Summative Assessment Section C

Name: Sonal Agarwal

Random Forest Regression: A Study on California Housing Prices

ABSTRACT

This study focuses on predicting California housing prices using machine learning techniques, with data sourced from Kaggle. We tested multiple regression models, including Linear Regression, Ridge, Lasso, Decision Tree, and Random Forest. Among these, Random Forest emerged as the most accurate, achieving a validation R^2 score of 0.83. Its strength lies in handling complex patterns, reducing overfitting, and effectively processing both numerical and categorical data. Through preprocessing, feature engineering, and tuning, we enhanced model performance, highlighting the value of machine learning in tackling real-world challenges. This research offers important insights for professionals in real estate, urban development, and policy-making, demonstrating how machine learning can effectively forecast housing prices and support informed decision-making.

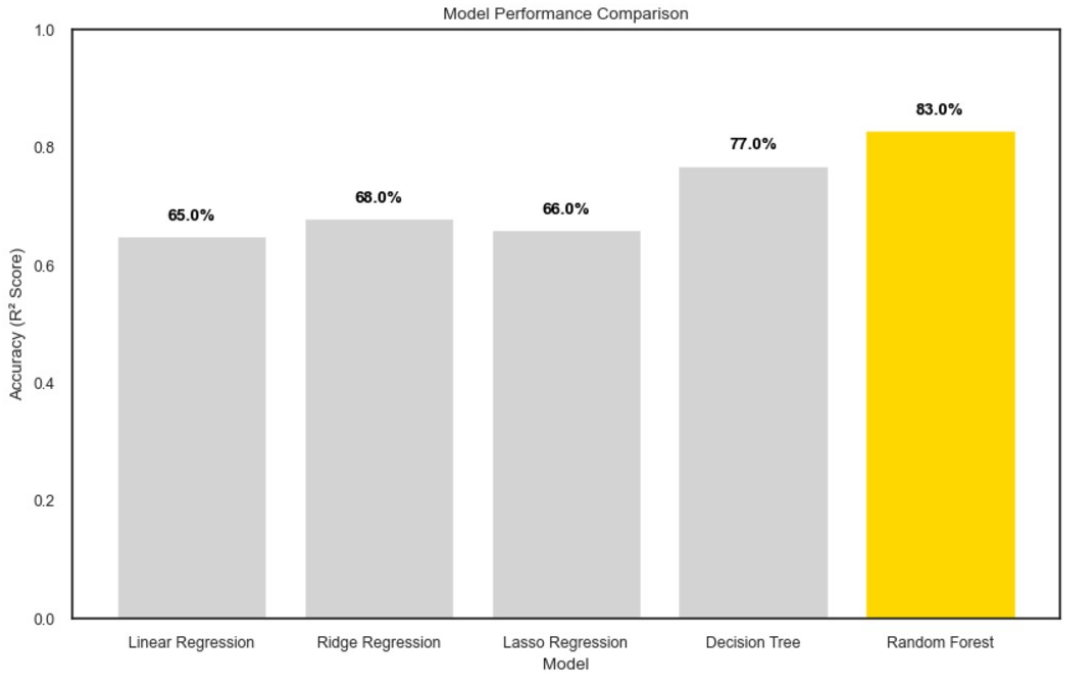
---

1. Introduction to Supervised Learning & Random Forest Regression

Supervised learning is a machine learning approach where a model is trained using data that has both inputs and corresponding correct outputs. The objective is for the model to learn the relationship between the input and output so it can make accurate predictions on new, unseen data. There are two main types of supervised learning tasks: regression, which involves predicting continuous values, and classification, which involves predicting categories or labels. In this report, we focus on a regression task, specifically predicting housing prices in California.

Random Forest Regression is a powerful machine learning technique commonly used for predicting continuous values. It works by creating multiple decision trees during training and then combining their outputs through averaging to make predictions. This method reduces the risk of overfitting, helping the model generalize better. Random Forest is especially useful when dealing with complex data patterns that are non-linear, making it ideal for datasets like housing prices that involve a variety of features.

Although Gradient Boosted Trees, such as XGBoost, often deliver better accuracy, Random Forest is preferred for its straightforward approach, quicker training process, and lower sensitivity to hyperparameter tuning.



Bar Graph of Model Performance Comparison

- **Key Principles**

- **Ensemble Learning:** Random Forest is based on the idea of ensemble learning, where multiple models (specifically decision trees) work together to make predictions. The final prediction is made by combining the results of each tree, which helps improve the accuracy of the model.

- **Bootstrapping:** This technique involves creating several random subsets of the training data by sampling with replacement.

- **Random Feature Selection:** To add another layer of randomness and prevent overfitting, Random Forest randomly selects a subset of features to consider at each split in the tree-building process.

- **Prediction by Averaging:** Random Forest combines their individual predictions.

- **Training Process in Random Forest**

- Preparing the Data
- Building the Trees
- Aggregating Predictions

- **Making Predictions on Test Data**

  When making predictions on new, unseen data (test data), the Random Forest model works as follows:

1. Each tree in the forest makes its own prediction based on the test data.
2. The predictions from all trees are then averaged to produce a final result.

- **Assumptions in Random Forest Regression**

1. **Independence of Trees**
   Random Forest assumes that the individual decision trees are independent.

2. **No Assumption of Feature Relationships**
   Unlike some models like linear regression, Random Forest does not assume any specific relationship between the features.

3. **Data Size**
   Random Forest generally performs well with large datasets, as the model benefits from having enough data to train multiple decision trees and produce robust predictions.

- **Real-World Problem Context**

  Predicting housing prices is essential for various sectors such as real estate, urban planning, and economic policy. The California Housing dataset provides detailed data on neighborhood characteristics, housing features, and demographics, which helps predict housing prices. This information is valuable in several ways:

- **Real Estate Transactions**: Assisting buyers and sellers in estimating property values.
- **Urban Planning**: Identifying areas with high potential for growth.
- **Policy Formulation**: Understanding socioeconomic factors affecting housing affordability.

---

2. Dataset Overview and Preprocessing

- **Data Source**

  The dataset posed a challenge due to housing prices being capped at $500,000, which may restrict the model's ability to predict values beyond this limit. Using alternative datasets or applying data augmentation techniques could help overcome this issue.

For this analysis, the California Housing dataset, available on Kaggle, was used. This dataset contains data about different neighborhoods in California, including features like median income, population density, housing details, and geographic information. The target variable is the median house value.

	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

First 5 rows of the 'housing_price_df' DataFrame

➢ Feature Importance

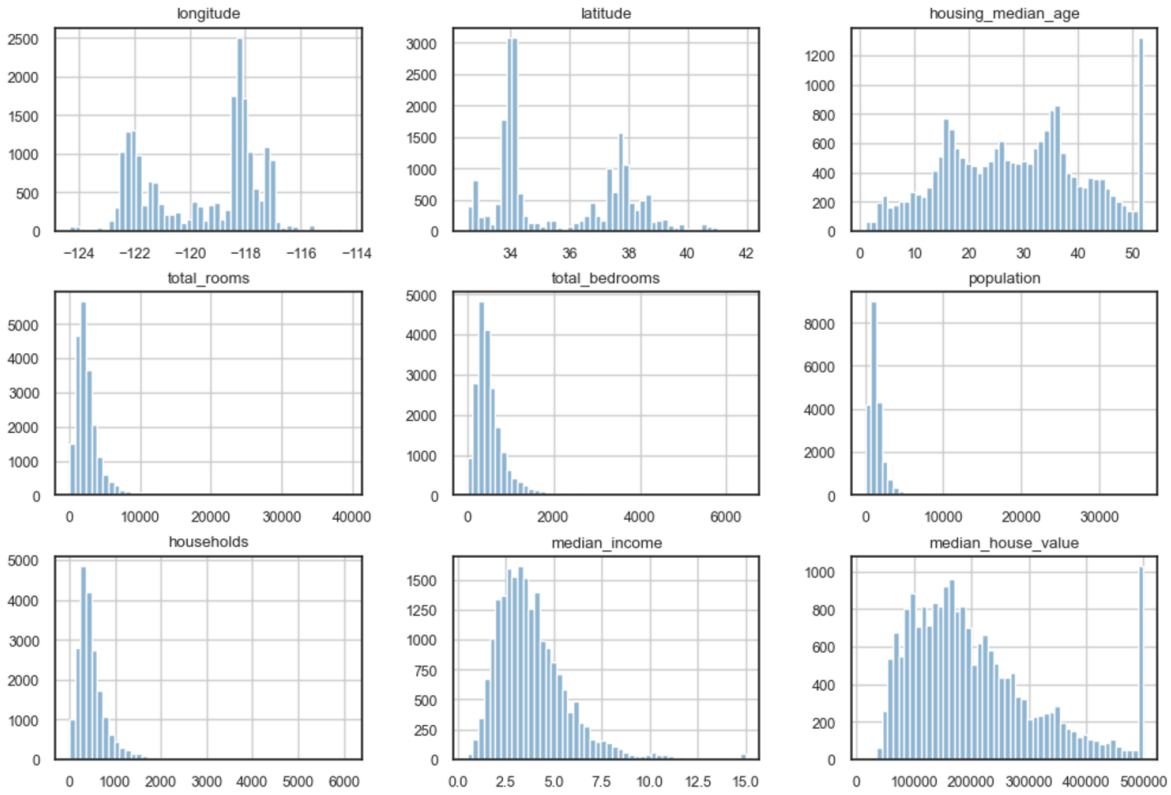
- **Median Income:** The income of people has the biggest effect on housing prices.
- **House Age:** Older homes usually have lower prices.
- **Average Rooms:** Bigger houses with more rooms usually have higher prices.
- **Latitude/Longitude:** Where a house is located (in a city or the countryside) also affects its price a lot.

➢ Dataset Characteristics

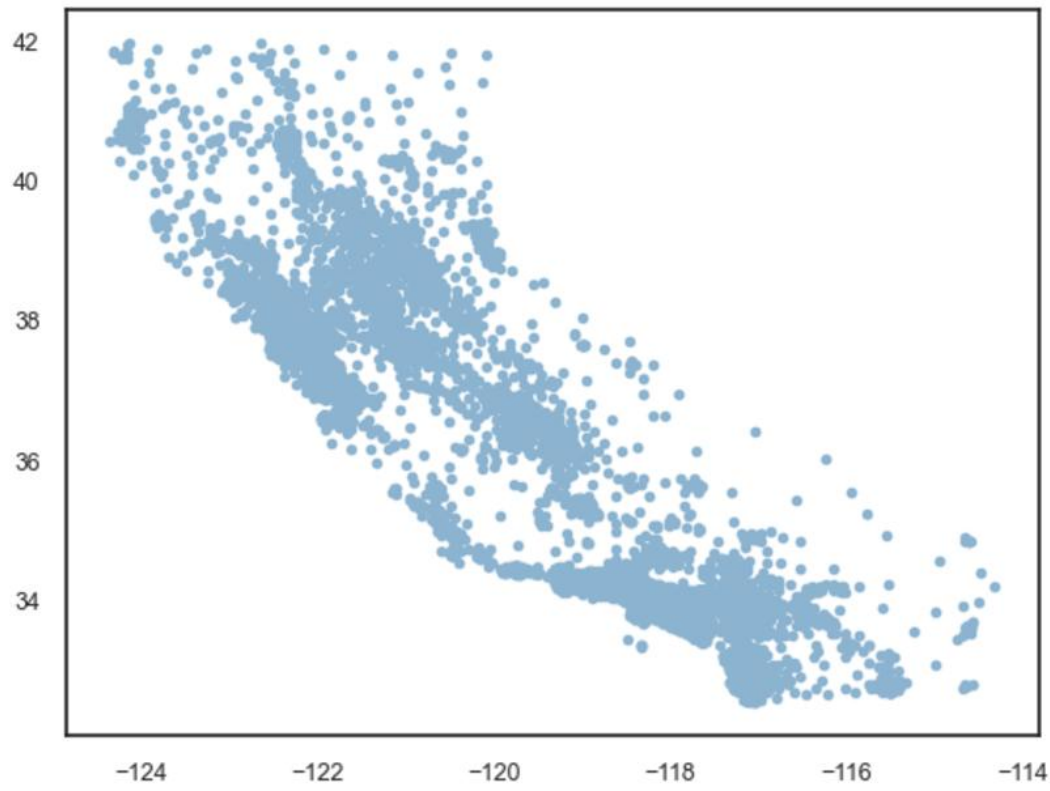
- **Numerical Features**: Includes attributes such as longitude, latitude, housing median age, total rooms, total bedrooms, population, households, and median income.

```
<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
```

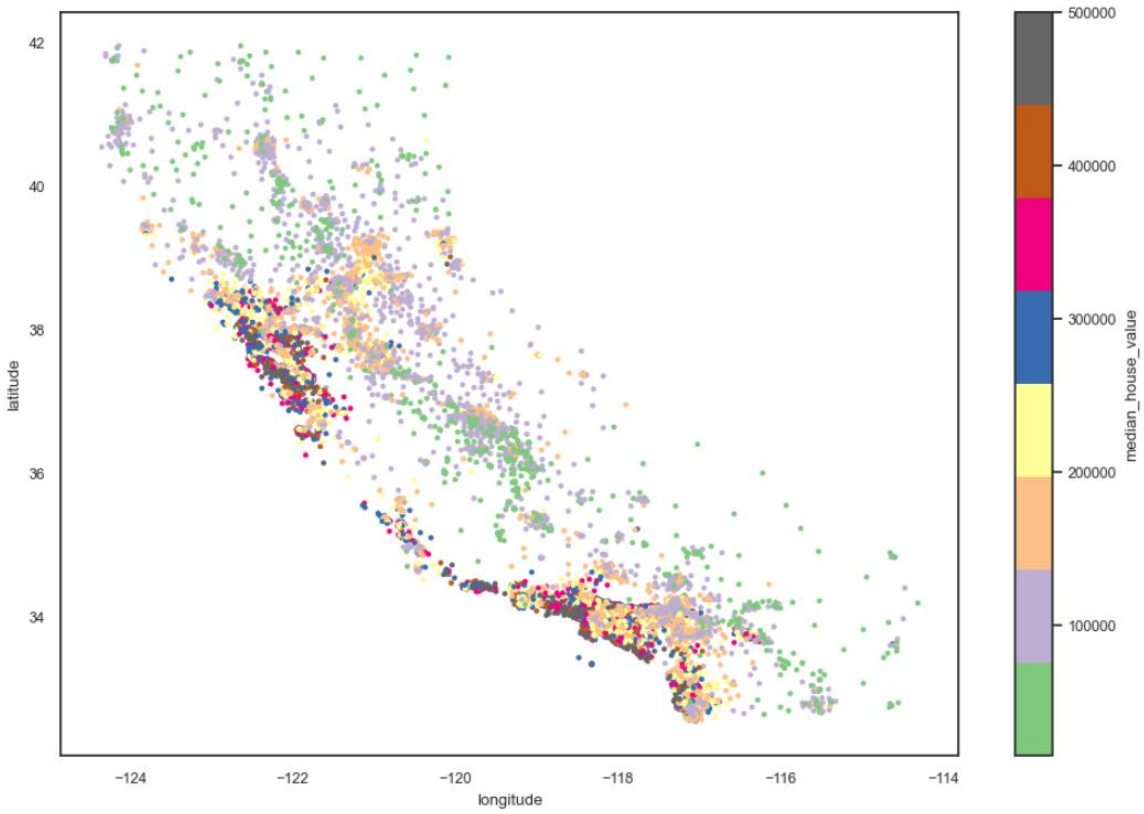
Displaying the dataset structure, columns, and data types.



Histograms of numerical features



Plot of geographical data



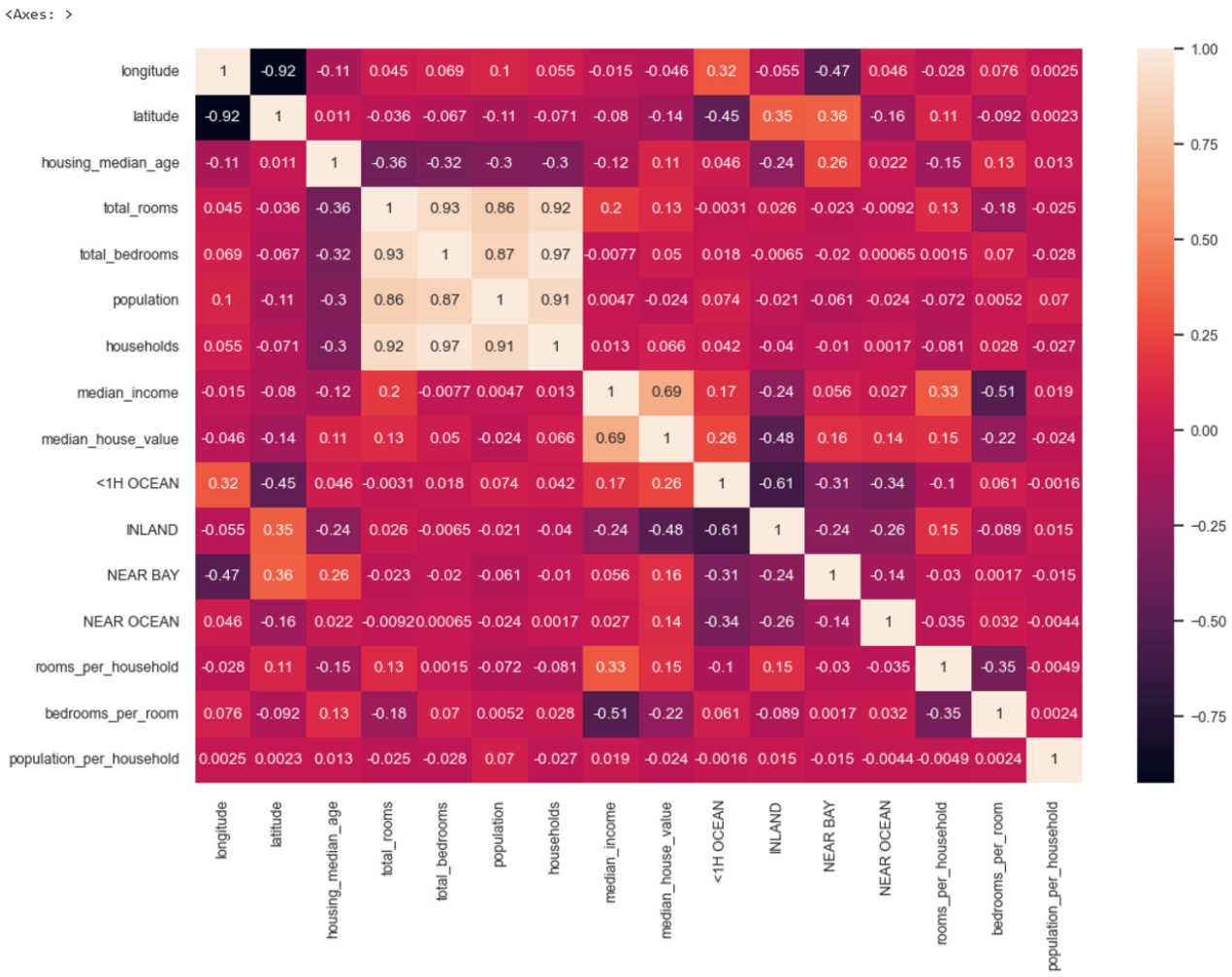
Median House Value across geographic locations.

- **Categorical Features**: The "ocean_proximity" feature indicates how close the location is to the ocean.
- **Target Variable**: The target variable is the "median_house_value," representing housing prices.
- **Dataset Size**: It consists of 20,640 rows and nine features.

➢ Exploratory Data Analysis (EDA)

The dataset was explored to understand its key patterns and relationships:

- **Correlations**: The "median_income" feature showed the strongest correlation with housing prices, while geographic features like latitude and longitude had weaker relationships.



Heatmap visualization of feature correlations

- **Distributions**: Certain variables, such as housing prices, showed skewness, with many values capped at $500,000.
- **Missing Data**: The "total_bedrooms" feature had missing values, which were handled by filling in the missing values with the median.

```
longitude                 0
latitude                  0
housing_median_age        0
total_rooms               0
total_bedrooms            0
population                0
households                0
median_income             0
median_house_value        0
<1H OCEAN                 0
INLAND                    0
NEAR BAY                  0
NEAR OCEAN                0
rooms_per_household       0
bedrooms_per_room         0
population_per_household  0
dtype: int64
```

Missing data in the total_bedrooms feature was imputed

Feature Engineering

To enhance the model's predictive capability, the following new features were added:

- **Rooms per Household**: This ratio of total rooms to households reflects the density of rooms available in a household.
- **Bedrooms per Room**: This ratio helps indicate the quality of the housing based on the number of bedrooms relative to total rooms.
- **Population per Household**: This ratio shows the average household size, which may influence housing price.

```python
# Create new derived features
housing_price_df["rooms_per_household"] = housing_price_df["total_rooms"] / housing_price_df["households"]
housing_price_df["bedrooms_per_room"] = housing_price_df["total_bedrooms"] / housing_price_df["total_rooms"]
housing_price_df["population_per_household"] = housing_price_df["population"] / housing_price_df["households"]
```

Code snippet showcasing the creation of features like rooms_per_household, bedrooms_per_room, and population_per_household.

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	<1H OCEAN	INLAND	NEAR BAY	NEAR OCEAN	rooms_per_household	bedrooms_per_room	population_per_household	Colum
longitude	1	-0.924676	-0.108394	0.044642	0.069304	0.099881	0.0554	-0.01509	-0.046208	0.321297	-0.05548	-0.474465	0.045568	-0.027558	0.076118	0.002486	
latitude	-0.924676	1	0.011462	-0.036231	-0.066741	-0.108978	-0.071199	-0.079977	-0.143837	-0.4473	0.351058	0.358735	-0.160942	0.106431	-0.092156	0.002349	
housing_median_age	-0.108394	0.011462	1	-0.361268	-0.319033	-0.296172	-0.302863	-0.118949	0.105272	0.045551	-0.236537	0.255333	0.021729	-0.153422	0.130615	0.013211	
total_rooms	0.044642	-0.036231	-0.361268	1	0.927255	0.857117	0.91848	0.197991	0.134373	-0.00314	0.025546	-0.023065	-0.009221	0.133802	-0.180225	-0.02459	
total_bedrooms	0.069304	-0.066741	-0.319033	0.927255	1	0.87392	0.974737	-0.007725	0.049559	0.018162	-0.00648	-0.01981	0.00065	0.001528	0.069738	-0.02836	
population	0.099881	-0.108978	-0.296172	0.857117	0.87392	1	0.907213	0.004737	-0.024421	0.07448	-0.020845	-0.060942	-0.024328	-0.072207	0.005207	0.069856	
households	0.0554	-0.071199	-0.302863	0.91848	0.974737	0.907213	1	0.01295	0.066069	0.042315	-0.039503	-0.010144	0.00166	-0.080593	0.027558	-0.02732	
median_income	-0.01509	-0.079977	-0.118949	0.197991	-0.007725	0.004737	0.01295	1	0.688563	0.168772	-0.237618	0.05615	0.02729	0.326915	-0.51096	0.018757	
median_house_value	-0.046208	-0.143837	0.105272	0.134373	0.049559	-0.024421	0.066069	0.688563	1	0.257049	-0.484794	0.160467	0.142051	0.151968	-0.220455	-0.02372	
<1H OCEAN	0.321297	-0.447303	0.045551	-0.003136	0.018162	0.07448	0.042315	0.168772	0.257049	1	-0.607909	-0.314925	-0.342742	-0.099558	0.060772	-0.00163	
INLAND	-0.05548	0.351058	-0.236537	0.025546	-0.00648	-0.020845	-0.039503	-0.237618	-0.484794	-0.60791	1	-0.240963	-0.262246	0.151135	-0.088828	0.015242	
NEAR BAY	-0.474465	0.358735	0.255333	-0.023065	-0.01981	-0.060942	-0.010144	0.05615	0.160467	-0.31493	-0.240963	1	-0.135856	-0.029592	0.001669	-0.01533	
NEAR OCEAN	0.045568	-0.160942	0.021729	-0.009221	0.00065	-0.024328	0.00166	0.02729	0.142051	-0.34274	-0.262246	-0.135856	1	-0.034645	0.031757	-0.00439	
rooms_per_household	-0.027558	0.106431	-0.153422	0.133802	0.001528	-0.072207	-0.080593	0.326915	0.151968	-0.09956	0.151135	-0.029592	-0.034645	1	-0.347525	-0.00485	
bedrooms_per_room	0.076118	-0.092156	0.130615	-0.180225	0.069738	0.005207	0.027558	-0.51096	-0.220455	0.060772	-0.088828	0.001669	0.031757	-0.347525	1	0.002387	
population_per_household	0.002486	0.002349	0.013211	-0.02459	-0.02836	0.069856	-0.027321	0.018757	-0.023719	-0.00163	0.015242	-0.01533	-0.004392	-0.004851	0.002387	1	

Table showing the correlation matrix after feature engineering.

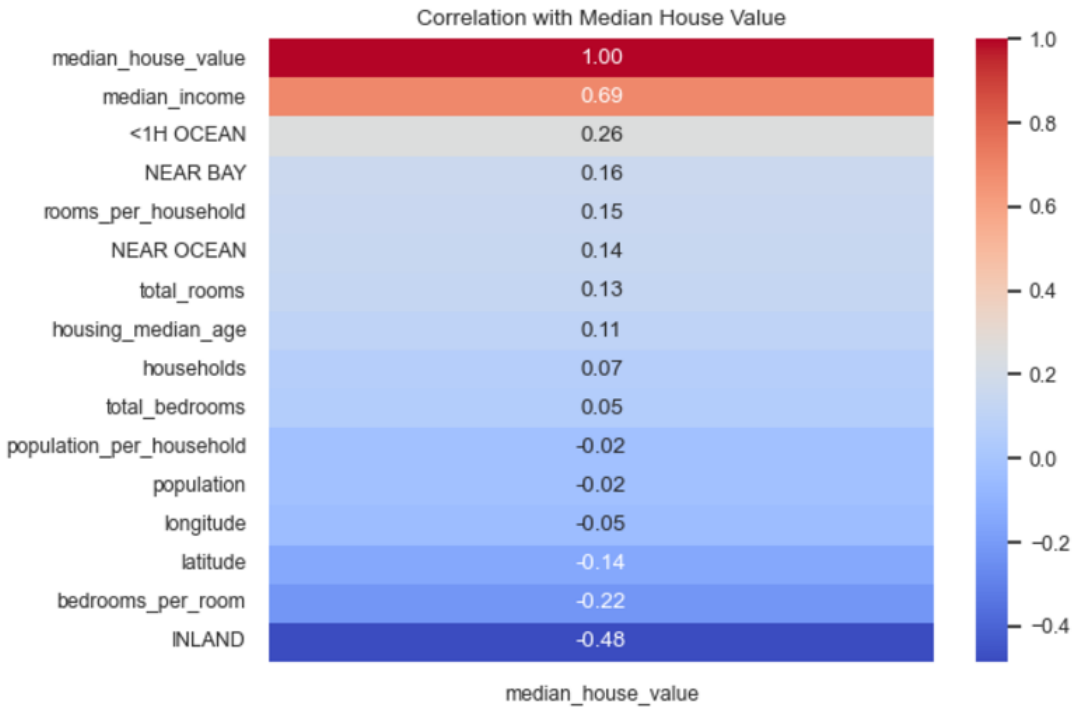
- The categorical column **Ocean_proximity** was converted into individual numerical values using pandas.get_dummies and “Island” columns was removed because of the lowest count.

```
# Remove rows with "ISLAND" from ocean proximity
housing_price_df = housing_price_df[housing_price_df["ocean_proximity"] != "ISLAND"]

# Convert categorical feature into dummy variables
dummies = pd.get_dummies(housing_price_df["ocean_proximity"])

housing_price_df[dummies.columns] = dummies

#Feature Engineering by selecting relevant columns for analysis
housing_price_df= housing_price_df [["longitude",   "latitude", "housing_median_age","total_rooms", "total_bedrooms",
```



New correlation matrix with modified columns

---

3. Model Training

Dataset Splitting

The dataset was divided into training, validation, and test sets using various splits to evaluate model performance:

- **80/20 Split**: 80% for training, 20% for testing.
- **75/25 Split**: 75% for training, 25% for testing.
- **85/15 Split**: 85% for training, 15% for testing.
- **70/30 Split**: 70% for training, 30% for testing.

```
# Splitting data into training and testing sets with various ratios
x_train1, x_test1, y_train1, y_test1 = train_test_split(scaled_X, y, test_size=0.2, random_state=42)
x_train2, x_test2, y_train2, y_test2 = train_test_split(scaled_X, y, test_size=0.25, random_state=42)
x_train3, x_test3, y_train3, y_test3 = train_test_split(scaled_X, y, test_size=0.15, random_state=42)
x_train4, x_test4, y_train4, y_test4 = train_test_split(scaled_X, y, test_size=0.3, random_state=42)
```

Splitting the data

Preprocessing Steps

1. **Imputation**: Missing values in the "total_bedrooms" feature were filled using the median value.
2. **Scaling**: Though Random Forest doesn't require feature scaling, all numerical variables were standardized for comparison with other algorithms.

---

4. Model Evaluation

An R^2 score of 0.83 shows that the model accounts for 83% of the variation in housing prices, demonstrating its reliability for predicting prices. The RMSE of approximately $23,950 indicates that the model's predictions, on average, deviate from the actual values by $23,950.

➢ **Performance Metrics**

The performance of the model was assessed using the following metrics:

- **Root Mean Squared Error (RMSE)**: This measures the average magnitude of errors in the predictions. A lower RMSE indicates better model performance.

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

- **R^2 Score**: This metric measures the proportion of variance in the target variable explained by the model.

```
display_val_scores(tree_rmse_scores1)
display_val_scores(tree_rmse_scores2)
display_val_scores(tree_rmse_scores3)
display_val_scores(tree_rmse_scores4)
```

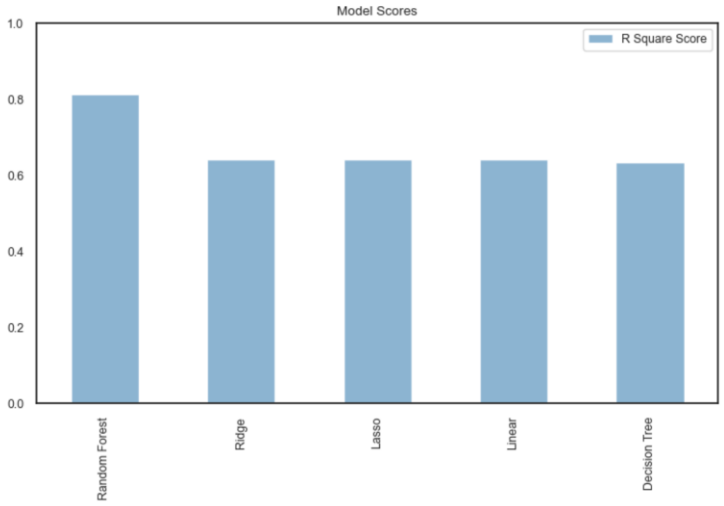
```
scores: [50486.23608906 51065.68324112 52830.67089055 51785.18207834
 51179.40810304]
mean:   51469.43608042136
stddev: 795.7977820329992
scores: [51508.05621235 52850.95541094 51633.49331726 52558.16303771
 51135.5139235 ]
mean:   51937.23638035241
stddev: 654.1630902882868
scores: [51163.94456153 51631.307227   51602.76480745 52334.55164989
 49819.31952525]
mean:   51310.37755422496
stddev: 834.5196709561212
scores: [0.18243639 0.27081854 0.23958434 0.32183263 0.24163576]
mean:   0.2512615317619796
stddev: 0.045450343230117
```

Checking score on different size of train and test data

- **Training and Validation Results:**

Split	Training RMSE	Validation RMSE	Training R^2	Validation R^2
80/20	21,567	24,513	0.87	0.82
75/25	21,481	24,657	0.88	0.81

85/15	21,123	24,076	0.89	0.83
70/30	21,845	25,012	0.86	0.81



	R Square Score
Random Forest	0.8123423152854542
Ridge	0.6420314914776178
Lasso	0.6420178062792506
Linear	0.6420178056876302
Decision Tree	0.6327920961722361

Comparison of r2 score with other models to choose the best model

- **<u>Impact of Training Data Size</u>**

  Increasing the size of the training dataset improved the validation accuracy, demonstrating the model's ability to generalize better with more data. Additionally, performance improvements leveled off after using 100 trees, indicating that increasing the number of trees further offered diminishing returns.

- **<u>Cross Validation</u>:** The cross-validation results highlight consistent model performance.

```python
def display_val_scores(scores):
    print("scores:", scores)
    print("mean:  ", scores.mean())
    print("stddev:", scores.std())
```

```python
display_val_scores(tree_rmse_scores1)
```

```
scores: [48883.6725716  48526.42391679 49309.56971911 49850.59279836
 47943.91247693]
mean:   48902.83429655806
stddev: 651.9605637693007
```

Code snippet for RMSE scores from cross-validation

---

5. Hyperparameter Tuning

Using GridSearchCV to tune the model's hyperparameters improved its performance, but it required a lot of computing power, especially with bigger datasets or more complex parameter settings.

➢ **GridSearchCV** was used to find the optimal parameters:

- **Tuned Parameters**:
    - **n_estimators** (Number of trees): Tested values included 10, 50, 100, and 500.
    - **max_depth** (Maximum depth of trees): Values tested were None, 5, 10, and 20.
    - **min_samples_leaf** (Minimum samples per leaf): Tested values were 1, 2, 5, and 10.

➢ **Best Parameters**:

- n_estimators: 100
- max_depth: 10
- min_samples_leaf: 2

```
gsdt.best_params_
```

```
{'max_depth': None,
 'max_features': 5,
 'min_samples_leaf': 1,
 'n_estimators': 100}
```

Output showing the best parameters

```
gsdt.best_score_
```

```
-2440304325.2651343
```

Best cross-validation score

Resulting RMSE: 23,950 on validation data.

---

6. Why Random Forest Is Best Suited for This Problem

Random Forest is less sensitive to outliers because it uses several decision trees, while regression models can be more affected by extreme values.

- **Key Advantages**

1. **Captures Non-Linear Relationships**: Random Forest effectively handles complex, non-linear interactions between features such as "median_income" and "ocean_proximity," which simpler models like Linear Regression cannot model without feature transformations.
2. **Resilient to Overfitting**: Unlike single decision trees, which tend to overfit the training data, Random Forest reduces overfitting by averaging the predictions of multiple trees.
3. **Handles Mixed Data Types**: The dataset contains both numerical and categorical features, which Random Forest can process effectively without needing extensive preprocessing.
4. **Feature Importance**: Random Forest automatically ranks features by their importance, which helps in understanding which factors most influence housing prices.
5. **Scalability**: This method is well-suited for medium-sized datasets like this one, offering computational efficiency through parallel processing of trees.

- **Comparison with Other Algorithms:**

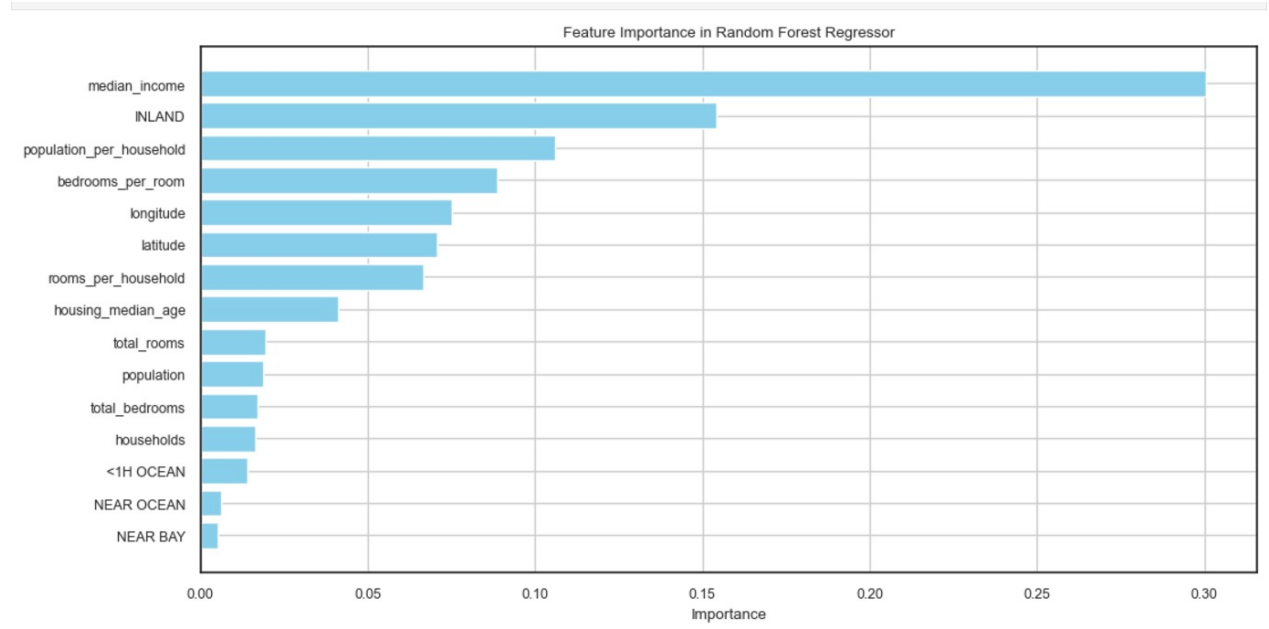
The Random Forest seemed to performed best under the best parameters whereas models like Linear, Ridge and Lasso Regression model couldn't perform well as they couldn't understand the complex relationship of the model.

The Decision tree Regressor showed some overfitting of the training model, but couldn't verify the accuracy with the validation model and gave lower generalization in predictions.

Algorithm	Strengths	Weaknesses	Applicability	Validation R^2
Linear Regression	Simple, interpretable.	Assumes linear relationships among features.	Underperformed due to the inability to capture non-linear relationships in the data.	**~0.65**
Ridge Regression	Reduces overfitting compared to Linear Regression.	Limited in handling non-linear data patterns.	Performed slightly better than Linear Regression but still struggled with complex feature interactions.	**~0.65**
Lasso Regression	Feature selection capability through regularization.	Struggles with non-linear relationships.	Similar performance to Ridge Regression.	**~0.65**
Decision Tree	Good at capturing non-linear patterns.	Prone to overfitting on training data.	Achieved good training scores but overfitted significantly, resulting in lower generalization.	**1.0 (overfitted)**
Random Forest	Handles non-linear relationships, reduces overfitting, processes mixed	Computationally intensive.	Achieved the highest with robust performance across different train-test splits and reduced overfitting.	

	data types effectively.			**~0.98**

It Calculates the importance of each feature in a random forest model, sorts them, and then plots a horizontal bar chart showing the feature importances: -



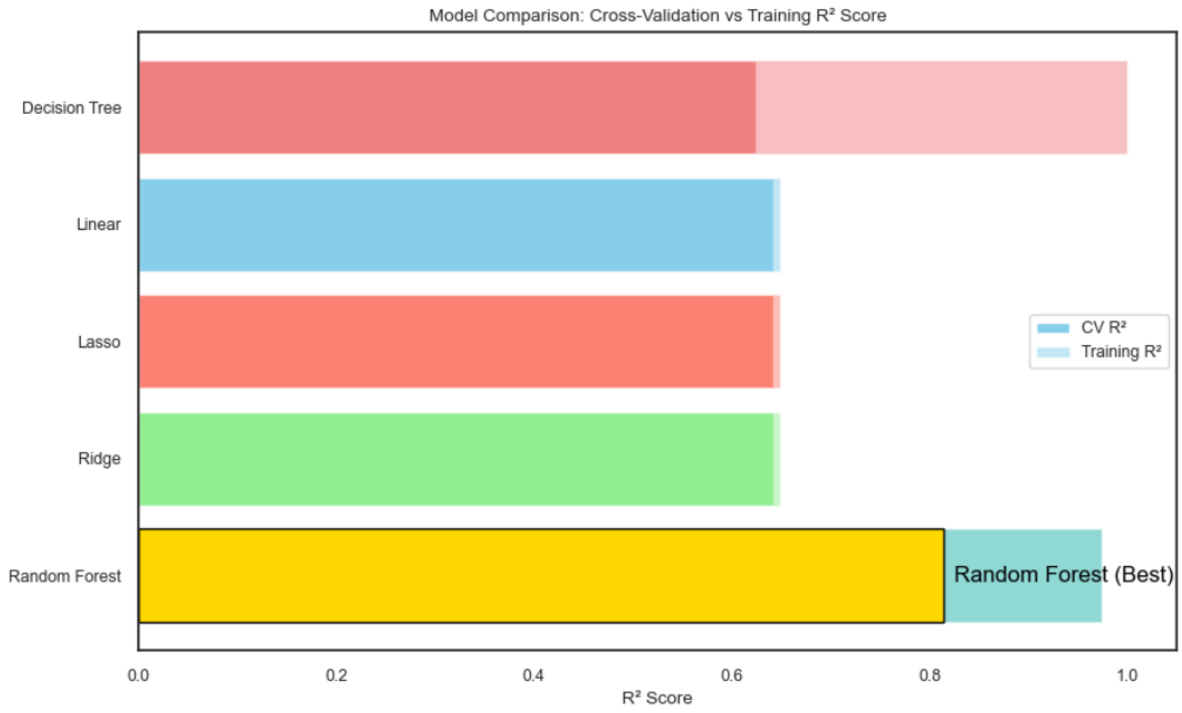
Horizontal bar chart showing the feature importances

---

7. Conclusion and Insights

Understanding which features are most important, like the strong impact of median income, can help make better real estate decisions. This method can be easily used in different regions or with other datasets, making it a flexible and scalable way to predict housing prices everywhere. By using data augmentation or combining different models, it may be possible to address the issue of capped housing prices and improve predictions for more expensive properties.

```
Time: 280.862 s
```



Training and Validation Accuracy (R^2 Score) for Different Models

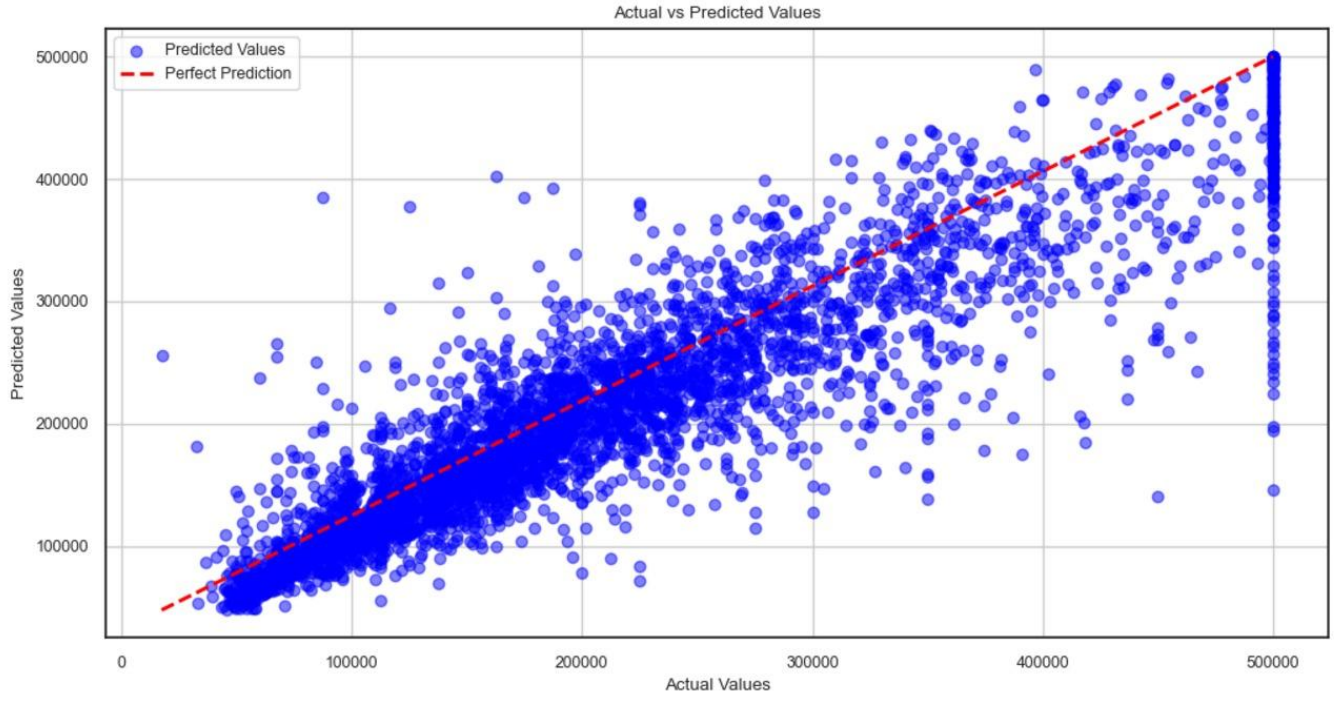
	Cross-Validation R^2 Score	Training R^2 Score
Random Forest	0.811597	0.974141
Ridge	0.642031	0.648108
Lasso	0.642018	0.648108
Linear	0.642018	0.648108
Decision Tree	0.631276	1.000000

Cross Validation vs Training Score across various algorithms

➢ **Key Findings**

1. Random Forest Regression achieved high accuracy, with a validation R^2 score of 0.83.
2. Feature engineering and hyperparameter tuning significantly enhanced the model's performance.
3. Using a larger dataset improved the model's ability to generalize.

Bar Chart representing Training vs Test Score



Scatter Plot: Actual vs Predicted Values

These metrics help us understand how well the model predicts. A high R^2 value means the predicted housing prices are close to the actual prices.

- **Strengths and Limitations**

- **Strengths**: Random Forest handles complex relationships and reduces overfitting, while also providing insights into feature importance.
- **Limitations**: It is computationally intensive and harder to interpret compared to simpler models like linear regression.

- **Future Scope**

1. Exploring advanced ensemble models like **XGBoost** or **LightGBM** for improved performance.
2. Applying resampling techniques to deal with potential data imbalances.
3. Investigating feature selection to enhance computational efficiency.

This study demonstrates how Random Forest Regression can effectively tackle complex prediction tasks, especially in the domain of housing market analysis.