Artificial Intelligence and Machine Learning

Project Report

Semester-IV (Batch-2022)

House Sale Prediction

Supervised By:

Dr. Jatin Arora

Submitted By:

Dhruv Shukla, 2210990278

Dhruv Sood, 2210990279

Divanshu Garg,221990289

Khushdeep Sharma,2210990509

Department of Computer Science and Engineering

Chitkara University Institute of Engineering & Technology,

Chitkara University, Punjab

CONTENT

INTRODUCTION

Background

In the realm of real estate, predicting house prices accurately holds immense significance for buyers, sellers, and investors alike. Leveragingthe power of Artificial Intelligence (AI) and Machine Learning (ML), our project delves into this complex domain, aiming to develop robust predictive models. By analyzing diverse datasets encompassing key features such as location, size, and amenities, we seek to provide valuable insights into the factors influencing property values. This reportoffers a comprehensive account of our methodology, findings, and their potential implications in the housing market.

Objectives

1. Develop predictive models using AI and ML techniques to accurately forecast house prices based on key features such as location, size, and amenities.

2. Conduct thorough data analysis and preprocessing to identify relevantpatterns and relationships between input variables and property values.

3. Evaluate the performance of the constructed models through rigoroustesting and validation processes, aiming to provide actionable insights for stakeholders in the real estate industry.

Significance

The significance of this project lies in its potential to revolutionize the real estate landscape by enhancing the accuracy of house sale predictions. This advancement directly benefits buyers, equipping them with the tools to make informed decisions regarding their budgets, negotiations, and overall property selection. With a clearer understanding of market trends and property values, buyers can confidently navigate the purchasing process, leading to greater satisfaction and financial security.

Accurate house sale predictions also empower sellers and real estate professionals. Sellers can strategically price their properties, ensuring optimal market exposure and faster sales. Real estate agents and brokers can utilize this data to provide valuable insights to clients, facilitating smoother transactions and fostering trust. Moreover, appraisers can leverage accurate predictions to assess property values more precisely, contributing to a more stable and efficient real estate market.

Beyond individual transactions, this project has broader implications for the entire industry. By promoting transparency

and fairness through reliable data, it fosters a healthier economic environment and reduces market volatility. Additionally, accurate house sale predictions can inform urban planning and development, leading to better resource allocation and sustainable growth. This project ultimately aims to create a more informed, efficient, and equitable real estate market for all stakeholders involved.

Problem Definition and Requirements

Problem Statement

The current methods for Predicting House price often lack precision dueto their reliance on generic formulas. This project aims to create a more accurate model by leveraging machine learning algorithms and comprehensive datasets.

Software/Hardware Requirements

- Software: Python, Jupyter Notebooks,, Pandas, Matplotlib/Seaborn,Scikit-Learn
- Hardware: A machine with at least 8GB RAM and a multi-coreprocessor
- Datasets: Publicly available fitness datasets or custom datasetscreated by the project team

Proposed Design / Methodology

The proposed design for the House Price Prediction project involves the following key steps:

1. Data Collection and Preprocessing: Collect and clean data sets withrelevant attributes such as House Price , Neighbourhood , Availability etc.

2. Feature Engineering: Identify and create meaningful features fortraining the ML model.

3. Model Selection: Choose suitable algorithms for prediction, such as linear regression, decision trees, or neural networks.

4. Model Training and Validation: Train the model on the dataset and validate its accuracy using appropriate metrics like RMSE, MAE, orR-squared.

5. Model Tuning and Optimization: Tune hyperparameters and optimizethe model for better performance.

6. Model Deployment: Outline how the model could be deployed in areal-world application.

Tool Used

- Google colaboratory is used as IDE:

  Google Colab, short for Google Colaboratory, is a cloud-based platform that provides free access to computational resources for machine learning and data analysis tasks. It offers a convenient environment for writing, running, and sharing Python code through Jupyter notebooks. With built-in support for popular libraries like TensorFlow, PyTorch, and scikit-learn, Colab enables seamless integration with state-of-the-art machine learning frameworks. Its collaborative features allow multiple users to workon the same notebook simultaneously, fostering teamwork and knowledge sharing. Additionally, Colab leverages Google's robust infrastructure to provide access to powerful GPU and TPU accelerators, enabling accelerated model training and experimentation. Its user-friendly interface and integration with Google Drive make it an ideal choice for both beginners and experienced practitioners looking to explore, prototype, and deploy machine learning solutions in a collaborative and scalable manner.

- Pandas and NumPy are used for Data Manipulation & Pre-processingand Mathematical functions respectively:

  Pandas is a Python library widely used for data manipulation andanalysis. It provides data structures like DataFrame and Series, which are powerful tools for handling structured data.
  With its intuitive and flexible syntax, Pandas simplifies tasks such as datacleaning, transformation, and aggregation. It offers a wide rangeof functions and methods for indexing, slicing, merging, and reshaping datasets, making it a go-to choice for data preprocessing in data science and machine learning workflows.

NumPy, on the other hand, is a fundamental library for numerical computing in Python. It provides support for multidimensional arrays, along with a collection of mathematical functions to operate on these arrays efficiently. NumPy's array operations are significantly faster than traditional Python lists, making it essential for tasks involving large datasets or complex mathematical computations. With its rich set of mathematical functions and capabilities for linear algebra, Fourier analysis, and random number generation, NumPy serves as the backbone for many scientific computing tasks and machine learning algorithms.

- For visualization of the plots, Matplotlib, Seaborn are used:

    Matplotlib and Seaborn are popular Python libraries used for datavisualization, each offering unique capabilities and styles.

    Matplotlib is a comprehensive plotting library that provides a widerange of plotting functions to create static, interactive, and publication-quality visualizations. It offers fine-grained control over every aspect of a plot, allowing users to customize colors, styles, annotations, and more. Matplotlib's versatility makes it suitable for a broad spectrum of visualization tasks, from simple line plots and scatter plots to complex heatmaps and 3D plots.

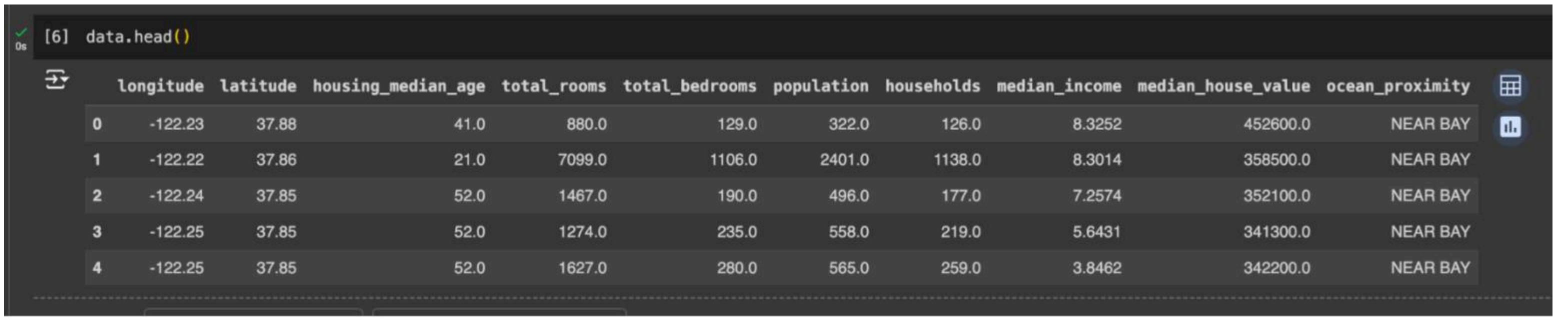
Seaborn, built on top of Matplotlib, offers a higher-level interface for creating attractive and informative statistical graphics. It simplifies the process of creating complex visualizations by providing built-in support for statistical estimation and automatic styling options. Seaborn excels in creating visually appealing plots for exploring relationships between variables, such as scatter plotswith regression lines, distribution plots, and categorical plots.

Together, Matplotlib and Seaborn form a powerful duo for data visualization, catering to a diverse range of visualization needs andpreferences in data analysis, exploration, and presentation. Their seamless integration and complementary features make them indispensable tools for communicating insights and patterns in data effectively.

- GitHub is used as version control system

Data Summary

This is the House Price Prediction . In the below table it shows the top 5rows



```
[6] data.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY

Features Description

- Longitude: Geographic coordinate indicating the east-west positionof a house location.
- Latitude: Geographic coordinate indicating the north-south positionof a house location.
- Housing Median Age: Median age of houses in a specific area.
- Total Rooms: Total number of rooms in a house.
- Total Bedrooms: Total number of bedrooms in a house.
- Population: Total population in the vicinity of a house.
- Households: Total number of households in a specific area.
- Median Income: Median income of households in a given area.
- Median House Value: Median value of houses in a specific area.
- Ocean Proximity: Proximity of the house to the ocean, categorized into different

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

data = pd.read_csv("housing.csv")

# data.head()

data.info() # Checking how many values are null
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object 
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

```python
data.dropna(inplace=True)

data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 20433 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   longitude           20433 non-null  float64
 1   latitude            20433 non-null  float64
 2   housing_median_age  20433 non-null  float64
 3   total_rooms         20433 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20433 non-null  float64
 6   households          20433 non-null  float64
 7   median_income       20433 non-null  float64
 8   median_house_value  20433 non-null  float64
 9   ocean_proximity     20433 non-null  object 
dtypes: float64(9), object(1)
memory usage: 1.7+ MB
```

Data Splitting:

Data Splitting:

Data splitting is a crucial step in the machine learning pipeline, as it involves dividing the available dataset into separate subsets for training, validation, and testing. This process helps ensure the generalizability androbustness of the trained models by assessing their performance on unseen data.

To implement data splitting effectively, several strategies can be employed depending on the specific requirements of the task and the characteristics of the dataset. One common approach is the simple random sampling method, where data points are randomly assigned to each subset without any bias. This method is straightforward and easy to implement, making it suitable for various scenarios.

Another commonly used technique is the stratified sampling method, which ensures that the distribution of target variables or classes remainsconsistent across the different subsets. This is particularly useful when dealing with imbalanced datasets, where certain classes may be underrepresented. By maintaining the same class distribution in each subset, stratified sampling helps prevent bias and ensures fair evaluationof the models' performance.

Additionally, data splitting can be performed using techniques such as cross-validation, where the dataset is divided into multiple folds, and each fold is used for both training and validation in turn. This allows fora more comprehensive assessment of the model's performance across different subsets of the data and helps mitigate the risk of overfitting.

Overall, proper data splitting is essential for building reliable and accurate machine learning models. By carefully partitioning the dataset into training, validation, and testing sets, practitioners can train modelsthat generalize well to unseen data and make informed decisions about model selection and hyperparameter tuning.

Training Set:

The training set is used to train the machine learning model. It typicallycomprises the majority of the dataset, allowing the model to learn patterns and relationships from the data. A common split is to allocate around 70-80% of the data to the training set.

Validation Set:

The validation set plays a pivotal role in the machine learning workflow,serving as a critical tool for fine-tuning hyperparameters and assessing model performance throughout the training process. By utilizing a separate validation set, distinct from the training data, practitioners can effectively gauge how well the model generalizes to unseen examples and identify potential issues such as overfitting or underfitting.

During model training, hyperparameters control the learning process anddirectly influence the model's performance. Through iterative experimentation with different hyperparameter configurations, practitioners seek to optimize the model's performance on the validationset. This iterative process, often referred to as hyperparameter tuning oroptimization, involves adjusting parameters such as learning rates, regularization strengths, and network architectures to strike the right balance between bias and variance.

The validation set acts as a proxy for unseen data, allowing practitioners to evaluate the model's performance in a controlled environment before deploying it to real-world scenarios. By monitoring the model's performance on the validation set across multiple training iterations, practitioners can make informed decisions about hyperparameter settingsand model architectures, ultimately improving the model's generalizationability.

Typically, a portion of the available data, typically around 10-15%, is allocated to the validation set. This allocation ensures that an adequate number of examples are available for validation while still preserving asufficient amount of data for training. By striking this balance, practitioners can effectively leverage the validation set to optimize model performance while mitigating the risk of overfitting or hyperparameter bias.

Testing Set:

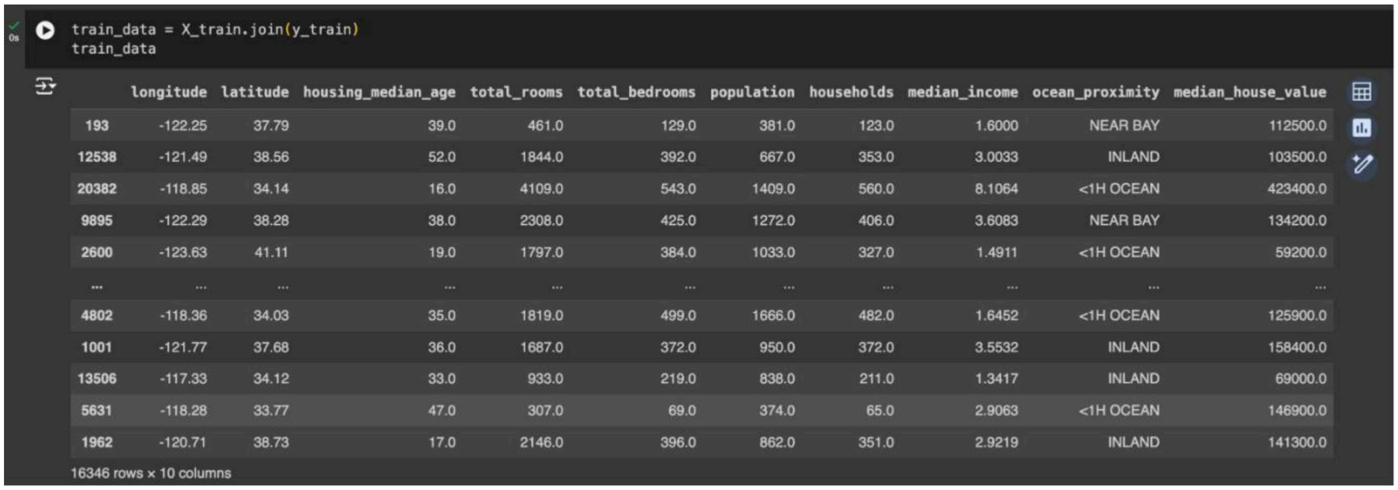
The testing set represents a critical component in the machine learning workflow, providing an independent dataset for the final evaluation of the trained model's performance. Unlike the training and validation sets,the testing set comprises unseen data examples that the model has not encountered during the training or validation phases. This ensures a rigorous assessment of the model's ability to generalize to new, unseen instances and provides valuable insights into its real-world performance.

By evaluating the model on the testing set, practitioners gain confidence in its generalization capabilities and its readiness for deployment in
real-world scenarios. The testing set serves as a proxy for the model's performance in production environments, where it encounters novel datapoints that may differ from those seen during training or validation.
Thus, the testing set provides a crucial benchmark for assessing the model's robustness, reliability, and effectiveness in real-world applications.

To maintain the integrity of the testing process, it is essential to keep thetesting set separate from the training and validation data throughout the model development cycle. This ensures that the model's performance on the testing set remains unbiased and accurately reflects its true generalization ability. Moreover, the testing set should only be used once, after finalizing the model and hyperparameter tuning, to avoid data leakage and overfitting.

In summary, the testing set serves as the ultimate arbiter of a model's performance, offering a reliable measure of its effectiveness in handlingunseen data. By rigorously evaluating the model on a separate testing set, practitioners can validate its generalization capabilities and make informed decisions about its deployment in real-world applications.

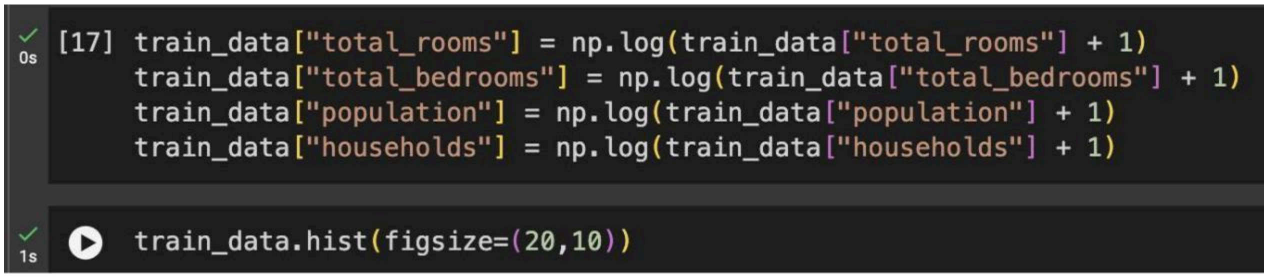
Code Snippets:



```
train_data = X_train.join(y_train)
train_data
```

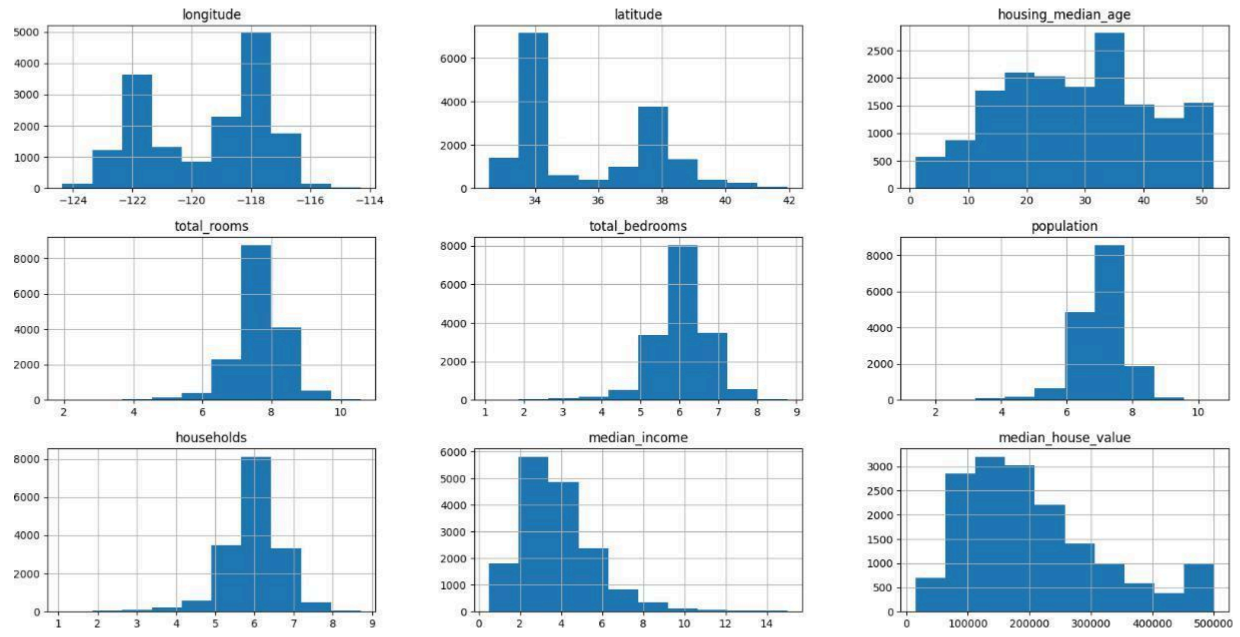
	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity	median_house_value
193	-122.25	37.79	39.0	461.0	129.0	381.0	123.0	1.6000	NEAR BAY	112500.0
12538	-121.49	38.56	52.0	1844.0	392.0	667.0	353.0	3.0033	INLAND	103500.0
20382	-118.85	34.14	16.0	4109.0	543.0	1409.0	560.0	8.1064	<1H OCEAN	423400.0
9895	-122.29	38.28	38.0	2308.0	425.0	1272.0	406.0	3.6083	NEAR BAY	134200.0
2600	-123.63	41.11	19.0	1797.0	384.0	1033.0	327.0	1.4911	<1H OCEAN	59200.0
...	...	...	...	...	...	...	...	...	...	...
4802	-118.36	34.03	35.0	1819.0	499.0	1666.0	482.0	1.6452	<1H OCEAN	125900.0
1001	-121.77	37.68	36.0	1687.0	372.0	950.0	372.0	3.5532	INLAND	158400.0
13506	-117.33	34.12	33.0	933.0	219.0	838.0	211.0	1.3417	INLAND	69000.0
5631	-118.28	33.77	47.0	307.0	69.0	374.0	65.0	2.9063	<1H OCEAN	146900.0
1962	-120.71	38.73	17.0	2146.0	396.0	862.0	351.0	2.9219	INLAND	141300.0

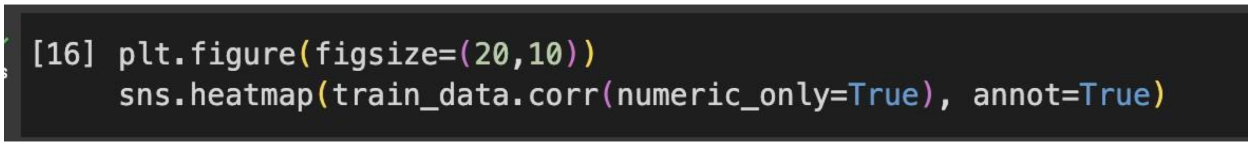
16346 rows × 10 columns



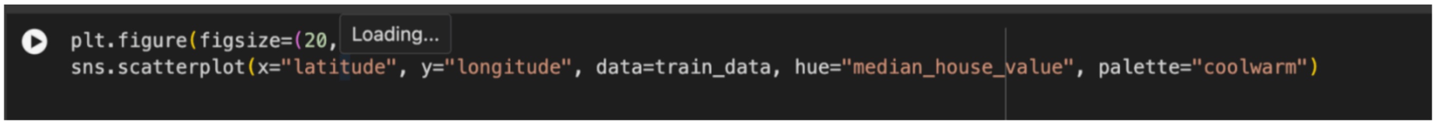
```
[17] train_data["total_rooms"] = np.log(train_data["total_rooms"] + 1)
     train_data["total_bedrooms"] = np.log(train_data["total_bedrooms"] + 1)
     train_data["population"] = np.log(train_data["population"] + 1)
     train_data["households"] = np.log(train_data["households"] + 1)
```

```
train_data.hist(figsize=(20,10))
```

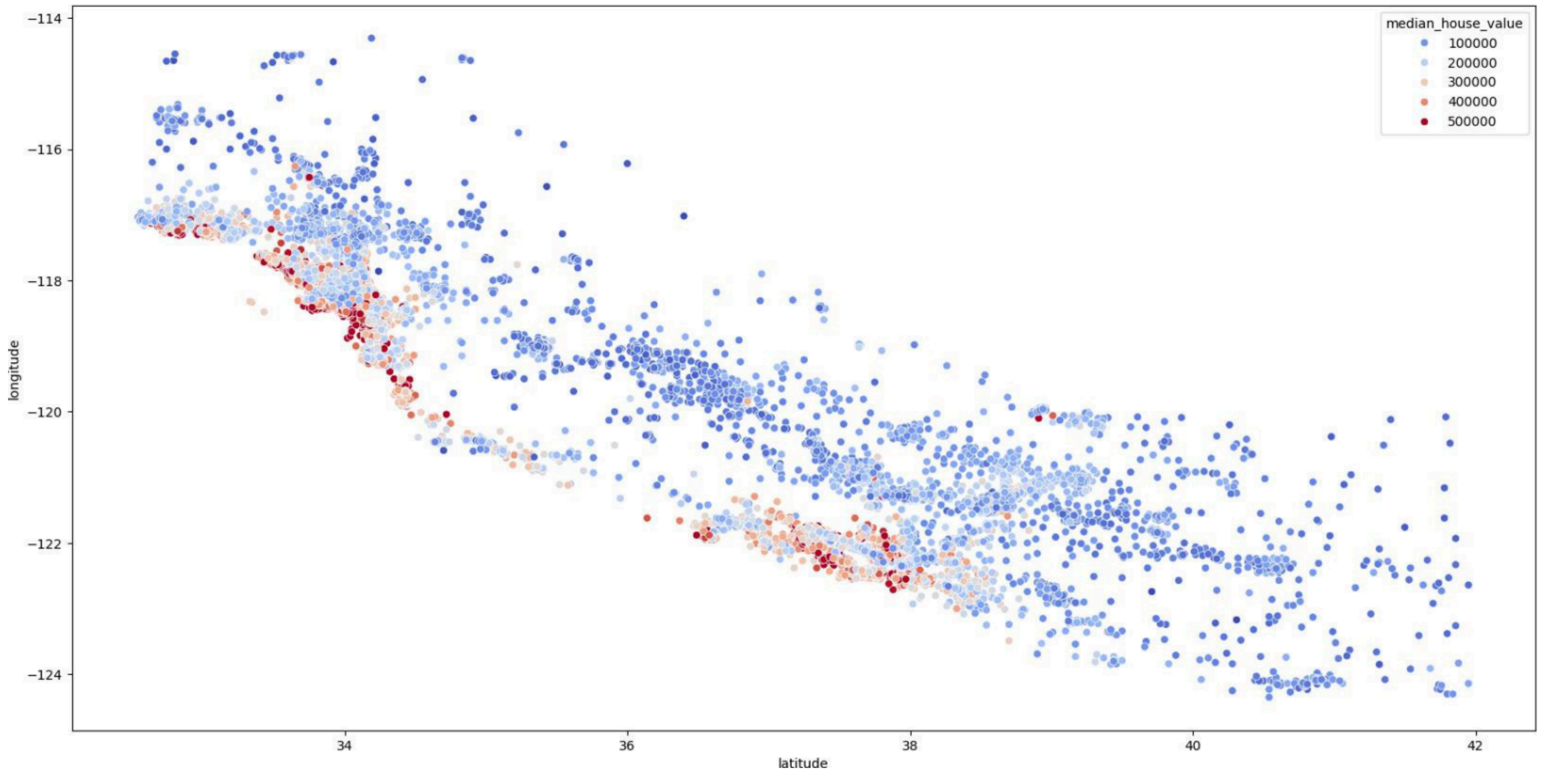


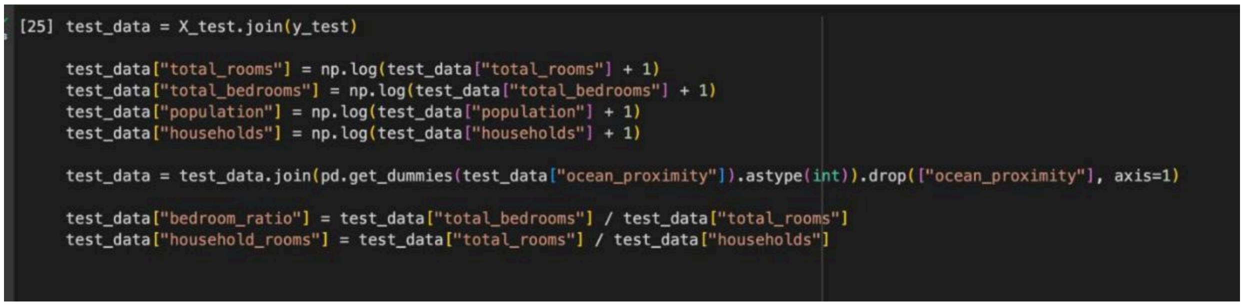


```
[16] plt.figure(figsize=(20,10))
     sns.heatmap(train_data.corr(numeric_only=True), annot=True)
```

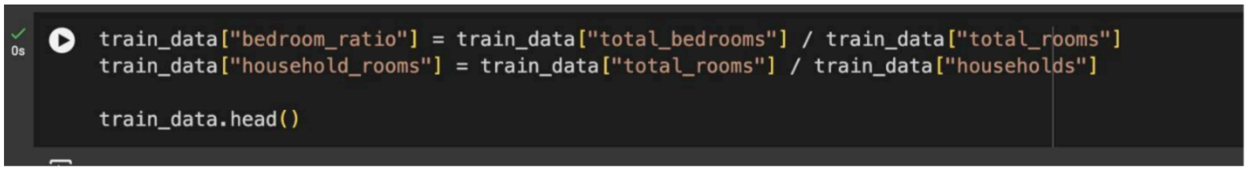


```
plt.figure(figsize=(20,
sns.scatterplot(x="latitude", y="longitude", data=train_data, hue="median_house_value", palette="coolwarm")
```

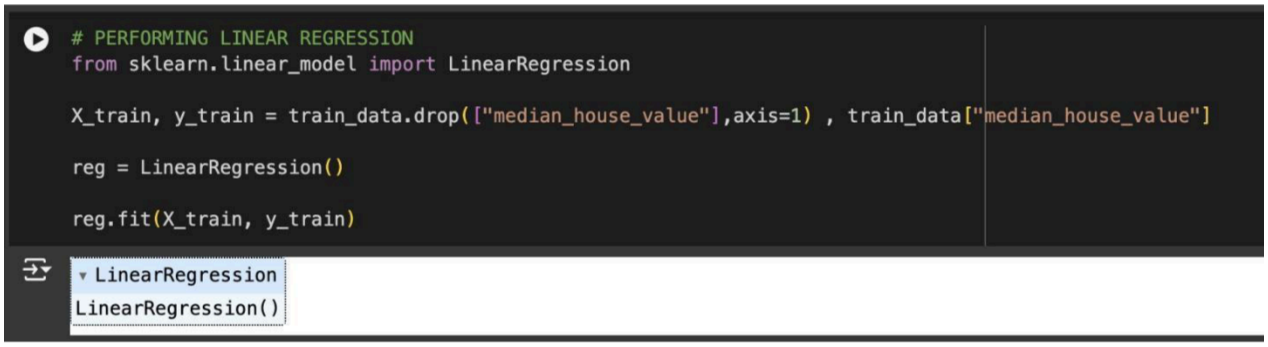




```
[25] test_data = X_test.join(y_test)

     test_data["total_rooms"] = np.log(test_data["total_rooms"] + 1)
     test_data["total_bedrooms"] = np.log(test_data["total_bedrooms"] + 1)
     test_data["population"] = np.log(test_data["population"] + 1)
     test_data["households"] = np.log(test_data["households"] + 1)

     test_data = test_data.join(pd.get_dummies(test_data["ocean_proximity"]).astype(int)).drop(["ocean_proximity"], axis=1)

     test_data["bedroom_ratio"] = test_data["total_bedrooms"] / test_data["total_rooms"]
     test_data["household_rooms"] = test_data["total_rooms"] / test_data["households"]
```

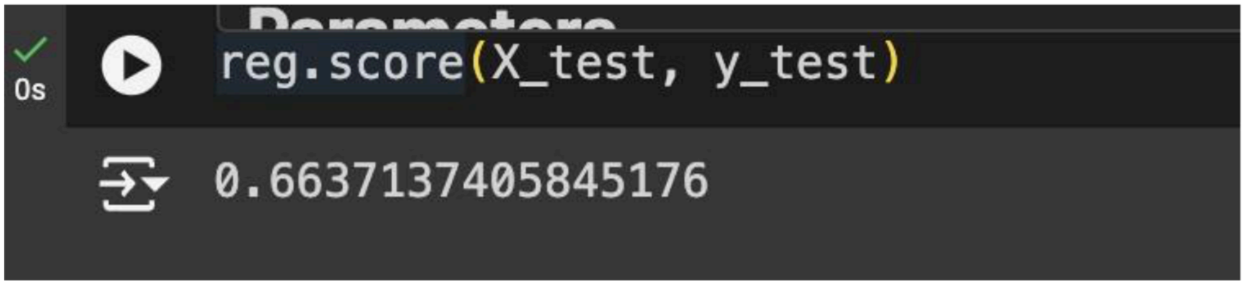


```
train_data["bedroom_ratio"] = train_data["total_bedrooms"] / train_data["total_rooms"]
train_data["household_rooms"] = train_data["total_rooms"] / train_data["households"]

train_data.head()
```



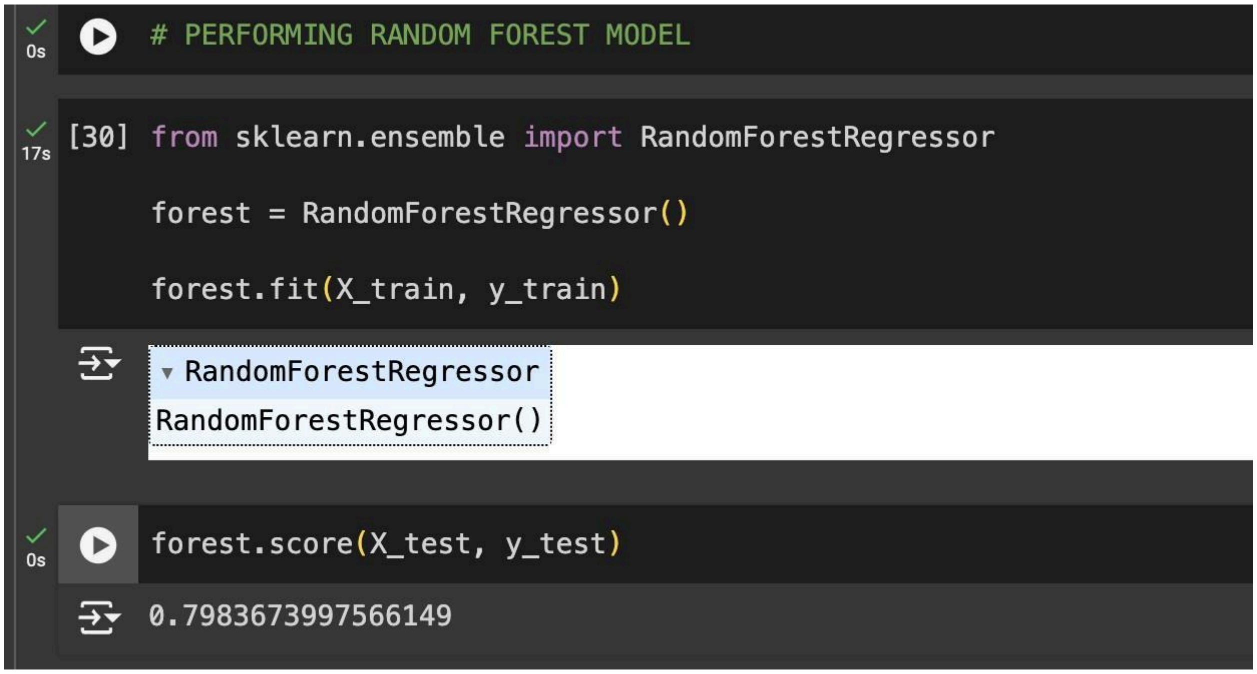
```
# PERFORMING LINEAR REGRESSION
from sklearn.linear_model import LinearRegression

X_train, y_train = train_data.drop(["median_house_value"],axis=1) , train_data["median_house_value"]

reg = LinearRegression()

reg.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```



```
reg.score(X_test, y_test)
```

```
0.6637137405845176
```



```
# PERFORMING RANDOM FOREST MODEL
```

```
[30] from sklearn.ensemble import RandomForestRegressor

     forest = RandomForestRegressor()

     forest.fit(X_train, y_train)
```

```
▾ RandomForestRegressor
RandomForestRegressor()
```

```
forest.score(X_test, y_test)
```

```
0.7983673997566149
```

Conclusion:

1. **Summary of Findings:** Recap the main insights gained from the analysis, including trends in sale prices, factors influencing property values, and patterns in buyer behavior.

2. **Implications for Stakeholders:** Discuss how the project findings can benefit various stakeholders, including real estate agents, property developers, investors, and homebuyers. Highlight specific actions they can take basedon the insights uncovered.

3. **Limitations:** Acknowledge any limitations or constraints encountered during the project, such as data quality issues, model assumptions, or scope constraints. This demonstratestransparency and helps contextualize the findings.

4. **Future Research Directions:** Suggest potential avenues for future research or enhancements to the project, such as incorporating additional data sources, refining models with advanced techniques, or exploring emerging trends in the housing market

REFERENCES:

California Housing Data (kaggle.com)

NueralNine Walkthrough