performance Evaluations for Decision Tree\*\*')

print(f'mae: {mae:.4f}')

print(f'R2 score from the Decision Tree model: {r2\_score(y\_test, y\_pred\_dtr):.2f}')

print (f'mse: {mse:.4f}')

print(f'rmse: {rmse:.4f}')

t\_end = time.time()

print(f'Execution time for DTR: {t\_end-t\_start:.2f} seconds')

return dtr

def RandomForestRegressorModel(X\_test, y\_test, X\_train, y\_train):

t\_start = time.time()

rfrm = RandomForestRegressor(random\_state=1234)

rfrm\_mean\_score = np.mean(cross\_val\_score(rfrm, X\_train, y\_train, cv=5))

print(f'Mean Score for RandomForest: {rfrm\_mean\_score:.4f}')

rfrm.fit(X\_train, y\_train)

y\_pred\_rfrm = rfrm.predict(X\_test)

rfrm\_mean\_score = np.mean(cross\_val\_score(rfrm, X\_train, y\_train, cv=5))

print(f'Mean Score for Random Forest: {rfrm\_mean\_score:.4f}')

# Calculate MSE (mean squared errors), RMSE (Root Mean Squred Error),and R^2 for errors.

mse = metrics.mean\_squared\_error(y\_test,y\_pred\_rfrm)

rmse = np.sqrt(mse)

print('\n', '\*\*Performance Evaluation for Random Forest Regressor\*\*')

print (' mse: ', mse,'\n','rmse:', rmse)

print(f'R2 score from the Random Forest model: {r2\_score(y\_test, y\_pred\_rfrm):.2f}')

t\_end = time.time()

print(f'Execution time for RFR: {t\_end-t\_start:.2f} seconds')

return rfrm

def GradientBoostingRegressorModel(X\_test, y\_test, X\_train, y\_train):

t\_start = time.time()

gbr = GradientBoostingRegressor(random\_state = 1234)

gbr\_mean\_score = np.mean(cross\_val\_score(gbr, X\_train, y\_train, cv=5))

print(f'Mean Score for GradientBoosting: {gbr\_mean\_score:.4f}'

gbr.fit(X\_train, y\_train)

y\_pred\_gbr = gbr.predict(X\_test)

mse = metrics.mean\_squared\_error(y\_test,y\_pred\_gbr)

print(print('\*\*Performance Evaluations for Gradient Boosting\*\*'))

print(' mse: ', mse,'\n')

print(f'R2 score from the Gradient Boosting model: {r2\_score(y\_test, y\_pred\_gbr):.2f}')

t\_end = time.time()

print(f'Execution time for GBR: {t\_end-t\_start:.2f} seconds')

return gbr

def SVMRegressorModel(kernel\_Value, C\_Value, epsilon\_value, X\_test, y\_test, X\_train, y\_train):

t\_start = time.time()

svr = SVR(kernel=kernel\_Value, C= C\_Value, epsilon=epsilon\_value)

svr\_mean\_score = np.mean(cross\_val\_score(svr, X\_train, y\_train, cv=5))

print(f'Mean Score for SVM: {svr\_mean\_score:.4f}')

# Train the model

svr.fit(X\_train, y\_train)

# Generate some test data

#X\_test = np.arange(0.0, 5.0, 0.01)[:, np.newaxis]

y\_pred\_svr = svr.predict(X\_test)

mse = metrics.mean\_squared\_error(y\_test,y\_pred\_svr)

print('\*\*Performance Evaluations for SVM Regressor\*\*')

print(' mse: ', mse,'\n')

print(f'R2 score from the SVM model: {r2\_score(y\_test, y\_pred\_svr):.2f}')

t\_end = time.time()

print(f'Execution time for SVM: {t\_end-t\_start:.2f} seconds')

return svr

def NeuralNetworkRegressorModel(X\_test, y\_test, X\_train, y\_train):

nnr = MLPRegressor(random\_state=1234)

nnr\_mean\_score = np.mean(cross\_val\_score(nnr, X\_train, y\_train, cv=5))

print(f'Mean Score for Neural Network: {nnr\_mean\_score:.4f}')

nnr.fit(X\_train,y\_train)

y\_pred\_nn = nnr.predict(X\_test)

mse = metrics.mean\_squared\_error(y\_test,y\_pred\_nn)

print('\*\*Performance Evaluations for Neural Network\*\*')

print(' mse: ', mse,'\n')

print(f'R2 score from the NN model: {r2\_score(y\_test, y\_pred\_nn):.2f}')

return nnr

def NeuralNetworkModel\_1Layer(hidden\_layer\_sizes, activationFunc, X\_test, y\_test, X\_train, y\_train):

t\_start = time.time()

nnr = MLPRegressor(hidden\_layer\_sizes=(hidden\_layer\_sizes),

max\_iter=3000, activation=activationFunc, random\_state=1234)

nnr.fit(X\_train,y\_train)

y\_pred\_nn = nnr.predict(X\_test)

mse = metrics.mean\_squared\_error(y\_test,y\_pred\_nn)

print('\*\*Performance Evaluations for Neural Network\*\*')

print(' mse: ', mse,'\n')

print(f'R2 score from the NN model One Layer: {activationFunc} {hidden\_layer\_sizes} {r2\_score(y\_test, y\_pred\_nn):.2f}')

t\_end = time.time()

print(f'Execution time for NNR\_1L: {t\_end-t\_start:.2f} seconds')

def NeuralNetworkModel\_2Layer(hidden\_layer\_sizes, hidden\_layer\_2, activationFunc, X\_test, y\_test, X\_train, y\_train):

t\_start = time.time()

model = MLPRegressor(hidden\_layer\_sizes=(hidden\_layer\_sizes, hidden\_layer\_2), max\_iter=1000, activation=activationFunc, random\_state=0)

model.fit(X\_train, y\_train)

y\_pred\_nn = model.predict(X\_test)

mse = metrics.mean\_squared\_error(y\_test,y\_pred\_nn)

print('\*\*Performance Evaluations for Neural Network\*\*')

print(' mse: ', mse,'\n')

print(f'R2 score from the NN model Three Layers: {activationFunc} {hidden\_layer\_sizes} {hidden\_layer\_2} {r2\_score(y\_test, y\_pred\_nn):.2f}')

t\_end = time.time()

print(f'Execution time for NNR\_2L: {t\_end-t\_start:.2f} seconds')

def NeuralNetworkModel\_3Layer(hidden\_layer\_sizes, hidden\_layer\_2, hidden\_layer\_3, activationFunc, X\_test, y\_test, X\_train, y\_train):

t\_start = time.time()

model = MLPRegressor(hidden\_layer\_sizes=(hidden\_layer\_sizes, hidden\_layer\_2, hidden\_layer\_3), max\_iter=1000,

activation=activationFunc, random\_state=0)

model.fit(X\_train, y\_train)

y\_pred\_nn = model.predict(X\_test)

mse = metrics.mean\_squared\_error(y\_test,y\_pred\_nn)

print('\*\*Performance Evaluations for Neural Network\*\*')

print(' mse: ', mse,'\n')

print(f'R2 score from the NN model Two Layers: {activationFunc} {hidden\_layer\_sizes} {hidden\_layer\_2} {hidden\_layer\_3} {r2\_score(y\_test, y\_pred\_nn):.2f}')

t\_end = time.time()

print(f'Execution time for NNR\_2L: {t\_end-t\_start:.2f} seconds')

def FindNeuralNetworkParam\_GridSearch(X\_test, y\_test, X\_train, y\_train):

t\_start = time.time()

r2\_scorer = make\_scorer(r2\_score)

param\_grid = {

'hidden\_layer\_sizes': [(40), (50), (100), (150), (50, 40), (100, 50), (150, 100),

(50, 40, 10), (50, 40, 20), (50, 40, 30), (50, 40, 40)],

'activation': ['relu'], # Activation function for hidden layers

'max\_iter': [4000,5000]

}

nnm\_r = MLPRegressor()

grid\_src = GridSearchCV(estimator= nnm\_r, param\_grid = param\_grid, cv=5, scoring=r2\_scorer)

grid\_src.fit(X\_train,y\_train)

t\_end = time.time()

print('\n\n \*\*Report\*\*')

print("Best Parameters: ", grid\_src.best\_params\_)

print("Best Mean Test R-squared Score: ", grid\_src.best\_score\_)

print(f'The best estimator: {grid\_src.best\_estimator\_}')

print(f'The best parameters:\n {grid\_src.best\_params\_}')

print(f'The best score: {grid\_src.best\_score\_:.4f}')

print(f'Total run time for GridSearchCV: {(t\_end-t\_start):.2f} seconds')

# Check the details of search

return pd.DataFrame(grid\_src.cv\_results\_)

def RandomForestRegressorModel\_WithParams(nEstimator, minSamplesSplit, X\_test, y\_test, X\_train, y\_train):

t\_start = time.time()

rfrm = RandomForestRegressor(n\_estimators=nEstimator, min\_samples\_split=minSamplesSplit, random\_state=1234)

rfrm\_mean\_score = np.mean(cross\_val\_score(rfrm, X\_train, y\_train, cv=5))

print(f'Mean Score for RandomForest: {rfrm\_mean\_score:.4f}')

rfrm.fit(X\_train, y\_train)

y\_pred\_rfrm = rfrm.predict(X\_test)

# Calculate MSE (mean squared errors), RMSE (Root Mean Squred Error),and R^2 for errors.

mse = metrics.mean\_squared\_error(y\_test,y\_pred\_rfrm)

mape = np.mean(np.abs((y\_test - y\_pred\_rfrm)/y\_test))\*100

rmse = np.sqrt(mse)

print('\n', '\*\*Performance Evaluation for Random Forest Regressor\*\*')

print (' mse: ', mse,'\n','rmse:', rmse)

print (f'mape: {mape:.4f}')

print(f'R2 score from the Random Forest model: {r2\_score(y\_test, y\_pred\_rfrm):.2f}')

t\_end = time.time()

print(f'Execution time for RFR: {t\_end-t\_start:.2f} seconds')

return rfrm

def FindRandomForestRegressorParam\_GridSearch(X\_test, y\_test, X\_train, y\_train):

t\_start = time.time()

r2\_scorer = make\_scorer(r2\_score)

param\_grid = {

'n\_estimator':[500, 750, 1000, 1500],

'min\_samples\_split': [5, 10, 15, 20],

}

rfrm = RandomForestRegressor(random\_state = 1234)

grid\_src = GridSearchCV(estimator= rfrm, param\_grid = param\_grid, cv=5, scoring=r2\_scorer)

grid\_src.fit(X\_train,y\_train)

t\_end = time.time()

print('\n\n \*\*Report\*\*')

print("Best Parameters: ", grid\_src.best\_params\_)

print("Best Mean Test R-squared Score: ", grid\_src.best\_score\_)

print(f'The best estimator: {grid\_src.best\_estimator\_}')

print(f'The best parameters:\n {grid\_src.best\_params\_}')

print(f'The best score: {grid\_src.best\_score\_:.4f}')

print(f'Total run time for GridSearchCV: {(t\_end-t\_start):.2f} seconds')

# Check the details of search

return pd.DataFrame(grid\_src.cv\_results\_)

def GradientBoostingRegressorModel\_WithParams(nEstimator, minSamplesSplit, X\_test, y\_test, X\_train, y\_train):

t\_start = time.time()

gbr = GradientBoostingRegressor(n\_estimators=nEstimator, min\_samples\_split=minSamplesSplit)

gbr\_mean\_score = np.mean(cross\_val\_score(gbr, X\_train, y\_train, cv=5))

print(f'Mean Score for GradientBoostingWithParams: {gbr\_mean\_score:.4f}')

gbr.fit(X\_train, y\_train)

y\_pred\_gbr = gbr.predict(X\_test)

mse = metrics.mean\_squared\_error(y\_test,y\_pred\_gbr)

mape = np.mean(np.abs((y\_test - y\_pred\_gbr)/y\_test))\*100

print(print('\*\*Performance Evaluations for Gradient Boosting\*\*'))

print(' mse: ', mse,'\n')

print (f'mape: {mape:.4f}')

print(f'R2 score from the Gradient Boosting model WithParams: {r2\_score(y\_test, y\_pred\_gbr):.2f}')

t\_end = time.time()

print(f'Execution time for GBR: {t\_end-t\_start:.2f} seconds')

return gbr

def FindGradientBoostingRegressorParam\_GridSearch(X\_test, y\_test, X\_train, y\_train):

t\_start = time.time()

r2\_scorer = make\_scorer(r2\_score)

param\_grid = {

'n\_estimator':[500, 750, 1000, 1500],

'min\_samples\_split': [5, 10, 15, 20],

}

gbr = GradientBoostingRegressor(random\_state = 1234)

grid\_src = GridSearchCV(estimator= gbr, param\_grid = param\_grid, cv=5, scoring=r2\_scorer)

grid\_src.fit(X\_train,y\_train)

t\_end = time.time()

print('\n\n \*\*Report\*\*')

print("Best Parameters: ", grid\_src.best\_params\_)

print("Best Mean Test R-squared Score: ", grid\_src.best\_score\_)

print(f'The best estimator: {grid\_src.best\_estimator\_}')

print(f'The best parameters:\n {grid\_src.best\_params\_}')

print(f'The best score: {grid\_src.best\_score\_:.4f}')

print(f'Total run time for GridSearchCV: {(t\_end-t\_start):.2f} seconds')

# Check the details of search

return pd.DataFrame(grid\_src.cv\_results\_)

def AnalyzingDatasets(df):

## show the first five rows of the Appraisal dataset

print("First 5 rows of the dataset:")

print(df.head())

# Check the data types of each column

print("Data types:")

print(df.dtypes)

# Check the number of missing values in each column

print("Missing values:")

print(df.isnull().sum())

df.info()

# Get the summary statistics of numeric variables (appraisal dataset)

print("Summary statistics:")

print(df.describe())

# Distinct Count & frequency of nominal variables

df.nunique()

for colName in df.columns:

print("UniqueValues for " + colName)

print(df[colName].value\_counts())

#null values present in the Appraisal dataset

df.isnull().sum().sort\_values(ascending=False)

def DetermineCorrelatedVariables(df):

df\_numeric = df.select\_dtypes(include=np.number)

corrMatrix = df\_numeric.corr()

threshold = 0.7

highly\_correlated\_variables = []

# Iterate through each column in the correlation matrix

for i in range(len(df.columns)):

for j in range(i + 1, len(corrMatrix.columns)):

if abs(corrMatrix.iloc[i, j]) >= threshold:

# Append the names of highly correlated variables to the list

variable\_i = corrMatrix.columns[i]

variable\_j = corrMatrix.columns[j]

highly\_correlated\_variables.append((variable\_i, variable\_j))

return highly\_correlated\_variables

def CalculateAverageSchoolScore(dfMergedRow, schoolData):

schoolRadius = 2\*5280 # schools in 2 mile (5280 feet) radius

dfMergedX = dfMergedRow['X']

dfMergedY = dfMergedRow['Y']

zipCode = dfMergedRow['ZipCode']

filteredSchoolData = schoolData[(schoolData['X\_COORD'] <= dfMergedX + schoolRadius)

& (schoolData['X\_COORD'] >= dfMergedX - schoolRadius)

& (schoolData['Y\_COORD'] <= dfMergedY + schoolRadius)

& (schoolData['Y\_COORD'] >= dfMergedY - schoolRadius)

& (schoolData['ZIP'] == zipCode)]

if len(filteredSchoolData) == 0:

return 0

else:

return filteredSchoolData["score"].mean()

####################################################################

# 1. Reading the Datasets

####################################################################

input\_file\_appraisal = 'appraisal\_account.txt'

column\_names\_appraisal = ['Parcel Number', 'Appraisal Account Type', 'Business Name', 'Value Area ID',

'Land Economic Area', 'Buildings', 'Group Account Number',

'Land Gross Acres', 'Land Net Acres', 'Land Gross Square Feet',

'Land Net Square Feet', 'Land Gross Front Feet', 'Land Width', 'Land Depth',

'Submerged Area Square Feet', 'Appraisal Date', 'Waterfront Type',

'View Quality', 'Utility Electric', 'Utility Sewer', 'Utility Water',

'Street Type', 'Latitude', 'Longitude']

input\_file\_improvement = 'improvement.txt'

column\_names\_improvement = ['Parcel Number', 'Building ID', 'Property Type', 'Neighborhood',

'Neighborhood Extension', 'Square Feet', 'Net Square Feet',

'Percent Complete', 'Condition', 'Quality', 'Primary Occupancy Code',

'Primary Occupancy Description', 'Mobile Home Serial Number', 'Mobile Home Total Length',

'Mobile Home Make', 'Attic Finished Square Feet', 'Basement Square Feet',

'Basement Finished Square Feet', 'Carport Square Feet', 'Balcony Square Feet',

'Porch Square Feet', 'Attached Garage Square Feet', 'Detached Garage Square Feet',

'Fireplaces', 'Basement Garage Door']

input\_file\_improvementBuiltAs = 'improvement\_builtas.txt'

column\_names\_improvementBuiltAs = ['Parcel Number', 'Building ID', 'Built-As Number', 'Built-As ID',

'Built-As Description', 'Built-As Square Feet', 'HVAC', 'HVAC Description',

'Exterior', 'Interior', 'Stories', 'Story Height', 'Sprinkler Square Feet',

'Roof Cover', 'Bedrooms', 'Bathrooms', 'Units', 'Class Code', 'Class Description',

'Year Built', 'Year Remodeled', 'Adjusted Year Built', 'Physical Age',

'Built-As Length', 'Built-As Width', 'Mobile Home Mode']

input\_file\_sale = 'sale.txt'

column\_names\_sale = ['ETN', 'Parcel Count', 'Parcel Number', 'Sale Date',

'Sale Price', 'Deed Type', 'Grantor', 'Grantee',

'Valid/Invalid', 'Confirmed/Uncomfirmed', 'Exclude Reason',

'Improved/Vacant', 'Appraisal Account Type']

# Set the delimiter used in the input file

delimiter = '|' # Example: Tab-separated values ('\t'), Comma-separated values (','), etc.

# Read the delimited text file into a pandas DataFrame

df\_appraisal = pd.read\_csv(input\_file\_appraisal, delimiter=delimiter, encoding='ISO-8859-1', names=column\_names\_appraisal)

df\_improvement = pd.read\_csv(input\_file\_improvement, delimiter=delimiter, encoding='ISO-8859-1', names=column\_names\_improvement)

df\_improvementBuiltAs = pd.read\_csv(input\_file\_improvementBuiltAs, delimiter=delimiter, encoding='ISO-8859-1', names=column\_names\_improvementBuiltAs)

df\_sale = pd.read\_csv(input\_file\_sale, delimiter=delimiter, encoding='ISO-8859-1', names=column\_names\_sale)

# Read external Dataset files

df\_Address\_Points = pd.read\_csv('Address\_Points.csv')

df\_school\_Data= pd.read\_csv("school\_by\_zipcode.csv")

####################################################################

# 2. Datasets Preprocessing

####################################################################

# Analyzing Datasets

print("Analyzing Appraisal Data Set")

AnalyzingDatasets(df\_appraisal)

print("Analyzing Improvement Data Set")

AnalyzingDatasets(df\_improvement)

print("Analyzing Improvement BuiltAs Data Set")

AnalyzingDatasets(df\_improvementBuiltAs)

print("Analyzing Sales Data Set")

AnalyzingDatasets(df\_sale)

print("Analyzing AddressPoints Data Set")

AnalyzingDatasets(df\_Address\_Points)

print("Analyzing School Data Set")

AnalyzingDatasets(df\_school\_Data)

####################################################################

# 2.1 Appraisal Dataset Preprocessing

####################################################################

# Filtering the dataset to only include Residential Properties

appraisal\_dfFiltered = df\_appraisal[(df\_appraisal['Appraisal Account Type'] == 'Residential')]

appraisal\_dfFiltered.info()

appraisal\_dfFiltered.isnull().sum().sort\_values(ascending=False)

appraisal\_dfFiltered.dtypes

# Dropping the non-required columns which have more than 30% nulls, and not informative

appraisal\_dfFiltered.drop(['Appraisal Account Type', 'Value Area ID', 'Business Name', 'Submerged Area Square Feet',

'Group Account Number', 'Land Economic Area', 'Appraisal Date'] , axis=1, inplace=True)

appraisal\_dfFiltered['Waterfront Type'] = appraisal\_dfFiltered['Waterfront Type'].apply(lambda x: 1 if not pd.isnull(x) else 0)

appraisal\_dfFiltered['View Quality'] = appraisal\_dfFiltered['View Quality'].fillna('N/A')

viewQualityOrd = {'View Quality': {'N/A':0, 'View Lim -':1,'View Lim':2,'View Lim +':3,

'View Good' :4,'View Good +':5, 'View Avg':6,'View Avg +':7,

'View V-Good':8,'View V-Good +':9}}

appraisal\_dfFiltered = appraisal\_dfFiltered.replace(viewQualityOrd)

# Converting Parcel number to String

appraisal\_dfFiltered['Parcel Number']=(appraisal\_dfFiltered['Parcel Number']).apply(str)

# Finding the correlated variables

numericCorrelatedVariables = DetermineCorrelatedVariables(appraisal\_dfFiltered)

print(numericCorrelatedVariables)

# Dropping additional variables based on correlation analysis

appraisal\_dfFiltered.drop(['Land Depth', 'Land Gross Acres',

'Land Gross Square Feet', 'Land Width', 'Land Net Acres'] , axis=1, inplace=True)

#Covert nominal variables into dummy

df\_appraisalFinal = pd.get\_dummies(appraisal\_dfFiltered, columns=['Utility Electric',

'Utility Sewer', 'Utility Water', 'Street Type'], drop\_first=True)

df\_appraisalFinal.info()

# Finding if there are duplicate parcel numbers in appraisal data set

print([item for item, count in collections.Counter(df\_appraisalFinal['Parcel Number']).items() if count > 1])

####################################################################

# 2.2 Improvement Dataset Preprocessing

####################################################################

# Filtering datset to consider only residential properties

df\_improvement = df\_improvement[(df\_improvement['Property Type'] == 'Residential')]

# Dropping the non-required columns which have majority of nulls, not informative

df\_improvement.drop(['Mobile Home Serial Number', 'Mobile Home Total Length',

'Mobile Home Make', 'Basement Garage Door', 'Neighborhood', 'Neighborhood Extension', 'Primary Occupancy Code',

'Primary Occupancy Description', 'Property Type' ] , axis=1, inplace=True)

# Populating null values with 0

df\_improvement['Attic Finished Square Feet'] = df\_improvement['Attic Finished Square Feet'].fillna(0)

df\_improvement['Basement Square Feet'] = df\_improvement['Basement Square Feet'].fillna(0)

df\_improvement['Carport Square Feet'] = df\_improvement['Carport Square Feet'].fillna(0)

df\_improvement['Balcony Square Feet'] = df\_improvement['Balcony Square Feet'].fillna(0)

df\_improvement['Porch Square Feet'] = df\_improvement['Porch Square Feet'].fillna(0)

df\_improvement['Attached Garage Square Feet'] = df\_improvement['Attached Garage Square Feet'].fillna(0)

df\_improvement['Detached Garage Square Feet'] = df\_improvement['Detached Garage Square Feet'].fillna(0)

df\_improvement['Fireplaces'] = df\_improvement['Fireplaces'].fillna(0)

df\_improvement.isnull().sum().sort\_values(ascending=False)

# Removing rows with null values

df\_improvement = df\_improvement[~df\_improvement['Condition'].isnull()]

# Check the data types of each column

print("Data types:")

print(df\_improvement.dtypes)

# Converting Parcel number from Integer to String

df\_improvement['Parcel Number']=(df\_improvement['Parcel Number']).apply(str)

# Finding Correlated Variables

numericCorrelatedVariables = DetermineCorrelatedVariables(df\_improvement)

print(numericCorrelatedVariables)

# Dropping the correlated columns for analysis

df\_improvement.drop(['Square Feet','Basement Finished Square Feet',

'Net Square Feet'] , axis=1, inplace=True)

# Converting Ordinal Variables Condition and Quality to Numeric Variables

conditionOrd = {'Condition': {'Uninhabitable':0, 'Extra Poor':1,'Very Poor':2,'Poor':3,

'low':4,'Fair':5, 'Average':6, 'Avg' : 6, 'Avg.':6,'Good':7, 'Excellent':8}}

qualityOrd = {'Quality': {'Low':0, 'Low Plus':1,'Fair':2,'Fair Plus':3,

'Average' :4,'Average Plus':5, 'Good':6,'Good Plus':7,

'Very Good':8,'Very Good Plus':9,'Excellent':10}}

df\_improvement = df\_improvement.replace(conditionOrd)

df\_improvement = df\_improvement.replace(qualityOrd)

# Finding if there are duplicate parcel numbers in improvement data set

print([item for item, count in collections.Counter(df\_improvement['Parcel Number']).items() if count > 1])

####################################################################

# 2.3 Sales Dataset Preprocessing

####################################################################

# Filter the Dataset where Appraisal Account Type is Residential

df\_sale\_Filtered = df\_sale[(df\_sale['Appraisal Account Type'] == 'Residential')]

# Converting Sale Date from String to Date Time Object and Parcel Number as string

df\_sale\_Filtered['Sale Date'] = pd.to\_datetime(df\_sale\_Filtered['Sale Date'])

df\_sale\_Filtered['Parcel Number']=(df\_sale\_Filtered['Parcel Number']).apply(str)

# Filtering dataset to consider only sales for 2019 and 2022

df\_sale\_Filtered = df\_sale\_Filtered[(df\_sale\_Filtered['Sale Date'].dt.year == 2019) |

(df\_sale\_Filtered['Sale Date'].dt.year == 2022)]

df\_sale\_Filtered.isnull().sum().sort\_values(ascending=False)

# Check the data types of each column

print("Data types:")

print(df\_sale\_Filtered.dtypes)

# Dropping non required columns

df\_sale\_Filtered.drop(['ETN', 'Exclude Reason', 'Grantor', 'Grantee', 'Deed Type', 'Valid/Invalid',

'Appraisal Account Type', 'Confirmed/Uncomfirmed', 'Improved/Vacant'] , axis=1, inplace=True)

# Finding if there are duplicate parcel numbers in improvement data set

print([item for item, count in collections.Counter(df\_sale\_Filtered['Parcel Number']).items() if count > 1])

# For the duplicate sales records, choosing the latest sales record by sorting the dataset based on sale date and then

# selecting the last record

df\_sale\_Filtered = df\_sale\_Filtered.sort\_values(by=['Sale Date'], ascending=True)

df\_sale\_Filtered = df\_sale\_Filtered.drop\_duplicates(subset=['Parcel Number'], keep='last')

# Finding corrrelated variables

numericCorrelatedVariables = DetermineCorrelatedVariables(df\_sale)

print(numericCorrelatedVariables)

####################################################################

# 2.4 ImprovementBuiltAs Dataset Preprocessing

####################################################################

df\_improvementBuiltAs.info()

df\_improvementBuiltAs.isnull().sum().sort\_values(ascending=False)

# Converting Parcel number from integer to string

df\_improvementBuiltAs['Parcel Number']=(df\_improvementBuiltAs['Parcel Number']).apply(str)

# Populating Null values for Exterior with Not Applicable

df\_improvementBuiltAs['Exterior'] = df\_improvementBuiltAs['Exterior'].fillna('Not Applicable')

# Populating Null values for Interior with Not Applicable

df\_improvementBuiltAs['Interior'] = df\_improvementBuiltAs['Interior'].fillna('Not Applicable')

# Populating Null values for Roof Cover with Not Applicable

df\_improvementBuiltAs['Roof Cover'] = df\_improvementBuiltAs['Roof Cover'].fillna('Not Applicable')

df\_improvementBuiltAs.info()

df\_improvementBuiltAs.isnull().sum().sort\_values(ascending=False)

df\_improvementBuiltAs.dtypes

# Dropping coulmns withmajority of nulls, non-informative

df\_improvementBuiltAs.drop(['Built-As Number', 'Built-As ID','HVAC',

'Mobile Home Mode','Class Description', 'Class Code', 'Year Built',

'Adjusted Year Built'] , axis=1, inplace=True)

df\_improvementBuiltAs = df\_improvementBuiltAs[~df\_improvementBuiltAs['Physical Age'].isnull()]

df\_improvementBuiltAs = df\_improvementBuiltAs[~df\_improvementBuiltAs['Story Height'].isnull()]

# Dropping columns not required for analysis

df\_improvementBuiltAs.drop(['Exterior', 'Interior','Roof Cover', ] , axis=1, inplace=True)

# Finding corrrelated variables

numericCorrelatedVariables = DetermineCorrelatedVariables(df\_improvementBuiltAs)

print(numericCorrelatedVariables)

# Removing highly correlated variables

df\_improvementBuiltAs.drop(['Sprinkler Square Feet', 'Built-As Length', 'Built-As Width'] , axis=1, inplace=True)

#Covert nominal variables into dummy

df\_improvementBuiltAsFinal = pd.get\_dummies(df\_improvementBuiltAs, columns=['HVAC Description'],

drop\_first=True)

# Finding if there are duplicate parcel numbers in improvement data set

print([item for item, count in collections.Counter(df\_improvementBuiltAsFinal['Parcel Number']).items() if count > 1])

####################################################################

# 2.4 Address Points Dataset Preprocessing

####################################################################

df\_Address\_Points.info()

# Dropping Non required columns

df\_Address\_Points.drop(['OBJECTID', 'Address', 'Mail\_Stop', 'City', 'State', 'Last\_Edited',

'Status', 'HouseNumber', 'PrefixDirectional', 'StreetName', 'StreetType',

'PostDirectional', 'Jurisdiction', 'AddressID'] , axis=1, inplace=True)

df\_Address\_Points.isnull().sum().sort\_values(ascending=False)

df\_Address\_Points = df\_Address\_Points[~df\_Address\_Points['TaxParcelNumber'].isnull()]

print([item for item, count in collections.Counter(df\_Address\_Points['TaxParcelNumber']).items() if count > 1])

# Removing rows having duplicate Parcel number

df\_Address\_Points = df\_Address\_Points.drop\_duplicates(subset=['TaxParcelNumber'], keep='last')

####################################################################

# 2.5 School Dataset Preprocessing

####################################################################

df\_school\_Data.info()

# Dropping Non required columns

df\_school\_Data.drop(['X', 'Y', 'OBJECTID', 'NAME', 'ADDRESS', 'CITY', 'DISTRICT',

'DIST\_NO', 'TYPE', 'PHONE', 'WEBSITE', 'PRS\_ID', 'GRADE'] , axis=1, inplace=True)

df\_school\_Data.isnull().sum().sort\_values(ascending=False)

####################################################################

# 3 Merging various datasets

####################################################################

df\_appraisalFinal.info()

df\_improvement.info()

df\_sale\_Filtered.info()

df\_improvementBuiltAs.info()

# Merging sales and appraisal datasets

# Since there were no duplicate rows having same Parcel Number, merging the datasets using innner join to avoid any rows with null values

df\_merged = pd.merge(df\_appraisalFinal, df\_sale\_Filtered, left\_on='Parcel Number', right\_on='Parcel Number', how='inner')

df\_merged.info()

# Merging improvement and improvement BuiltAs dataset

# Merging datasets using left join to consider all rows of improvement table

df\_improvement\_merged = pd.merge(df\_improvement, df\_improvementBuiltAsFinal, left\_on=['Parcel Number', 'Building ID'],

right\_on=['Parcel Number', 'Building ID'], how='left')

# Dropping Building ID after merging as it is not required for further analysis

df\_improvement\_merged.drop(['Building ID'] , axis=1, inplace=True)

# Finding if there are duplicate parcel numbers in improvement merged data sets, for duplicate values

# select the property row which got remodelled in the last

df\_improvement\_merged= df\_improvement\_merged.sort\_values(by=['Year Remodeled'], ascending=True)

print([item for item, count in collections.Counter(df\_improvement\_merged['Parcel Number']).items() if count > 1])

df\_improvement\_merged = df\_improvement\_merged.drop\_duplicates(subset=['Parcel Number'], keep= 'last')

# Merging the improvement merged data set with the merged dataset

df\_merged = pd.merge(df\_merged, df\_improvement\_merged, left\_on='Parcel Number',

right\_on='Parcel Number', how='inner')

df\_merged.info()

# Finding if there are duplicate parcel numbers in merged dataset

print([item for item, count in collections.Counter(df\_merged['Parcel Number']).items() if count > 1])

df\_merged.isnull().sum().sort\_values(ascending=False)

# Finding the correlation matric for the merged dataset for highly correlated variables

numericCorrelatedVariables = DetermineCorrelatedVariables(df\_merged)

print(numericCorrelatedVariables)

# Dropping columns from merged datasets which are not relevant for analysis

df\_merged.drop(['Built-As Description', 'Buildings', 'Units', 'Parcel Count' ] , axis=1, inplace=True)

# Merging the merged dataset with address points to get the property coordinates information

# Merged the datasets using inner join to consider only properties which are present in both

df\_merged = pd.merge(df\_merged, df\_Address\_Points, left\_on='Parcel Number', right\_on='TaxParcelNumber', how='inner')

# Merging with merged dataset with school dataset

# For each property calculate the average school score of the all the schools which are within 2 mile property radius

# and are present in same zipcode

df\_merged['Average School Score'] = df\_merged.apply(lambda row: CalculateAverageSchoolScore(row,

df\_school\_Data), axis = 1)

df\_merged.isnull().sum().sort\_values(ascending=False)

# Dropping Non-Required Columns from the merged datasets

df\_merged.drop(['Utility Electric\_POWER INSTALLED', 'Utility Electric\_POWER NO - COMMENT', 'Utility Sewer\_SEWER/SEPTIC INSTALLED', 'Utility Sewer\_SEWER/SEPTIC NO', 'Utility Water\_WATER INSTALLED', 'Utility Water\_WATER NO', 'Street Type\_STREET NO ROAD', 'Street Type\_STREET UNPAVED', 'Utility Sewer\_SEWER/SEPTIC NO PERC', 'Parcel Number', 'Longitude', 'Latitude', 'X', 'Y', 'TaxParcelNumber', 'Year Remodeled', 'Bathrooms', 'Story Height', 'Percent Complete'], axis=1, inplace=True)

# Analyze the correlated variables in the merged datasets

numericCorrelatedVariables = DetermineCorrelatedVariables(df\_merged)

print(numericCorrelatedVariables)

df\_merged.info()

####################################################################

# 4 Analyzing Pre-Covid (2019) and Post-Covid (2022) Datasets

####################################################################

# Splitting the datasets into pre and post covid datasets based on sales date

df\_merged2019 = df\_merged[(df\_merged['Sale Date'].dt.year == 2019)]

df\_merged2022 = df\_merged[(df\_merged['Sale Date'].dt.year == 2022)]

# Dropping Non required columns

df\_merged2019.drop(['Sale Date'] , axis=1, inplace=True)

df\_merged2022.drop(['Sale Date'] , axis=1, inplace=True)

df\_merged2019.columns

df\_merged2022.columns

####################################################################

# 4.1 Analyzing Pre-Covid (2019) Datasets

####################################################################

# X and Y variables for 2019 Dataset

X\_2019 = df\_merged2019.loc[:, df\_merged2019.columns != 'Sale Price']

y\_2019 = df\_merged2019[['Sale Price']].values.ravel()

fn\_2019 = X\_2019.columns

#Scaling the variables

scaler = StandardScaler()

scaler.fit(X\_2019)

X\_s\_2019 = scaler.transform(X\_2019)

# Divide the scaled dataset into training and testing data

X\_train\_s\_2019, X\_test\_s\_2019, y\_train\_2019, y\_test\_2019 = train\_test\_split(X\_s\_2019, y\_2019, test\_size =.30, random\_state=1234)

# Divide the non-scaled dataset into training and testing data.

X\_train\_2019, X\_test\_2019, y\_train\_2019, y\_test\_2019 = train\_test\_split(X\_2019, y\_2019, test\_size =.30, random\_state=1234)

####################################################################

# 4.1.1 Building Models With Default & Hyperparameters Parameters

# Decision Tree Regressor

dtr\_2019 = DecisionTreeRegressorModel(X\_test\_2019, y\_test\_2019, X\_train\_2019, y\_train\_2019)

# Random Forest Regressor

rfr\_2019 = RandomForestRegressorModel(X\_test\_2019, y\_test\_2019, X\_train\_2019, y\_train\_2019)

# Grid Seach for best Random Forest Regressor

FindRandomForestRegressorParam\_GridSearch(X\_test, y\_test, X\_train, y\_train)

# With Hyperparameters for Random Forest

RandomForestRegressorModel\_WithParams(1000, 15, X\_test, y\_test, X\_train, y\_train)

# Gradient Boosting Regression

grbr\_2019 = GradientBoostingRegressorModel(X\_test\_2019, y\_test\_2019, X\_train\_2019, y\_train\_2019)

# Grid Seach for best Gradient Boosting

FindGradientBoostingRegressorParam\_GridSearch(X\_test, y\_test, X\_train, y\_train)

#With Hyperparameters for Gradient Boosting

GradientBoostingRegressorModel\_WithParams(750, 10, X\_test, y\_test, X\_train, y\_train)

# SVM Regression

svmr\_2019 = SVMRegressorModel('linear', 10, 0.01, X\_test\_s\_2019, y\_test\_2019, X\_train\_s\_2019, y\_train\_2019)

# Neural network with default parameters

NeuralNetworkRegressorModel(X\_test\_s\_2019, y\_test\_2019, X\_train\_s\_2019, y\_train\_2019)

# Optimizing the NN Model using Hyper Parameters Tuning Using Grid Search

gridSearch\_2019 = FindNeuralNetworkParam\_GridSearch(X\_test\_s\_2019, y\_test\_2019, X\_train\_s\_2019, y\_train\_2019)

results\_gs = pd.DataFrame(gridSearch\_2019.cv\_results\_)

# Evaluating Neural network results with grid search best estimators values

rSquare = NeuralNetworkModel\_3Layer(50, 40, 20,

'relu', X\_test\_s\_2019, y\_test\_2019, X\_train\_s\_2019, y\_train\_2019)

Output:

Mean Score for Decision Tree: 0.8338

\*\*Performance Evaluations for Decision Tree\*\*

R2 score from the Decision Tree model: 0.87

Mean Score for RandomForest: 0.8670

\*\* Performance Evaluations for Random Forest Regressor\*\*

R2 score from the Random Forest model: 0.85

Mean Score for RandomForestWithParams: 0.8990

R2 score from the Random Forest modelWithParams: 0.89

Mean Score for GradientBoosting: 0.7913

\*\*Performance Evaluations for Gradient Boosting\*\*

R2 score from the Gradient Boosting model: 0.80

Mean Score for GradientBoostingmodelwithParams: 0.8521

R2 score from the GradientBoostingmodelWithParamsl: 0.82

Mean Score for SVM: -0.0321

\*\*Performance Evaluations for SVM Regressor\*\*

R2 score from the SVM model: -0.03

Mean Score for Neural Network: -0.2060

\*\*Performance Evaluations for Neural Network\*\*

R2 score from the NN model: -0.21

The best estimators: MLPClassifier(hidden\_layer\_sizes=(50, 40, 20))

The best parameters for Layers:

{'activation': 'relu', 'hidden\_layer\_sizes': (50, 40, 20)}

The best score for Layers: 0.8043

####################################################################

# 4.1.2 Feature Selection

####################################################################

####################################################################

# 4.1.2.1 Feature Selection Using Random Forest Regressor

####################################################################

# Find the significant features for Pierce County with their importance values.

importances = rfr\_2019.feature\_importances\_

np.sum(importances)

plt.barh(fn\_2019, importances)

# Draw a bar chart to see the sorted importance values with feature names.

df\_importances = pd.DataFrame(data=importances, index=fn\_2019, columns=['importance\_value'])

df\_importances.sort\_values(by = 'importance\_value', ascending=False, inplace=True)

df\_importances

df\_importance\_top15 = df\_importances.head(11)

plt.barh(df\_importance\_top15.index, df\_importance\_top15.importance\_value)

plt.barh(df\_importances.index, df\_importances.importance\_value)

# Build a model with a subset of those features.

selector = SelectFromModel(estimator=RandomForestRegressor(), threshold= 0.015)

X\_reduced = selector.fit\_transform(X\_2019, y\_2019)

selector.threshold\_

selected\_TF = selector.get\_support()

print(f'\n\*\* {selected\_TF.sum()} features are selected.')

# Show those selected features.

selected\_features = []

for i,j in zip(selected\_TF, fn\_2019):

if i: selected\_features.append(j)

print(f'Selected features for Random Forest Regressor are:\n {selected\_features}')

# Build a model using reduced number of features

X\_reduced\_train, X\_reduced\_test, y\_reduced\_train, y\_reduced\_test \

= train\_test\_split(X\_reduced, y\_2019, test\_size =.3, random\_state=1234)

RandomForestRegressorModel(X\_reduced\_test, y\_reduced\_test, X\_reduced\_train, y\_reduced\_train)

# With Hyperparameters for Random Forest

RandomForestRegressorModel\_WithParams(1000, 15, X\_reduced\_test, y\_reduced\_test, X\_reduced\_train, y\_reduced\_train)

Output:

\*\* 8 features are selected.

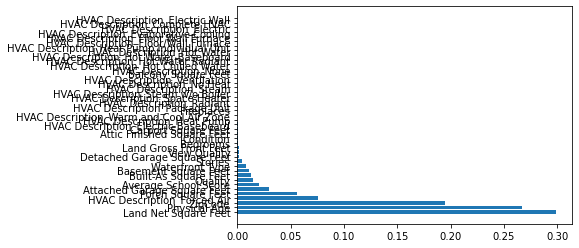
Selected features for Random Forest Regressor are:

['Land Net Square Feet', 'Quality', 'Porch Square Feet', 'Attached Garage Square Feet', 'Built-As Square Feet', 'Physical Age', 'ZipCode', 'Average School Score']

Mean Score for RandomForest: 0.8634

\*\*Performance Metrics for Random Forest Regressor\*\*

R2 score from the Random Forest model: 0.84



Mean Score for RandomForestWithParams: 0.914

R2 score from the Random Forest modelWithParams: 0.90

####################################################################

# 4.1.2.1 Feature Selection Using Gradient Boosting

####################################################################

# Find the significant features for Pierce County with their importance values.

importances = grbr\_2019.feature\_importances\_

np.sum(importances)

plt.barh(fn\_2019, importances)

# Draw a bar chart to see the sorted importance values with feature names.

df\_importances = pd.DataFrame(data=importances, index=fn\_2019, columns=['importance\_value'])

df\_importances.sort\_values(by = 'importance\_value', ascending=False, inplace=True)

df\_importances

plt.barh(df\_importances.index, df\_importances.importance\_value)

# Build a model with a subset of those features.

selector = SelectFromModel(estimator=GradientBoostingRegressor(), threshold=0.015)

X\_reduced = selector.fit\_transform(X\_2019, y\_2019)

selector.threshold\_

selected\_TF = selector.get\_support()

print(f'\n\*\* {selected\_TF.sum()} features are selected.')

# Show those selected features.

selected\_features = []

for i,j in zip(selected\_TF, fn\_2019):

if i: selected\_features.append(j)

print(f'Selected features for GBR are:\n {selected\_features}')

# Build a model using reduced number of features

X\_reduced\_train, X\_reduced\_test, y\_reduced\_train, y\_reduced\_test \

= train\_test\_split(X\_reduced, y\_2019, test\_size =.3, random\_state=1234)

GradientBoostingRegressorModel(X\_reduced\_test, y\_reduced\_test, X\_reduced\_train, y\_reduced\_train)

#With Hyperparameters for Gradient Boosting

GradientBoostingRegressorModel\_WithParams(750, 10, X\_test, y\_test, X\_train, y\_train)

Output:

Selected features for GBR are:

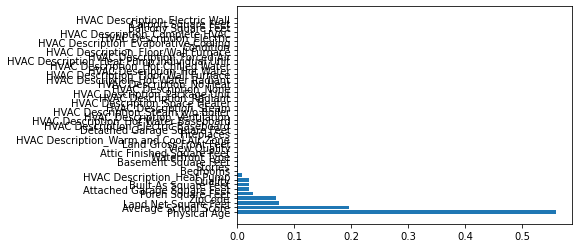
['Land Net Square Feet', 'Quality', 'Porch Square Feet', 'Built-As Square Feet', 'Physical Age', 'ZipCode', 'Average School Score']

Mean Score for GradientBoosting: 0.8275

\*\*Performance Evaluations for Gradient Boosting\*\*

R2 score from the Gradient Boosting model: 0.81

Execution time for GBR: 3.72 seconds



Mean Score for GradientBoostingWithParams: 0.8531

R2 score from the Gradient Boosting modelWithParams 0.89

####################################################################

# 4.1.3 PCA

####################################################################

# Create an instance PCA and build the model using Xn.

# We start from the same number of components as the number of original features.

pca\_prep = PCA().fit(X\_s\_2019)

pca\_prep.n\_components\_

# Consider the variances as the amount of information. Drop components providing less information

# (low variances)

pca\_prep.explained\_variance\_ratio\_

# Create a scree plot and find an "elbow" or an inflection point on the plot.

plt.plot(pca\_prep.explained\_variance\_ratio\_)

plt.xlabel('k number of components')

plt.ylabel('Explained variance')

plt.grid(True)

plt.show()

# From the scree plot, we choose k = 28

n\_pc = 28

pca\_2019 = PCA(n\_components = n\_pc).fit(X\_s\_2019)

Xp\_2019 = pca\_2019.transform(X\_s\_2019)

print(f'After PCA, we use {pca\_2019.n\_components\_} components.\n')

# Split the data into training and testing subsets.

Xp\_train\_2019, Xp\_test\_2019, yp\_train\_2019, yp\_test\_2019 = train\_test\_split(Xp\_2019, y\_2019, test\_size =.3, random\_state = 1234)

# Create random forest models using the transformed data.

rfr\_2019 = RandomForestRegressorModel(Xp\_test\_2019, yp\_test\_2019, Xp\_train\_2019, yp\_train\_2019)

# With Hyperparameters for Random Forest

RandomForestRegressorModel\_WithParams(1000, 15, Xp\_test\_2019, yp\_test\_2019, Xp\_train\_2019, yp\_train\_2019)

# Gradient Boosting models using the transformed data.

gbr\_2019 = GradientBoostingRegressorModel(Xp\_test\_2019, yp\_test\_2019, Xp\_train\_2019, yp\_train\_2019)

#With Hyperparameters for Gradient Boosting

GradientBoostingRegressorModel\_WithParams(750, 10, Xp\_test\_2019, yp\_test\_2019, Xp\_train\_2019, yp\_train\_2019)

# Neural Network models using the transformed data.

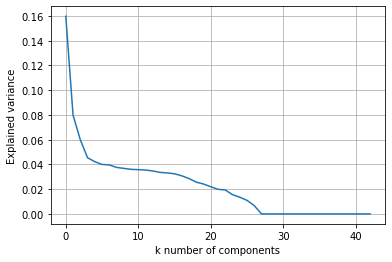
NeuralNetworkRegressorModel(Xp\_test\_2019, yp\_test\_2019, Xp\_train\_2019, yp\_train\_2019)

# Neural Network models with Optimized Parameters Values and transformed data.

rSquare = NeuralNetworkModel\_3Layer(50, 40, 20,

'relu', Xp\_test\_2019, yp\_test\_2019, Xp\_train\_2019, yp\_train\_2019)

Output:



Mean Score for RandomForest: 0.7903

\*\*Performance Evaluation for Random Forest Regressor\*\*

R2 score from the Random Forest model: 0.80

Execution time for RFR: 170.88 seconds

Mean Score for RandomForestWithParams: 0.8249

R2 score from the Random Forest modelWithParams: 0.82

Mean Score for GradientBoosting: 0.7638

\*\*Performance Evaluations for Gradient Boosting\*\*

R2 score from the Gradient Boosting model: 0.79

Mean Score for GradientBoostingWithParams: 0.8009

R2 score from the Gradient Boosting modelWithParams 0.81

Mean Score for SVM: -0.0321

\*\*Performance Evaluations for SVM Regressor\*\*

R2 score from the SVM model: -0.03

Mean Score for Neural Network: -0.2114

\*\*Performance Evaluations for Neural Network\*\*

R2 score from the NN model: -0.22

\*\*Performance Evaluations for Neural Network\*\*

R2 score from the NN model Three Layers: relu 50 40 20 0.72

####################################################################

# 4.1.4 Clustering

####################################################################

n\_pc = 2

pca\_2019 = PCA(n\_components = n\_pc).fit(X\_s\_2019)

Xp\_2019\_2Comp = pca\_2019.transform(X\_s\_2019)

# Create an instance (object) of the KMeans class with the parameters

# initialized (cluster count 2)

km\_2019 = KMeans(n\_clusters=2, random\_state=1234)

# Build a model.

km\_2019 = km\_2019.fit\_predict(Xp\_2019\_2Comp)

silhouette\_avg = silhouette\_score(Xp\_2019\_2Comp, km\_2019)

print('Silhouette Score:', silhouette\_avg)

# Splitting dataset into two clusters

c0\_2019 = df\_merged2019[km\_2019 == 0]

c1\_2019 = df\_merged2019[km\_2019 == 1]

c0\_2019.shape

c1\_2019.shape

Output:

Silhouette Score: 0.45084582614538116

(6823, 44)

(4472, 44)

A yellow and purple dots

Description automatically generated

####################################################################

# 4.1.4.1 Analyzing Cluster 0

####################################################################

plt.scatter(Xp\_2019\_2Comp[:, 0], Xp\_2019\_2Comp[:, 1], c=km\_2019, s=50, cmap='viridis')

plt.show()

X\_C02019 = c0\_2019.loc[:, c0\_2019.columns != 'Sale Price']

y\_C02019 = c0\_2019[['Sale Price']].values.ravel()

fn\_C02019 = X\_C02019.columns

#Scaling the variables

scaler = StandardScaler()

scaler.fit(X\_C02019)

X\_C0s\_2019 = scaler.transform(X\_C02019)

####################################################################

# 4.1.4.1.1 Building Models Wth Default Parameters for Cluster 0

####################################################################

# Divide the scaled dataset into training and testing data

X\_train\_C0s\_2019, X\_test\_C0s\_2019, y\_train\_C02019, y\_test\_C02019 = train\_test\_split(X\_C0s\_2019, y\_C02019, test\_size =.30 ,random\_state=1234)

# Divide the unscaled dataset into training and testing data.

X\_train\_C02019, X\_test\_C02019, y\_train\_C02019, y\_test\_C02019 = train\_test\_split(X\_C02019, y\_C02019, test\_size =.30, random\_state=1234)

# Decision Tree Regressor

dtr\_C02019 = DecisionTreeRegressorModel(X\_test\_C02019, y\_test\_C02019, X\_train\_C02019, y\_train\_C02019)

# Random Forest Regressor

rfr\_C02019 = RandomForestRegressorModel(X\_test\_C02019, y\_test\_C02019, X\_train\_C02019, y\_train\_C02019)

# With Hyperparameters for Random Forest

RandomForestRegressorModel\_WithParams(1000, 15 X\_test\_C02019, y\_test\_C02019, X\_train\_C02019, y\_train\_C02019)

# Gradient Boosting Regression

grbr\_C02019 = GradientBoostingRegressorModel(X\_test\_C02019, y\_test\_C02019, X\_train\_C02019, y\_train\_C02019)

#With Hyperparameters for Gradient Boosting

GradientBoostingRegressorModel\_WithParams(750, 10, X\_test\_C02019, y\_test\_C02019, X\_train\_C02019, y\_train\_C02019))

# SVM Regression

svmr\_C02019 = SVMRegressorModel('linear', 10, 0.01, X\_test\_C0s\_2019, y\_test\_C02019, X\_train\_C0s\_2019, y\_train\_C02019)

# NN Regression

# Neural network with default parameters

NeuralNetworkRegressorModel(X\_test\_C0s\_2019, y\_test\_C02019, X\_train\_C0s\_2019, y\_train\_C02019)

# Optimizing the NN Model using Hyper Parameters Tuning Using Grid Search

gridSearch\_2019 = FindNeuralNetworkParam\_GridSearch(X\_test\_C0s\_2019, y\_test\_C02019, X\_train\_C0s\_2019, y\_train\_C02019)

results\_gs = pd.DataFrame(gridSearch\_2019.cv\_results\_)

# Evaluating Neural network results with grid search best estimator values

rSquare = NeuralNetworkModel\_3Layer(50, 40, 20,

'relu', X\_test\_C0s\_2019, y\_test\_C02019, X\_train\_C0s\_2019, y\_train\_C02019)

Output:

Mean Score for Decision Tree: 0.7991

\*\*Performance Evaluations for Decision Tree\*\*

R2 score from the Decision Tree model: 0.9287

Mean Score for RandomForest: 0.8113

\*\*Evaluation of Errors for Random Forest Regressor\*\*

R2 score from the Random Forest model: 0.82

Mean Score for RandomForestWithParams: 0.8613

R2 score from the Random Forest modelWithParams: 0.87

Mean Score for GradientBoosting: 0.8705

\*\*Performance Evaluations for Gradient Boosting\*\*

R2 score from the Gradient Boosting model: 0.89

Mean Score for GradientBoostingWithParams: 0.9225

R2 score from the Gradient Boosting modelWithParams 0.93

Mean Score for SVM: -0.0753

\*\*Performance Evaluations for SVM Regressor\*\*

R2 score from the SVM model: -0.08

Mean Score for Neural Network: -0.2622

\*\*Performance Evaluations for Neural Network\*\*

R2 score from the NN model: -0.27

R2 score from the NN model Three Layers: relu 50 40 20 0.75

####################################################################

# 4.1.4.1.2 Feature Selection for cluster 0

####################################################################

####################################################################

# 4.1.4.1.2.1 Feature Selection Using Random Forest Regressor

####################################################################

# Find the significant features for Pierce County with their importance values.

importances = rfr\_C02019.feature\_importances\_

np.sum(importances)

plt.barh(fn\_C02019, importances)

# Draw a bar chart to see the sorted importance values with feature names.

df\_importances = pd.DataFrame(data=importances, index=fn\_C02019, columns=['importance\_value'])

df\_importances.sort\_values(by = 'importance\_value', ascending=False, inplace=True)

df\_importances

df\_importance\_top15 = df\_importances.head(11)

plt.barh(df\_importance\_top15.index, df\_importance\_top15.importance\_value)

plt.barh(df\_importances.index, df\_importances.importance\_value)

# Build a model with a subset of those features.

selector = SelectFromModel(estimator=RandomForestRegressor(), threshold= 0.015)

X\_C0\_reduced = selector.fit\_transform(X\_C02019, y\_C02019)

selector.threshold\_

selected\_TF = selector.get\_support()

print(f'\n\*\* {selected\_TF.sum()} features are selected.')

# Show those selected features.

selected\_features = []

for i,j in zip(selected\_TF, fn\_C02019):

if i: selected\_features.append(j)

print(f'Selected features for Random Forest Regressor are:\n {selected\_features}')

# Build a model using reduced number of features

X\_C0\_reduced\_train, X\_C0\_reduced\_test, y\_C0\_reduced\_train, y\_C0\_reduced\_test \

= train\_test\_split(X\_C0\_reduced, y\_C02019, test\_size =.3, random\_state=1234)

RandomForestRegressorModel(X\_C0\_reduced\_test, y\_C0\_reduced\_test, X\_C0\_reduced\_train, y\_C0\_reduced\_train)

# With Hyperparameters for Random Forest

RandomForestRegressorModel\_WithParams(1000, 15, X\_C0\_reduced\_test, y\_C0\_reduced\_test, X\_C0\_reduced\_train, y\_C0\_reduced\_train)

Output:

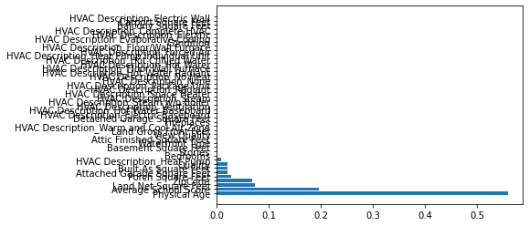
\*\* 9 features are selected.

Selected features for Random Forest Regressor are:

['Land Net Square Feet', 'Quality', 'Porch Square Feet', 'Attached Garage Square Feet', 'Built-As Square Feet', 'Stories', 'Physical Age', 'ZipCode', 'Average School Score']

Mean Score for RandomForest: 0.8710

R2 score from the Random Forest model: 0.89



Mean Score for RandomForestWithParams: 0.9219

R2 score from the Random Forest modelWithParams: 0.95

####################################################################

# 4.1.4.1.2.2 Feature Selection Using Gradient Boosting

####################################################################

# Find the significant features for Pierce County with their importance values.

importances = grbr\_C02019.feature\_importances\_

np.sum(importances)

plt.barh(fn\_C02019, importances)

# Draw a bar chart to see the sorted importance values with feature names.

df\_importances = pd.DataFrame(data=importances, index=fn\_C02019, columns=['importance\_value'])

df\_importances.sort\_values(by = 'importance\_value', ascending=False, inplace=True)

df\_importances

plt.barh(df\_importances.index, df\_importances.importance\_value)

# Build a model with a subset of those features.

selector = SelectFromModel(estimator=GradientBoostingRegressor(), threshold=0.015)

X\_C0\_reduced = selector.fit\_transform(X\_C02019, y\_C02019)

selector.threshold\_

selected\_TF = selector.get\_support()

print(f'\n\*\* {selected\_TF.sum()} features are selected.')

# Show those selected features.

selected\_features = []

for i,j in zip(selected\_TF, fn\_C02019):

if i: selected\_features.append(j)

print(f'Selected features for GBR are:\n {selected\_features}')

# Build a model using reduced number of features

X\_C0\_reduced\_train, X\_C0\_reduced\_test, y\_C0\_reduced\_train, y\_C0\_reduced\_test \

= train\_test\_split(X\_C0\_reduced, y\_C02019, test\_size =.3, random\_state=1234)

GradientBoostingRegressorModel(X\_C0\_reduced\_test, y\_C0\_reduced\_test, X\_C0\_reduced\_train, y\_C0\_reduced\_train)

#With Hyperparameters for Gradient Boosting

GradientBoostingRegressorModel\_WithParams(750, 10, X\_C0\_reduced\_test, y\_C0\_reduced\_test, X\_C0\_reduced\_train, y\_C0\_reduced\_train)

Output:

\*\* 6 features are selected.

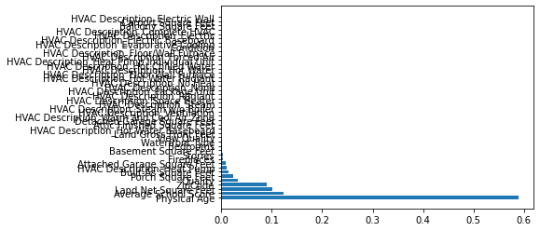
Selected features for GBR are:

['Land Net Square Feet', 'Quality', 'Porch Square Feet', 'Physical Age', 'ZipCode', 'Average School Score']

Mean Score for GradientBoosting: 0.7963

\*\*Performance Evaluations for Gradient Boosting\*\*

R2 score from the Gradient Boosting model: 0.82



Mean Score for GradientBoostingWithParams: 0.8516

R2 score from the Gradient Boosting modelWithParams 0.88

####################################################################

# 4.1.4.1.3 PCA For Cluster 0

####################################################################

pca\_prep = PCA().fit(X\_C0s\_2019)

pca\_prep.n\_components\_

pca\_prep.explained\_variance\_ratio\_

# Create a scree plot and find an "elbow" or an inflection point on the plot.

plt.plot(pca\_prep.explained\_variance\_ratio\_)

plt.xlabel('k number of components')

plt.ylabel('Explained variance')

plt.grid(True)

plt.show()

# From the scree plot, we choose k = 26

n\_pc = 26

pca\_C02019 = PCA(n\_components = n\_pc).fit(X\_C0s\_2019)

Xp\_C02019 = pca\_C02019.transform(X\_C0s\_2019)

print(f'After PCA, we use {pca\_C02019.n\_components\_} components.\n')

# Split the data into training and testing subsets.

Xp\_train\_C02019, Xp\_test\_C02019, yp\_train\_C02019, yp\_test\_C02019 = train\_test\_split(Xp\_C02019, y\_C02019, test\_size =.2,

random\_state = 1234)

# Create random forest models using the transformed data.

rfr\_C02019 = RandomForestRegressorModel(Xp\_test\_C02019, yp\_test\_C02019, Xp\_train\_C02019, yp\_train\_C02019)

# With Hyperparameters for Random Forest

RandomForestRegressorModel\_WithParams(1000, 15, Xp\_test\_C02019, yp\_test\_C02019, Xp\_train\_C02019, yp\_train\_C02019)

# For Gradient Boosting

gbr\_C02019 = GradientBoostingRegressorModel(Xp\_test\_C02019, yp\_test\_C02019, Xp\_train\_C02019, yp\_train\_C02019)

#With Hyperparameters for Gradient Boosting

GradientBoostingRegressorModel\_WithParams(750, 10, Xp\_test\_C02019, yp\_test\_C02019, Xp\_train\_C02019, yp\_train\_C02019)

# SVM Regression

svmr\_C02019 = SVMRegressorModel('linear', 10, 0.01, Xp\_test\_C02019, yp\_test\_C02019, Xp\_train\_C02019, yp\_train\_C02019)

# For Neural Network

# Neural Network models using the transformed data.

NeuralNetworkRegressorModel(Xp\_test\_C02019, yp\_test\_C02019, Xp\_train\_C02019, yp\_train\_C02019)

# Neural Network models with Optimized Parameters Values and transformed data.

rSquare = NeuralNetworkModel\_3Layer(50, 40, 20,

'relu', Xp\_test\_C02019, yp\_test\_C02019, Xp\_train\_C02019, yp\_train\_C02019)

Output:

A graph with a line

Description automatically generated

Mean Score for Random Forest: 0.7171

\*\*Performance Evaluation for Random Forest Regressor\*\*

R2 score from the Random Forest model: 0.75

Mean Score for RandomForestWithParams: 0.7854

R2 score from the Random Forest modelWithParams: 0.80

Mean Score for Gradient Boosting: 0.6918

\*\*Performance Evaluations for Gradient Boosting\*\*

R2 score from the Gradient Boosting model: 0.70

Mean Score for GradientBoostingWithParams: 0.7215

R2 score from the Gradient Boosting modelWithParams 0.74

Mean Score for SVM: -0.0730

\*\*Performance Evaluations for SVM Regressor\*\*

R2 score from the SVM model: -0.09

Mean Score for Neural Network: 0.2220

\*\*Performance Evaluations for Neural Network\*\*

R2 score from the NN model: 0.26

R2 score from the NN model Three Layers: relu 50 40 20 0.6032

####################################################################

# 4.1.4.2 Analyzing Cluster 1

####################################################################

X\_C12019 = c1\_2019.loc[:, c1\_2019.columns != 'Sale Price']

y\_C12019 = c1\_2019[['Sale Price']].values.ravel()

fn\_C12019 = X\_C12019.columns

#Scaling the variables

scaler = StandardScaler()

scaler.fit(X\_C12019)

X\_C1s\_2019 = scaler.transform(X\_C12019)

# Divide the dataset into training and testing data.

# Divide the scaled dataset into training and testing data

X\_train\_C1s\_2019, X\_test\_C1s\_2019, y\_train\_C12019, y\_test\_C12019 = train\_test\_split(X\_C1s\_2019, y\_C12019, test\_size =.30,random\_state=1234)

# Divide the unscaled dataset into training and testing data.

X\_train\_C12019, X\_test\_C12019, y\_train\_C12019, y\_test\_C12019 = train\_test\_split(X\_C12019, y\_C12019, test\_size =.30,random\_state=1234)

####################################################################

# 4.1.4.2.1 Building Models Wth Default Parameters for Cluster 1

####################################################################

# Decision Tree Regressor

dtr\_C12019 = DecisionTreeRegressorModel(X\_test\_C12019, y\_test\_C12019, X\_train\_C12019, y\_train\_C12019)

# Random Forest Regressor

rfr\_C12019 = RandomForestRegressorModel(X\_test\_C12019, y\_test\_C12019, X\_train\_C12019, y\_train\_C12019)

# With Hyperparameters for Random Forest

RandomForestRegressorModel\_WithParams(1000, 15, X\_test\_C12019, y\_test\_C12019, X\_train\_C12019, y\_train\_C12019)

# Gradient Boosting Regression

grbr\_C12019 = GradientBoostingRegressorModel(X\_test\_C12019, y\_test\_C12019, X\_train\_C12019, y\_train\_C12019)

#With Hyperparameters for Gradient Boosting

GradientBoostingRegressorModel\_WithParams(750, 10, X\_test\_C12019, y\_test\_C12019, X\_train\_C12019, y\_train\_C12019)

# SVM Regression

svmr\_C12019 = SVMRegressorModel('linear', 10, 0.01, X\_test\_C1s\_2019, y\_test\_C12019, X\_train\_C1s\_2019, y\_train\_C12019)

# NN Regression

# Neural network with default parameters

NeuralNetworkRegressorModel(X\_test\_C1s\_2019, y\_test\_C12019, X\_train\_C1s\_2019, y\_train\_C12019)

# Optimizing the NN Model using Hyper Parameters Tuning Using Grid Search

gridSearch\_2019 = FindNeuralNetworkParam\_GridSearch(X\_test\_C1s\_2019, y\_test\_C12019, X\_train\_C1s\_2019, y\_train\_C12019)

results\_gs = pd.DataFrame(gridSearch\_2019.cv\_results\_)

# Evaluating Neural network results with grid search best estimators values

rSquare = NeuralNetworkModel\_3Layer(50, 40, 20,

'relu', X\_test\_C1s\_2019, y\_test\_C12019, X\_train\_C1s\_2019, y\_train\_C12019)

Output:

Mean Score for Decision Tree: 0.8846

\*\*Performance Evaluations for Decision Tree\*\*

R2 score from the Decision Tree model: 0.83

Mean Score for RandomForest: 0.8487

\*\*Evaluation of Errors for Random Forest Regressor\*\*

R2 score from the Random Forest model: 0.87

Mean Score for RandomForestWithParams: 0.9216

R2 score from the Random Forest modelWithParams: 0.91

Mean Score for GradientBoosting: 0.8265

\*\*Performance Evaluations for Gradient Boosting\*\*

R2 score from the Gradient Boosting model: 0.84

Mean Score for GradientBoostingWithParams: 0. 8896

R2 score from the Gradient Boosting modelWithParams 0.87

Mean Score for SVM: 0.0283

\*\*Performance Evaluations for SVM Regressor\*\*

R2 score from the SVM model: 0.02

Mean Score for Neural Network: 0.23

\*\*Performance Evaluations for Neural Network\*\*

R2 score from the NN model: 0.21

R2 score from the NN model Three Layers: relu 50 40 20 0.61

####################################################################

# 4.1.4.2.2 Feature Selection for cluster 1

####################################################################

####################################################################

# 4.1.4.2.2.1 Feature Selection Using Random Forest Regressor

####################################################################

# Find the significant features for Pierce County with their importance values.

importances = rfr\_C12019.feature\_importances\_

np.sum(importances)

plt.barh(fn\_C12019, importances)

# Draw a bar chart to see the sorted importance values with feature names.

df\_importances = pd.DataFrame(data=importances, index=fn\_C12019, columns=['importance\_value'])

df\_importances.sort\_values(by = 'importance\_value', ascending=False, inplace=True)

df\_importances

df\_importance\_top15 = df\_importances.head(11)

plt.barh(df\_importance\_top15.index, df\_importance\_top15.importance\_value)

plt.barh(df\_importances.index, df\_importances.importance\_value)

# Build a model with a subset of those features.

selector = SelectFromModel(estimator=RandomForestRegressor(), threshold= 0.015)

X\_C1\_reduced = selector.fit\_transform(X\_C12019, y\_C12019)

selector.threshold\_

selected\_TF = selector.get\_support()

print(f'\n\*\* {selected\_TF.sum()} features are selected.')

# Show those selected features.

selected\_features = []

for i,j in zip(selected\_TF, fn\_C12019):

if i: selected\_features.append(j)

print(f'Selected features for Random Forest Regressor are:\n {selected\_features}')

# Build a model using reduced number of features

X\_C1\_reduced\_train, X\_C1\_reduced\_test, y\_C1\_reduced\_train, y\_C1\_reduced\_test \

= train\_test\_split(X\_C1\_reduced, y\_C12019, test\_size =.3, random\_state=1234)

RandomForestRegressorModel(X\_C1\_reduced\_test, y\_C1\_reduced\_test, X\_C1\_reduced\_train, y\_C1\_reduced\_train)

# With Hyperparameters for Random Forest

RandomForestRegressorModel\_WithParams(1000, 15, X\_C1\_reduced\_test, y\_C1\_reduced\_test, X\_C1\_reduced\_train, y\_C1\_reduced\_train)

Output:

\*\* 8 features are selected.

Selected features for Random Forest Regressor are:

['Land Net Square Feet', 'Basement Square Feet', 'Porch Square Feet', 'Attached Garage Square Feet', 'Built-As Square Feet', 'Physical Age', 'ZipCode', 'Average School Score']

Mean Score for RandomForest: 0.6964

\*\*Performance Evaluation for Random Forest Regressor\*\*

R2 score from the Random Forest model: 0.71

Mean Score for RandomForestWithParams: 0.7423

R2 score from the Random Forest modelWithParams: 0.75

####################################################################

# 4.1.4.1.2.2 Feature Selection Using Gradient Boosting

####################################################################

# Find the significant features for Pierce County with their importance values.

importances = grbr\_C12019.feature\_importances\_

np.sum(importances)

plt.barh(fn\_C12019, importances)

# Draw a bar chart to see the sorted importance values with feature names.

df\_importances = pd.DataFrame(data=importances, index=fn\_C12019, columns=['importance\_value'])

df\_importances.sort\_values(by = 'importance\_value', ascending=False, inplace=True)

df\_importances

plt.barh(df\_importances.index, df\_importances.importance\_value)

# Build a model with a subset of those features.

selector = SelectFromModel(estimator=GradientBoostingRegressor(), threshold=0.015)

X\_C1\_reduced = selector.fit\_transform(X\_C12019, y\_C12019)

selector.threshold\_

selected\_TF = selector.get\_support()

print(f'\n\*\* {selected\_TF.sum()} features are selected.')

# Show those selected features.

selected\_features = []

for i,j in zip(selected\_TF, fn\_C12019):

if i: selected\_features.append(j)

print(f'Selected features for GBR are:\n {selected\_features}')

# Build a model using reduced number of features

X\_C1\_reduced\_train, X\_C1\_reduced\_test, y\_C1\_reduced\_train, y\_C1\_reduced\_test \

= train\_test\_split(X\_C1\_reduced, y\_C12019, test\_size =.3, random\_state=1234)

GradientBoostingRegressorModel(X\_C1\_reduced\_test, y\_C1\_reduced\_test, X\_C1\_reduced\_train, y\_C1\_reduced\_train)

#With Hyperparameters for Gradient Boosting

GradientBoostingRegressorModel\_WithParams(750, 10, X\_C1\_reduced\_test, y\_C1\_reduced\_test, X\_C1\_reduced\_train, y\_C1\_reduced\_train)

Output:

\*\* 6 features are selected.

Selected features for GBR are:

['Land Net Square Feet', 'Attached Garage Square Feet', 'Built-As Square Feet', 'Physical Age', 'ZipCode', 'Average School Score']

Mean Score for GradientBoosting: 0.7109

\*\*Performance Evaluations for Gradient Boosting\*\*

R2 score from the Gradient Boosting model: 0.73

Mean Score for GradientBoostingWithParams: 0.7697

R2 score from the Gradient Boosting modelWithParams 0.78

####################################################################

# 4.1.4.2.3 PCA For Cluster 1

####################################################################

pca\_prep = PCA().fit(X\_C1s\_2019)

pca\_prep.n\_components\_

pca\_prep.explained\_variance\_ratio\_

# Create a scree plot and find an "elbow" or an inflection point on the plot.

plt.plot(pca\_prep.explained\_variance\_ratio\_)

plt.xlabel('k number of components')

plt.ylabel('Explained variance')

plt.title('Scree Plot 2019 Cluster2')

plt.grid(True)

plt.show()

# From the scree plot, we choose k = 26

n\_pc = 26

pca\_C12019 = PCA(n\_components = n\_pc).fit(X\_C1s\_2019)

Xp\_C12019 = pca\_C12019.transform(X\_C1s\_2019)

print(f'After PCA, we use {pca\_C12019.n\_components\_} components.\n')

# Split the data into training and testing subsets.

Xp\_train\_C12019, Xp\_test\_C12019, yp\_train\_C12019, yp\_test\_C12019 = train\_test\_split(Xp\_C12019, y\_C12019, test\_size =.2,

random\_state = 1234)

# Create random forest models using the transformed data.

rfr\_C12019 = RandomForestRegressorModel(Xp\_test\_C12019, yp\_test\_C12019, Xp\_train\_C12019, yp\_train\_C12019)

# With Hyperparameters for Random Forest

RandomForestRegressorModel\_WithParams(1000, 15, Xp\_test\_C12019, yp\_test\_C12019, Xp\_train\_C12019, yp\_train\_C12019)

# For Gradient Boosting

gbr\_C12019 = GradientBoostingRegressorModel(Xp\_test\_C12019, yp\_test\_C12019, Xp\_train\_C12019, yp\_train\_C12019)

#With Hyperparameters for Gradient Boosting

GradientBoostingRegressorModel\_WithParams(750, 10, Xp\_test\_C12019, yp\_test\_C12019, Xp\_train\_C12019, yp\_train\_C12019)

# For Neural Network

# Neural Network models using the transformed data.

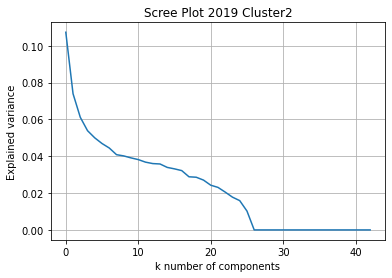
NeuralNetworkRegressorModel(Xp\_test\_C12019, yp\_test\_C12019, Xp\_train\_C12019, yp\_train\_C12019)

# Neural Network models with Optimized Parameters Values and transformed data.

rSquare = NeuralNetworkModel\_3Layer(50, 40, 20,

'relu', Xp\_test\_C12019, yp\_test\_C12019, Xp\_train\_C12019, yp\_train\_C12019)

Output:



Mean Score for Random Forest: 0.7734

\*\*Performance Metrics for Random Forest Regressor\*\*

R2 score from the Random Forest model: 0.79

Mean Score for RandomForestWithParams: 0.8596

R2 score from the Random Forest modelWithParams: 0.82

Mean Score for GradientBoosting: 0.7245

\*\*Performance Evaluations for Gradient Boosting\*\*

R2 score from the Gradient Boosting model: 0.74

Mean Score for GradientBoostingWithParams: 0.7720

R2 score from the Gradient Boosting modelWithParams 0.79

Mean Score for Neural Network: 0.30

\*\*Performance Evaluations for Neural Network\*\*

R2 score from the NN model: 0.28

R2 score from the NN model Three Layers: relu 50 40 20 0.69

####################################################################

# 4.2 Analyzing Post-Covid (2022) Datasets

####################################################################

X\_2022 = df\_merged2022.loc[:, df\_merged2022.columns != 'Sale Price']

y\_2022 = df\_merged2022[['Sale Price']].values.ravel()

fn\_2022 = X\_2022.columns

#Scaling the variables

scaler = StandardScaler()

scaler.fit(X\_2022)

X\_s\_2022 = scaler.transform(X\_2022)

# Divide the scaled dataset into training and testing data

X\_train\_s\_2022, X\_test\_s\_2022, y\_train\_2022, y\_test\_2022 = train\_test\_split(X\_s\_2022, y\_2022, test\_size =.30,random\_state=1234)

# Divide the unscaled dataset into training and testing data.

X\_train\_2022, X\_test\_2022, y\_train\_2022, y\_test\_2022 = train\_test\_split(X\_2022, y\_2022, test\_size =.30,random\_state=1234)

####################################################################

# 4.2.1 Building Models Wth Default Parameters

####################################################################

# Decision Tree Regressor

dtr\_2022 = DecisionTreeRegressorModel(X\_test\_2022, y\_test\_2022, X\_train\_2022, y\_train\_2022)

# Random Forest Regressor

rfr\_2022 = RandomForestRegressorModel(X\_test\_2022, y\_test\_2022, X\_train\_2022, y\_train\_2022)

# Grid Seach for best Random Forest Regressor

FindRandomForestRegressorParam\_GridSearch(X\_test\_2022, y\_test\_2022, X\_train\_2022, y\_train\_2022)

# With Hyperparameters for Random Forest

RandomForestRegressorModel\_WithParams(1000, 15, X\_test\_2022, y\_test\_2022, X\_train\_2022, y\_train\_2022)

# Gradient Boosting Regression

gbr\_2022 = GradientBoostingRegressorModel(X\_test\_2022, y\_test\_2022, X\_train\_2022, y\_train\_2022)

# Grid Seach for best Gradient Boosting

FindGradientBoostingRegressorParam\_GridSearch(X\_test\_2022, y\_test\_2022, X\_train\_2022, y\_train\_2022)

#With Hyperparameters for Gradient Boosting

GradientBoostingRegressorModel\_WithParams(500, 10, X\_test\_2022, y\_test\_2022, X\_train\_2022, y\_train\_2022)

# SVM Regression

svmr\_2022 = SVMRegressorModel('linear', 10, 0.01, X\_test\_s\_2022, y\_test\_2022, X\_train\_s\_2022, y\_train\_2022)

# NN Regression

nnr\_2022 = NeuralNetworkRegressorModel(X\_test\_s\_2022, y\_test\_2022, X\_train\_s\_2022, y\_train\_2022)

# Optimizing the NN Model using Hyper Parameters Tuning Using Grid Search

gridSearch\_2022 = FindNeuralNetworkParam\_GridSearch( X\_test\_s\_2022, y\_test\_2022, X\_train\_s\_2022, y\_train\_2022)

results\_gs = pd.DataFrame(gridSearch\_2022.cv\_results\_)

# Evaluating Neural network results with grid search best estimators values

rSquare\_2022 = NeuralNetworkModel\_3Layer(50, 40, 20,

'relu', X\_test\_s\_2022, y\_test\_2022, X\_train\_s\_2022, y\_train\_2022)

Output:

Mean Score for Decision Tree: 0.8879

\*\*Performance Evaluations for Decision Tree\*\*

R2 score from the Decision Tree model: 0.85

Mean Score for RandomForest: 0.8010

\*\* Performance Evaluations for Random Forest Regressor\*\*

R2 score from the Random Forest model: 0.82

Mean Score for RandomForestWithParams: 0.8562

R2 score from the Random Forest modelWithParams: 0.87

Mean Score for GradientBoosting: 0.8109

\*\*Performance Evaluations for Gradient Boosting\*\*

R2 score from the Gradient Boosting model: 0.84

Mean Score for GradientBoostingWithParams: 0.8720

R2 score from the Gradient Boosting modelWithParams 0.88

Mean Score for SVM: 0.142

\*\*Performance Evaluations for SVM Regressor\*\*

R2 score from the SVM model: 0.02

Mean Score for Neural Network: -0.2060

\*\*Performance Evaluations for Neural Network\*\*

R2 score from the NN model: 0.18

The best estimators: MLPClassifier(hidden\_layer\_sizes=(50, 40, 20))

The best parameters for Layers:

{'activation': 'relu', 'hidden\_layer\_sizes': (50, 40, 20)}

The best score for Layers: 0.83

####################################################################

# 4.2.2 Feature Selection

####################################################################

####################################################################

# 4.2.2.1 Feature Selection Using Random Forest Regressor

####################################################################

# Find the significant features for Pierce County Dataset with their importance values.

importances = rfr\_2022.feature\_importances\_

np.sum(importances)

plt.barh(fn\_2022, importances)

# Draw a bar chart to see the sorted importance values with feature names.

df\_importances = pd.DataFrame(data=importances, index=fn\_2022, columns=['importance\_value'])

df\_importances.sort\_values(by = 'importance\_value', ascending=False, inplace=True)

df\_importances

df\_importance\_top15 = df\_importances.head(1)

plt.barh(df\_importance\_top15.index, df\_importance\_top15.importance\_value)

plt.barh(df\_importances.index, df\_importances.importance\_value)

# Build a model with a subset of those features.

selector = SelectFromModel(estimator=RandomForestRegressor(), threshold=0.015)

X\_reduced = selector.fit\_transform(X\_2022, y\_2022)

selector.threshold\_

selected\_TF = selector.get\_support()

print(f'\n\*\* {selected\_TF.sum()} features are selected.')

# Show those selected features.

selected\_features = []

for i,j in zip(selected\_TF, fn\_2022):

if i: selected\_features.append(j)

print(f'Selected features are:\n {selected\_features}')

# Build a model using reduced number of features

X\_reduced\_train, X\_reduced\_test, y\_reduced\_train, y\_reduced\_test \

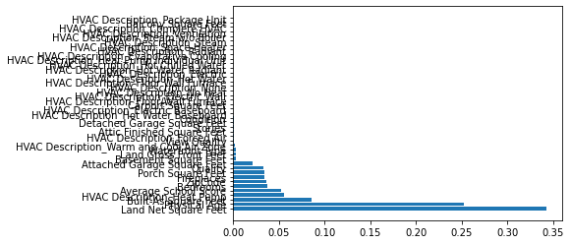
= train\_test\_split(X\_reduced, y\_2022, test\_size =.3, random\_state=1234)

RandomForestRegressorModel(X\_reduced\_test, y\_reduced\_test, X\_reduced\_train, y\_reduced\_train)

# With Hyperparameters for Random Forest

RandomForestRegressorModel\_WithParams(1000, 15, X\_reduced\_test, y\_reduced\_test, X\_reduced\_train, y\_reduced\_train)

Output:



\*\* 11 features are selected.

Selected features are:

['Land Net Square Feet', 'Quality', 'Porch Square Feet', 'Attached Garage Square Feet', 'Fireplaces', 'Built-As Square Feet', 'Bedrooms', 'Physical Age', 'HVAC Description\_Heat Pump', 'ZipCode', 'Average School Score']

Mean Score for RandomForest: 0.7976

R2 score from the Random Forest modelWithParams: 0.81

Mean Score for RandomForestWithParams: 0.8438

R2 score from the Random Forest modelWithParams: 0.86

####################################################################

# 4.2.2.2 Feature Selection Using Gradient Booster Regressor

####################################################################

# Find the significant features for Pierce County Data with their importance values.

importances = gbr\_2022.feature\_importances\_

np.sum(importances)

plt.barh(fn\_2022, importances)

# Draw a bar chart to see the sorted importance values with feature names.

df\_importances = pd.DataFrame(data=importances, index=fn\_2022, columns=['importance\_value'])

df\_importances.sort\_values(by = 'importance\_value', ascending=False, inplace=True)

df\_importances

plt.barh(df\_importances.index, df\_importances.importance\_value)

# Build a model with a subset of those features.

selector = SelectFromModel(estimator=GradientBoostingRegressor(), threshold=0.015)

X\_reduced = selector.fit\_transform(X\_2022, y\_2022)

selector.threshold\_

selected\_TF = selector.get\_support()

print(f'\n\*\* {selected\_TF.sum()} features are selected.')

# Show those selected features.

selected\_features = []

for i,j in zip(selected\_TF, fn\_2022):

if i: selected\_features.append(j)

print(f'Selected features for 2022 GBR are:\n {selected\_features}')

# Build a model using reduced number of features

X\_reduced\_train, X\_reduced\_test, y\_reduced\_train, y\_reduced\_test \

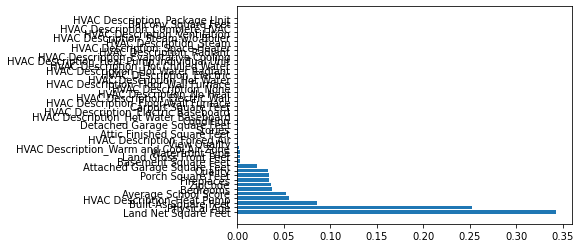
= train\_test\_split(X\_reduced, y\_2022, test\_size =.3, random\_state=1234)

GradientBoostingRegressorModel(X\_reduced\_test, y\_reduced\_test, X\_reduced\_train, y\_reduced\_train)

#With Hyperparameters for Gradient Boosting

GradientBoostingRegressorModel\_WithParams(750, 10, X\_reduced\_test, y\_reduced\_test, X\_reduced\_train, y\_reduced\_train)

Output:



\*\* 8 features are selected.

Selected features for 2022 GBR are:

['Land Net Square Feet', 'Quality', 'Fireplaces', 'Built-As Square Feet', 'Physical Age', 'HVAC Description\_Heat Pump', 'ZipCode', 'Average School Score']

Mean Score for GradientBoosting: 0.8143

\*\*Performance Evaluations for Gradient Boosting\*\*

R2 score from the Gradient Boosting model: 0.83

Mean Score for GradientBoostingWithParams: 0.8598

R2 score from the GradientBoostingWithParams: 0.88

####################################################################

# 4.2.3 PCA

####################################################################

pca\_prep = PCA().fit(X\_s\_2022)

pca\_prep.n\_components\_

# Consider the variances as the amount of information. Drop components providing less information

# (low variances)

pca\_prep.explained\_variance\_ratio\_

# Create a scree plot and find an "elbow" or an inflection point on the plot.

plt.plot(pca\_prep.explained\_variance\_ratio\_)

plt.xlabel('k number of components')

plt.ylabel('Explained variance')

plt.grid(True)

plt.show()

# From the scree plot, we choose k = 28

n\_pc = 28

pca\_2022 = PCA(n\_components = n\_pc).fit(X\_s\_2022)

Xp\_2022 = pca\_2022.transform(X\_s\_2022)

print(f'After PCA, we use {pca\_2022.n\_components\_} components.\n')

# Split the data into training and testing subsets.

Xp\_train\_2022, Xp\_test\_2022, yp\_train\_2022, yp\_test\_2022 = train\_test\_split(Xp\_2022, y\_2022, test\_size =.3, random\_state = 1234)

# Create random forest models using the transformed data.

rfr\_2022 = RandomForestRegressorModel(Xp\_test\_2022, yp\_test\_2022, Xp\_train\_2022, yp\_train\_2022)

# With Hyperparameters for Random Forest

RandomForestRegressorModel\_WithParams(1000, 15, Xp\_test\_2022, yp\_test\_2022, Xp\_train\_2022, yp\_train\_2022)

# For Gradient Boosting

gbr\_2022 = GradientBoostingRegressorModel(Xp\_test\_2022, yp\_test\_2022, Xp\_train\_2022, yp\_train\_2022)

#With Hyperparameters for Gradient Boosting

GradientBoostingRegressorModel\_WithParams(750, 10, Xp\_test\_2022, yp\_test\_2022, Xp\_train\_2022, yp\_train\_2022)

# Neural Network models using the transformed data.

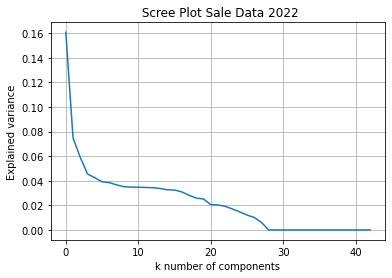
NeuralNetworkRegressorModel(Xp\_test\_2022, yp\_test\_2022, Xp\_train\_2022, yp\_train\_2022)

# Neural Network models with Optimized Parameters Values and transformed data.

rSquare = NeuralNetworkModel\_3Layer(50, 40, 20,

'relu', Xp\_test\_2022, yp\_test\_2022, Xp\_train\_2022, yp\_train\_2022)

Output:



Mean Score for RandomForest: 0.7388

\*\*Performance Metrics Random Forest Regressor\*\*

R2 score from the Random Forest model: 0.76

Mean Score for RandomForestWithParams: 0.7987

R2 score from the Random Forest modelWithParams: 0.80

Mean Score for GradientBoosting: 0.7965

R2 score from the Gradient Boosting model: 0.81

Mean Score for GradientBoostingWithParams: 0.8213

R2 score from the Gradient Boosting modelWithParams 0.84

Mean Score for SVM: 0.0142

\*\*Performance Evaluations for SVM Regressor\*\*

R2 score from the SVM model: 0.02

Mean Score for Neural Network: 0.22

\*\*Performance Evaluations for Neural Network\*\*

R2 score from the NN model: 0.19

R2 score from the NN model Three Layers: relu 50 40 20 0.71

####################################################################

# 4.2.4 Clustering

####################################################################

n\_pc = 2

pca\_2022 = PCA(n\_components = n\_pc).fit(X\_s\_2022)

Xp\_2022\_2Comp = pca\_2022.transform(X\_s\_2022)

# Create an instance (object) of the KMeans class with the parameters

# initialized (cluster count 2)

km\_2022 = KMeans(n\_clusters=2, random\_state=1234)

# Build a model.

km\_2022 = km\_2022.fit\_predict(Xp\_2022\_2Comp)

silhouette\_avg = silhouette\_score(Xp\_2022\_2Comp, km\_2022)

print('Silhouette Score:', silhouette\_avg)

# Splitting dataset into two clusters

c0\_2022 = df\_merged2022[km\_2022 == 0]

c1\_2022 = df\_merged2022[km\_2022 == 1]

c0\_2022.shape

c1\_2022.shape

plt.scatter(Xp\_2022\_2Comp[:, 0], Xp\_2022\_2Comp[:, 1], c=km\_2022, s=50, cmap='viridis')

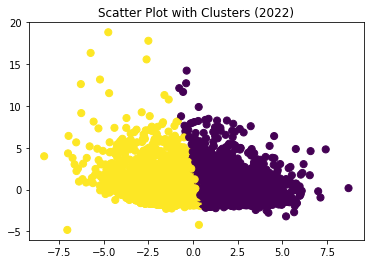
plt.show()

Output:

Silhouette Score: 0.45787567925493555

(4341, 44)

(5906, 44)



####################################################################

# 4.2.4.1 Analyzing Cluster 0

####################################################################

X\_C02022 = c0\_2022.loc[:, c0\_2022.columns != 'Sale Price']

y\_C02022 = c0\_2022[['Sale Price']].values.ravel()

fn\_C02022 = X\_C02022.columns

#Scaling the variables

scaler = StandardScaler()

scaler.fit(X\_C02022)

X\_C0s\_2022 = scaler.transform(X\_C02022)

####################################################################

# 4.2.4.1.1 Building Models Wth Default Parameters for Cluster 0

####################################################################

# Divide the scaled dataset into training and testing data

X\_train\_C0s\_2022, X\_test\_C0s\_2022, y\_train\_C02022, y\_test\_C02022 = train\_test\_split(X\_C0s\_2022, y\_C02022, test\_size =.30,random\_state=1234)

# Divide the dataset into training and testing data.

X\_train\_C02022, X\_test\_C02022, y\_train\_C02022, y\_test\_C02022 = train\_test\_split(X\_C02022, y\_C02022, test\_size =.30,random\_state=1234)

# Decision Tree Regressor

dtr\_C02022 = DecisionTreeRegressorModel(X\_test\_C02022, y\_test\_C02022, X\_train\_C02022, y\_train\_C02022)

# Random Forest Regressor

rfr\_C02022 = RandomForestRegressorModel(X\_test\_C02022, y\_test\_C02022, X\_train\_C02022, y\_train\_C02022)

# Grid Seach for best Random Forest Regressor

FindRandomForestRegressorParam\_GridSearch(X\_test\_C02022, y\_test\_C02022, X\_train\_C02022, y\_train\_C02022)

# With Hyperparameters for Random Forest

RandomForestRegressorModel\_WithParams(1000, 15, X\_test\_C02022, y\_test\_C02022, X\_train\_C02022, y\_train\_C02022)

# Gradient Boosting Regression

grbr\_C02022 = GradientBoostingRegressorModel(X\_test\_C02022, y\_test\_C02022, X\_train\_C02022, y\_train\_C02022)

# Grid Seach for best Gradient Boosting

FindGradientBoostingRegressorParam\_GridSearch(X\_test\_C02022, y\_test\_C02022, X\_train\_C02022, y\_train\_C02022)

#With Hyperparameters for Gradient Boosting

GradientBoostingRegressorModel\_WithParams(750, 10, X\_test\_C02022, y\_test\_C02022, X\_train\_C02022, y\_train\_C02022)

# SVM Regression

svmr\_C02022 = SVMRegressorModel('linear', 10, 0.01, X\_test\_C0s\_2022, y\_test\_C02022, X\_train\_C0s\_2022, y\_train\_C02022)

# NN Regression

# Neural network with default parameters

NeuralNetworkRegressorModel(X\_test\_C0s\_2022, y\_test\_C02022, X\_train\_C0s\_2022, y\_train\_C02022)

# Optimizing the NN Model using Hyper Parameters Tuning Using Grid Search

gridSearch\_2019 = FindNeuralNetworkParam\_GridSearch(X\_test\_C0s\_2022, y\_test\_C02022, X\_train\_C0s\_2022, y\_train\_C02022)

results\_gs = pd.DataFrame(gridSearch\_2019.cv\_results\_)

# Evaluating Neural network results with grid search best estimators values

rSquare = NeuralNetworkModel\_3Layer(50, 40, 20,

'relu', X\_test\_C0s\_2022, y\_test\_C02022, X\_train\_C0s\_2022, y\_train\_C02022)

Output:

Mean Score for Random Forest: 0.7007

\*\*Evaluation of Errors for Random Forest Regressor\*\*

R2 score from the Random Forest model: 0.71

Mean Score for RandomForestWithParams: 0.7394

R2 score from the Random Forest modelWithParams: 0.75

Mean Score for GradientBoosting: 0.6987

\*\*Performance Evaluations for Gradient Boosting\*\*

R2 score from the Gradient Boosting model: 0.72

Mean Score for GradientBoostingWithParams: 0.7431

R2 score from the Gradient Boosting modelWithParams 0.76

Mean Score for SVM: 0.1359

\*\*Performance Evaluations for SVM Regressor\*\*

R2 score from the SVM model: 0.12

Mean Score for Neural Network: 0.36

\*\*Performance Evaluations for Neural Network\*\*

R2 score from the NN model: 0.27

R2 score from the NN model Three Layers: relu 50 40 20 0.70

####################################################################

# 4.2.4.1.2 Feature Selection for cluster 0

####################################################################

####################################################################

# 4.2.4.1.2.1 Feature Selection Using Random Forest Regressor

####################################################################

# Find the significant features for Pierce County with their importance values.

importances = rfr\_C02022.feature\_importances\_

np.sum(importances)

plt.barh(fn\_C02022, importances)

# Draw a bar chart to see the sorted importance values with feature names.

df\_importances = pd.DataFrame(data=importances, index=fn\_C02022, columns=['importance\_value'])

df\_importances.sort\_values(by = 'importance\_value', ascending=False, inplace=True)

df\_importances

df\_importance\_top15 = df\_importances.head(11)

plt.barh(df\_importance\_top15.index, df\_importance\_top15.importance\_value)

plt.barh(df\_importances.index, df\_importances.importance\_value)

# Build a model with a subset of those features.

selector = SelectFromModel(estimator=RandomForestRegressor(), threshold= 0.015)

X\_C0\_reduced = selector.fit\_transform(X\_C02022, y\_C02022)

selector.threshold\_

selected\_TF = selector.get\_support()

print(f'\n\*\* {selected\_TF.sum()} features are selected.')

# Show those selected features.

selected\_features = []

for i,j in zip(selected\_TF, fn\_C02022):

if i: selected\_features.append(j)

print(f'Selected features for Random Forest Regressor are:\n {selected\_features}')

# Build a model using reduced number of features

X\_C0\_reduced\_train, X\_C0\_reduced\_test, y\_C0\_reduced\_train, y\_C0\_reduced\_test \

= train\_test\_split(X\_C0\_reduced, y\_C02022, test\_size =.3, random\_state=1234)

RandomForestRegressorModel(X\_C0\_reduced\_test, y\_C0\_reduced\_test, X\_C0\_reduced\_train, y\_C0\_reduced\_train)

# With Hyperparameters for Random Forest

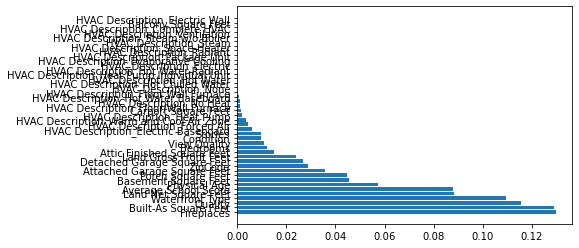
RandomForestRegressorModel\_WithParams(1000, 15, X\_C0\_reduced\_test, y\_C0\_reduced\_test, X\_C0\_reduced\_train, y\_C0\_reduced\_train)

Output:

\*\* 13 features are selected.

Selected features for Random Forest Regressor are:

['Land Net Square Feet', 'Land Gross Front Feet', 'Waterfront Type', 'View Quality', 'Quality', 'Basement Square Feet', 'Porch Square Feet', 'Attached Garage Square Feet', 'Fireplaces', 'Built-As Square Feet', 'Physical Age', 'ZipCode', 'Average School Score']



Mean Score for Random Forest: 0.6864

\*\*Performance Metrics for Random Forest Regressor\*\*

R2 score from the Random Forest model: 0.69

Mean Score for RandomForestWithParams: 0.7392

R2 score from the Random Forest modelWithParams: 0.76

####################################################################

# 4.2.4.1.2.2 Feature Selection Using Gradient Boosting

####################################################################

# Find the significant features for Pierce County with their importance values.

importances = grbr\_C02022.feature\_importances\_

np.sum(importances)

plt.barh(fn\_C02022, importances)

# Draw a bar chart to see the sorted importance values with feature names.

df\_importances = pd.DataFrame(data=importances, index=fn\_C02022, columns=['importance\_value'])

df\_importances.sort\_values(by = 'importance\_value', ascending=False, inplace=True)

df\_importances

plt.barh(df\_importances.index, df\_importances.importance\_value)

# Build a model with a subset of those features.

selector = SelectFromModel(estimator=GradientBoostingRegressor(), threshold=0.015)

X\_C0\_reduced = selector.fit\_transform(X\_C02022, y\_C02022)

selector.threshold\_

selected\_TF = selector.get\_support()

print(f'\n\*\* {selected\_TF.sum()} features are selected.')

# Show those selected features.

selected\_features = []

for i,j in zip(selected\_TF, fn\_C02022):

if i: selected\_features.append(j)

print(f'Selected features for GBR are:\n {selected\_features}')

# Build a model using reduced number of features

X\_C0\_reduced\_train, X\_C0\_reduced\_test, y\_C0\_reduced\_train, y\_C0\_reduced\_test \

= train\_test\_split(X\_C0\_reduced, y\_C02022, test\_size =.3, random\_state=1234)

GradientBoostingRegressorModel(X\_C0\_reduced\_test, y\_C0\_reduced\_test, X\_C0\_reduced\_train, y\_C0\_reduced\_train)

#With Hyperparameters for Gradient Boosting

GradientBoostingRegressorModel\_WithParams(750, 10, X\_C0\_reduced\_test, y\_C0\_reduced\_test, X\_C0\_reduced\_train, y\_C0\_reduced\_train)

Output:

\*\* 11 features are selected.

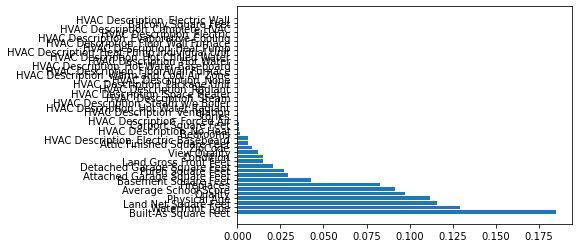
Selected features for GBR are:

['Land Net Square Feet', 'Waterfront Type', 'Quality', 'Basement Square Feet', 'Porch Square Feet', 'Attached Garage Square Feet', 'Fireplaces', 'Built-As Square Feet', 'Physical Age', 'ZipCode', 'Average School Score']

Mean Score for GradientBoosting: 0.6452

\*\*Performance Evaluations for Gradient Boosting\*\*

R2 score from the Gradient Boosting model: 0.68



Mean Score for GradientBoostingWithParams: 0.6892

R2 score from the Gradient Boosting modelWithParams 0.70

####################################################################

# 4.2.4.1.3 PCA For Cluster 0

####################################################################

pca\_prep = PCA().fit(X\_C0s\_2022)

pca\_prep.n\_components\_

# Consider the variances as the amount of information. Drop components providing less information

# (low variances)

pca\_prep.explained\_variance\_ratio\_

# Create a scree plot and find an "elbow" or an inflection point on the plot.

plt.plot(pca\_prep.explained\_variance\_ratio\_)

plt.xlabel('k number of components')

plt.ylabel('Explained variance')

plt.grid(True)

plt.show()

# From the scree plot, we choose k = 26

n\_pc = 26

pca\_C02022 = PCA(n\_components = n\_pc).fit(X\_C0s\_2022)

Xp\_C02022 = pca\_C02022.transform(X\_C0s\_2022)

print(f'After PCA, we use {pca\_C02022.n\_components\_} components.\n')

# Split the data into training and testing subsets.

Xp\_train\_C02022, Xp\_test\_C02022, yp\_train\_C02022, yp\_test\_C02022 = train\_test\_split(Xp\_C02022, y\_C02022, test\_size =.2,

random\_state = 1234)

# Create random forest models using the transformed data.

rfr\_C02022 = RandomForestRegressorModel(Xp\_test\_C02022, yp\_test\_C02022, Xp\_train\_C02022, yp\_train\_C02022)

# Grid Seach for best Random Forest Regressor

FindRandomForestRegressorParam\_GridSearch(Xp\_test\_C02022, yp\_test\_C02022, Xp\_train\_C02022, yp\_train\_C02022)

# With Hyperparameters for Random Forest

RandomForestRegressorModel\_WithParams(1000, 15, Xp\_test\_C02022, yp\_test\_C02022, Xp\_train\_C02022, yp\_train\_C02022)

# For Gradient Boosting

gbr\_C02022 = GradientBoostingRegressorModel(Xp\_test\_C02022, yp\_test\_C02022, Xp\_train\_C02022, yp\_train\_C02022)

# Grid Seach for best Gradient Boosting

FindGradientBoostingRegressorParam\_GridSearch(Xp\_test\_C02022, yp\_test\_C02022, Xp\_train\_C02022, yp\_train\_C02022)

#With Hyperparameters for Gradient Boosting

GradientBoostingRegressorModel\_WithParams(750, 10, Xp\_test\_C02022, yp\_test\_C02022, Xp\_train\_C02022, yp\_train\_C02022)

# For Neural Network

# Neural Network models using the transformed data.

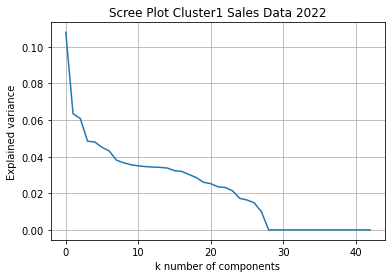
NeuralNetworkRegressorModel(Xp\_test\_C02022, yp\_test\_C02022, Xp\_train\_C02022, yp\_train\_C02022)

# Neural Network models with Optimized Parameters Values and transformed data.

rSquare = NeuralNetworkModel\_3Layer(50, 40, 20,

'relu', Xp\_test\_C02022, yp\_test\_C02022, Xp\_train\_C02022, yp\_train\_C02022)

Output:



Mean Score for Random Forest: 0.6782

\*\*Performance Metrics for Random Forest Regressor\*\*

R2 score from the Random Forest model: 0.69

Mean Score for RandomForestWithParams: 0.7003

R2 score from the Random Forest modelWithParams: 0.72

Mean Score for GradientBoosting: 0.7683

\*\*Performance Evaluations for Gradient Boosting\*\*

R2 score from the Gradient Boosting model: 0.73

Mean Score for GradientBoostingWithParams: 0.7320

R2 score from the Gradient Boosting modelWithParams 0.75

Mean Score for Neural Network: 0.18

\*\*Performance Evaluations for Neural Network\*\*

R2 score from the NN model: 0.16

R2 score from the NN model Three Layers: relu 50 40 20 0.61

####################################################################

# 4.2.4.2 Analyzing Cluster 1

####################################################################

X\_C12022 = c1\_2022.loc[:, c1\_2022.columns != 'Sale Price']

y\_C12022 = c1\_2022[['Sale Price']].values.ravel()

fn\_C12022 = X\_C12022.columns

#Scaling the variables

scaler = StandardScaler()

scaler.fit(X\_C12022)

X\_C1s\_2022 = scaler.transform(X\_C12022)

# Divide the dataset into training and testing data.

# Divide the scaled dataset into training and testing data

X\_train\_C1s\_2022, X\_test\_C1s\_2022, y\_train\_C12022, y\_test\_C12022 = train\_test\_split(X\_C1s\_2022, y\_C12022, test\_size =.30,random\_state=1234)

# Divide the dataset into training and testing data.

X\_train\_C12022, X\_test\_C12022, y\_train\_C12022, y\_test\_C12022 = train\_test\_split(X\_C12022, y\_C12022, test\_size =.30,random\_state=1234)

####################################################################

# 4.2.4.2.1 Building Models With Default Parameters for Cluster 1

####################################################################

# Decision Tree Regressor

dtr\_C12022 = DecisionTreeRegressorModel(X\_test\_C12022, y\_test\_C12022, X\_train\_C12022, y\_train\_C12022)

# Random Forest Regressor

rfr\_C12022 = RandomForestRegressorModel(X\_test\_C12022, y\_test\_C12022, X\_train\_C12022, y\_train\_C12022)

# With Hyperparameters for Random Forest

RandomForestRegressorModel\_WithParams(1000, 15, X\_test\_C12022, y\_test\_C12022, X\_train\_C12022, y\_train\_C12022)

# Gradient Boosting Regression

grbr\_C12022 = GradientBoostingRegressorModel(X\_test\_C12022, y\_test\_C12022, X\_train\_C12022, y\_train\_C12022)

#With Hyperparameters for Gradient Boosting

GradientBoostingRegressorModel\_WithParams(750, 10, X\_test\_C12022, y\_test\_C12022, X\_train\_C12022, y\_train\_C12022)

# SVM Regression

svmr\_C12022 = SVMRegressorModel('linear', 10, 0.01, X\_test\_C1s\_2022, y\_test\_C12022, X\_train\_C1s\_2022, y\_train\_C12022)

# NN Regression

# Neural network with default parameters

NeuralNetworkRegressorModel(X\_test\_C1s\_2022, y\_test\_C12022, X\_train\_C1s\_2022, y\_train\_C12022)

# Optimizing the NN Model using Hyper Parameters Tuning Using Grid Search

gridSearch\_2022 = FindNeuralNetworkParam\_GridSearch(X\_test\_C1s\_2022, y\_test\_C12022, X\_train\_C1s\_2022, y\_train\_C12022)

results\_gs = pd.DataFrame(gridSearch\_2022.cv\_results\_)

# Evaluating Neural network results with grid search best estimators values

rSquare = NeuralNetworkModel\_3Layer(50, 40, 20,

'relu', X\_test\_C1s\_2022, y\_test\_C12022, X\_train\_C1s\_2022, y\_train\_C12022)

Output:

Mean Score for Decision Tree: 0.8209

\*\*Performance Evaluations for Decision Tree\*\*

R2 score from the Decision Tree model: 0.80

Mean Score for RandomForest: 0.7652

\*\*Evaluation of Errors for Random Forest Regressor\*\*

R2 score from the Random Forest model: 0.79

Mean Score for RandomForestWithParams: 0.8119

R2 score from the Random Forest modelWithParams: 0.82

Mean Score for GradientBoosting: 0.7963

\*\*Performance Evaluations for Gradient Boosting\*\*

R2 score from the Gradient Boosting model: 0.78

Mean Score for GradientBoostingWithParams: 0.8005

R2 score from the Gradient Boosting modelWithParams 0.81

Mean Score for SVM: 0.0348

\*\*Performance Evaluations for SVM Regressor\*\*

R2 score from the SVM model: 0.08

Mean Score for Neural Network: 0.1202

\*\*Performance Evaluations for Neural Network\*\*

R2 score from the NN model: 0.11

R2 score from the NN model Three Layers: relu 50 40 20 0.80

####################################################################

# 4.2.4.2.2 Feature Selection for cluster 1

####################################################################

####################################################################

# 4.2.4.2.2.1 Feature Selection Using Random Forest Regressor

####################################################################

# Find the significant features for Pierce County with their importance values.

importances = rfr\_C12022.feature\_importances\_

np.sum(importances)

plt.barh(fn\_C12022, importances)

# Draw a bar chart to see the sorted importance values with feature names.

df\_importances = pd.DataFrame(data=importances, index=fn\_C12022, columns=['importance\_value'])

df\_importances.sort\_values(by = 'importance\_value', ascending=False, inplace=True)

df\_importances

df\_importance\_top15 = df\_importances.head(11)

plt.barh(df\_importance\_top15.index, df\_importance\_top15.importance\_value)

plt.barh(df\_importances.index, df\_importances.importance\_value)

# Build a model with a subset of those features.

selector = SelectFromModel(estimator=RandomForestRegressor(), threshold= 0.015)

X\_C1\_reduced = selector.fit\_transform(X\_C12022, y\_C12022)

selector.threshold\_

selected\_TF = selector.get\_support()

print(f'\n\*\* {selected\_TF.sum()} features are selected.')

# Show those selected features.

selected\_features = []

for i,j in zip(selected\_TF, fn\_C12022):

if i: selected\_features.append(j)

print(f'Selected features for Random Forest Regressor are:\n {selected\_features}')

# Build a model using reduced number of features

X\_C1\_reduced\_train, X\_C1\_reduced\_test, y\_C1\_reduced\_train, y\_C1\_reduced\_test \

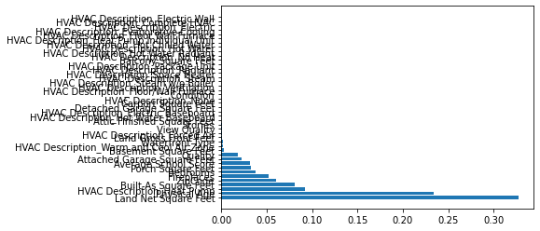
= train\_test\_split(X\_C1\_reduced, y\_C12022, test\_size =.3, random\_state=1234)

RandomForestRegressorModel(X\_C1\_reduced\_test, y\_C1\_reduced\_test, X\_C1\_reduced\_train, y\_C1\_reduced\_train)

# With Hyperparameters for Random Forest

RandomForestRegressorModel\_WithParams(1000, 15, X\_C1\_reduced\_test, y\_C1\_reduced\_test, X\_C1\_reduced\_train, y\_C1\_reduced\_train))

Output:



\*\* 11 features are selected.

Selected features for Random Forest Regressor are:

['Land Net Square Feet', 'Quality', 'Porch Square Feet', 'Attached Garage Square Feet', 'Fireplaces', 'Built-As Square Feet', 'Bedrooms', 'Physical Age', 'HVAC Description\_Heat Pump', 'ZipCode', 'Average School Score']

Mean Score for Random Forest: 0.6842

\*\*Performance Evaluation for Random Forest Regressor\*\*

R2 score from the Random Forest model: 0.68

Mean Score for RandomForestWithParams: 0.7112

R2 score from the Random Forest modelWithParams: 0.70

####################################################################

# 4.2.4.1.2.2 Feature Selection Using Gradient Boosting

####################################################################

# Find the significant features for Pierce County with their importance values.

importances = grbr\_C12022.feature\_importances\_

np.sum(importances)

plt.barh(fn\_C12022, importances)

# Draw a bar chart to see the sorted importance values with feature names.

df\_importances = pd.DataFrame(data=importances, index=fn\_C12022, columns=['importance\_value'])

df\_importances.sort\_values(by = 'importance\_value', ascending=False, inplace=True)

df\_importances

plt.barh(df\_importances.index, df\_importances.importance\_value)

# Build a model with a subset of those features.

selector = SelectFromModel(estimator=GradientBoostingRegressor(), threshold=0.015)

X\_C1\_reduced = selector.fit\_transform(X\_C12022, y\_C12022)

selector.threshold\_

selected\_TF = selector.get\_support()

print(f'\n\*\* {selected\_TF.sum()} features are selected.')

# Show those selected features.

selected\_features = []

for i,j in zip(selected\_TF, fn\_C12022):

if i: selected\_features.append(j)

print(f'Selected features for GBR are:\n {selected\_features}')

# Build a model using reduced number of features

X\_C1\_reduced\_train, X\_C1\_reduced\_test, y\_C1\_reduced\_train, y\_C1\_reduced\_test \

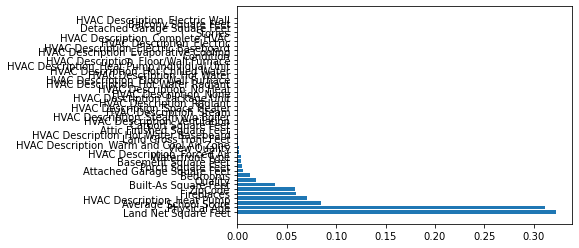
= train\_test\_split(X\_C1\_reduced, y\_C12022, test\_size =.3, random\_state=1234)

GradientBoostingRegressorModel(X\_C1\_reduced\_test, y\_C1\_reduced\_test, X\_C1\_reduced\_train, y\_C1\_reduced\_train)

#With Hyperparameters for Gradient Boosting

GradientBoostingRegressorModel\_WithParams(750, 10, X\_C1\_reduced\_test, y\_C1\_reduced\_test, X\_C1\_reduced\_train, y\_C1\_reduced\_train)

Output:



\*\* 8 features are selected.

Selected features for GBR are:

['Land Net Square Feet', 'Quality', 'Fireplaces', 'Built-As Square Feet', 'Physical Age', 'HVAC Description\_Heat Pump', 'ZipCode', 'Average School Score']

Mean Score for GradientBoosting: 0.6112

\*\*Performance Evaluations for Gradient Boosting\*\*

R2 score from the Gradient Boosting model: 0.63

Execution time for GBR: 2.29 seconds

Mean Score for GradientBoostingWithParams: 0.6453

R2 score from the Gradient Boosting modelWithParams 0.65

####################################################################

# 4.2.4.2.3 PCA For Cluster 1

####################################################################

pca\_prep = PCA().fit(X\_C1s\_2022)

pca\_prep.n\_components\_

pca\_prep.explained\_variance\_ratio\_

# Create a scree plot and find an "elbow" or an inflection point on the plot.

plt.plot(pca\_prep.explained\_variance\_ratio\_)

plt.xlabel('k number of components')

plt.ylabel('Explained variance')

plt.grid(True)

plt.show()

# From the scree plot, we choose k = 24

n\_pc = 24

pca\_C12022 = PCA(n\_components = n\_pc).fit(X\_C1s\_2022)

Xp\_C12022 = pca\_C12022.transform(X\_C1s\_2022)

print(f'After PCA, we use {pca\_C12022.n\_components\_} components.\n')

# Split the data into training and testing subsets.

Xp\_train\_C12022, Xp\_test\_C12022, yp\_train\_C12022, yp\_test\_C12022 = train\_test\_split(Xp\_C12022, y\_C12022, test\_size =.2, random\_state = 1234)

# Create random forest models using the transformed data.

rfr\_C12022 = RandomForestRegressorModel(Xp\_test\_C12022, yp\_test\_C12022, Xp\_train\_C12022, yp\_train\_C12022)

# With Hyperparameters for Random Forest

RandomForestRegressorModel\_WithParams(1000, 15, Xp\_test\_C12022, yp\_test\_C12022, Xp\_train\_C12022, yp\_train\_C12022)

# For Gradient Boosting

gbr\_C12022 = GradientBoostingRegressorModel(Xp\_test\_C12022, yp\_test\_C12022, Xp\_train\_C12022, yp\_train\_C12022)

#With Hyperparameters for Gradient Boosting

GradientBoostingRegressorModel\_WithParams(750, 10, Xp\_test\_C12022, yp\_test\_C12022, Xp\_train\_C12022, yp\_train\_C12022)

# For Neural Network

# Neural Network models using the transformed data.

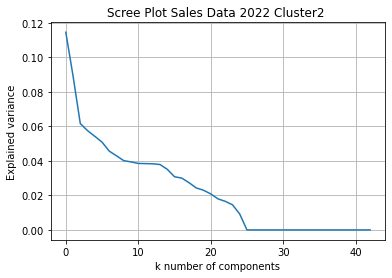
NeuralNetworkRegressorModel(Xp\_test\_C12022, yp\_test\_C12022, Xp\_train\_C12022, yp\_train\_C12022)

# Neural Network models with Optimized Parameters Values and transformed data.

rSquare = NeuralNetworkModel\_3Layer(50, 40, 20,

'relu', Xp\_test\_C12022, yp\_test\_C12022, Xp\_train\_C12022, yp\_train\_C12022)

Output:



Mean Score for RandomForest: 0.7105

\*\*Performance Evaluation for Random Forest Regressor\*\*

R2 score from the Random Forest model: 0.72

Mean Score for RandomForestWithParams: 0.7653

R2 score from the Random Forest modelWithParams: 0.77

Mean Score for GradientBoosting: 0.7002

\*\*Performance Evaluations for Gradient Boosting\*\*

R2 score from the Gradient Boosting model: 0.71

Mean Score for GradientBoostingWithParams: 0.7320

R2 score from the Gradient Boosting modelWithParams 0.74

Mean Score for Neural Network: 0.4557

\*\*Performance Evaluations for Neural Network\*\*

R2 score from the NN model: 0.38

R2 score from the NN model Three Layers: relu 50 40 20 0.63

**Statement 2 Code**

def DecisionTreeRegressorModel(X\_test, y\_test, X\_train, y\_train):

t\_start = time.time()

dtr = DecisionTreeRegressor(max\_leaf\_nodes = 500, max\_features='auto', splitter='best', random\_state = 0)

dtr\_mean\_score = np.mean(cross\_val\_score(dtr, X\_train, y\_train, cv=5))

print(f'Mean Score for Decision Tree: {dtr\_mean\_score:.4f}')

dtr.fit(X\_train, y\_train)

y\_pred\_dtr = dtr.predict(X\_test)

mae = metrics.mean\_absolute\_error(y\_test, y\_pred\_dtr)

mse = metrics.mean\_squared\_error(y\_test,y\_pred\_dtr)

rmse = np.sqrt(mse)

print('\*\*Performance Evaluations for Decision Tree\*\*')

print(f'mae: {mae:.4f}')

print(f'R2 score from the Decision Tree model: {r2\_score(y\_test, y\_pred\_dtr):.2f}')

print (f'mse: {mse:.4f}')

print(f'rmse: {rmse:.4f}')

t\_end = time.time()

print(f'Execution time for DTR: {t\_end-t\_start:.2f} seconds')

return dtr

def RandomForestRegressorModel(X\_test, y\_test, X\_train, y\_train):

t\_start = time.time()

rfrm = RandomForestRegressor(random\_state=1234)

rfrm\_mean\_score = np.mean(cross\_val\_score(rfrm, X\_train, y\_train, cv=5))

print(f'Mean Score for RandomForest: {rfrm\_mean\_score:.4f}')

rfrm.fit(X\_train, y\_train)

y\_pred\_rfrm = rfrm.predict(X\_test)

# Calculate MSE (mean squared errors), RMSE (Root Mean Squred Error),and R^2 for errors.

mse = metrics.mean\_squared\_error(y\_test,y\_pred\_rfrm)

mape = np.mean(np.abs((y\_test - y\_pred\_rfrm)/y\_test))\*100

rmse = np.sqrt(mse)

print('\n', '\*\*Performance Evaluation for Random Forest Regressor\*\*')

print (' mse: ', mse,'\n','rmse:', rmse)

print (f'mape: {mape:.4f}')

print(f'R2 score from the Random Forest model: {r2\_score(y\_test, y\_pred\_rfrm):.2f}')

t\_end = time.time()

print(f'Execution time for RFR: {t\_end-t\_start:.2f} seconds')

return rfrm

def RandomForestRegressorModel\_WithParams(nEstimator, minSamplesSplit, X\_test, y\_test, X\_train, y\_train):

t\_start = time.time()

rfrm = RandomForestRegressor(n\_estimators=nEstimator, min\_samples\_split=minSamplesSplit, random\_state=1234)

rfrm\_mean\_score = np.mean(cross\_val\_score(rfrm, X\_train, y\_train, cv=5))

print(f'Mean Score for RandomForest: {rfrm\_mean\_score:.4f}')

rfrm.fit(X\_train, y\_train)

y\_pred\_rfrm = rfrm.predict(X\_test)

# Calculate MSE (mean squared errors), RMSE (Root Mean Squred Error),and R^2 for errors.

mse = metrics.mean\_squared\_error(y\_test,y\_pred\_rfrm)

mape = np.mean(np.abs((y\_test - y\_pred\_rfrm)/y\_test))\*100

rmse = np.sqrt(mse)

print('\n', '\*\*Performance Evaluation for Random Forest Regressor\*\*')

print (' mse: ', mse,'\n','rmse:', rmse)

print (f'mape: {mape:.4f}')

print(f'R2 score from the Random Forest model: {r2\_score(y\_test, y\_pred\_rfrm):.2f}')

t\_end = time.time()

print(f'Execution time for RFR: {t\_end-t\_start:.2f} seconds')

return rfrm

def FindRandomForestRegressorParam\_GridSearch(X\_test, y\_test, X\_train, y\_train):

t\_start = time.time()

r2\_scorer = make\_scorer(r2\_score)

param\_grid = {

'n\_estimator':[500, 750, 1000, 1500],

'min\_samples\_split': [5, 10, 15, 20],

}

rfrm = RandomForestRegressor(random\_state = 1234)

grid\_src = GridSearchCV(estimator= rfrm, param\_grid = param\_grid, cv=5, scoring=r2\_scorer)

grid\_src.fit(X\_train,y\_train)

t\_end = time.time()

print('\n\n \*\*Report\*\*')

print("Best Parameters: ", grid\_src.best\_params\_)

print("Best Mean Test R-squared Score: ", grid\_src.best\_score\_)

print(f'The best estimator: {grid\_src.best\_estimator\_}')

print(f'The best parameters:\n {grid\_src.best\_params\_}')

print(f'The best score: {grid\_src.best\_score\_:.4f}')

print(f'Total run time for GridSearchCV: {(t\_end-t\_start):.2f} seconds')

# Check the details of search

return pd.DataFrame(grid\_src.cv\_results\_)

def GradientBoostingRegressorModel(X\_test, y\_test, X\_train, y\_train):

t\_start = time.time()

gbr = GradientBoostingRegressor()

gbr\_mean\_score = np.mean(cross\_val\_score(gbr, X\_train, y\_train, cv=5))

print(f'Mean Score for GradientBoosting: {gbr\_mean\_score:.4f}')

gbr.fit(X\_train, y\_train)

y\_pred\_gbr = gbr.predict(X\_test)

mse = metrics.mean\_squared\_error(y\_test,y\_pred\_gbr)

mape = np.mean(np.abs((y\_test - y\_pred\_gbr)/y\_test))\*100

print(print('\*\*Performance Evaluations for Gradient Boosting\*\*'))

print(' mse: ', mse,'\n')

print (f'mape: {mape:.4f}')

print(f'R2 score from the Gradient Boosting model: {r2\_score(y\_test, y\_pred\_gbr):.2f}')

t\_end = time.time()

print(f'Execution time for GBR: {t\_end-t\_start:.2f} seconds')

return gbr

def GradientBoostingRegressorModel\_WithParams(nEstimator, minSamplesSplit, X\_test, y\_test, X\_train, y\_train):

t\_start = time.time()

gbr = GradientBoostingRegressor(n\_estimators=nEstimator, min\_samples\_split=minSamplesSplit)

gbr\_mean\_score = np.mean(cross\_val\_score(gbr, X\_train, y\_train, cv=5))

print(f'Mean Score for GradientBoosting: {gbr\_mean\_score:.4f}')

gbr.fit(X\_train, y\_train)

y\_pred\_gbr = gbr.predict(X\_test)

mse = metrics.mean\_squared\_error(y\_test,y\_pred\_gbr)

mape = np.mean(np.abs((y\_test - y\_pred\_gbr)/y\_test))\*100

print(print('\*\*Performance Evaluations for Gradient Boosting\*\*'))

print(' mse: ', mse,'\n')

print (f'mape: {mape:.4f}')

print(f'R2 score from the Gradient Boosting model: {r2\_score(y\_test, y\_pred\_gbr):.2f}')

t\_end = time.time()

print(f'Execution time for GBR: {t\_end-t\_start:.2f} seconds')

return gbr

def FindGradientBoostingRegressorParam\_GridSearch(X\_test, y\_test, X\_train, y\_train):

t\_start = time.time()

r2\_scorer = make\_scorer(r2\_score)

param\_grid = {

'n\_estimator':[500, 750, 1000, 1500],

'min\_samples\_split': [5, 10, 15, 20],

}

gbr = GradientBoostingRegressor(random\_state = 1234)

grid\_src = GridSearchCV(estimator= gbr, param\_grid = param\_grid, cv=5, scoring=r2\_scorer)

grid\_src.fit(X\_train,y\_train)

t\_end = time.time()

print('\n\n \*\*Report\*\*')

print("Best Parameters: ", grid\_src.best\_params\_)

print("Best Mean Test R-squared Score: ", grid\_src.best\_score\_)

print(f'The best estimator: {grid\_src.best\_estimator\_}')

print(f'The best parameters:\n {grid\_src.best\_params\_}')

print(f'The best score: {grid\_src.best\_score\_:.4f}')

print(f'Total run time for GridSearchCV: {(t\_end-t\_start):.2f} seconds')

# Check the details of search

return pd.DataFrame(grid\_src.cv\_results\_)

def SVMRegressorModel(X\_test, y\_test, X\_train, y\_train):

t\_start = time.time()

svr = SVR(kernel='rbf', C=1.0, epsilon=0.2)

svr\_mean\_score = np.mean(cross\_val\_score(svr, X\_train, y\_train, cv=5))

print(f'Mean Score for SVM: {svr\_mean\_score:.4f}')

# Train the model

svr.fit(X\_train, y\_train)

y\_pred\_svr = svr.predict(X\_test)

mse = metrics.mean\_squared\_error(y\_test,y\_pred\_svr)

mape = np.mean(np.abs((y\_test - y\_pred\_svr)/y\_test))\*100

print('\*\*Performance Evaluations for SVM Regressor\*\*')

print(' mse: ', mse,'\n')

print (f'mape: {mape:.4f}')

print(f'R2 score from the SVM model: {r2\_score(y\_test, y\_pred\_svr):.2f}')

t\_end = time.time()

print(f'Execution time for SVM: {t\_end-t\_start:.2f} seconds')

return svr

def NeuralNetworkRegressorModel(X\_test, y\_test, X\_train, y\_train):

nnr = MLPRegressor(random\_state=1234)

nnr\_mean\_score = np.mean(cross\_val\_score(nnr, X\_train, y\_train, cv=5))

print(f'Mean Score for Neural Network: {nnr\_mean\_score:.4f}')

nnr.fit(X\_train,y\_train)

y\_pred\_nn = nnr.predict(X\_test)

mse = metrics.mean\_squared\_error(y\_test,y\_pred\_nn)

print('\*\*Performance Evaluations for Neural Network\*\*')

print(' mse: ', mse,'\n')

print(f'R2 score from the NN model: {r2\_score(y\_test, y\_pred\_nn):.2f}')

return nnr

def NeuralNetworkModel\_1Layer(hidden\_layer\_sizes, activationFunc, X\_test, y\_test, X\_train, y\_train):

t\_start = time.time()

nnr = MLPRegressor(hidden\_layer\_sizes=(hidden\_layer\_sizes),

max\_iter=3000, activation=activationFunc, random\_state=1234)

nnr.fit(X\_train,y\_train)

y\_pred\_nn = nnr.predict(X\_test)

mse = metrics.mean\_squared\_error(y\_test,y\_pred\_nn)

mape = np.mean(np.abs((y\_test - y\_pred\_nn)/y\_test))\*100

print('\*\*Performance Evaluations for Neural Network\*\*')

print(' mse: ', mse,'\n')

print (f'mape: {mape:.4f}')

print(f'R2 score from the NN model One Layer: {activationFunc} {hidden\_layer\_sizes} {r2\_score(y\_test, y\_pred\_nn):.2f}')

t\_end = time.time()

print(f'Execution time for NNR\_1L: {t\_end-t\_start:.2f} seconds')

def NeuralNetworkModel\_2Layer(hidden\_layer\_sizes, hidden\_layer\_2, activationFunc, X\_test, y\_test, X\_train, y\_train):

t\_start = time.time()

model = MLPRegressor(hidden\_layer\_sizes=(hidden\_layer\_sizes, hidden\_layer\_2), max\_iter=1000,

activation=activationFunc, random\_state=0)

model.fit(X\_train, y\_train)

y\_pred\_nn = model.predict(X\_test)

mse = metrics.mean\_squared\_error(y\_test,y\_pred\_nn)

mape = np.mean(np.abs((y\_test - y\_pred\_nn)/y\_test))\*100

print('\*\*Performance Evaluations for Neural Network\*\*')

print(' mse: ', mse,'\n')

print (f'mape: {mape:.4f}')

print(f'R2 score from the NN model Two Layers: {activationFunc} {hidden\_layer\_sizes} {hidden\_layer\_2} {r2\_score(y\_test, y\_pred\_nn):.2f}')

t\_end = time.time()

print(f'Execution time for NNR\_2L: {t\_end-t\_start:.2f} seconds')

def NeuralNetworkModel\_3Layer(hidden\_layer\_sizes, hidden\_layer\_2, hidden\_layer\_3, activationFunc,

X\_test, y\_test, X\_train, y\_train):

t\_start = time.time()

model = MLPRegressor(hidden\_layer\_sizes=(hidden\_layer\_sizes, hidden\_layer\_2, hidden\_layer\_3), max\_iter=1000,

activation=activationFunc, random\_state=0)

nnr\_mean\_score = np.mean(cross\_val\_score(model, X\_train, y\_train, cv=5))

print(f'Mean Score for Neural Network: {nnr\_mean\_score:.4f}')

model.fit(X\_train, y\_train)

y\_pred\_nn = model.predict(X\_test)

mse = metrics.mean\_squared\_error(y\_test,y\_pred\_nn)

print('\*\*Performance Evaluations for Neural Network\*\*')

print(' mse: ', mse,'\n')

print(f'R2 score from the NN model Two Layers: {activationFunc} {hidden\_layer\_sizes} {hidden\_layer\_2} {hidden\_layer\_3} {r2\_score(y\_test, y\_pred\_nn):.2f}')

t\_end = time.time()

print(f'Execution time for NNR\_2L: {t\_end-t\_start:.2f} seconds')

def FindNeuralNetworkParam\_GridSearch(X\_test, y\_test, X\_train, y\_train):

t\_start = time.time()

r2\_scorer = make\_scorer(r2\_score)

param\_grid = {

'hidden\_layer\_sizes':[(20), (30)],

'activation': ['relu'],

'max\_iter': [4000,5000]

}

nnm\_r = MLPRegressor()

grid\_src = GridSearchCV(estimator= nnm\_r, param\_grid = param\_grid, cv=5, scoring=r2\_scorer)

grid\_src.fit(X\_train,y\_train)

t\_end = time.time()

print('\n\n \*\*Report\*\*')

print("Best Parameters: ", grid\_src.best\_params\_)

print("Best Mean Test R-squared Score: ", grid\_src.best\_score\_)

print(f'The best estimator: {grid\_src.best\_estimator\_}')

print(f'The best parameters:\n {grid\_src.best\_params\_}')

print(f'The best score: {grid\_src.best\_score\_:.4f}')

print(f'Total run time for GridSearchCV: {(t\_end-t\_start):.2f} seconds')

# Check the details of search

return pd.DataFrame(grid\_src.cv\_results\_)

def AnalyzingDatasets(df):

## show the first five rows of the Appraisal dataset

print("First 5 rows of the dataset:")

print(df.head())

# Check the data types of each column

print("Data types:")

print(df.dtypes)

# Check the number of missing values in each column

print("Missing values:")

print(df.isnull().sum())

df.info()

# Get the summary statistics of numeric variables (appraisal dataset)

print("Summary statistics:")

print(df.describe())

# Distinct Count & frequency of nominal variables

df.nunique()

for colName in df.columns:

print("UniqueValues for " + colName)

print(df[colName].value\_counts())

#null values present in the Appraisal dataset

df.isnull().sum().sort\_values(ascending=False)

def DetermineCorrelatedVariables(df):

df\_numeric = df.select\_dtypes(include=np.number)

corrMatrix = df\_numeric.corr()

threshold = 0.7

highly\_correlated\_variables = []

# Iterate through each column in the correlation matrix

for i in range(len(df.columns)):

for j in range(i + 1, len(corrMatrix.columns)):

if abs(corrMatrix.iloc[i, j]) >= threshold:

# Append the names of highly correlated variables to the list

variable\_i = corrMatrix.columns[i]

variable\_j = corrMatrix.columns[j]

highly\_correlated\_variables.append((variable\_i, variable\_j))

return highly\_correlated\_variables

def CalculateAverageSchoolScore(dfMergedRow, schoolData):

schoolRadius = 2\*5280 # schools in 2 mile (5280 feet) radius

dfMergedX = dfMergedRow['X']

dfMergedY = dfMergedRow['Y']

zipCode = dfMergedRow['ZipCode']

filteredSchoolData = schoolData[(schoolData['X\_COORD'] <= dfMergedX + schoolRadius)

& (schoolData['X\_COORD'] >= dfMergedX - schoolRadius)

& (schoolData['Y\_COORD'] <= dfMergedY + schoolRadius)

& (schoolData['Y\_COORD'] >= dfMergedY - schoolRadius)

& (schoolData['ZIP'] == zipCode)]

if len(filteredSchoolData) == 0:

return 0

else:

return filteredSchoolData["score"].mean()

####################################################################

# 1. Reading the Datasets

####################################################################

input\_file\_appraisal = 'appraisal\_account.txt'

column\_names\_appraisal = ['Parcel Number', 'Appraisal Account Type', 'Business Name', 'Value Area ID',

'Land Economic Area', 'Buildings', 'Group Account Number',

'Land Gross Acres', 'Land Net Acres', 'Land Gross Square Feet',

'Land Net Square Feet', 'Land Gross Front Feet', 'Land Width', 'Land Depth',

'Submerged Area Square Feet', 'Appraisal Date', 'Waterfront Type',

'View Quality', 'Utility Electric', 'Utility Sewer', 'Utility Water',

'Street Type', 'Latitude', 'Longitude']

input\_file\_improvement = 'improvement.txt'

column\_names\_improvement = ['Parcel Number', 'Building ID', 'Property Type', 'Neighborhood',

'Neighborhood Extension', 'Square Feet', 'Net Square Feet',

'Percent Complete', 'Condition', 'Quality', 'Primary Occupancy Code',

'Primary Occupancy Description', 'Mobile Home Serial Number', 'Mobile Home Total Length',

'Mobile Home Make', 'Attic Finished Square Feet', 'Basement Square Feet',

'Basement Finished Square Feet', 'Carport Square Feet', 'Balcony Square Feet',

'Porch Square Feet', 'Attached Garage Square Feet', 'Detached Garage Square Feet',

'Fireplaces', 'Basement Garage Door']

input\_file\_improvementBuiltAs = 'improvement\_builtas.txt'

column\_names\_improvementBuiltAs = ['Parcel Number', 'Building ID', 'Built-As Number', 'Built-As ID',

'Built-As Description', 'Built-As Square Feet', 'HVAC', 'HVAC Description',

'Exterior', 'Interior', 'Stories', 'Story Height', 'Sprinkler Square Feet',

'Roof Cover', 'Bedrooms', 'Bathrooms', 'Units', 'Class Code', 'Class Description',

'Year Built', 'Year Remodeled', 'Adjusted Year Built', 'Physical Age',

'Built-As Length', 'Built-As Width', 'Mobile Home Mode']

input\_file\_taxAccount = 'tax\_account.txt'

column\_names\_taxAccount = ['Parcel Number', 'Account Type', 'Property Type', 'Site Address',

'Use Code', 'Use Description', 'Tax Year - Prior', 'Tax Code Area - Prior Year',

'Exemption Type - Prior Year', 'Current Use Code - Prior Year',

'Land Value - Prior Year', 'Improvement Value - Prior Year', 'Total Market Value - Prior Year',

'Taxable Value - Prior Year', 'Tax Year - Current', 'Tax Code Area - Current Year',

'Exemption Type - Current Year', 'Current Use Code - Current Year',

'Land Value - Current Year', 'Improvement Value - Current Year', 'Total Market Value - Current Year',

'Taxable Value - Current Year', 'Range', 'Township', 'Section', 'Quarter Section',

'Subdivision Name', 'Located On Parcel']

# Set the delimiter used in the input file

delimiter = '|' # Example: Tab-separated values ('\t'), Comma-separated values (','), etc.

# Read the delimited text file into a pandas DataFrame

df\_appraisal = pd.read\_csv(input\_file\_appraisal, delimiter=delimiter, encoding='ISO-8859-1', names=column\_names\_appraisal)

df\_improvement = pd.read\_csv(input\_file\_improvement, delimiter=delimiter, encoding='ISO-8859-1', names=column\_names\_improvement)

df\_improvementBuiltAs = pd.read\_csv(input\_file\_improvementBuiltAs, delimiter=delimiter, encoding='ISO-8859-1', names=column\_names\_improvementBuiltAs)

df\_taxAccount = pd.read\_csv(input\_file\_taxAccount, delimiter=delimiter, encoding='ISO-8859-1', names=column\_names\_taxAccount)

# Read external Dataset files

df\_Address\_Points = pd.read\_csv('Address\_Points.csv')

df\_school\_Data= pd.read\_csv("school\_by\_zipcode.csv")

####################################################################

# 2. Datasets Preprocessing

####################################################################

# Analyzing Datasets

print("Analyzing Appraisal Data Set")

AnalyzingDatasets(df\_appraisal)

print("Analyzing Improvement Data Set")

AnalyzingDatasets(df\_improvement)

print("Analyzing Improvement BuiltAs Data Set")

AnalyzingDatasets(df\_improvementBuiltAs)

print("Analyzing Tax Account Data Set")

AnalyzingDatasets(df\_taxAccount)

print("Analyzing AddressPoints Data Set")

AnalyzingDatasets(df\_Address\_Points)

print("Analyzing School Data Set")

AnalyzingDatasets(df\_school\_Data)

####################################################################

# 2.1 Appraisal Dataset Preprocessing

####################################################################

# Filtering the dataset to only include Residential Properties

appraisal\_dfFiltered = df\_appraisal[(df\_appraisal['Appraisal Account Type'] == 'Residential')]

appraisal\_dfFiltered.info()

appraisal\_dfFiltered.isnull().sum().sort\_values(ascending=False)

appraisal\_dfFiltered.dtypes

# Dropping the non-required columns which have more than 30% nulls, correlated and not informative

appraisal\_dfFiltered.drop(['Appraisal Account Type', 'Value Area ID', 'Business Name', 'Submerged Area Square Feet',

'Group Account Number', 'Land Economic Area', 'Latitude',

'Appraisal Date'] , axis=1, inplace=True)

appraisal\_dfFiltered['Waterfront Type'] = appraisal\_dfFiltered['Waterfront Type'].apply(lambda x: 1 if not pd.isnull(x) else 0)

appraisal\_dfFiltered['View Quality'] = appraisal\_dfFiltered['View Quality'].fillna('N/A')

viewQualityOrd = {'View Quality': {'N/A':0, 'View Lim -':1,'View Lim':2,'View Lim +':3,

'View Good' :4,'View Good +':5, 'View Avg':6,'View Avg +':7,

'View V-Good':8,'View V-Good +':9}}

appraisal\_dfFiltered = appraisal\_dfFiltered.replace(viewQualityOrd)

# Converting Parcel number and Appraisal Date Format

appraisal\_dfFiltered['Parcel Number']=(appraisal\_dfFiltered['Parcel Number']).apply(str)

numericCorrelatedVariables = DetermineCorrelatedVariables(appraisal\_dfFiltered)

print(numericCorrelatedVariables)

# Dropping additional variables based on correlation analysis

appraisal\_dfFiltered.drop(['Land Depth', 'Land Gross Acres',

'Land Gross Square Feet', 'Land Width','Land Net Acres'] , axis=1, inplace=True)

appraisal\_dfFiltered.columns

#Covert nominal variables into dummy

df\_appraisalFinal = pd.get\_dummies(appraisal\_dfFiltered, columns=['Utility Electric',

'Utility Sewer', 'Utility Water', 'Street Type'], drop\_first=True)

df\_appraisalFinal.info()

# Finding if there are duplicate parcel numbers in appraisal data set

print([item for item, count in collections.Counter(df\_appraisalFinal['Parcel Number']).items() if count > 1])

####################################################################

# 2.2 Improvement Dataset Preprocessing

####################################################################

# Filtering datset to consider only residential properties

df\_improvement = df\_improvement[(df\_improvement['Property Type'] == 'Residential')]

# Dropping the non-required columns which have majority of nulls, highly correlated and not informative

df\_improvement.drop(['Mobile Home Serial Number', 'Mobile Home Total Length',

'Mobile Home Make', 'Neighborhood', 'Neighborhood Extension', 'Primary Occupancy Code',

'Primary Occupancy Description', 'Property Type' ] , axis=1, inplace=True)

# Populating null values with 0

df\_improvement['Attic Finished Square Feet'] = df\_improvement['Attic Finished Square Feet'].fillna(0)

df\_improvement['Basement Square Feet'] = df\_improvement['Basement Square Feet'].fillna(0)

df\_improvement['Carport Square Feet'] = df\_improvement['Carport Square Feet'].fillna(0)

df\_improvement['Balcony Square Feet'] = df\_improvement['Balcony Square Feet'].fillna(0)

df\_improvement['Porch Square Feet'] = df\_improvement['Porch Square Feet'].fillna(0)

df\_improvement['Attached Garage Square Feet'] = df\_improvement['Attached Garage Square Feet'].fillna(0)

df\_improvement['Detached Garage Square Feet'] = df\_improvement['Detached Garage Square Feet'].fillna(0)

df\_improvement['Fireplaces'] = df\_improvement['Fireplaces'].fillna(0)

df\_improvement.isnull().sum().sort\_values(ascending=False)

df\_improvement.info()

# Removing rows with null values

df\_improvement = df\_improvement[~df\_improvement['Condition'].isnull()]

# Check the data types of each column

print("Data types:")

print(df\_improvement.dtypes)

# Converting Parcel number from Integer to String

df\_improvement['Parcel Number']=(df\_improvement['Parcel Number']).apply(str)

numericCorrelatedVariables = DetermineCorrelatedVariables(df\_improvement)

print(numericCorrelatedVariables)

# Converting Ordinal Variables Condition and Quality to Numeric Variables

conditionOrd = {'Condition': {'Uninhabitable':0, 'Extra Poor':1,'Very Poor':2,'Poor':3,

'low':4,'Fair':5, 'Average':6, 'Avg' : 6, 'Avg.':6,'Good':7, 'Excellent':8}}

qualityOrd = {'Quality': {'Low':0, 'Low Plus':1,'Fair':2,'Fair Plus':3,

'Average' :4,'Average Plus':5, 'Good':6,'Good Plus':7,

'Very Good':8,'Very Good Plus':9,'Excellent':10}}

df\_improvement = df\_improvement.replace(conditionOrd)

df\_improvement = df\_improvement.replace(qualityOrd)

# Dropping the non-required columns highly correlated

df\_improvement.drop(['Basement Garage Door', 'Square Feet', 'Basement Finished Square Feet',

'Net Square Feet' ] , axis=1, inplace=True)

# Finding if there are duplicate parcel numbers in improvement data set

print([item for item, count in collections.Counter(df\_improvement['Parcel Number']).items() if count > 1])

####################################################################

# 2.3 ImprovementBuiltAs Dataset Preprocessing

####################################################################

df\_improvementBuiltAs.info()

df\_improvementBuiltAs.isnull().sum().sort\_values(ascending=False)

df\_improvementBuiltAs= df\_improvementBuiltAs[df\_improvementBuiltAs['Year Built'] >= 2014]

# Converting Parcel number from int to string

df\_improvementBuiltAs['Parcel Number']=(df\_improvementBuiltAs['Parcel Number']).apply(str)

# Populating Null values for Exterior with Not Applicable

df\_improvementBuiltAs['Exterior'] = df\_improvementBuiltAs['Exterior'].fillna('Not Applicable')

# Populating Null values for Interior with Not Applicable

df\_improvementBuiltAs['Interior'] = df\_improvementBuiltAs['Interior'].fillna('Not Applicable')

# Populating Null values for Roof Cover with Not Applicable

df\_improvementBuiltAs['Roof Cover'] = df\_improvementBuiltAs['Roof Cover'].fillna('Not Applicable')

df\_improvementBuiltAs.info()

df\_improvementBuiltAs.isnull().sum().sort\_values(ascending=False)

df\_improvementBuiltAs.dtypes

# Dropping coulmns with majority of nulls, non-informative $ highly correlated

df\_improvementBuiltAs.drop(['Built-As Number', 'Built-As ID','HVAC',

'Mobile Home Mode','Class Description', 'Class Code'] , axis=1, inplace=True)

df\_improvementBuiltAs = df\_improvementBuiltAs[~df\_improvementBuiltAs['Physical Age'].isnull()]

df\_improvementBuiltAs = df\_improvementBuiltAs[~df\_improvementBuiltAs['Story Height'].isnull()]

df\_improvementBuiltAs.drop(['HVAC Description',

'Exterior', 'Interior', 'Roof Cover'], axis=1, inplace=True)

df\_improvementBuiltAs = df\_improvementBuiltAs[~df\_improvementBuiltAs['Bedrooms'].isnull()]

df\_improvementBuiltAs.drop(['Units'] , axis=1, inplace=True)

df\_improvementBuiltAs = df\_improvementBuiltAs[~df\_improvementBuiltAs['Stories'].isnull()]

df\_improvementBuiltAs = df\_improvementBuiltAs[~df\_improvementBuiltAs['Bathrooms'].isnull()]

# Analyzing highly correlated variables

numericCorrelatedVariables = DetermineCorrelatedVariables(df\_improvementBuiltAs)

print(numericCorrelatedVariables)

# Dropping coulmnswhich are non informative and highly correlated

df\_improvementBuiltAs.drop(['Sprinkler Square Feet', 'Built-As Length', 'Built-As Width', 'Year Built',

'Adjusted Year Built'] , axis=1, inplace=True)

# Finding if there are duplicate parcel numbers in improvement data set

print([item for item, count in collections.Counter(df\_improvementBuiltAs['Parcel Number']).items() if count > 1])

df\_improvementBuiltAs.isnull().sum().sort\_values(ascending=False)

####################################################################

# 2.4 TaxAccount Dataset Preprocessing

####################################################################

df\_taxAccount.info()

df\_taxAccount.isnull().sum().sort\_values(ascending=False)

print("Data types:")

print(df\_taxAccount.dtypes)

# Converting Parcel number from int to string

df\_taxAccount['Parcel Number']=(df\_taxAccount['Parcel Number']).apply(str)

# selecting the single family dwelling properties only

df\_taxAccount = df\_taxAccount[df\_taxAccount['Use Code'] == 1101]

# considering properties only whose prior year market value is > 100K

df\_taxAccount = df\_taxAccount[(df\_taxAccount['Total Market Value - Current Year'] >= 100000) &

(df\_taxAccount['Total Market Value - Current Year'] <= 30000000)]

# Analyzing Correlated Varianles

numericCorrelatedVariables = DetermineCorrelatedVariables(df\_taxAccount)

print(numericCorrelatedVariables)

# Dropping variables highly correlated, non-informative and not required for analysis

df\_taxAccount.drop(['Account Type', 'Property Type', 'Site Address',

'Use Code', 'Use Description', 'Tax Year - Prior', 'Tax Code Area - Prior Year',

'Exemption Type - Prior Year', 'Current Use Code - Prior Year',

'Land Value - Prior Year', 'Improvement Value - Prior Year',

'Taxable Value - Prior Year', 'Tax Year - Current', 'Tax Code Area - Current Year',

'Exemption Type - Current Year', 'Current Use Code - Current Year',

'Land Value - Current Year', 'Improvement Value - Current Year','Range',

'Township', 'Section', 'Quarter Section', 'Subdivision Name',

'Located On Parcel', 'Taxable Value - Current Year',

'Total Market Value - Prior Year'] , axis=1, inplace=True)

# Finding if there are duplicate parcel numbers in improvement data set

print([item for item, count in collections.Counter(df\_taxAccount['Parcel Number']).items() if count > 1])

####################################################################

# 2.5 Address Points Dataset Preprocessing

####################################################################

df\_Address\_Points.info()

# Dropping Non required columns

df\_Address\_Points.drop(['OBJECTID', 'Address', 'Mail\_Stop', 'City', 'State', 'Last\_Edited',

'Status', 'HouseNumber', 'PrefixDirectional', 'StreetName', 'StreetType',

'PostDirectional', 'Jurisdiction', 'AddressID'] , axis=1, inplace=True)

df\_Address\_Points.isnull().sum().sort\_values(ascending=False)

df\_Address\_Points = df\_Address\_Points[~df\_Address\_Points['TaxParcelNumber'].isnull()]

print([item for item, count in collections.Counter(df\_Address\_Points['TaxParcelNumber']).items() if count > 1])

# Removing rows having duplicate Parcel number

df\_Address\_Points = df\_Address\_Points.drop\_duplicates(subset=['TaxParcelNumber'], keep='last')

####################################################################

# 2.6 School Dataset Preprocessing

####################################################################

df\_school\_Data.info()

# Dropping Non required columns

df\_school\_Data.drop(['X', 'Y', 'OBJECTID', 'NAME', 'ADDRESS', 'CITY', 'DISTRICT',

'DIST\_NO', 'TYPE', 'PHONE', 'WEBSITE', 'PRS\_ID', 'GRADE'] , axis=1, inplace=True)

df\_school\_Data.isnull().sum().sort\_values(ascending=False)

####################################################################

# 3 Merging various datasets

####################################################################

df\_appraisalFinal.info()

df\_improvement.info()

df\_improvementBuiltAs.info()

df\_taxAccount.info()

# Merging tax account and appraisal datasets

df\_merged = pd.merge(df\_appraisalFinal, df\_taxAccount, left\_on='Parcel Number', right\_on='Parcel Number', how='inner')

df\_merged.info()

# Checking for Duplicate Parcel Numbers

print([item for item, count in collections.Counter(df\_merged['Parcel Number']).items() if count > 1])

# Merging improvement and improvement BuiltAs dataset

# Merging datasets using left join to consider all rows of improvement table

df\_improvement\_merged = pd.merge(df\_improvement, df\_improvementBuiltAs, left\_on=['Parcel Number', 'Building ID'],

right\_on=['Parcel Number', 'Building ID'], how='left')

# Dropping Building ID after merging as it is not required for further analysis

df\_improvement\_merged.drop(['Building ID'] , axis=1, inplace=True)

# Finding if there are duplicate parcel numbers in improvement merged data sets, for duplicate values

# select the property row which got remodelled in the last

df\_improvement\_merged= df\_improvement\_merged.sort\_values(by=['Year Remodeled'], ascending=True)

print([item for item, count in collections.Counter(df\_improvement\_merged['Parcel Number']).items() if count > 1])

df\_improvement\_merged = df\_improvement\_merged.drop\_duplicates(subset=['Parcel Number'], keep= 'last')

# Merging the improvement merged data set with the merged dataset

df\_merged = pd.merge(df\_merged, df\_improvement\_merged, left\_on='Parcel Number',

right\_on='Parcel Number', how='inner')

print([item for item, count in collections.Counter(df\_merged['Parcel Number']).items() if count > 1])

df\_merged.info()

# Finding the correlation matric for the merged dataset for highly correlated variables

numericCorrelatedVariables = DetermineCorrelatedVariables(df\_merged)

print(numericCorrelatedVariables)

# Finding if there are duplicate Parcel numbers in merged data set

print([item for item, count in collections.Counter(df\_merged['Parcel Number']).items() if count > 1])

df\_merged.isnull().sum().sort\_values(ascending=False)

df\_merged.drop(['Built-As Description', 'Buildings'] , axis=1, inplace=True)

# Merging the merged dataset with address points to get the property coordinates information

# Merged the datasets using inner join to consider only properties which are present in both

df\_merged = pd.merge(df\_merged, df\_Address\_Points, left\_on='Parcel Number', right\_on='TaxParcelNumber', how='inner')

# Merging with merged dataset with school dataset

# For each property calculate the average school score of the all the schools which are within 2 mile property radius

# and are present in same zipcode

df\_merged['Average School Score'] = df\_merged.apply(lambda row: CalculateAverageSchoolScore(row,

df\_school\_Data), axis = 1)

df\_merged.isnull().sum().sort\_values(ascending=False)

# Finding the correlation matric for the merged dataset for highly correlated variables

numericCorrelatedVariables = DetermineCorrelatedVariables(df\_merged)

print(numericCorrelatedVariables)

# Dropping Non Required Columns from the merged datasets

df\_merged.drop(['Parcel Number', 'Longitude', 'X', 'Y', 'TaxParcelNumber',

'Utility Electric\_POWER INSTALLED', 'Utility Electric\_POWER NO - COMMENT',

'Utility Sewer\_SEWER/SEPTIC INSTALLED', 'Utility Sewer\_SEWER/SEPTIC NO',

'Utility Water\_WATER INSTALLED', 'Utility Water\_WATER NO', 'Street Type\_STREET NO ROAD',

'Street Type\_STREET UNPAVED', 'Utility Sewer\_SEWER/SEPTIC NO PERC',

'Year Remodeled'] , axis=1, inplace=True)

# Removing rows with null values

df\_merged = df\_merged.dropna(axis=0)

# Analyzing the correlation matrix of the merged datasets

corrMatrixMarketValue = df\_merged.corr()

sns.heatmap(corrMatrixMarketValue, annot=True, cmap='coolwarm')

plt.title('Correlation Matrix - Numeric Variables')

plt.show()

df\_merged.info()

df\_merged.columns

####################################################################

# 4 Building Market Value Prediction Model

####################################################################

# For Final Datasets

X = df\_merged.loc[:, df\_merged.columns != 'Total Market Value - Current Year']

y = df\_merged[['Total Market Value - Current Year']].values.ravel()

fn = X.columns

#Scaling the variables

scaler = StandardScaler()

scaler.fit(X)

X\_s = scaler.transform(X)

# Divide the scaled dataset into training and testing data

X\_train\_s, X\_test\_s, y\_train, y\_test = train\_test\_split(X\_s, y, test\_size =.30,random\_state=1234)

# Divide the dataset into training and testing data.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size =.30,random\_state=1234)

# Decision Tree Regressor

dtr = DecisionTreeRegressorModel(X\_test, y\_test, X\_train, y\_train)

# Random Forest Regressor

rfr = RandomForestRegressorModel(X\_test, y\_test, X\_train, y\_train)

# Grid Seach for best Random Forest Regressor

FindRandomForestRegressorParam\_GridSearch(X\_test, y\_test, X\_train, y\_train)

# With Hyperparameters for Random Forest

RandomForestRegressorModel\_WithParams(750, 10, X\_test, y\_test, X\_train, y\_train)

# Gradient Boosting Regression

grbr = GradientBoostingRegressorModel(X\_test, y\_test, X\_train, y\_train)

# Grid Seach for best Gradient Boosting

FindGradientBoostingRegressorParam\_GridSearch(X\_test, y\_test, X\_train, y\_train)

GradientBoostingRegressorModel\_WithParams(500, 10, X\_test, y\_test, X\_train, y\_train)

# SVM Regression

svmr = SVMRegressorModel(X\_test\_s, y\_test, X\_train\_s, y\_train)

# Neural network with default parameters

NeuralNetworkRegressorModel(X\_test\_s, y\_test, X\_train\_s, y\_train)

# Analyzing Neural Network Model with various combinations

'''

for activationFuncName in ['logistic', 'tanh','relu']:

for hidden\_layer in [250,500,750]:

rSquare = NeuralNetworkModel\_1Layer(hidden\_layer, activationFuncName, X\_test\_s, y\_test,

X\_train\_s, y\_train)

# Continuing with Relu as other activation functions are giving negative values

for activationFuncName in ['relu']:

for hidden\_layer in [40, 50,60, 70, 80]:

rSquare = NeuralNetworkModel\_1Layer(hidden\_layer, activationFuncName, X\_test\_s, y\_test,

X\_train\_s, y\_train)

# Two hidden layers

for activationFuncName in ['relu']:

for hidden\_layer in [50]:

for hidden\_layer2 in [60, 70]:

rSquare = GetNeuralNetworkParams\_2Layer(hidden\_layer, hidden\_layer2,

activationFuncName, X\_test\_s, y\_test, X\_train\_s, y\_train)

for activationFuncName in ['relu']:

for hidden\_layer in [20, 30, 40]:

for hidden\_layer2 in [10, 20, 30]:

rSquare = GetNeuralNetworkParams\_2Layer(hidden\_layer, hidden\_layer2,

activationFuncName , X\_test\_s, y\_test, X\_train\_s, y\_train

'''

# Optimizing the NN Model using Hyper Parameters Tuning Using Grid Search

gridSearch\_MarketValue = FindNeuralNetworkParam\_GridSearch(X\_test\_s, y\_test, X\_train\_s, y\_train)

results\_gs = pd.DataFrame(gridSearch\_MarketValue.cv\_results\_)

# Evaluating Neural network results with grid search best estimators values

rSquare = NeuralNetworkModel\_3Layer(50, 50, 20,

'relu', X\_test\_s, y\_test, X\_train\_s, y\_train)

**Output:**

Mean Score for Decision Tree: 0.7539

\*\*Performance Evaluations for Decision Tree\*\*

R2 score from the Decision Tree model: 0.72

Mean Score for RandomForest: 0.8540

R2 score from the Random Forest model: 0.82

Mean Score for RandomForestWithParams: 0.8529

R2 score from the Random Forest modelWithParams: 0.86

Mean Score for GradientBoosting: 0.8425

R2 score from the Gradient Boosting model: 0.83

Mean Score for GradientBoostingWithParams: 0.8320

R2 score from the Gradient Boosting modelWithParams 0.85

\*\*Performance Evaluations for SVM Regressor\*\*

R2 score from the SVM model: -0.02

Mean Score for Neural Network: 0.2688

R2 score from the NN model: 0.23

\*\*Performance Evaluations for Neural Network\*\*

R2 score from the NN model: 0.80

The best estimators: MLPClassifier(hidden\_layer\_sizes=(50, 40, 20))

The best parameters for Layers:

{'activation': 'relu', 'hidden\_layer\_sizes': (50, 50, 20)}

The best score for Layers: 0.81

####################################################################

# 4.1.2 Feature Selection

####################################################################

####################################################################

# 4.1.2.1 Feature Selection Using Random Forest Regressor

####################################################################

# Find the significant features for Pierce County with their importance values.

importances = rfr.feature\_importances\_

np.sum(importances)

plt.barh(fn, importances)

# Draw a bar chart to see the sorted importance values with feature names.

df\_importances = pd.DataFrame(data=importances, index=fn, columns=['importance\_value'])

df\_importances.sort\_values(by = 'importance\_value', ascending=False, inplace=True)

df\_importances

df\_importance\_top15 = df\_importances.head(12)

plt.barh(df\_importance\_top15.index, df\_importance\_top15.importance\_value)

plt.barh(df\_importances.index, df\_importances.importance\_value)

# Build a model with a subset of those features.

selector = SelectFromModel(estimator=RandomForestRegressor(), threshold=0.015)

X\_reduced = selector.fit\_transform(X, y)

selector.threshold\_

selected\_TF = selector.get\_support()

print(f'\n\*\* {selected\_TF.sum()} features are selected.')

# Show those selected features.

selected\_features = []

for i,j in zip(selected\_TF, fn):

if i: selected\_features.append(j)

print(f'Selected features for Random Forest Regressor are:\n {selected\_features}')

# Build a model using reduced number of features

X\_reduced\_train, X\_reduced\_test, y\_reduced\_train, y\_reduced\_test \

= train\_test\_split(X\_reduced, y, test\_size =.3, random\_state=1234)

RandomForestRegressorModel(X\_reduced\_test, y\_reduced\_test, X\_reduced\_train, y\_reduced\_train)

# With Hyperparameters

RandomForestRegressorModel\_WithParams(750, 10, X\_reduced\_test, y\_reduced\_test, X\_reduced\_train, y\_reduced\_train)

Output:

A graph with blue and black text

Description automatically generated

\*\* 12 features are selected.

Selected features for Random Forest Regressor are:

['Land Net Square Feet', 'Land Gross Front Feet', 'Waterfront Type', 'View Quality', 'Quality', 'Basement Square Feet', 'Porch Square Feet', 'Attached Garage Square Feet', 'Built-As Square Feet', 'Physical Age', 'ZipCode', 'Average School Score']

Mean Score for RandomForest: 0.8455

R2 score from the Random Forest model: 0.86

Mean Score for RandomForestWithParams: 0.8621

R2 score from the Random Forest modelWithParams: 0.86

####################################################################

# 4.1.2.2 Feature Selection Using Gradient Boosting

####################################################################

# Find the significant features for Pierce County with their importance values.

importances = grbr.feature\_importances\_

np.sum(importances)

plt.barh(fn, importances)

# Draw a bar chart to see the sorted importance values with feature names.

df\_importances = pd.DataFrame(data=importances, index=fn, columns=['importance\_value'])

df\_importances.sort\_values(by = 'importance\_value', ascending=False, inplace=True)

df\_importances

plt.barh(df\_importances.index, df\_importances.importance\_value)

# Build a model with a subset of those features.

selector = SelectFromModel(estimator=GradientBoostingRegressor(), threshold=0.015)

X\_reduced = selector.fit\_transform(X, y)

selector.threshold\_

selected\_TF = selector.get\_support()

print(f'\n\*\* {selected\_TF.sum()} features are selected.')

# Show those selected features.

selected\_features = []

for i,j in zip(selected\_TF, fn):

if i: selected\_features.append(j)

print(f'Selected features for GBR are:\n {selected\_features}')

# Build a model using reduced number of features

X\_reduced\_train, X\_reduced\_test, y\_reduced\_train, y\_reduced\_test \

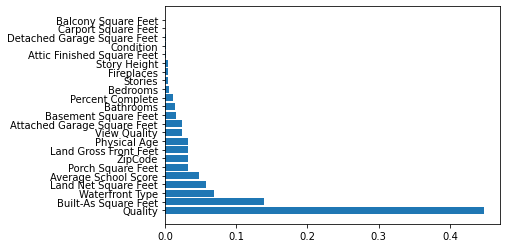
= train\_test\_split(X\_reduced, y, test\_size =.3, random\_state=1234)

GradientBoostingRegressorModel(X\_reduced\_test, y\_reduced\_test, X\_reduced\_train, y\_reduced\_train)

# With hyperparameters

GradientBoostingRegressorModel\_WithParams(500, 10, X\_reduced\_test, y\_reduced\_test, X\_reduced\_train, y\_reduced\_train)

Output:



\*\* 11 features are selected.

Selected features for GBR are:

['Land Net Square Feet', 'Land Gross Front Feet', 'Waterfront Type', 'View Quality', 'Quality', 'Basement Square Feet', 'Porch Square Feet', 'Built-As Square Feet', 'Physical Age', 'ZipCode', 'Average School Score']

Mean Score for GradientBoosting: 0.8368

\*\*Performance Evaluations for Gradient Boosting\*\*

R2 score from the Gradient Boosting model: 0.83

Mean Score for GradientBoostingWithParams: 0.8678

R2 score from the GradientBoostinglWithParams: 0.84

####################################################################

# 4.1.3 PCA

####################################################################

# Create an instance PCA and build the model using Xn.

# We start from the same number of components as the number of original features.

pca\_prep = PCA().fit(X\_s)

pca\_prep.n\_components\_

# Consider the variances as the amount of information. Drop components providing less information

# (low variances)

pca\_prep.explained\_variance\_ratio\_

# Create a scree plot and find an "elbow" or an inflection point on the plot.

plt.plot(pca\_prep.explained\_variance\_ratio\_)

plt.xlabel('k number of components')

plt.ylabel('Explained variance')

plt.title('Scree Plot Market Value Data')

plt.grid(True)

plt.show()

# From the scree plot, we choose k = 16

n\_pc = 15

pca = PCA(n\_components = n\_pc).fit(X\_s)

Xp = pca.transform(X\_s)

print(f'After PCA, we use {pca.n\_components\_} components.\n')

# Split the data into training and testing subsets.

Xp\_train, Xp\_test, yp\_train, yp\_test = train\_test\_split(Xp, y, test\_size =.3, random\_state = 1234)

# Create random forest models using the transformed data.

rfr\_pca = RandomForestRegressorModel(Xp\_test, yp\_test, Xp\_train, yp\_train)

# With Hyperparameters

RandomForestModel\_WithParams(750, 10, Xp\_test, yp\_test, Xp\_train, yp\_train)

# For Gradient Boosting

gbr\_pca = GradientBoostingRegressorModel(Xp\_test, yp\_test, Xp\_train, yp\_train)

# With Hyperparameters

GradientBoostingModel\_WithParams(750, 10, Xp\_test, yp\_test, Xp\_train, yp\_train)

# SVM Regression

svmr\_2019 = SVMRegressorModel('linear', 10, 0.01, Xp\_test, yp\_test, Xp\_train, yp\_train)

# Neural Network models using the transformed data.

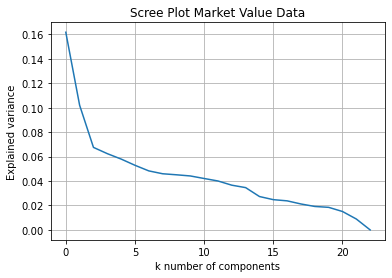
NeuralNetworkRegressorModel(Xp\_test, yp\_test, Xp\_train, yp\_train)

# Neural Network models with Optimized Parameters Values and transformed data.

rSquare = NeuralNetworkModel\_3Layer(50, 50, 20,

'relu', Xp\_test, yp\_test, Xp\_train, yp\_train)

Output:



Mean Score for RandomForest: 0.8137

R2 score from the Random Forest model: 0.81

Mean Score for RandomForestWithParams: 0.8923

R2 score from the Random Forest modelWithParams: 0.82

Mean Score for GradientBoosting: 0.7461

R2 score from the Gradient Boosting model: 0.72

Mean Score for GradientBoostingWithParams: 0.7099

R2 score from the Gradient Boosting modelWithParams 0.75

\*\*Performance Evaluations for SVM Regressor\*\*

R2 score from the SVM model: -0.03

Mean Score for Neural Network: 0.1688

R2 score from the NN model: 0.23

Mean Score for Neural Network: 0.7888

R2 score from the NN model Three Layers: relu 50 50 20 0.78

####################################################################

# 4.1.4 Clustering

####################################################################

n\_pc = 2

pca\_K = PCA(n\_components = n\_pc).fit(X\_s)

Xp\_3 = pca\_K.transform(X\_s)

# Create an instance (object) of the KMeans class with the parameters

# initialized (cluster count 2)

km = KMeans(n\_clusters=2, random\_state=1234)

# Build a model.

km = km.fit\_predict(Xp\_3)

silhouette\_avg = silhouette\_score(Xp\_3, km)

print('Silhouette Score:', silhouette\_avg)

c0 = df\_merged[km == 0]

c1 = df\_merged[km == 1]

c0.shape

c1.shape

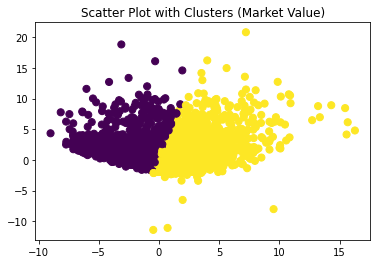
#Cluster Plot

plt.scatter(Xp\_3[:, 0], Xp\_3[:, 1], c=km, s=50, cmap='viridis')

plt.title("Scatter Plot with Clusters (Market Value)")

plt.show()

Output:



Silhouette Score: 0.48540732684901117

####################################################################

# 4.1.4.1 Analyzing Cluster 0

####################################################################

X\_C0 = c0.loc[:, c0.columns != 'Total Market Value - Current Year']

y\_C0 = c0[['Total Market Value - Current Year']].values.ravel()

fn\_C0 = X\_C0.columns

#Scaling the variables

scaler = StandardScaler()

scaler.fit(X\_C0)

X\_C0s = scaler.transform(X\_C0)

####################################################################

# 4.1.4.1.1 Building Models Wth Default Parameters for Cluster 0

####################################################################

# Divide the scaled dataset into training and testing data

X\_train\_C0s, X\_test\_C0s, y\_train\_C0, y\_test\_C0 = train\_test\_split(X\_C0s, y\_C0, test\_size =.30,random\_state=1234)

# Divide the dataset into training and testing data.

X\_train\_C0, X\_test\_C0, y\_train\_C0, y\_test\_C0 = train\_test\_split(X\_C0, y\_C0, test\_size =.30,random\_state=1234)

# Decision Tree Regressor

dtr\_C0 = DecisionTreeRegressorModel(X\_test\_C0, y\_test\_C0, X\_train\_C0, y\_train\_C0)

# Random Forest Regressor

rfr\_C0 = RandomForestRegressorModel(X\_test\_C0, y\_test\_C0, X\_train\_C0, y\_train\_C0)

# With Hyperparameters

RandomForestRegressorModel\_WithParams(750, 10, X\_test\_C0, y\_test\_C0, X\_train\_C0, y\_train\_C0)

# Gradient Boosting Regression

grbr\_C0 = GradientBoostingRegressorModel(X\_test\_C0, y\_test\_C0, X\_train\_C0, y\_train\_C0)

GradientBoostingRegressorModel\_WithParams(500, 10, X\_test\_C0, y\_test\_C0, X\_train\_C0, y\_train\_C0)

# SVM Regression

svmr\_C0 = SVMRegressorModel(X\_test\_C0s, y\_test\_C0, X\_train\_C0s, y\_train\_C0)

# NN Regression

# Neural network with default parameters

NeuralNetworkRegressorModel(X\_test\_C0s, y\_test\_C0, X\_train\_C0s, y\_train\_C0)

# Optimizing the NN Model using Hyper Parameters Tuning Using Grid Search

gridSearch = FindNeuralNetworkParam\_GridSearch(X\_test\_C0s, y\_test\_C0, X\_train\_C0s, y\_train\_C0)

results\_gs = pd.DataFrame(gridSearch.cv\_results\_)

# Evaluating Neural network results with grid search best estimators values

rSquare = NeuralNetworkModel\_3Layer(50, 50, 20,

'relu', X\_test\_C0s, y\_test\_C0, X\_train\_C0s, y\_train\_C0)

Output:

Mean Score for DecisionTree: 0.8137

R2 score from the DecisionTree model: 0.81

Mean Score for RandomForest: 0.8137

R2 score from the Random Forest model: 0.81

Mean Score for RandomForestWithParams: 0.8923

R2 score from the Random Forest modelWithParams: 0.90

Mean Score for GradientBoosting: 0.8431

R2 score from the Gradient Boosting model: 0.83

Mean Score for GradientBoostingWithParams: 0.8566

R2 score from the Gradient Boosting modelWithParams 0.87

\*\*Performance Evaluations for SVM Regressor\*\*

R2 score from the SVM model: 0.05

Mean Score for Neural Network: 0.4398

R2 score from the NN model: 0.53

Mean Score for Neural Network: 0.7888

R2 score from the NN model Three Layers: relu 50 50 20 0.81

####################################################################

# 4.1.4.1.2 Feature Selection for cluster 0

####################################################################

####################################################################

# 4.1.4.1.2.1 Feature Selection Using Random Forest Regressor

####################################################################

# Find the significant features for Pierce County with their importance values.

importances = rfr\_C0.feature\_importances\_

np.sum(importances)

plt.barh(fn\_C0, importances)

# Draw a bar chart to see the sorted importance values with feature names.

df\_importances = pd.DataFrame(data=importances, index=fn\_C0, columns=['importance\_value'])

df\_importances.sort\_values(by = 'importance\_value', ascending=False, inplace=True)

df\_importances

plt.barh(df\_importances.index, df\_importances.importance\_value)

# Build a model with a subset of those features.

selector = SelectFromModel(estimator=RandomForestRegressor(), threshold= 0.015)

X\_C0\_reduced = selector.fit\_transform(X\_C0, y\_C0)

selector.threshold\_

selected\_TF = selector.get\_support()

print(f'\n\*\* {selected\_TF.sum()} features are selected.')

# Show those selected features.

selected\_features = []

for i,j in zip(selected\_TF, fn\_C0):

if i: selected\_features.append(j)

print(f'Selected features for Random Forest Regressor are:\n {selected\_features}')

# Build a model using reduced number of features

X\_C0\_reduced\_train, X\_C0\_reduced\_test, y\_C0\_reduced\_train, y\_C0\_reduced\_test \

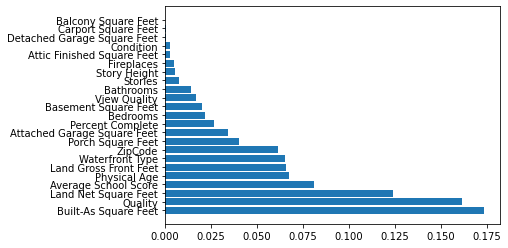
= train\_test\_split(X\_C0\_reduced, y\_C0, test\_size =.3, random\_state=1234)

RandomForestRegressorModel(X\_C0\_reduced\_test, y\_C0\_reduced\_test, X\_C0\_reduced\_train, y\_C0\_reduced\_train)

# With Hyperparameters

RandomForestRegressorModel\_WithParams(750, 10, X\_C0\_reduced\_test, y\_C0\_reduced\_test, X\_C0\_reduced\_train, y\_C0\_reduced\_train)

Output:



\*\* 12 features are selected.

Selected features for Random Forest Regressor are:

['Land Net Square Feet', 'Land Gross Front Feet', 'Waterfront Type', 'View Quality', 'Quality', 'Basement Square Feet', 'Porch Square Feet', 'Attached Garage Square Feet', 'Built-As Square Feet', 'Physical Age', 'ZipCode', 'Average School Score']

Mean Score for RandomForest: 0.6879

R2 score from the Random Forest model: 0.69

Mean Score for RandomForestWithParams: 0.7388

R2 score from the Random Forest modelWithParams: 0.70

####################################################################

# 4.1.4.1.2.2 Feature Selection Using Gradient Boosting

####################################################################

# Find the significant features for Pierce County with their importance values.

importances = grbr\_C0.feature\_importances\_

np.sum(importances)

plt.barh(fn\_C0, importances)

# Draw a bar chart to see the sorted importance values with feature names.

df\_importances = pd.DataFrame(data=importances, index=fn\_C0, columns=['importance\_value'])

df\_importances.sort\_values(by = 'importance\_value', ascending=False, inplace=True)

df\_importances

plt.barh(df\_importances.index, df\_importances.importance\_value)

# Build a model with a subset of those features.

selector = SelectFromModel(estimator=GradientBoostingRegressor(), threshold=0.015)

X\_C1\_reduced = selector.fit\_transform(X\_C0, y\_C0)

selector.threshold\_

selected\_TF = selector.get\_support()

print(f'\n\*\* {selected\_TF.sum()} features are selected.')

# Show those selected features.

selected\_features = []

for i,j in zip(selected\_TF, fn\_C0):

if i: selected\_features.append(j)

print(f'Selected features for GBR are:\n {selected\_features}')

# Build a model using reduced number of features

X\_C0\_reduced\_train, X\_C0\_reduced\_test, y\_C0\_reduced\_train, y\_C0\_reduced\_test \

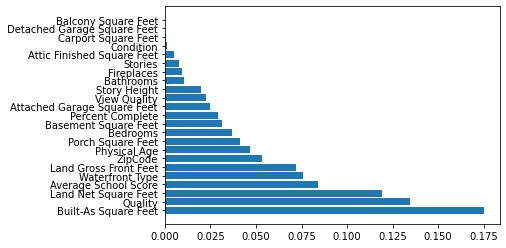
= train\_test\_split(X\_C0\_reduced, y\_C0, test\_size =.3, random\_state=1234)

GradientBoostingRegressorModel(X\_C0\_reduced\_test, y\_C0\_reduced\_test, X\_C0\_reduced\_train, y\_C0\_reduced\_train)

#With hyperparameters

GradientBoostingRegressorModel\_WithParams(500, 10, X\_C0\_reduced\_test, y\_C0\_reduced\_test, X\_C0\_reduced\_train, y\_C0\_reduced\_train)

Output:



Selected features for GBR are:

['Land Net Square Feet', 'Land Gross Front Feet', 'Waterfront Type', 'View Quality', 'Quality', 'Basement Square Feet', 'Porch Square Feet', 'Built-As Square Feet', 'Physical Age', 'ZipCode', 'Average School Score']

Mean Score for GradientBoosting: 0.8853

R2 score from the GradientBoosting model: 0.82

Mean Score for GradientBoostingWithParams: 0.8972

R2 score from the GradientBoostingmodelWithParams: 0.87

####################################################################

# 4.1.4.1.3 PCA For Cluster 0

####################################################################

pca\_prep = PCA().fit(X\_C0s)

pca\_prep.n\_components\_

# Consider the variances as the amount of information. Drop components providing less information

# (low variances)

pca\_prep.explained\_variance\_ratio\_

# Create a scree plot and find an "elbow" or an inflection point on the plot.

plt.plot(pca\_prep.explained\_variance\_ratio\_)

plt.xlabel('k number of components')

plt.ylabel('Explained variance')

plt.title('Scree Plot Cluster1: Market Value')

plt.grid(True)

plt.show()

# From the scree plot, we choose k = 20

n\_pc = 20

pca\_C0 = PCA(n\_components = n\_pc).fit(X\_C0s)

Xp\_C0 = pca\_C0.transform(X\_C0s)

print(f'After PCA, we use {pca\_C0.n\_components\_} components.\n')

# Split the data into training and testing subsets.

Xp\_train\_C0, Xp\_test\_C0, yp\_train\_C0, yp\_test\_C0 = train\_test\_split(Xp\_C0, y\_C0, test\_size =.2,

random\_state = 1234)

# Create random forest models using the transformed data.

rfr\_C0 = RandomForestRegressorModel(Xp\_test\_C0, yp\_test\_C0, Xp\_train\_C0, yp\_train\_C0)

# With Hyperparameters

RandomForestRegressorModel\_WithParams(750, 10, Xp\_test\_C0, yp\_test\_C0, Xp\_train\_C0, yp\_train\_C0)

# For Gradient Boosting

gbr\_C0 = GradientBoostingRegressorModel(Xp\_test\_C0, yp\_test\_C0, Xp\_train\_C0, yp\_train\_C0)

# With Hyperparameters

GradientBoostingRegressorModel\_WithParams(500, 10, Xp\_test\_C0, yp\_test\_C0, Xp\_train\_C0, yp\_train\_C0)

# SVM Regression

svmr\_C02019 = SVMRegressorModel('linear', 10, 0.01, Xp\_test\_C0, yp\_test\_C0, Xp\_train\_C0, yp\_train\_C0)

# For Neural Network

# Neural Network models using the transformed data.

NeuralNetworkRegressorModel(Xp\_test\_C0, yp\_test\_C0, Xp\_train\_C0, yp\_train\_C0)

# Neural Network models with Optimized Parameters Values and transformed data.

rSquare = NeuralNetworkModel\_3Layer(50, 50, 20,

'relu', Xp\_test\_C0, yp\_test\_C0, Xp\_train\_C0, yp\_train\_C0)

Output:



Mean Score for RandomForest: 0.8073

R2 score from the Random Forest model: 0.81

Mean Score for RandomForestWithParams: 0.8543

R2 score from the Random Forest modelWithParams: 0.87

Mean Score for GradientBoosting: 0.8231

R2 score from the Gradient Boosting model: 0.83

Mean Score for GradientBoostingWithParams: 0.7985

R2 score from the Gradient Boosting modelWithParams 0.82

\*\*Performance Evaluations for SVM Regressor\*\*

R2 score from the SVM model: 0.06

Mean Score for Neural Network: 0.8345

R2 score from the NN model: 0.80

Mean Score for Neural Network: 0.8103

R2 score from the NN model Three Layers: relu 50 50 20 0.84

####################################################################

# 4.1.4.2 Analyzing Cluster 1

####################################################################

X\_C1 = c1.loc[:, c1.columns != 'Total Market Value - Current Year']

y\_C1 = c1[['Total Market Value - Current Year']].values.ravel()

fn\_C1 = X\_C1.columns

#Scaling the variables

scaler = StandardScaler()

scaler.fit(X\_C1)

X\_C1s = scaler.transform(X\_C1)

# Divide the dataset into training and testing data.

# Divide the scaled dataset into training and testing data

X\_train\_C1s, X\_test\_C1s, y\_train\_C1, y\_test\_C1 = train\_test\_split(X\_C1s, y\_C1, test\_size =.30,random\_state=1234)

# Divide the dataset into training and testing data.

X\_train\_C1, X\_test\_C1, y\_train\_C1, y\_test\_C1 = train\_test\_split(X\_C1, y\_C1, test\_size =.30,random\_state=1234)

####################################################################

# 4.1.4.2.1 Building Models Wth Default Parameters for Cluster 1

####################################################################

# Decision Tree Regressor

dtr\_C1 = DecisionTreeRegressorModel(X\_test\_C1, y\_test\_C1, X\_train\_C1, y\_train\_C1)

# Random Forest Regressor

rfr\_C1 = RandomForestRegressorModel(X\_test\_C1, y\_test\_C1, X\_train\_C1, y\_train\_C1)

# With Hyperparameters

RandomForestRegressorModel\_WithParams(750, 10, X\_test\_C1, y\_test\_C1, X\_train\_C1, y\_train\_C1)

# Gradient Boosting Regression

grbr\_C1 = GradientBoostingRegressorModel(X\_test\_C1, y\_test\_C1, X\_train\_C1, y\_train\_C1)

# With Hyperparameters

GradientBoostingRegressorModel\_WithParams(500, 10, X\_test\_C1, y\_test\_C1, X\_train\_C1, y\_train\_C1)

# SVM Regression

svmr\_C1 = SVMRegressorModel(X\_test\_C1s, y\_test\_C1, X\_train\_C1s, y\_train\_C1)

# NN Regression

# Neural network with default parameters

NeuralNetworkRegressorModel(X\_test\_C1s, y\_test\_C1, X\_train\_C1s, y\_train\_C1)

# Optimizing the NN Model using Hyper Parameters Tuning Using Grid Search

gridSearch\_2019 = FindNeuralNetworkParam\_GridSearch(X\_test\_C1s, y\_test\_C1, X\_train\_C1s, y\_train\_C1)

results\_gs = pd.DataFrame(gridSearch\_2019.cv\_results\_)

# Evaluating Neural network results with grid search best estimators values

rSquare = NeuralNetworkModel\_3Layer(50, 50, 20,

'relu', X\_test\_C1s, y\_test\_C1, X\_train\_C1s, y\_train\_C1)

Output:

Mean Score for RandomForest: 0.7709

R2 score from the Random Forest model: 0.78

Mean Score for RandomForestWithParams: 0.7983

R2 score from the Random Forest modelWithParams: 0.82

Mean Score for GradientBoosting: 0.7786

R2 score from the Gradient Boosting model: 0.78

Mean Score for GradientBoostingWithParams: 0.7654

R2 score from the Gradient Boosting modelWithParams 0.79

\*\*Performance Evaluations for SVM Regressor\*\*

R2 score from the SVM model: 0.02

Mean Score for Neural Network: 0.1809

R2 score from the NN model: 0.17

Mean Score for Neural Network: 0.4987

R2 score from the NN model Three Layers: relu 50 50 20 0.51

####################################################################

# 4.2.4.2.2 Feature Selection for cluster 1

####################################################################

####################################################################

# 4.2.4.2.2.1 Feature Selection Using Random Forest Regressor

####################################################################

# Find the significant features for Pierce County with their importance values.

importances = rfr\_C1.feature\_importances\_

np.sum(importances)

plt.barh(fn\_C1, importances)

# Draw a bar chart to see the sorted importance values with feature names.

df\_importances = pd.DataFrame(data=importances, index=fn\_C1, columns=['importance\_value'])

df\_importances.sort\_values(by = 'importance\_value', ascending=False, inplace=True)

df\_importances

df\_importance\_top15 = df\_importances.head(11)

plt.barh(df\_importance\_top15.index, df\_importance\_top15.importance\_value)

plt.barh(df\_importances.index, df\_importances.importance\_value)

# Build a model with a subset of those features.

selector = SelectFromModel(estimator=RandomForestRegressor(), threshold= 0.015)

X\_C1\_reduced = selector.fit\_transform(X\_C1, y\_C1)

selector.threshold\_

selected\_TF = selector.get\_support()

print(f'\n\*\* {selected\_TF.sum()} features are selected.')

# Show those selected features.

selected\_features = []

for i,j in zip(selected\_TF, fn\_C1):

if i: selected\_features.append(j)

print(f'Selected features for Random Forest Regressor are:\n {selected\_features}')

# Build a model using reduced number of features

X\_C1\_reduced\_train, X\_C1\_reduced\_test, y\_C1\_reduced\_train, y\_C1\_reduced\_test \

= train\_test\_split(X\_C1\_reduced, y\_C1, test\_size =.3, random\_state=1234)

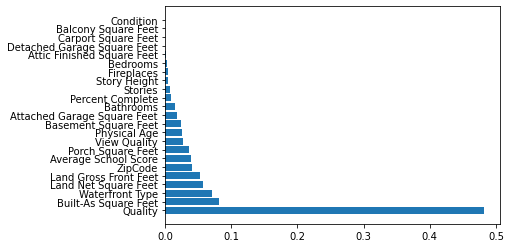
# With Default

RandomForestRegressorModel(X\_C1\_reduced\_test, y\_C1\_reduced\_test, X\_C1\_reduced\_train, y\_C1\_reduced\_train)

# With Hyperparameters

RandomForestRegressorModel\_WithParams(750, 10, X\_test, y\_test, X\_train, y\_train)

Output:



\*\* 14 features are selected.

Selected features for Random Forest Regressor are:

['Land Net Square Feet', 'Land Gross Front Feet', 'Waterfront Type', 'Percent Complete', 'Quality', 'Basement Square Feet', 'Porch Square Feet', 'Attached Garage Square Feet', 'Built-As Square Feet', 'Bedrooms', 'Bathrooms', 'Physical Age', 'ZipCode', 'Average School Score']

Mean Score for RandomForest: 0.6476

R2 score from the Random Forest model: 0.62

Mean Score for RandomForestWithParams: 0.6632

R2 score from the Random Forest modelWithParams: 0.67

####################################################################

# 4.2.4.1.2.2 Feature Selection Using Gradient Boosting

####################################################################

# Find the significant features for Pierce County with their importance values.

importances = grbr\_C1.feature\_importances\_

np.sum(importances)

plt.barh(fn\_C1, importances)

# Draw a bar chart to see the sorted importance values with feature names.

df\_importances = pd.DataFrame(data=importances, index=fn\_C1, columns=['importance\_value'])

df\_importances.sort\_values(by = 'importance\_value', ascending=False, inplace=True)

df\_importances

plt.barh(df\_importances.index, df\_importances.importance\_value)

# Build a model with a subset of those features.

selector = SelectFromModel(estimator=GradientBoostingRegressor(), threshold=0.015)

X\_C1\_reduced = selector.fit\_transform(X\_C1, y\_C1)

selector.threshold\_

selected\_TF = selector.get\_support()

print(f'\n\*\* {selected\_TF.sum()} features are selected.')

# Show those selected features.

selected\_features = []

for i,j in zip(selected\_TF, fn\_C1):

if i: selected\_features.append(j)

print(f'Selected features for GBR are:\n {selected\_features}')

# Build a model using reduced number of features

X\_C1\_reduced\_train, X\_C1\_reduced\_test, y\_C1\_reduced\_train, y\_C1\_reduced\_test \

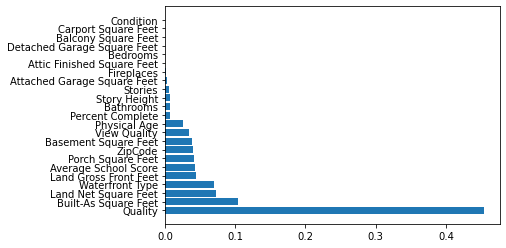
= train\_test\_split(X\_C1\_reduced, y\_C1, test\_size =.3, random\_state=1234)

GradientBoostingRegressorModel(X\_C1\_reduced\_test, y\_C1\_reduced\_test, X\_C1\_reduced\_train, y\_C1\_reduced\_train)

# With Hyperparameters

GradientBoostingRegressorModel\_WithParams(500, 10, X\_C1\_reduced\_test, y\_C1\_reduced\_test, X\_C1\_reduced\_train, y\_C1\_reduced\_train)

Output:



\*\* 15 features are selected.

Selected features for GBR are:

['Land Net Square Feet', 'Land Gross Front Feet', 'Waterfront Type', 'View Quality', 'Percent Complete', 'Quality', 'Basement Square Feet', 'Porch Square Feet', 'Attached Garage Square Feet', 'Built-As Square Feet', 'Bedrooms', 'Bathrooms', 'Physical Age', 'ZipCode', 'Average School Score']

Mean Score for RandomForest: 0.7649

R2 score from the Random Forest model: 0.79

Mean Score for RandomForestWithParams: 0.8369

R2 score from the Random Forest modelWithParams: 0.86

####################################################################

# 4.1.4.2.3 PCA For Cluster 1

####################################################################

pca\_prep = PCA().fit(X\_C1s)

pca\_prep.n\_components\_

# Consider the variances as the amount of information. Drop components providing less information

# (low variances)

pca\_prep.explained\_variance\_ratio\_

# Create a scree plot and find an "elbow" or an inflection point on the plot.

plt.plot(pca\_prep.explained\_variance\_ratio\_)

plt.xlabel('k number of components')

plt.ylabel('Explained variance')

plt.grid(True)

plt.show()

# From the scree plot, we choose k = 15

n\_pc = 16

pca\_C1 = PCA(n\_components = n\_pc).fit(X\_C1s)

Xp\_C1 = pca\_C1.transform(X\_C1s)

print(f'After PCA, we use {pca\_C1.n\_components\_} components.\n')

# Split the data into training and testing subsets.

Xp\_train\_C1, Xp\_test\_C1, yp\_train\_C1, yp\_test\_C1 = train\_test\_split(Xp\_C1, y\_C1, test\_size =.2,

random\_state = 1234)

# Create random forest models using the transformed data.

rfr\_C1 = RandomForestRegressorModel(Xp\_test\_C1, yp\_test\_C1, Xp\_train\_C1, yp\_train\_C1)

# With Hyperparameters

RandomForestRegressorModel\_WithParams(750, 10, Xp\_test\_C1, yp\_test\_C1, Xp\_train\_C1, yp\_train\_C1)

# For Gradient Boosting

gbr\_C1 = GradientBoostingRegressorModel(Xp\_test\_C1, yp\_test\_C1, Xp\_train\_C1, yp\_train\_C1)

# With Hyperparameters

GradientBoostingRegressorModel\_WithParams(500, 10, Xp\_test\_C1, yp\_test\_C1, Xp\_train\_C1, yp\_train\_C1)

# For Neural Network

# Neural Network models using the transformed data.

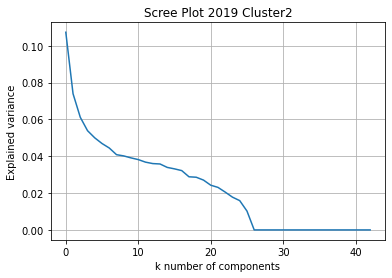
NeuralNetworkRegressorModel(Xp\_test\_C1, yp\_test\_C1, Xp\_train\_C1, yp\_train\_C1)

# Neural Network models with Optimized Parameters Values and transformed data.

rSquare = NeuralNetworkModel\_3Layer(50, 50, 20,

'relu', Xp\_test\_C1, yp\_test\_C1, Xp\_train\_C1, yp\_train\_C1)

Output:



Mean Score for RandomForest: 0.6835

R2 score from the Random Forest model: 0.70

Mean Score for RandomForestWithParams: 0.7109

R2 score from the Random Forest modelWithParams: 0.72

Mean Score for GradientBoosting: 0.6548

R2 score from the Gradient Boosting model: 0.66

Mean Score for GradientBoostingWithParams: 0.7154

R2 score from the Gradient Boosting modelWithParams 0.73

\*\*Performance Evaluations for SVM Regressor\*\*

R2 score from the SVM model: 0.06

Mean Score for Neural Network: 0.1265

R2 score from the NN model: 0.11

Mean Score for Neural Network: 0.5986

R2 score from the NN model Three Layers: relu 50 50 20 0.68

Table #. Highly Correlated Attributes

|  |  |  |
| --- | --- | --- |
| table | attributes | corr |
| appraisal | Land Gross Acres & Land Net Acres | 0.997 |
| Land Gross Acres & Land Gross Square Feet | 1.000 |
| Land Gross Acres & Land Net Square Feet | 0.997 |
| Land Net Acres & Land Gross Square Feet | 0.997 |
| Land Net Acres & Land Net Square Feet | 1.000 |
| Land Gross Square Feet & Land Net Square Feet | 0.997 |
| improvement | Mobile Home Total Length & Detached Garage Square Feet | - 1.000 |
| Basement Square Feet & Basement Finished Square Feet | 0.752 |
| improvementBuiltAs | Built-As Square Feet & Sprinkler Square Feet | 0.915 |
| Bathrooms & Bedrooms | 0.855 |
| Year Built & Physical Age | - 0.762 |
| Built-As Width & Built-As Length | 0.949 |
| sale | Parcel Count & Sale Price | 0.752 |
| tax | Tax Code Area-Prior Year & Tax Code Area-Current Year | 1.000 |
| Land Value-Prior Year & Total Market Value-Prior Year | 0.782 |
| Land Value-Prior Year & Land Value-Current Year | 0.988 |
| Land Value-Prior Year & Total Market Value-Current Year | 0.757 |
| Improvement Value-Prior Year & Total Market Value-Prior Year | 0.952 |
| Improvement Value-Prior Year & Taxable Value-Prior Year | 0.789 |
| Improvement Value-Prior Year & Improvement Value-Current Year | 0.993 |
| Improvement Value-Prior Year & Total Market Value-Current Year | 0.941 |
| Improvement Value-Prior Year & Taxable Value-Current Year | 0.712 |
| Total Market Value-Prior Year & Taxable Value-Prior Year | 0.809 |
| Total Market Value-Prior Year & Land Value-Current Year | 0.775 |
| Total Market Value-Prior Year & Improvement Value-Current Year | 0.938 |
| Total Market Value-Prior Year & Total Market Value-Current Year | 0.995 |
| Total Market Value-Prior Year & Taxable Value-Current Year | 0.743 |
| Taxable Value-Prior Year & Total Market Value-Current Year | 0.734 |
| Taxable Value-Prior Year & Taxable Value-Current Year | 0.993 |
| Land Value-Current Year & Total Market Value-Current Year | 0.772 |
| Improvement Value-Current Year & Total Market Value-Current Year | 0.947 |
| Improvement Value-Current Year & Taxable Value-Current Year | 0.706 |
| Total Market Value-Current Year & Improvement Value-Current Year | 0.947 |
| Total Market Value-Current Year & Taxable Value-Current Year | 0.740 |
| Taxable Value-Current Year & Improvement Value-Current Year | 0.706 |

Table#. Null value

|  |  |  |  |
| --- | --- | --- | --- |
| table | Columns with more than 30% of missing value | Columns with 10~30% of missing value | Columns with less than 10% of missing value |
| appraisal\_account | Business\_Name 95.92  Group\_Account\_Number 94.66  Submerged\_Area\_Square\_Feet 83.91 |  | Land\_Economic\_Area 0.01 %  Buildings 0.03 %  Land\_Gross\_Acres 0.0 %  Land\_Net\_Acres 4.97 %  Land\_Net\_Square\_Feet 4.97 %  Land\_Width 3.71 %  Appraisal\_Date 0.16 %Latitude 0.35 %  Longitude 0.35 % |
| improvement |  |  | Neighborhood 0.01 %  Neighborhood\_Extension 0.01 %  Net\_Square\_Feet 0.03 %  Condition 0.05 %  Quality 0.05 % |
| improvement\_builtas | HVAC 100.0  Class\_Code 87.23  Class\_Description 87.23  Mobile\_Home\_Model 93.61 | Exterior  Interior  Roof\_Cover | HVAC\_Description 5.21 %  Stories 0.39 %  Sprinkler\_Square\_Feet 0.05 %  Bedrooms 8.05 %  Bathrooms 8.02 %  Units 1.59 %  Built-As\_Length 2.24 %  Built-As\_Width 2.24 % |
| sale | - | - | Grantor 2.02 %  Grantee 1.97 % |
| Tax\_account | - | - | Property\_Type 3.03 %  Use\_Code 0.19 %  Use\_Description 0.19 %Tax\_Code\_Area-Prior\_Year 0.13 %  Land\_Value-Prior\_Year 3.45 %  Improvement\_Value-Prior\_Year 3.09 %  Total\_Market\_Value-Prior\_Year 0.14 %  Taxable\_Value-Prior\_Year 0.14 %  Tax\_Code\_Area-Current\_Year 3.34 %  Land\_Value-Current\_Year 3.45 %  Improvement\_Value-Current\_Year 3.45 %  Total\_Market\_Value-Current\_Year 3.44 %  Taxable\_Value-Current\_Year 3.44 %  Range 6.82 %  Township 6.82 %  Section 6.82 %  Quarter\_Section 6.82 % |
| School score | - | - |  |

Table #. nominal attributes

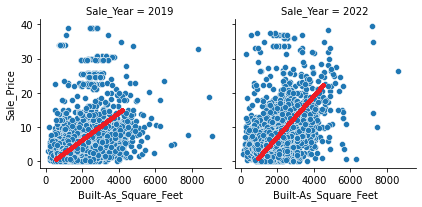
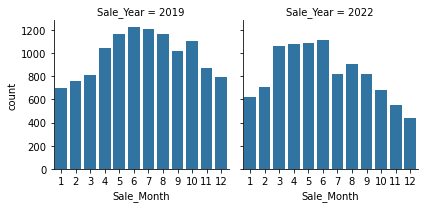
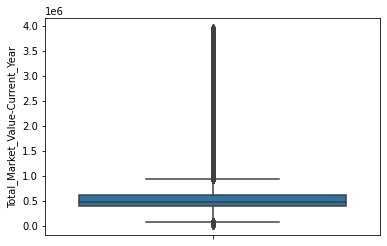
|  |  |
| --- | --- |
| Table | Nominal attributes |
| Appraisal account | Appraisal\_Account\_Type Value\_Area\_ID  Utility\_Water  Utility\_Electric  Utility\_Sewer  Street\_Type  Land\_Economic\_Area |
| improvement | Property\_Type  Neighborhood  Neighborhood\_Extension  Primary\_Occupancy\_Code  Primary\_Occupancy\_Description |
| improvement\_builtas | Built-As\_Number  Interior  Exterior  Roof\_Cover  Parcel\_Number  Building\_ID |
| sale | Deed\_Type  Appraisal\_Account\_Type  Grantor  Grantee |
| tax\_account | Account\_Type  Property\_Type  Site\_Address  Use\_Code  Tax\_Code\_Area-Prior\_Year  Tax\_Code\_Area-Current\_Year  Range  Township  Section  Quarter\_Section |

Table#. Ordinal Attributes

|  |  |
| --- | --- |
| Table | Ordinal attributes |
|  | View Quality  Quality  Condition |

Table#. Date attributes

|  |  |
| --- | --- |
| Table | Date type attributes |
| Appraisal\_account | Appraisal\_Date |
| Sale | Sale\_Date |
| Improvement\_builtas | Year\_Built  Year\_Remodeled  Adjusted\_Year\_Built |
| Tax\_account | Tax\_Year-Prior  Tax\_Year-Current |



**Variables for Sales 2019 and 2022 Data:** Land Net Square Feet', 'Land Gross Front Feet', 'Waterfront Type’, 'View Quality', 'Sale Price', 'Condition', 'Quality’, 'Attic Finished Square Feet', 'Basement Square Feet’, 'Carport Square Feet', 'Balcony Square Feet', 'Porch Square Feet’, 'Attached Garage Square Feet', 'Detached Garage Square Feet’, 'Fireplaces', 'Built-As Square Feet', 'Stories', 'Bedrooms’, 'Physical Age', 'HVAC Description, 'ZipCode’, 'Average School Score'.

**Variables for Market Value prediction Data:** 'Land Net Square Feet', 'Land Gross Front Feet', 'Waterfront Type, 'View Quality', 'Total Market Value - Current Year', 'Percent Complete', 'Condition', 'Quality', 'Attic Finished Square Feet', 'Basement Square Feet', 'Carport Square Feet', 'Balcony Square Feet', 'Porch Square Feet', 'Attached Garage Square Feet', 'Detached Garage Square Feet', 'Fireplaces', 'Built-As Square Feet', 'Stories', 'Story Height', 'Bedrooms', 'Bathrooms', 'Physical Age', 'ZipCode', 'Average School Score'