MACHINE LEARNING APPLICATION
LAND OWNER'S WILLINGNESS TO SELL

Amanda G Foess
Stanford University
Bachelor of Science in Computer Science
CS 229: Machine Learning Theory
agfoess@stanford.edu

December 5, 2024

ABSTRACT

The purpose of this model is to understand the factors influencing an owner's willingness to sell their land in the greater Charlotte area. The target variable for this analysis is the the distance between the owner's residence, and the parcel of the land they own. By identifying trends and patterns in the quantitative and categorical properties of the parcel of land, the analysis aims to provide insights into some of the main cues experts in this industry look for when investing, which could inform land development and real estate investors about high-potential investments in specified locations.

1 Data set

1.1 Data Collection and Sources

Primary Data Source: The dataset utilized in this project was sourced through a subscription to Latapult, a Geographic Information System (GIS) platform specializing in land development and real estate data. Latapult provides detailed information on property records, zoning regulations, land use, and market trends. According to Latapult's End User License Agreement (EULA), users are granted a limited, non-transferable license to use the Latapult database.

Additional Data Source: The Google Cloud Distance Matrix API is used to calculate distances between landowners' residences and their parcels of land. All requests were authenticated via API key and complied with Google Cloud's Terms of Service.

Scope of the Data: The dataset covers the geographical region of the greater Charlotte area in North Carolina. The data sourced from Latapult is specific to this region, providing detailed insights into property records, ownership history, and transactional timelines. Some aspects of the dataset include property purchase and sale dates, allowing for an analysis of land ownership duration over time.

1.2 Data Preprocessing

My **transform_dataset** function takes raw real estate data, transforms it into a structured format, and outputs a clean dataset ready for model training and analysis. The transformation process begins by removing unnecessary features, such as Assessed Improvement Value Ratio, Mortgage Date, and Condition, which are deemed irrelevant or redundant for the predictive task.

A significant aspect of the transformation involves feature engineering. For instance, the function calculates the "Ownership Distance" by determining the number of miles between an owner's residence and the land asset. Additionally, financial ratios such as the "Loan-to-Value Ratio" and "Tax-to-Value Ratio" are computed to capture the financial stress or investment characteristics of the property, which can influence selling behavior.

(FUTURE) To prepare categorical variables for regression modeling, one option is to apply one-hot encoding to nominal features like Zoning Type and Land Use. This encoding process converts categorical data into numerical form while preserving the original information, ensuring compatibility with machine learning algorithms.

Additionally, numerical features are standardized using a StandardScaler to normalize their distributions. Normalization is essential for regression models as it prevents features with larger scales from dominating the learning process.

1.3 Statistical Summaries

My **analyze_transformed_dataset** function and its helper methods provide a framework for exploring and summarizing the features by calculating statistics and generating visualizations of feature distributions. By focusing on numerical features, the function computes measures of central tendency (mean, median, mode) and dispersion (standard deviation, variance, range, and interquartile range). The statistical summaries are included at the end of this report.

1.4 Data Splitting

Training and Test Sets: For now, the training to testing ratio is 80 to 20 respectively. This ratio is set based on the standards explained in class, however, the opportunity to modify the ratio will be a future decision when I am confident enough in analyzing the bias and variance reports of the model's outputs. As I increase the amount of data being trained, the ratio will increase to 90 or 99 to 10 or 1 respectively. The current size of the dataset is 100 examples randomly pulled from a downloaded .csv file of over 5000 examples.

2 Methods and Experiments

2.1 Initial Linear Regression Model

An initial linear regression model with minimal computational overhead was developed to serve as a benchmark for predicting property sale prices and guide the early stages of developing the more complex model. At this stage, the focus is on understanding the relationships between key features, such as land acreage, market land values, and geographical coordinates to see how they influence sale prices.

The code begins by loading and preprocessing the dataset, including handling missing values and standardizing numerical features for consistency and variance across features. It splits the data into training and testing sets based on the ratio mentioned above. A linear regression model is then trained on the processed features and targets to predict sale prices. Finally, the model's accuracy is assessed using metrics like Mean Squared Error (MSE) and R^2, providing insights into its predictive ability and areas for improvement.

2.2 Milestone

At this stage in the project, I have spent weeks transforming the data and communicating with experts in the land development field to fully understand the value of this prediction model. I then spent a week developing a simple linear regression model to predict the sale price of parcels of land. This is NOT the final target value, however, I wanted to test prediction models on the data to initially grasp how the model will work, what the data processing looks like, and additional concerns that might arise during the training.

This lead me to statistically analyze the features within the dataset and start the data augmentation process which was described above. Although they are not included in this milestone report, a distribution plot was created for each feature and was inspected for quality purposes in regards to model variance. Many features were removed when creating the augmented dataset as many of the cells were empty, or contained the same value for every record. These features would provided little information towards training the weights.

The Distance Matrix API within Google Cloud's API & Services is currently being used to engineer an additional feature that stores the distance between the owner's residence, and the parcel of the land they own. This will be my final target value, and I have cross-referenced this feature design with field experts to back the decision.

With the dataset obtained and new features developed, I modified the existing simple linear regression model to set a new starting point in this development phase. The results were not promising as I received high MSE values in both the training and test sets, indicating significant prediction errors. Additionally, low extremely R^2 scores were reported on training data suggesting that the model is barely able to explain the variance in the training set. This could mean the features are not informative enough to predict the target variable or the relationship between features and the target variable is non-existent or very weak. There is evidence within this industry to prove the relationship between the

distance between an owner's residence and the parcel's address and the length of time they possess the parcel of land under their name, however, this model is not able to capture this relationship.

With these results and initial assessment, I started tracking the individual predictions of the model. The first issue I recognized was that some predictions from the model produced negative distances, which is not possible. This could be the result of many reasons, and my analysis is described below:

1. **Too much variance with the features included in the dataset and the target value.** For testing purposes, I decreased the number of features included in the dataset to try and combat this issue. I noticed a lower MSE error for both sets, however, they were marginally small changes.
2. **To ensure that the linear regression model does not predict negative distances (since distances cannot be negative), I transformed the target feature using log algorithms.** I applied the log function with an small epsilon value to the feature before training, and reversed the process after predictions. This modification, however, did not work well as many distance values were zero, meaning the transformed distance value was equal to the epsilon value, resulting in almost all predictions equaling epsilon. This left me with the same prediction error as before, except the model is predicting epsilon and not zero.
3. **L1 or L2 regularization in models can stabilize predictions and reduce noise.** With some extreme cases of negative predictions, I applied the ridge regression with L2 regularization during training and noticed an increase the R^2 score and more values along the $y = x$ line when plotting predicted vs actual target values. However, I experienced a negative impact towards the values that mattered more than the majority of the examples with a distance equal to zero. This lead me to believe the target feature distribution of my examples were not spread equally.
4. **Distance distribution values are not uniformly scattered between the min and max distance value.** Due to the fact that many owners live at the same address as the land parcels they own, a majority of the records have a target value of zero, while a minority of them have a value greater than zero. The goal of the model is to predict the distance between the owner's resident and the parcel's address leading me to conclude that many of my examples have a target value of zero, and the model is having a hard time finding the relationship between features when the distance is greater than zero. I dove deep into this analysis by first understanding the distribution of distances in the dataset through plots in a similar fashion of what I applied to the original dataset features before augmentation. Over 90% of the data was examples with a distance of 0, raising a major concern that needs to be addressed.
5. *(FUTURE)* **Stratified Sampling can be used to divide the augmented dataset into distance categories of [0, 1-50, 50-99, 100-149, 150-200].** The intention is to balance the training and testing set distributions of the target feature value between the min and max.
6. *(FUTURE)* **Weighted Loss Function can be used to modify the loss function to give higher weight to examples from underrepresented distances.** This approach can be used if the original dataset size is not big enough to create a uniformly distributed dataset across target values.

Additionally, I trained a simple random forest regression model and ran it after the linear regression model for each change/modification listed above to see if the issue was related to the shape of the model being linear or not. I have not come to any conclusion yet based on the generated outputs.

My biggest concern right now is addressing the value distribution across target values during training and testing. The relationship between the target value and features exists within the industry and is a usable quantitative value during investment decisions, however, minimal models have been created to combat this gap. I am balancing both the model development aspect of the project as well as the data quality used during training. Many features are accessible to me and it is a matter of filtering the data to ensure the relationships benefit the prediction model and are not a hindrance to its variation.

Currently, the model's accuracy is not usable and the relationship between the features and target value has not been found. There is a lot of room for testing in regards to the additional methods and algorithms during this developmental process, however, I am focusing on the dataset architecture and the high-level scope of the model with the goal of getting a "good" prediction. From there, I can use complementary methods or algorithms to better the accuracy and usability of the model in the industry.

3 Model Architecture (Code Summary)

3.1 MeckArcGISSales.csv

The data used in this analysis was obtained from Mecklenburg County's Open Mapping platform, specifically the *Parcel Sales* dataset (`https://maps.mecknc.gov/openmapping/data.html`). This dataset provides tabular sales data for Mecklenburg County, NC, maintained by the Mecklenburg County Tax Assessor's Office. It contains over 1,000,000 records and includes fields such as `parcelid`, `transferid`, `saleprice`, `saledate`, and other attributes describing the properties and transactions. The data encompasses sales spanning multiple years and is regularly updated to support resource management decisions via Geographic Information Systems (GIS).

The dataset serves as a critical input for this project, as it captures the underlying spatial and transactional patterns necessary for predictive modeling. However, it is important to note that the dataset comes with use limitations. Mecklenburg County explicitly disclaims any warranties regarding the accuracy or completeness of the information, as it may contain errors or outdated data resulting from collection methods or physical conditions. Despite these constraints, the dataset remains a robust resource for analyzing land transactions and forming the basis of the predictive AI model developed in this report.

3.2 processArcGISSales.py

This script, `processArcGISSales.py`, preprocesses municipal sales data extracted from ArcGIS to prepare it for predictive modeling. The script is designed to handle large datasets efficiently by processing the data in chunks of 10,000 rows, which helps manage memory usage. The preprocessing begins with data cleaning, where rows with missing values in critical fields (`transferid` and `parcelid`) are removed, and transactions with a `saleprice` of zero are filtered out. Two new features are engineered during this step: `Acres`, which converts land area from square feet to acres, and a placeholder column, `VDL Sale Price`. For each unique `transferid`, the script computes the average `saleprice` per transaction and populates the `VDL Sale Price` column.

After processing, the cleaned and augmented data chunks are saved to an intermediate file, which is subsequently read back for sorting. The dataset is sorted by `saledate` (standardized to a consistent datetime format) and `transferid`, and the sorted data is saved to the final output file, `Mod_MeckArcGISSales_Sorted.csv`. Additionally, the script identifies and isolates records where a `transferid` is associated with multiple transactions. These records are further cleaned by removing unnecessary columns and are saved to a separate file, `TransferID_MeckArcGISSales.csv`.

Finally, the script deletes the intermediate file to optimize storage usage. By performing comprehensive cleaning, feature engineering, and efficient sorting, this script ensures that raw municipal data is well-prepared for downstream analytical and predictive tasks.

3.3 1_findFinishedHomeValue.py

The script `1_findFinishedHomeValue.py` focuses on enriching the processed municipal sales data by retrieving unique addresses associated with each parcel ID. This step bridges the gap between parcel identifiers and their corresponding property locations, enabling deeper analysis of property attributes. Using the ArcGIS REST API, the script queries addresses for each parcel ID in the dataset. The API request fetches data from the `CLTEx_MoreInfo` service, ensuring accurate mapping of parcel IDs to their respective addresses. Any duplicates in the returned addresses are removed to maintain a clean list of unique locations.

To handle large datasets efficiently, the script processes the data in chunks of 10 records at a time. A log file mechanism is implemented to keep track of already processed parcel IDs, preventing redundant API calls and optimizing runtime. The retrieved addresses are stored in a new column, `Addresses`, within the dataset. The script writes progress incrementally to an output CSV file, `TransferID_with_Home_Value.csv`, ensuring data persistence throughout the process.

This step is essential for linking parcel data to specific property addresses, enriching the dataset for downstream tasks such as predictive modeling and spatial analysis. By combining the cleaned municipal sales data with address information, the enriched dataset enables more detailed exploration of the relationships between property attributes and transaction values.

3.4 2_findHomeToLotRatio.py

The script `2_findHomeToLotRatio.py` calculates the ratio of finished home value to lot price, a key metric for analyzing the relationship between property development costs and potential market value. Starting with the enriched

dataset TransferID_with_Home_Value.csv, the script removes records with missing Home Transfer ID, ensuring the integrity of the calculations. The Home Transfer ID is standardized by rounding and converting it to a string format for consistency.

Next, the script calculates the count of transactions for each unique Home Transfer ID and adjusts the Finished Home Value by dividing it by the corresponding transaction count. This adjusted value accounts for multiple transactions under the same Home Transfer ID, providing a normalized representation of the finished home value.

The core computation of this step involves calculating the House to Lot Ratio, defined as the ratio of the adjusted finished home value to the VDL Sale Price. To ensure meaningful results, this calculation is performed only for records with a non-zero VDL Sale Price. The resulting ratio is rounded to three decimal places for precision and stored in a new column, House to Lot Ratio.

Finally, the modified dataset is saved to the output file TransferID_with_Ratio.csv, which now includes both the adjusted finished home value and the house-to-lot ratio. This step enhances the dataset's utility for predictive modeling by incorporating a key feature that directly relates finished home values to lot prices.

3.5 3_queryRecordAddress.py

The script 3_queryRecordAddress.py enhances the dataset by associating parcel IDs with geocoded addresses, including their latitude and longitude coordinates. Using a combination of the ArcGIS REST API and the Google Geocoding API, this step bridges the gap between parcel-level data and geographic coordinates, enabling spatial analysis and location-based modeling.

For each parcel ID in the input file TransferID_with_Ratio.csv, the script first queries the ArcGIS REST API to fetch the primary address associated with the parcel. If a valid address is found, it is then geocoded using the Google Geocoding API to obtain latitude and longitude coordinates. This geocoding process ensures accurate spatial placement of parcels for downstream analysis.

The script processes the dataset in chunks of 10 records to maintain memory efficiency and logs progress by appending results incrementally to the output file Addresses_with_Ratio.csv. Previously processed parcel IDs are skipped by comparing against an existing output file, preventing redundant API calls. Columns for Address, Latitude, and Longitude are dynamically added to the dataset if not already present.

This step is crucial for transforming raw parcel data into geospatially enriched records, facilitating the integration of geographic coordinates into the predictive modeling pipeline. By providing precise location data, the enriched dataset supports advanced analyses such as mapping, clustering, and proximity-based predictions.

3.6 4_addFeatures.py

The script 4_addFeatures.py augments the dataset by integrating additional features and applying domain-specific filtering to refine the data for modeling. It begins by loading two input datasets: Addresses_with_Ratio.csv, containing enriched parcel data with geocoded addresses, and Mod_MeckArcGISSales_Sorted.csv, which provides additional land use information. The datasets are merged on the parcelid and transferid columns using a left join to ensure all records from the primary dataset are retained.

A domain-specific filtering step leverages a dictionary of landuseful categories to classify land use types as either *keep* or *drop*. Only parcels with *keep* land use types are retained in the dataset, ensuring the data aligns with the intended scope of residential and multi-family properties. Additionally, the script removes grantee records that appear only once in the dataset, further refining the data to focus on recurring entities that may have more predictive relevance.

The processed dataset is saved as Addresses_with_Ratios_and_New_Features.csv, now enriched with land use details and filtered for quality. This step is critical for preparing a high-quality dataset tailored to the analysis of finished housing prices relative to lot prices, while focusing on relevant property categories and excluding outliers.

3.7 4.12_sortStandardizedGranteeMap.py

The script 4.12_sortStandardizedGranteeMap.py focuses on standardizing grantee names in the dataset to improve data consistency and usability. It begins by loading the input file Addresses_with_Standardized_Grantees.csv, which contains columns for Original_Grantee and Standardized_Grantee. The script verifies that these required columns exist and exits with an error message if they are missing.

The script iterates through each record, prompting the user to decide whether the `Original_Grantee` value should be copied to the `Standardized_Grantee` field. If the `Standardized_Grantee` field is already filled, the record is skipped. For each update, the script allows the user to confirm or deny copying the original value. Upon confirmation, the `Standardized_Grantee` field is updated, and the changes are immediately saved to the CSV file. This approach ensures incremental saving and prevents data loss in the event of an interruption.

This step is essential for ensuring consistency in grantee naming conventions, particularly when grantee names appear in varying formats across records. The standardized grantee names contribute to a cleaner dataset, facilitating more accurate analyses and modeling in subsequent steps.

3.8 `4.23_addStandardizedGranteeNames.py`

The script `4.23_addStandardizedGranteeNames.py` integrates standardized grantee names into the dataset, enhancing consistency and usability for further analysis. It begins by loading two input files: `Addresses_with_Ratios_and_New_Features.csv`, which contains enriched property data, and `Addresses_with_Standardized_Grantees.csv`, which provides a mapping of `Original_Grantee` to `Standardized_Grantee`. Both files are validated to ensure the presence of required columns.

Using the grantee mapping file, a dictionary is created to map original grantee names to their standardized equivalents. This mapping is then applied to the `grantee` column in the property dataset, creating a new column, `Mapped_Grantee`, that contains the standardized names. This step ensures that variations in grantee naming conventions across records are resolved, improving the dataset's consistency.

The updated dataset is saved to the output file `Addresses_with_Mapped_Grantees.csv`. By incorporating standardized grantee names, this step facilitates cleaner and more reliable analyses in subsequent phases of the project, particularly for tasks that involve grantee-specific grouping or trends.

3.9 `4.24_transformData.py`

The script `4.24_transformData.py` performs final transformations and cleaning on the dataset to prepare it for analysis and modeling. The input file `Addresses_with_Mapped_Grantees.csv` is loaded, and several columns deemed unnecessary for further analysis are dropped, including `Home Transfer ID`, `Finished Home Value`, and `Address`. The script then handles missing values by removing records with null entries in critical fields such as `Acres`, `VDL Sale Price`, `Latitude`, `Longitude`, `landuseful`, `Mapped Grantee`, `naldesc`, and `Adjusted Finished Home Value`.

The `saledate` column is transformed into a year-month format (`YYYY-MM`) to simplify temporal analyses. Outliers are addressed using the interquartile range (IQR) method, targeting critical features such as `House to Lot Ratio`, `Acres`, `VDL Sale Price`, `Latitude`, and `Longitude`. For each feature, values outside the defined IQR thresholds are removed to reduce noise and improve dataset reliability.

The cleaned and transformed dataset is saved to `Addresses_with_Mapped_Grantees_Cleaned.csv`. By removing irrelevant columns, addressing missing values, and handling outliers, this script ensures the dataset is optimized for downstream modeling tasks and predictive analysis.

3.10 `4.25_findBestIQRGrid.py`

The script `4.25_findBestIQRGrid.py` conducts a grid search to identify the optimal interquartile range (IQR) parameters for outlier removal, aiming to enhance the predictive performance of the dataset. Using combinations of IQR thresholds for key features (`House to Lot Ratio`, `Acres`, and `VDL Sale Price`), the script evaluates the effects of varying bottom and top quantile thresholds on model performance.

The process begins by loading and preprocessing the input dataset `Addresses_with_Mapped_Grantees.csv`. Features such as `saledate` are transformed into numeric representations for temporal analysis, and missing values are addressed to ensure data integrity. For each combination of IQR thresholds, the script removes outliers, ensuring that the resulting dataset retains a sufficient number of records for model training and evaluation.

A machine learning pipeline is built using the `XGBRegressor` algorithm, with preprocessing handled by a `ColumnTransformer`. Numeric features are passed through directly, while categorical features are encoded using one-hot encoding. The pipeline is trained and evaluated on a train-test split, with model performance assessed using R^2 and mean squared error (MSE) metrics on the test set.

Results for each IQR configuration, including the number of records used and the corresponding performance metrics, are recorded in `IQR_Grid_Search_Results.csv`. The grid search iteratively appends results to the file, ensuring data persistence and allowing for real-time analysis of performance trends.

The script identifies and prints the optimal IQR parameters that maximize the R^2 score. This step ensures that the dataset is optimally prepared for predictive modeling by balancing outlier removal with the preservation of valuable data records.

3.11 Model: `2_regressionModel.py`

The script `2_regressionModel.py` implements a regression model using a linear regression approach to predict the `House to Lot Ratio`. The dataset `Addresses_with_Ratio.csv` is preprocessed by handling missing values and removing outliers in key columns (`House to Lot Ratio`, `Longitude`, and `Latitude`) using the interquartile range (IQR) method. Temporal data from the `saledate` column is transformed into a numeric feature, `sale_year`, to incorporate temporal effects into the model.

The model uses features such as `Acres`, `VDL Sale Price`, `Finished Home Value`, `Latitude`, `Longitude`, and `sale_year`. These features are preprocessed using a `ColumnTransformer`, and a pipeline is constructed to streamline preprocessing and model training. The dataset is split into training and testing subsets to evaluate the model. Performance is measured using metrics like mean squared error (MSE) and R^2 on both the training and test sets. Additionally, a scatter plot of predicted versus actual values is generated and saved to `Models/Outputs/predicted_vs_actual.png`, providing a visual assessment of the model's performance.

Pros and Cons

Pros:

- **Simplicity:** The pipeline integrates preprocessing and model training seamlessly, reducing complexity and potential errors.
- **Efficiency:** Linear regression is computationally inexpensive, making it suitable for initial exploration of the dataset.
- **Feature Engineering:** Incorporating temporal information (`sale_year`) and removing outliers improve data quality.

Cons:

- **Limited Flexibility:** Linear regression assumes linear relationships, which may not capture complex patterns in the data.
- **Outlier Sensitivity:** Despite IQR-based handling, linear regression remains sensitive to outliers, potentially skewing results.
- **Underfitting Risk:** The model may underfit the data due to its simplicity, especially for non-linear relationships.
- **Feature Utilization:** The model does not include categorical features such as `naldesc`, limiting the use of all available data.

This script establishes a baseline model for comparison with more advanced architectures, highlighting areas for improvement in capturing non-linear relationships and utilizing categorical features.

3.12 Model: `2.1_regressionModel_regularization.py`

The script `2.1_regressionModel_regularization.py` extends the baseline linear regression model by incorporating regularization techniques, specifically Ridge and Lasso regression. The dataset `Addresses_with_Ratio.csv` undergoes preprocessing, including the handling of missing values and outlier removal using the interquartile range (IQR) method. Temporal data from the `saledate` column is transformed into a numeric feature, `sale_year`, to incorporate temporal effects.

Ridge regression applies ℓ_2-norm regularization to penalize large coefficient magnitudes, while Lasso regression employs ℓ_1-norm regularization, which can lead to feature selection by driving some coefficients to zero. Both models are trained using features such as `Acres`, `VDL Sale Price`, `Finished Home Value`, `Latitude`, `Longitude`, and `sale_year`. Preprocessing is achieved through a `ColumnTransformer` that standardizes numeric features using `StandardScaler`.

The models are evaluated using mean squared error (MSE) and $\mathtt{R}^2$ metrics on both training and test sets. Separate scatter plots of predicted versus actual values are generated for Ridge and Lasso regression models, visually assessing their performance against the ideal fit line ($y = x$). These plots are saved as `ridge_predicted_vs_actual.png` and `lasso_predicted_vs_actual.png` in the `Models/Outputs` directory.

Pros and Cons

Pros:

- **Improved Generalization:** Regularization helps prevent overfitting, particularly in Ridge regression, by constraining the magnitude of coefficients.
- **Feature Selection:** Lasso regression inherently performs feature selection by setting less important feature coefficients to zero.
- **Scalability:** The inclusion of `StandardScaler` ensures that features are on a comparable scale, improving model convergence.
- **Comparison of Regularization Techniques:** Side-by-side evaluation of Ridge and Lasso provides insight into the suitability of each approach for the dataset.

Cons:

- **Model Complexity:** Regularization introduces hyperparameters (α) that require tuning for optimal performance.
- **Potential Underfitting:** Excessive regularization may lead to underfitting, particularly if α is too large in Ridge regression.
- **Lasso Limitations:** Lasso may struggle in datasets with multicollinearity, where highly correlated features can produce inconsistent coefficient selection.
- **Assumption of Linearity:** Despite regularization, the models still assume a linear relationship between features and the target, which may limit performance on complex datasets.

This script represents an incremental improvement over the baseline model by reducing overfitting and introducing a mechanism for feature selection. However, further optimization, such as hyperparameter tuning and exploration of non-linear models, may be necessary for complex relationships in the data.

3.13 Model: `2.2_regressionModel_randomForest.py`

The script `2.2_regressionModel_randomForest.py` utilizes a Random Forest regression model to predict the `House to Lot Ratio`, introducing a non-linear approach to model the relationships between features and the target variable. The dataset `Addresses_with_Ratio.csv` is preprocessed to handle missing values and remove outliers in key columns (`House to Lot Ratio`, `Longitude`, and `Latitude`) using the interquartile range (IQR) method. Temporal data from the `saledate` column is transformed into a numeric feature, `sale_year`.

The model uses features such as `Acres`, `VDL Sale Price`, `Finished Home Value`, `Latitude`, `Longitude`, and `sale_year`, with numeric features standardized using a `ColumnTransformer`. A Random Forest model, configured with 100 estimators and a maximum depth of 10, is built into a pipeline for seamless integration with the preprocessing step. The dataset is split into training and testing subsets for evaluation.

The model's performance is assessed using mean squared error (MSE) and $\mathtt{R}^2$ metrics on both the training and test sets. A scatter plot comparing predicted versus actual values is generated, providing a visual assessment of the model's accuracy. The plot is saved as `rf_predicted_vs_actual.png` in the `Models/Outputs` directory.

Pros and Cons

Pros:

- **Captures Non-Linear Relationships:** Random Forest can model complex patterns and interactions between features, improving predictive performance on non-linear datasets.
- **Feature Importance:** The model inherently ranks feature importance, offering insights into the most influential variables.
- **Robust to Outliers:** Random Forest is less sensitive to outliers compared to linear models.
- **Scalable:** The approach is effective on large datasets with diverse feature sets.

Cons:

- **Overfitting Risk:** While the maximum depth of 10 mitigates this risk, Random Forest can overfit on small or imbalanced datasets.
- **Interpretability:** The model is more challenging to interpret compared to linear regression models due to its ensemble nature.
- **Hyperparameter Tuning:** Model performance depends on parameters such as the number of estimators and tree depth, which require optimization.
- **Computational Cost:** Training and prediction times are higher than simpler models, especially as dataset size or model complexity increases.

This script demonstrates a significant advancement by leveraging Random Forest regression to handle non-linear relationships and feature interactions, providing a robust and flexible modeling framework.

3.14 Model: `2.3_regressionModel_xgboost.py`

The script `2.3_regressionModel_xgboost.py` employs the XGBoost regression algorithm to predict the `House to Lot Ratio`, introducing a gradient boosting approach for handling complex relationships in the data. The dataset `Addresses_with_Ratio.csv` undergoes preprocessing, including the handling of missing values and outlier removal in key columns (`House to Lot Ratio`, `Longitude`, and `Latitude`) using the interquartile range (IQR) method. Temporal data from the `saledate` column is transformed into a numeric feature, `sale_year`.

The model uses features such as `Acres`, `VDL Sale Price`, `Finished Home Value`, `Latitude`, `Longitude`, and `sale_year`, with numeric features standardized using a `ColumnTransformer`. The XGBoost model is configured with hyperparameters including `n_estimators`=100, `max_depth`=6, `learning_rate`=0.1, `subsample`=0.8, and `colsample_bytree`=0.8, which control the number of trees, tree depth, learning rate, and sampling proportions for features and rows.

The model's performance is evaluated using mean squared error (MSE) and $\mathtt{R}^2$ metrics on both the training and test sets. A scatter plot comparing predicted versus actual values is generated to visually assess the model's accuracy. This plot is saved as `xgb_predicted_vs_actual.png` in the `Models/Outputs` directory.

Pros and Cons

Pros:

- **Powerful Non-Linear Modeling:** XGBoost excels at capturing complex patterns and interactions between features, improving predictive accuracy.
- **Feature Importance Insights:** The model provides feature importance metrics, offering valuable insights into the most influential variables.
- **Efficient Handling of Missing Values:** XGBoost can handle missing values directly, simplifying preprocessing.
- **Regularization:** Built-in regularization (ℓ_1 and ℓ_2) reduces the risk of overfitting.
- **Scalability:** The algorithm is optimized for speed and scalability, making it suitable for large datasets.

Cons:

- **Hyperparameter Tuning Complexity:** Optimal performance requires careful tuning of multiple hyperparameters, which can be computationally intensive.
- **Interpretability:** The complexity of the ensemble model reduces interpretability compared to simpler models.
- **Potential Overfitting:** Without proper tuning, XGBoost may overfit, especially on small or noisy datasets.
- **Computational Cost:** The algorithm is more resource-intensive compared to simpler models, particularly during hyperparameter optimization.

This script represents a significant advancement in predictive modeling, leveraging the strength of gradient boosting to handle non-linear relationships and feature interactions effectively.

3.15 Model: `2.31_regressionModel_xgboost_hypTuning.py`

The script `2.31_regressionModel_xgboost_hypTuning.py` enhances the XGBoost regression approach by introducing hyperparameter tuning through grid search. This model predicts the `House to Lot Ratio` using optimized configurations for XGBoost, aiming to improve model performance through systematic exploration of hyperparameter combinations. The dataset `Addresses_with_Ratio.csv` is preprocessed to handle missing values, remove outliers in key columns (`House to Lot Ratio`, `Longitude`, and `Latitude`) using the interquartile range (IQR) method, and extract temporal features (`sale_year`) from the `saledate` column.

The features used include `Acres`, `VDL Sale Price`, `Finished Home Value`, `Latitude`, `Longitude`, and `sale_year`. These are standardized using a `ColumnTransformer`. A pipeline integrates preprocessing and the XGBoost regressor, ensuring seamless model construction and training. Hyperparameters such as `n_estimators`, `max_depth`, `learning_rate`, `subsample`, and `colsample_bytree` are tuned via a grid search with 5-fold cross-validation, optimizing for the highest R^2 score.

The optimal hyperparameters identified through grid search are applied to evaluate the model on the test set. Performance is measured using mean squared error (MSE) and R^2 metrics. A scatter plot of predicted versus actual values is generated, assessing the model's accuracy with the best parameters. This plot is saved as `xgb_best_predicted_vs_actual.png` in the `Models/Outputs` directory.

Pros and Cons

Pros:

- **Improved Model Performance:** Hyperparameter tuning ensures optimal configuration for the dataset, enhancing predictive accuracy.
- **Comprehensive Evaluation:** Cross-validation ensures robustness of the selected parameters across different data splits.
- **Advanced Non-Linear Modeling:** XGBoost captures complex interactions and non-linear relationships effectively.
- **Efficient Resource Utilization:** Parallelized computations during grid search reduce runtime.
- **Visualization of Performance:** The scatter plot provides an intuitive assessment of model accuracy.

Cons:

- **Computationally Intensive:** Grid search with multiple hyperparameters can be resource-intensive and time-consuming.
- **Complexity in Parameter Tuning:** Managing and interpreting the results of high-dimensional parameter grids can be challenging.
- **Risk of Overfitting:** Extensive tuning may inadvertently overfit the model to the training data, though cross-validation mitigates this risk.
- **Reduced Interpretability:** The complexity of the XGBoost model and its tuned hyperparameters make it less interpretable than simpler models.

This model represents a highly optimized version of XGBoost, demonstrating the benefits of hyperparameter tuning while balancing computational trade-offs for improved predictive performance.

3.16 Model: `3_regressionModel_newFeatures_xgboost_hypTuning.py`

The script `3_regressionModel_newFeatures_xgboost_hypTuning.py` extends the XGBoost regression model by incorporating additional categorical features and refining hyperparameter tuning for improved predictive accuracy. The dataset `Addresses_with_Ratios_and_New_Features.csv` is preprocessed to handle missing values, remove outliers in critical columns (`House to Lot Ratio`, `Longitude`, `Latitude`) using the interquartile range (IQR) method, and extract temporal features (`sale_year`) from the `saledate` column.

The model uses features such as `Acres`, `VDL Sale Price`, `Latitude`, `Longitude`, `sale_year`, and the categorical feature `landuseful`. The numerical features are passed directly, while the categorical feature is encoded using `OneHotEncoder`. These preprocessing steps are integrated into a pipeline along with the XGBoost regressor to streamline model training and evaluation.

Hyperparameter tuning is performed using `GridSearchCV`, testing combinations of key parameters such as `n_estimators`, `max_depth`, `learning_rate`, `subsample`, and `colsample_bytree`. Five-fold cross-validation is employed to ensure robust parameter selection, optimizing for the highest R^2 score. The best configuration is then used to evaluate the model on the test set, with performance measured using mean squared error (MSE) and R^2 metrics.

A scatter plot comparing predicted versus actual values is generated, providing a visual representation of the model's accuracy with the tuned parameters. The plot is saved as `xgboost_predicted_vs_actual.png` in the `Models/Outputs` directory.

Pros and Cons

Pros:

- **Improved Feature Utilization:** The inclusion of the categorical feature `landuseful` enhances the model's ability to capture diverse patterns in the data.
- **Hyperparameter Optimization:** Grid search ensures the selection of optimal model parameters, improving predictive accuracy.
- **Advanced Non-Linear Modeling:** XGBoost effectively handles complex feature interactions and non-linear relationships.
- **Comprehensive Evaluation:** Cross-validation provides robust parameter validation across multiple data splits.
- **Efficient Feature Encoding:** The integration of `OneHotEncoder` ensures compatibility with categorical variables.

Cons:

- **Computational Overhead:** The addition of categorical features and hyperparameter tuning increases computational complexity.
- **Reduced Interpretability:** The increased feature space from one-hot encoding and the complexity of the XGBoost model reduce interpretability.
- **Risk of Overfitting:** Extensive hyperparameter tuning and inclusion of categorical variables may increase the risk of overfitting, though cross-validation mitigates this.
- **Resource-Intensive:** Grid search for multiple hyperparameters requires significant computational resources and time.

This script represents a comprehensive approach to improving model performance by integrating additional features and leveraging hyperparameter tuning to enhance the predictive capabilities of XGBoost.

3.17 Model: 3.1_regressionModel_xgboost_addRegHypTuning.py

The script 3.1_regressionModel_xgboost_addRegHypTuning.py builds upon the previous XGBoost model by incorporating both ℓ_1 (Lasso) and ℓ_2 (Ridge) regularization into the hyperparameter tuning process. This model predicts the House to Lot Ratio using optimized configurations of XGBoost, including advanced regularization to further reduce overfitting. The dataset Addresses_with_Ratios_and_New_Features.csv is preprocessed to handle missing values, remove outliers in critical columns (House to Lot Ratio, Longitude, Latitude) using the interquartile range (IQR) method, and extract temporal features (sale_year) from the saledate column.

The model leverages features such as Acres, VDL Sale Price, Latitude, Longitude, sale_year, and the categorical feature landuseful. A ColumnTransformer standardizes numeric features and applies one-hot encoding to categorical features, ensuring compatibility with the model. Hyperparameter tuning is conducted using GridSearchCV, which evaluates combinations of key parameters such as n_estimators, max_depth, learning_rate, subsample, colsample_bytree, reg_alpha (ℓ_1), and reg_lambda (ℓ_2) with 5-fold cross-validation to optimize for the highest R^2 score.

The best model identified through grid search is evaluated on the test set, with performance measured using mean squared error (MSE) and R^2 metrics. A scatter plot of predicted versus actual values is generated to visually assess the model's accuracy with the tuned parameters. This plot is saved as 3.1xgboost_predicted_vs_actual_cv5.png in the Models/Outputs directory.

Pros and Cons

Pros:

- **Enhanced Overfitting Control:** Incorporating ℓ_1 and ℓ_2 regularization into hyperparameter tuning provides additional control over model complexity and overfitting.
- **Comprehensive Feature Utilization:** The inclusion of categorical features and advanced regularization improves the model's ability to capture diverse patterns in the data.
- **Optimal Model Configuration:** Grid search ensures the selection of the best parameter combinations for the dataset.
- **Advanced Non-Linear Modeling:** XGBoost captures complex feature interactions and non-linear relationships effectively.
- **Cross-Validation Robustness:** Five-fold cross-validation ensures reliable parameter validation across data splits.

Cons:

- **Computational Complexity:** The addition of regularization parameters and extended hyperparameter tuning increases computational requirements.
- **Interpretability Challenges:** The complexity of XGBoost and regularization makes the model less interpretable compared to simpler linear models.
- **Potential Risk of Overfitting:** While regularization mitigates overfitting, improper tuning or excessive parameters could still overfit the training data.
- **Longer Training Time:** The expanded parameter grid significantly increases training time due to the large number of evaluations required.

This script demonstrates a highly sophisticated approach to predictive modeling, balancing model complexity and generalization through advanced regularization and hyperparameter optimization.

3.18 Model: 3.2_regressionModel_xgboost_hyptuning_dropFeatures.py

The script 3.2_regressionModel_xgboost_hyptuning_dropFeatures.py builds on the XGBoost regression model by selectively removing less important features based on domain knowledge and feature importance analysis. The dataset Addresses_with_Ratios_and_New_Features.csv undergoes preprocessing to handle missing values, remove outliers in critical columns (House to Lot Ratio, Longitude, Latitude) using the interquartile range

(IQR) method, and exclude records associated with unimportant `landuseful` categories. Temporal data from the `saledate` column is transformed into the numeric feature `sale_year`.

The model uses features such as `Acres`, `VDL Sale Price`, `Latitude`, `Longitude`, `sale_year`, and the filtered categorical feature `landuseful`. Numeric features are passed through directly, while categorical features are encoded using `OneHotEncoder`. These preprocessing steps are integrated into a pipeline with the XGBoost regressor for streamlined training and evaluation.

Hyperparameter tuning is conducted via `GridSearchCV`, evaluating combinations of parameters including `n_estimators`, `max_depth`, `learning_rate`, `subsample`, `colsample_bytree`, `reg_alpha` (ℓ_1), and `reg_lambda` (ℓ_2) with 3-fold cross-validation. The optimal parameter set is applied to evaluate the model on the test set, with performance measured using mean squared error (MSE) and $\mathtt{R}^2$ metrics.

A scatter plot comparing predicted versus actual values is generated, visually assessing the model's accuracy after feature reduction. This plot is saved as `xgboost_predicted_vs_actual_after_dropping_features.png` in the `Models/Outputs` directory.

Pros and Cons

Pros:

- **Feature Simplification:** Removing less relevant features (`landuseful` categories) reduces feature space and simplifies the model.
- **Improved Generalization:** The focus on important features mitigates the risk of overfitting to noise in irrelevant data.
- **Advanced Regularization:** Incorporation of ℓ_1 and ℓ_2 regularization further controls overfitting.
- **Efficient Training:** Reducing feature space decreases computational overhead during model training.
- **Cross-Validation Robustness:** Hyperparameter tuning with cross-validation ensures robust parameter selection across data splits.

Cons:

- **Loss of Potentially Informative Features:** Excluding features deemed less important may result in the loss of subtle patterns in the data.
- **Risk of Over-Simplification:** Aggressive feature reduction might overly constrain the model, reducing its flexibility.
- **Interpretability Challenges:** While feature reduction simplifies the model, the remaining features may still interact in complex, non-linear ways, reducing interpretability.
- **Computational Cost of Tuning:** Hyperparameter tuning with a large parameter grid remains computationally expensive, even with reduced feature space.

This script highlights the importance of feature selection and regularization in building efficient and generalized predictive models while maintaining the flexibility of advanced hyperparameter tuning.

3.19 Model: `3.3_regressionModel_xgboost_setParams_increaseCV.py`

The script `3.3_regressionModel_xgboost_setParams_increaseCV.py` refines the XGBoost regression model by narrowing the hyperparameter grid and increasing cross-validation to enhance the robustness of parameter selection. The dataset `Addresses_with_Ratios_and_New_Features.csv` undergoes preprocessing to handle missing values, remove outliers in critical columns (`House to Lot Ratio`, `Longitude`, `Latitude`) using the interquartile range (IQR) method, and exclude records associated with unimportant `landuseful` categories. Temporal data from the `saledate` column is transformed into the numeric feature `sale_year`.

The model utilizes features such as `Acres`, `VDL Sale Price`, `Latitude`, `Longitude`, `sale_year`, and the filtered categorical feature `landuseful`. Numeric features are passed directly, while categorical features are encoded using `OneHotEncoder`. These preprocessing steps are integrated into a pipeline with the XGBoost regressor for streamlined training and evaluation.

Hyperparameter tuning is conducted via `GridSearchCV`, focusing on a smaller, refined parameter grid. Parameters such as `n_estimators`, `max_depth`, `learning_rate`, `subsample`, `colsample_bytree`, `reg_alpha` (ℓ_1), and `reg_lambda` (ℓ_2) are tuned with 5-fold cross-validation to optimize for the highest R^2 score.

The best configuration is applied to evaluate the model on the test set, with performance measured using mean squared error (MSE) and R^2 metrics. A scatter plot comparing predicted versus actual values is generated to visually assess the model's accuracy with the tuned parameters. This plot is saved as `xgboost_predicted_vs_actual_after_dropping_features.png` in the `Models/Outputs` directory.

Pros and Cons

Pros:

- **Robust Parameter Selection:** The increased cross-validation folds enhance the reliability of hyperparameter tuning, reducing variance in model evaluation.
- **Refined Feature Set:** The exclusion of less important `landuseful` categories simplifies the model and focuses on relevant data patterns.
- **Advanced Regularization:** Incorporation of ℓ_1 and ℓ_2 regularization parameters helps manage overfitting effectively.
- **Efficient Tuning:** A smaller, focused parameter grid reduces computational overhead while maintaining tuning quality.
- **Improved Generalization:** The balance between feature reduction and advanced regularization supports better generalization to unseen data.

Cons:

- **Potential Loss of Informative Features:** Aggressive feature filtering may exclude subtle but informative data patterns.
- **Interpretability Challenges:** The complex interactions modeled by XGBoost and regularization parameters reduce the ease of interpretation.
- **Risk of Overfitting to Validation Sets:** Extended cross-validation increases the risk of overfitting to validation splits, although mitigated by careful tuning.
- **Limited Exploration of Parameters:** The smaller grid restricts the exploration of potentially beneficial configurations outside the pre-defined ranges.

This script demonstrates a highly focused and efficient approach to refining the XGBoost model, balancing computational efficiency and predictive performance through targeted parameter tuning and robust cross-validation.

3.20 Model: `3.4_regressionModel_xgboost_landuse3categories.py`

The script `3.4_regressionModel_xgboost_landuse3categories.py` focuses on improving the predictive performance of the XGBoost model by mapping detailed land use categories into three broader categories (`single family`, `townhomes`, and `multifamily`). This simplification enhances the model's ability to generalize across the dataset. The dataset `Addresses_with_Ratios_and_New_Features.csv` undergoes preprocessing to handle missing values, remove outliers in critical columns (`House to Lot Ratio`, `Longitude`, `Latitude`) using the interquartile range (IQR) method, and reclassify land use features based on a predefined mapping.

Key features such as `Acres`, `VDL Sale Price`, `Latitude`, `Longitude`, and `sale_year` are combined with the simplified land use categories to train the model. Numeric features are processed directly, while the categorical feature `landuseful` is encoded using `OneHotEncoder`. These preprocessing steps are integrated into a pipeline with the XGBoost regressor, streamlining model training and evaluation.

Hyperparameter tuning is performed via `GridSearchCV`, testing combinations of parameters including `n_estimators`, `max_depth`, `learning_rate`, `subsample`, `colsample_bytree`, `reg_alpha` (ℓ_1), and `reg_lambda` (ℓ_2) with 10-fold cross-validation. The best parameters are used to evaluate the model on the test set, with performance measured using mean squared error (MSE) and R^2 metrics.

A scatter plot comparing predicted versus actual values is generated to visually assess the model's accuracy with the tuned parameters. This plot is saved as `xgboost_predicted_vs_actual_after_dropping_features.png` in the `Models/Outputs` directory.

Pros and Cons

Pros:

- **Simplified Land Use Categories:** Mapping detailed categories into three broader categories (`single family`, `townhomes`, and `multifamily`) reduces complexity while retaining essential information.
- **Robust Parameter Tuning:** The use of 10-fold cross-validation ensures reliable hyperparameter optimization and reduces the likelihood of overfitting.
- **Advanced Regularization:** Incorporation of ℓ_1 and ℓ_2 regularization parameters enhances the model's ability to generalize to unseen data.
- **Efficient Training:** Feature simplification reduces computational overhead while maintaining predictive performance.
- **Improved Interpretability:** Broader land use categories provide a more intuitive understanding of feature contributions.

Cons:

- **Loss of Granularity:** Collapsing detailed land use categories into broader groups may obscure nuanced patterns present in the original data.
- **Computational Cost of Tuning:** Despite the simplified feature set, the use of 10-fold cross-validation increases training time.
- **Risk of Overgeneralization:** Simplifying categorical data risks losing specific interactions or relationships unique to smaller categories.
- **Model Complexity:** While simplified, the interactions captured by XGBoost and regularization parameters still make the model challenging to interpret.

This script demonstrates the effectiveness of simplifying categorical data to enhance model generalization while balancing computational efficiency and predictive performance.

3.21 Model: `3.5_regressionModel_removeCategories.py`

The script `3.5_regressionModel_removeCategories.py` builds upon previous iterations of the XGBoost regression model by further refining the dataset to exclude land use categories that add noise or complexity to the model. This approach aims to enhance the model's ability to generalize and predict the `House to Lot Ratio`.

The dataset `Addresses_with_Ratios_and_New_Features.csv` undergoes preprocessing to handle missing values, remove outliers in critical columns (`House to Lot Ratio`, `Longitude`, `Latitude`) using the interquartile range (IQR) method, and exclude irrelevant records based on a simplified list of land use categories. Temporal data from the `saledate` column is transformed into the numeric feature `sale_year`.

Key features, such as `Acres`, `VDL Sale Price`, `Latitude`, `Longitude`, and `sale_year`, along with the filtered `landuseful` categories, are utilized in the model. Numeric features are passed through directly, while categorical features are encoded using `OneHotEncoder`. These preprocessing steps are integrated into a pipeline with the XGBoost regressor, streamlining model training and evaluation.

Hyperparameter tuning is performed using `GridSearchCV`, exploring combinations of parameters including `n_estimators`, `max_depth`, `learning_rate`, `subsample`, `colsample_bytree`, `reg_alpha` (ℓ_1), and `reg_lambda` (ℓ_2) with 5-fold cross-validation. The best parameter set is used to evaluate the model, with performance measured using mean squared error (MSE) and R^2 metrics.

A scatter plot comparing predicted versus actual values is generated to visually assess the model's accuracy with the tuned parameters. This plot is saved as `xgboost_predicted_vs_actual.png` in the `Models/Outputs` directory.

Pros and Cons

Pros:

- **Focused Dataset:** Removing less relevant land use categories reduces noise, improving the model's performance and generalization.
- **Streamlined Feature Selection:** The reduced feature set simplifies the model, lowering computational requirements for training.
- **Advanced Regularization:** Incorporation of ℓ_1 and ℓ_2 regularization helps manage overfitting, particularly on smaller datasets.
- **Robust Evaluation:** The use of 5-fold cross-validation ensures that the model is tested thoroughly across different data splits.
- **Improved Prediction Accuracy:** Simplifying the dataset and optimizing parameters result in better predictive performance on the test set.

Cons:

- **Risk of Over-Simplification:** Aggressively removing categories may discard subtle but useful patterns in the data.
- **Loss of Interpretability:** While the model is simplified, the combination of advanced regularization and feature encoding may reduce transparency in predictions.
- **Computational Expense:** Although the dataset is smaller, the extensive grid search for hyperparameters remains computationally expensive.
- **Dependency on Category Mapping:** The effectiveness of this approach relies heavily on the accuracy of the land use category filtering and mapping.

This script demonstrates the benefits of targeted data filtering and parameter optimization in improving the performance and efficiency of predictive models.

3.22 Model: `4_regressionModel_xgboost_mappedGrantees.py`

The script `4_regressionModel_xgboost_mappedGrantees.py` incorporates `Mapped_Grantee` as an additional categorical feature in the predictive model. This feature represents standardized grantee names, which aim to capture latent patterns related to property transactions. The dataset `Addresses_with_Mapped_Grantees_Cleaned.csv` is used for this analysis.

The preprocessing steps include converting the `saledate` column into a numeric representation (`saledate_numeric`) based on the number of months since the earliest year in the dataset. This transformation ensures temporal data is appropriately captured for regression. Features include `Acres`, `VDL Sale Price`, `Latitude`, `Longitude`, `saledate_numeric`, `landuseful`, and `Mapped_Grantee`. Numeric features are used directly, while categorical features are encoded using `OneHotEncoder`.

The XGBoost model is trained using a pipeline structure, which combines preprocessing and regression into a streamlined workflow. Hyperparameter tuning is conducted via `GridSearchCV`, with a 5-fold cross-validation to explore parameter combinations such as `n_estimators`, `max_depth`, `learning_rate`, `subsample`, `colsample_bytree`, `reg_alpha` (ℓ_1), and `reg_lambda` (ℓ_2). The model is evaluated using mean squared error (MSE) and $\mathtt{R}^2$ metrics on training and test datasets.

A scatter plot is created to compare predicted versus actual `House-to-Lot Ratio`, visually assessing the model's accuracy. The plot is saved as `xgboost_predicted_vs_actual.png` in the `Models/Outputs` directory.

Pros and Cons

Pros:

- **Inclusion of Grantee Data:** Adding `Mapped_Grantee` captures transaction-specific patterns that may enhance predictive accuracy.

- **Temporal Encoding:** Transforming `saledate` into `saledate_numeric` provides a continuous representation of time, improving regression capabilities.
- **Regularization Parameters:** Use of ℓ_1 and ℓ_2 regularization reduces overfitting risks, especially when dealing with categorical data.
- **Comprehensive Tuning:** Grid search with 5-fold cross-validation ensures a robust exploration of parameter space.
- **Interpretable Workflow:** Feature preprocessing and modeling are consolidated in a pipeline, making the process reproducible and modular.

Cons:

- **Complexity of Categorical Encoding:** The inclusion of multiple categorical features increases computational cost and model complexity.
- **Risk of Overfitting:** While regularization mitigates this, the added features, especially `Mapped_Grantee`, may still overfit the training data.
- **Potential Redundancy:** Some grantee mappings might not significantly contribute to the prediction, diluting the model's focus on relevant features.
- **Time-Consuming Hyperparameter Tuning:** Despite its benefits, grid search adds significant computational overhead.

This model iteration highlights the potential of leveraging standardized grantee data for improved predictive performance while balancing computational and modeling complexities.

- **Data Acquisition:**
  - Use the `MeckArcGISSales.csv` dataset from Mecklenburg County's Open Mapping platform, including fields like `parcelid`, `saleprice`, and `saledate`.
- **Initial Preprocessing (`processArcGISSales.py`):**
  - Clean rows with missing `parcelid`, `transferid`, or zero `saleprice`.
  - Engineer `Acres` (converted from square feet) and `VDL Sale Price`.
  - Sort by `saledate` and output a refined dataset (`Mod_MeckArcGISSales_Sorted.csv`).
- **Address Enrichment (`1_findFinishedHomeValue.py`):**
  - Link parcel IDs to unique addresses using the ArcGIS REST API.
  - Add a new `Addresses` column and log processed IDs to avoid redundant API calls.
- **Ratio Calculation (`2_findHomeToLotRatio.py`):**
  - Normalize `Finished Home Value` by transaction count.
  - Compute `House to Lot Ratio` (`House Value`/`VDL Sale Price`), storing results in `TransferID_with_Ratio.csv`.
- **Geospatial Enrichment (`3_queryRecordAddress.py`):**
  - Add geocoded `Latitude` and `Longitude` using the ArcGIS REST and Google Geocoding APIs.
- **Feature Augmentation (`4_addFeatures.py`):**
  - Merge datasets on `parcelid` and `transferid`.
  - Filter land use categories (`landuseful`) using a predefined dictionary and remove infrequent grantee records.
- **Grantee Standardization (`4.12_sortStandardizedGranteeMap.py` and `4.23_addStandardizedGranteeNames.py`):**
  - Standardize and map `Original_Grantee` to `Mapped_Grantee` for consistency across records.
- **Final Transformation (`4.24_transformData.py`):**
  - Handle missing values in critical fields (e.g., `Acres`, `VDL Sale Price`).
  - Apply IQR-based outlier removal for `House to Lot Ratio` and other key features.

  - Convert `saledate` to `YYYY-MM` format for temporal analysis.
- **IQR Grid Search (`4.25_findBestIQRGrid.py`):**
  - Explore optimal IQR thresholds for outlier removal.
  - Train `XGBRegressor` models with various configurations and evaluate R^2 and MSE.
- **Baseline Model (`2_regressionModel.py`):**
  - Train a linear regression model on features like `Acres`, `VDL Sale Price`, and `sale_year`.
  - Evaluate with scatter plots of predicted vs. actual values.
- **Regularized Models (`2.1_regressionModel_regularization.py`):**
  - Apply Ridge (ℓ_2) and Lasso (ℓ_1) regression, leveraging `StandardScaler` for feature scaling.
- **Non-Linear Models (`2.2_regressionModel_randomForest.py`):**
  - Use Random Forest regression with 100 estimators and a maximum depth of 10.
  - Output feature importance metrics.
- **XGBoost Regression (`2.3_regressionModel_xgboost.py`):**
  - Train an XGBoost model with parameters (`n_estimators`=100, `max_depth`=6, `learning_rate`=0.1).
  - Evaluate predictions using scatter plots and MSE.
- **Hyperparameter Tuning (`2.31_regressionModel_xgboost_hypTuning.py`):**
  - Optimize XGBoost parameters via grid search with 5-fold cross-validation, focusing on ℓ_1 and ℓ_2 regularization.
- **Feature Refinements:**
  - Add `landuseful` as a categorical feature (`3_regressionModel_newFeatures_xgboost_hypTuning.py`).
  - Introduce regularization parameters (`reg_alpha`, `reg_lambda`) for XGBoost (`3.1_regressionModel_xgboost_addRegHypTuning.py`).
  - Remove irrelevant features (`3.2_regressionModel_xgboost_hyptuning_dropFeatures.py`).
- **Land Use Simplification (`3.4_regressionModel_xgboost_landuse3categories.py`):**
  - Map `landuseful` to broader categories (`single family`, `townhomes`, `multifamily`).
- **Grantee Feature Integration (`4_regressionModel_xgboost_mappedGrantees.py`):**
  - Add standardized `Mapped_Grantee` as a feature and apply one-hot encoding.
  - Optimize parameters with grid search and assess predictive performance.