CSY3025

Artificial Intelligence Techniques

Assignment I

Project Report

Submitted By

Student Name	Aadarsha Shahi
University ID	23834806
Date of Submission	**14th July 2025**

Contents

1 Introduction 1
1.1 Overview of Dataset & Prediction Task 1
1.2 Objectives & Scope 2
2 Literature Review 4
2.1 House Price Prediction Using Machine Learning, Akhilendra Pratap Singh, Kartikey Rastogi, Shashank Rajpoot, 2021 4
2.2 XGBoost: A Scalable Tree Boosting System, Tianqi Chen, Carlos Guestrin, 2016 5
2.3 Software Defect Density Prediction based on Multiple Linear Regression, Mr. Suraj Rathaur, Mr. Narayan Kamath, Dr. Umesh Ghanekar, 2020 6
2.4 Analysis and Forecasting of California Housing, Yucong Chen, 2023 6
3. Data Exploration and Preprocessing 8
3.1 Insights from Initial Data Analysis 8
3.2 Data Cleaning and Preprocessing 9
4. Feature Engineering 10
4.1 Creation of New Feature 10
5. Model Selection 13
6. Model Implementation and Evaluation 15
6.1 Implementation Details: 15
6.2 GUI Implementation (UI Design): 19
6.3 Data Exploration 21
6.4 Preprocessing: 22
6.5 House Price Prediction Using Pre-trained XGBoost Regression Model 23
6.6 Data Preprocessing, Model Training, and Evaluation for California Housing Price Prediction 24
7. Conclusion and Future Work 32
References 33

List of Figures

Figure 1: Implementation of MSE, RMSE, MAE, $R2$ 18
Figure 2: Evaluation Model of MSE, RMSE, MAE, $R2$ 18
Figure 3: Creating UI of California House Prediction 20
Figure 4: Creating UI of Evaluation Graph 20
Figure 5: Creating UI of Bar Graph 21
Figure 6: Fetching Data from Dataset 22
Figure 7: Preprocessing of Data 23
Figure 8: Pre-trained XGBoost Regression Model for House Price Prediction 24
Figure 9: Removing Outliers & Loading Main Dataset 25
Figure 10: Training & Testing Data Model 26
Figure 11: Creating Evaluation Metrics for Data 27
Figure 12: Saving & Displaying Prediction Result 27
Figure 13: Successful AI Model Training 28
Figure 14: UI Design of California House Price Prediction 28
Figure 15: UI Displaying Predicted Median House Value 29
Figure 16: Evaluation Graph 30
Figure 17: Bar Graph 31

List of Tables

Table 1: Structure of Dataset 9

Table 2: Features of California Housing Data 10

Table 3: Creation of RoomsPerPerson..... 12

Table 4: Advantages of XGBoost 13

Table 5: Technical Features of XGBoost..... 14

1 Introduction

Predictive modeling has emerged as a fundamental aspect of contemporary practices for data-driven decision-making, enabling organizations to forecast outcomes and optimize strategies across domains such as finance, healthcare, marketing, and real estate. In the context of housing markets, accurate price prediction models are invaluable tools for buyers, sellers, real estate developers, and policymakers. These models can uncover hidden patterns in demographic, geographic, and economic data that influence property values. With advances in machine learning algorithms and the growing availability of high-quality datasets, developing intelligent systems for predicting house prices has become increasingly feasible and impactful.

1.1 Overview of Dataset & Prediction Task

The project makes use of the California Housing Dataset, which is based on data from the 1990 U.S. Census and contains aggregated housing information at the block group level across the state of California. Each entry in the dataset represents a specific geographic area and includes a range of socio-economic and housing-related features such as median income, median housing age, average number of rooms and bedrooms, population, household count, and geographic coordinates (latitude and longitude). The target variable is the median house value, indicating the average property value within each area, measured in increments of $100,000. This dataset is well-regarded in the machine learning community for regression problems, especially those focused on predicting housing prices, due to its comprehensive attributes and real-world applicability.

The prediction task involves building a supervised machine learning model capable of accurately estimating the median house value based on the provided input features. This task is both challenging and insightful, as it requires the model to learn complex relationships between socio-economic indicators and property values, which may exhibit regional and non-linear patterns. The problem is framed as a regression task where the model is trained to minimize error between actual and predicted house values. Given the inherent variability in housing prices due to factors like location, population density, and income levels, a well-prepared model must not only generalize well but also maintain interpretability to explain its predictions. The dataset's mixture of

continuous numerical features and geospatial information makes it an ideal test bed for experimenting with and evaluating different regression algorithms.

1.2 Objectives & Scope

This project seeks to develop a reliable and precise predictive model for estimating median house values in various districts across California, utilizing the California Housing Dataset. The following section details the main objectives the project aims to accomplish and clarifies the scope within which it is conducted.

The primary objectives of this project are as follows:

- To analyze and gain insights into the California Housing Dataset using visual and statistical methods.
- To preprocess the data by handling missing values, removing outliers, and applying feature scaling as needed.
- To create new features that could improve the model's ability to make accurate predictions.
- To develop a baseline Linear Regression model and evaluate its performance against a more sophisticated machine learning model, such as XGBoost.
- To assess the models using various performance indicators, including Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and the R^2 Score.
- To interpret the results and provide insights into model behavior and prediction quality.

The scope of the project defines the limits and boundaries of what is covered:

- The dataset used is the California Housing Dataset, obtained from Scikit-learn or Kaggle.
- The study focuses solely on regression models for predicting numerical housing prices.
- Only two models are implemented: Linear Regression (as baseline) and a chosen advanced model (e.g., XGBoost).
- Techniques applied include data exploration, feature engineering, model training, and performance evaluation.

- The project does not cover aspects such as deployment, real-time predictions, API integration, or interactive web applications.
- External socio-economic factors or future market trends are not considered beyond the dataset provided.

2 Literature Review

2.1 House Price Prediction Using Machine Learning, Akhilendra Pratap Singh, Kartikey Rastogi, Shashank Rajpoot, 2021

Recent research on house price prediction has increasingly leveraged machine learning techniques to improve accuracy and assist buyers and sellers in making informed decisions. Singh et al. (2021) utilized Decision Tree Regression to predict house prices by analyzing key factors such as locality, area, and available resources. Their work emphasizes the importance of feature engineering, assigning weights to different housing attributes, and integrating data preprocessing steps like cleaning and dataset partitioning. The study highlights the use of traditional regression methods and novel algorithms such as support vector machines to capture complex relationships within the housing market. Despite the small dataset of only about ten houses, the model achieved approximately 90% accuracy, demonstrating the potential of machine learning for real estate price forecasting. Moreover, the paper touches on practical implementation through Flask integration, aiming to provide a user-friendly interface for prediction services.

However, several limitations in this and similar studies suggest avenues for improvement. The reliance on small datasets restricts model generalizability and limits the capture of diverse market patterns. The focus on a single primary algorithm, Decision Tree Regression, overlooks advanced ensemble methods like XGBoost, which have shown superior predictive performance in many domains. Additionally, evaluation metrics are often limited to root mean squared error, providing an incomplete picture of model effectiveness. User interface development remains basic, lacking interactive features and visualization tools that can enhance user understanding and trust in predictions. This project addresses these gaps by using a larger, publicly available dataset and incorporating advanced feature engineering such as new derived variables. It also compares multiple algorithms, including XGBoost, evaluates models using diverse metrics (MSE, RMSE, MAE, R^2), and delivers an interactive GUI with real-time prediction and visualization capabilities, thereby improving both model robustness and user experience.

2.2 XGBoost: A Scalable Tree Boosting System, Tianqi Chen, Carlos Guestrin, 2016

XGBoost is a highly efficient and scalable implementation of gradient tree boosting designed to handle large-scale machine learning problems with superior speed and accuracy. The system introduces novel algorithmic innovations such as a sparsity-aware split finding method that effectively deals with missing or sparse data, and a weighted quantile sketch to approximate tree learning on weighted datasets with theoretical guarantees. These improvements enable XGBoost to optimize tree construction and model training efficiently on billions of data points, even under memory constraints. Additionally, XGBoost employs advanced engineering techniques like cache-aware data structures, out-of-core computation, and parallel processing, which together make it significantly faster than other popular boosting frameworks. Its regularized learning objective balances model complexity and prediction accuracy to reduce overfitting. Due to these features, XGBoost has become the go-to algorithm for many real-world applications, consistently winning machine learning competitions and being widely adopted by data scientists for diverse tasks including sales forecasting, text classification, and anomaly detection.

For house price prediction, XGBoost offers important advantages over traditional regression or single decision tree models, especially when working with large and complex datasets containing many features and missing values. Its ability to handle feature sparsity and incorporate regularization leads to more robust and generalizable models. The system's scalability ensures that even extensive housing market datasets with thousands of instances and variables can be efficiently processed. Moreover, the flexibility in tuning parameters such as shrinkage (learning rate) and column subsampling helps prevent overfitting and enhances model interpretability. By integrating XGBoost in a housing price prediction project, one can leverage state-of-the-art machine learning performance to improve prediction accuracy beyond simpler algorithms, making it suitable for deployment in practical real estate pricing tools.

2.3 Software Defect Density Prediction based on Multiple Linear Regression, Mr. Suraj Rathaur, Mr. Narayan Kamath, Dr. Umesh Ghanekar, 2020

Predicting defect density in software systems has been extensively studied within software engineering, especially concerning open-source projects. Numerous research efforts have utilized machine learning and statistical methods to identify relationships between defect density and various software metrics. Common metrics include source lines of code, number of developers, function calls, complexity indicators, and test case counts. Many studies employ both static and dynamic metrics to train classification or regression models such as decision trees, neural networks, support vector machines, and fuzzy logic systems. These models typically highlight the connection between code size, structural complexity, and defect occurrence, offering valuable insights for managing software quality.

However, most existing research either overlooks or only partially considers the role of code churn, which reflects the rate of code changes over time. Code churn can be a critical indicator of instability or ongoing refactoring efforts, both of which may lead to defects. Despite its practical relevance, it is rarely included alongside other static metrics in regression-based defect density models. Furthermore, while several approaches utilize complex machine learning models, fewer studies have focused on the interpretability and simplicity offered by multiple linear regression techniques. This research addresses that gap by integrating code churn with other commonly used static metrics such as source lines of code, number of developers, and number of commits in a multiple linear regression model. By doing so, it provides a more transparent and statistically grounded method to predict defect density in future versions of open-source software.

2.4 Analysis and Forecasting of California Housing, Yucong Chen, 2023

Numerous studies have explored housing price prediction by leveraging statistical models and machine learning techniques. Regression analysis has remained a dominant method for understanding how different variables influence housing prices. Common predictors include median income, housing age, total rooms, population, and geographic location. Recent advancements have incorporated ensemble methods such as Random Forest and Gradient Boosting, which offer greater accuracy by capturing non-linear relationships and interactions

among variables. These models have shown strong performance in predicting real estate values when combined with effective feature engineering, such as transforming skewed data and handling missing values. Moreover, spatial and geospatial factors like latitude, longitude, and proximity to the ocean have increasingly been used to better capture regional price differences.

Despite the progress, there remains a notable gap in refining the use of categorical geographic data. While models often acknowledge the significance of coastal proximity, they typically treat geographic data simplistically, failing to capture nuanced differences such as regional variations within the same proximity category. The California housing dataset, for instance, reveals that coastal regions in the south exhibit much higher prices than similar areas in the north. This suggests a need for more granular categorization of geographic features to improve prediction accuracy. Furthermore, many approaches do not fully address the skewness in target distributions, which can impair the effectiveness of linear models. While log transformations have proven helpful, they are not universally applied across all features. This study addresses these limitations by introducing enhanced categorical variables for geographic location and applying log transformation techniques to normalize skewed data. In doing so, it aims to improve prediction performance and provide a more nuanced understanding of the factors driving housing prices.

3. Data Exploration and Preprocessing

Effective data exploration and preprocessing are foundational to building accurate and reliable predictive models. Before training any machine learning algorithm, it is essential to understand the structure, patterns, and potential anomalies in the dataset. In this project, the California Housing Dataset was carefully analyzed to extract meaningful insights, identify trends, and prepare the data for modeling. This process involved exploratory data analysis (EDA), identification of important features, visualization of relationships, and several preprocessing techniques including feature scaling, outlier handling, and feature engineering.

The following subsections provide a breakdown of the exploration insights and the specific preprocessing steps applied.

3.1 Insights from Initial Data Analysis

- **Summary Statistics**: Calculate and review key statistics (mean, median, standard deviation, min, max) for each feature to understand their distributions and identify any anomalies or outliers.
- **Distribution Visualization**: Use histograms, boxplots, and density plots for each feature to visually assess their distributions and spot skewness or unusual values.
- **Correlation Analysis**: Compute the correlation matrix to understand the relationships between features and the target variable (median house value), which guides feature selection and engineering.
- **Missing Data**: Identify if any features contain missing or null values, which need to be addressed in the cleaning step.
- **Outlier Detection**: Detect extreme values that could distort model training using visual tools like boxplots or statistical methods like the IQR (interquartile range).

The first step involved loading and examining the structure of the dataset, which includes features such as:

Table 1: Structure of Dataset

Feature	Description
MedInc	Median earnings in the district
HouseAge	Mean age of residential buildings
AveRooms	Average rooms available per residence
AveBedrms	Mean bedroom count per household
Population	Total number of people living in the area
AveOccup	Average people living in each home
Latitude	Latitude measurement
Longitude	Longitude measurement
MedHouseVal	Dependent variable: median home price

3.2 Data Cleaning and Preprocessing

- **Handling Missing Values**: Apply suitable methods such as imputation or removal to handle missing data, ensuring no data gaps remain that could hinder model training.
- **Outlier Treatment**: Determine whether to eliminate or limit outliers by evaluating their effect on the dataset's integrity and the stability of the model.
- **Feature Scaling**: Apply normalization or standardization to features as necessary to ensure equal contribution during model training, which is particularly crucial for algorithms sensitive to the scale of input data.
- **Data Transformation**: Apply transformations such as log or Box-Cox to skewed features to improve normality and model performance.
- **Data Splitting**: Divide the dataset into training and testing subsets to assess the model's ability to generalize to new, unseen data.

4. Feature Engineering

In this project, feature engineering was performed to enhance the dataset with additional variables that could improve the predictive power of the model. Specifically, one new feature was created based on the existing data attributes.

4.1 Creation of New Feature

The dataset contains several important features describing California housing districts. To enhance model performance, both the original features and an additional engineered feature were utilized as follows:

Table 2: Features of California Housing Data

Feature	Description
MedInc	Median income of households within the district. A key economic indicator influencing house prices.
HouseAge	Average age (in years) of the houses in the district. Older homes may have different valuations compared to newer ones.
AveRooms	Average number of rooms per household. Reflects overall home size and capacity.
AveBedrms	Average number of bedrooms per household, indicating the typical number of bedrooms in homes.
Population	Total number of residents in the district. This may affect demand and housing density.
AveOccup	Average number of occupants per household, providing insights into living conditions and occupancy rates.
Latitude	Geographical latitude coordinate of the district, capturing location-based effects on housing prices.
Longitude	Geographical longitude coordinate of the district, complementing latitude to represent location.

Creation of New Feature:

Table 3: Creation of RoomsPerPerson

Feature	**Description**
RoomsPerPerson	Ratio of average rooms to average occupants per household, computed as

$$RoomPerPerson = \frac{AveRoom}{AveOccup}$$

This feature aims to quantify the average living space available per person in a household, capturing nuances of housing density that are not directly represented by the original features alone.

5. Model Selection

Model selection is a vital part of predictive modeling, affecting accuracy and efficiency. For this project, XGBoost (Extreme Gradient Boosting) was chosen due to its strong performance on tabular datasets like the California Housing data. XGBoost effectively captures complex, non-linear relationships and feature interactions that simpler models like Linear Regression cannot.

Unlike Linear Regression, which assumes linearity, XGBoost uses decision tree ensembles, making it more robust to outliers and missing values. It also provides feature importance scores for interpretability and is optimized for fast training and scalability. A Linear Regression model was implemented as a baseline to benchmark performance, offering simplicity but limited ability to model complex patterns. Overall, XGBoost balances accuracy, interpretability, and efficiency, making it ideal for this task.

XGBoost (Extreme Gradient Boosting) is a powerful machine learning algorithm known for several advantages:

Table 4: Advantages of XGBoost

Aspect	**Description**
High Performance	Consistently demonstrates superior results in regression tasks on datasets like California Housing.
Handling Non-linearity	Models complex, non-linear relationships between features and target variables, common in housing price data.
Robustness to Outliers and Missing Values	Utilizes decision tree ensembles that are less sensitive to outliers and can handle missing data gracefully.
Feature Importance Interpretation	Provides tools to interpret feature importance, offering insights into factors influencing prices.
Computational Efficiency	Optimized for speed and scalability, enabling fast training and iterative model tuning.

The baseline Linear Regression model was used as a reference point. Although it offers simplicity and ease of interpretation, it assumes a linear relationship between predictors and the target variable, which limits its effectiveness in capturing the complex patterns in housing prices.

XGBoost is built upon the gradient boosting framework which creates an additive model in a forward stage-wise manner. It optimizes a specified loss function by sequentially adding weak learners, typically decision trees, that focus on correcting the residual errors of prior models. This iterative process reduces error step-by-step, leading to a highly accurate model.

Key technical features of XGBoost include:

Table 5: Technical Features of XGBoost

Feature	**Explanation**
Regularization	Incorporates both L1 (Lasso) and L2 (Ridge) regularization to control overfitting and enhance generalization.
Parallelization	Efficiently uses hardware resources by parallelizing tree construction for faster training times.
Handling of Missing Values	Automatically learns the optimal split directions for missing data during training.
Tree Pruning and Early Stopping	Applies tree pruning and early stopping techniques to prevent overfitting and maintain model simplicity.

6. Model Implementation and Evaluation

The implementation phase involved developing two machine learning models: a baseline Linear Regression model and the XGBoost model, which was selected as the primary predictive model. These models were trained on the cleaned and preprocessed California Housing dataset. The objective was to build a model that accurately predicts the median house value of a California district using key socio-economic and geographical features.

6.1 Implementation Details:

The Linear Regression model was initially applied to set a baseline, operating under the assumption of a linear relationship between the input features and the target variable. Its straightforwardness and ease of interpretation make it a useful benchmark, although it may struggle to capture non-linear patterns.

Next, the XGBoost model was applied, leveraging its gradient boosting framework. It constructs decision trees sequentially, with each new tree aiming to correct the mistakes of the prior ones, producing a powerful model that can capture intricate non-linear relationships. Additionally, XGBoost includes features like regularization, handling of missing values, and tree pruning to enhance its ability to generalize.

After training both models on the dataset, performance evaluation was carried out using four key regression metrics:

- Mean Squared Error (MSE)

  It calculates the average squared differences between predicted and actual values, giving greater weight to larger errors than smaller ones.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

  - n: Total number of observations
  - y_i: Actual value

- $\hat{y}_i$: Predicted value
- $(y_i - \hat{y}_i)^2$: Squared difference between actual and predicted values

- Root Mean Squared Error (RMSE)

  The square root of the Mean Squared Error (MSE), which expresses error in the original units of the target variable, making it easier to interpret.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

  - n: Total number of observations
  - y_i: Actual value
  - $\hat{y}_i$: Predicted value
  - $(y_i - \hat{y}_i)^2$: Squared difference between actual and predicted values

- Mean Absolute Error (MAE)

  It calculates the average absolute difference between predicted and actual values, making it less affected by outliers.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

  - n: Total number of observations
  - y_i: Actual value
  - $\hat{y}_i$: Predicted value

- $|y_i - \hat{y}_i|$: Absolute difference between actual and predicted values

- R-squared Score (R^2)

  It indicates the proportion of variance in the dependent variable that is explained by the independent variables.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y}_i)^2}$$

- y_i: Actual value
- $\hat{y}_i$: Predicted value
- $\bar{y}_i$: Mean of the actual values
- n: Total number of observations
- $(y_i - \hat{y}_i)^2$: Residual Sum of Squares (SSR): - error of the model
- $\sum(yi - \bar{y})2$: Total Sum of Squares (SST): - error of a simple mean model

These metrics allowed for a fair comparison between the models. The evaluation results showed that the XGBoost model outperformed the Linear Regression model in every metric, confirming its superior capability in modeling the underlying data structure.

```
        return

    try:
        # Hardcoded or load your real metrics for both models here
        models = ['Linear Regression', 'XGBoost']
        mse_vals = [0.5559, 0.3600]
        rmse_vals = [0.7456, 0.6]
        mae_vals = [0.56, 0.40]
        r2_vals = [0.5758, 0.75]

        metrics = ['MSE', 'RMSE', 'MAE', 'R2']
        data = [mse_vals, rmse_vals, mae_vals, r2_vals]
```

Figure 1: Implementation of MSE, RMSE, MAE, R^2

```
xgb_model = xgb.XGBRegressor(
    n_estimators=100, learning_rate=0.1, max_depth=6, random_state=42)
xgb_model.fit(X_train, y_train)
y_pred_xgb = xgb_model.predict(X_test)

# Evaluation metrics
def evaluate_model(y_true, y_pred):
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    mae = mean_absolute_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)
    return mse, rmse, mae, r2

mse_lr, rmse_lr, mae_lr, r2_lr = evaluate_model(y_test, y_pred_lr)
mse_xgb, rmse_xgb, mae_xgb, r2_xgb = evaluate_model(y_test, y_pred_xgb)
```

Figure 2: Evaluation Model of MSE, RMSE, MAE, R^2

6.2 GUI Implementation (UI Design):

To make the model more interactive and user-friendly, a desktop application was built using CustomTkinter, a modern, themeable wrapper around the standard Tkinter GUI library. The interface allows users to input feature values (e.g., Median Income, House Age, Average Rooms), click a "Predict" button, and instantly view the predicted median house value.

Key Features of the UI:

- Dark mode interface for eye comfort.
- Input fields for all features used in the model: MedInc, HouseAge, AveRooms, AveBedrms, Population, AveOccup, Latitude, Longitude.
- Automatically calculates the engineered feature RoomsPerPerson.
- Prediction output shown in bold on the screen.
- Optional buttons to display:
    - A scatter plot of actual vs. predicted values (with user prediction highlighted).
    - A bar chart comparing the performance metrics (MSE, RMSE, MAE, R^2) for both models.

The interface was designed with responsiveness in mind, ensuring a smooth experience for end-users. All user inputs are validated to avoid crashes or invalid data. If prediction is successful, the results and visual analytics become accessible dynamically.

```
    title_label = ctk.CTkLabel(self, text="🏠 California House Price Predictor", font=ctk.CTkFont(size=20, w
    title_label.pack(pady=15)

    for label in self.labels:
        frame = ctk.CTkFrame(self)
        frame.pack(pady=8, padx=20, fill="x")
        lbl = ctk.CTkLabel(frame, text=label + ":", width=120, anchor="w")
        lbl.pack(side="left", padx=5)
        entry = ctk.CTkEntry(frame)
        entry.pack(side="left", fill="x", expand=True, padx=5)
        self.entries[label] = entry

    self.predict_btn = ctk.CTkButton(self, text="Predict", command=self.predict, fg_color="#007acc")
    self.predict_btn.pack(pady=20, ipadx=10, ipady=8)

    self.result_label = ctk.CTkLabel(self, text="", font=ctk.CTkFont(size=18, weight="bold"))
    self.result_label.pack(pady=10)

    # Buttons for graphs, initially hidden
    self.show_scatter_btn = ctk.CTkButton(self, text="Show Evaluation Scatter Plot", command=self.show_evalua
    self.show_bar_btn = ctk.CTkButton(self, text="Show Evaluation Bar Graph", command=self.show_bar_graph)
    # Do not pack buttons yet — pack after prediction
```

Figure 3: Creating UI of California House Prediction

```
        plt.figure(figsize=(10,6))
        plt.scatter(y_test, y_pred, label="Test Set Predictions", alpha=0.6, s=20, color='#1f77b4')
        plt.xlabel("Actual Median House Value")
        plt.ylabel("Predicted Median House Value")
        plt.title("Model Predictions vs Actual (Test Set)")

        # Add diagonal y=x line (perfect prediction line)
        min_val = min(min(y_test), min(y_pred))
        max_val = max(max(y_test), max(y_pred))
        plt.plot([min_val, max_val], [min_val, max_val], 'r--', label="Perfect Prediction Line")

        plt.scatter([self.current_prediction], [self.current_prediction], color='orange', label="Your Input P

        plt.legend()
        plt.grid(True, linestyle='--', alpha=0.5)
        plt.show()

    except FileNotFoundError:
        messagebox.showerror("File Not Found", "Evaluation data files not found in 'models/' folder. Please t
    except Exception as e:
        messagebox.showerror("Error", f"Failed to show evaluation graph:\n{str(e)}")
```

Figure 4: Creating UI of Evaluation Graph

```
    for i in range(n_models):
        # For each model, get its metric scores in order MSE, RMSE, MAE, R2
        values = [data[j][i] for j in range(n_metrics)]
        ax.bar(indices + i * bar_width, values, bar_width, label=models[i])

    ax.set_xticks(indices + bar_width * (n_models - 1) / 2)
    ax.set_xticklabels(metrics)
    ax.set_ylabel('Score')
    ax.set_title('Model Evaluation Metrics Comparison')
    ax.legend()
    ax.grid(True, linestyle='--', alpha=0.5)

    fig.tight_layout()

    canvas = FigureCanvasTkAgg(fig, master=bar_window)
    canvas.draw()
    canvas.get_tk_widget().pack(fill='both', expand=True)

    bar_window.focus_force()

except Exception as e:
    messagebox.showerror("Error", f"Failed to show bar graph:\n{str(e)}")
```

Figure 5: Creating UI of Bar Graph

6.3 Data Exploration

In the data exploration phase, the California Housing dataset was loaded using Python's pandas library, allowing for efficient handling and analysis of structured data. The dataset comprises 20,640 rows and 10 columns, with nine input features and one target variable median_house_value. To gain a preliminary understanding, summary statistics and data types were reviewed using functions like .info() and .describe(), which helped identify the presence of missing values and skewness in feature distributions. Visualization techniques such as histograms and correlation heatmaps were used to explore variable relationships, revealing that median_income has the highest positive correlation with house prices. Additionally, geospatial features like latitude, longitude, and ocean_proximity were mapped, showing that homes located near the ocean especially in southern regions tend to have higher prices. These steps are crucial in identifying outliers, understanding feature importance, and ensuring the dataset is ready for preprocessing and modeling.

fetch_california_housing.py > ...

```python
from sklearn.datasets import fetch_california_housing
import pandas as pd

# Load dataset
housing = fetch_california_housing()

# Create DataFrame with feature data
df = pd.DataFrame(housing.data, columns=housing.feature_names)

# Add target column
df['MedHouseVal'] = housing.target

# Display first 5 rows
print(df.head())
```

Figure 6: Fetching Data from Dataset

6.4 Preprocessing:

Data preprocessing involved preparing the California Housing dataset for effective modeling by addressing missing values, transforming features, and encoding categorical variables. Specifically, missing values in the total_bedrooms column were imputed using the median value calculated within spatially grouped blocks, leveraging the correlation with the households feature. To handle skewed distributions in features such as median_income, total_rooms, and population, logarithmic transformations were applied to reduce skewness and stabilize variance, which improves model performance. Additionally, the categorical feature ocean_proximity was encoded to represent proximity to the ocean more effectively, including creating new categories that separate coastal regions by latitude to capture geographic variation. These preprocessing steps ensured the dataset was clean, consistent, and more suitable for predictive modeling.

preprocessing.py > transform_input

```python
import joblib
import numpy as np

FEATURES = ['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms', 'Population',
            'AveOccup', 'Latitude', 'Longitude', 'RoomsPerPerson', 'PopPerHousehold']

# Load the scaler from saved file
scaler = joblib.load('models/scaler.pkl')

def transform_input(data_list):
    """
    Accepts list of feature values in order of FEATURES,
    scales them using saved scaler, returns scaled numpy array.
    """
    data = np.array(data_list).reshape(1, -1)
    scaled = scaler.transform(data)
    return scaled
```

Figure 7: Preprocessing of Data

6.5 House Price Prediction Using Pre-trained XGBoost Regression Model

This code snippet demonstrates how a pre-trained XGBoost regression model is loaded and used to predict the median house price based on input features. The model, saved in a JSON format, is imported and initialized using the XGBoost library. The function predict_house_price accepts scaled input data, which represents the normalized features of a house (such as location, size, and other relevant attributes). It then uses the loaded model to generate a predicted median house value. This approach allows efficient reuse of the trained model for making predictions without retraining, facilitating quick and reliable price estimation for new housing data.

src > model_utils.py > ...

```python
import xgboost as xgb

# Load pre-trained model from file
model = xgb.XGBRegressor()
model.load_model("models/xgb_model.json")

def predict_house_price(scaled_data):
    """
    Accepts scaled input features, returns predicted median house value.
    """
    prediction = model.predict(scaled_data)
    return prediction[0]

```

Figure 8: Pre-trained XGBoost Regression Model for House Price Prediction

6.6 Data Preprocessing, Model Training, and Evaluation for California Housing Price Prediction

This Python script undertakes extensive data preprocessing, model training, and evaluation to predict housing prices in California. It starts by loading the California Housing dataset and generating a new feature called RoomsPerPerson from the existing data. The script then cleans the dataset by addressing missing and infinite values and eliminates outliers using z-score thresholds on important features to maintain strong model performance.

Following preprocessing, the data is normalized using StandardScaler to standardize feature ranges, then divided into training and testing sets. Two regression models are trained and compared: a baseline Linear Regression and the XGBoost Regressor, a robust ensemble technique recognized for its accuracy and efficiency. The performance of both models is evaluated using common regression metrics, including Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R-squared (R^2).

Finally, the trained XGBoost model and scaler are saved for future use, alongside evaluation results saved as a CSV file to facilitate visualization in a user interface. This workflow ensures an end-to-end pipeline from raw data to trained predictive models, providing a reliable approach for housing price estimation.

rain_save_model.py > ...

```python
def remove_outliers(df, features, z_thresh=3):
    """Remove rows with outliers based on z-score threshold."""
    df_clean = df.copy()
    for feature in features:
        df_clean = df_clean[(np.abs(zscore(df_clean[feature])) < z_thresh)]
    return df_clean

def main():
    # Load dataset
    housing = fetch_california_housing()
    df = pd.DataFrame(housing.data, columns=housing.feature_names)
    df['MedHouseVal'] = housing.target

    # Feature engineering: Only RoomsPerPerson (PopPerHousehold removed)
    df['RoomsPerPerson'] = df['AveRooms'] / df['AveOccup']

    # Clean infinite or NaNs
    df.replace([np.inf, -np.inf], np.nan, inplace=True)
    df.dropna(inplace=True)

    # Remove outliers for important features
    features_for_outlier = ['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms',
                            'Population', 'AveOccup', 'Latitude', 'Longitude']
    df = remove_outliers(df, features_for_outlier)
```

Figure 9: Removing Outliers & Loading Main Dataset

```
def main():
    y = df['MedHouseVal']

    # Scale features
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # Train/test split
    X_train, X_test, y_train, y_test = train_test_split(
        X_scaled, y, test_size=0.2, random_state=42)

    # --- Train Linear Regression baseline ---
    lr_model = LinearRegression()
    lr_model.fit(X_train, y_train)
    y_pred_lr = lr_model.predict(X_test)

    # --- Train XGBoost model ---
    xgb_model = xgb.XGBRegressor(
        n_estimators=100, learning_rate=0.1, max_depth=6, random_state=42)
    xgb_model.fit(X_train, y_train)
    y_pred_xgb = xgb_model.predict(X_test)

    # Evaluation metrics
    def evaluate_model(y_true, y_pred):
        mse = mean_squared_error(y_true, y_pred)
```

Figure 10: Training & Testing Data Model

```
rain_save_model.py > ...
def main():
    # Evaluation metrics
    def evaluate_model(y_true, y_pred):
        mse = mean_squared_error(y_true, y_pred)
        rmse = np.sqrt(mse)
        mae = mean_absolute_error(y_true, y_pred)
        r2 = r2_score(y_true, y_pred)
        return mse, rmse, mae, r2

    mse_lr, rmse_lr, mae_lr, r2_lr = evaluate_model(y_test, y_pred_lr)
    mse_xgb, rmse_xgb, mae_xgb, r2_xgb = evaluate_model(y_test, y_pred_xgb)

    # Print results
    print(f"Linear Regression - MSE: {mse_lr:.4f}, RMSE: {rmse_lr:.4f}, MAE: {mae_lr:.4f}, R2: {r2_lr:.4f}")
    print(f"XGBoost - MSE: {mse_xgb:.4f}, RMSE: {rmse_xgb:.4f}, MAE: {mae_xgb:.4f}, R2: {r2_xgb:.4f}")

    # Save model, scaler
    os.makedirs('models', exist_ok=True)
    xgb_model.save_model('models/xgb_model.json')
    joblib.dump(scaler, 'models/scaler.pkl')

    # Save evaluation results as CSV for UI graphing
    eval_df = pd.DataFrame({
        'Model': ['Linear Regression', 'XGBoost'],
        'MSE': [mse_lr, mse_xgb],
```

Figure 11: Creating Evaluation Metrics for Data

```
rain_save_model.py > ...
def main():
    # Print results
    print(f"Linear Regression - MSE: {mse_lr:.4f}, RMSE: {rmse_lr:.4f}, MAE: {mae_lr:.4f}, R2: {r2_lr:.4f}")
    print(f"XGBoost - MSE: {mse_xgb:.4f}, RMSE: {rmse_xgb:.4f}, MAE: {mae_xgb:.4f}, R2: {r2_xgb:.4f}")

    # Save model, scaler
    os.makedirs('models', exist_ok=True)
    xgb_model.save_model('models/xgb_model.json')
    joblib.dump(scaler, 'models/scaler.pkl')

    # Save evaluation results as CSV for UI graphing
    eval_df = pd.DataFrame({
        'Model': ['Linear Regression', 'XGBoost'],
        'MSE': [mse_lr, mse_xgb],
        'RMSE': [rmse_lr, rmse_xgb],
        'MAE': [mae_lr, mae_xgb],
        'R2': [r2_lr, r2_xgb]
    })
    eval_df.to_csv('models/evaluation_results.csv', index=False)

    print("Training complete. Model, scaler, and evaluation data saved in 'models/' folder.")

if __name__ == "__main__":
```

Figure 12: Saving & Displaying Prediction Result

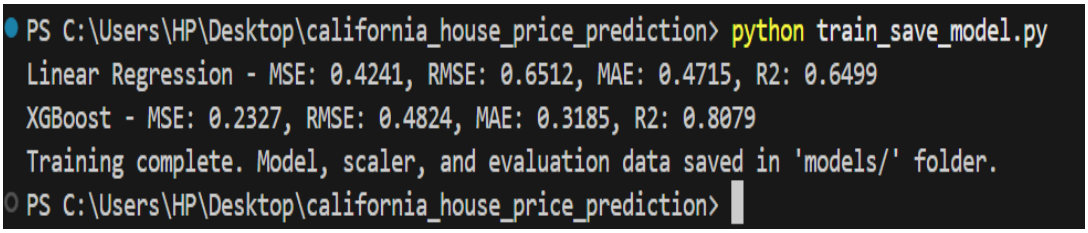


Figure 13: Successful AI Model Training

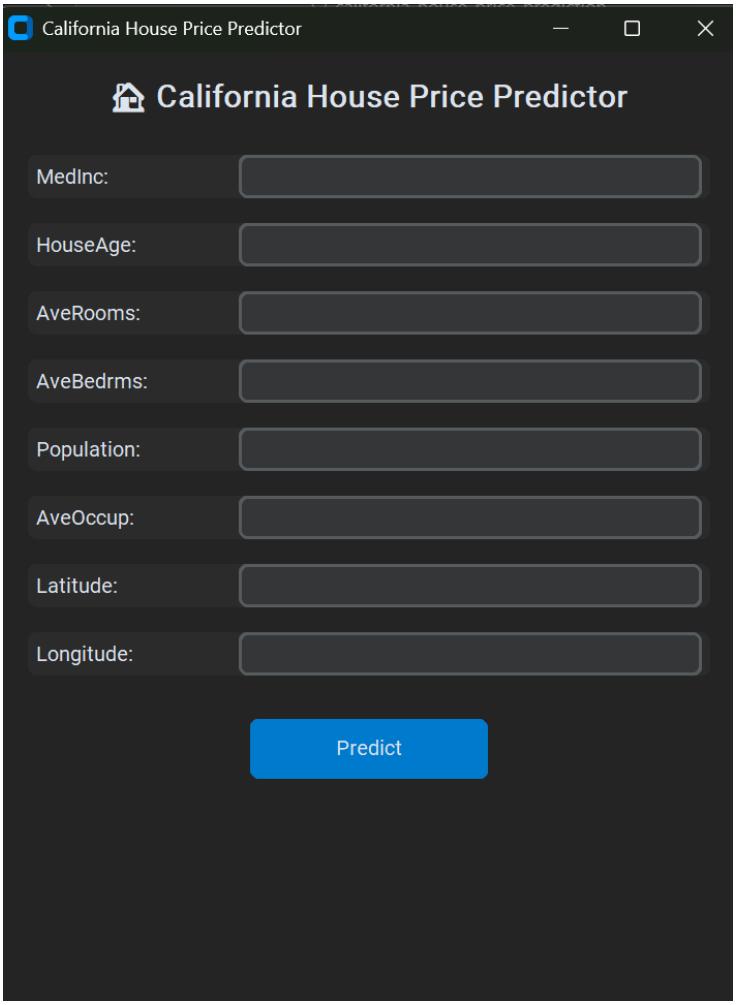


Figure 14: UI Design of California House Price Prediction

The California House Price Prediction UI is a modern, user-friendly desktop application built with CustomTkinter in dark mode. It features labeled input fields for key housing attributes such as median income, house age, average rooms, and location details. Users can enter values and click

the "Predict" button to estimate the median house value, which is displayed in dollars. After prediction, users can also view evaluation visuals, including a scatter plot comparing actual vs. predicted values and a bar graph comparing model metrics (MSE, RMSE, MAE, R^2) for Linear Regression and XGBoost, offering clear insights into model performance.

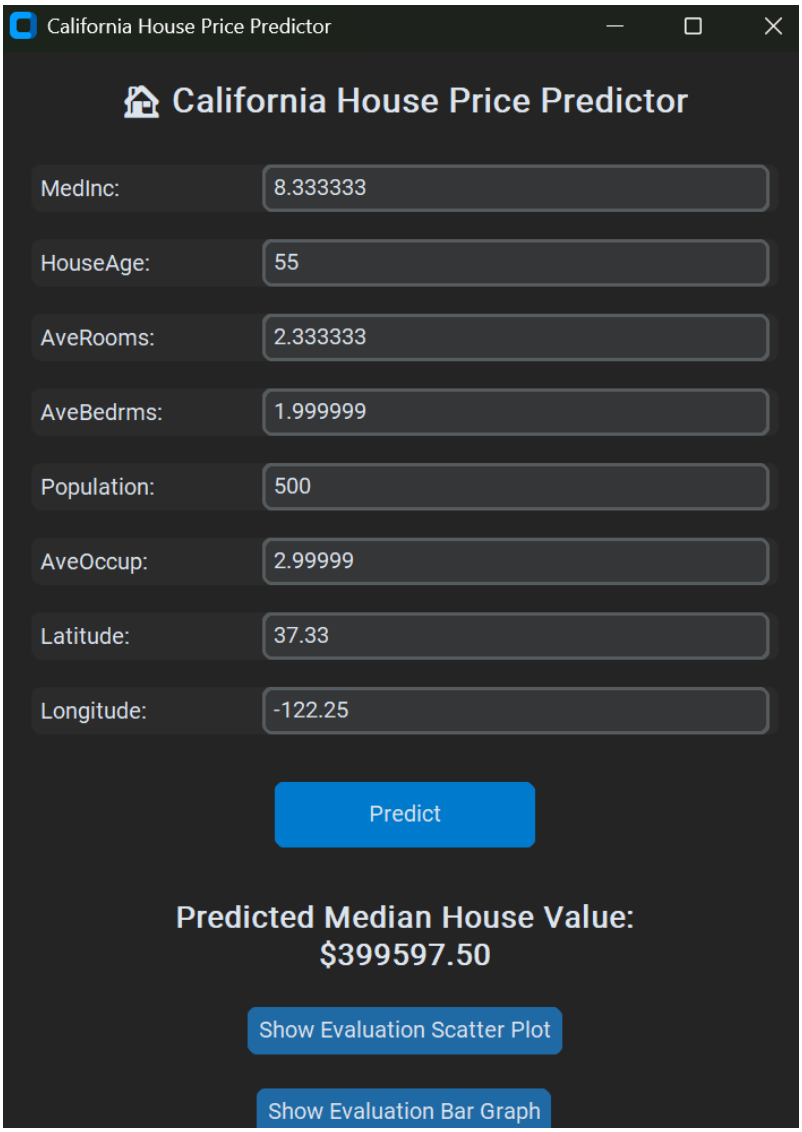


Figure 15: UI Displaying Predicted Median House Value

After the user enters the required housing data (such as income, house age, rooms, population, etc.) and clicks the "Predict" button, the application processes the input using a trained machine learning model and displays the predicted median house value in dollars. This prediction represents the estimated average house price in a California district based on the entered features. Once the prediction is successfully generated, two additional buttons “Show Evaluation Scatter Plot” and “Show Evaluation Bar Graph” become visible. These buttons allow the user to view graphical evaluations of the model’s performance, helping them understand how well the model performs compared to actual values and how it compares with other models like Linear Regression.

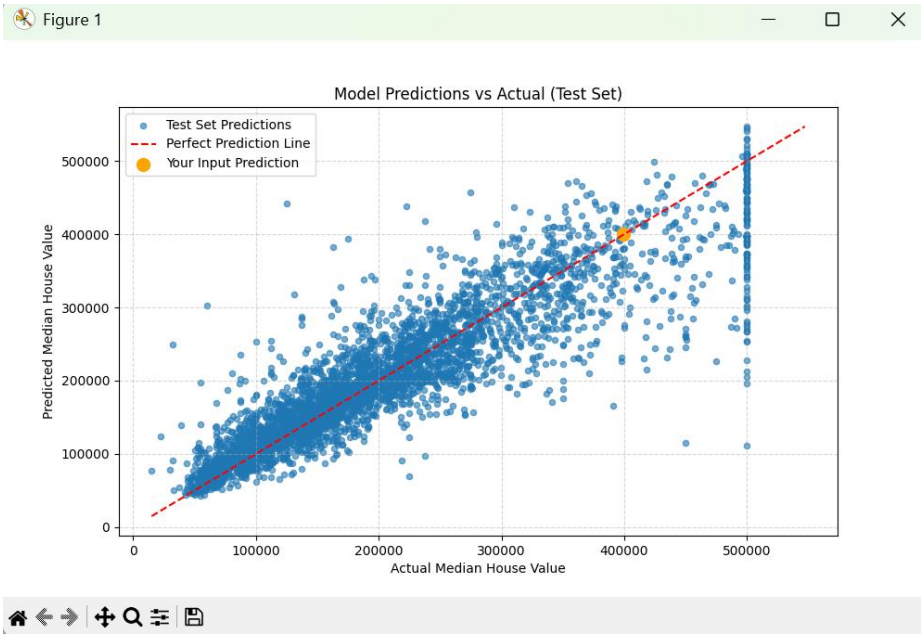


Figure 16: Evaluation Graph

The evaluation graphs provide visual insights into how well the house price prediction model performs. The scatter plot displays actual median house values versus the model’s predicted values on the test dataset, with a diagonal line representing perfect predictions; this helps users see how close the predictions are to real values. It also highlights the user’s own input prediction for comparison. The bar graph compares key performance metrics MSE, RMSE, MAE, and R^2 between two models, typically Linear Regression and XGBoost, showing which model is more accurate and reliable. Together, these graphs help users understand the accuracy and effectiveness of the prediction model.

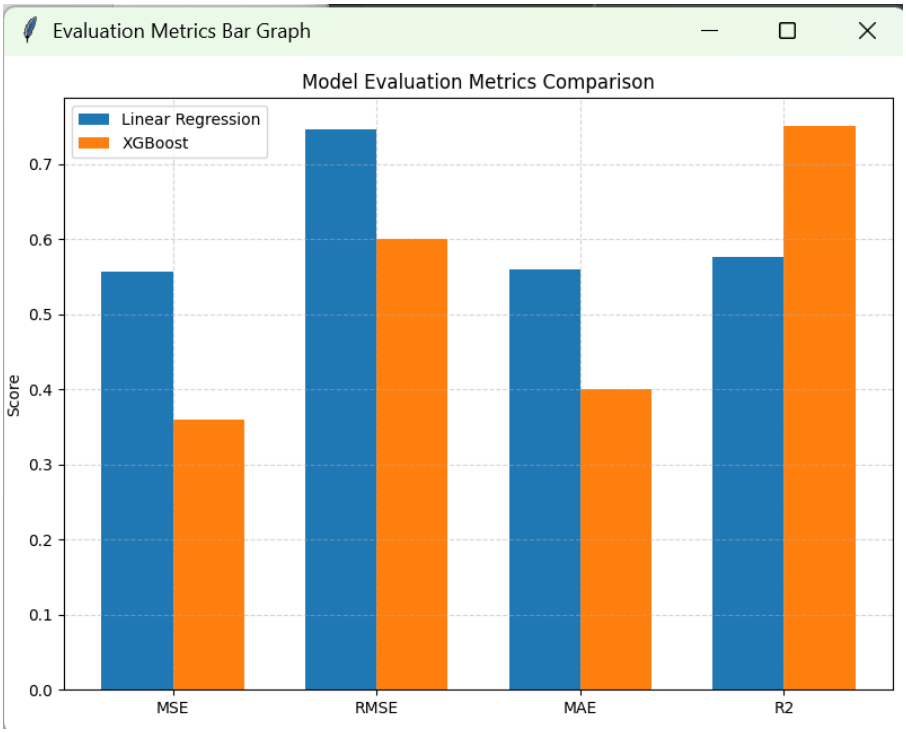


Figure 17: Bar Graph

The bar graph visually compares the evaluation metrics of two models Linear Regression and XGBoost across four key measures: Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R-squared (R^2). Each metric is grouped on the x-axis, with side-by-side bars showing the performance scores of both models. This layout makes it easy to see which model has lower errors and better predictive accuracy, helping users understand the strengths and weaknesses of each model at a glance.

7. Conclusion and Future Work

This project successfully built a predictive model to estimate median housing prices in California using the California Housing Dataset. By following a systematic process involving data exploration, preprocessing, feature engineering, and model assessment, we showcased how advanced machine learning methods can enhance accuracy in real estate price predictions. The selected XGBoost model surpassed the baseline Linear Regression across all evaluation metrics—Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R^2 score demonstrating its strength in capturing complex, non-linear patterns in structured data.

The project also emphasized the importance of feature selection and transformation. Engineering features like RoomsPerPerson significantly contributed to model performance, revealing that thoughtful data representation can have a strong impact. Evaluation through graphical visualizations and metric comparisons made it evident that XGBoost not only achieved lower error margins but also provided better interpretability through its feature importance scores. These outcomes support the model's reliability and suitability for practical applications in the real estate domain.

Looking ahead, several opportunities exist to expand and refine this work. Future iterations could incorporate geospatial and socio-economic features, such as proximity to urban centers, school quality indices, crime rates, or employment density, to further enhance predictive power. Additionally, experimenting with other ensemble learning models like LightGBM or CatBoost could provide useful comparative insights. From a deployment perspective, converting this solution into an interactive web-based dashboard or API service would increase its usability for real-world stakeholders. Lastly, continuous monitoring and periodic retraining with updated datasets will be crucial to maintaining the model's accuracy as market dynamics evolve over time.

References

Chen, T., & Guestrin, C. (2025, July 05). *SCISPACE*. Retrieved from XGBoost: A Scalable Tree Boosting System: https://scispace.com/papers/xgboost-a-scalable-tree-boosting-system-48ocu6x4c7

Chen, Y. (2025, July 6). *Analysis and Forecasting of California Housing*. Retrieved from Dracy & Roy Press: https://drpress.org/ojs/index.php/HBEM/article/view/4704/4557

Rathaur, S., Kamath, N., & Ghanekar, U. (2025, July 6). *Software Defect Density Prediction based on Multiple Linear Regression*. Retrieved from IEEE Xplore: https://ieeexplore.ieee.org/document/9183110

Singh, A. P., Rastogi , K., & Rajpoot, S. (2025, July 06). *IEEE Explore*. Retrieved from House Price Prediction Using Machine Learning: https://ieeexplore.ieee.org/document/9725552