ANDROID-BASED HOUSE RENT PREDICTION USING MACHINE LEARNING

A Project Report submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted by

P. Praneet - 221910301043

Ch. Aryan Reddy - 221910301017

K. Vivek Reddy - 221910301032

P. Sree Paada Reddy - 221910301044

Under the esteemed guidance of

Dr. Mahammed Rafi D.

Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE &ENGINEERING

GITAM

(Deemed to be University)

HYDERABAD

MARCH 2023

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM INSTITUTE OF TECHNOLOGY

GITAM

(Deemed to be University)

DECLARATION

I/We, hereby declare that the project report entitled "**ANDROID-BASED HOUSE RENT PREDICTION USING MACHINE LEARNING**" is an original work done in the Department of Computer Science and Engineering, GITAM Institute of Technology, GITAM (Deemed to be University) submitted in partial fulfillment of the requirements for the award of the degree of B.Tech. in Computer Science and Engineering. The work has not been submitted to any other college or University for the award of any degree or diploma.

Date:

Registration No(s).	**Name(s)**	**Signature(s)**
221910301043	**P. Praneet**	
221910301017	**Ch. Aryan Reddy**	
221910301032	**K. Vivek Reddy**	
221910301044	**P. Sree Paada Reddy**	

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM SCHOOL OF TECHNOLOGY

GITAM

(Deemed to be University)

CERTIFICATE

This is to certify that the project report entitled "**ANDROID-BASED HOUSE RENT PREDICTION USING MACHINE LEARNING**" is a bonafide record of work carried out by **P. Praneet (221910301043), Ch. Aryan Reddy (221910301017), K. Vivek Reddy (221910301032), P. Sree Paada Reddy (221910301044)** students submitted in partial fulfillment of requirement for the award of degree of Bachelors of Technology in Computer Science and Engineering.

Project Guide	Project Coordinator	Head of the Department
Dr. Mahammed Rafi D.	**Dr. S Aparna**	**Prof. S. Phani Kumar**
Assistant Professor	**Assistant Professor**	**Professor & HOD**
Dept. of CSE	**Dept. of CSE**	**Dept. of CSE**

ACKNOWLEDGEMENT

Our project would not have been successful without the help of several people. We would like to thank the personalities who were part of our project in numerous ways, those who gave us outstanding support from the birth of the project.

We are extremely thankful to our honorable Pro-Vice Chancellor, **Prof. D. Sambasiva Rao** for providing necessary infrastructure and resources for the accomplishment of our project.

We are highly indebted to **Prof. N. Seetharamaiah**, Principal, School of Technology, for his support during the tenure of the project.

We are very much obliged to our beloved **Prof. S. Phani Kumar**, Head of the Department of Computer Science & Engineering for providing the opportunity to undertake this project and encouragement in completion of this project.

We hereby wish to express our deep sense of gratitude to **Dr. S Aparna**, Project Coordinator, Department of Computer Science and Engineering, School of Technology and to our guide **Dr. Mahammad Rafi D.** , Assistant Professor, Department of Computer Science and Engineering, School of Technology for the esteemed guidance, moral support and invaluable advice provided by them for the success of the project.

We are also thankful to all the staff members of the Computer Science and Engineering department who have cooperated in making our project a success. We would like to thank all our parents and friends who extended their help, encouragement and moral support either directly or indirectly in our project work.

Sincerely,

P. Praneet 221910301043

Ch. Aryan Reddy 221910301017

K. Vivek Reddy 221910301032

P. Sree Paada Reddy 221910301044

TABLE OF CONTENTS

ABSTRACT

The main intention of this project was to develop a user-friendly android application capable of predicting the rental prices of properties across Indian metropolitan cities. Real estate is a $326.5 trillion market globally. Traditionally, approaches to calculate the rent prices have been very imprecise. Individual valuers would review different data sources and vaguely estimate contributing factors and market conditions to arrive at a valuation. Leveraging the growing abundance of data and by using Machine Learning algorithms, we can predict house rents faster and more accurately than traditional methods. This helps buyers, tenants, sellers, banks, and insurers to make more informed decisions.

The dataset consisting of features such as price, count of rooms and bathrooms, location, etc., was utilized for this project. The next step was to pre-process the data and then to apply various Machine Learning algorithms, which were evaluated by metrics such as Mean Absolute Error (MAE), Normalized Root Mean Squared Log Error(NRMSE) and Adjusted R-squared. Finally, the application was developed where the user could enter the data and get the predicted rent value of the house.

LIST OF FIGURES

1. INTRODUCTION

House rent is the payment made by a tenant to a landlord for the use of a rental property. Various factors such as location, size, number of bedrooms and bathrooms, layout, and city are considered to predict house rent. Regression analysis is used to build a model that predicts rent based on these factors. To create the model, we need a dataset of past rental prices that includes these factors. This model can be used to predict the rent of a new house based on its characteristics. It's important to be aware that the model is based on historical data and might not consider market changes, which could impact future rental pricing. Hence, the model should be utilized in conjunction with market knowledge and judgment to make wise decisions about rental prices.

1.1 MOTIVATION

House rent prediction is driven by several causes, including the necessity for reliable rental market data, the desire to rent or buy rental properties with knowledge, and the possibility to spot trends and patterns in the rental market that can help influence policy decisions.

Accurate rent forecasts can aid current and prospective tenants make housing-related budgets, plan expenses, and find locations within their price range. For landlords and property managers, rent forecasts can help determine rental rates that are competitive with the market while still generating a profit.

Authorities may additionally utilize rent prediction to spot patterns and trends in the rental market, which can help them decide on affordable housing policies and other actions that promote stability and affordability in the housing market. Overall, rent prediction can offer valuable info about the rental market, assisting consumers, businesses, and authorities in making sensible housing choices.

1.1.2 Factors that affect House Rent

1. **Location**: The location of the property is a crucial factor that affects the rental price. Properties in prime areas with easy access to transportation, schools, hospitals, shopping complexes, and other amenities tend to have higher rental prices than those in less desirable areas.

2. **House size and type**: The size and type of the property also play a significant role in determining the rental price.

3. **Condition of the House**: The property's condition can affect the rental price. Properties in excellent condition with modern amenities and furnishings typically have higher rental prices than those in poor condition.

4. **Supply and demand**: The demand and supply dynamics in the housing market can affect the rental price. If the demand for rental housing and the supply is low, rental prices tend to be higher, and vice versa.

5. **Economic conditions**: Economic conditions, such as inflation and interest rates, can affect the rental price. Inflation can cause rental prices to increase, while low-interest rates can make it easier for individuals to purchase homes, reducing the demand for rental properties.

6. **Government policies**: Government policies such as rent control, property tax, and zoning laws can affect the rental price. Rent control policies limit the amount landlords can charge for rent. At the same time, property tax and zoning laws can increase the cost of owning and managing rental properties, leading to higher rental prices.

Overall, the rental price of residential properties in Indian Metropolitan cities is affected by a complex set of factors that interact, making it challenging to accurately predict rental prices without leveraging machine learning algorithms and data analysis techniques.

1.2 IMPORTANCE

Predicting house rent value is important for several reasons such as

1. **Budgeting**: By predicting house rent value, tenants can budget their expenses better and plan their finances accordingly. This can help tenants avoid situations where they are unable to pay rent and face the risk of eviction.

2. **Negotiating**: Predicting house rent value can help tenants negotiate a fair rent amount with landlords. By clearly understanding market rent value, tenants can negotiate effectively and avoid overpaying for rental property.

3. **Investment**: It is important for real estate investors who want to purchase rental properties. By having a clear understanding of the market rent value, investors can make informed decisions about which properties to purchase and how much rent to charge.

4. **Risk Management**: Predicting house rent value can help landlords manage their risk. By setting the rent at an appropriate level, landlords can ensure that they are able to cover their costs and make a profit while avoiding the risk of having vacant properties or tenants who are unable to pay rent.

5. **Market Analysis**: Predicting house rent value can be important for market analysis. By analyzing trends in rental prices, authorities and researchers can gain insights into the housing market and make informed decisions about housing policy.

As a result, predicting the value of the house rent can assist renters, landlords, investors, authorities, and experts in making wise choices and effectively managing risk. It's a powerful component for evaluating the property market and making sure rental homes are priced fairly.

1.3 PROBLEM STATEMENT

Landlords, tenants, and real estate agents find it difficult to anticipate the rental cost of new houses due to the significant fluctuation and lack of transparency in the rental costs of residential properties. The housing market becomes inefficient as a result of potential property mispricing and poor decision-making. The objective of this project is to create an Android-based system for predicting house rent in Indian metropolitan cities, making use of machine learning algorithms to deliver accurate and reliable rental cost estimations and support efficiency and transparency in the housing market.

1.4 PROBLEM DEFINITION

House rent prediction is the process of estimating the expected rental cost for a residential property in a given location based on various factors such as the property type, location, size, amenities, and demand-supply dynamics.

The goal of the model is to provide a reliable estimate of the rental cost of a property to landlords, tenants, and real estate agents, and to help them make informed decisions about renting or leasing properties. The model should be able to generalize well to new properties in different locations and with different attributes, and should be robust to noise and outliers in the dataset.

To achieve this goal, the model should be trained using a supervised learning algorithm, such as linear regression, and decision trees and the dataset should be carefully pre-processed and cleaned to remove missing values, outliers, and irrelevant features. Feature engineering techniques such as one-hot encoding, scaling, and feature selection can be used to extract relevant information from the dataset and improve the performance of the model.

Evaluation of the model can be done using standard regression metrics such as mean squared error (MSE), root mean squared error (RMSE), mean absolute error (MAE), and R-squared (R2) score.

1.5 OBJECTIVES

1. Providing an easy-to-use and user-friendly interface for landlords, tenants, and real estate agents to access accurate and reliable rental cost estimates for residential properties in Indian Metropolitan cities.

2. Using machine learning algorithms to analyse large datasets of residential properties in Indian Metropolitan cities and extract relevant features that can be used to accurately predict the rental cost of new properties.

3. Enabling customers to look for properties according to their preferred location, size, and amenities.

4. Developing a reusable environment that permits integration with other real estates services, such as property management, leasing, and other regulatory and financial services, in order to offer a simple and seamless user experience.

5. Provide a more real-time prediction of rent as compared to existing approaches.

1.6 OUTCOMES

1. Enhanced accuracy and reliability of rental cost estimates: With the help of machine learning algorithms, the system can analyze large datasets of residential properties in Indian Metropolitan cities and extract relevant features that can be used to accurately predict the rental cost of new properties. This can assist landlords, tenants, and real estate agents in making better-educated decisions regarding renting or leasing properties.

2. Increased transparency and efficiency in the housing market: By providing users with access to accurate and reliable rental cost estimates, the system can help promote transparency and efficiency in the housing market. This can lead to better pricing decisions, more competition among landlords, and a more balanced supply-demand ratio.

3. Increased market penetration: An Android-based house rent prediction system can potentially reach a large number of users in Indian Metropolitan cities who have access to Android devices. This can help increase market penetration and adoption of the system and can potentially lead to a larger user base.

4. Improved decision-making: With accurate rental cost estimates and personalized recommendations, users can make more informed decisions about renting or leasing properties. This can help them save time and money, potentially leading to better outcomes in the housing market.

5. Enhanced competitiveness: With the integration of other real estate services, such as property management, leasing, and payment processing, the system can potentially provide a competitive advantage to real estate companies and agents who adopt the system.

1.7 LIMITATIONS

Some limitations of an Android-based house rent prediction system for Indian Metropolitan cities can be:

1. Limited data availability: The accuracy of the rental cost estimates may depend on the availability and quality of data on residential properties in Indian Metropolitan cities. If the dataset is small or biased, the predictions may not be reliable.

2. Inaccurate data input: The accuracy of the rental cost estimates may also depend on the accuracy of the data input by users. If users input incorrect or incomplete information about a property, the system may generate inaccurate predictions.

3. External factors: Rental prices for residential properties can be impacted by external factors such as economic conditions, changes in government policies, and fluctuations in the real estate market. The system may not be able to capture all of these external factors, which can lead to inaccurate predictions.

4. Privacy and security concerns: Collecting and analysing user data can raise privacy and security concerns. The system must ensure that user data is protected and comply with all applicable data protection laws and regulations.

5. Dependence on machine learning algorithms: The accuracy of the rental cost estimates depends on the performance of the machine learning algorithms used by the system. When algorithms are not properly trained or tuned, the predictions may not be accurate.

1.8 APPLICATIONS

The applications of an Android-based house rent prediction system for Indian Metropolitan cities can be:

1. Landlords: Landlords can use the system to estimate the rental cost of their properties and adjust their prices accordingly. They can also use the system to monitor market trends and adjust their rental prices based on demand and supply dynamics.

2. Tenants: Tenants can use the system to search for properties that meet their needs and budget. They can also use the system to compare different properties based on their predicted rental cost and make informed decisions about renting or leasing properties.

3. Government: The government can use the system to monitor the rental market trends and identify areas where affordable housing is needed. They can also use the system to evaluate the impact of their policies on the rental market and make informed decisions about future policies.

4. Real estate agents: Real estate agents can use the system to provide accurate rental cost estimates to their clients and help them make informed decisions about renting or leasing properties. They can also use the system to monitor market

trends and provide recommendations to their clients based on demand and supply dynamics.

5. Property managers: Property managers can use the system to monitor the rental prices of properties under their management and adjust their pricing strategy accordingly. They can also use the system to identify potential opportunities for rental income growth.

6. Real estate companies: Real estate companies can use the system to provide better services to their clients and gain a competitive advantage in the market. They can also use the system to improve their pricing strategy and optimize their rental income.

7. Property listing sites: These websites can use both the quoted price and the predicted price to show potential customers the actual valuation.

Overall, an Android-based house rent prediction system can benefit various stakeholders in the real estate market by providing accurate rental cost estimates and promoting transparency and efficiency in the housing market.

2. LITERATURE REVIEW

The majority of the literature review is based on Research publications that are accessible online, open access articles and textbooks. The goal of our literature review is to establish a strong foundation for machine learning usage such that the rent is precisely predicted.

The review of related studies and the feature engineering techniques applied in this study are provided in the literature study. Furthermore, since there is no direct way to judge regression algorithms, certain evaluation criteria that assess the effectiveness of the algorithms will be studied.

2.1 INTRODUCTION

The real estate sector has expanded significantly in recent years. There is an increase in demand for renting and buying homes as a result of urbanisation. Every year, there is a growth in the demand for homes, which indirectly drives up home prices. The issue arises when there are many factors, such as location and property demand, that could affect the house price. As a result, the majority of stakeholders, including buyers, developers, home builders, and the real estate industry, would like to know the precise characteristics or the accurate factors influencing the house price to assist investors in their decisions and to assist home builders in setting the house price. In addition to the fact that it has always been problematic for property owners, structures, and real estate.

Every year, there is a growth in the demand for homes, which indirectly drives up home prices. The issue arises when there are many factors, such as location and property demand, that could affect the house price. As a result, the majority of stakeholders, including buyers, developers, home builders, and the real estate industry, would like to know the precise characteristics or the accurate factors influencing the house price to assist investors in their decisions and to assist home builders in setting the house price.

The change has been implemented since the current rent indexes frequently aren't accurate and don't have a solid theoretical basis. As a part of the comparative rent control system, this has implications for the evaluation of rent control measures and legal challenges over rent price hikes.

We can predict the rent of the houses with ease using machine learning models. Machine learning models are created from historical data and are then used to forecast future data. Because of the growing population, there is a daily growth in the market for homes. Because there aren't enough jobs in rural areas, people are moving there for economic reasons. The demand for housing in cities is therefore rising as a result. Individuals who are unaware of the true cost of a certain residence in a city may encounter several difficulties. Thus, it's crucial to make a precise rent prediction.

2.2 EXISTING SYSTEM

2.2.1 Quang Truong, Minh Nguyen, Hy Dang, Bo Mei, "Housing Price Prediction via Improved Machine Learning Techniques" [4]

The work done in [4] has been primarily focused on developing algorithms that can correctly predict the housing costs depending on a variety of variables, such as location, size, and amenities. They have made a thorough examination of huge datasets and produced incredibly accurate predictions using cutting-edge machine learning methods, such as deep learning and neural networks. The researchers have used their algorithms for both residential and commercial buildings in a variety of housing markets.

The results of [4] showed that the existing methodology outperforms individual machine learning models and traditional regression models in predicting housing prices. The authors attribute this improvement to the combination of multiple models and the use of advanced feature selection techniques.

The dataset used in [4] belongs to "Housing Price in Beijing" which contains more than 300,000 data with 26 variables representing housing prices between 2009 and 2018. These variables, which served as features of the dataset, were then used to predict the average price per square meter of each house. Their next step was to investigate missing data. Any observations which had missing values were also removed from the dataset. The few feature engineering processes which were done to cleanse the dataset were:

- Remove attributes indicating the number of kitchens, bathrooms, and drawing rooms considering their ambiguity.
- Set the number of bedroom to [1,4]
- Adding an attribute "distance" indicating the distance of the house from the center of Beijing.
- Replace attribute "constructionTime" with attribute "age" by deducting the year that the house was constructed from their current year (2019).
- Set minimum values for attributes "price" and "area".
- Split the attribute "floor" into attributes "floorType" and "floorHeight

While their numerical values were standardized and categorical values were one-hot-encoded. After being processed, the dataset included 58 features. Since most of the features are categorical, it is evident that the variance almost converges at the 30th component. Then, the dataset was split into a training set and test set with a ratio of 4 : 1. Evaluation function used in this paper is Root Mean Squared Logarithmic Error (RMSLE) and the algorithms used in this paper are Random Forest, XGBoost, Light Gradient Boosting Machine and Stacked Generalisation.

Their training set's best results go to Random Forest, and their test set's best results went with stacked generalization regression. The Random Forest performed well on the one-hot-encoded dataset as 49 of the 58 features were boolean values. Despite this, Random Forest had performed well on untried data due to its propensity for overfitting. Although XGBoost and LightGBM were not prone to overfitting, the accuracy of their predictions on both training and test data was inferior to Random Forest.

2.2.2 Gabriel M. Ahlfeldt, Stephan Heblich, Tobias Seidel, "Micro-geographic property price and rent indices" [1]

The objective of [1] was to create a panel of micro-geographic house price and housing rent indices using an algorithmic methodology that uses spatial methodologies to get beyond the drawbacks of sparse data information on property transactions. Their method does not rely on administrative boundaries because it is purely point-pattern based and thus applicable to arbitrary geographical units provides details on pricing or rents, location, dates of transactions, and property attributes. The data is not suitable for this project as there isn't enough satellite data available for Indian cities showing houses.

[1] has made use of comprehensive market data for Germany to construct house price and rent indices for regional labour markets and postcode areas in order to demonstrate the efficiency of prediction methods that can be applied. The models implemented cannot be directly applied due to absence of the geographic data. The data set also includes a text description, which they used to extract a variety of other features, along with asking prices and the standard property features (such as price, date, floor space, etc.). The prices of geolocated real estate transactions are converted in [1] study into a mix-adjusted balanced-panel house price index for any given spatial unit.. Their algorithm's strength is its ability to mix nonparametric and parametric calculation techniques in order to deliver an exact local fit in situations where the data are plentiful and accurate extrapolations in situations where the data was scarce. The absence of transactional data makes it non-implementable for our project.

The result of [1] was shown in two applications. Application 1 which is Local Labour Market(LLM) and Application 2 which is Postcodes.

The success of [1] seemed immense as it linked the price-to-rent ratio in major cities to foreign investment given the substantial capital inflows into the German real estate market, which served as one of the few "safe havens" through the U.S. subprime mortgage crisis and the European sovereign debt crisis. The differential influx of the foreign capital would tend to accentuate geographical differences in the price-to-rent

ratio especially as the foreign investments were more likely towards larger cities, whether it be because these markets are more liquid, less fragmented in terms of ownership, or simply because they are visible more boldly on the map.

2.2.3 Other research

The work done in [7] has given insights about using neural networks for prediction of real estate pricing, however significant attention needs to be given to the speed at which the prediction happens while neural networks are being implemented. Additionally, the model is being used for Singapore based properties which is a small nation with completely digitized records, it is very possible to accurately predict the prices.

The research in [2] suggests According to the trained model to predict house sale prices, it can be extended to K neighbour's regression. This approach can be used for house rent prediction as well. In order to achieve accurate predictions, it is essential to have correct data. Their study specifies the need for uniform and accurate data for mass appraisal models to estimate real rent prices. Regression models can be used to yield fitted values and residuals that predict the response and errors in the predictions.

Several other studies have also proposed models for predicting house rent prices. For instance, a study proposed a model based on machine learning algorithms such as Decision Tree Regressor (DTR) and Random Forest Regression (RFR). Overall, the use of machine learning and regression models has shown promising results in the prediction of house rent prices. However, it is important to ensure the accuracy and correctness of the data used for the models to achieve accurate predictions.

2.3 LIMITATIONS OF EXISTING SYSTEMS

While the existing systems are robust, there are key elements which have not been addressed by the current systems.

- Existing Models have extensively focussed on the developed countries like USA, China where the digitalization has been more intensive and public data is very easily available and accessible.

- Existing systems may not perform well in certain markets or for certain types of rental properties, which can result in inaccurate or unreliable rent estimates in these situations.

- The existing techniques have been used in stable markets where home inflation is very modest and has no impact on rent.

3. PROBLEM ANALYSIS

3.1 PROBLEM IDENTIFICATION

The existing method of calculating rent in Indian metropolitan areas is often time-consuming, inaccurate, and ambiguous. Determining a fair rental rate for a property is a challenge for both tenants and landlords. In addition, the rental market is dynamic and ever-evolving, making it difficult to stay on top of current rental trends and patterns. So, there is a need for a credible estimate of the cost to rent a property that can be given to tenants, landlords, and real estate agents and is based on pertinent aspects including location, property features, and current economic conditions. By offering a simple and precise method to estimate rent, the machine learning-based house rent prediction system seeks to address these problems.

3.2 PROBLEM STAKEHOLDERS

The stakeholders for an Android-based house rent prediction system using machine learning include:

1. **Tenants**: Tenants are the primary beneficiaries of the system, as they can use the app to estimate the rental cost of a property accurately and negotiate with landlords.

2. **Landlords**: Landlords can benefit from the system by using it to set competitive rental prices for their properties based on market conditions and property features.

3. **Real estate agents**: Real estate agents can leverage the app to provide more accurate rental estimates to their clients and improve their overall customer experience.

4. **Property investors**: Property investors can use the system to evaluate potential rental income and make informed decisions on property acquisitions and sales.

5. **Government agencies**: Government agencies responsible for housing policies and regulations can use the app to monitor and analyze housing market trends and make data-driven policy decisions.

6. **Researchers**: Researchers interested in housing market dynamics can leverage the app's data to conduct research and analysis on rental trends and patterns in Indian Metropolitan cities.

Overall, an Android-based house rent prediction system using machine learning has the potential to benefit a wide range of stakeholders by providing accurate rental cost estimates, improving transparency in the housing market, and promoting more efficient decision-making.

3.3 BOUNDARIES

1. **Geographical Boundary** : The application should be limited to a specific geographical region, such as a city or a country, depending on the availability of relevant data.

2. **Use Case Boundary**: The application should be designed specifically for house rent prediction and should not be used for other purposes, such as property valuation or real estate related financial advice.

3.4 SOFTWARE REQUIREMENTS SPECIFICATION (SRS)

3.4.1 INTRODUCTION

This document outlines the software requirements for a machine learning-based application that predicts the rent of a house based on its location, layout, and other attributes.

3.4.2 USER REQUIREMENTS

Functional Requirements

1. Allow users to enter the location, layout, and other attributes of a house to obtain a predicted rent value.
2. Provide an intuitive user interface that is easy to navigate and use.
3. Allow users to clear their previous inputs and predicted rent values.

Non-Functional Requirements

1. The application should be responsive and provide fast predictions.
2. The application should have a modern and visually appealing design.
3. The application should be secure and protect user data.
4. The application should be compatible with a wide range of devices and web browsers.

3.4.3 SYSTEM REQUIREMENTS

Hardware Requirements

The application should run on any system with a modern operating system.

Software Requirements

The application should have the ability to use machine learning libraries to deploy the prediction model.

3.4.4 DESIGN REQUIREMENTS

1. The application should have a modern and visually appealing design that is easy to navigate and use.
2. The application should use clear and concise language to communicate with the user.
3. The application should follow web design best practices and be consistent with other web applications.

3.4.5 TESTING REQUIREMENTS

1. The application should be thoroughly tested for functionality, performance, and security.
2. The application should be tested on a range of devices and web browsers.
3. The application should be tested with a variety of user inputs to ensure accurate predictions.

3.4.6 ACCEPTANCE CRITERIA

1. The application should accurately predict the rent of a house based on its location, layout, and other attributes with a high degree of accuracy.
2. The application should be easy to use and navigate, with clear and concise language and error messages.
3. The application should be fast and responsive, with quick prediction times.
4. The application should protect user data and be secure.
5. The application should be compatible with a wide range of devices and web browsers.

3.5 REQUIREMENTS

3.5.1 MINIMUM HARDWARE REQUIREMENTS

- System: Intel i3 2.2 GHz.
- Hard Disk : 50 GB.
- Display: 13-inch color monitor.
- Ram : 2 GB

3.5.2 MINIMUM SOFTWARE REQUIREMENTS

- Microsoft Windows OS – Windows 7 and later
- The latest version of Python 3
- Jupyter Notebook
- Android Emulator for version 12

3.6 CONSTRAINTS

1. **Hardware Constraints**: The application should be designed to work with a range of screen sizes and resolutions.
2. **Time and Performance Constraints**: The application should provide fast and accurate rent predictions, ideally within a few seconds, and the application should not consume an excessive amount of device resources, such as CPU or battery while running.

4. SYSTEM DESIGN

4.1 SYSTEM ARCHITECTURE

The process of planning and detailing how a system's components will behave, interact, and be organized is known as system architecture. We outline the various interfaces, subsystems, and components that make up the system as well as how they interact with one another to accomplish the project goals.

The overall structure of the system, including its functional and non-functional requirements, design limitations, and performance goals, is defined by the system architecture. Also, it describes the hardware, software, communication networks, and data storage systems that make up the system's components.

In order to guarantee that the system is well-designed, effective, scalable, and maintainable, the system architecture is essential. In addition to helping stakeholders comprehend the system's capabilities, constraints, and dependencies, it offers a blueprint for system development, integration, and testing.

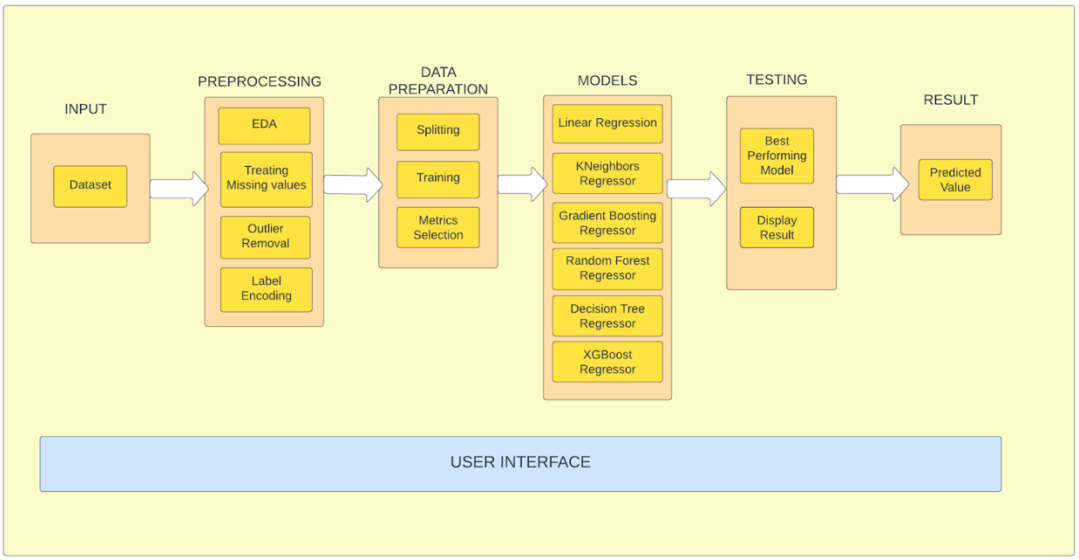


Fig 4.1 System Architecture

4.2 PROJECT WORKFLOW

Project workflow refers to the sequence of tasks and activities that are necessary to complete a project from start to finish. It includes all the steps involved in planning, executing, monitoring, and closing a project and the relationships between these steps. The project workflow is a crucial component of project management, providing a framework for organizing and completing projects effectively, efficiently, and with high quality.

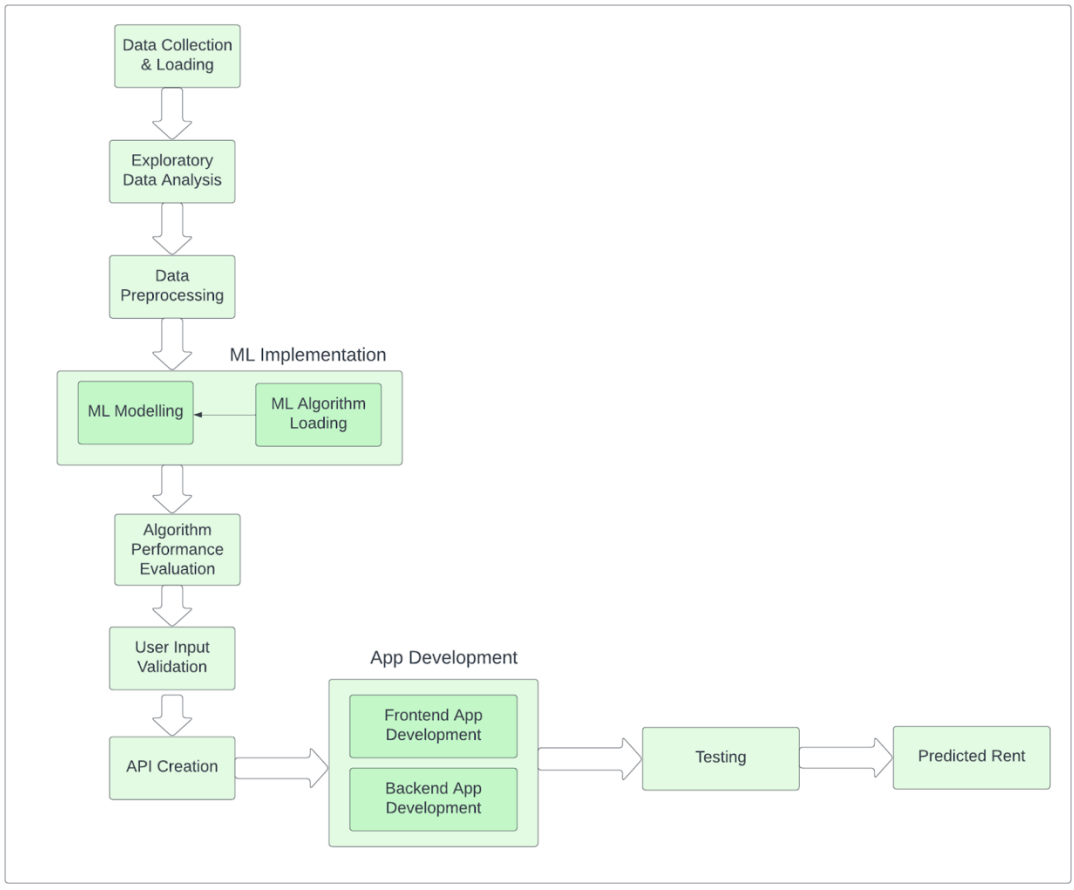


Fig 4.2 Project Workflow

4.3 UML DIAGRAMS

UML (Unified Modelling Language) diagrams are a graphical representation of a software system or application that depicts various aspects of the system's structure, behavior and interactions between its components. UML diagrams are widely used in software development to help developers and stakeholders visualize, design, and document a system's architecture and functionalities.

4.3.1 USE CASE DIAGRAM

In a use case diagram, we represent the actors (users, devices, or other systems) as stick figures, and use cases (the tasks or actions the system performs) are represented as ovals or rectangles. Lines connecting the actors and use cases represent the interactions or transactions between them.

Fig 4.3 Use Case Diagram

4.3.2 SEQUENCE DIAGRAM

A sequence diagram shows the messages between the objects as horizontal arrows and the items as vertical lifelines. The messages show the order in which the items communicate with one another, and the lifelines display each object's lifespan. The arrows may also represent the passage of time or the length of the message.

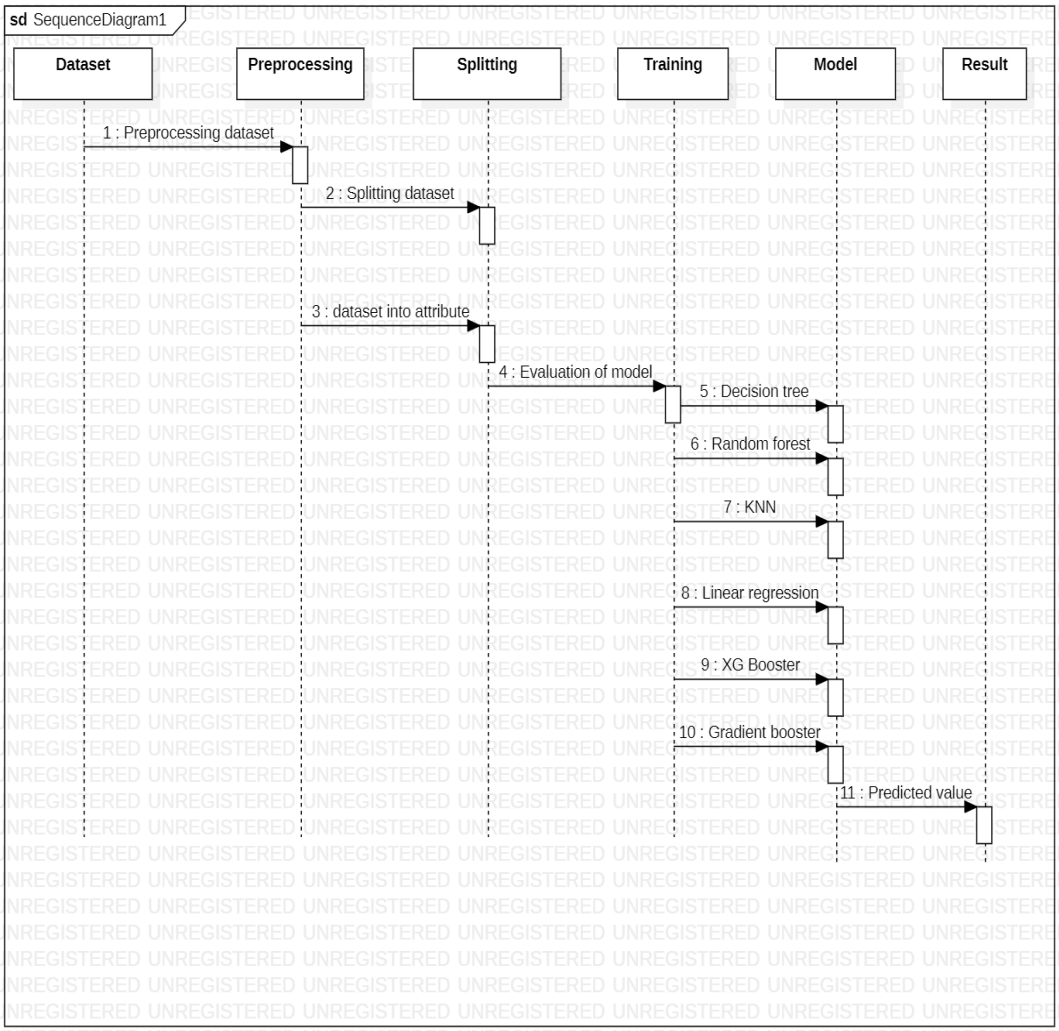


Fig 4.4 Sequence Diagram

4.3.3 COMPONENT DIAGRAM

In a component diagram, the components are indicated as boxes or rectangles, and the connections between them are represented by lines or connectors. The parts could be software modules, libraries, subsystems, or other system functional elements.

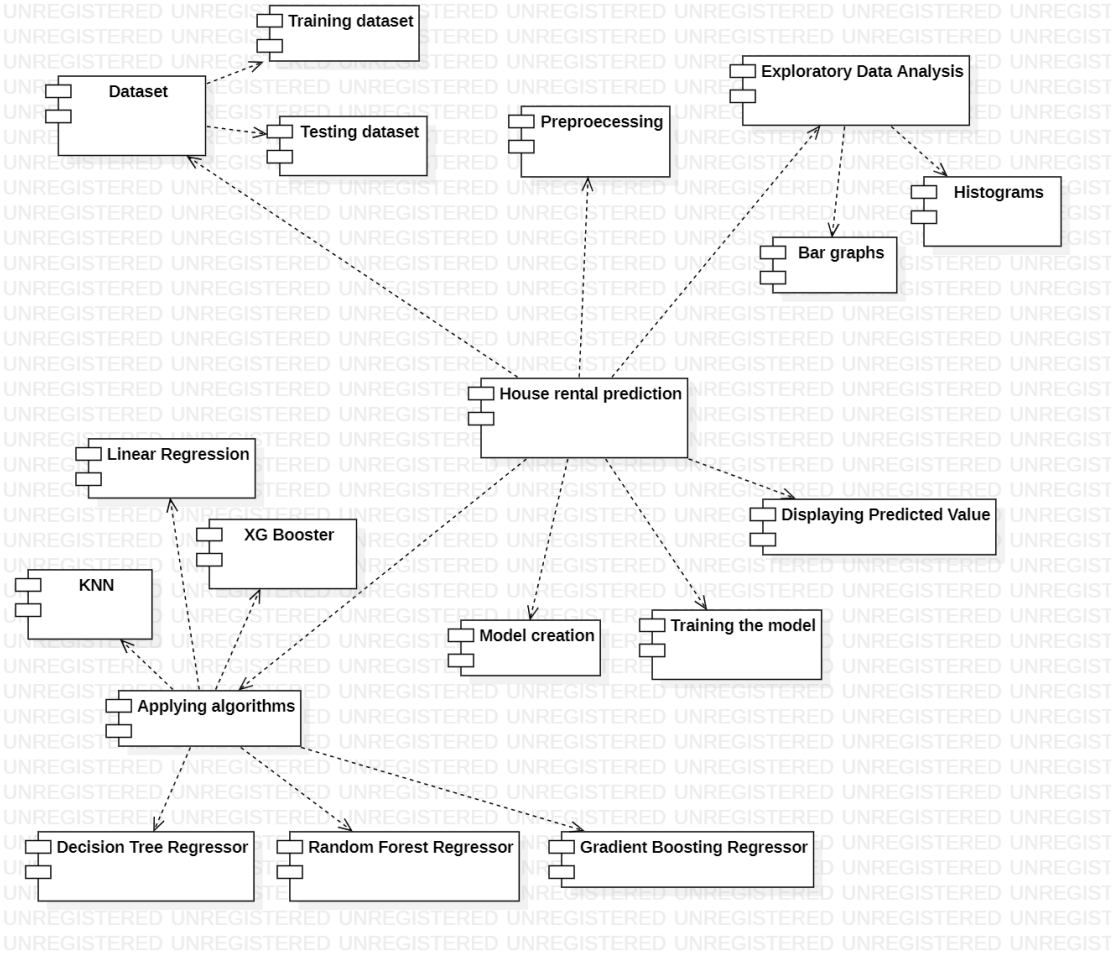


Fig 4.5 Component Diagram

4.3.4 CLASS DIAGRAM

In a class diagram, classes are represented as boxes with the class name at the top, the attributes in the middle, and the methods at the bottom. The attributes and methods are shown as a list, and the relationships between classes are represented as lines connecting the boxes.

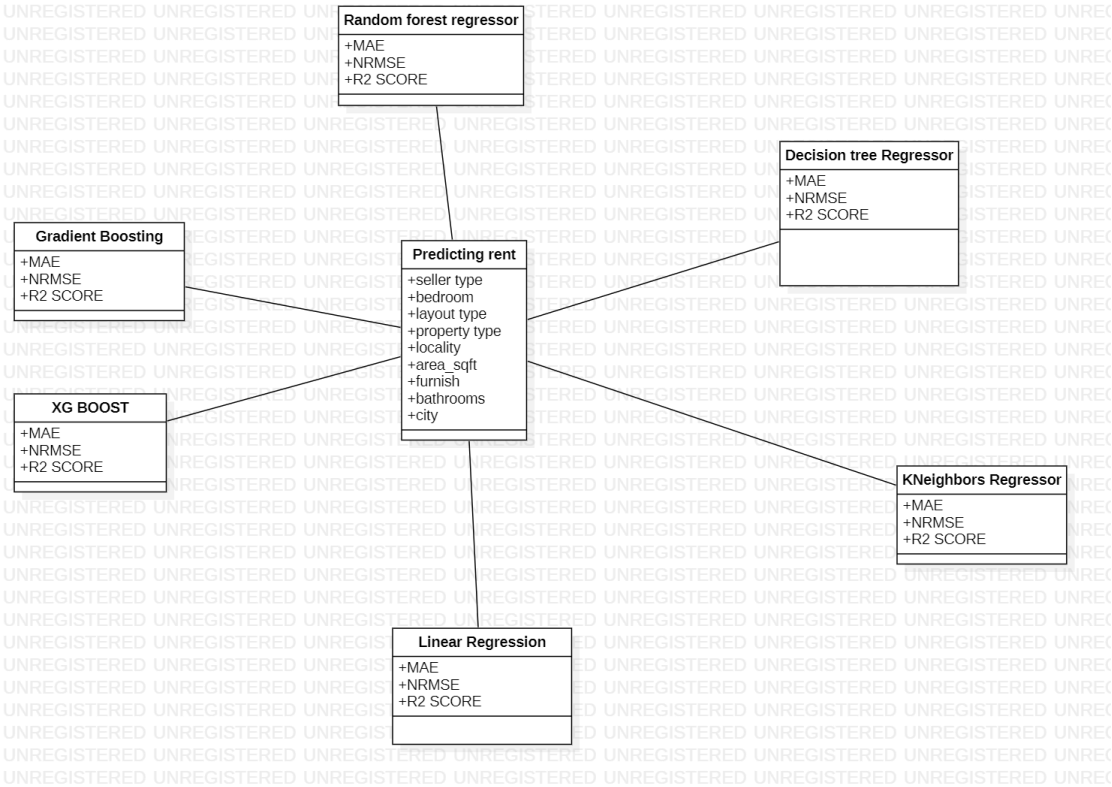


Fig 4.6 Class Diagram

4.3.5 DATA FLOW DIAGRAM

A Data Flow Diagram(DFD) shows the system or process as a circle or square, and the data flows are displayed as arrows connecting the two. Rectangles are used to represent data storage, whereas parallelograms are used to represent external entities. Annotations that describe the data being transferred, the procedures that modify the data, and the entities that produce or use the data are all included in the DFD.

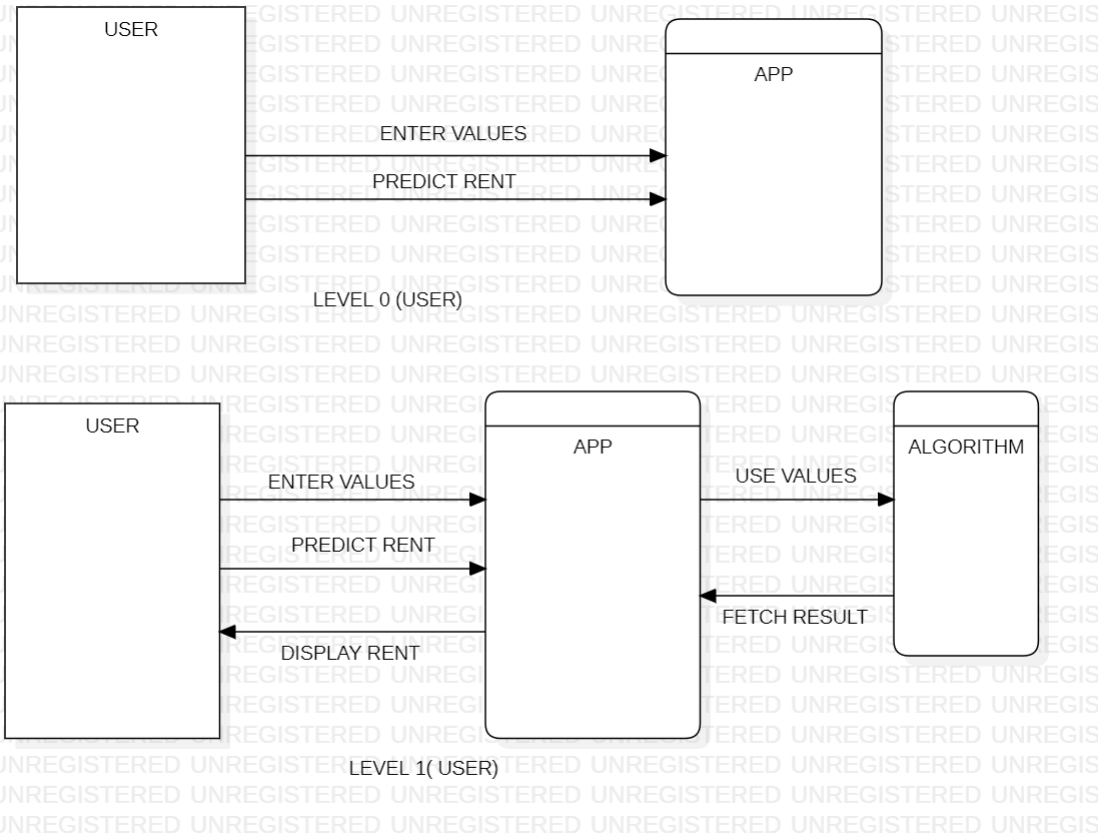


Fig 4.7 Data Flow Diagram

4.3.6 STATE CHART DIAGRAM

In a state chart diagram, the states are represented as rounded rectangles and the transitions between states as arrows. The events or stimuli that trigger the transitions are shown on the arrows, and the actions or behaviours associated with each state are shown inside the rectangles.

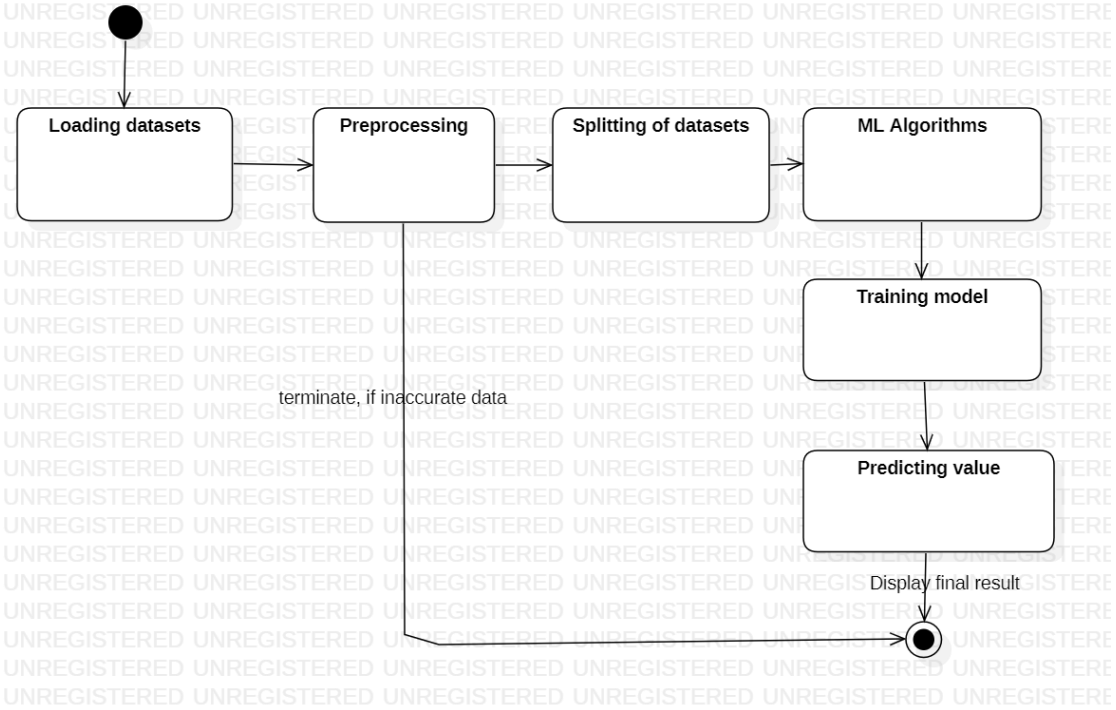


Fig 4.8 State Chart Diagram

4.4 OVERVIEW OF TECHNOLOGIES

4.4.1 PYTHON

Python is a programming language that is used in various fields, such as data science, machine learning, web development, and scientific computing. It is well-regarded for being straightforward and easy to read, making it simple to learn and use for newcomers. Python has a simple and clear syntax compared to other languages, necessitating fewer lines of code to complete tasks. Moreover, Python provides a wide selection of libraries and frameworks, such as Flask for web development, Sci-kit learn, and TensorFlow for machine learning, as well as NumPy and Pandas for data manipulation. Users can create complicated apps quickly and simply with these tools and frameworks.

4.4.2 JUPYTER NOTEBOOK

Jupyter Notebook is an open-source web application that allows users to create and share documents that contain live code, equations, visualizations, and narrative text. It is widely used in data science, scientific computing, and machine learning. It is extensively used in machine learning, scientific computing, and data science. It supports more than 40 programming languages, including Python, R, Julia, and Scala, and enables interactive, iterative code writing and execution. It enables the integration of well-known libraries like NumPy, Matplotlib, and Scikit-Learn and offers a simple approach to exploring, analyze, and visualize data.

4.5 ALGORITHMS

Linear Regression

Linear regression is a statistical technique used to model the relationship between a dependent variable and one or more independent variables. It is a simple and widely used method for predicting a quantitative response based on one or more predictor variables.

The basic idea behind linear regression is to fit a straight line to a set of data points such that the line minimizes the distance between the predicted values and the actual values. The line is represented in the form of :

$y = mx + b$

where y is the dependent variable, x is the independent variable, 'm' is the slope of the line, and 'b' is the y-intercept. The goal of linear regression is to find the values of 'm' and 'b' that best fits the data.

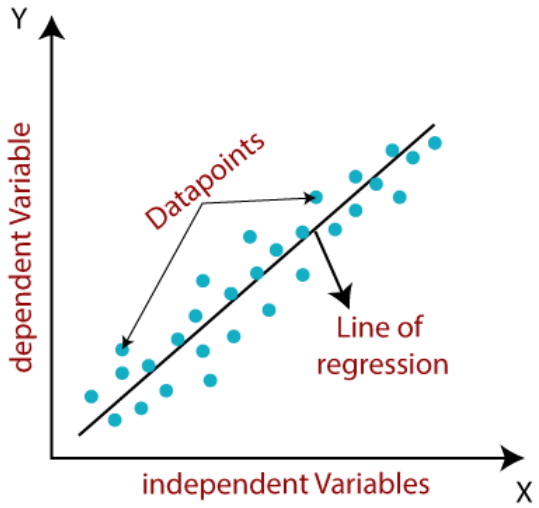


Fig 4.9 Linear Regression

K Neighbors Regressor

K Neighbors Regressor is a machine learning algorithm that can be used for regression tasks. It is a type of instance-based learning where the algorithm does not create a model during training but instead stores the training dataset and uses it during prediction.

The K Neighbors Regressor algorithm works by finding the k-nearest neighbour in the training dataset to a new input data point, where 'K' is a hyperparameter that is set by the user. The algorithm then predicts the output value of the new data point by taking the average of the output values of its k-nearest neighbors.

K Neighbors Regressor is a non-parametric algorithm, which means it does not make any assumptions about the underlying distribution of the data. It can handle both linear and non-linear relationships between the input and output variables.

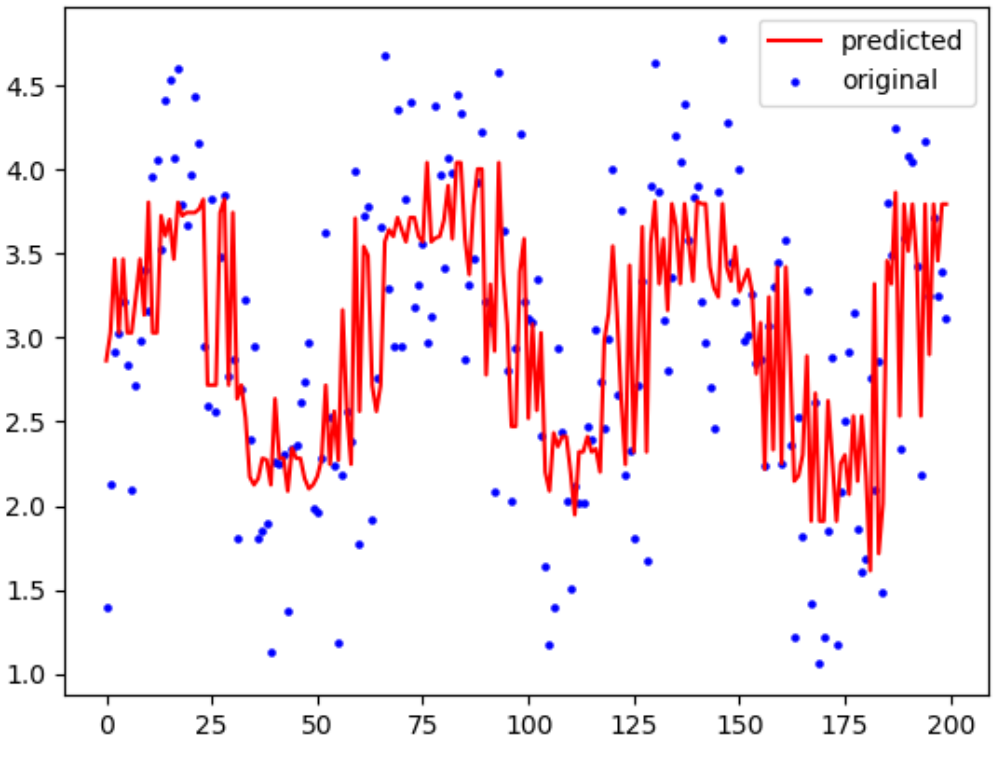


Fig 4.10 KNeighbors Regressor

Gradient Boosting Regressor

Gradient Boosting Regressor is a machine learning algorithm used for various regression tasks. It is a type of boosting algorithm where a set of weak models are combined to form a stronger model.

Decision trees are iteratively added to the model by the Gradient Boosting Regressor technique, with each tree learning from the residual mistakes of the preceding trees. The process begins with a base model, which is typically a straightforward model like a decision tree or a linear regression model. After that, it adds the data and trains a new model to forecast the residual errors of the old model. As it can learn the significance of each feature and only use the most essential ones, it can automatically manage feature selection.

One of the main disadvantages of Gradient Boosting Regressor is that it can be computationally resource-intensive and time-consuming for large datasets since it requires training a large number of decision trees. It can also be prone to overfitting if the model is too complex or if the training dataset is too small.

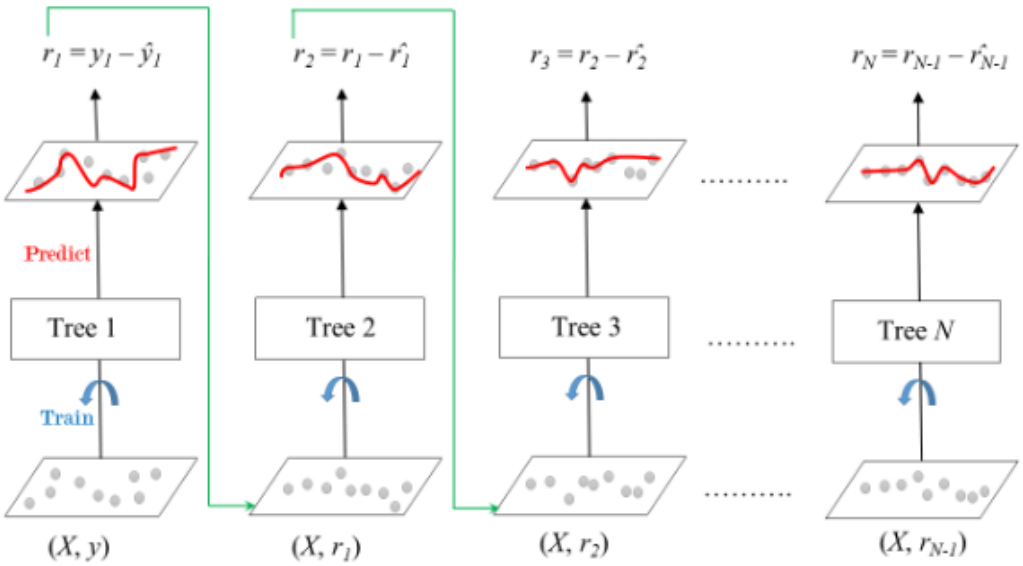


Fig 4.11 Gradient Boosting Regressor

Decision Tree Regressor

Decision Tree Regressor is a machine learning algorithm that can be used for regression tasks. It is a type of supervised learning algorithm that builds a decision tree from the training data to predict the output value of the new input data point. The Decision Tree Regressor algorithm works by recursively splitting the training data into subsets based on the values of the input features.

On each node of the tree, the algorithm chooses the optimum feature to split the on depending on factors like information gain or mean squared error (MSE). Until the tree meets a stopping requirement, such as a maximum depth or a minimum quantity of samples per leaf node, the procedure is repeated. A straightforward and understandable method known as a decision tree is capable of handling both linear and non-linear interactions between the input and output variables. Since it is based on the data's local structure, it can also manage noisy data and outliers.

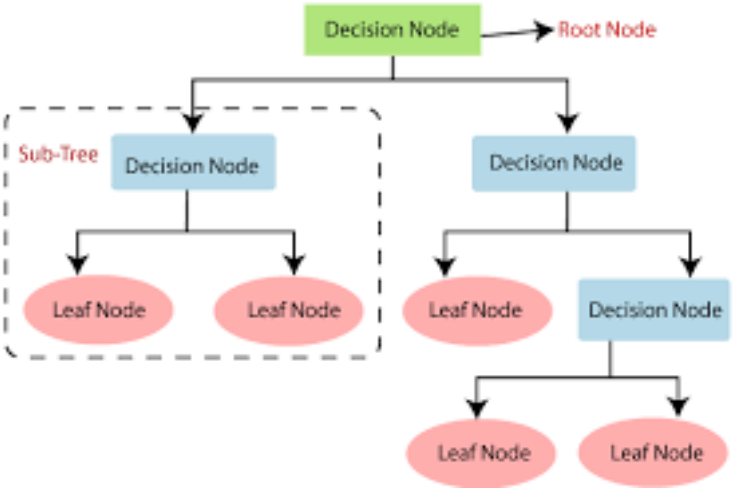


Fig 4.12 Decision Tree Regressor

Random Forest Regressor

One of the most efficient algorithms for regression is the Random Forest Regressor. It is an ensemble learning technique that builds numerous decision trees during the training phase and outputs the average forecast of each tree.

The fundamental principle of Random Forest is to build many decision trees, each trained on a randomly chosen subset of the training data and a randomly selected subset of the input features. The target variable is separately predicted by each decision tree in the forest, and the final forecast is made by averaging the predictions of all the trees. This lessens overfitting and enhances the model's generalization capabilities. It works effectively even when working with high-dimensionality datasets.

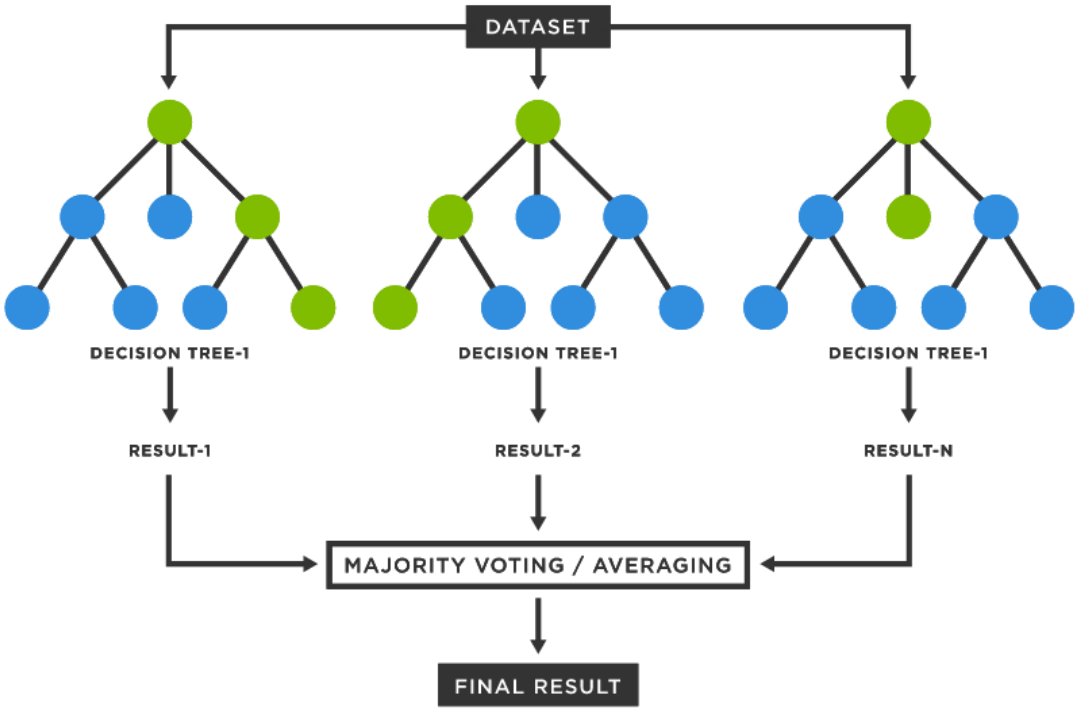


Fig 4.13 Random Forest Regressor

XGBoost Regressor

Extreme Gradient Boosting, or XGBoost, is an effective machine learning approach that may be applied to classification and regression issues. It is an application of the gradient boosting technique that employs several weak learning algorithms to combine them into a powerful predictive model. The fundamental principle of XGBoost is to iteratively train a number of weak decision tree models on the residual errors of the preceding tree with the aim of lowering the model's overall error.

Fig 4.14 XGBoost Regressor

The XGBoost approach employs a number of methods, such as shrinkage, feature subsampling, and regularisation terms, to regularise the decision trees and avoid overfitting. These methods strengthen the model's generalization capabilities and increase its resistance to noisy or insufficient data.

5. IMPLEMENTATION

Step 1: Install python, jupyter notebook

Python version 3.11 is required for the successful execution of the project

Step 2 : Import the required libraries

- **Pandas :** It is used for manipulating the large datasets in a simpler manner.
- **NumPy :** It is used in scientific computing, data analysis and machine learning. It has extensive coverage of mathematical functions.
- **Sklearn :** t is a machine learning library that provides tools for data pre-processing, model selection and evaluation.
- **XGB** : XGBoost (eXtreme Gradient Boosting) is a powerful and efficient open-source library for gradient boosting algorithms.
- **Matplotlib :** This library helps in creating visualizations and plots. We can create static or even dynamic and animated plots using matplotlib.
- **Pickle:** It converts Python objects into a byte stream and back, making it easier to store and transfer data. It is mainly used for the storage of data and more straightforward data communication.
- **Tabulate:** Tabulate is a library used for creating simple tables in python.
- **Tkinter:** It is used for creating windows, dialog boxes, buttons, menus, and other GUI components using python.
- **Kivy**: Kivy is primarily used for building apps across various operating systems, such as windows, macOS, Android, etc., using python.

Kivy, XGB cannot be directly imported. They first need to be installed with pip install in the command prompt before importing in the jupyter notebook.

Step 3: Load the required dataset

DATASET DESCRIPTION

This dataset consists of 10 features that describe various attributes of residential homes. The features include:

1. **Bedroom**: The number of bedrooms in the property.
2. **Bathroom**: The number of bathrooms on the property.
3. **Layout_type**: This feature describes the layout of the property. It has two values - RK or BHK. RK stands for Room Kitchen ,and BHK stands for Bedroom Hall Kitchen.
4. **Property_type**: This feature describes the type of property. It can take on values such as villa, independent house, apartment, independent floor, studio apartment, or penthouse.
5. **Furnish_type**: This feature describes the level of furnishing in the property. It can take on values such as furnished, semi-furnished, or unfurnished.
6. **Seller_type**: This feature describes the type of seller - builder, agent, or owner.
7. **City**: The city where the property is located. It has six possible values - Hyderabad, Mumbai, Chennai, Kolkata, Bangalore, or Delhi.
8. **Area**: The size of the property in square feet.
9. **Locality**: The name of the locality or neighbourhood where the property is located.
10. **Price**: The rent value of the property.

This dataset can be used to explore various relationships between these features and the price of residential properties.

Step 4: The nature of the attributes of the dataset is observed

```
Int64Index: 155498 entries, 0 to 23539
Data columns (total 10 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   seller_type    154434 non-null  object
 1   bedroom        154437 non-null  float64
 2   layout_type    154437 non-null  object
 3   property_type  154437 non-null  object
 4   locality       154436 non-null  object
 5   price          154437 non-null  object
 6   area           154437 non-null  float64
 7   furnish_type   154437 non-null  object
 8   bathroom       152387 non-null  object
 9   city           155498 non-null  object
dtypes: float64(2), object(8)
memory usage: 17.1+ MB
```

Step 5 Exploratory Data Analysis

Exploratory Data Analysis(EDA) is the process of examining and analyzing datasets to extract insights and identify patterns. The data has been visualized according to the numeric features and categorical features which are as follows :

1. Visualizing the numeric features:

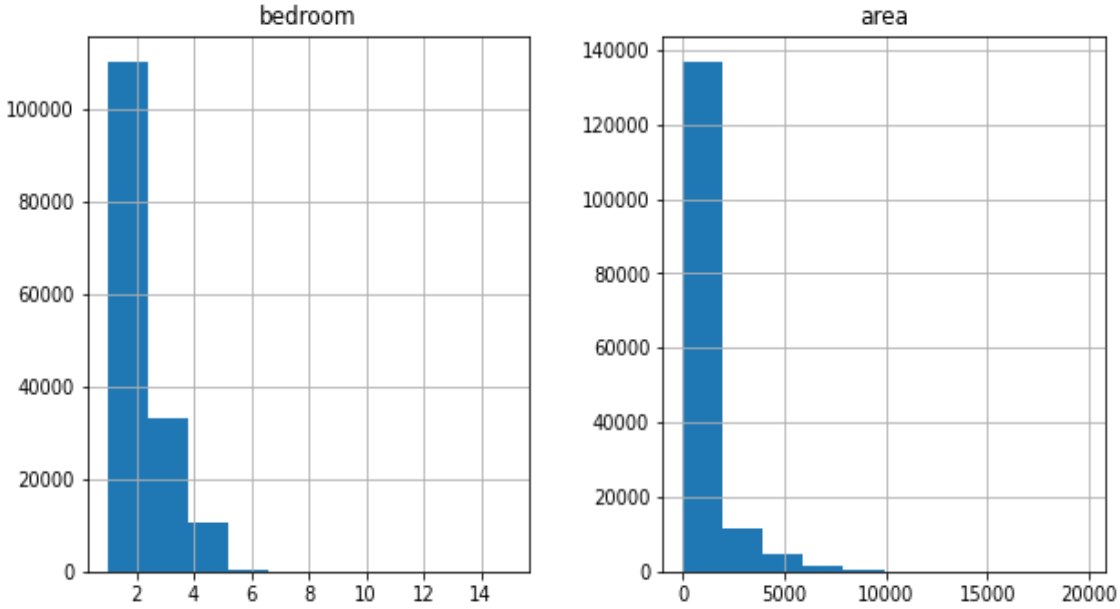


Fig 5.1 Analysis of bedroom and area using histogram

2. Visualizing the categorical features:

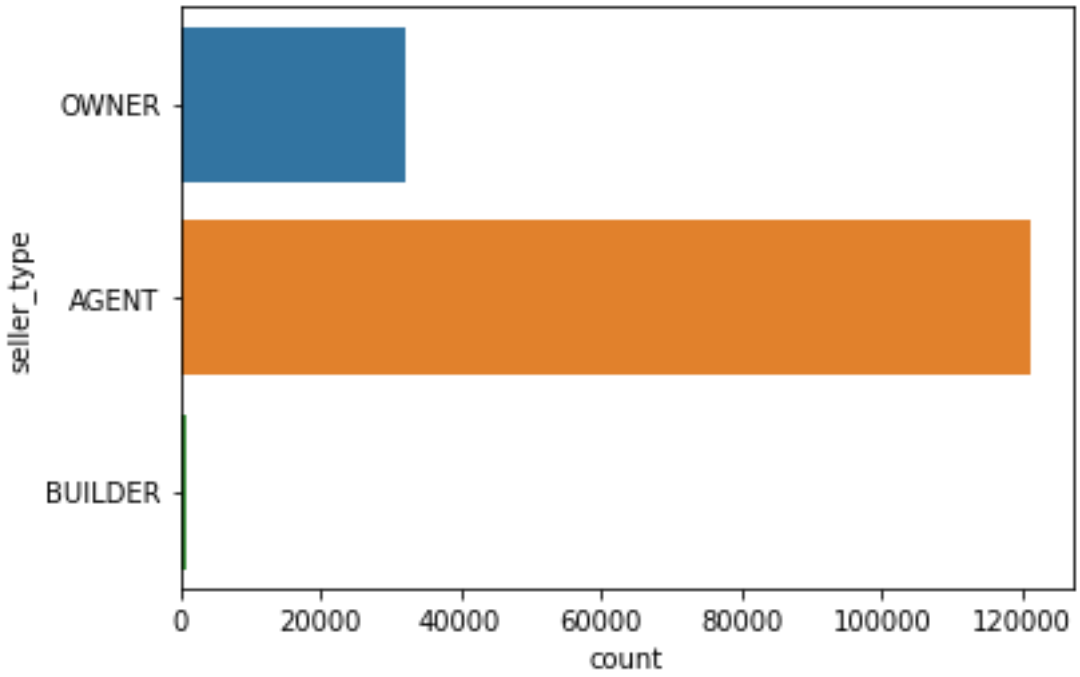


Fig 5.2 seller_type attribute

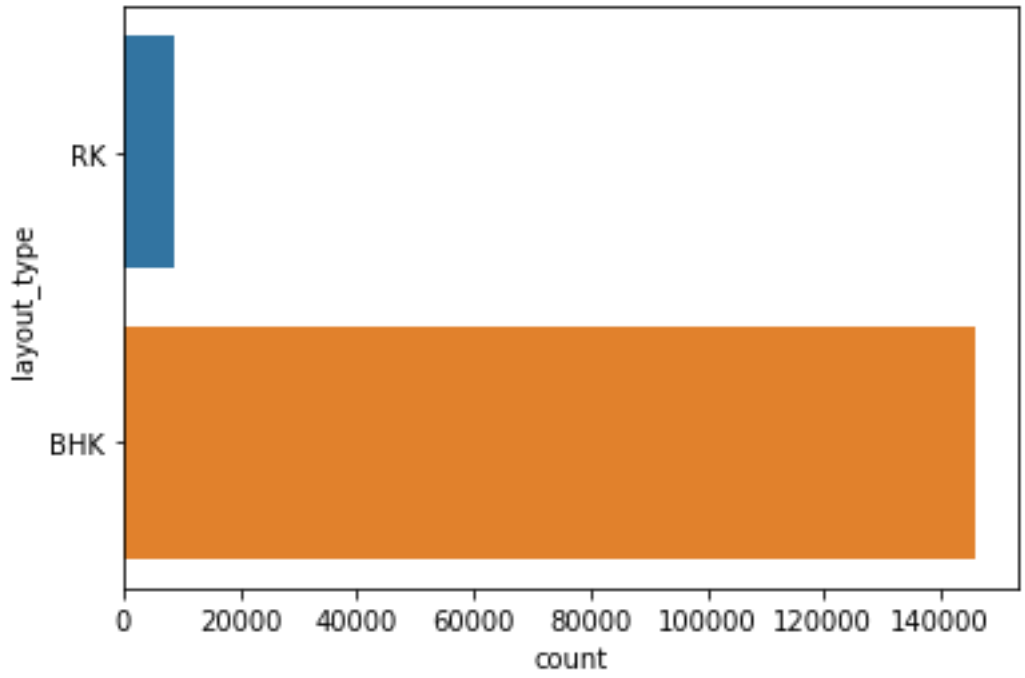


Fig 5.3 layout_type attribute

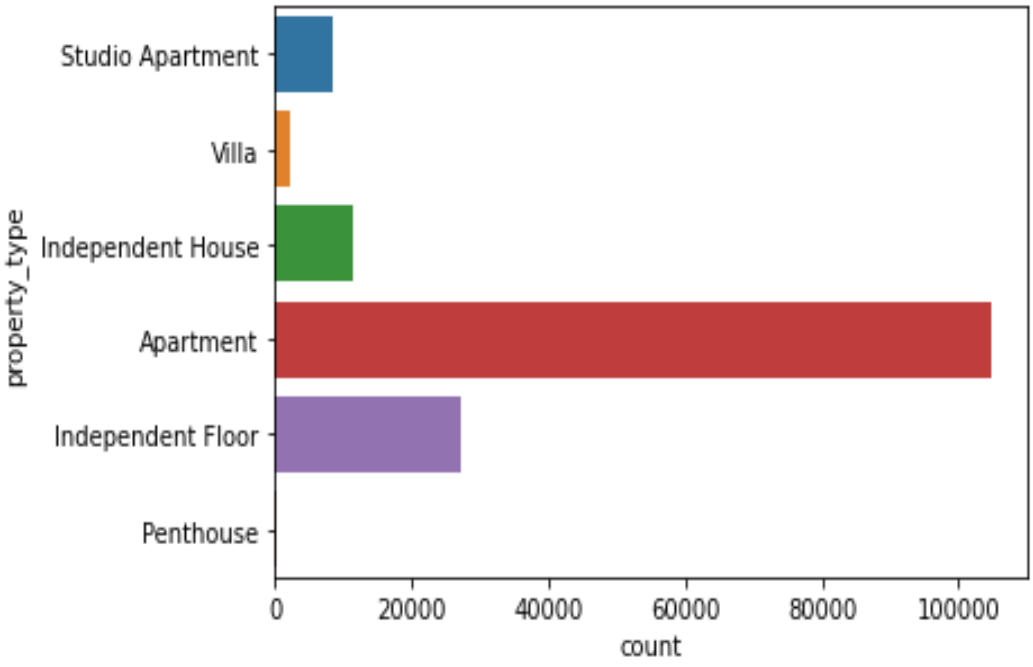


Fig 5.4 property_type attribute

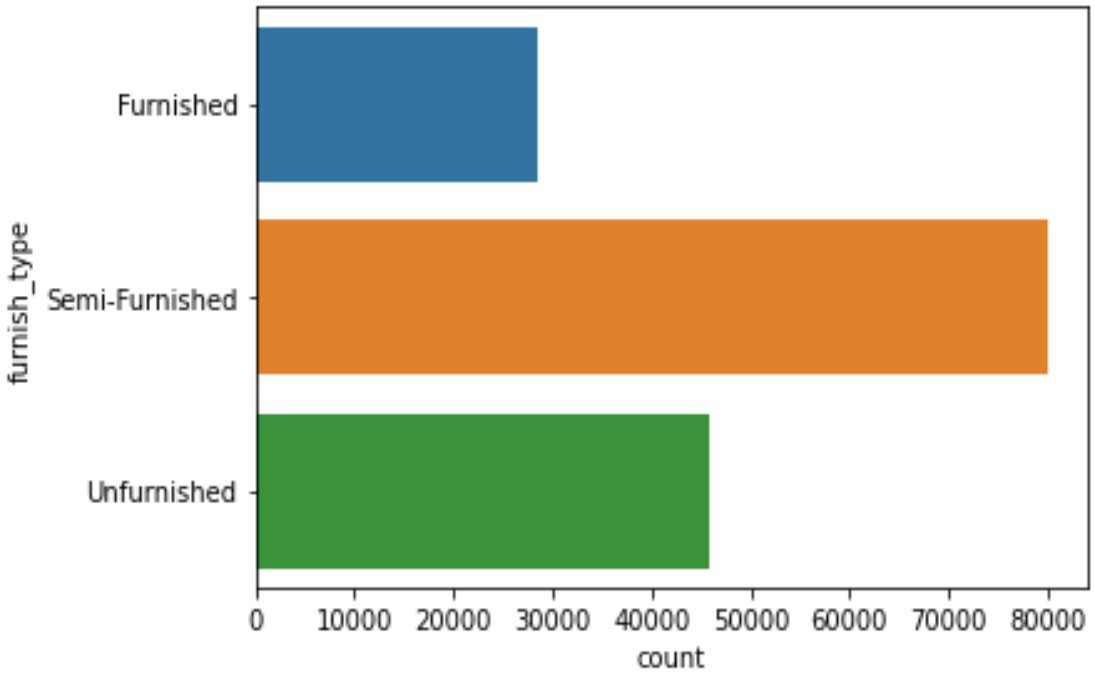
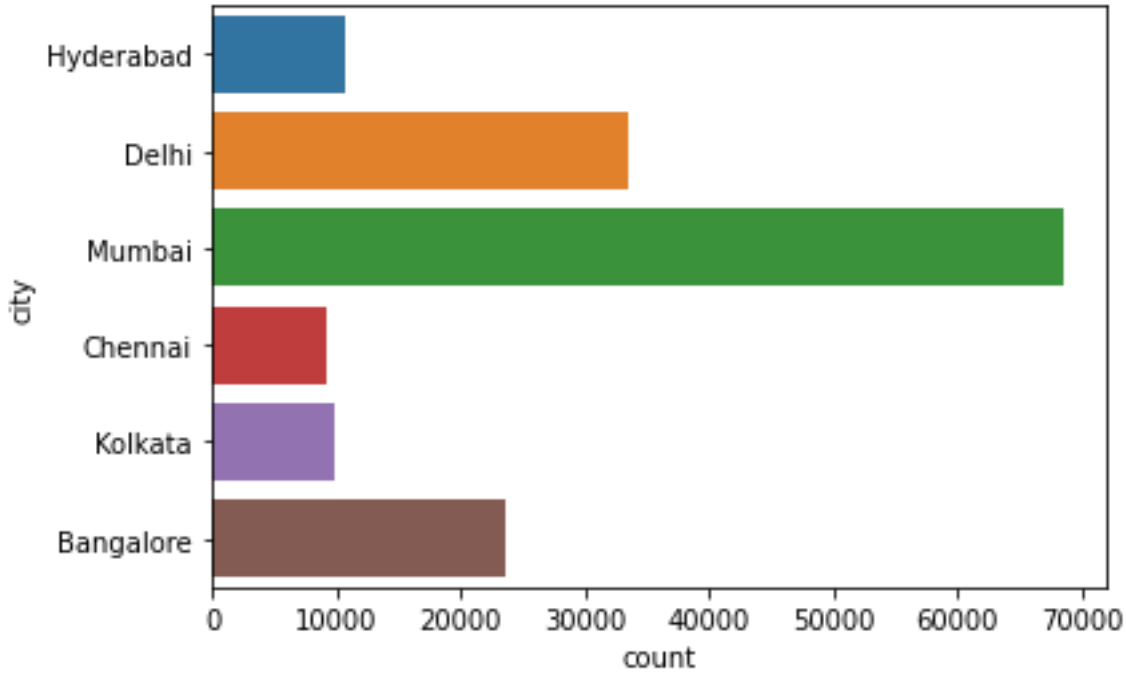


Fig 5.5 furnish_type Attribute



5.6 City Attribute

As the data contains a lot of categorical and string-related features which cannot be directly provided to the ML algorithms. Hence, it requires extensive data pre-processing to extract the required valuable information that can be utilized by the ML algorithm.

Step 6: Data Pre-processing

1. Checking for missing values
2. Checking for unique values in every feature to ensure that no erroneous values exist
3. Eliminate the erroneous values.
4. Convert bathroom & Bedroom attributes to float datatype
5. Converting the price feature from object to float
6. Annual housing Inflation is added on price attributes in order to make it real-time.
7. Converting locality values from upper case to lower case to ensure case sensitivity issues don't arise.
8. Converting the remaining categorical data to numerical data using the label encoding technique.
9. Verify the shape of the dataset after data pre-processing.

Step 7: Making the data suitable for the implementation of ML algorithms

The dataset is split into training and testing data in the ratio of 75:25

Step 8: Defining the methods for evaluation metrics

Three evaluation metrics are used for the ML algorithms to check the performance of the algorithm for predicting the rent.

The three evaluation metrics used are :

- Mean Absolute Error (MAE)
- Normalized Mean Root Square Error (NMRSE)
- R2 Score

Step 9: Algorithms

Various ML algorithms are trained and tested with the three evaluation metrics. The algorithms that are performed are :

- Linear Regression
- KNeighbors Regressor
- Gradient Boosting Regressor
- Random Forest Regressor

- Decision Tree Regressor
- XGBoost Regressor

Step 10: Tabulate the performance of every algorithm on the test data.

Step 11: Select the algorithm which gives the best performance over various parameters.

In our project, the random forest algorithm has the best performance, and hence it is selected as the final model. This model is further used as the backend for the user interface.

Step 12: Load the selected algorithm in a pickle file

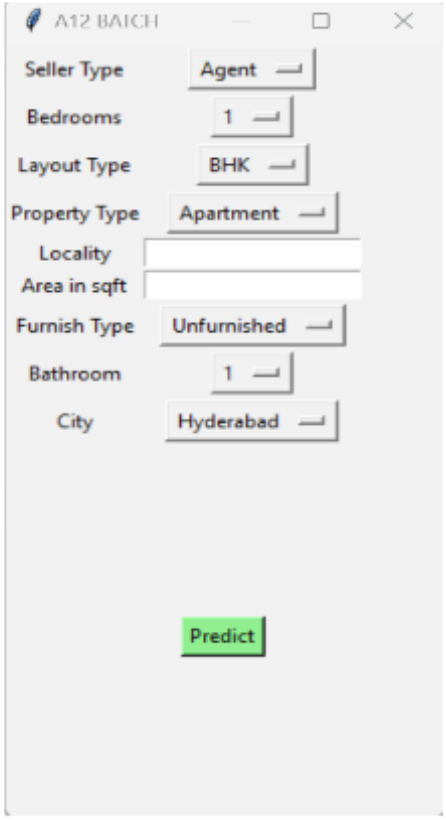
The pickle module is used for serializing and de-serializing a Python object structure. It converts a Python object into a byte stream and then back into the original object. The byte stream can be stored as a file, transmitted over a network, or saved to a database.

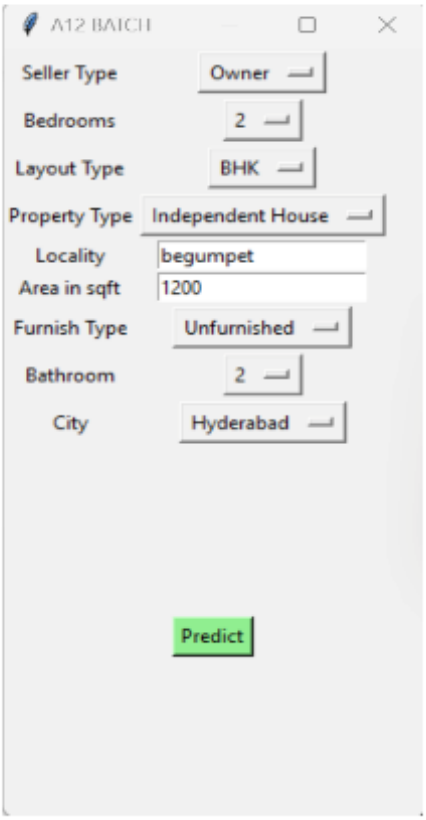
Step 13: Define a function named prediction to map the user input to the ML algorithm.

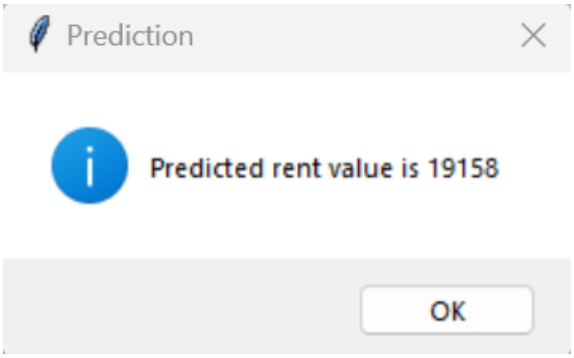
Step 14: Perform prediction with the features in order to successfully test the algorithm with user-defined values.

Step 15: User Interface

The User Interface for this is built using Tkinter. Tkinter is a standard python library used for creating a graphical user interface(GUI). It provides a set of tools and widgets for building applications that provide windows, buttons, textboxes, and many other elements that the user can interact with. It follows the object-oriented programming paradigm, which makes it easy to create and manage GUI elements. It is based on the concept of event-driven programming, where the user actions trigger specific events that can be handled by the application.







5.7 User Interface

6. TESTING & VALIDATION

The goal of software testing is to improve the quality of software, ensure that it meets the user requirements and reduce the risks associated with the software.

6.1 TEST PLAN

6.1.1 Introduction

The house rent prediction app is an user-friendly application designed to help users predict the rental prices for houses and apartments in different locations. It provides accurate and reliable rent based on user input for various factors such as locality, size of the area, property type, layout type, seller type, city, and the number of bathrooms and bedrooms.

Features of the app include :

- User-friendly interface
- Provides accurate rent
- Accessible to a wide range of users

The primary goal of the testing process for house rent prediction is to ensure that the app functions as intended and provides accurate rent prediction to the users. The testing approach for house rent prediction involves a manual testing technique. The testing was conducted in different stages, including unit testing and integration testing.

User Interface Testing

1. Verify that the user interface is intuitive and easy to use for the target audience.
2. Verify that the interface is consistent across all screens and devices.
3. Verify that all input fields and buttons are working correctly.

Functional Testing

1. Verify that the machine learning model is able to make accurate rent predictions.
2. Verify that the application is able to display the results of the rent prediction in an understandable and user-friendly format.

Performance Testing

1. Verify that the application is able to make rent predictions within a reasonable amount of time.
2. Verify that the application is not consuming an excessive amount of device resources, such as battery or CPU.

Compatibility Testing

1. Verify that the application is compatible with a range of screen sizes and resolutions.
2. Verify that the application is compatible with different versions of the operating systems.

Usability Testing

Verify that the application is easy to navigate and understand for the target audience.

Acceptance Testing:

1. Verify that the application meets the requirements specified in the software requirement specification.
2. Verify that the application provides value to the target audience and meets their expectations.

6.2 TEST SCENARIOS

The house rent prediction has been tested against a variety of scenarios to ensure that it provides accurate and reliable rent predictions. The scenarios that will be tested include the following:

1. Prediction accuracy for different property features: The app was tested to determine its accuracy in predicting the rent prices for different types of properties with features such as number of bedrooms, number of bathrooms, furnish type, locality, area size, layout type, and city.

2. User input scenarios: The app was tested with different user inputs to ensure that it can handle various inputs and provide accurate rent predictions. The user-defined values should be accurately forwarded to the algorithm.

6.3 ALGORITHM EVALUATION METRICS

Since regression cannot be directly evaluated for accuracy we shall be using the following metrics for evaluation of our algorithms.

Mean Absolute Error(MAE)

Mean Absolute Error (MAE) is a metric used to evaluate the performance of regression models. It measures the average absolute difference between the predicted values and the actual values of a dataset. To calculate MAE, we take the absolute value of the difference between each predicted value and the actual value, then take the mean of all these absolute differences. A lower MAE indicates that the model is better at predicting the actual values.

For a specific dataset, the MAE shows the average absolute difference between the target values that were actually achieved and those that were anticipated. Low MAE thereby means better performance.

Normalized Root Mean Squared Error(NRMSE)

Normalized Root Mean Squared Error (NRMSE) is a commonly used metric to evaluate the performance of regression models. It is a variation of the Root Mean Squared Error (RMSE) that takes into account the range of the target variable. NRMSE is calculated as the ratio of RMSE to the range of the target variable. The range is the difference between the maximum and minimum values of the target variable in the dataset.

NRMSE provides a more interpretable measure of model performance than RMSE alone. By normalizing the RMSE, it is possible to compare the performance of models trained on different datasets or with different units of measurement. NRMSE values range from 0 to 1, with values closer to 0 indicating better model performance.

R-squared Score

R-squared (R2) is a statistical measure that determines the proportion of variation in the dependent variable that can be explained by the independent variables in the model. The R2 score is calculated by comparing the sum of squared errors (SSE) of the regression model with the total sum of squares (SST) of the data.

The formula for R2 is 1 - (SSE/SST), where SSE is the sum of the squared differences between the predicted and actual values, and SST is the total sum of squared differences between the actual values and the mean of the dependent variable. A high R2 score indicates a good fit, while a low R2 score indicates a poor fit.

6.4 USER INPUT VALIDATION

The following are the test cases that are executed during the testing house rent prediction

Test Case 1 :

```
In [68]: pred('Agent',3,'BHK','Apartment','miyapur',2000,'Unfurnished',3,'Hyderabad')
Out[68]: 31665
```

Test Case 2 :

```
In [69]: pred('Agent',3,'BHK','Independent House','Andheri',1800,'Furnished',3,'Mumbai')
Out[69]: 56283
```

Test Case 3 :

```
In [70]: pred('Agent',5,'BHK','Villa','KPHB',2200,'Unfurnished',4,'Hyderabad')
Out[70]: 36319
```

Test Case 4 :

```
In [71]: pred('Builder',2,'BHK','Apartment','Ameerpet', 1200,'Semifurnished',2,'Hyderabad')
Out[71]: 15240
```

Test Case 5 :

```
In [72]: pred('Owner',1,'RK','Apartment','SR Nagar',900,'Semifurnished',1,'Hyderabad')
Out[72]: 10979
```

7. RESULTS ANALYSIS

The results obtained by the algorithms are tabulated below:

Algorithm	**MAE**	**NMRSE**	**R2 SCORE**
Linear Regression	8380.82	0.148	0.491
KNeighbors Regressor	4528.42	0.103	0.751
Gradient Boosting Regressor	6130.76	0.114	0.698
Decision Tree Regressor	3172.23	0.084	0.833
Random Forest Regressor	3087.16	0.073	0.874
XGB Regressor	4084.23	0.081	0.848

The metrics that have been used to evaluate the machine learning algorithms include :

1. Mean Absolute Error(MAE)
2. Normalized Root Mean Squared Error(NRMSE)
3. R-Squared Test(R2 Score)

7.1 Mean Absolute Error(MAE)

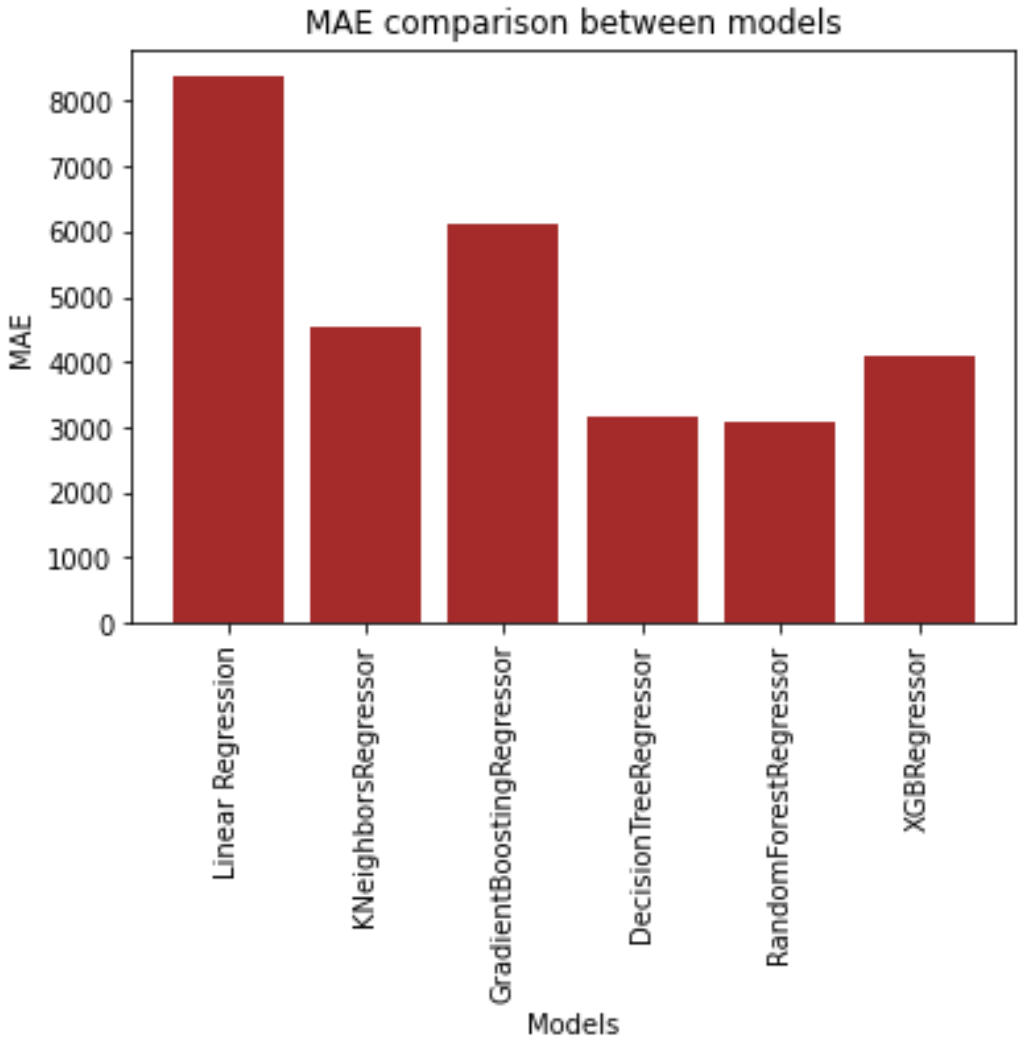


Fig 7.1 MAE Comparison between models

On comparison of the MAE it is clearly distinguished that the Random Forest Regressor is the best performing algorithm, closely followed by the Decision Tree Regressor. Linear regression has a very poor performance considering that house rent is a very vast topic with significant outliers. We can conclude that a single line wouldn't fit every housing category.

7.2 Normalized Root Mean Squared Error(NRMSE)

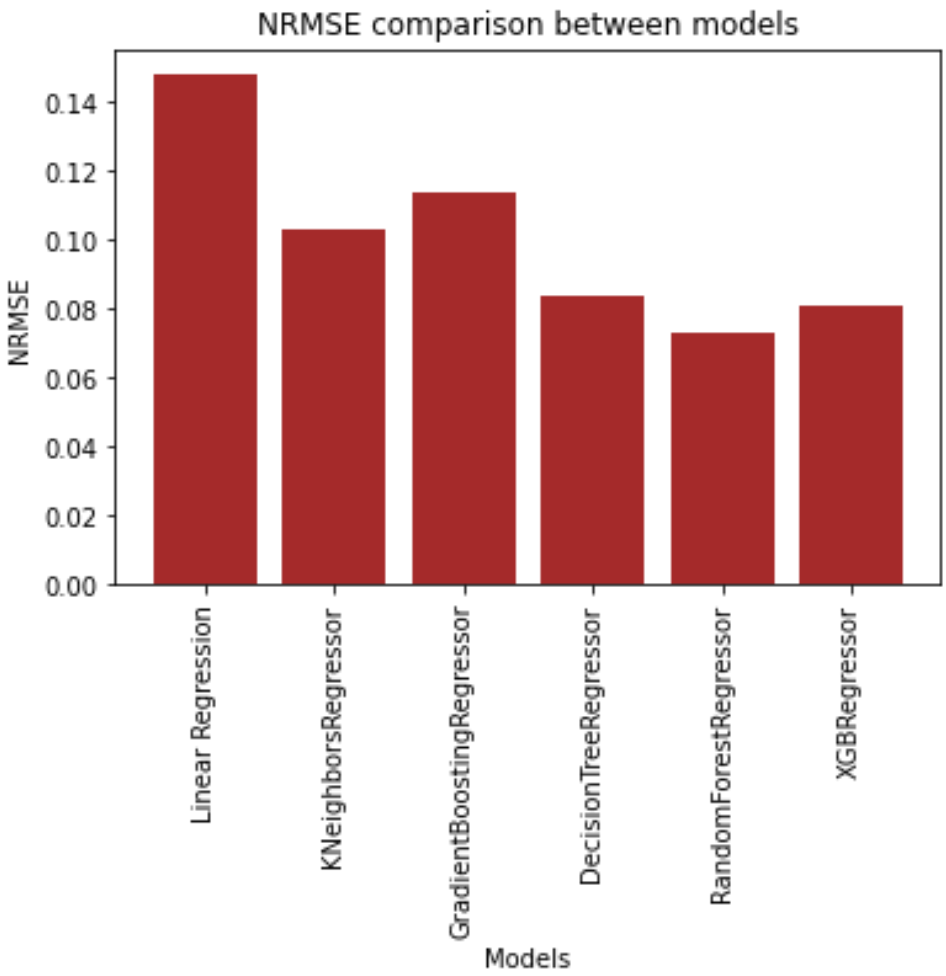


Fig 7.2 NRMSE comparison between models

The Random Forest Regressor has outperformed all the other algorithms and has the least NRMSE value. Ensemble-based bagging algorithms tend to perform better than the other regression techniques. This is due to the fact that the Random Forest Regressor is a powerful and versatile algorithm that is able to handle complex datasets with high dimensionality, and non-linear relationships between input and output variables.

7.3 R-squared Score

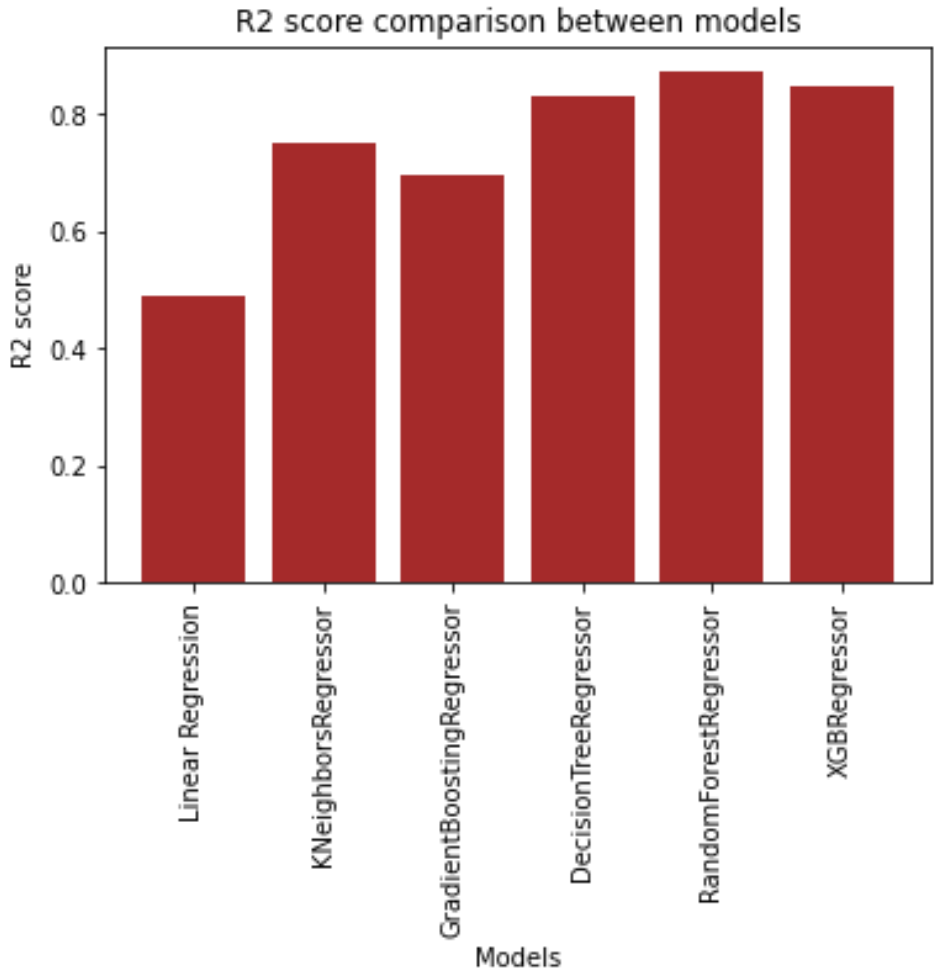


Fig 7.3 R2 Score comparison between models

Random Forest Regressor has a higher R2 score than the other algorithms. Overall, the Random Forest Regressor has proven to be an effective and reliable algorithm for house rent prediction and is the algorithm used in the implementation of the user-interface.

8. CONCLUSION & FUTURE SCOPE

CONCLUSION

House rent prediction system using machine learning has shown great results in accurately predicting the rental prices of properties across Indian metropolitan cities. By leveraging machine learning algorithms such as Random Forest Regressor, the system is able to take into account various factors that affect rental prices, such as location, property type, amenities, and more. Low mean absolute error and high R2 score have enabled random forest to be the final algorithm used in the application. Overall, the house rent prediction system has the potential to revolutionize the rental market in Indian metropolitan cities, providing more transparency and accuracy in rental pricing, and enabling people to make more informed decisions.

FUTURE SCOPE

The project can improve further by allowing users to contribute to the dataset. A larger dataset is essential to train the algorithm more effectively. It can be done via both traditional and a more modern ways of directly scraping data from property rent listing websites where rent would be explicitly mentioned and data would be more up to date. In addition, the number of features is currently extremely constrained, new features, such as floor count, lift availability etc., are to be added to make more accurate predictions. The project scope can be further enhanced from metropolitan cities to the entire nation. It can further be utilized for commercial use by having it as an additional feature in the popular market place related websites.

9. REFERENCES

A. Journals/Articles

1. Gabriel M. Ahlfeldt, Stephan Heblich, Tobias Seidel, "Micro-geographic property price and rent indices", *Regional Science and Urban Economics* **Volume 98, January 2023, 103836**
2. Nivitha Shree R H; Poojaa R; Rithick Roshan R; Mohan Kumar P, "Price Prediction of House using KNN based Lasso and Ridge Model", *2022 International Conference on Sustainable Computing and Data Communication Systems* **(ICSCDS) 10.1109/ICSCDS53736.2022.9760832**
3. Basetty Mallikarjuna, Sethu Ram M., Supriya Addanke, Munish Sabharwal, "An Improved Model for House Price/Land Price Prediction using Deep Learning", *Handbook of Research on Advances in Data Analytics and Complex Communication Networks*, **pp.76, 2022.**
4. Quang Truong, Minh Nguyen, Hy Dang, Bo Mei, "Housing Price Prediction via Improved Machine Learning Techniques", *Procedia Computer Science* **174 (2020) 433–442**
5. A. Ng and Deisenroth, "M. Machine learning for a London housing price prediction mobile application", **Imperial College London, 2020.**
6. Atharva Chogle, Priyanka Khaire, Akshata Gaud and Jinal Jain, "House Price Forecasting using Data Mining Techniques", *International Journal of Advanced Research in Computer and Communication Engineering*, **vol. 6, December 2017.**
7. WT Lim, L Wang, Y Wang and Q. Chang, "Housing price prediction using neural networks", *Natural Computation Fuzzy Systems and Knowledge Discovery (ICNC-FSKD) 2016 12th International Conference*, **pp. 518-522, Aug 2016.**

B. Books

8. Brian K. Jones and David M. Beazley, Python Cookbook: Recipes for Mastering Python 3 (3rd Edition)
9. Aurelion Geron, Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools and Techniques to build Intelligent Systems
10. Tom M. Mitchell, Machine Learning, McGraw Hill, 2017.
11. EthemAlpaydin, Introduction to Machine Learning, 3/e, PHI, 2015.

C. e-Websites / Downloads

12. https://www.kaggle.com/datasets/saisaathvik/house-rent-prices-of-metropolitan-cities-in-india/discussion
13. https://towardsdatascience.com/sampling-techniques-a4e34111d808
14. https://www.makaan.com/iq/real-estate-trends/q4-real-estate-report-proptiger-datalabs
15. https://xgboost.readthedocs.io/en/stable/python/python_api.html
16. https://www.analyticsvidhya.com/blog/2021/04/steps-to-complete-a-machine-learning-project/

ANNEXURE A : CODE

CSEBNUM_A12 - 2019 - HOUSE RENT PREDICTION USING ML

Importing libraries

In [1]:

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import r2_score
import xgboost as xgb
import matplotlib.pyplot as plt
import seaborn as sns
import math
import pickle
from tabulate import tabulate
from tkinter import *
import tkinter.messagebox
```

Preparing the dataset

In [2]:

```
dfH = pd.read_csv('/Users/praneetpeshkar/Desktop/Major Project/datasets/Hyderabad_rent.csv'
dfD = pd.read_csv('/Users/praneetpeshkar/Desktop/Major Project/datasets/Delhi_rent.csv')
dfM = pd.read_csv('/Users/praneetpeshkar/Desktop/Major Project/datasets/Mumbai_rent.csv')
dfC = pd.read_csv('/Users/praneetpeshkar/Desktop/Major Project/datasets/Chennai_rent.csv')
dfK = pd.read_csv('/Users/praneetpeshkar/Desktop/Major Project/datasets/Kolkata_rent.csv')
dfB = pd.read_csv('/Users/praneetpeshkar/Desktop/Major Project/datasets/Bangalore_rent.csv'
```

In [3]:

```
dfs = [dfH,dfD,dfM,dfC,dfK,dfB]
cities = ['Hyderabad', 'Delhi', 'Mumbai', 'Chennai', 'Kolkata', 'Bangalore']
for df,city in zip(dfs,cities):
    df['city']=city
    print(city,df.shape)
```

```
Hyderabad (10757, 10)
Delhi (33500, 10)
Mumbai (68518, 10)
Chennai (9283, 10)
Kolkata (9900, 10)
Bangalore (23540, 10)
```

In [4]:

```
df = pd.concat(dfs)
df.shape
```

Out[4]:

```
(155498, 10)
```

In [5]:

```
df.head()
```

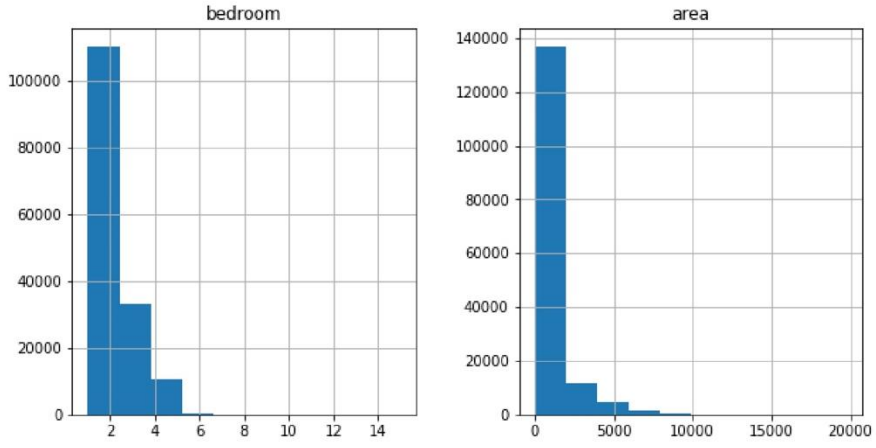
Out[5]:

	seller_type	bedroom	layout_type	property_type	locality	price	area	furnish_type	bathroom
0	OWNER	1.0	RK	Studio Apartment	Serilingampally	6,720	320.0	Furnished	1 bathrooms
1	OWNER	5.0	BHK	Villa	Sri Nagar Colony	36,000	3000.0	Semi-Furnished	4 bathrooms
2	OWNER	2.0	BHK	Independent House	Shaikpet	11,000	900.0	Semi-Furnished	2 bathrooms
3	OWNER	3.0	BHK	Apartment	Nanakramguda	45,000	2165.0	Unfurnished	3 bathrooms
4	OWNER	3.0	BHK	Apartment	Kondapur	20,000	1600.0	Semi-Furnished	3 bathrooms

Exploratory Data Analysis

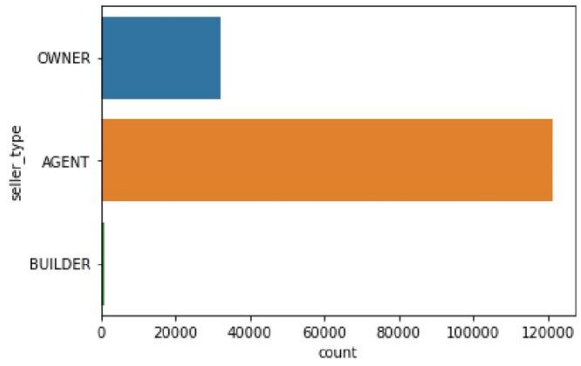
In [7]:

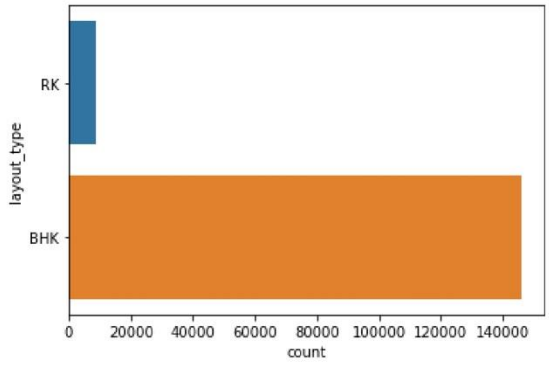
```
df.hist(figsize = (10,5))
plt.show()
```

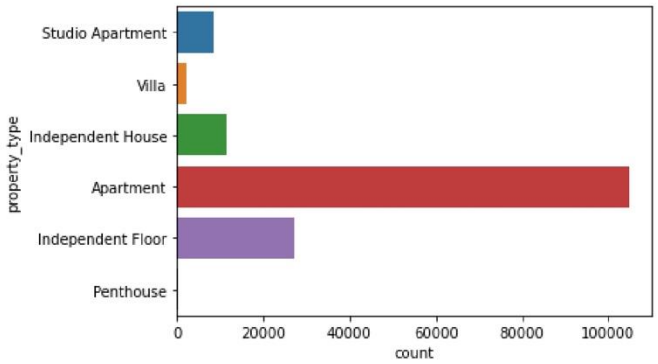


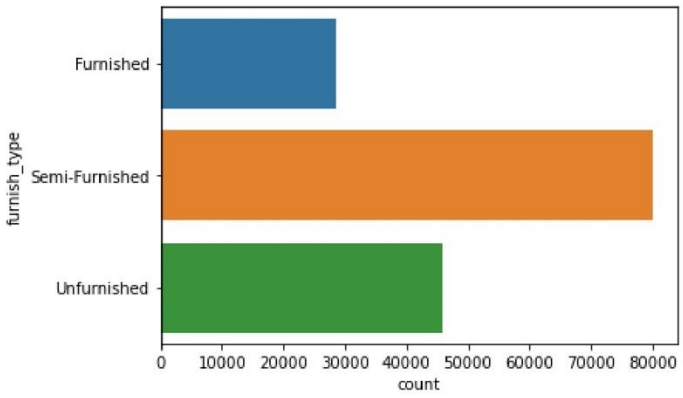
In [8]:

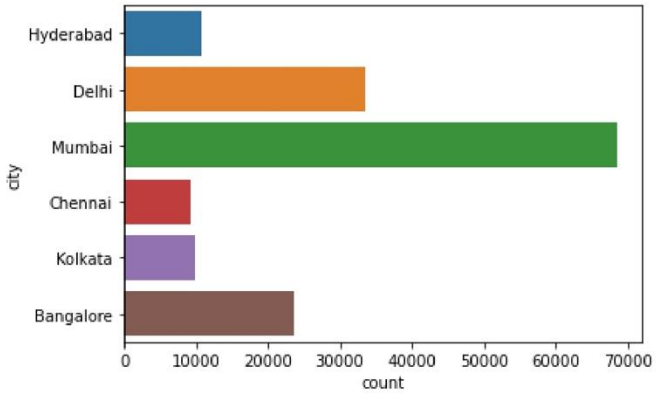
```
for column in df.select_dtypes(include='object'):
    if df[column].nunique() < 10:
        sns.countplot(y=column, data=df)
        plt.show()
```











In [17]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 155498 entries, 0 to 23539
Data columns (total 10 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   seller_type    154434 non-null  object
 1   bedroom        154437 non-null  float64
 2   layout_type    154437 non-null  object
 3   property_type  154437 non-null  object
 4   locality       154436 non-null  object
 5   price          154437 non-null  object
 6   area           154437 non-null  float64
 7   furnish_type   154437 non-null  object
 8   bathroom       152387 non-null  object
 9   city           155498 non-null  object
dtypes: float64(2), object(8)
memory usage: 17.1+ MB
```

Data Preprocessing

checking missing values

In [10]:

```
df.isnull().sum()
```

Out[10]:

```
seller_type      1064
bedroom          1061
layout_type      1061
property_type    1061
locality         1062
price            1061
area             1061
furnish_type     1061
bathroom         3111
city                0
dtype: int64
```

In [11]:

```
df=df.dropna()
df.shape
```

Out[11]:

```
(152383, 10)
```

checking all unique values

In [12]:

```
print(df.seller_type.unique())
print(df.seller_type.value_counts())
```

```
['OWNER' 'AGENT' 'BUILDER']
AGENT      119819
OWNER       31632
BUILDER       932
Name: seller_type, dtype: int64
```

In [13]:

```
print(df.bedroom.unique())
print(df.bedroom.value_counts())
```

```
[ 1.  5.  2.  3.  6.  4. 10. 11.  8.  7.  9. 15. 12. 14.]
2.0     59208
1.0     49411
3.0     32901
4.0      9310
5.0      1199
6.0       131
10.0       74
8.0        46
7.0        45
9.0        23
15.0       22
12.0        6
14.0        4
11.0        3
Name: bedroom, dtype: int64
```

In [14]:

```
print(df.layout_type.unique())
print(df.layout_type.value_counts())
```

```
['RK' 'BHK']
BHK    144188
RK       8195
Name: layout_type, dtype: int64
```

In [15]:

```
print(df.property_type.unique())
print(df.property_type.value_counts())
```

```
['Studio Apartment' 'Villa' 'Independent House' 'Apartment'
 'Independent Floor' 'Penthouse']
Apartment            103764
Independent Floor     26840
Independent House     11253
Studio Apartment       8195
Villa                  2199
Penthouse               132
Name: property_type, dtype: int64
```

In [16]:

```
print(df.furnish_type.unique())
print(df.furnish_type.value_counts())
```

```
['Furnished' 'Semi-Furnished' 'Unfurnished']
Semi-Furnished    79129
Unfurnished       45250
Furnished         28004
Name: furnish_type, dtype: int64
```

In [17]:

```
print(df.bathroom.unique())
print(df.bathroom.value_counts())
```

```
['1 bathrooms' '4 bathrooms' '2 bathrooms' '3 bathrooms' '5 bathrooms'
 'East facing' 'West facing' 'North facing' 'South facing' '6 bathrooms'
 '7 bathrooms' 'Grfloor' '10 bathrooms' '12 bathrooms' '8 bathrooms'
 'NorthWest facing' '4 of 5floor' '9 bathrooms' 'NorthEast facing'
 'SouthEast facing' '16 bathrooms' '14 bathrooms' '15 bathrooms'
 'Gr of 2floor' '3 of 3floor' '4 of 9floor' '3 of 4floor' '2 of 4floor'
 '2 of 3floor' '3 of 5floor' '1 of 3floor' '2 of 5floor'
 'SouthWest facing' 'availability immediately' '2 of 7floor'
 '7 of 12floor' 'Family only' '17 of 29floor' '10 of 16floor'
 '4 of 7floor' '17 of 25floor' '6 of 7floor' '11 of 17floor'
 '15 of 28floor' '5 of 14floor' '6 of 8floor' '9 of 12floor' '3 of 7floor'
 '1 of 4floor' '12 of 16floor' '7 of 16floor' '1 of 6floor' '18 bathrooms'
 '19 bathrooms' '1 of 2floor']
2 bathrooms                 71127
1 bathrooms                 45390
3 bathrooms                 24654
4 bathrooms                  8630
5 bathrooms                  1593
6 bathrooms                   331
East facing                   152
7 bathrooms                    74
North facing                   62
9 bathrooms                    55
NorthEast facing               50
8 bathrooms                    48
10 bathrooms                   34
West facing                    23
Grfloor                        15
SouthWest facing               15
1 of 4floor                    13
NorthWest facing               11
SouthEast facing               10
2 of 4floor                     8
15 bathrooms                    7
16 bathrooms                    6
14 bathrooms                    6
South facing                    6
3 of 3floor                     5
3 of 4floor                     5
Family only                     4
1 of 3floor                     4
2 of 5floor                     4
12 bathrooms                    4
2 of 7floor                     3
5 of 14floor                    2
11 of 17floor                   2
6 of 7floor                     2
10 of 16floor                   2
Gr of 2floor                    2
7 of 12floor                    2
19 bathrooms                    2
2 of 3floor                     2
4 of 5floor                     2
availability immediately        2
3 of 7floor                     1
1 of 6floor                     1
7 of 16floor                    1
18 bathrooms                    1
12 of 16floor                   1
17 of 29floor                   1
9 of 12floor                    1
6 of 8floor                     1
15 of 28floor                   1
17 of 25floor                   1
4 of 7floor                     1
3 of 5floor                     1
```

```
4 of 9floor                     1
1 of 2floor                     1
Name: bathroom, dtype: int64
```

removing unnecessary values

In [18]:

```
df=df[df.bathroom.str.contains('bathrooms')]
df.shape
```

Out[18]:

```
(151962, 10)
```

In [19]:

```
print(df.bathroom.unique())
print(df.bathroom.value_counts())
```

```
['1 bathrooms' '4 bathrooms' '2 bathrooms' '3 bathrooms' '5 bathrooms'
 '6 bathrooms' '7 bathrooms' '10 bathrooms' '12 bathrooms' '8 bathrooms'
 '9 bathrooms' '16 bathrooms' '14 bathrooms' '15 bathrooms' '18 bathrooms'
 '19 bathrooms']
2 bathrooms     71127
1 bathrooms     45390
3 bathrooms     24654
4 bathrooms      8630
5 bathrooms      1593
6 bathrooms       331
7 bathrooms        74
9 bathrooms        55
8 bathrooms        48
10 bathrooms       34
15 bathrooms        7
16 bathrooms        6
14 bathrooms        6
12 bathrooms        4
19 bathrooms        2
18 bathrooms        1
Name: bathroom, dtype: int64
```

In [20]:

```
df['bathroom'] = df['bathroom'].str.replace(r'[^\d.]+', '', regex=True)
df.head()
```

Out[20]:

	seller_type	bedroom	layout_type	property_type	locality	price	area	furnish_type	bathroom
0	OWNER	1.0	RK	Studio Apartment	Serilingampally	6,720	320.0	Furnished	1
1	OWNER	5.0	BHK	Villa	Sri Nagar Colony	36,000	3000.0	Semi-Furnished	4
2	OWNER	2.0	BHK	Independent House	Shaikpet	11,000	900.0	Semi-Furnished	2
3	OWNER	3.0	BHK	Apartment	Nanakramguda	45,000	2165.0	Unfurnished	3
4	OWNER	3.0	BHK	Apartment	Kondapur	20,000	1600.0	Semi-Furnished	3

In [21]:

```
df['bathroom'] = df['bathroom'].astype('float64', errors = 'raise')
```

In [22]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 151962 entries, 0 to 23539
Data columns (total 10 columns):
 #   Column         Non-Null Count   Dtype  
---  ------         --------------   -----  
 0   seller_type    151962 non-null  object 
 1   bedroom        151962 non-null  float64
 2   layout_type    151962 non-null  object 
 3   property_type  151962 non-null  object 
 4   locality       151962 non-null  object 
 5   price          151962 non-null  object 
 6   area           151962 non-null  float64
 7   furnish_type   151962 non-null  object 
 8   bathroom       151962 non-null  float64
 9   city           151962 non-null  object 
dtypes: float64(3), object(7)
memory usage: 12.8+ MB
```

In [23]:

```
d = df[df['property_type'] == 'Penthouse'].index
d
```

Out[23]:

```
Int64Index([ 5892,  6513,  6535,  6692,  6693,  7037,  7066,  7186,  8046,
             8594,
            ...
            17811, 17832, 17836, 17838, 17839, 17857, 17968, 18615, 18991,
            22901],
           dtype='int64', length=132)
```

In [24]:

```
df.drop(d, inplace = True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 151601 entries, 0 to 23539
Data columns (total 10 columns):
 #   Column         Non-Null Count   Dtype  
---  ------         --------------   -----  
 0   seller_type    151601 non-null  object 
 1   bedroom        151601 non-null  float64
 2   layout_type    151601 non-null  object 
 3   property_type  151601 non-null  object 
 4   locality       151601 non-null  object 
 5   price          151601 non-null  object 
 6   area           151601 non-null  float64
 7   furnish_type   151601 non-null  object 
 8   bathroom       151601 non-null  float64
 9   city           151601 non-null  object 
dtypes: float64(3), object(7)
memory usage: 12.7+ MB
```

In [25]:

```
df['p2'] = df['price'].apply(lambda x : '0' if ',' in x else '1')
df.head()
```

Out[25]:

	seller_type	bedroom	layout_type	property_type	locality	price	area	furnish_type	bathroom
0	OWNER	1.0	RK	Studio Apartment	Serilingampally	6,720	320.0	Furnished	1.0
1	OWNER	5.0	BHK	Villa	Sri Nagar Colony	36,000	3000.0	Semi-Furnished	4.0
2	OWNER	2.0	BHK	Independent House	Shaikpet	11,000	900.0	Semi-Furnished	2.0
3	OWNER	3.0	BHK	Apartment	Nanakramguda	45,000	2165.0	Unfurnished	3.0
4	OWNER	3.0	BHK	Apartment	Kondapur	20,000	1600.0	Semi-Furnished	3.0

In [26]:

```
df['price'] = df['price'].str.replace(r'[^\d.]+', '', regex=True)
df.head()
```

Out[26]:

	seller_type	bedroom	layout_type	property_type	locality	price	area	furnish_type	bathroom
0	OWNER	1.0	RK	Studio Apartment	Serilingampally	6720	320.0	Furnished	1.0
1	OWNER	5.0	BHK	Villa	Sri Nagar Colony	36000	3000.0	Semi-Furnished	4.0
2	OWNER	2.0	BHK	Independent House	Shaikpet	11000	900.0	Semi-Furnished	2.0
3	OWNER	3.0	BHK	Apartment	Nanakramguda	45000	2165.0	Unfurnished	3.0
4	OWNER	3.0	BHK	Apartment	Kondapur	20000	1600.0	Semi-Furnished	3.0

In [27]:

```
df['price'] = df['price'].astype('float64')
df.loc[df.p2.str.contains('1'), 'price'] *= 100000.0
df.drop('p2',axis=1,inplace=True)
df.head()
```

Out[27]:

	seller_type	bedroom	layout_type	property_type	locality	price	area	furnish_type	bathroom
0	OWNER	1.0	RK	Studio Apartment	Serilingampally	6720.0	320.0	Furnished	1.0
1	OWNER	5.0	BHK	Villa	Sri Nagar Colony	36000.0	3000.0	Semi-Furnished	4.0
2	OWNER	2.0	BHK	Independent House	Shaikpet	11000.0	900.0	Semi-Furnished	2.0
3	OWNER	3.0	BHK	Apartment	Nanakramguda	45000.0	2165.0	Unfurnished	3.0
4	OWNER	3.0	BHK	Apartment	Kondapur	20000.0	1600.0	Semi-Furnished	3.0

In [28]:

```
inflation = 4.47/100
df['price']=df['price']+(df['price']*inflation*6/12)
df.head()
```

Out[28]:

	seller_type	bedroom	layout_type	property_type	locality	price	area	furnish_type	bathroc
0	OWNER	1.0	RK	Studio Apartment	Serilingampally	6870.192	320.0	Furnished	
1	OWNER	5.0	BHK	Villa	Sri Nagar Colony	36804.600	3000.0	Semi-Furnished	
2	OWNER	2.0	BHK	Independent House	Shaikpet	11245.850	900.0	Semi-Furnished	
3	OWNER	3.0	BHK	Apartment	Nanakramguda	46005.750	2165.0	Unfurnished	
4	OWNER	3.0	BHK	Apartment	Kondapur	20447.000	1600.0	Semi-Furnished	

In [29]:

```
df['locality'] = df['locality'].apply(str.lower)
```

In [30]:

```
locality_c = df.groupby('locality').size().sort_values(ascending=True)
```

In [31]:

```
for i in range(1,11):
    print('no of localities with houses less than ',i,'is ',len(locality_c[locality_c<i]))
```

```
no of localities with houses less than  1 is  0
no of localities with houses less than  2 is  1027
no of localities with houses less than  3 is  1582
no of localities with houses less than  4 is  1844
no of localities with houses less than  5 is  2008
no of localities with houses less than  6 is  2138
no of localities with houses less than  7 is  2239
no of localities with houses less than  8 is  2308
no of localities with houses less than  9 is  2375
no of localities with houses less than  10 is  2424
```

In [32]:

```
loc_10=locality_c[locality_c<7]
```

In [33]:

```
df.locality = df.locality.apply(lambda x: 'other' if x in loc_10 else x)
```

Removing Outliers

In [34]:

```
df=df[(df['price']<75000) & (df['bathroom']<7) & (df['bedroom']<7) & (df['area']<10000)]
```

In [35]:

```
df.head()
```

Out[35]:

	seller_type	bedroom	layout_type	property_type	locality	price	area	furnish_type	bathroo
0	OWNER	1.0	RK	Studio Apartment	serilingampally	6870.192	320.0	Furnished	1
1	OWNER	5.0	BHK	Villa	sri nagar colony	36804.600	3000.0	Semi-Furnished	4
2	OWNER	2.0	BHK	Independent House	shaikpet	11245.850	900.0	Semi-Furnished	2
3	OWNER	3.0	BHK	Apartment	nanakramguda	46005.750	2165.0	Unfurnished	3
4	OWNER	3.0	BHK	Apartment	kondapur	20447.000	1600.0	Semi-Furnished	3

In [36]:

```
df2=df.copy()
```

Converting categorical data into numeric data

In [37]:

```
enc=LabelEncoder()
```

In [38]:

```
seller_type=enc.fit_transform(df['seller_type'])
layout_type=enc.fit_transform(df['layout_type'])
property_type=enc.fit_transform(df['property_type'])
locality=enc.fit_transform(df['locality'])
furnish_type=enc.fit_transform(df['furnish_type'])
city=enc.fit_transform(df['city'])
```

In [39]:

```
df['seller_type']=seller_type
df['layout_type']=layout_type
df['property_type']=property_type
df['locality']=locality
df['furnish_type']=furnish_type
df['city']=city
```

In [40]:

```
idx = pd.RangeIndex(start=0, stop=len(df), step=1)
df.index = idx
```

In [41]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 132220 entries, 0 to 132219
Data columns (total 10 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   seller_type    132220 non-null  int64
 1   bedroom        132220 non-null  float64
 2   layout_type    132220 non-null  int64
 3   property_type  132220 non-null  int64
 4   locality       132220 non-null  int64
 5   price          132220 non-null  float64
 6   area           132220 non-null  float64
 7   furnish_type   132220 non-null  int64
 8   bathroom       132220 non-null  float64
 9   city           132220 non-null  int64
dtypes: float64(4), int64(6)
memory usage: 10.1 MB
```

After Data Preprocessing

In [44]:

```
df.shape
```

Out[44]:

```
(132220, 10)
```

In [45]:

```
df.corr()
```

Out[45]:

	seller_type	bedroom	layout_type	property_type	locality	price	area	furnish_typ
seller_type	1.000000	-0.034635	0.072962	0.250540	0.001767	-0.283165	-0.054221	0.0501
bedroom	-0.034635	1.000000	-0.277177	-0.165683	0.075980	0.487996	0.802696	-0.1182
layout_type	0.072962	-0.277177	1.000000	0.694924	-0.008871	-0.213222	-0.296756	0.0211
property_type	0.250540	-0.165683	0.694924	1.000000	0.009879	-0.272940	-0.159630	0.0015
locality	0.001767	0.075980	-0.008871	0.009879	1.000000	-0.082606	0.058483	-0.0106
price	-0.283165	0.487996	-0.213222	-0.272940	-0.082606	1.000000	0.516461	-0.2679
area	-0.054221	0.802696	-0.296756	-0.159630	0.058483	0.516461	1.000000	-0.1527
furnish_type	0.050179	-0.118255	0.021160	0.001537	-0.010646	-0.267975	-0.152711	1.0000
bathroom	-0.102503	0.791860	-0.272431	-0.220154	0.052402	0.530335	0.743415	-0.1154
city	-0.288936	-0.173075	0.022325	-0.282801	-0.050140	0.213709	-0.209197	0.1542

Making data suitable for ML algorithms

In [46]:

```
X = df.drop(['price'], axis=1)
y = df['price']
```

In [47]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=12)
```

In [48]:

```
print('X train - ',X_train.shape)
print('Y train - ',y_train.shape)
print('X test - ',X_test.shape)
print('Y test - ',y_test.shape)
```

```
X train -  (99165, 9)
Y train -  (99165,)
X test -  (33055, 9)
Y test -  (33055,)
```

Defining Evaluation metrics

In [49]:

```
def mae (y_true, y_pred):
    return (np.mean(abs(y_true-y_pred)))
```

In [50]:

```
def nrmse (y_true, y_pred):
    return (math.sqrt(np.square(np.subtract(y_true,y_pred)).mean()))/75000
```

In [51]:

```
mod=[]
header=['Algorithm', 'MAE', 'NRMSE', 'R2 score']
```

In [52]:

```
def model_test(model,name):
    model.fit(X_train.values,y_train)
    predic = model.predict(X_test.values)
    MAE=round(mae(y_test,predic),3)
    norm_rmse=round(nrmse(y_test,predic),3)
    r2s=round(r2_score(y_test,predic),3)
    l1=[name,MAE,norm_rmse,r2s]
    mod.append(l1)
    print('MAE : ', MAE)
    print('Normalized RMSE : ',norm_rmse)
    print('R2 Score : ',r2s)
```

ML Algorithms implementation

In [53]:

```
lr = LinearRegression()
model_test(lr,'Linear Regression')
```

```
MAE :  8380.815
Normalized RMSE :  0.148
R2 Score :  0.491
```

In [54]:

```
knn = KNeighborsRegressor(n_neighbors=5)
model_test(knn, 'KNeighborsRegressor')
```

```
MAE :  4532.403
Normalized RMSE :  0.103
R2 Score :  0.751
```

In [55]:

```
gb = GradientBoostingRegressor(random_state=12)
model_test(gb,'GradientBoostingRegressor')
```

```
MAE :  6130.755
Normalized RMSE :  0.114
R2 Score :  0.698
```

In [56]:

```
tree = DecisionTreeRegressor(random_state=12)
model_test(tree,'DecisionTreeRegressor')
```

```
MAE :  3172.228
Normalized RMSE :  0.084
R2 Score :  0.833
```

In [57]:

```
rf = RandomForestRegressor(random_state=12)
model_test(rf,'RandomForestRegressor')
```

```
MAE :  3087.154
Normalized RMSE :  0.073
R2 Score :  0.874
```

In [58]:

```
xgb = xgb.XGBRegressor(objective='reg:squarederror',random_state=12)
model_test(xgb,'XGBRegressor')
```

```
MAE :  4084.234
Normalized RMSE :  0.081
R2 Score :  0.848
```

Model Evaluation

In [59]:

```
print(tabulate(mod, headers=header))
```

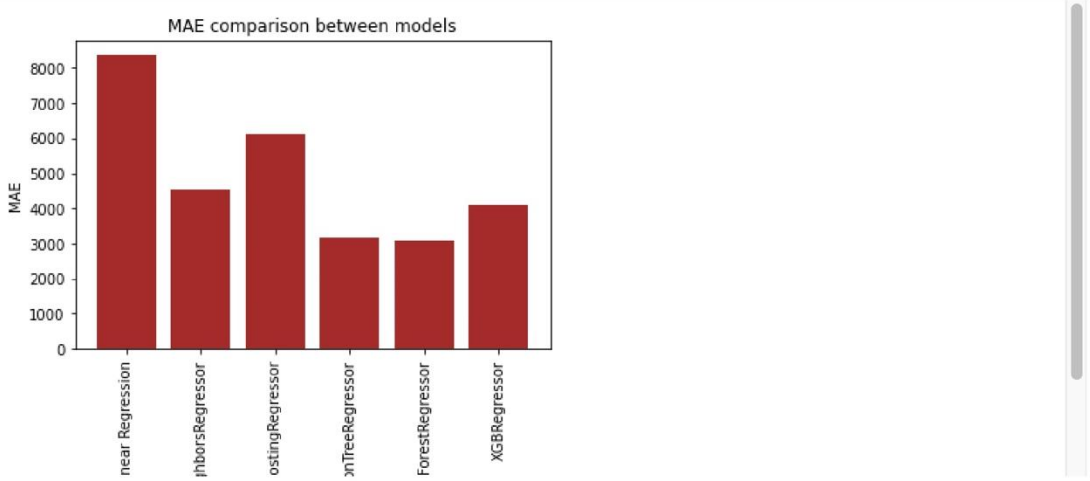
Algorithm	MAE	NRMSE	R2 score
Linear Regression	8380.82	0.148	0.491
KNeighborsRegressor	4532.4	0.103	0.751
GradientBoostingRegressor	6130.76	0.114	0.698
DecisionTreeRegressor	3172.23	0.084	0.833
RandomForestRegressor	3087.15	0.073	0.874
XGBRegressor	4084.23	0.081	0.848

In [60]:

```
Model_names = [model[0] for model in mod]
MAE = [model[1] for model in mod]

plt.bar(Model_names, MAE , color='brown')

plt.xticks(rotation=90)
plt.xlabel('Models')
plt.ylabel('MAE')
plt.title('MAE comparison between models')
plt.show()
```

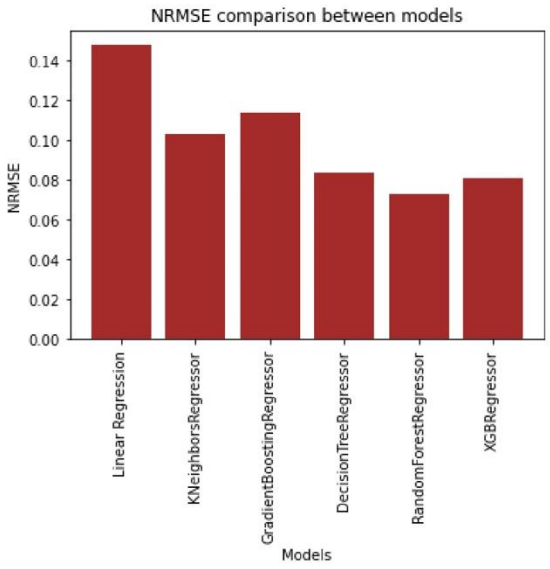


In [61]:

```
Model_names = [model[0] for model in mod]
NRMSE = [model[2] for model in mod]

plt.bar(Model_names, NRMSE , color='brown')

plt.xticks(rotation=90)
plt.xlabel('Models')
plt.ylabel('NRMSE')
plt.title('NRMSE comparison between models')
plt.show()
```

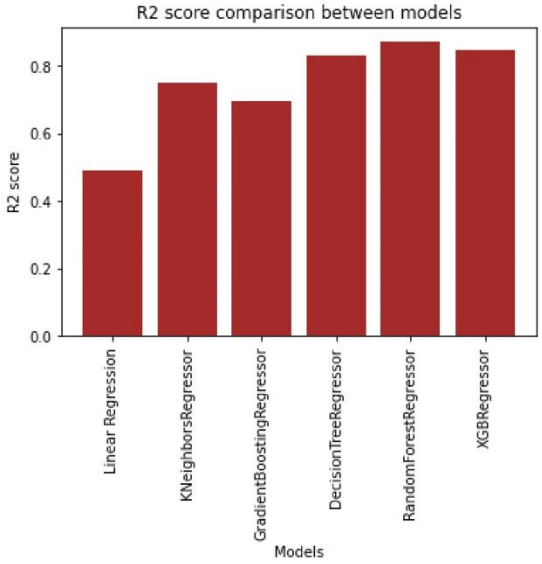


In [62]:

```
Model_names = [model[0] for model in mod]
R2_score = [model[3] for model in mod]

plt.bar(Model_names, R2_score , color='brown')

plt.xticks(rotation=90)
plt.xlabel('Models')
plt.ylabel('R2 score')
plt.title('R2 score comparison between models')
plt.show()
```



Method to implement ML algorithm user input values

In [63]:

```
def pred(seller,bed_c,layout_,propertytyp,localit,area_c,furnish,bath_c,city_):

    localit=localit.lower()
    if localit in df2['locality'].values:
        locality_index=df2.index[df2['locality']==localit][0]
        locality_c= df.at[locality_index, 'locality']
    else:
        locality_c=695


    seller_dict = {'Agent': 0, 'Builder': 1, 'Owner': 2}
    seller_c = seller_dict[seller]

    layout_dict = {'RK': 0, 'BHK': 1}
    layout_c = layout_dict[layout_]

    property_dict = {'Villa': 4, 'Independent House': 2, 'Apartment': 0, 'Independent Floor'
    property_c = property_dict[propertytyp]

    furnish_dict = {'Furnished': 0, 'Semifurnished': 1, 'Unfurnished': 2}
    furnish_c = furnish_dict[furnish]

    city_dict = {'Hyderabad': 3, 'Delhi': 2, 'Mumbai': 5, 'Chennai': 1, 'Kolkata': 4, 'Banga
    city_c = city_dict[city_]

    final_pred=[seller_c,bed_c,layout_c,property_c,locality_c,area_c,furnish_c,bath_c,city_c
    rent = round(rf.predict([final_pred])[0])
    return rent
```

Validating the ML Model

In [64]:

```
pred('Agent',3,'BHK','Apartment','miyapur',2000,'Unfurnished',3,'Hyderabad')
```

Out[64]:

```
31665
```

In [65]:

```
pred('Agent',3,'BHK','Independent House','Andheri',1800,'Furnished',3,'Mumbai')
```

Out[65]:

```
56283
```

In [66]:

```
pred('Agent',5,'BHK','Villa','KPHB',2200,'Unfurnished',4,'Hyderabad')
```

Out[66]:

```
36319
```

In [67]:

```
pred('Builder',2,'BHK','Apartment','Ameerpet', 1200,'Semifurnished',2,'Hyderabad')
```

Out[67]:

```
15240
```

In [68]:

```
pred('Owner',1,'RK','Apartment','SR Nagar',900,'Semifurnished',1,'Hyderabad')
```

Out[68]:

```
10979
```

Converting Model to bytestream data

In [69]:

```
with open('model.pkl', 'wb') as file:
    pickle.dump((rf,df,df2), file)
```

System-based User Interface

```
window = Tk()
window.title('A12 BATCH')
window.geometry("250x500")

seller_type_label = Label(window, text="Seller Type")
seller_type_label.grid(row=1, column=0)
seller_type_option = OptionMenu(window, StringVar(value="Agent"), "Builder", "Agent", "Owne
seller_type_option.grid(row=1, column=1)

bedroom_label = Label(window, text="Bedrooms",anchor='w')
bedroom_label.grid(row=2, column=0)
bedroom_option = OptionMenu(window, StringVar(value="1"), "1", "2", "3", "4", "5", "6", "7"
bedroom_option.grid(row=2, column=1)

layout_label = Label(window, text="Layout Type")
layout_label.grid(row=3, column=0)
layout_option = OptionMenu(window, StringVar(value="BHK"), "RK", "BHK")
layout_option.grid(row=3, column=1)

property_type_label = Label(window, text="Property Type")
property_type_label.grid(row=4, column=0)
property_type_option = OptionMenu(window, StringVar(value="Apartment"), "Villa", "Independe
property_type_option.grid(row=4, column=1)

locality_label = Label(window, text="Locality")
locality_label.grid(row=5, column=0)
locality_input = Entry(window)
locality_input.grid(row=5, column=1)

area_label = Label(window, text="Area in sqft")
area_label.grid(row=6, column=0)
area_input = Entry(window)
area_input.grid(row=6, column=1)

furnish_type_label = Label(window, text="Furnish Type")
furnish_type_label.grid(row=7, column=0)
furnish_type_option = OptionMenu(window, StringVar(value="Unfurnished"), "Furnished", "Semi
furnish_type_option.grid(row=7, column=1)

bathroom_label = Label(window, text="Bathroom")
bathroom_label.grid(row=8, column=0)
bathroom_option = OptionMenu(window, StringVar(value="1"), "1", "2", "3", "4", "5", "6", "7
bathroom_option.grid(row=8, column=1)

city_label = Label(window, text="City")
city_label.grid(row=9, column=0)
city_option = OptionMenu(window, StringVar(value="Hyderabad"), "Hyderabad", "Mumbai", "Chen
city_option.grid(row=9, column=1)

def get_input():
    seller_type = seller_type_option.cget("text")
    bedroom = int(bedroom_option.cget("text"))
    layout_type = layout_option.cget("text")
    property_type = property_type_option.cget("text")
    locality = locality_input.get()
    area = float(area_input.get())
    furnish_type = furnish_type_option.cget("text")
    city = city_option.cget("text")
    bathroom = int(bathroom_option.cget("text"))
    result1=pred(seller_type,bedroom,layout_type,property_type,locality,area,furnish_type,b
    return result1

def onClick():
    result1=get_input()
    result =f"Predicted rent value is {result1}"
    tkinter.messagebox.showinfo("Prediction",result)

button = Button(window, text="Predict",bg='lightgreen', command=onClick).place(x=100, y=370

label = Label(window)

window.mainloop()
```

Android Interface

In [73]:

```
from kivy.uix.gridlayout import GridLayout
from kivy.app import App
from kivy.uix.label import Label
from kivy.uix.button import Button
from kivy.uix.textinput import TextInput
from kivy.uix.spinner import Spinner
from kivy.config import Config
import pickle

Config.set('graphics', 'width', '250')
Config.set('graphics', 'height', '550')

class RentPredictionApp(App):
    def pred(self,seller,bed_c,layout_,propertytyp,localit,area_c,furnish,bath_c,city_):
        with open('model.pkl', 'rb') as f:
            rf,df,df2 = pickle.load(f)
        localit=localit.lower()
        if localit in df2['locality'].values:
            locality_index=df2.index[df2['locality']==localit][0]
            locality_c= df.at[locality_index, 'locality']
        else:
            locality_c=695


        seller_dict = {'Agent': 0, 'Builder': 1, 'Owner': 2}
        seller_c = seller_dict[seller]

        layout_dict = {'RK': 0, 'BHK': 1}
        layout_c = layout_dict[layout_]

        property_dict = {'Villa': 4, 'Independent House': 2, 'Apartment': 0, 'Independent F
        property_c = property_dict[propertytyp]

        furnish_dict = {'Furnished': 0, 'Semifurnished': 1, 'Unfurnished': 2}
        furnish_c = furnish_dict[furnish]

        city_dict = {'Hyderabad': 3, 'Delhi': 2, 'Mumbai': 5, 'Chennai': 1, 'Kolkata': 4, '
        city_c = city_dict[city_]

        final_pred=[seller_c,bed_c,layout_c,property_c,locality_c,area_c,furnish_c,bath_c,c
        rent = round(rf.predict([final_pred])[0])
        return rent
     # Define clear_inputs method
    def clear_inputs(self, instance):
        self.seller_type_option.text = "Agent"
        self.bedroom_option.text = "1"
        self.layout_option.text = "BHK"
        self.property_type_option.text = "Apartment"
        self.locality_input.text = ""
        self.area_input.text = ""
        self.furnishing_option.text = "Furnished"
        self.bathroom_option.text = "1"
        self.city_option.text = "Bangalore"
        self.rent_display.text = ""



    def build(self):
        layout = GridLayout(cols=2, spacing=10, padding=20)

        # Add Seller Type input
        layout.add_widget(Label(text="Seller Type"))
        self.seller_type_option = Spinner(text="Agent", values=("Builder", "Agent", "Owner"
        layout.add_widget(self.seller_type_option)
```

```
        # Add Bedrooms input
        layout.add_widget(Label(text="Bedrooms"))
        self.bedroom_option = Spinner(text="1", values=("1", "2", "3", "4", "5", "6", "7"))
        layout.add_widget(self.bedroom_option)

        # Add Layout Type input
        layout.add_widget(Label(text="Layout Type"))
        self.layout_option = Spinner(text="BHK", values=("RK", "BHK"))
        layout.add_widget(self.layout_option)

        # Add Property Type input
        layout.add_widget(Label(text="Property Type"))
        self.property_type_option = Spinner(text="Apartment", values=("Villa", "Independent
        layout.add_widget(self.property_type_option)

        # Add Locality input
        layout.add_widget(Label(text="Locality"))
        self.locality_input = TextInput(multiline=False)
        layout.add_widget(self.locality_input)

        #Add Area input
        layout.add_widget(Label(text="Area (sqft)"))
        self.area_input = TextInput(multiline=False)
        layout.add_widget(self.area_input)

        # Add Furnishing input
        layout.add_widget(Label(text="Furnishing"))
        self.furnishing_option = Spinner(text="Furnished", values=("Furnished", "Semifurnis
        layout.add_widget(self.furnishing_option)

        # Add Bathrooms input
        layout.add_widget(Label(text="Bathrooms"))
        self.bathroom_option = Spinner(text="1", values=("1", "2", "3", "4", "5"))
        layout.add_widget(self.bathroom_option)

        # Add City input
        layout.add_widget(Label(text="City"))
        self.city_option = Spinner(text="Hyderabad", values=("Bangalore", "Chennai", "Delhi
        layout.add_widget(self.city_option)

        # Add Button to calculate rent
        self.calculate_button = Button(text="Predict Rent",background_color=(0, 1, 0, 1))
        self.calculate_button.bind(on_press=self.show_rent)
        layout.add_widget(self.calculate_button)

        # Add Clear Button
        self.clear_button = Button(text="Clear",background_color=(1, 0, 0, 1))
        self.clear_button.bind(on_press=self.clear_inputs)
        layout.add_widget(self.clear_button)

        # Add Label to display Rent
        self.rent_display = Label(text="")
        layout.add_widget(self.rent_display)

        return layout

    def show_rent(self, instance):
        seller = self.seller_type_option.text
        bed_c = int(self.bedroom_option.text)
        layout_ = self.layout_option.text
        propertytyp = self.property_type_option.text
        localit = self.locality_input.text
        area_c = int(self.area_input.text)
        furnish = self.furnishing_option.text
        bath_c = int(self.bathroom_option.text)
        city_ = self.city_option.text
        rent = self.pred(seller,bed_c,layout_,propertytyp,localit,area_c,furnish,bath_c,city
```

```
        self.rent_display.text = "Predicted Rent is Rs. " + str(rent)

if __name__ == '__main__':
    RentPredictionApp().run()
```

```
[INFO   ] [Logger      ] Record log in /Users/praneetpeshkar/.kivy/logs/kivy_2
3-03-26_6.txt
[INFO   ] [Kivy        ] v2.1.0
[INFO   ] [Kivy        ] Installed at "/Users/praneetpeshkar/opt/anaconda3/li
b/python3.9/site-packages/kivy/__init__.py"
[INFO   ] [Python      ] v3.9.7 (default, Sep 16 2021, 08:50:36)
[Clang 10.0.0 ]
[INFO   ] [Python      ] Interpreter at "/Users/praneetpeshkar/opt/anaconda3/b
in/python"
[INFO   ] [Logger      ] Purge log fired. Processing...
[INFO   ] [Logger      ] Purge finished!
[INFO   ] [Factory     ] 189 symbols loaded
[INFO   ] [Image       ] Providers: img_tex, img_imageio, img_dds, img_sdl2, i
mg_pil (img_ffpyplayer ignored)
[INFO   ] [Text        ] Provider: sdl2
[INFO   ] [Window      ] Provider: sdl2
[INFO   ] [GL          ] Using the "OpenGL ES 2" graphics system
[INFO   ] [GL          ] Backend used <sdl2>
[INFO   ] [GL          ] OpenGL version <b'2.1 INTEL-18.8.6'>
[INFO   ] [GL          ] OpenGL vendor <b'Intel Inc.'>
[INFO   ] [GL          ] OpenGL renderer <b'Intel(R) HD Graphics 6000'>
[INFO   ] [GL          ] OpenGL parsed version: 2, 1
[INFO   ] [GL          ] Shading version <b'1.20'>
[INFO   ] [GL          ] Texture max size <16384>
[INFO   ] [GL          ] Texture max units <16>
[INFO   ] [Window      ] auto add sdl2 input provider
[INFO   ] [Window      ] virtual keyboard not allowed, single mode, not docked
[INFO   ] [Base        ] Start application main loop
[INFO   ] [GL          ] NPOT texture support is available
[INFO   ] [Base        ] Leaving application in progress...
```

Modified Hybrid Regression Algorithm

In [70]:

```
def hybrid(model1,model2):
    dict1={}
    mod1=model1.predict(X_test.values)
    mod2=model2.predict(X_test.values)
    for i in range(0,101):
        hybmod=((mod1*i)+(mod2*(100-i)))/100
        dict1[i]=r2_score(y_test,hybmod)
    ratio = max(dict1, key=dict1.get)
    return ratio
```

In [71]:

```
def hybrid_test(model1,model2):
    t=hybrid(model1,model2)
    mod1=model1.predict(X_test.values)
    mod2=model2.predict(X_test.values)
    mod=(mod1*t+mod2*(100-t))/100
    print('MAE',mae(y_test,mod))
    print('NRMSE', nrmse(y_test,mod))
    print('R2 Score',round(r2_score(y_test,mod),3))
```

In [72]:

```
hybrid_test(rf,xgb)
```

```
MAE 3256.173062879278
NRMSE 0.07138861411286201
R2 Score 0.881
```