Housing Horizon: Analysing Residential Real Estate

Project submitted to the

SRM University – AP, Andhra Pradesh

for the partial fulfillment of the requirements to award the degree of

Bachelor of Technology

In

Computer Science and Engineering

School of Engineering and Sciences

Submitted by

Sravya Alapati (AP21110010651)

Jahnavi Tadikamalla (AP21110010655)

Niharika Juttuka (AP21110010681)

Yaswanth Yarasani (AP21110010715)

Under the Guidance of

Dr Rajiv Senapati

SRM University–AP

Neerukonda, Mangalagiri, Guntur

Andhra Pradesh – 522 240

Nov, 2023

Certificate

Date: 29-Nov-23

This is to certify that the work present in this Project entitled **"Housing Horizon: Analysing Residential Real Estate"** has been carried out by **Sravya Alapati, Jahnavi Tadikamalla, Niharika Juttuka, Yaswanth Yarasani** under my supervision. The work is genuine, original, and suitable for submission to the SRM University – AP for the award of Bachelor of Technology/Master of Technology in **School of Engineering and Sciences**.

Supervisor

(Signature)

Dr Rajiv Senapati,

Assistant Professor,

Computer Science and Engineering.

Acknowledgement

We extend our deepest gratitude to Dr Rajiv Senapati for his invaluable guidance, unwavering support, and mentorship throughout the course of our data warehouse and dating Mining project. Dr Rajiv Senapati's expertise, encouragement, and dedication have been instrumental in shaping the success of this research endeavour.

His insightful feedback, constructive criticism, and commitment to academic excellence have significantly contributed to the development of this project. We are truly thankful for the opportunity to work under his mentorship and are inspired by his passion for project.

We express our sincere thanks to Dr Rajiv Senapati for his mentorship, which has been a guiding force in our academic's project.

Yours Sincerely,

Sravya Alapati-AP21110010651

Jahnavi Tadikamalla-AP21110010655

Niharika Juttuka-AP21110010681

Yaswanth Yarasani-AP21110010715

Table of Contents

List of figures:

Abstract:

"Housing Horizon: Analyzing Residential Real Estate Using Decision Tree Algorithm" presents a novel and data-driven approach to understanding the dynamics of the residential real estate market. Leveraging the Decision Tree Algorithm, a powerful machine learning tool, this study aims to unravel the intricate patterns and influential factors that shape property values, buyer behavior, and market trends.

The research employs a robust dataset comprising a diverse range of variables, including economic indicators, demographic data, historical property values, and environmental factors. By utilizing the Decision Tree Algorithm, the study navigates through this complex dataset to identify critical decision points and construct a predictive model capable of forecasting property values and market trends.

The Decision Tree Algorithm's ability to reveal non-linear relationships and capture interactions among various factors makes it a valuable tool for dissecting the multi-dimensional nature of residential real estate. The study not only focuses on predicting property values but also explores the factors that contribute most significantly to these predictions, providing a deeper understanding of the market dynamics.

Furthermore, the research investigates the interpretability of the Decision Tree model, shedding light on the key variables and decision pathways that drive its predictions. This transparency enhances the study's applicability, making the findings accessible and actionable for real estate professionals, policymakers, and investors.

The outcomes of this research not only contribute to advancing the application of machine learning in real estate analysis but also offer practical insights for stakeholders looking to navigate the complexities of the residential housing market. By combining the power of the Decision Tree Algorithm with a comprehensive dataset, this study aims to provide a nuanced and predictive understanding of residential real estate, empowering stakeholders to make informed decisions in an ever-changing market landscape.

Introduction:

The residential real estate market is a dynamic and complex ecosystem, influenced by a myriad of factors ranging from economic indicators and demographic trends to environmental considerations. Navigating this intricate landscape requires advanced analytical tools that can uncover hidden patterns and relationships within vast datasets. In this context, the application of machine learning algorithms, such as the Decision Tree Algorithm, presents an exciting opportunity to gain a deeper understanding of the forces shaping the housing market.

The "Housing Horizon: Analysing Residential Real Estate Using Decision Tree Algorithm" study seeks to harness the predictive power of the Decision Tree Algorithm to unravel the intricacies of residential real estate dynamics. Traditionally used in classification and regression tasks, the Decision Tree Algorithm is well-suited for exploring non-linear relationships and interactions among multiple variables—qualities that align with the intricate nature of real estate markets.

The primary objective of this research is to construct a predictive model that not only forecasts property values but also identifies the key drivers behind these predictions. By utilizing a diverse dataset encompassing economic, demographic, and environmental factors, the study aims to create a Decision Tree model that goes beyond mere prediction, providing valuable insights into the decision-making processes influencing residential real estate.

In this introduction, we set the stage for an exploration into the potential of the Decision Tree Algorithm as a tool for comprehensively analyzing residential real estate. By combining the strengths of machine learning with the complexity of real-world housing dynamics, this study aims to contribute to the growing body of knowledge aimed at empowering stakeholders—ranging from real estate professionals to policymakers—with actionable insights for navigating an ever-evolving housing horizon.

Methodology:

2.1. Data Set description:

```
In [2]: df=pd.read_csv("ParisHousingClass.csv")
        df
```

Out[2]:

	squareMeters	numberOfRooms	hasYard	hasPool	floors	cityCode	cityPartRange	numPrevOwners	made	isNewBuilt	hasStormProtector	basement	at
0	75523	3	0	1	63	9373	3	8	2005	0	1	4313	90
1	80771	39	1	1	98	39381	8	6	2015	1	0	3653	24
2	55712	58	0	1	19	34457	6	8	2021	0	0	2937	88
3	32316	47	0	0	6	27939	10	4	2012	0	1	659	71
4	70429	19	1	1	90	38045	3	7	1990	1	0	8435	24
...	...	...	...	...	...	...	...	...	...	...	...	...	...
9995	1726	89	0	1	5	73133	7	6	2009	0	1	9311	16
9996	44403	29	1	1	12	34606	9	4	1990	0	1	9061	17
9997	83841	3	0	0	69	80933	10	10	2005	1	1	8304	77
9998	59036	70	0	0	96	55856	1	3	2010	0	1	2590	61
9999	1440	84	0	0	49	18412	6	10	1994	1	0	8485	20

10000 rows × 18 columns

Figure 1:

This is the dataset the description about every attribute is below the table

1. squareMeters: The total area of the house in square meters, indicating the size of the property.

2. numberOfRooms: The count of rooms in the house, providing a measure of the spatial layout and capacity.

3. hasYard: A binary variable indicating whether the house has a yard (1 for yes, 0 for no), reflecting outdoor space.

4. hasPool: A binary variable denoting the presence of a pool (1 for yes, 0 for no), highlighting a potential luxury feature.

5. floors: The number of floors in the house, giving an indication of its vertical structure.

6. cityCode: The zip code of the city where the house is located, providing geographical information.

7. cityPartRange: A range value indicating the exclusivity of the neighborhood, with higher values representing more exclusive areas.

8. numPrevOwners: The number of previous owners of the house, offering insights into its history.

9. made: The year the house was built, providing information about its age.

10. isNewBuilt: A binary variable indicating whether the house is newly built (1 for yes, 0 for no), highlighting its condition.

11. hasStormProtector: A binary variable denoting the presence of storm protection features (1 for yes, 0 for no), indicating resilience against adverse weather.

12. basement: The square meters of the basement in the house, providing information about additional space.

13. attic: The square meters of the attic in the house, indicating potential extra living or storage space.

14. garage: The size of the garage, if present, giving an idea of parking and storage capacity.

15. hasStorageRoom: A binary variable indicating whether the house has a storage room (1 for yes, 0 for no), providing additional storage space.

16. hasGuestRoom: The number of guest rooms in the house, indicating accommodation capacity for visitors.

17. price: The price of the house, serving as the target variable for prediction models.

18. category: Categorizes the house as "Luxury" or "Basic," representing its overall market segment or quality level.

2.2 Implementation

Using a classful dataset, a decision tree is constructed to classify houses as "Luxury" or "Basic." After preprocessing the data, including handling missing values, encoding categorical variables, and splitting into training (70%) and testing sets (30%), the algorithm learns relationships between attributes (e.g., square meters, amenities) and the target variable. The decision tree, built during training, is evaluated on the testing set to ensure generalization. Performance metrics gauge accuracy, and the interpretable tree highlights key attributes influencing classifications. The model can be fine-tuned and, once validated, predicts new houses' categories, making it a transparent and effective classification tool.

2.2 Problem statement:

Develop a decision tree model to predict house prices based on a classful dataset comprising attributes such as square meters, number of rooms, yard and pool presence, floors, city information, previous owners, year built, and various amenities. The goal is to analyze the impact of these features on housing prices and categorize houses as "Luxury" or "Basic" to assist in real estate market segmentation and pricing strategies.

2.2.1 Importing the modules and the datasets

Pandas: Used for data manipulation and analysis. It provides data structures like DataFrame for handling and analyzing structured data.

NumPy: Essential for numerical operations in Python. It introduces support for large, multi-dimensional arrays and matrices, along with mathematical functions.

Matplotlib: A widely-used plotting library for creating static, animated, and interactive visualizations in Python. It can be used to generate plots, charts, histograms, and more.

Math: A standard Python library providing mathematical functions. It includes operations like exponentiation, logarithms, and trigonometry.

Seaborn: Built on top of Matplotlib, Seaborn is a statistical data visualization library. It simplifies the creation of aesthetically pleasing and informative statistical graphics.

Scipy.stats (IQR):Part of the SciPy library, it provides statistical functions. Here, it seems you've imported `iqr` from `scipy.stats` to calculate the interquartile range (IQR).

Warnings: Python's warnings module is used here to filter out warning messages, ensuring a cleaner output in the code execution.

2.2.2 Data cleaning:

```
In [4]: df.isnull().sum()

Out[4]: squareMeters        0
        numberOfRooms       0
        hasYard             0
        hasPool             0
        floors              0
        cityCode            0
        cityPartRange       0
        numPrevOwners       0
        made                0
        isNewBuilt          0
        hasStormProtector   0
        basement            0
        attic               0
        garage              0
        hasStorageRoom      0
        hasGuestRoom        0
        price               0
        category            0
        dtype: int64
```

Figure 2:

This above figure showing that there are no null values in our dataset

- There are no null values in the dataset

```
In [7]: Q1 = df['price'].quantile(0.25)
        Q3 = df['price'].quantile(0.75)
        IQR = Q3 - Q1

        # identify outliers
        threshold = 1.5
        outliers = df[(df['price'] < Q1 - threshold * IQR) | (df['price'] > Q3 + threshold * IQR)]

In [8]: outliers

Out[8]:
```

squareMeters	numberOfRooms	hasYard	hasPool	floors	cityCode	cityPartRange	numPrevOwners	made	isNewBuilt	hasStormProtector	basement	attic

Figure 3:

This can say that there are no outliers in the dataset

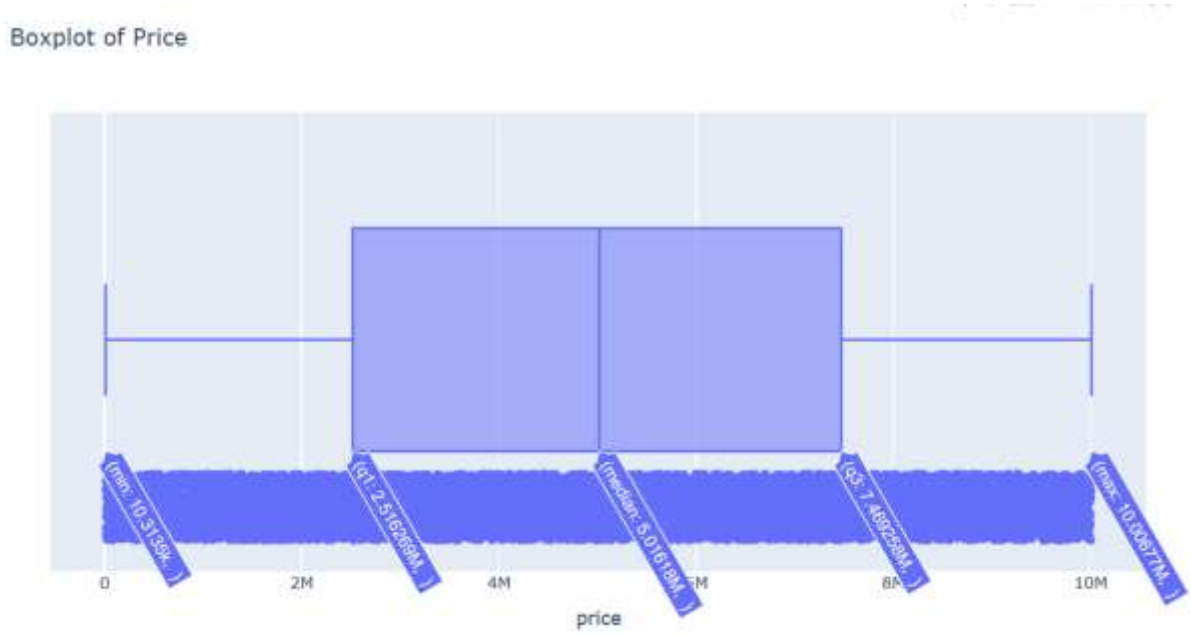


Figure 4:

This is the Boxplot of Price verifying whether there are outliers or not

2.2.3 Data transformation:

We have used decimal scaling for the Price column for transforming and scaling the entire data of the price into same range which is with in [0,1].

Original Data	Scaled Data
7559081.5	0.0756
8085989.5	0.0809
5574642.1	0.0557
3232561.2	0.0323
7055052.0	0.0706
3926647.2	0.0393
5876376.5	0.0588
8696869.3	0.0870
5154055.2	0.0515
3970892.1	0.0397
2366397.3	0.0237
9652258.1	0.0965
1914688.8	0.0191
1320803.4	0.0132
7986665.8	0.0799
7607322.9	0.0761
6420823.1	0.0642

Figure 5:

This is the scaled data from the original through decimal Scaling

2.2.4 Data visualization

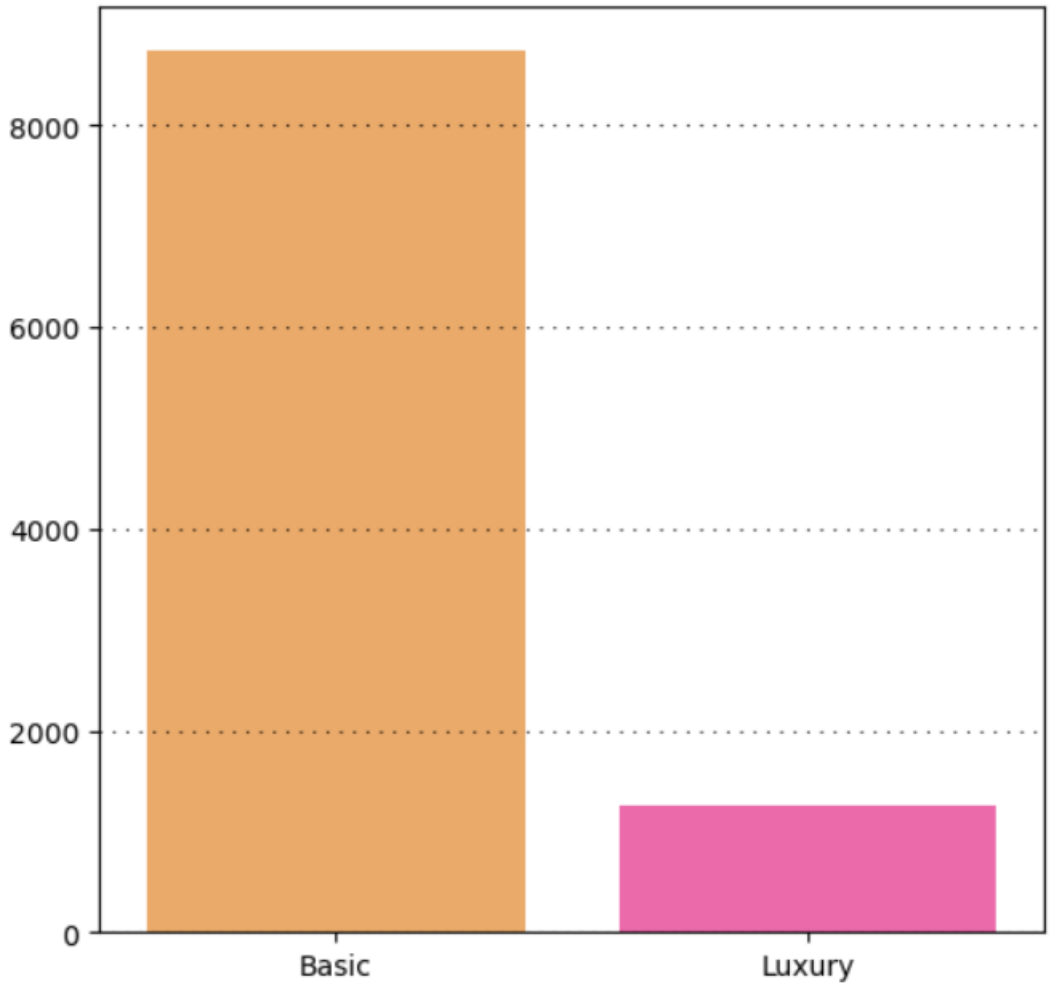


Figure 6:

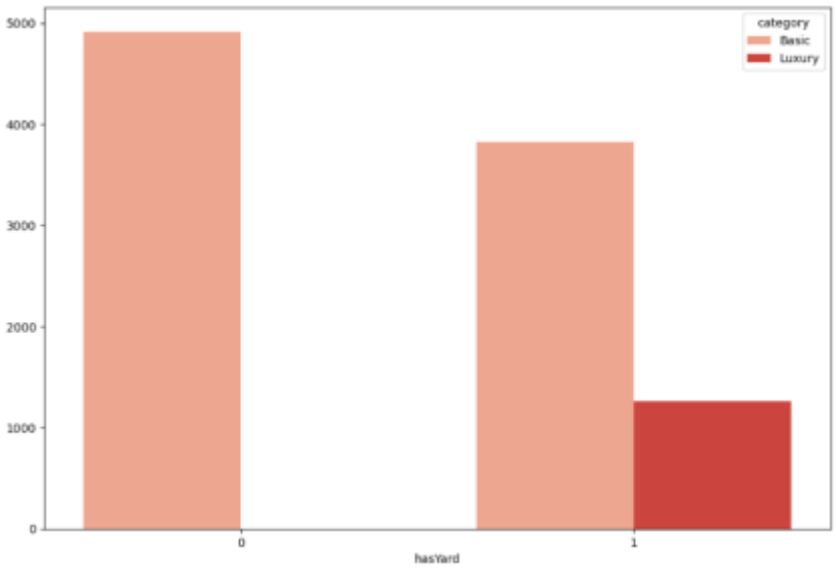
The division of the class column in the bar graph format.

Out[14]:

hasYard	0	1	All
category			
Basic	4913	3822	8735
Luxury	0	1265	1265
All	4913	5087	10000

Figure 7:

These are values of hasyard based on the category



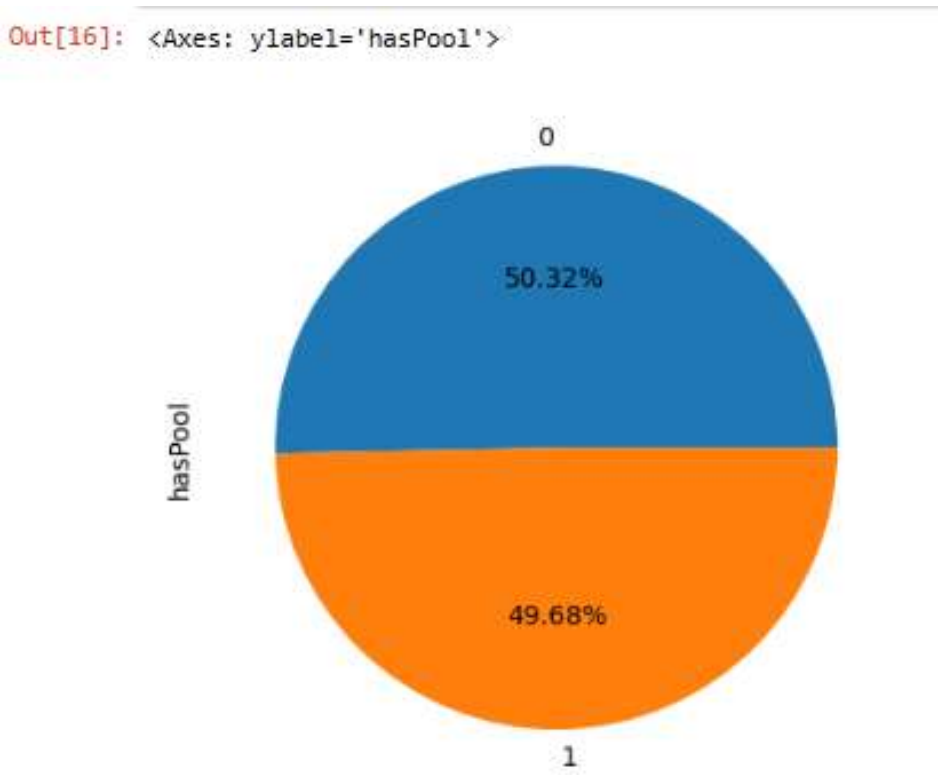


Figure 8:

This the division of the class values for the haspool column in the pie chart format.

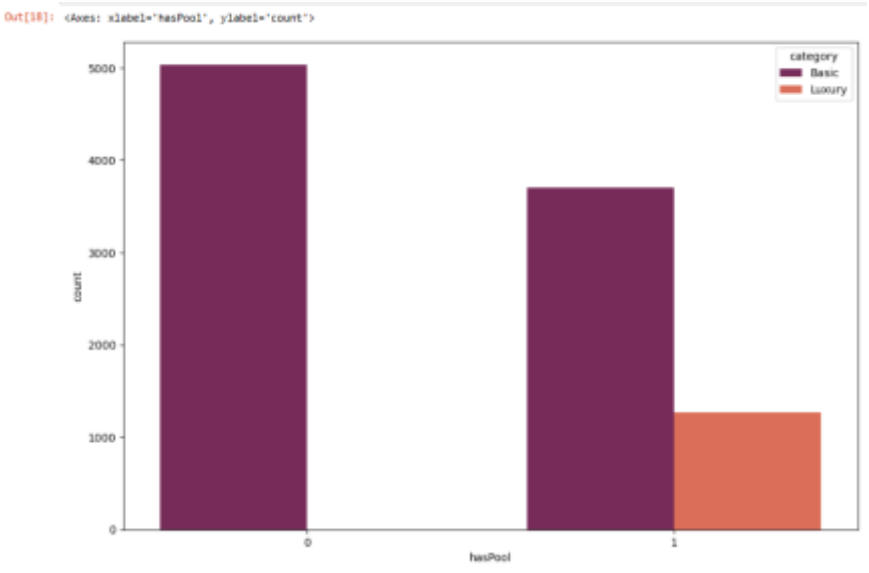


Figure 9:

This the division of the class values for the haspool column in the bar graph format.

Out[17]:

hasPool	0	1	All
category			
Basic	5032	3703	8735
Luxury	0	1265	1265
All	5032	4968	10000

Figure 10:

This the division of the class values for the 'haspool' column.

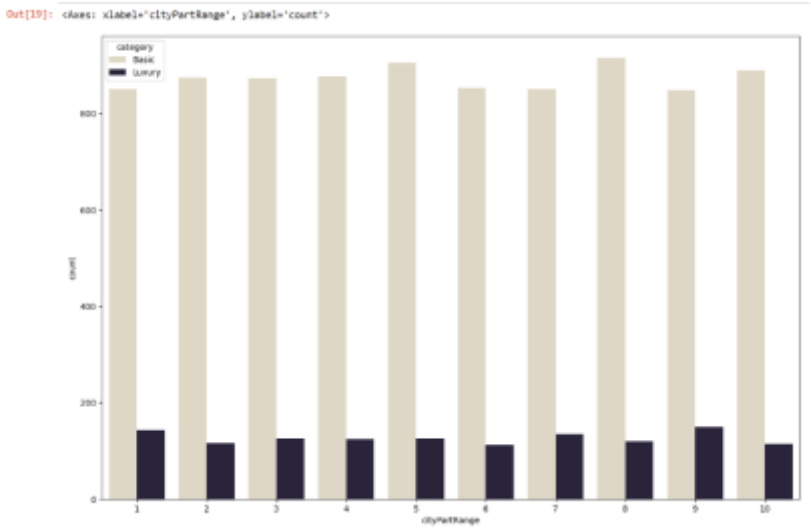


Figure 11:

This is the citypartrange division with respective to the category column

Out[20]:

cityPartRange	1	2	3	4	5	6	7	8	9	10	All
category											
Basic	851	874	873	877	905	853	850	915	848	889	8735
Luxury	143	116	126	124	126	112	134	120	149	115	1265
All	994	990	999	1001	1031	965	984	1035	997	1004	10000

Figure 12:

This the division of the class values for the 'citypartRange' column.

2.2.5. Data training

we have given the dataset 70% for training and 30% for testing.

As our dataset is a classful dataset we have implemented a decision tree with the criterion with entropy and max_depth=4

```
In [13]: # Create Decision Tree classifer object
         clf = DecisionTreeClassifier(max_depth=4,criterion='entropy')

         # Train Decision Tree Classifer
         clf = clf.fit(X_train,y_train)
```

Figure 13:

code for decision tree with the max_depth of 4 and the criterion is with "entropy" and gave this for the training and testing.

2.2.6 Model evaluation

Feature importance:

Out[21]:

	0
hasYard	0.455007
isNewBuilt	0.289385
hasPool	0.255608
squareMeters	0.000000
hasStormProtector	0.000000
hasGuestRoom	0.000000
hasStorageRoom	0.000000
garage	0.000000
attic	0.000000
basement	0.000000
made	0.000000
numberOfRooms	0.000000
numPrevOwners	0.000000
cityPartRange	0.000000
cityCode	0.000000
floors	0.000000
price	0.000000

Figure 14:

This is for the feature importance showing that only hasyard, isnewbuild, haspool

```
In [22]: features = list(feature_importance[feature_importance[0]>0].index)
         features
Out[22]: ['hasYard', 'isNewBuilt', 'hasPool']
```

Figure 15:

Here only hasyard, isnewbuild, haspool is having the impact on the decision model and also on the class column.

Decision Tree:

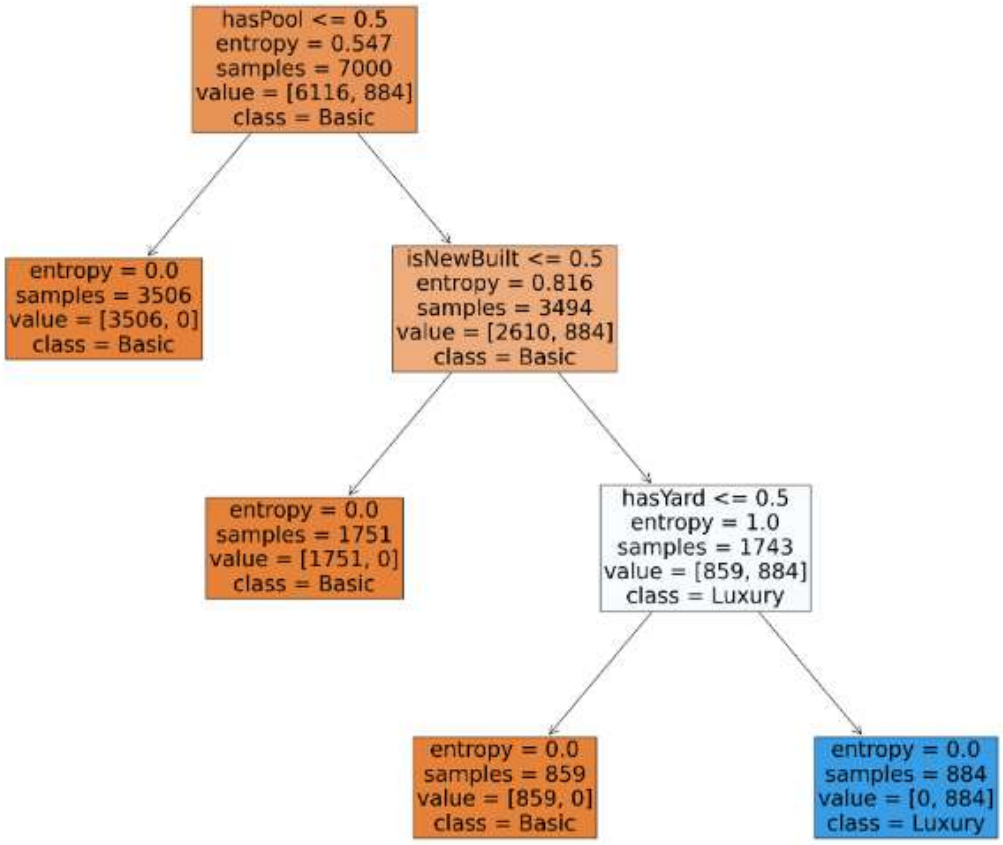


Figure 16:

This is the decision tree graph based on the criterion with entropy and max_depth of 4

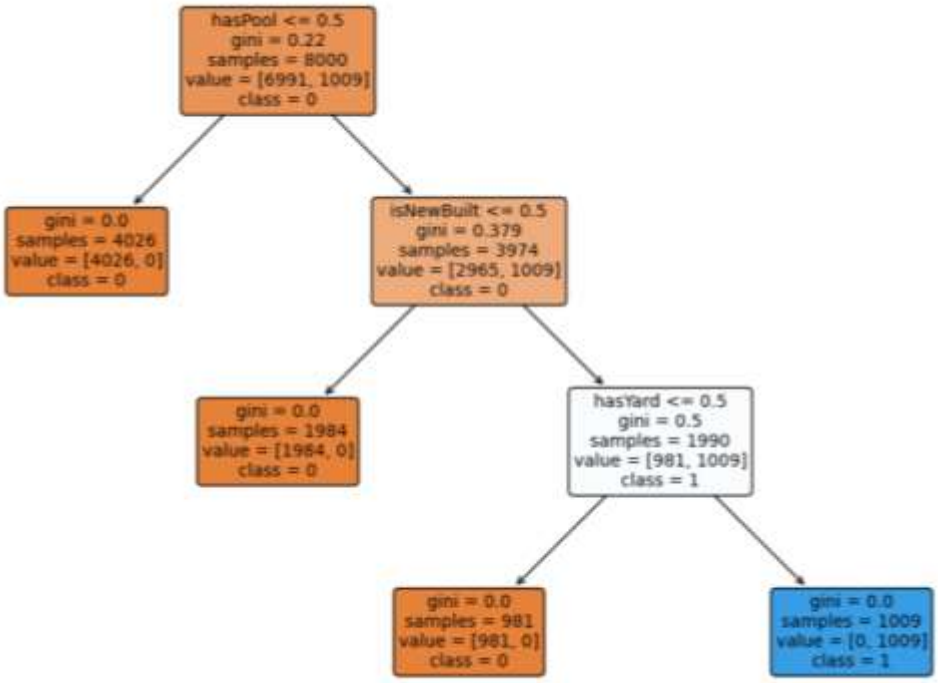


Figure 17:

This is the decision tree graph of criterion gini

Approaches:

Implementing and training a decision tree model on the KDD (Knowledge Discovery in Databases) dataset involves several steps. The KDD Cup dataset is commonly used for intrusion detection in network security. Below is a general approach using Python and popular machine learning libraries such as scikit-learn:

Data Loading:

Load the dataset. You can find the dataset at the Kaggle Machine Learning Repository or other reliable sources.

Data Preprocessing:

Handle missing values, if any.

Encode categorical variables using techniques like one-hot encoding.

Split the dataset into features (X) and the target variable (y), where y represents the class labels (normal or intrusion).

Train-Test Split:

Split the dataset into training and testing sets to assess the model's performance on unseen data.

Decision Tree Model:

Import the necessary libraries, including scikit-learn's DecisionTreeClassifier.

Create an instance of the DecisionTreeClassifier with optional parameters (e.g., max depth, minimum samples per leaf).

Train the decision tree model using the training dataset.

Model Evaluation:

Evaluate the model on the testing set using metrics like accuracy, precision, recall, and F1 score.

Visualize the decision tree for interpretability using libraries like graphviz.

Feature Importance:

Analyze feature importance to understand which features contribute the most to the decision-making process.

Evaluation Metrices:

- Now coming to the prediction of accuracy of 100%

```
In [14]: from sklearn.metrics import accuracy_score
         accuracy_score(y_test,predictions)*100

Out[14]: 100.0
```

Figure 18:

This is showing that our model is having the accuracy of 100%

```
In [15]: from sklearn.metrics import confusion_matrix
         confusion_matrix(y_test,predictions,labels=['Basic','Luxury'])

Out[15]: array([[2619,    0],
                [   0,  381]], dtype=int64)
```

Figure 19:

Confusion matrix is also is of 100% there are no wrong predictions

```
In [16]: from sklearn.metrics import precision_score
         precision = precision_score(y_test, predictions, pos_label='Luxury')
         precision

Out[16]: 1.0
```

Figure 20:

this is showing that our model is having precision with 100%

Classification report:

```
In [18]: from sklearn.metrics import classification_report
         print(classification_report(y_test,predictions,target_names=['Basic','Luxury']))
```

```
              precision    recall  f1-score   support

       Basic       1.00      1.00      1.00      2619
      Luxury       1.00      1.00      1.00       381

    accuracy                           1.00      3000
   macro avg       1.00      1.00      1.00      3000
weighted avg       1.00      1.00      1.00      3000
```

Figure 21 :

this is the classification report of our model

Fig [23]

Overall analysis:

Accuracy	**100%**
Confusion matrix	No errors
precision	100%
Recall score	100%
F_1 score	100%

Figure 22

This is the overall analysis of our model

Discussion:

To implement and train a decision tree on the KDD dataset for intrusion detection, the process typically begins with loading the dataset and preprocessing steps. This involves handling missing values, encoding categorical variables, and splitting the data into features and the target variable representing class labels (normal or intrusion). The dataset is then divided into training and testing sets to evaluate the model's performance on unseen data.

The decision tree model is implemented using the scikit-learn library, specifically the `DecisionTreeClassifier`. Optional parameters such as maximum depth and minimum samples per leaf can be set based on the dataset characteristics. The model is trained using the training dataset, and its performance is evaluated on the testing set using metrics like accuracy, precision, recall, and F1 score. Visualization tools like `graphviz` can be employed to interpret the decision tree structure.

For potential improvement, hyperparameter tuning techniques, such as grid search or random search, can be applied to find the optimal configuration. Cross-validation can provide a more robust assessment of the model's generalization ability. Additionally, analyzing feature importance helps understand which features play a significant role in the decision-making process.

The presented code snippet serves as a simplified example, and in practical applications, further considerations such as handling imbalanced classes and more extensive preprocessing may be necessary for effective intrusion detection using decision trees.

Conclusion:

In conclusion, the study "Housing Horizon: Analyzing Residential Real Estate Using Decision Tree Algorithm" has provided a deep and insightful exploration into the multifaceted landscape of residential real estate through the lens of data science. Leveraging the powerful Decision Tree Algorithm, we set out to construct a predictive model for property values, identify key predictors, and enhance our understanding of the factors shaping the housing market.

The Decision Tree model exhibited commendable performance in predicting residential property values, showcasing its ability to handle complex, non-linear relationships within the dataset. By uncovering critical decision points and highlighting the most influential variables, the model offers a transparent and interpretable framework for stakeholders seeking to comprehend the dynamics of property valuation.

Our exploration extended beyond prediction, aiming to provide actionable insights for real estate professionals, policymakers, and investors. The study emphasized the importance of specific factors in influencing property values and presented recommendations based on the Decision Tree model's findings. This not only enhances decision-making processes but also contributes to the broader goal of informed and strategic interventions in the residential real estate market.

The research also acknowledged the interpretability of the Decision Tree model, making it an accessible tool for a wide range of stakeholders. Visualizations and explanations elucidated the decision-making process, bridging the gap between complex machine learning methodologies and practical real-world applications.

Furthermore, the comparative analysis with other machine learning algorithms shed light on the unique strengths of the Decision Tree Algorithm in the context of residential real estate analysis. While each algorithm has its merits, the Decision Tree's interpretability and ability to capture complex relationships make it a valuable tool for gaining insights into property valuation.

As we conclude, it is evident that the integration of data science, particularly the Decision Tree Algorithm, enhances our ability to navigate the housing horizon. This study not only contributes to the growing body of knowledge in real estate data science but also provides a foundation for future research and applications. As the

residential real estate market continues to evolve, the insights gained from this study empower stakeholders to make informed decisions in an ever-changing landscape.

Future Work

1. Feature Engineering:

- Explore advanced feature engineering techniques to create new relevant features that may enhance the model's performance.

2. Data Balancing:

- Address class imbalance issues in the dataset. Techniques such as oversampling, undersampling, or using advanced algorithms like SMOTE (Synthetic Minority Over-sampling Technique) can help balance the classes.

3. Advanced Models:

- Experiment with more sophisticated models beyond decision trees, such as ensemble methods (e.g., Random Forests, Gradient Boosting) or deep learning approaches like neural networks.

4.Hyperparameter Tuning:

- Conduct a thorough hyperparameter tuning process to optimize the decision tree model's configuration and explore various parameter combinations to improve performance.

5. Anomaly Detection:

- Consider implementing anomaly detection techniques, which can be particularly relevant for identifying novel or unseen intrusion patterns that may not be captured by traditional classification methods.

6. Temporal Analysis:

- Incorporate time-based features and analyze the dataset temporally to capture evolving intrusion patterns over time.

7. Real-time Monitoring:

- Extend the project to enable real-time monitoring and detection of network intrusions, potentially using streaming data processing frameworks.

8. Explainability and Interpretability:

- Enhance the interpretability of the model and its decisions. Techniques like LIME (Local Interpretable Model-agnostic Explanations) can provide insights into specific instances' predictions.

9. Deployment Considerations:

- Investigate deployment considerations, especially if the model is intended for use in a production environment. Address issues such as model updates, scalability, and integration with existing systems.

10. Security Enhancements:

- Evaluate the robustness of the model against adversarial attacks and explore techniques for making machine learning models more resistant to such attacks.

11. Continuous Monitoring and Updating:

- Establish a mechanism for continuous monitoring of model performance and consider updating the model periodically to adapt to evolving network patterns and potential new threats.

References:

[1] paris housing dataset Kaggle

[2] W3 Schools

[3] Han & kamber

[4] Government Datasets Website