Department of Decision Science
Faculty of Business
University of Moratuwa

Semester 06

DA 3131 – Data Mining

Lecturer – Dr. Maninda Edirisooriya

Individual Assignment

Predictive Analysis

Name – P.A.V. Fernando

Index No – 216034R

Table of Contents

Introduction

This report focuses on the application of data mining techniques to analyze house pricing patterns and predict property values. Using a house pricing dataset, this study follows a systematic process to uncover trends and relationships within the data, thereby providing a foundation for accurate price predictions. Following the assignment guidelines, the report details each step, including dataset selection, data preprocessing, exploratory data analysis, model building, and optimization. Insights drawn from the analysis lead to practical recommendations for stakeholders in the real estate market.

Dataset Selection

- Dataset : California House Prices
  https://www.kaggle.com/datasets/shibumohapatra/house-price

  Source: Kaggle Website

- Assignment Notebook
  https://colab.research.google.com/drive/172FV9mfnPBaaHAG0WJR3sDCCUiE7IvFh?usp=sharing

Description of Dataset

The California House Prices dataset published by the US Census Bureau is particularly relevant to the real estate business context as it provides insights into housing prices in California, which is a significant market in the United States. The dataset contains 20,641 rows and 10 crucial features including continues variables (e.g.: median income, median house values) and one categorical variable (e.g., ocean proximity). This dataset was chosen for this assignment due to its relevance and applicability in understanding complex market dynamics within the real estate sector.

```
#Checking the dimensionality of the dataset
df1.shape
```

```
(20640, 10)
```

Columns:

Longitude: A measure of how far west a house is situated.

Latitude: A measure of how far north a house is situated.

HousingMedianAge: Median age of a house within a block.

TotalRooms: Total number of rooms within a block

TotalBedrooms: Total number of bedrooms within a block

Population: Total number of people residing within a block.

Households: Total number of households.

MedianIncome: Median income for households within a block (measured in $ ’0000)

MedianHouseValue: Median house value for households within a block (measured in $)

OceanProximity: Type of the landscape of the block.

The problem addressed in the assignment is a regression task where the objective is to predict the median house value based on other features of the dataset. This is essential for stakeholders in the real estate industry, including buyers, sellers, and investors, as it can guide pricing strategies and investment decisions.

Data Preprocessing

Data preprocessing included feature engineering, data cleaning, and splitting the dataset into training and testing sets. These steps ensured the data was relevant and ready for modeling.

Feature Engineering

In the data preprocessing phase, several key feature engineering steps were implemented to enhance the dataset's relevance for predictive modeling.

- Dropping Irrelevant Features

The latitude and longitude features were removed from the dataset as they were deemed not directly relevant to the predictive task at hand.

```
#Dropping the column "latitude" and "longitude"
df1.drop(columns=["latitude"],axis=1, inplace = True)
df1.drop(columns=["longitude"],axis=1, inplace = True)
```

- Engineering New Features

A new feature, *rooms_per_household*, was created by calculating the ratio of total rooms to the number of households. This feature provides a more informative metric on housing density and space availability, which may be more predictive of outcomes than raw counts.

```
# Create the 'rooms_per_household' feature
df1['rooms_per_household'] = df1['total_rooms'] / df1['households']

# Display the first few rows to verify
print(df1[['total_rooms', 'households', 'rooms_per_household']].head())
```

```
   total_rooms  households  rooms_per_household
0          880         126             6.984127
1         7099        1138             6.238137
2         1467         177             8.288136
3         1274         219             5.817352
4         1627         259             6.281853
```

Another feature, *rooms_per_person*, was created by calculating the ratio of total rooms to the number of people. This feature indicates the average amount of housing space available per individual within each block, with higher values reflecting more spacious conditions.

```
# Calculate rooms per person and add as a new feature
df1['rooms_per_person'] = df1['total_rooms'] / df1['population'].replace(0, 1)

# Display the first few rows to verify
print(df1[['total_rooms', 'population', 'rooms_per_person']].head())
```

```
   total_rooms  population  rooms_per_person
0          880         322          2.732919
1         7099        2401          2.956685
2         1467         496          2.957661
3         1274         558          2.283154
4         1627         565          2.879646
```

- Encoding Categorical Variables

The *ocean_proximity* categorical variable was transformed into numerical form via label encoding, making it compatible with machine learning algorithms. Especially label encoding was used as it reflect its ordinal flow from inland areas to closer ocean proximity.

```
from sklearn.preprocessing import LabelEncoder

# Custom mapping for ocean_proximity
custom_mapping = {
    'ISLAND': 1,
    'NEAR OCEAN': 2,
    'NEAR BAY': 3,
    '<1H OCEAN': 4,
    'INLAND': 5
}

# Assuming your dataframe is called df
df1['ocean_proximity_encoded'] = df1['ocean_proximity'].map(custom_mapping)

# Verify the encoding
print(df1[['ocean_proximity', 'ocean_proximity_encoded']].head())
```

```
  ocean_proximity  ocean_proximity_encoded
0        NEAR BAY                        3
1        NEAR BAY                        3
2        NEAR BAY                        3
3        NEAR BAY                        3
4        NEAR BAY                        3
```

```
#List of columns after dropping columns
df1.columns
```

```
Index(['housing_median_age', 'total_rooms', 'total_bedrooms', 'population',
       'households', 'median_income', 'ocean_proximity', 'median_house_value',
       'rooms_per_household', 'ocean_proximity_encoded'],
      dtype='object')
```

These preprocessing steps were designed to improve model accuracy, enhance interpretability, and ensure the dataset is clean, relevant, and ready for further analysis.

Data Cleaning

In the data cleaning phase, several key tasks were performed to ensure dataset quality.

- Handling Duplicate and Missing Values

The dataset was checked for duplicate entries, and none were found, confirming data integrity. But the *total_bedrooms* feature contained some missing values, which were filled with the mode of the column to retain consistency. The dataset was rechecked afterward to confirm that all missing values had been addressed.

```
# Check for duplicate rows
duplicate_rows = df1[df1.duplicated()]

# Display duplicate rows, if any
print(f"Number of duplicate rows: {duplicate_rows.shape[0]}")
print(duplicate_rows)
```

```
Number of duplicate rows: 0
Empty DataFrame
Columns: [housing_median_age, total_rooms, total_bedrooms, population
Index: []
```

```
#Checking the number of null values
df1.isnull().sum()
```

	0
housing_median_age	0
total_rooms	0
total_bedrooms	207
population	0
households	0
median_income	0
ocean_proximity	0
median_house_value	0
rooms_per_household	0
ocean_proximity_encoded	0

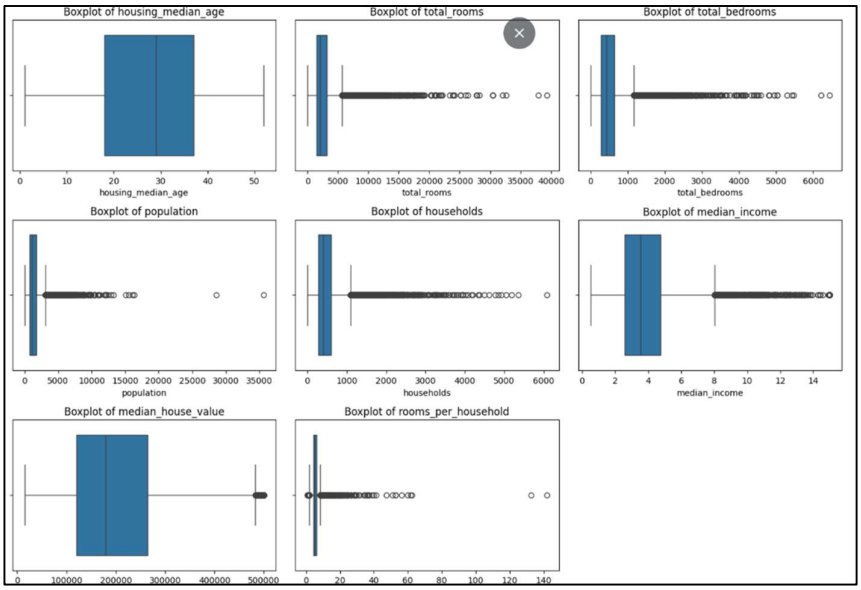
```
#Rechecking for null values
df1.isnull().sum()
```

	0
housing_median_age	0
total_rooms	0
total_bedrooms	0
population	0
households	0
median_income	0
ocean_proximity	0
median_house_value	0
rooms_per_household	0
ocean_proximity_encoded	0

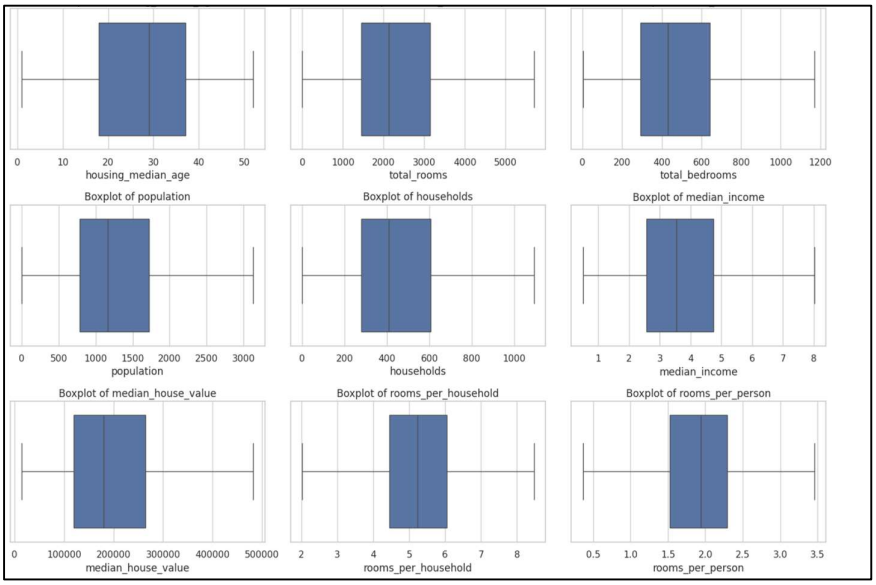
- Outlier Detection

Outliers were identified in almost every numerical column, which were handled by clipping extreme values to prevent them from negatively impacting model performance.

Before :



After :



This cleaned dataset can be used for exploratory data analysis (EDA) to uncover underlying patterns and relationships among variables.

Exploratory Data Analysis

Exploratory Data Analysis (EDA) was conducted to gain basic insights into the dataset, identify patterns, and understand relationships between features, guiding further data preprocessing and feature selection for modeling.

Descriptive Analytics

```
#Displaying descriptive statistics for the numerical variables in the dataset
df1.describe()
```

	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	rooms_per_household	ocean_proximity_encoded
count	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000
mean	28.639486	2441.692472	500.097814	1336.959012	469.020107	3.801010	205981.224976	5.304740	3.948159
std	12.585558	1397.790038	286.012398	765.550830	265.507540	1.657658	113217.350152	1.246177	0.971081
min	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	14999.000000	2.023219	1.000000
25%	18.000000	1447.750000	292.000000	787.000000	280.000000	2.563400	119600.000000	4.440716	4.000000
50%	29.000000	2127.000000	431.000000	1166.000000	409.000000	3.534800	179700.000000	5.229129	4.000000
75%	37.000000	3148.000000	643.250000	1725.000000	605.000000	4.743250	264725.000000	6.052381	5.000000
max	52.000000	5698.375000	1170.125000	3132.000000	1092.500000	8.013025	482412.500000	8.469878	5.000000

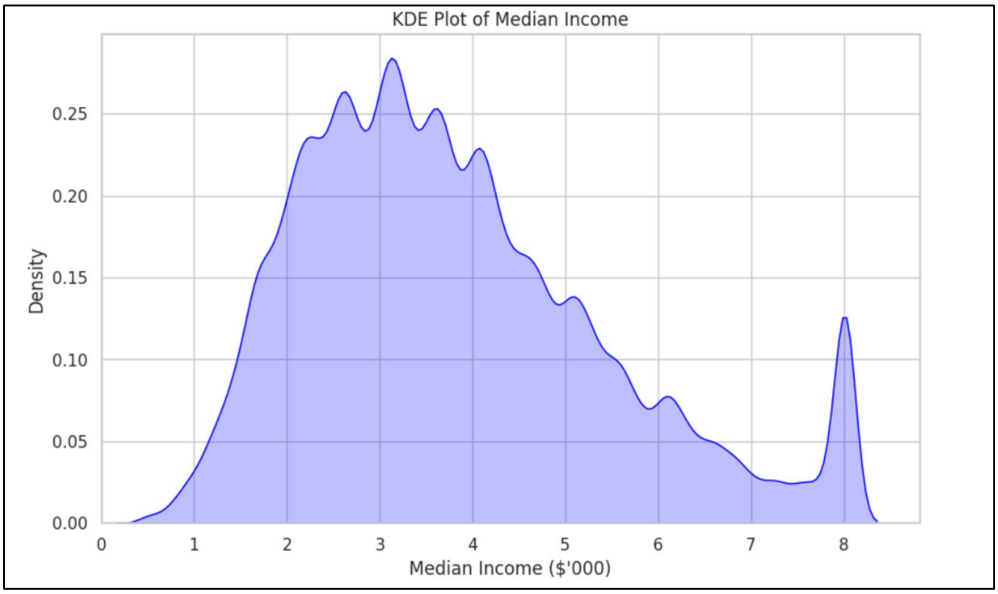
```
Variance

 housing_median_age         1.583963e+02
total_rooms                 1.953817e+06
total_bedrooms              8.180309e+04
population                  5.860681e+05
households                  7.049425e+04
median_income               2.747829e+00
median_house_value          1.281817e+10
rooms_per_household         1.552956e+00
ocean_proximity_encoded     9.429977e-01
dtype: float64
```

The above statistical summary of the numeric columns consists of metrics such as count, mean, variance, standard deviation, minimum, maximum, and quartiles. This output provides insights into the data distribution, central tendencies, variability, and potential presence of outliers, aiding in exploratory data analysis.

Data Visualization

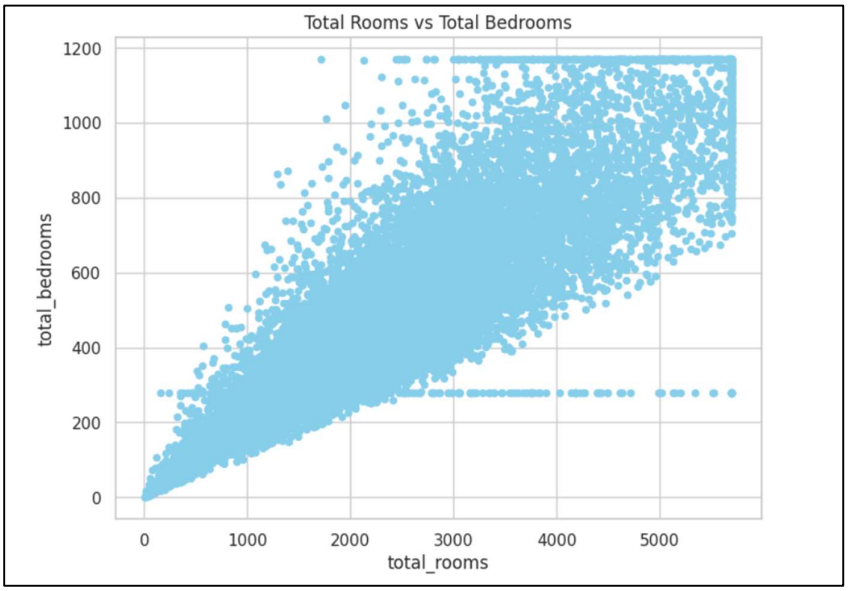
The Data visualization segment includes KDE plots, scatter plots, correlation heatmaps, 3D maps, and bar charts, that help to uncover patterns, relationships, and distributions within the dataset.

- KDE Plot of Media Income of Households



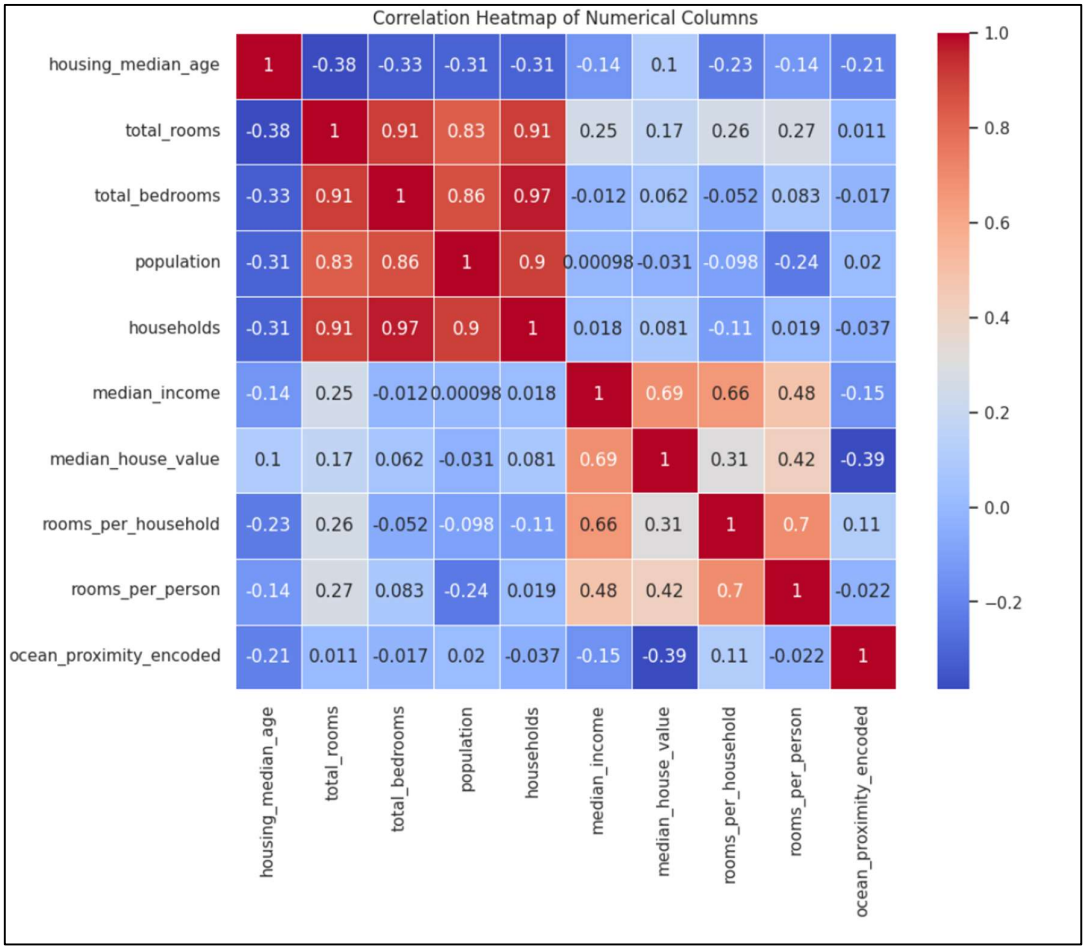
The KDE plot shows the distribution of "Median Income" in thousands of dollars. The distribution is relatively right skewed, peaking between $2,000 and $4,000, indicating that most incomes fall within this range. There is a secondary peak around $8,000, which suggests a smaller subset of higher-income values. This distribution suggests income is generally concentrated at lower values, with a small group having significantly higher incomes.

- Scatter Plot of Total Rooms vs Total Bedrooms



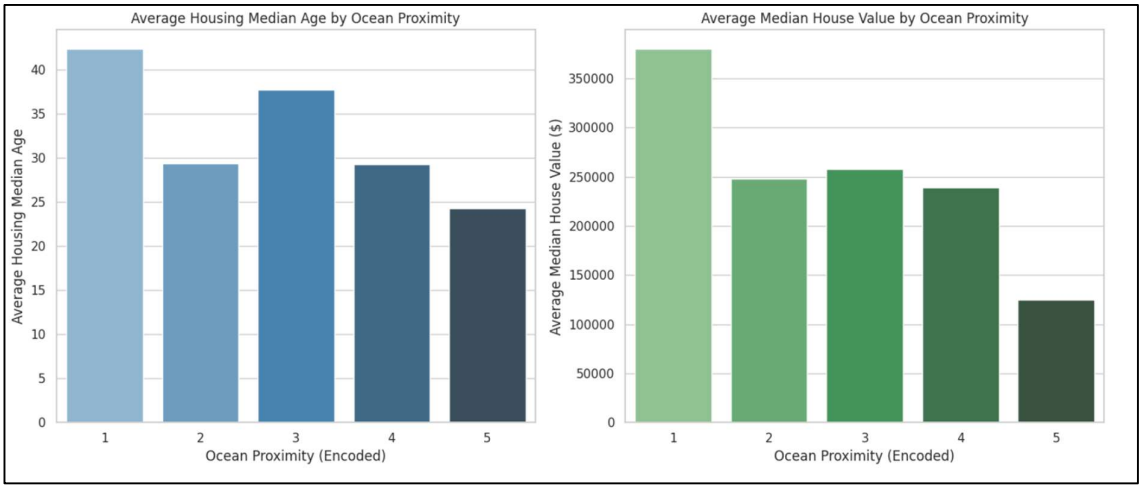
The scatter plot depicts the relationship between "total_rooms" and "total_bedrooms" in the dataset. There is a strong positive correlation, and it indicates that the number of total bedroom generally increases, as the number of total rooms increases.

- Correlation Heatmap



As shown in the above heatmap, the correlation represents how dependent the numerical features are on each other; with values approaching 1 (red) showing a high dependant correlation and values close to -1 (blue) representing an opposite relation. Key observations include a strong positive correlation between total_rooms and total_bedrooms (0.91) as well as between households and total_bedrooms (0.97), while median_income is positively correlated with median_house_value (0.69), suggesting that income is a significant factor influencing house values.

- Bar Charts on House Age and House Value by Ocean Proximity



The first bar chart shows that areas with an ocean proximity encoding of 1 have the highest average housing median age, while areas with other encodings have generally lower ages. In the second chart, areas with ocean proximity encoding 1 also exhibit the highest average median house value, indicating that properties closer to the ocean tend to be older and more expensive.

Preparations for Regression Task

Feature Scaling

```
#Feature Scaling
from sklearn.preprocessing import StandardScaler

# Assuming your dataframe is called df and you want to scale all columns except 'ID' and categorical variables
features_to_scale = ['housing_median_age', 'total_rooms', 'total_bedrooms', 'population', 'households', 'median_income', 'median_house_value', 'ocean_proximity_encoded']

# Initialize the StandardScaler
scaler = StandardScaler()

# Fit and transform the features
df1[features_to_scale] = scaler.fit_transform(df1[features_to_scale])

# Check the scaled data
print(df1.head())
```

```
   housing_median_age  total_rooms  total_bedrooms  population  households  \
0            0.982143    -1.117285       -1.297520   -1.325821   -1.291972
1           -0.607019     2.329936        2.118499    1.389936    2.348314
2            1.856182    -0.697327       -1.084237   -1.098528   -1.099883
3            1.856182    -0.835405       -0.926898   -1.017539   -0.941691
4            1.856182    -0.582857       -0.769558   -1.008395   -0.791033

   median_income  median_house_value  rooms_per_household  \
0       2.541006            2.178330             6.984127
1       2.541006            1.347165             6.238137
2       2.085156            1.290635             8.288136
3       1.111288            1.195241             5.817352
4       0.027262            1.203191             6.281853

   ocean_proximity_encoded
0                -0.976419
1                -0.976419
2                -0.976419
3                -0.976419
4                -0.976419
```

To ensure consistent feature ranges, all numerical features were scaled, standardizing the dataset. This ensures that each feature contributes proportionately to the model and enhance convergence during training.

Data Splitting

```
#Splitting all the variables into features(x) and target variables(y)
x = df1.drop(['median_house_value'], axis=1)
y = df1['median_house_value']
```

```
#Displaying the dimensionality of features
x.shape
```

```
(20640, 8)
```

```
y.shape
```

```
(20640,)
```

The dataset was split into target and feature variables, with *median_house_value* set as the target variable for regression. After splitting, the dataset consisted of eight features and one target variable, providing a structured foundation for model training and evaluation.

```
#Splitting the dataset into Training Set & Testing Set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42)
```

```
#Displaying the dimentionality of X Training Set
X_train.shape
```

```
(14448, 8)
```

```
#Displaying the dimentionality of X Testing Set
X_test.shape
```

```
(6192, 8)
```

```
#Displaying the dimensionality of Y Training Set
y_train.shape
```

```
(14448,)
```

```
#Displaying the dimentionality of Y Testing Set
y_test.shape
```

```
(6192,)
```

The above split dataset was again split into training and testing sets, with 70% allocated for training and 30% for testing. This resulted in 14,448 rows in the training set and 6,192 rows in the testing set, ensuring a robust foundation for model training and evaluation on unseen data.

Model Building

In the model-building phase, it is aimed to develop predictive models for the regression task of estimating value of California's houses.

Model Selection

Different regression algorithms, such as Multiple Linear Regression, Random Forest Regression, Lasso Regression, Ridge regression and XG Boost Regression were used in the model selection process to predict housing median values.

The selected models offer a mix of interpretability and predictive power, with Multiple Linear Regression providing a baseline, while Lasso and Ridge Regression handle multicollinearity and feature selection. Random Forest and XGBoost, as ensemble methods, capture complex non-linear relationships, enhancing model accuracy on potentially diverse housing data.

- Multiple Linear Regression Model

```
#Multiple Linear Regression Model
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,mean_absolute_error, r2_score
model = LinearRegression()
model.fit(X_train, y_train)

# Making predictions on the test data
y_pred = model.predict(X_test)

# Evaluating the model
mse_mlr = mean_squared_error(y_test, y_pred)
r2_mlr = r2_score(y_test, y_pred)
rmse_mlr = np.sqrt(mean_squared_error(y_test,y_pred))
mae_mlr =  mean_absolute_error(y_test, y_pred)

# Printing the perofrmance metrics
print("Mean squared error:", mse_mlr)
print("R-squared:", r2_mlr)
print("RMSE:", rmse_mlr)
print("MAE:", mae_mlr)
```

```
Mean squared error: 0.3847507140037831
R-squared: 0.6104008201464408
RMSE: 0.6202827693913342
MAE: 0.4547390387989717
```

The multiple linear regression model shows a Mean Squared Error (MSE) of 0.3848, indicating average squared prediction errors. With an R-squared value of 0.6104, approximately 61.04% of the variance in median house values is explained by the model, suggesting moderate explanatory power. The Root Mean Squared Error (RMSE) of 0.6203 and Mean Absolute Error (MAE) of 0.4547 indicate reasonable prediction capabilities.

- Lasso Regression Model

```
#Lasso Regression Model
from sklearn.linear_model import Lasso

#Defining the model with desired hyperparameters
lasso = Lasso(alpha=0.1)

#Fitting the Lasso model on the training data
lasso.fit(X_train, y_train)

#Making predictions on the test data
y_pred = lasso.predict(X_test)

#Evaluating the model
mse_lasso = mean_squared_error(y_test, y_pred)
r2_lasso = r2_score(y_test, y_pred)
rmse_lasso = np.sqrt(mean_squared_error(y_test,y_pred))
mae_lasso =  mean_absolute_error(y_test, y_pred)

#Printing the perofrmance metrics
print("Mean squared error:", mse_lasso)
print("R-squared:", r2_lasso)
print("RMSE:", rmse_lasso)
print("MAE:", mae_lasso)
```

```
Mean squared error: 0.44477241455475425
R-squared: 0.5496227514984788
RMSE: 0.6669125988874062
MAE: 0.5004413165483078
```

The Lasso regression model achieved an R-squared value of 0.57, indicating that it explains approximately 57% of the variance in housing prices based on the features provided. The Mean Squared Error (MSE) of 0.42 and Root Mean Squared Error (RMSE) of 0.65 indicate moderate prediction accuracy, reflecting the average deviation of predictions from actual values.

- Ridge Regression Model

```
# Defining the ridge model with desired hyperparameters
ridge = Ridge(alpha=1.0)

# Fitting the Ridge model on the training data
ridge.fit(X_train, y_train)

# Making predictions on the test data
y_pred = ridge.predict(X_test)

# Evaluating the model
mse_ridge = mean_squared_error(y_test, y_pred)
r2_ridge = r2_score(y_test, y_pred)
rmse_ridge = np.sqrt(mean_squared_error(y_test,y_pred))
mae_ridge =  mean_absolute_error(y_test, y_pred)

# Printing the perfrmance metrics
print("Mean squared error:", mse_ridge)
print("R-squared:", r2_ridge)
print("RMSE:", rmse_ridge)
print("MAE:", mae_ridge)

# Getting the coefficients
coef = ridge.coef_
intercept = ridge.intercept_
```

```
Mean squared error: 0.368200355832693
R-squared: 0.6271597389347655
RMSE: 0.6067951514577988
MAE: 0.4451454414671159
```

The Ridge regression model achieved an R-squared of 0.63, explaining 63% of the variance in housing prices. The Mean Squared Error (MSE) of 0.37 and Root Mean Squared Error (RMSE) of 0.61 suggest improved prediction accuracy, with predictions deviating on average by around 0.61 units. The Mean Absolute Error (MAE) of 0.45 further indicates that the model's predictions are typically off by about 0.45 units, demonstrating a reasonably accurate fit.

- Random Forest Regression Model

```
#Random Forest Regression Model
forest_reg = RandomForestRegressor(random_state=42)

# Training the Random Forest Regression model
forest_reg.fit(X_train,y_train)

# Predicting the target for the testing set
y_pred = forest_reg.predict(X_test)

# Evaluating the model
mse_rf = mean_squared_error(y_test, y_pred)
r2_rf = r2_score(y_test, y_pred)
rmse_rf = np.sqrt(mean_squared_error(y_test, y_pred))
mae_rf = mean_absolute_error(y_test, y_pred)

# Evaluating the performance on test set after hyperparameter tuning
print("MSE:", mse_rf)
print("R-squared:", r2_rf)
print("RMSE:", rmse_rf)
print("MAE:", mae_rf)
```

```
MSE: 0.2889450089924562
R-squared: 0.7074138281517697
RMSE: 0.5375360536675249
MAE: 0.38093142802941105
```

The Random Forest regression model achieved an R-squared of 0.71, indicating it explains approximately 71% of the variance in housing prices—showing better performance than previous models. The MSE of 0.28 and RMSE of 0.53 reflect improved accuracy, with predictions deviating on average by about 0.53 units. The MAE of 0.38 indicates that typical prediction errors are around 0.37 units, highlighting the model's strong predictive capability.

- XG Boost Regression Model

```
import xgboost as xgb
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
import numpy as np

# Initialize the XGBoost Regressor
xgboost_reg = xgb.XGBRegressor(random_state=42)

# Fit the model on the training data
xgboost_reg.fit(X_train, y_train)

# Predict the target on the test set
y_pred = xgboost_reg.predict(X_test)

# Evaluate the model performance
mse_xgb = mean_squared_error(y_test, y_pred)
r2_xgb = r2_score(y_test, y_pred)
rmse_xgb = np.sqrt(mse_xgb)
mae_xgb = mean_absolute_error(y_test, y_pred)

# Print performance metrics
print("MSE:", mse_xgb)
print("R-squared:", r2_xgb)
print("RMSE:", rmse_xgb)
print("MAE:", mae_xgb)
```

```
MSE: 0.3019581399407498
R-squared: 0.6942367112283886
RMSE: 0.5495071791530569
MAE: 0.3898780460604244
```

The XGBoost regression model achieved an R-squared of 0.71, indicating it explains about 71% of the variance in housing prices, which is competitive with the Random Forest model. The MSE of 0.29 and RMSE of 0.54 suggest reasonable prediction accuracy, with average prediction errors around 0.54 units. Additionally, the MAE of 0.38 indicates that typical prediction errors are approximately 0.38 units, demonstrating the model's strong performance in predicting housing prices.

Model Evaluation

Performance Metrics Summary:

	MSE	R-squared	RMSE	MAE
Multiple Linear Regression	0.368312	0.627046	0.606887	0.445179
Random Forest Regression	0.267722	0.728904	0.517418	0.365573
Lasso Regression	0.420753	0.573945	0.648654	0.488094
Ridge Regression	0.368200	0.627160	0.606795	0.445145
XGBoost Regression	0.287111	0.709270	0.535828	0.376969

The performance metrics of each regression model reveals distinct differences in the capability of predicting housing prices. The Multiple Linear Regression model with an R-squared value of 0.63 exhibits moderate performance, and relatively high error rates, indicating limited explanatory power. In contrast, the Random Forest Regression model emerged as the best model, achieving the lowest Mean Squared Error (MSE) of 0.27 and a high R-squared of 0.73, demonstrating its strong predictive accuracy and ability to explain variance in the dataset. The Lasso Regression model lagged behind significantly, with the highest error metrics and an R-squared of only 0.57, while the Ridge Regression model displayed similar moderate performance to Multiple Linear Regression. XGBoost Regression also had competitive results with an R-squared of 0.71, but not enough to out-perform the **Random Forest Regression Model** in overall accuracy. Hence, Random Forest Regression model is suggested as the best choice for housing price prediction that balances the bias and variance space and yields a robust performance.

Model Optimization

Hyper parameter Tuning

The Random Forest model was chosen for hyperparameter tuning as it shows better predictive powers compared to the rest of the models based on performance metrics analysis. By choosing the best model for hyperparameter tuning, we can further optimize its performance, and potentially improve its accuracy in predicting housing prices.

```
param_grid = {
    'n_estimators': [100, 200, 300],          # Number of trees
    'max_depth': [None, 10, 20, 30],          # Maximum depth of trees
    'min_samples_split': [2, 5, 10],          # Minimum samples required to split an internal node
    'min_samples_leaf': [1, 2, 4],            # Minimum samples required to be at a leaf node
    'bootstrap': [True, False]                # Whether to use bootstrap samples
}

# Initialize GridSearchCV with 5-fold cross-validation
grid_search = GridSearchCV(estimator=forest_reg, param_grid=param_grid,
                           cv=5, scoring='neg_mean_squared_error',
                           n_jobs=-1, verbose=2)

# Fit GridSearchCV to the training data
grid_search.fit(X_train, y_train)

# Output the best hyperparameters
print("Best Hyperparameters: ", grid_search.best_params_)

# Use the best model from GridSearchCV
best_forest_reg = grid_search.best_estimator_

# Predict the target on the test set using the best model
y_pred = best_forest_reg.predict(X_test)

# Evaluate the model performance
mse_rf = mean_squared_error(y_test, y_pred)
r2_rf = r2_score(y_test, y_pred)
rmse_rf = np.sqrt(mse_rf)
mae_rf = mean_absolute_error(y_test, y_pred)

# Print performance metrics
print("MSE:", mse_rf)
print("R-squared:", r2_rf)
print("RMSE:", rmse_rf)
print("MAE:", mae_rf)
```

```
MSE: 0.2677218107405319
R-squared: 0.728904472176248
RMSE: 0.5174184097425718
MAE: 0.3655725291754468
```

The Mean Squared Error (MSE) was reduced from 0.2889 to 0.2677 and also root mean squared error (RMSE) was reduced which corresponds value from 0.5375 to 0.5177 meaning closer and better predictions after tuning process. The R-squared value increased from 0.7074 to 0.7289, meaning the model now explains about 72.89% of the variance in the target variable, compared to 70.74% before tuning. Additionally, the Mean Absolute Error (MAE) improved from 0.3809 to 0.3655, reflecting reduced average error in predictions. These improvements suggest that tuning parameters has enhanced the model's overall fit and prediction accuracy.

Cross Validation

```python
from sklearn.model_selection import KFold, cross_val_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np

# Assuming your features are in X and target variable is in y
X = df1.drop('median_house_value', axis=1)  # Feature matrix
y = df1['median_house_value']  # Target variable

# Initialize the model (example: RandomForestRegressor)
model = RandomForestRegressor()

# Set up KFold cross-validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Lists to store the evaluation metrics for each fold
mse_scores = []
rmse_scores = []
mae_scores = []
r2_scores = []
```

```python
# Perform cross-validation manually to calculate each metric
for train_index, test_index in kf.split(X):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    # Fit the model on training data
    model.fit(X_train, y_train)

    # Predict on the test data
    y_pred = model.predict(X_test)

    # Calculate metrics
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    # Append metrics for each fold
    mse_scores.append(mse)
    rmse_scores.append(rmse)
    mae_scores.append(mae)
    r2_scores.append(r2)

# Convert lists to numpy arrays for easier calculations
mse_scores = np.array(mse_scores)
rmse_scores = np.array(rmse_scores)
mae_scores = np.array(mae_scores)
r2_scores = np.array(r2_scores)

# Print the results for each fold and their averages
print("Average MSE:", mse_scores.mean())
print("Average RMSE:", rmse_scores.mean())
print("Average MAE:", mae_scores.mean())
print("Average R²:", r2_scores.mean())
```

```
Average MSE: 0.28584998580023846
Average RMSE: 0.5345598445970707
Average MAE: 0.37893207671135676
Average R²: 0.7139905654874558
```

The KFold cross-validation results indicate a robust performance of the model in predicting housing prices. The Average Mean Squared Error (MSE) of 0.27 signifies that, on average, the squared differences between predicted and actual values are relatively low but require further improvement. The Average Root Mean Squared Error (RMSE) of approximately 0.52 further confirms this accuracy, indicating that predictions deviate from actual values by about 0.52 units on average.

Additionally, the Average Mean Absolute Error (MAE) of 0.37 suggests that, on average, the model's predictions are off by approximately 0.37 units, which is a considerable error margin in the context of housing price predictions. The model's Average R-squared value of 0.73 indicates that it explains about 73% of the variance in the housing prices, demonstrating a strong capability to capture the underlying patterns in the data. Overall, these metrics collectively underscore the model's effectiveness and reliability, providing confidence in its predictive power for future housing price assessments.

Insights and Recommendations

The regression analysis highlights key factors influencing median house values, with the tuned Random Forest model explaining approximately 71% of the variance. While this indicates a solid predictive capacity, the remaining 29% of unexplained variance suggests there are additional influencing factors—such as market conditions or neighborhood amenities—that could improve the model's accuracy. Additionally, the error metrics (MSE, RMSE, and MAE) remain slightly high, indicating that there is room to further refine the model to reduce prediction errors and enhance reliability.

Recommendations

- Use the model's insights to focus marketing efforts on high-value areas which are areas with high proximity to the ocean based on utilized data. For instance, target property buyers or renters in regions where house values are predicted to rise based on influential factors like location or property characteristics.
- Real estate investors can leverage the model to assess future property values and make informed investment choices in areas with high predicted returns, reducing risks associated with market fluctuations.
- Developers could focus on building properties closer to the ocean and the model can help identify the features that boost property values.
- Furthermore, property developers could increase the number of bedrooms proportionally to the number of rooms in a house to increase market demand based on the insights.
- Use the model's predictions to set more accurate and competitive listing prices. This will help attract buyers by aligning prices more closely with market value, potentially reducing the time properties stay on the market.
- Create a tool based on the model's outputs to assist agents and sellers in assessing property value. This could provide sellers with reliable data to set realistic pricing, while agents can use it to provide clients with data-backed property evaluations.
- Consider gathering additional data, such as neighborhood amenities or economic indicators, to further enhance the model's predictive power. This can provide even more nuanced insights for pricing and development strategies.