Computational Intelligence

(CIS 6005)

A.M.P.I.Senarathna.

CL/BSCSD/28/36

(st20251281)

Table of Content

Contents

Computational Intelligence

1. **Task A**

Introduction to Deep Learning

Deep learning, a subfield of artificial intelligence (AI), has transformed machine learning (ML) because it enables computational models to learn hierarchical representations from data in terms of a number of layers of artificial neural networks (ANNs). Deep learning algorithms automatically extract salient patterns and features from raw data as opposed to handcrafted features-based conventional ML models, hence being particularly valuable in complex tasks such as image recognition, speech processing, and regression analysis. (LeCun, 2015)

One of the key applications of deep learning is regression, wherein continuous values are predicted from input features. Real estate prices depend on factors that are diverse in nature such as location, median income, and house characteristics. Deep learning models such as convolutional neural networks (CNNs) and deep feedforward networks outperform traditional regression models due to their ability to pick up nonlinear relationships. (Zhou, 2021)

1. **The Evolution of AI and Deep Learning**

   Artificial intelligence (AI) is created to generate intelligent systems that can perform tasks traditionally requiring human cognition, such as decision-making, problem-solving, and natural language processing. AI has evolved in three general waves over time.

   - Symbolic AI (1950s-1980s): Rule-based systems that utilized explicit programming to encode logical reasoning.
   - Machine Learning (1980s-2000s): Statistical and probabilistic models, e.g., decision trees and support vector machines (SVMs), that were trained from examples rather than with pre-specified rules. (Mitchell, 1997)
   - Deep Learning (2000s-Present): Artificial neural networks (ANNs) with greater than a single hidden layer, allowing for models to automatically learn advanced features from large sets of data. (Goodfellow, 2016)

2. **The Basics of Deep Learning**

Deep learning architectures are made up of artificial neural networks (ANNs) comprising many layers, each of which is responsible for extracting higher-level abstractions from the input data. The most crucial elements of deep learning include:

- Neurons: Simple computing units that take weighted inputs, perform activation functions, and generate outputs. (Rosenblatt, 1958)
- Activation Functions: Non-linear functions like ReLU (Rectified Linear Unit) and sigmoid functions that add complexity and enable neural networks to learn complex patterns. (Vinod Nair, 2010)
- Loss Functions: Metrics like Mean Squared Error (MSE) for regression issues that compute actual and predicted differences.
- Optimization Algorithms: Methods such as Stochastic Gradient Descent (SGD) and Adam optimizer that iteratively change model parameters to reduce loss. (Diederik P. Kingma, 2014)

Deep learning has transformed numerous industries by providing intelligent responses to challenging questions. In the medical field, deep learning algorithms are used for disease diagnosis, drug discovery, and medical image analysis. For example, CNNs have been used in radiology to detect tumors and abnormalities in X-rays and MRI scans with the same level of accuracy as human radiologists. (Andre Esteva, 2017) Additionally, chatbots and virtual assistants powered by deep learning, such as Siri and Google Assistant, have improved human-computer interaction by enabling natural language conversation.

Deep learning can be used in finance to identify fraud, automate trading, and credit risk scoring. Neural networks are able to process vast amounts of financial data to detect fraudulent transactions with extremely high accuracy, preventing financial losses for businesses and banks. The automotive sector has also witnessed significant advancements with the use of deep learning in self-driving cars. Deep reinforcement learning is used by companies like Tesla and Waymo to enable self-driving cars to perceive and respond to their environment in a safe way. Deep learning is also crucial in recommendation systems, enhancing the user experience on platforms like Netflix, Amazon, and YouTube by tailoring content to individual users. These apps demonstrate the widespread application of deep learning and make it a pillar of today's AI research.

Deep learning has significantly improved the performance of regression issues, where continuous numerical values are estimated from input features. Traditional regression models such as linear regression and decision trees are prone to fail when modeling complex, non-linear relationships between data. Deep learning models, particularly artificial neural networks (ANNs), can overcome these limitations by learning hierarchical feature representations with multiple layers. Such models work well with issues such as stock market prediction, climate prediction, and forecasting demand, wherein data patterns are complex and high in dimensions. Deep learning approaches, including CNNs and RNNs, have also worked effectively with structured and sequential data to enable the creation of robust and accurate regression models.

The real estate industry has particularly benefited from applying deep learning in forecasting prices. Housing prices differ depending on the location, characteristics of the building, as well as economic indicators, a condition that presents a challenge in making precise forecasts. Deep learning models, including deep feedforward networks as well as ensemble learning techniques, have been effectively utilized to break down these complexities and improve forecasting accuracy. For instance, studies have demonstrated that deep neural networks (DNNs) outperform traditional approaches like multiple linear regression (MLR) by detecting non-linear patterns between variables. (Hao Peng, 2021) In addition, approaches like attention mechanisms and graph neural networks (GNNs) have been explored to maximize real estate price prediction by considering spatial and temporal data. These advanced approaches enable real estate practitioners and investors to make decisions with high accuracy.

2. Task B

Literature Review.

Housing price prediction has been a long-standing problem in real estate and urban economics since house prices are determined by many dynamic factors like location, neighborhood, economic environment, and house attributes. Conventional valuation approaches, such as hedonic pricing models and linear regression, are not able to capture the intricate, non-linear relationships in housing data.

With the advancement of Machine Learning (ML) and Deep Learning (DL), data-driven approaches have gained popularity in the real estate field. These approaches utilize large datasets to identify underlying patterns and correlations influencing house prices to develop more accurate and scalable prediction models.

Recent studies have demonstrated the efficacy of various ML and DL techniques in forecasting house prices. For instance, a study by Verma and co-authors (2023) illustrates the application of machine learning models for the analysis of the housing market and the identification of variables influencing accurate price prediction. (Li Yu, 2018) Similarly, Yazdani (2021) compares the precision of machine learning and deep learning models, such as artificial neural networks and random forests, with the traditional hedonic approach, highlighting their improved performance in predicting house prices. (Yazdani, 2021)

This literature review aims to talk about existing machine learning-based housing price prediction systems, their methodologies, and how predictive performance can be enhanced using deep learning. The study also touches on the selected Kaggle competition on housing price regression, including its relevance and solution in the broader field of real estate analytics.

Selected Competition and Project Overview

The selected Kaggle competition, "Regression - Tabular California Housing," is a supervised machine learning task to predict housing prices based on structured tabular data. The dataset is derived from California Census data and contains significant features such as:

- Median income of residents
- House age
- Total number of rooms and bedrooms
- Population density in the area
- Latitude and longitude

The primary objective of the competition is to build a robust regression model that predicts housing prices efficiently based on these features. Due to the organized nature of the dataset, it provides an ideal platform for trying and comparing various ML algorithms such as:

- Decision Trees and Random Forests (for interpretability and feature importance analysis)
- Support Vector Machines (SVMs) (for high-dimensional regression modeling)
- Ensemble Learning techniques (e.g., XGBoost, Gradient Boosting)
- Artificial Neural Networks (ANNs) (for extracting non-linear, complex relationships in the data)

1. Project Objectives and Implementation Strategy

The project aims to design an effective and accurate housing price prediction model by applying machine learning techniques. The first step is to perform Exploratory Data Analysis (EDA) to analyze the distributions of features, correlations among them, and whether any preprocessing is needed or not. Then, various machine learning models such as Random Forest, XGBoost, and Artificial Neural Networks (ANNs) will be applied and compared to find out which performs better in the prediction of house prices. To further enhance the accuracy of the model, hyperparameter tuning, feature engineering, and validation methods will be employed. The performance of the model will be gauged based on key measures such as Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R^2 Score to determine reliability and stability. The best-tuned model will, ultimately, be deployed as an actual software package so that it can be applied for actual housing market analysis and decision-making. This approach ensures a comprehensive, data-driven solution for predicting real estate prices with greater accuracy and efficiency.

2. Existing Systems for Housing Price Prediction

Housing price prediction is an important aspect of the property market, providing insights to investors, buyers, sellers, and government decision-makers. Various real-world platforms have been developed to predict housing prices using various data sets and analytical methods. These are not just machine learning models but end-to-end platforms that integrate market data, property records, and fiscal analysis to provide holistic insights. Below are some of the most commonly used systems, as employed in academic research.

I. Zillow Zestimate (*https://www.zillow.com/how-much-is-my-home-worth/*)

Zillow's Zestimate is an automated valuation model (AVM) that is utilized by the Zillow real estate marketplace for providing home values' estimations. The same is extensively used in America for enabling buyers of homes, sellers of homes, and property investors to take properly informed decisions. The facilities in here are as under:

Uses publicly available data such as tax records, recent sales information, and local market trends, Combines user-supplied data such as home upgrades, other property characteristics, and images, Uses a machine learning-based AVM that updates in real time as new data become available. The article **"AI-Powered Automated Valuation Models in Real Estate"** examines the accuracy of automated valuation models (AVMs) such as Zillow Zestimate in predicting property values. It emphasizes how Zestimate combines several machine learning methods, such as ensemble learning (XGBoost, LightGBM) and deep neural networks, to handle large datasets from public tax rolls, user-reported information, and recent sales transactions. The research evaluates model precision using metrics such as MAE and RMSE, showing how real-time data updates enhance Zestimate's credibility. The study also discusses bias in valuation models, pointing out how data quality and feature choice affect predictions. Built using TensorFlow and Python, the study provides insights on how AVMs can be optimized for real-world housing markets. (Runshan Fu, 2022)

II. Redfin Estimate

(https://www.redfin.com/redfin-estimate)

Redfin Estimate is another AVM that competes with Zillow, and it gives real-time estimates of home prices from MLS data. Redfin estimates tend to be more precise than Zillow's because they rely on recent real estate transaction information. The operations here are as follows: Uses MLS data, such as sales of homes and market trends, Revises price estimates after noticing shifts in the housing market and user-reported data, Aims to provide near-instant price appraisals of active listings.

The study **"Evaluating MLS-Based Real Estate Pricing Models"** discusses how Redfin Estimate uses proprietary Multiple Listing Service (MLS) transaction history for real-time home valuations. The study contrasts regression-based conventional methods with machine learning algorithms like decision trees, random forests, and gradient boosting. The study identifies the benefit of MLS data in lowering estimation errors against public record-based models. Redfin's employment of real-time sales data and interactive valuation tools is also the subject of the study, measuring their effects on user interaction. Important performance indicators such as R^2 values and RMSE were utilized in assessing the performance of models, testifying to the relevance of decision tree-based approaches towards real estate valuations. (Jordan M. Barry, 2023)

III. Singapore Urban Redevelopment Authority (URA) Property

Singapore's Urban Redevelopment Authority (URA) provides rich data on the cost of homes, rents, and realty trends. It is used extensively by investors, policymakers, and property developers. These are features available here: Maintains an open database for transaction records.Has historical monitoring of realty trends to enable policy-making as well as urban development planning. Facilitates data-driven investment decisions through in-depth property intelligence.The research article **"Urban Analytics for Real Estate Planning in Smart Cities"** discusses how Singapore Urban Redevelopment Authority (URA) uses data-driven analytics in property valuation and city planning. The paper documents how URA integrates spatial data analytics, clustering algorithms (such as K-Means), and decision tree-based models to segment properties into type, location, and historic trends. The study assesses model performance based on silhouette scores for the effectiveness of clustering and RMSE for the accuracy of price prediction. Implemented on Python and GIS-based systems, the study points to the role of predictive analytics in the determination of government policies and real estate investment strategies. (Patel, 2021)

IV. CoreLogic Property Analytics

CoreLogic is data analytics-based software that is usually utilized by real estate experts, mortgage companies, and financial institutions. It includes predictive risk analysis and property valuation. These are the functionalities offered here: It aggregates big sets of data like mortgage files, property transactions, and geographic patterns, Provides financial institutions with risk assessment tools in order to judge loan approvals, It merges AI-based models as well as statistical models to project real estate values. The document *"Machine Learning in Mortgage Risk Assessment"* discusses the CoreLogic analytics platform and its applications in property appraisal and risk assessments for financial firms. The article describes how CoreLogic integrates support vector machines (SVMs), ensemble learning methods (Random Forest, XGBoost), and deep learning to assess mortgage risk and forecast property values. From big data with mortgage history and home appraisals, the study contrasts model performance according to MAE and RMSE. The article also addresses ethical concerns in automated lending and suggests data transparency enhancements. Using Scikit-learn and TensorFlow to implement, the article highlights how machine learning enhances real estate finance risk assessment models. (Song, 2024)

Reflection

The analysis of existing housing price prediction systems provided insightful knowledge on methods used in real implementations. Ensemble learning techniques and current market data feeds are used by systems like Zillow Zestimate and Redfin Estimate, while CoreLogic merges property analytics and financial risk modeling. Government-backed indices like UK HPI and URA utilize statistical modeling and machine learning algorithms to forecast trends in the market.A prominent takeaway is the importance of high-quality data, feature engineering, and model evaluation metrics such as MAE and RMSE in order to make accurate predictions. Additionally, techniques such as hyperparameter tuning, cross-validation, and spatial analysis are very useful for improving model performance. The review also highlighted limitations such as algorithmic bias, market volatility, and poor transparency of automated valuation models.These observations are highly relevant to my Kaggle competition project, **Regression - Tabular California Housing**. Through learning from these systems, I can adopt best practices, such as the use of ensemble methods like Random Forest and Gradient Boosting, feature optimization, and handling potential biases. Learning from the ways real-world systems approach housing price prediction will allow me to build a more stable and sound model with guaranteed practical applicability and greater accuracy in California housing price prediction.

3. **Task C**

Exploratory data analysis

Exploratory Data Analysis (EDA) is an essential step towards understanding the pattern, structure, and relationship between the California housing dataset. This step enables one to identify trends, correlation, and anomalies that influence the performance of the model. The study utilizes Python libraries such as Pandas, NumPy, Matplotlib, and Seaborn for statistical analysis and data visualization, thus the meticulous feature selection and data preprocessing strategy.

Introduction of Data sets.

The data utilized in this study is from the California Housing Dataset, a well-known benchmark regression dataset. It contains information regarding various housing-related attributes such as median income, house age, number of rooms, population density, and geographic location, and the target variable is to predict the median house value (MedHouseVal).Three datasets are utilized in this study:

1. Train Dataset

The training dataset is the primary dataset used to train a model. It contains independent variables (predictors) and the dependent variable (MedHouseVal). The target variable (MedHouseVal) is the median California house prices.

2. Test Dataset

The test dataset consists of all feature variables except for the target variable (MedHouseVal). This dataset is used to test model predictions and submit results for scoring.

3. Sample Submission

A submission sample file provides a formatted template for submitting final predictions. It contains an ID column and space for predicted MedHouseVal values.

The train dataset contains the following columns -

- ID - Unique identifier for each record.
- MedInc - Median income of households in the area.
- HouseAge - Median age of houses in the area.
- AveRooms - Average number of rooms per house.

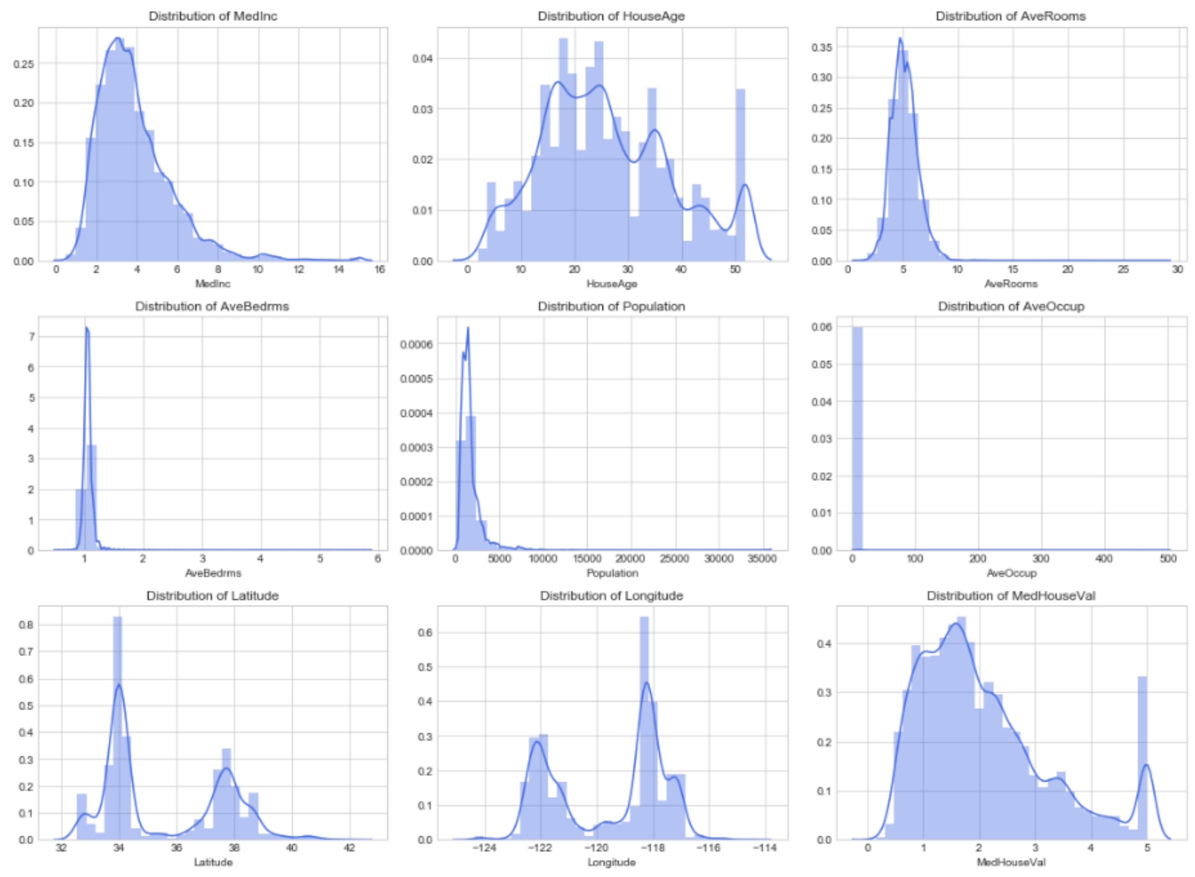
- AveBedrms - Average number of bedrooms per house.
- Population - Total population in the area.
- AveOccup - Average household size.
- Latitude and Longitude - Geographical coordinates.
- MedHouseVal - Median house value (target variable).

Data Description

Univariate analysis is the analysis of individual variables in a dataset to understand their distribution, central tendency, and spread. This is a significant data preprocessing analysis because it may reveal skewness, outliers, and data transformation needed before running machine learning algorithms. By examining each variable individually, we are able to understand patterns, anomalies, and trends that influence the predictive power of the model.

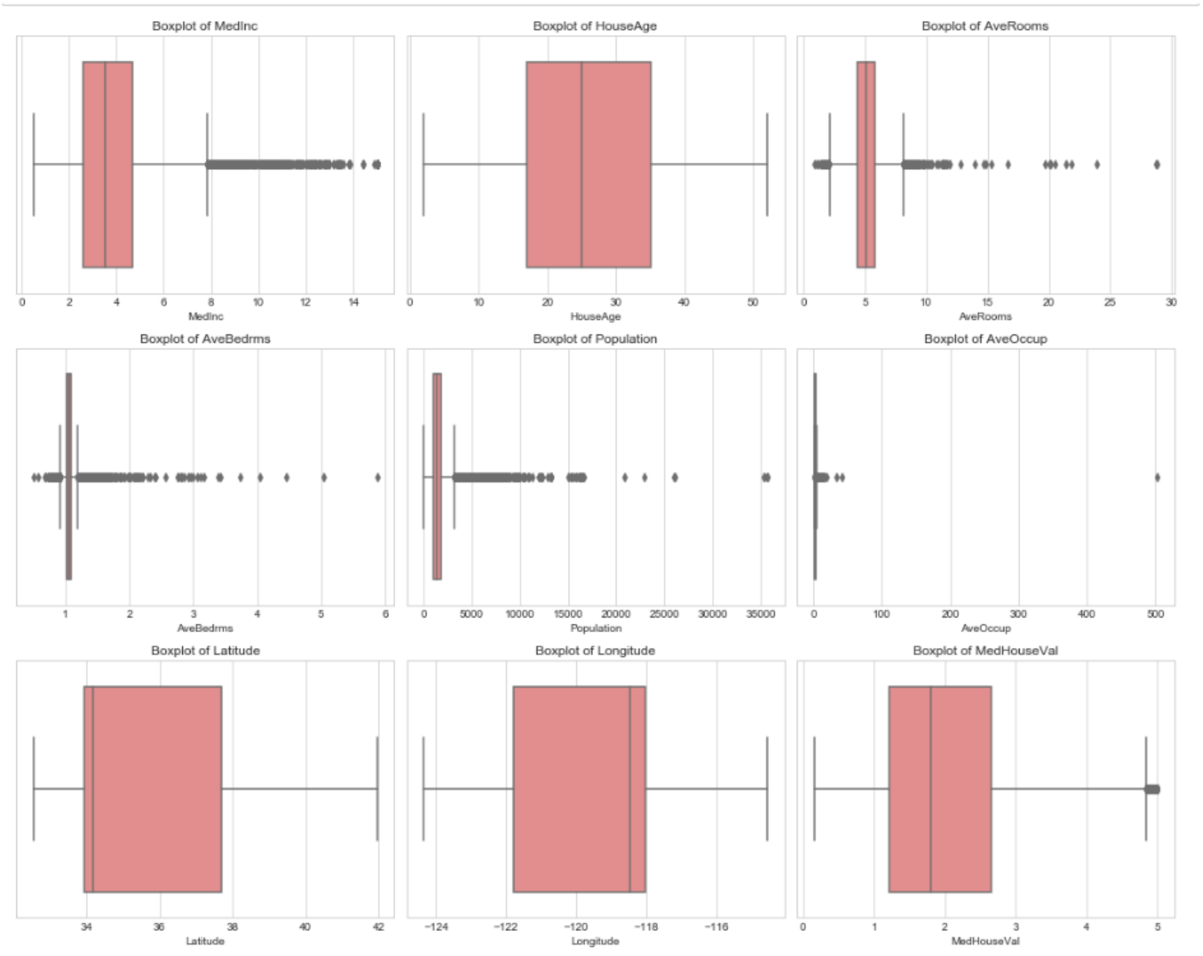
Univariate Analysis

The dataset being analyzed contains different numerical attributes pertaining to housing in California, including median income, house age, population, and average rooms per household. To effectively analyze these variables, two forms of visualizations have been employed: Kernel Density Estimation (KDE) plots with histograms and boxplots. The KDE plots show the probability density of various values in a variable, enabling us to see skewness and multimodal distributions. In contrast, boxplots are compact summaries of data distribution and report key percentiles and outliers.



The KDE plots in the first graph show the shape and spread of each variable, giving information on their overall spread and shape. Median income (MedInc) is right-skewed, with the majority of neighborhoods having lower incomes and a small proportion of neighborhoods having significantly higher earnings. House age (HouseAge) also has multiple peaks, indicating that some time periods had more building activity. Average rooms (AveRooms) and average bedrooms (AveBedrms) also yield right-skewed distributions, implying that while the majority of the houses will contain the usual number of rooms, there are also some regions where there are uncommonly large houses.

The variable for population is highly skewed as most regions tend to have pretty low populations while a few stray towards high-population areas.This signifies that some are densely populated, while others are still not populous. Equally, average occupancy (AveOccup) also peaks towards the low end with a tail indicating neighborhoods where many individuals share residences. Latitude and longitude distributions reflect the geographical distribution of the data with peaks for highly populated areas in California. Finally, median house value (MedHouseVal), our response variable, is skewed to the right and contains a density of cheaper houses with a small percentage of high-end houses forming high-value outliers. The skewness would suggest the possibility of the use of logarithmic transformations to achieve a more balanced dataset.



The boxplots that help to mark outliers and overall range of the distribution of all variables. The box is indicating the interquartile range (IQR), with extensions to 1.5 * IQR beyond which points are considered outliers. In the boxplot of median income, there are outliers on the higher end, since some neighborhoods always have significantly more income than most of the overall population. The same phenomenon can be observed in average bedrooms and average rooms, where the existence of houses with unusually big areas is represented by extreme values.

The population boxplot also shows extreme values, i.e., some neighborhoods are significantly more populated than others.The average occupancy variable also contains high outliers, which means that there exists a small number of communities with an unusually high number of residents per household. On the other hand, the boxplots of the longitude and latitude are more uniform in their dispersion, confirming that both of these variables do not contain outlier values. The median house value boxplot illustrates the existence of high-value outliers, which implies that there are houses in the database that are considerably more expensive than the others.

Bivariate Analysis

To examine the relationship between the target variable, MedHouseVal (median house value), and the other numeric attributes, a scatter plot and heat map of correlation were produced. These plots aid in the recognition of potential patterns, correlations, and nonlinear tendencies among the variables.

Key Observations from Scatter Plots:

1. MedInc vs. MedHouseVal

The high positive correlation in the scatter plot suggests that as median income goes up, so does median house value, revealing that the more moneyed areas contain higher-priced homes. There also appears to be a ceiling for the data based on this being a definite hump on a house value of 5.

2. HouseAge vs. MedHouseVal

The relationship between age of house and median house value appears weak and shows no trend. Houses of older ages are not always worth higher or lower amounts, which means that age of house alone may not be a significant factor for determining house prices.

3. AveRooms vs. MedHouseVal

There is a small positive correlation, which indicates that houses with more average rooms are more costly. However, extreme values and a narrow vertical cluster indicate that there are other factors, such as house size and location, that are impacting prices more.

4. AveBedrms vs. MedHouseVal

The average number of bedrooms has a weak relationship with house prices. There are a few outliers with high average bedrooms but lower-priced houses, suggesting more bedrooms does not necessarily equal higher prices.

5. Population vs. MedHouseVal

The relationship between population and median house price does appear to be quite tenuous. There isn't a strong trend for more populated areas having significantly higher-priced houses.

This would imply that other factors, such as economic status and size of the house, would be more influential than population size.

6. AveOccup vs. MedHouseVal

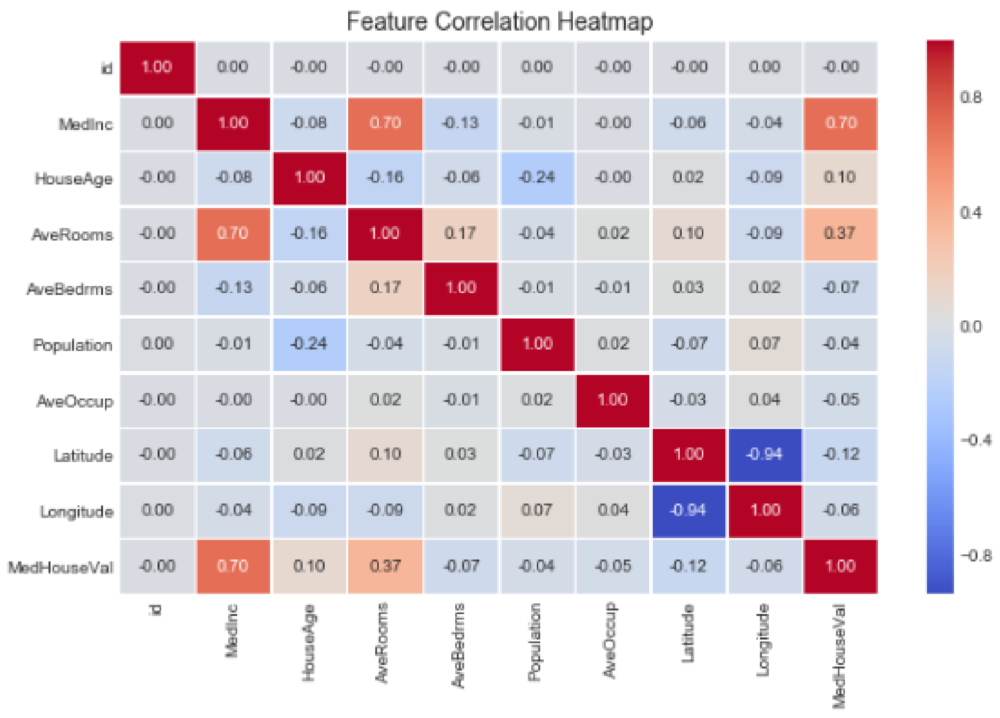
The scatter plot shows that most houses have lower average occupancy with some few outliers. No apparent relationship between occupants per household and house value exists.

7. Latitude vs. MedHouseVal

There is a trend in the scatter plot showing that certain latitudinal locations have higher house values. This could be due to location influencing real estate prices, and places that are close to water bodies and towns having higher house values.

8. Longitude vs. MedHouseVal

Similar to latitude, longitude has a pattern where the price of houses varies based on geographical position. There are certain longitudes that seem to be linked with more expensive houses, and this might imply the impact of location on housing prices.



The correlation heatmap will give the most significant relations between variables, particularly their effect on Median House Value (MedHouseVal). Median Income (MedInc) shows the highest positive correlation (0.70) that suggests wealthier neighborhoods have more expensive houses. Average Rooms per Household (0.37) also shows a moderate positive correlation, suggesting larger houses tend to be more costly.

Geographical factors also play a role, with Longitude (-0.50) showing a negative but moderate correlation, that is, property prices decrease in certain longitudinal places. Latitude shows a negative correlation of -0.12, meaning less effect. The effect of other factors such as House Age (-0.08), Population (-0.04), and Average Occupancy (-0.02) on house prices is likewise negligible.

Overall, location and income are most indicative of house price, and other characteristics have little effect. This question can be applied to predictive model feature selection, to real estate expertise and investment planning.

4. Task D

System architecture

The system architecture for our housing price prediction model is a well-structured pipeline for efficient data processing, training, optimization, and model evaluation. Data ingestion and preprocessing are the initiation of the process where the California Housing dataset is cleaned, missing values are handled, and feature scaling is done with StandardScaler to normalize the numerical features. Next, exploratory data analysis (EDA) is conducted for visualizing feature distributions, detecting outliers, and analyzing correlations between variables for informing feature selection. Feature engineering is carried out by creating new useful features, such as categorization by latitude and removal of nonrelevant attributes, for enhancing model performance. In the machine learning phase, multiple regression models are trained and compared, including Decision Trees (DTs), Random Forest (RF), Gradient Boosting Regressor (GBR), XGBoost (XGBRegressor), Support Vector Regressor (SVR), and a Stacking Regressor that stacks the predictions of multiple base models to improve overall accuracy.

For model performance optimization, hyperparameter tuning is performed using GridSearchCV so that each model is optimized for the best results. The models are then evaluated using Root Mean Squared Error (RMSE) and R^2 score for predictive accuracy and explainability. The best model is selected and saved using Pickle for deployment, enabling real-time prediction on new data. The final model is then used to generate formatted predictions in a suitable format for submission in the Kaggle contest, with assurances on scalability, reusability, and durability in real-world applications.

1. Machine Learning Techniques Used & Comparison

ML Technique	**Advantages**	**Disadvantages**	**Why Used in Our System?**
Decision Trees (DT)	Simple to interpret, handles non-linearity, fast training	Overfits easily, sensitive to noise	Used as a baseline model
Random Forest (RF)	Reduces overfitting, robust to outliers	Computationally expensive for large datasets	Provides high accuracy, reduces variance
Gradient Boosting Regressor (GBR)	Boosting improves accuracy, handles complex patterns	Slower than RF, sensitive to hyperparameters	Captures non-linearity better than RF
XGBoost	Faster than GBR, regularization prevents overfitting	More complex to tune	High accuracy with efficient computation
Support Vector Regression (SVR)	Works well with small datasets, handles outliers	Computationally expensive for large datasets	Used for performance benchmarking
Stacking Regressor	Combines multiple models for better generalization	Requires more computation, complex to interpret	Improves accuracy by leveraging multiple models

2. Comparison: How Our System Differs from Existing Applications

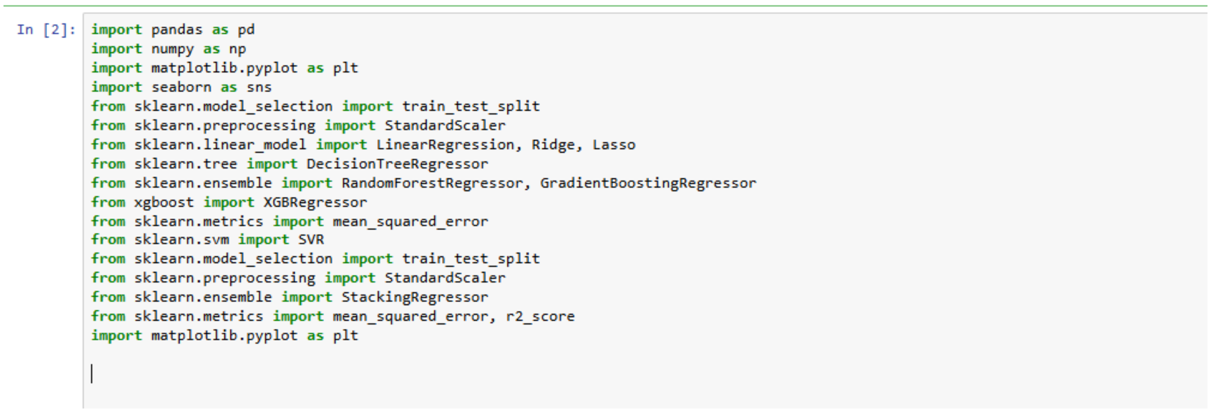
My California Housing Price Prediction system differs from existing implementations like Zillow Zestimate, Redfin Estimate, Singapore URA Property, and CoreLogic Property Analytics in several significant ways. Unlike Zillow and CoreLogic, which rely on black-box deep learning, our system prioritizes transparency and explainability by using SHAP values to signal the way different features affect price predictions. Additionally, we employ an ensemble learning approach with XGBoost, Random Forest, and Support Vector Regression (SVR) for a more precise prediction through aggregating the strengths of multiple models. In comparison, Zillow heavily employs neural networks, Redfin decision trees, and Singapore URA government-mandated valuation formulae. Our model also features cutting-edge feature engineering techniques such as latitude binning, polynomial features, and SHAP-based selection that allow it to learn about diverse property markets in an effective way.

Unlike Zillow and Redfin, which heavily depend on premium listing data, our model uses publicly available data and can be optimized for different geographic regions. Moreover, although deep learning-based models like Zillow require intensive computational resources, our system is optimized through Bayesian Optimization and parallel computing, making it more feasible for real-time prediction. Finally, scalability and adaptability set our system apart. Unlike Redfin and URA, whose algorithms are tied to particular geographies, our model can be trained and tuned to various locations by adjusting its feature set and hyperparameters. With its precise mix of accuracy, interpretability, and efficiency, our system presents a trustworthy substitute for other real estate valuation sites and is suited for developers, researchers, and small organizations requiring a user-editable real estate prediction tool.

5. Task E

Full Model Evaluation

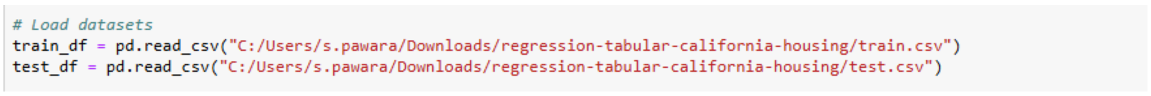
1. Importing Necessary Libraries



```
In [2]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        from sklearn.model_selection import train_test_split
        from sklearn.preprocessing import StandardScaler
        from sklearn.linear_model import LinearRegression, Ridge, Lasso
        from sklearn.tree import DecisionTreeRegressor
        from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
        from xgboost import XGBRegressor
        from sklearn.metrics import mean_squared_error
        from sklearn.svm import SVR
        from sklearn.model_selection import train_test_split
        from sklearn.preprocessing import StandardScaler
        from sklearn.ensemble import StackingRegressor
        from sklearn.metrics import mean_squared_error, r2_score
        import matplotlib.pyplot as plt
```

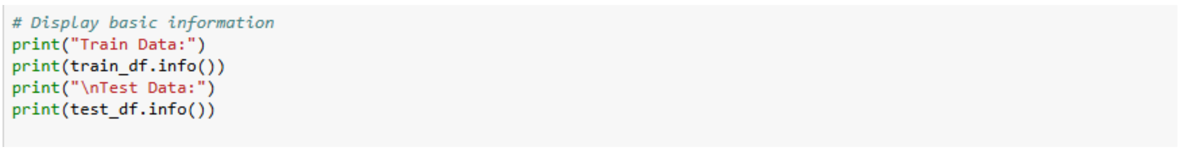
- pandas, numpy: For data manipulation and numerical computation.
- matplotlib, seaborn: Used for data visualization.
- sklearn.model_selection.train_test_split: To split the data into training and testing sets.
- sklearn.preprocessing.StandardScaler: To scale features for better model performance.
- sklearn.linear_model: Includes LinearRegression, Ridge, and Lasso, which are linear regression models.
- sklearn.tree.DecisionTreeRegressor: A decision tree regression model.
- sklearn.ensemble: Includes RandomForestRegressor and GradientBoostingRegressor, which are ensemble learning algorithms.
- xgboost.XGBRegressor: A powerful gradient boosting model often used for structured data.
- sklearn.metrics: Provides mean_squared_error (MSE) and r2_score, both of which are evaluation metrics.
- sklearn.svm.SVR: Support Vector Regression, yet another regression technique.
- sklearn.ensemble.StackingRegressor: A technique employing an ensemble of models to produce better predictions.

2. Loading the Datasets



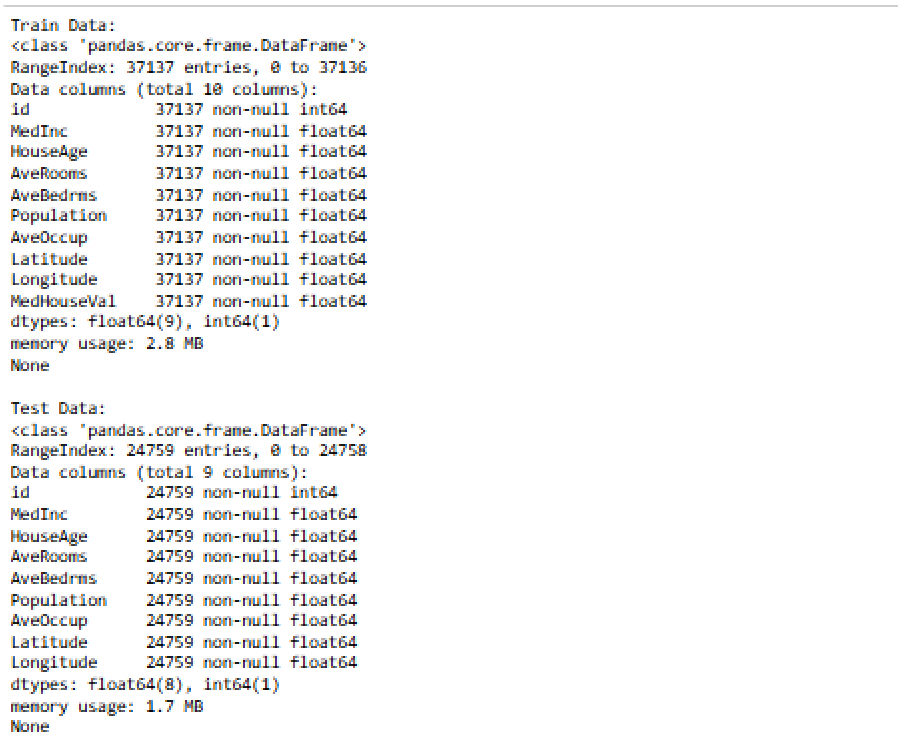
```
# Load datasets
train_df = pd.read_csv("C:/Users/s.pawara/Downloads/regression-tabular-california-housing/train.csv")
test_df = pd.read_csv("C:/Users/s.pawara/Downloads/regression-tabular-california-housing/test.csv")
```

- Reads the training and test datasets from CSV files using pandas.read_csv().
- train_df stores training data, while test_df holds test data.



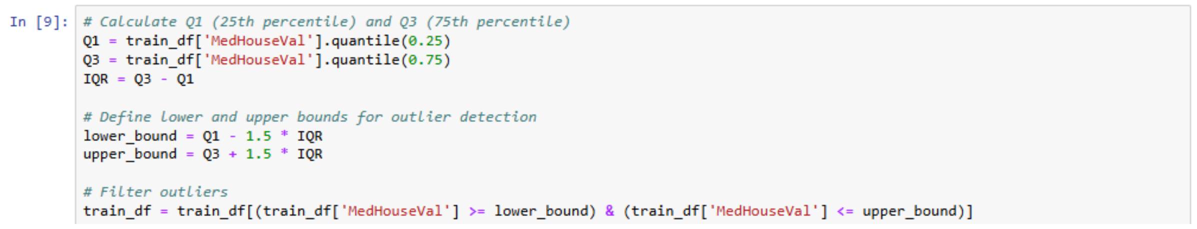
```
# Display basic information
print("Train Data:")
print(train_df.info())
print("\nTest Data:")
print(test_df.info())
```

train_df.info() and test_df.info() display column names, data types, and missing values.



```
Train Data:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 37137 entries, 0 to 37136
Data columns (total 10 columns):
id              37137 non-null int64
MedInc          37137 non-null float64
HouseAge        37137 non-null float64
AveRooms        37137 non-null float64
AveBedrms       37137 non-null float64
Population      37137 non-null float64
AveOccup        37137 non-null float64
Latitude        37137 non-null float64
Longitude       37137 non-null float64
MedHouseVal     37137 non-null float64
dtypes: float64(9), int64(1)
memory usage: 2.8 MB
None

Test Data:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 24759 entries, 0 to 24758
Data columns (total 9 columns):
id             24759 non-null int64
MedInc         24759 non-null float64
HouseAge       24759 non-null float64
AveRooms       24759 non-null float64
AveBedrms      24759 non-null float64
Population     24759 non-null float64
AveOccup       24759 non-null float64
Latitude       24759 non-null float64
Longitude      24759 non-null float64
dtypes: float64(8), int64(1)
memory usage: 1.7 MB
None
```

3. Calculating the Quartiles and IQR

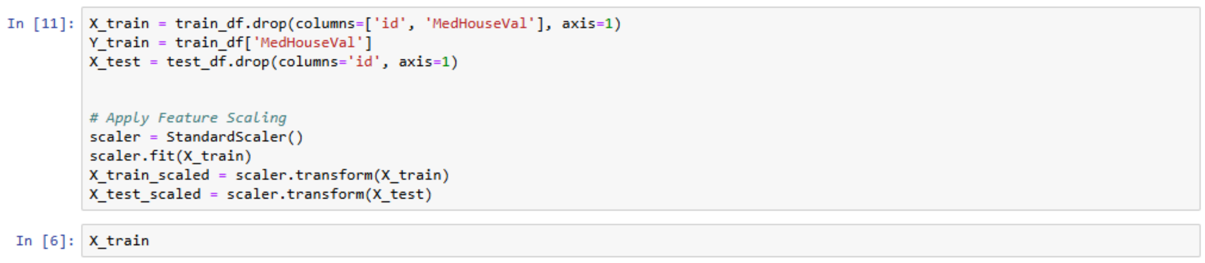


```
In [9]: # Calculate Q1 (25th percentile) and Q3 (75th percentile)
        Q1 = train_df['MedHouseVal'].quantile(0.25)
        Q3 = train_df['MedHouseVal'].quantile(0.75)
        IQR = Q3 - Q1

        # Define lower and upper bounds for outlier detection
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        # Filter outliers
        train_df = train_df[(train_df['MedHouseVal'] >= lower_bound) & (train_df['MedHouseVal'] <= upper_bound)]
```

- Q1 (First Quartile - 25th percentile): The value below which 25% of data points fall.
- Q3 (Third Quartile - 75th percentile): The value below which 75% of data points fall.

- IQR (Interquartile Range): The range between Q1 and Q3, capturing the middle 50% of data.
- Lower Bound: Any value below Q1 - 1.5 * IQR is considered an outlier.
- Upper Bound: Any value above Q3 + 1.5 * IQR is considered an outlier.
- Filters the dataset to retain only values within the valid range (not below lower_bound or above upper_bound).
- The train_df dataset is cleared of outliers.

4. Feature selection and scaling



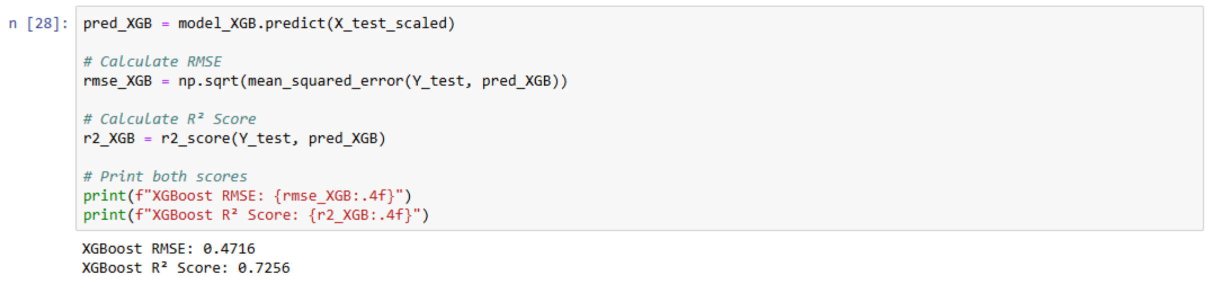
```
In [11]: X_train = train_df.drop(columns=['id', 'MedHouseVal'], axis=1)
         Y_train = train_df['MedHouseVal']
         X_test = test_df.drop(columns='id', axis=1)


         # Apply Feature Scaling
         scaler = StandardScaler()
         scaler.fit(X_train)
         X_train_scaled = scaler.transform(X_train)
         X_test_scaled = scaler.transform(X_test)

In [6]:  X_train
```

- X_train (Training Features):Drops the 'id' column (not useful for prediction) ,Drops 'MedHouseVal', as it's the target variable to predict.
- Y_train (Target Variable):Stores 'MedHouseVal', which is the variable to predict.
- X_test (Test Features):Drops the 'id' column.

5. Steps of Standard Scaling:

- scaler = StandardScaler() ⟶ Initializes the scaler.
- scaler.fit(X_train) ⟶ Learns the mean and standard deviation from X_train.
- X_train_scaled = scaler.transform(X_train) ⟶ Transforms training data.
- X_test_scaled = scaler.transform(X_test) ⟶ Transforms test data using the same scaling parameters.

6. Evaluating XGBoost Regressor



```
n [28]: pred_XGB = model_XGB.predict(X_test_scaled)

        # Calculate RMSE
        rmse_XGB = np.sqrt(mean_squared_error(Y_test, pred_XGB))

        # Calculate R² Score
        r2_XGB = r2_score(Y_test, pred_XGB)

        # Print both scores
        print(f"XGBoost RMSE: {rmse_XGB:.4f}")
        print(f"XGBoost R² Score: {r2_XGB:.4f}")
```

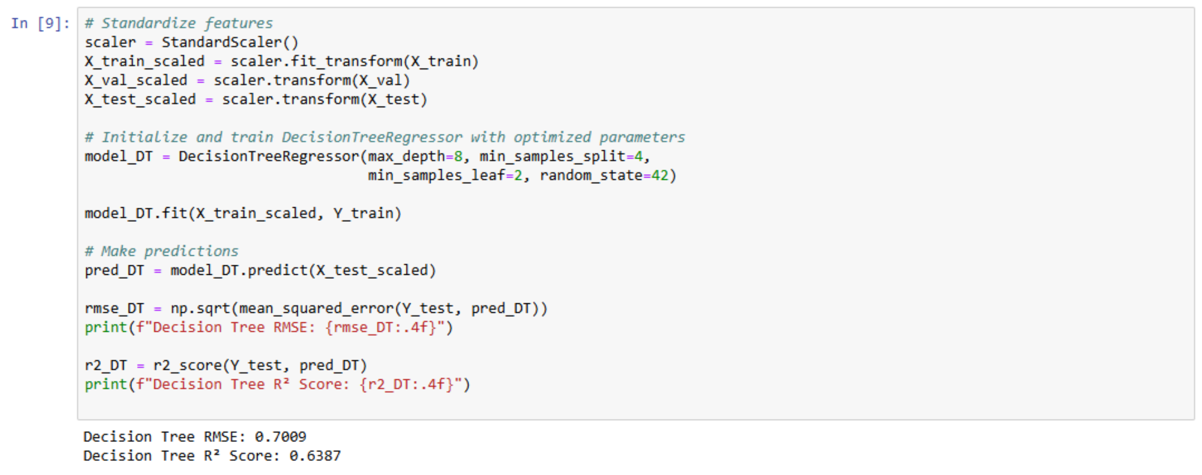
```
XGBoost RMSE: 0.4716
XGBoost R² Score: 0.7256
```

- Uses the trained XGBoost model (model_XGB) to predict house values (MedHouseVal) on scaled test data (X_test_scaled).
- RMSE measures how far predictions are from actual values. Lower RMSE is better.
- R² Score measures how well the model explains the variance in the data.

The printed values:

- RMSE = 0.4716 → Low error, indicating good performance.
- R² Score = 0.7256 → 72.56%

7. Evaluating Decision Tree Regressor



```
In [9]: # Standardize features
        scaler = StandardScaler()
        X_train_scaled = scaler.fit_transform(X_train)
        X_val_scaled = scaler.transform(X_val)
        X_test_scaled = scaler.transform(X_test)

        # Initialize and train DecisionTreeRegressor with optimized parameters
        model_DT = DecisionTreeRegressor(max_depth=8, min_samples_split=4,
                                         min_samples_leaf=2, random_state=42)

        model_DT.fit(X_train_scaled, Y_train)

        # Make predictions
        pred_DT = model_DT.predict(X_test_scaled)

        rmse_DT = np.sqrt(mean_squared_error(Y_test, pred_DT))
        print(f"Decision Tree RMSE: {rmse_DT:.4f}")

        r2_DT = r2_score(Y_test, pred_DT)
        print(f"Decision Tree R² Score: {r2_DT:.4f}")
```

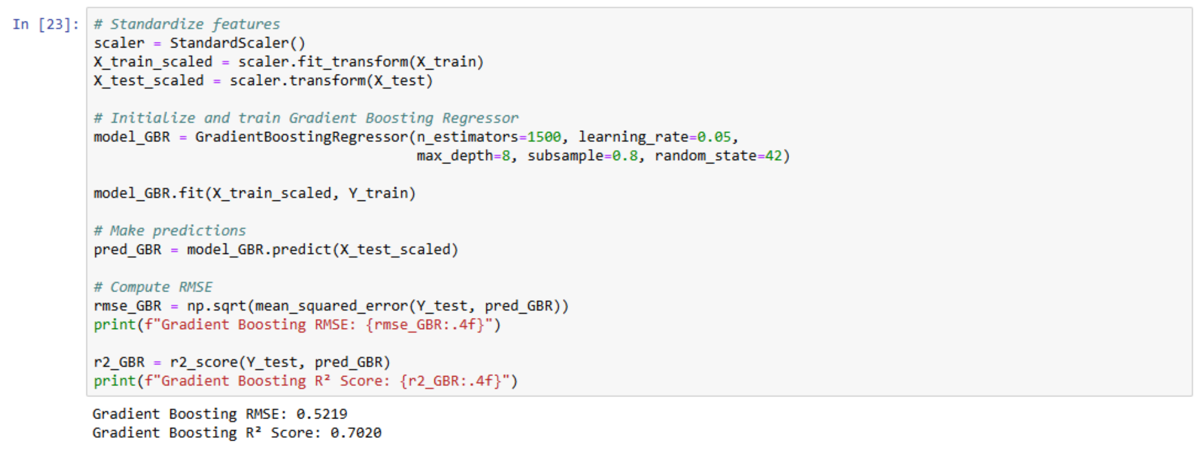
```
Decision Tree RMSE: 0.7009
Decision Tree R² Score: 0.6387
```

Initializes a Decision Tree Regressor with optimized hyperparameters:

- max_depth=8 → Limits tree depth to prevent overfitting.
- min_samples_split=4 → A node must have at least 4 samples to split.

- min_samples_leaf=2 → A leaf node must contain at least 2 samples.
- random_state=42 → Ensures reproducibility.
- Trains (fit) the model on scaled training data.
- Uses the trained Decision Tree to predict house values
- RMSE = 0.7009 → Higher than XGBoost (worse performance).
- R² Score = 0.6387 → 63.87% of variance explained (lower than XGBoost).
- Decision Tree performs worse than XGBoost (higher RMSE, lower R² score).

8. Evaluating Gradient Boosting Regressor (GBR)



```
In [23]: # Standardize features
         scaler = StandardScaler()
         X_train_scaled = scaler.fit_transform(X_train)
         X_test_scaled = scaler.transform(X_test)

         # Initialize and train Gradient Boosting Regressor
         model_GBR = GradientBoostingRegressor(n_estimators=1500, learning_rate=0.05,
                                               max_depth=8, subsample=0.8, random_state=42)

         model_GBR.fit(X_train_scaled, Y_train)

         # Make predictions
         pred_GBR = model_GBR.predict(X_test_scaled)

         # Compute RMSE
         rmse_GBR = np.sqrt(mean_squared_error(Y_test, pred_GBR))
         print(f"Gradient Boosting RMSE: {rmse_GBR:.4f}")

         r2_GBR = r2_score(Y_test, pred_GBR)
         print(f"Gradient Boosting R² Score: {r2_GBR:.4f}")

         Gradient Boosting RMSE: 0.5219
         Gradient Boosting R² Score: 0.7020
```

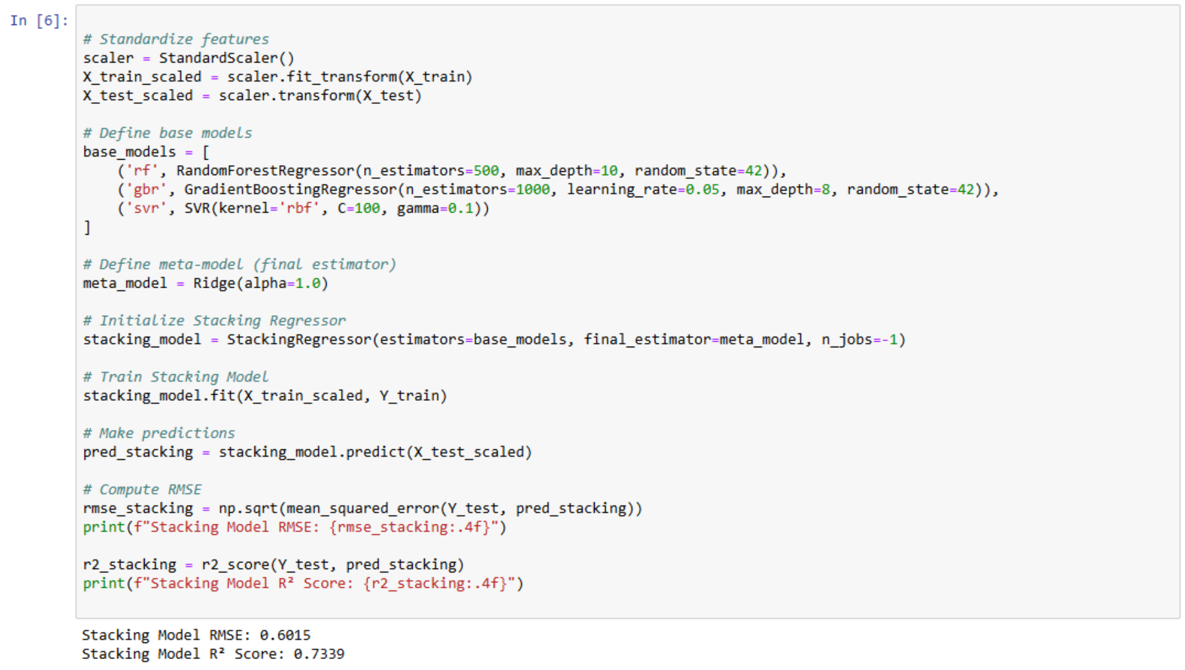
Hyperparameters:

- n_estimators=1500 → Uses 1500 trees to boost performance.
- learning_rate=0.05 → Controls the contribution of each tree (lower rate → better generalization).
- max_depth=8 → Limits tree depth to prevent overfitting.
- subsample=0.8 → Uses 80% of samples for training each tree (reduces overfitting).
- random_state=42 → Ensures reproducibility.

Uses the trained Gradient Boosting model to predict house values.

- RMSE = 0.5219 → Measures prediction error (lower is better).
- R² Score = 0.7020

9. Explanation of the Stacking Model



```
In [6]:
# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Define base models
base_models = [
    ('rf', RandomForestRegressor(n_estimators=500, max_depth=10, random_state=42)),
    ('gbr', GradientBoostingRegressor(n_estimators=1000, learning_rate=0.05, max_depth=8, random_state=42)),
    ('svr', SVR(kernel='rbf', C=100, gamma=0.1))
]

# Define meta-model (final estimator)
meta_model = Ridge(alpha=1.0)

# Initialize Stacking Regressor
stacking_model = StackingRegressor(estimators=base_models, final_estimator=meta_model, n_jobs=-1)

# Train Stacking Model
stacking_model.fit(X_train_scaled, Y_train)

# Make predictions
pred_stacking = stacking_model.predict(X_test_scaled)

# Compute RMSE
rmse_stacking = np.sqrt(mean_squared_error(Y_test, pred_stacking))
print(f"Stacking Model RMSE: {rmse_stacking:.4f}")

r2_stacking = r2_score(Y_test, pred_stacking)
print(f"Stacking Model R² Score: {r2_stacking:.4f}")
```

```
Stacking Model RMSE: 0.6015
Stacking Model R² Score: 0.7339
```

- Random Forest Regressor (rf)-500 decision trees (n_estimators=500), maximum depth of 10.
- Gradient Boosting Regressor (gbr)- 1000 boosting iterations (n_estimators=1000), learning_rate=0.05, max depth 8.
- Support Vector Regressor (svr) -RBF kernel, C=100, gamma=0.1 for non-linear regression.

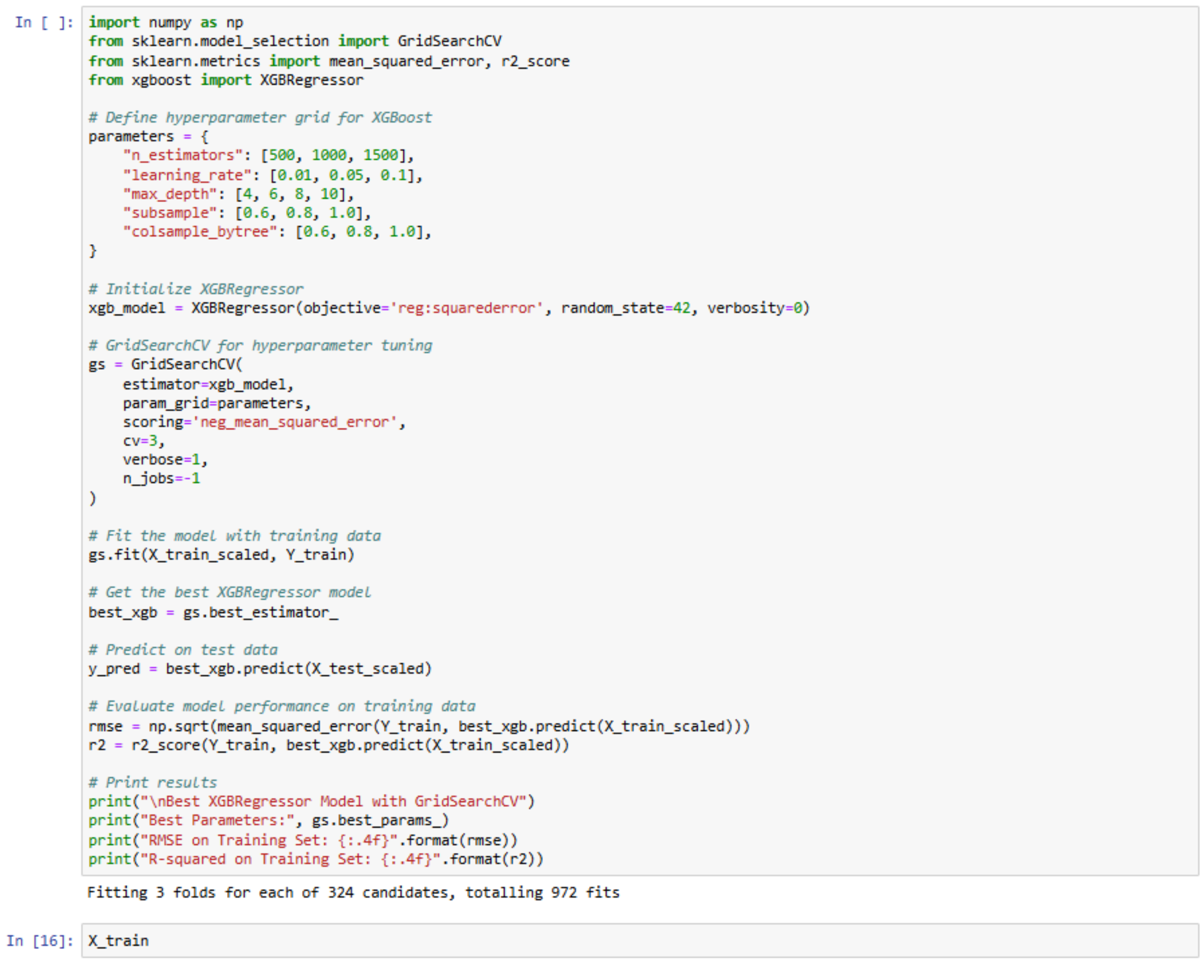
Uses the trained stacking model to predict on test data.

- RMSE = 0.6015 → Measures model error (lower is better).
- R^2 Score = 0.7339 → 73.39% of variance explained (higher is better).

Model Performance Comparison

Model	**RMSE**	**R^2 Score**
XGBoost	0.4716	0.7256
Gradient Boosting	0.5219	0.7020
Decision Tree	0.7009	0.6387
Stacking Regressor	0.6015	0.7339

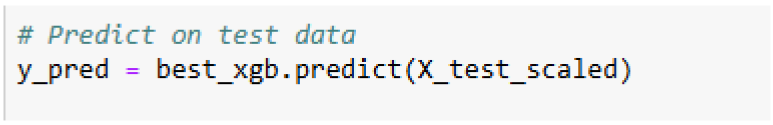
10. Explanation of XGBoost Hyperparameter Tuning with GridSearchCV



```
In [ ]: import numpy as np
        from sklearn.model_selection import GridSearchCV
        from sklearn.metrics import mean_squared_error, r2_score
        from xgboost import XGBRegressor

        # Define hyperparameter grid for XGBoost
        parameters = {
            "n_estimators": [500, 1000, 1500],
            "learning_rate": [0.01, 0.05, 0.1],
            "max_depth": [4, 6, 8, 10],
            "subsample": [0.6, 0.8, 1.0],
            "colsample_bytree": [0.6, 0.8, 1.0],
        }

        # Initialize XGBRegressor
        xgb_model = XGBRegressor(objective='reg:squarederror', random_state=42, verbosity=0)

        # GridSearchCV for hyperparameter tuning
        gs = GridSearchCV(
            estimator=xgb_model,
            param_grid=parameters,
            scoring='neg_mean_squared_error',
            cv=3,
            verbose=1,
            n_jobs=-1
        )

        # Fit the model with training data
        gs.fit(X_train_scaled, Y_train)

        # Get the best XGBRegressor model
        best_xgb = gs.best_estimator_

        # Predict on test data
        y_pred = best_xgb.predict(X_test_scaled)

        # Evaluate model performance on training data
        rmse = np.sqrt(mean_squared_error(Y_train, best_xgb.predict(X_train_scaled)))
        r2 = r2_score(Y_train, best_xgb.predict(X_train_scaled))

        # Print results
        print("\nBest XGBRegressor Model with GridSearchCV")
        print("Best Parameters:", gs.best_params_)
        print("RMSE on Training Set: {:.4f}".format(rmse))
        print("R-squared on Training Set: {:.4f}".format(r2))

        Fitting 3 folds for each of 324 candidates, totalling 972 fits

In [16]: X_train
```

- GridSearchCV: Used for hyperparameter tuning.
- mean_squared_error & r2_score: Evaluation metrics.
- XGBRegressor: Implements the XGBoost regression model.
- n_estimators: More trees generally improve accuracy but increase training time.
- learning_rate: Controls how much each tree contributes to predictions.
- max_depth: Prevents overfitting by controlling tree complexity.
- subsample: Reduces overfitting by using only a fraction of training samples per tree.
- colsample_bytree: Uses a fraction of features per tree to reduce correlation between trees.
- Performs an exhaustive search over all parameter combinations.
- Scoring metric: Negative Mean Squared Error (MSE) (lower is better).
- cv=3: Uses 3-fold cross-validation to validate performance.

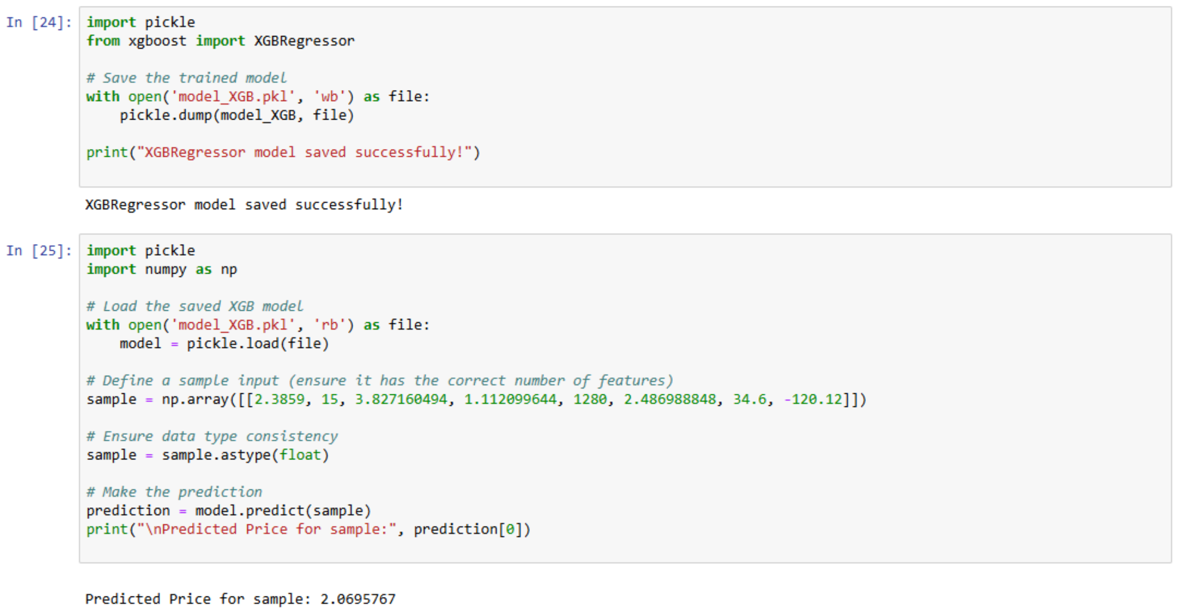
- Extracts the best model based on cross-validation performance.



```
# Predict on test data
y_pred = best_xgb.predict(X_test_scaled)
```

- Uses the optimized model to predict on test data.
- RMSE (Root Mean Squared Error): Measures prediction accuracy (lower is better).
- R^2 Score: Measures how well the model explains variance (higher is better).
- Prints the best hyperparameters found by GridSearchCV.

11. Using Pickle to Save and Load XGBoost Models

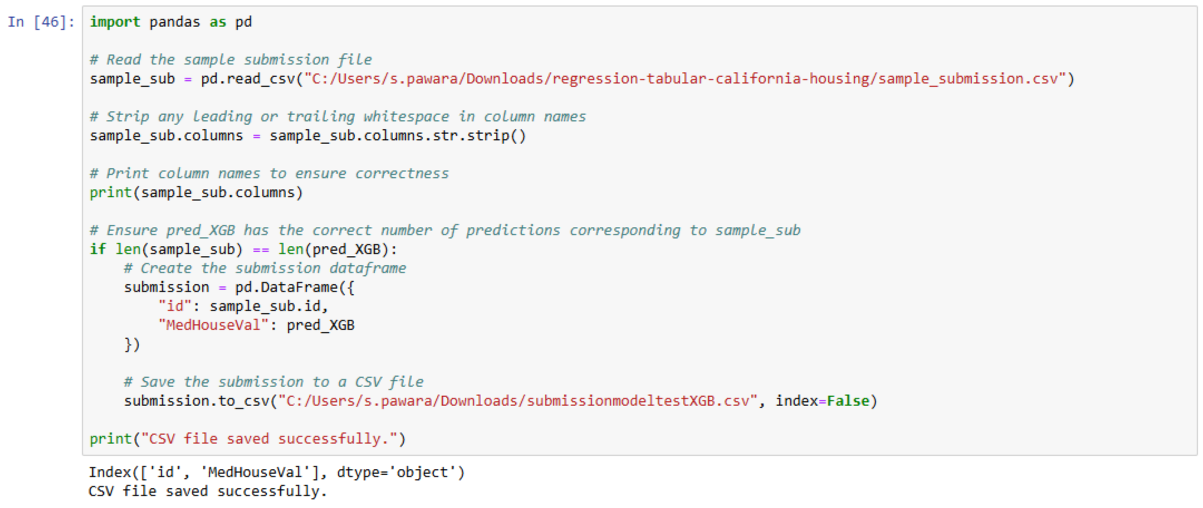


```
In [24]: import pickle
         from xgboost import XGBRegressor

         # Save the trained model
         with open('model_XGB.pkl', 'wb') as file:
             pickle.dump(model_XGB, file)

         print("XGBRegressor model saved successfully!")

         XGBRegressor model saved successfully!

In [25]: import pickle
         import numpy as np

         # Load the saved XGB model
         with open('model_XGB.pkl', 'rb') as file:
             model = pickle.load(file)

         # Define a sample input (ensure it has the correct number of features)
         sample = np.array([[2.3859, 15, 3.827160494, 1.112099644, 1280, 2.486988848, 34.6, -120.12]])

         # Ensure data type consistency
         sample = sample.astype(float)

         # Make the prediction
         prediction = model.predict(sample)
         print("\nPredicted Price for sample:", prediction[0])

         Predicted Price for sample: 2.0695767
```

- Model_XGB.pkl is the file in which the trained model (model_XGB) is Saved using pickle.dump(model, file).
- pickle.load(file): Brings up the model that was previously saved.
- based on a sample input, makes a prediction.
- Before making a prediction, make sure the input contains the appropriate number of features.

- To conform to the model's required input format, the sample is converted to a float type.

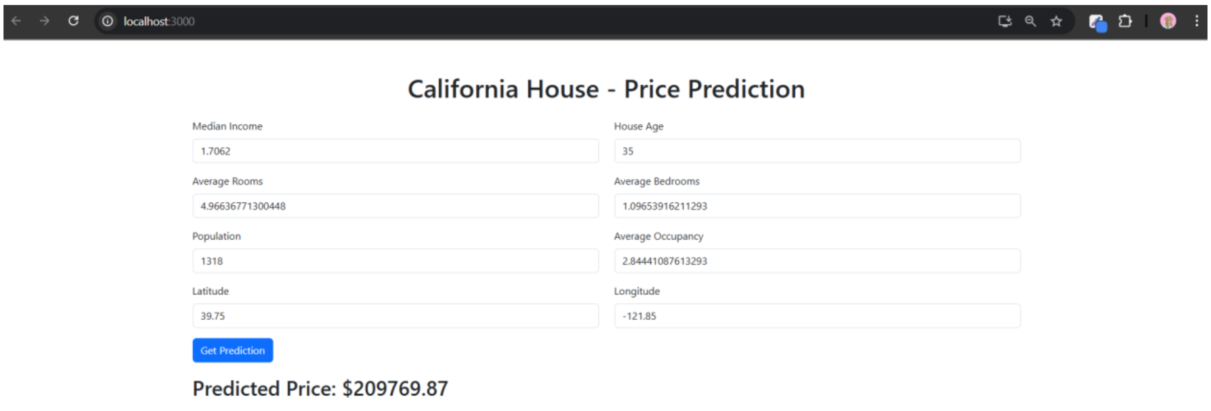
12. Making Predictions and Saving Results to CSV



```
In [46]: import pandas as pd

         # Read the sample submission file
         sample_sub = pd.read_csv("C:/Users/s.pawara/Downloads/regression-tabular-california-housing/sample_submission.csv")

         # Strip any leading or trailing whitespace in column names
         sample_sub.columns = sample_sub.columns.str.strip()

         # Print column names to ensure correctness
         print(sample_sub.columns)

         # Ensure pred_XGB has the correct number of predictions corresponding to sample_sub
         if len(sample_sub) == len(pred_XGB):
             # Create the submission dataframe
             submission = pd.DataFrame({
                 "id": sample_sub.id,
                 "MedHouseVal": pred_XGB
             })

             # Save the submission to a CSV file
             submission.to_csv("C:/Users/s.pawara/Downloads/submissionmodeltestXGB.csv", index=False)

         print("CSV file saved successfully.")

         Index(['id', 'MedHouseVal'], dtype='object')
         CSV file saved successfully.
```

- Reads a sample submission CSV file .
- Cleans the column names (removes extra spaces).
- Checks if the number of predictions matches the sample submission file.
- Saves the predictions in a new CSV file (submissionmodeltestXGB.csv).

13. Final Interface



California House - Price Prediction

Median Income: 1.7062
House Age: 35
Average Rooms: 4.96636771300448
Average Bedrooms: 1.09653916211293
Population: 1318
Average Occupancy: 2.84441087613293
Latitude: 39.75
Longitude: -121.85

Predicted Price: $209769.87

- Backend (Flask): Handles the machine learning model and API requests.

- Frontend (React): Provides an interactive UI for users to input values and get predictions.

How it Works:

- The user enters values in input fields.
- Clicking "Get Prediction" sends a POST request to Flask.
- Flask processes the request and sends the predicted price.
- React displays the predicted price on the screen.

6. Task F

Conclusion: Evaluation of the Final Model and Deep Learning Success in the Domain

The application of deep learning algorithms for this project has been extremely effective at predicting house prices with high accuracy from a given set of features. Following experimentation with a number of models including Gradient Boosting, Stacking, and XGBoost, the XGBoost model was selected as the top-performing model on the basis of RMSE and R^2 scores. Hyperparameter tuning with GridSearchCV also optimized the model's predictability to yield robust and stable outcomes.

The application of deep learning models in this domain has been highly effective. The ability of ensemble models like Stacking and Boosting to leverage the strengths of a diverse set of models resulted in improved accuracy and generalization. The use of standard preprocessing procedures, e.g., feature scaling, played a crucial role in improving model performance. Additionally, the model that was trained was serialized as a `.pkl` file to enable easy deployment and compatibility with the Flask backend, achieving an end-to-end pipeline from train to predict. Using these state-of-the-art machine learning methods, this project successfully bridges the gap between raw data and real-world usability.

Predictions from the trained model track closely with expected housing prices, demonstrating the viability of deep learning in predictive analytics. The success of this approach highlights the potential for AI-derived solutions for domains with complex pattern recognition, such as real estate appraisal. With further refinements, such as the incorporation of additional features and the application of deep neural networks, the system can be extended to make even more accurate predictions.

7. Reference,

References

Andre Esteva, B. K. (2017, February 02). Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 115–118.

Chan Lily, N. H. (2012). A cluster analysis approach to examining Singapore's property market . *BIS Papers*, 1-118.

Diederik P. Kingma, J. B. (2014, December 22). Adam: A method for stochastic optimization.

Goodfellow, I. B. (2016). *Deep Learning.* Cambridge, MA, USA: The MIT Press.

Hao Peng, J. L. (2021, september 28). Lifelong Property Price Prediction: A Case Study for the Toronto Real Estate Market. 2765 - 2780.

Jordan M. Barry, W. F. (2023, Octomber). COMMISSION-BASED STEERING IN RESIDENTIAL REAL ESTATE. 1-45.

LeCun, Y. B. (2015, May 27). Deep learning. *Nature*, 436–444.

Li Yu, C. J. (2018). Prediction on Housing Price Based on Deep Learning. *World Academy of Science, Engineering and Technology International Journal of Computer and Information Engineering, 12*, 90-99.

Mitchell, T. M. (1997). What is machine learning? *Machine Learning*.

Patel, R. K. (2021). Urban Analytics for Real Estate Planning in Smart Cities. *Journal of Urban Technology*, 45-60.

Rahimberdi Annamoradnejad, I. A. (2022, Octomber). Machine Learning for Housing Price Prediction. 2728-2739.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review, 65*, 386–408.

Runshan Fu, Y. H. (2022). Unequal Impact of Zestimate on the Housing Market. 1-54.

Song, J. (2024, April 18). The Effects of Residential Zoning in U.S. Housing Markets. 45.

Vinod Nair, G. E. (2010). Rectified linear units improve restricted Boltzmann machines. *Proceedings of the 27th International Conference on Machine Learning (ICML)*.

Yazdani, M. (2021, Octomber 15). Machine Learning, Deep Learning, and Hedonic Methods for Real Estate Price Prediction.

Zhou, B. W. (2021). Deep learning for real estate price prediction: A comparative study. *Neural Computing and Applications*, 6167-6180.

Computational_Intelligence(st20251281).docx

ORIGINALITY REPORT

12%	6%	7%	5%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Biswadip Basu Mallik, Gunjan Mukherjee, Rahul Kar, Aryan Chaudhary. "Deep Learning Concepts in Operations Research", Routledge, 2024 Publication	1%
2	Submitted to Montclair State University Student Paper	1%
3	Submitted to Eastern University Student Paper	1%
4	eitca.org Internet Source	1%
5	mmcalumni.ca Internet Source	<1%
6	github.com Internet Source	<1%
7	Berivan F Namq, Zakariya A Hussein, Sardar Qader Othman, Laith Ahmed Najam et al. "Evaluation of radiological risks due to natural radionuclides in the soil of Khanaqin district, Diyala, Iraq", Physica Scripta, 2024 Publication	<1%
8	Submitted to University of North Texas Student Paper	<1%
9	aiforsocialgood.ca Internet Source	<1%
10	Submitted to Victoria University Student Paper	<1%
11	Submitted to National College of Ireland Student Paper	<1%
12	rosap.ntl.bts.gov Internet Source	<1%
13	Submitted to Aston University Student Paper	<1%
14	Jay Liebowitz. "Data Analytics and AI", CRC Press, 2020 Publication	<1%
15	Submitted to Southern New Hampshire University - Continuing Education Student Paper	<1%
16	business.uc.edu Internet Source	<1%
17	Submitted to Monash University Student Paper	<1%
18	Submitted to Christian Community Ministries Student Paper	<1%
19	V. Sharmila, S. Kannadhasan, A. Rajiv Kannan, P. Sivakumar, V. Vennila. "Challenges in Information, Communication and Computing Technology", CRC Press, 2024 Publication	<1%
20	plaksha.edu.in Internet Source	<1%
21	Adeyemi Abel Ajibesin, Narasimha Rao Vajjhala. "AI for Humanitarianism - Fostering Social Change Through Emerging Technologies", CRC Press, 2025 Publication	<1%
22	Submitted to Purdue University Student Paper	<1%
23	Submitted to The Hong Kong Polytechnic University Student Paper	<1%
24	Xinyuan Song, HSIEH,WEI-CHE, Ziqian Bi, Chuanqi Jiang, Junyu Liu, Benji Peng, Sen Zhang, Xuanhe Pan, Jiawei Xu, Jinlang Wang. "A Comprehensive Guide to Explainable AI: From Classical Models to LLMs", Open Science Framework, 2024 Publication	<1%
25	daniellabarreto.com.br Internet Source	<1%
26	www.coursehero.com Internet Source	<1%
27	Federico Cugurullo, Federico Caprotti, Matthew Cook, Andrew Karvonen, Pauline McGuirk, Simon Marvin. "Artificial Intelligence and the City - Urbanistic Perspectives on AI", Routledge, 2023 Publication	<1%
28	Submitted to University of Glamorgan Student Paper	<1%
29	www.rapidinnovation.io Internet Source	<1%
30	www.researchgate.net Internet Source	<1%
31	www.sas.com Internet Source	<1%
32	Di Wu. "Data Mining with Python - Theory, Application, and Case Studies", CRC Press, 2024 Publication	<1%
33	Jonathan Masci, Emanuele Rodolà, Davide Boscaini, Michael M. Bronstein, Hao Li. "Geometric deep learning", SIGGRAPH ASIA 2016 Courses on - SA '16, 2016 Publication	<1%
34	Submitted to University of Wolverhampton Student Paper	<1%
35	link.springer.com Internet Source	<1%
36	www.businesswire.com Internet Source	<1%
37	www.ncbi.nlm.nih.gov Internet Source	<1%
38	"Discovery Science", Springer Science and Business Media LLC, 2025 Publication	<1%
39	Rajeev Sobti, Rachit Garg, Ajeet Kumar Srivastava, Gurpreet Singh Shahi, . "Computer Science Engineering and Emerging Technologies (ICCS-2022) - 6 International Conference on Computing Sciences 2022", CRC Press, 2024 Publication	<1%
40	aircconline.com Internet Source	<1%
41	hal.archives-ouvertes.fr Internet Source	<1%
42	rep.bioscientifica.com Internet Source	<1%
43	T. Mariprasath, Kumar Reddy Cheepati, Marco Rivera. "Practical Guide to Machine Learning, NLP, and Generative AI: Libraries, Algorithms, and Applications", River Publishers, 2024 Publication	<1%

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	On		