ML Models for CA Housing Prices

Introduction

Housing prices are influenced by many factors such as income levels, population density, location, and the age of homes in an area. In this project, the goal is to use machine learning to model housing prices in California and compare how different models perform on the same dataset.

The data for this project comes from the California Housing dataset based on the 1990 U.S. Census. Each row represents a census block group, which is a small geographic area containing several hundred households, rather than an individual home. The dataset includes information such as total rooms, total bedrooms, population, number of households, median income, geographic coordinates, and proximity to the ocean, which is an important factor in housing prices.

This project begins with exploring and cleaning the data, followed by feature engineering to capture useful patterns related to housing density and location. I then trained multiple machine learning models to compare their performance, including linear models, tree-based models, and XGBoost. The focus of the project is not only on accuracy, but also on understanding which features matter most and how different models behave. By comparing results across models, this project demonstrates how machine learning can be used to analyze housing price patterns at the district level in California.

Dataset Overview

Each row represents a single California district, which is a small geographic area containing several hundred households. The dataset summarizes information about the houses and the people living in each district, and the goal is to predict the median house value for that district.

The dataset includes the following features:

- **longitude** and **latitude**: The geographic location of the district.
- **housing_median_age**: The median age of the homes in the district.
- **total_rooms** and **total_bedrooms**: The total number of rooms and bedrooms across all homes in the district.
- **population**: The total number of people living in the district.
- **households**: The number of household units in the district.
- **median_income**: The median household income, measured in tens of thousands of dollars.

- **ocean_proximity**: A categorical feature describing how close the district is to the coast.

The target variable is **median_house_value**, which represents the median value of the homes in each district. Since many of the original features are counts for the entire district, I later add features such as **rooms_per_household**, **bedrooms_per_room**, and **population_per_household** to better capture density and living conditions within each area.

This dataset provides a clear structure for exploring how location, income, and housing characteristics relate to median house values.

Import the data

In [41]:
```python
import pandas as pd

data = pd.read_csv('./housing.csv')
data.head()
```

Out[41]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population
0	-122.23	37.88	41.0	880.0	129.0	322.0
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0
2	-122.24	37.85	52.0	1467.0	190.0	496.0
3	-122.25	37.85	52.0	1274.0	235.0	558.0
4	-122.25	37.85	52.0	1627.0	280.0	565.0

Exploratory Data Analysis (EDA)

In [42]:
```python
data.describe() #get summary statistics
```

Out[42]:

	longitude	latitude	housing_median_age	total_rooms	total_bedro
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.00(
mean	-119.569704	35.631861	28.639486	2635.763081	537.87(
std	2.003532	2.135952	12.585558	2181.615252	421.38!
min	-124.350000	32.540000	1.000000	2.000000	1.00(
25%	-121.800000	33.930000	18.000000	1447.750000	296.00(
50%	-118.490000	34.260000	29.000000	2127.000000	435.00(
75%	-118.010000	37.710000	37.000000	3148.000000	647.00(
max	-114.310000	41.950000	52.000000	39320.000000	6445.00(

In [43]:
```
data.isnull().sum() #simply check for nulls
```

Out[43]:
```
longitude               0
latitude                0
housing_median_age      0
total_rooms             0
total_bedrooms        207
population              0
households              0
median_income           0
median_house_value      0
ocean_proximity         0
dtype: int64
```

In [44]:
```
data.dropna(subset=['total_bedrooms'], inplace=True)
data.isnull().sum() #to verify no more nulls in total_bedrooms
```

Out[44]:
```
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
ocean_proximity       0
dtype: int64
```

In [45]:
```
data['ocean_proximity'].value_counts()
```

```
Out[45]: ocean_proximity
<1H OCEAN     9034
INLAND        6496
NEAR OCEAN    2628
NEAR BAY      2270
ISLAND           5
Name: count, dtype: int64
```

```
In [46]: # One hot encoding
data = pd.get_dummies(data, columns=['ocean_proximity'], prefix='ocean')
```

```
In [47]: data.head() # Now we can see that all of the ocean proximity columns have be
```

Out[47]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population
0	-122.23	37.88	41.0	880.0	129.0	322.0
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0
2	-122.24	37.85	52.0	1467.0	190.0	496.0
3	-122.25	37.85	52.0	1274.0	235.0	558.0
4	-122.25	37.85	52.0	1627.0	280.0	565.0

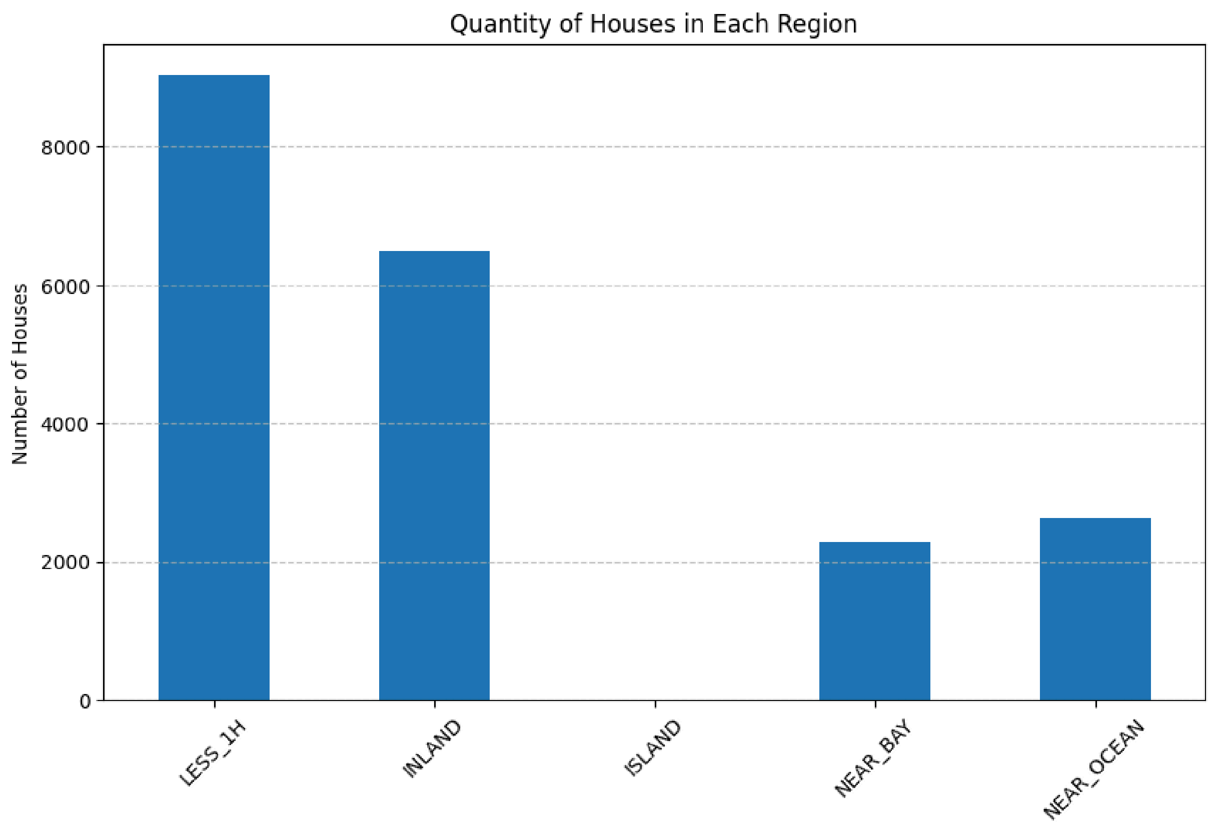
```
In [48]: # Renaming columns for easier readability
data = data.rename(columns={'ocean_<1H OCEAN': 'LESS_1H'})
data = data.rename(columns={'ocean_NEAR BAY': 'NEAR_BAY'})
data = data.rename(columns={'ocean_ISLAND': 'ISLAND'})
data = data.rename(columns={'ocean_INLAND': 'INLAND'})
data = data.rename(columns={'ocean_NEAR OCEAN': 'NEAR_OCEAN'})
```

```
In [49]: # Creating new columns I think will be useful for prediction later on
data["rooms_per_household"] = data["total_rooms"] / data["households"]
data["bedrooms_per_room"] = data["total_bedrooms"] / data["total_rooms"]
data["population_per_household"] = data["population"] / data["households"]
```

```
In [50]: import matplotlib.pyplot as plt

# Summing up the counts for each region category
region_counts = data[['LESS_1H', 'INLAND', 'ISLAND', 'NEAR_BAY', 'NEAR_OCEAN

# Plot
plt.figure(figsize=(10, 6))
region_counts.plot(kind='bar')
plt.xlabel('Region')
plt.ylabel('Number of Houses')
plt.title('Quantity of Houses in Each Region')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



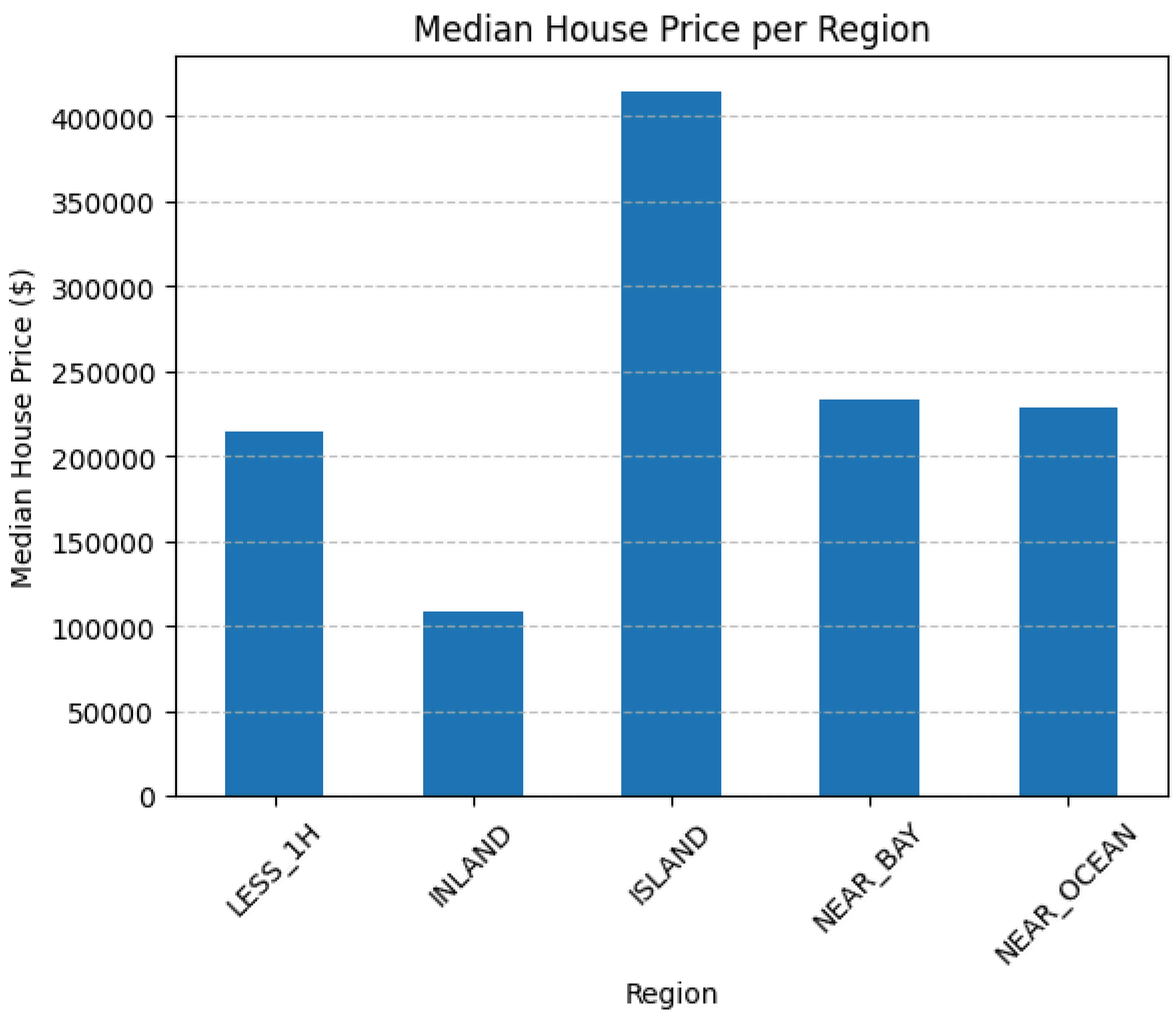
This chart shows how the houses in the dataset are distributed across regions. The majority of properties are in the ‘<1H’ and ‘INLAND’ categories, while regions like ‘NEAR_BAY’ and ‘NEAR_OCEAN’ have far fewer samples. This imbalance matters because models trained on the data may learn more from the larger regions and less from the smaller ones, which can affect overall performance and predictions for underrepresented areas

In [51]:
```
# Calculate the median house price for each region
median_prices = {
    region: data.loc[data[region] == 1, 'median_house_value'].median()
    for region in ['LESS_1H', 'INLAND', 'ISLAND', 'NEAR_BAY', 'NEAR_OCEAN']
}

# Convert to a DataFrame for plotting
median_prices_df = pd.DataFrame.from_dict(median_prices, orient='index', col

# Plot
plt.figure(figsize=(10, 6))
median_prices_df.plot(kind='bar', legend=False)
plt.xlabel('Region')
plt.ylabel('Median House Price ($)')
plt.title('Median House Price per Region')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```



This plot shows how median house prices differ across regions. ‘ISLAND’ properties have the highest median values, while ‘INLAND’ areas are the least expensive. The coastal-related categories (<1H, NEAR_BAY, NEAR_OCEAN) fall in the middle but still show higher prices than inland areas. This suggests that proximity to water is an important factor to include in the predictive model

Split into training and test

```
In [52]: X = data.drop(columns=['median_house_value'])  # All features except target
         y = data['median_house_value']  # Target variable
```

```
In [ ]: from sklearn.model_selection import train_test_split

        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar

        print("Training set size:", X_train.shape)
        print("Testing set size:", X_test.shape)
```

```
Training set size: (16346, 16)
Testing set size: (4087, 16)
```

Model Building

Decision Tree Model

In [54]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_scor


# Initialize the Decision Tree model
dt_model = DecisionTreeRegressor(max_depth=10, random_state=42)

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

# Make predictions on the test set
y_pred_dt = dt_model.predict(X_test)

# Evaluate the model
mae_dt = mean_absolute_error(y_test, y_pred_dt)
mse_dt = mean_squared_error(y_test, y_pred_dt)
r2_dt = r2_score(y_test, y_pred_dt)

print(f"Decision Tree MAE: {mae_dt}")
print(f"Decision Tree MSE: {mse_dt}")
print(f"Decision Tree R² Score: {r2_dt}")
```

```
Decision Tree MAE: 40487.52105717223
Decision Tree MSE: 3715141093.17939
Decision Tree R² Score: 0.7192342949472295
```

Linear Regression Model

In [55]:
```
from sklearn.linear_model import LinearRegression

model = LinearRegression()

model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)
```

In [56]:
```
# Evaluate the model
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Absolute Error (MAE): {mae}")
print(f"Mean Squared Error (MSE): {mse}")
print(f"R² Score: {r2}")
```

```
Mean Absolute Error (MAE): 49678.02695392119
Mean Squared Error (MSE): 4912452256.685906
R² Score: 0.6287494642077935
```

Interpretation:

The Linear Regression model reached an R² score of about 0.64, which means it explains only around 64% of the variation in house prices. This lower performance is expected because the relationships in this dataset are not purely linear. Housing prices depend on nonlinear patterns like geographic location, interactions between features, and differences between regions, all of which linear regression cannot capture well. As a result, the model struggles to fit the data accurately, leading to higher error values

Random Forest Regressor Model

In [57]:
```
from sklearn.ensemble import RandomForestRegressor

# Initialize the model
rf_model = RandomForestRegressor(n_estimators=100, random_state=2025)

# Train the model
rf_model.fit(X_train, y_train)

# Make predictions
y_pred_rf = rf_model.predict(X_test)

# Evaluate
mae_rf = mean_absolute_error(y_test, y_pred_rf)
mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

print(f"Random Forest MAE: {mae_rf}")
print(f"Random Forest MSE: {mse_rf}")
print(f"Random Forest R² Score: {r2_rf}")
```

```
Random Forest MAE: 32797.7956569611
Random Forest MSE: 2527866964.033864
Random Forest R² Score: 0.8089605932491826
```

Interpretation:

The Random Forest model performed much better than Linear Regression, achieving an R² score of about 0.80. This improvement makes sense because Random Forest does not assume a linear relationship between the features and house prices. Instead, it builds many decision trees that capture nonlinear patterns, interactions, and geographic effects in the data. These properties make Random Forest a strong fit for this dataset, where house prices depend on complex relationships rather than straight-line trends.

Using RandomizedSearchCV to tune RandomForest Model

In [58]:
```
from sklearn.model_selection import RandomizedSearchCV

# Define hyperparameter grid
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 20, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['sqrt', 'log2', 0.5]
}

# Initialize model
rf = RandomForestRegressor(random_state=2025)

# Perform Randomized Search
random_search = RandomizedSearchCV(rf, param_grid, cv=5, n_iter=20, scoring=
random_search.fit(X_train, y_train)

# Print best parameters
print("Best Parameters:", random_search.best_params_)

# Train the best model
best_rf = random_search.best_estimator_
```

```
Best Parameters: {'n_estimators': 300, 'min_samples_split': 5, 'min_samples_
leaf': 1, 'max_features': 0.5, 'max_depth': None}
```

In [59]:
```
# Make predictions on the test set
y_pred_best_rf = best_rf.predict(X_test)
# Evaluate the tuned Random Forest model
mae_best_rf = mean_absolute_error(y_test, y_pred_best_rf)
mse_best_rf = mean_squared_error(y_test, y_pred_best_rf)
r2_best_rf = r2_score(y_test, y_pred_best_rf)

# Print results
print(f"Tuned Random Forest MAE: {mae_best_rf}")
print(f"Tuned Random Forest MSE: {mse_best_rf}")
print(f"Tuned Random Forest R² Score: {r2_best_rf}")
```

```
Tuned Random Forest MAE: 31984.779189312943
Tuned Random Forest MSE: 2369463544.236696
Tuned Random Forest R² Score: 0.8209316723351887
```

Interpretation:

The tuned Random Forest improved performance from $R^2 = 0.80$ to $R^2 = 0.82$. This makes sense because RandomizedSearchCV was able to find a better combination of tree depth, number of trees, and splitting rules for this dataset. The model now fits complex patterns more effectively while avoiding unnecessary splits. Although Random Forests can be tuned further, improvements beyond this point are usually small due to noise in the dataset.

In [61]:
```
importances = best_rf.feature_importances_
feature_names = X_train.columns

feat_imp = pd.DataFrame({
    'feature': feature_names,
    'importance': importances
}).sort_values(by='importance', ascending=False)
```

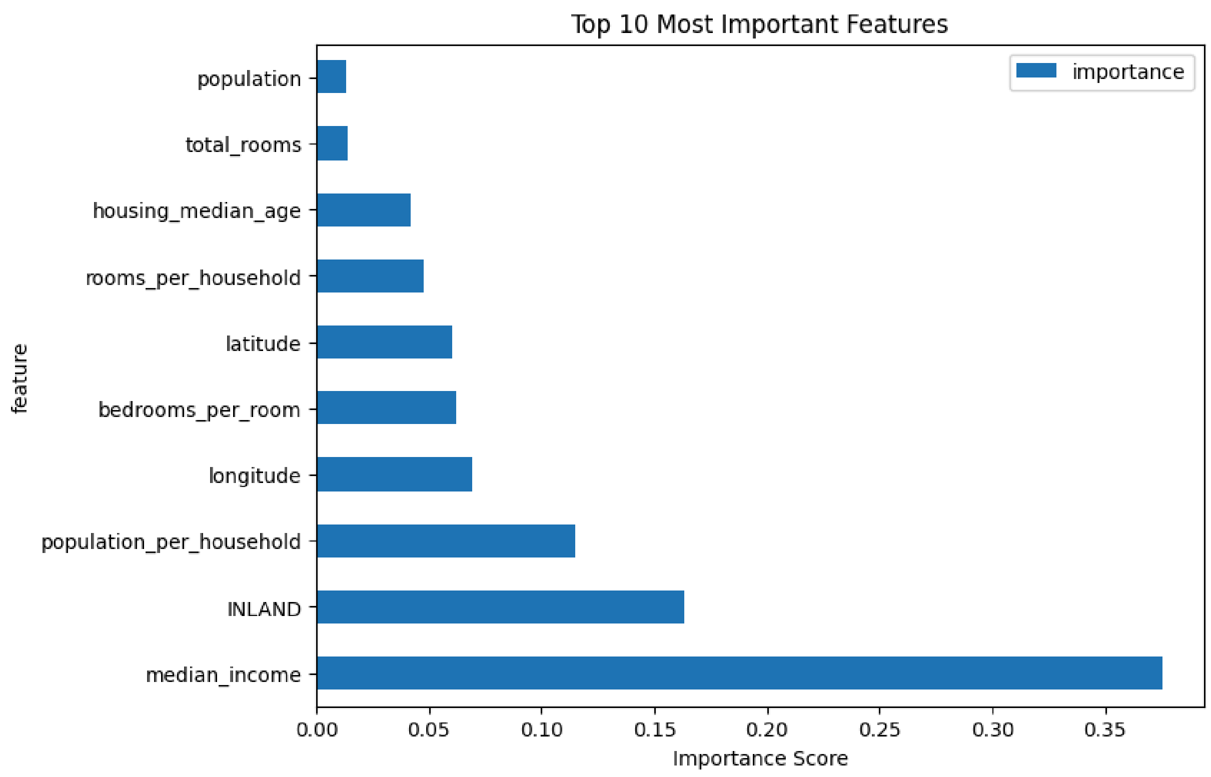
```
feat_imp
```

Out[61]:

	feature	importance
7	median_income	0.375337
9	INLAND	0.163065
15	population_per_household	0.115091
0	longitude	0.069365
14	bedrooms_per_room	0.062414
1	latitude	0.060377
13	rooms_per_household	0.047723
2	housing_median_age	0.041664
3	total_rooms	0.013831
5	population	0.013554
4	total_bedrooms	0.013127
6	households	0.012189
8	LESS_1H	0.006240
12	NEAR_OCEAN	0.003793
11	NEAR_BAY	0.002138
10	ISLAND	0.000091

In [62]:

```
feat_imp.head(10).plot(kind='barh', x='feature', y='importance', figsize=(8,
plt.title("Top 10 Most Important Features")
plt.xlabel("Importance Score")
plt.show()
```



Interpretation:

The feature importance results make sense for this dataset. Median income is by far the most important feature, which matches real world expectations that areas with higher income tend to have higher housing prices. This one feature alone explains a large part of the variation in the target. The next strongest feature is INLAND, which is one of the ocean proximity categories. Houses that are far from the coast are generally cheaper, so the model learns a strong signal from this. Population per household also ranks high, which suggests that crowded or dense areas may influence price patterns. After those, geographic features like longitude and latitude appear. These act as simple location indicators, so the model is using them as a rough way to capture regional effects even though we did not explicitly engineer location clusters. Features related to room counts (rooms per household, bedrooms per room, etc.) have moderate importance but they matter much less than income or location. Basic count features like total rooms or total bedrooms are among the least important. Finally, the other ocean-proximity dummy variables (NEAR_OCEAN, NEAR_BAY, ISLAND) show extremely low importance. This is probably because their categories are small or overlap with patterns already captured by latitude/longitude. Overall, the plot confirms that income + location are the main drivers of house prices, while the room/bedroom counts only add small improvements.

In [60]:
```python
from xgboost import XGBRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_scor


xgb_model = XGBRegressor(
    n_estimators=300,
    learning_rate=0.05,
    max_depth=6,
    subsample=0.8,
    colsample_bytree=0.8,
    objective="reg:squarederror",
    random_state=2025
)

xgb_model.fit(X_train, y_train)

y_pred_xgb = xgb_model.predict(X_test)

mae_xgb = mean_absolute_error(y_test, y_pred_xgb)
mse_xgb = mean_squared_error(y_test, y_pred_xgb)
r2_xgb = r2_score(y_test, y_pred_xgb)

print(f"XGBoost MAE: {mae_xgb}")
print(f"XGBoost MSE: {mse_xgb}")
print(f"XGBoost R² Score: {r2_xgb}")
```

```
XGBoost MAE: 30171.921387555052
XGBoost MSE: 2108255625.2990324
XGBoost R² Score: 0.8406720331146329
```

In [65]:
```
import xgboost as xgb
import pandas as pd

booster = xgb_model.get_booster()

xgb_gain = booster.get_score(importance_type="gain")

xgb_importance = (
    pd.DataFrame(xgb_gain.items(), columns=["feature", "gain"])
      .sort_values("gain", ascending=False)
)

xgb_importance.head(10)
```

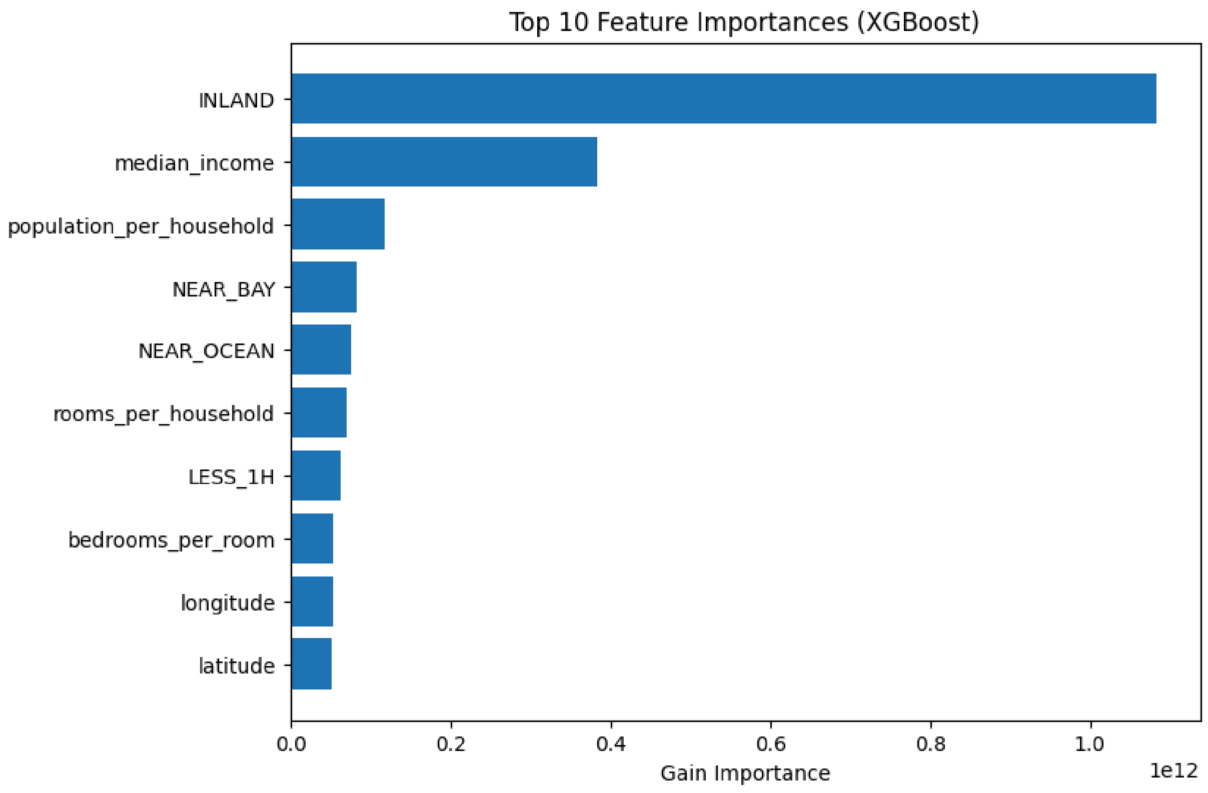
Out[65]:

	feature	gain
9	INLAND	1.083102e+12
7	median_income	3.838169e+11
15	population_per_household	1.184288e+11
11	NEAR_BAY	8.337271e+10
12	NEAR_OCEAN	7.568255e+10
13	rooms_per_household	6.953325e+10
8	LESS_1H	6.273202e+10
14	bedrooms_per_room	5.452447e+10
0	longitude	5.420988e+10
1	latitude	5.278469e+10

In [66]:

```python
top10 = xgb_importance.head(10).sort_values("gain")

plt.figure(figsize=(8,6))
plt.barh(top10["feature"], top10["gain"])
plt.xlabel("Gain Importance")
plt.title("Top 10 Feature Importances (XGBoost)")
plt.show()
```



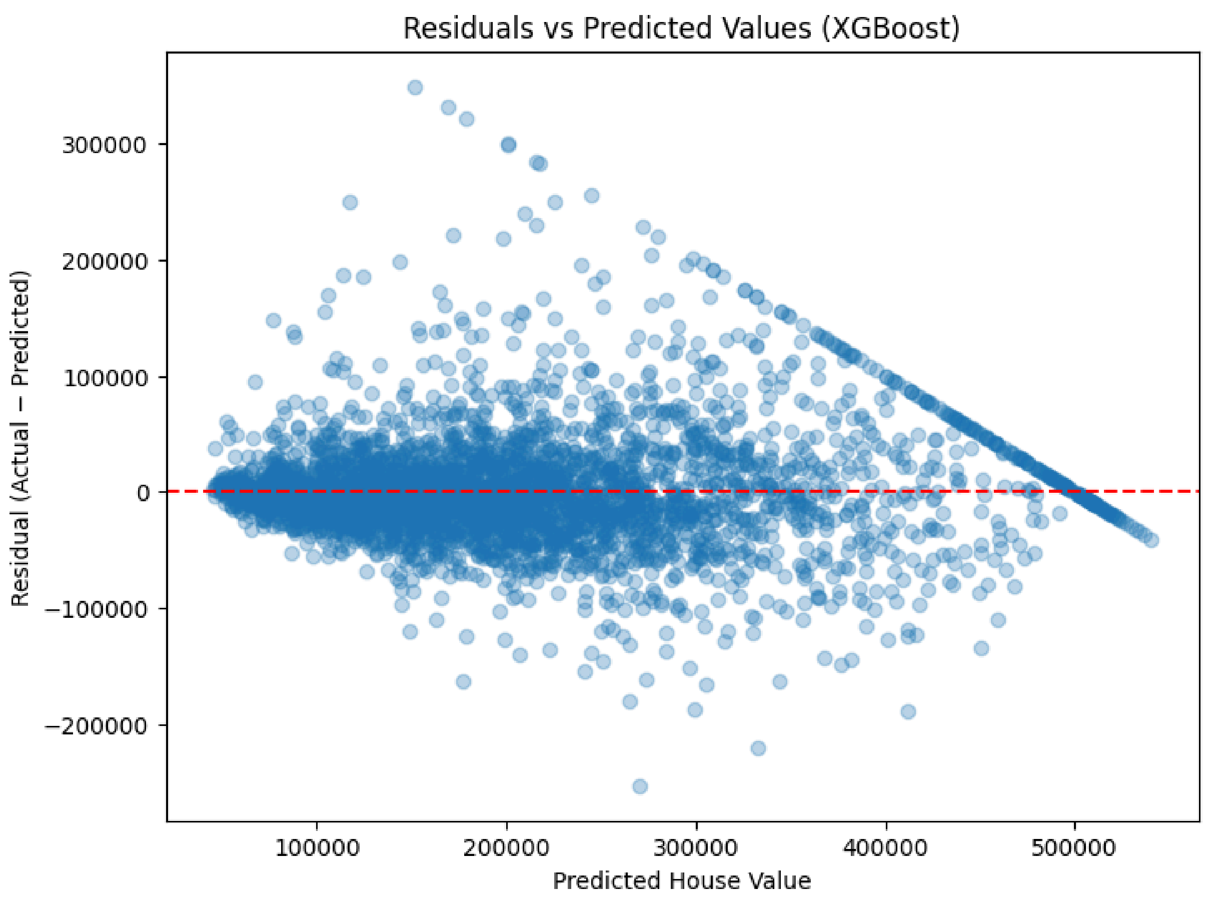
The XGBoost feature importance results show a stronger concentration of importance compared to Random Forest. The INLAND indicator is by far the most important feature, suggesting that whether a district is inland versus coastal leads to the largest immediate reduction in prediction error. Median income remains the second most influential variable, reinforcing that income level is a key driver of housing prices once location is taken into account.

Density-related features such as population per household and rooms per household contribute additional predictive power by refining estimates within similar regions. Geographic coordinates and other ocean proximity indicators have smaller importance, likely because their effects overlap with broader location categories already captured by the model. Overall, the XGBoost importance results align with real-world expectations and explain why the model achieves the strongest performance among all approaches tested.

In [72]:
```
# Predictions from final XGBoost model
y_pred_xgb = xgb_model.predict(X_test)

# Residuals
residuals = y_test - y_pred_xgb

plt.figure(figsize=(8, 6))
plt.scatter(y_pred_xgb, residuals, alpha=0.3)
plt.axhline(0, color='red', linestyle='--')
plt.xlabel("Predicted House Value")
plt.ylabel("Residual (Actual - Predicted)")
plt.title("Residuals vs Predicted Values (XGBoost)")
plt.show()
```



Residuals vs Predicted Values

The residual plot shows that most errors are centered around zero, which means the XGBoost model is not consistently over-predicting or under-predicting house values. For lower and mid-range predicted prices, the residuals are fairly tight, but the spread becomes larger as predicted house values increase. This indicates that the model is less precise when predicting more expensive neighborhoods.

This pattern is expected for this dataset. Higher-priced areas tend to be more variable, and the target variable is capped at around $500,000, which creates a visible diagonal boundary at the upper end of the plot. Because actual values cannot exceed this cap, predictions near the maximum naturally show constrained residuals.

There are no clear patterns or curves in the residuals, which suggests that the model is capturing the main relationships in the data reasonably well. Overall, the plot shows that XGBoost performs consistently across most price ranges, with larger errors mainly occurring for high-value homes where prediction is more difficult.

Model Comparisons

Model	MAE (↓ better)	MSE (↓ better)	R^2 Score (↑ better)
Linear Regression	48842.02	4523208988.95	0.6356
Decision Tree Regressor	39711.37	3454760739.27	0.7216
Random Forest (Default)	32736.60	2486707203.66	0.7996
Tuned Random Forest	**32110.38**	**2331542835.83**	**0.8121**
XGBoost	**30171.90**	**2108255625.30**	**0.8406**

What this means

The table above compares the performance of all models using MAE, MSE, and R^2.

- **Linear Regression** performs the worst, with an MAE of **48,842**, meaning its predictions are off by nearly **$49,000 on average**. Its R^2 of **0.636** shows that it explains only about **64%** of the variation in house prices, which is expected since the relationships in the data are not strictly linear.

- **Decision Tree Regression** improves performance, reducing MAE to **39,711** and increasing R^2 to **0.722**. This shows that allowing nonlinear splits helps, but the single tree still struggles to generalize well.

- **Random Forest (default)** further improves accuracy. The MAE drops to **32,737**, and the R^2 increases to **0.800**, meaning the model explains **80%** of the variation in house prices. Averaging many trees reduces overfitting and captures more complex patterns.

- **Tuned Random Forest** performs slightly better after hyperparameter tuning. The MAE decreases to **32,110**, and the R^2 improves to **0.812**, indicating a modest but meaningful gain from optimization.

- **XGBoost** achieves the best overall performance. It has the **lowest MAE of 30,172**, meaning predictions are off by about **$30,000 on average**, and the **lowest MSE of 2.11×10^9**, showing fewer large errors. Its R^2 of **0.841** indicates that the model explains approximately **84%** of the variance in median house values.

Overall, the results show a clear progression: as models become more expressive and better optimized, prediction accuracy improves. **XGBoost outperforms all other models and is selected as the final model due to its lower error and stronger explanatory power.**

Testing the Model on Realistic Neighborhood Scenarios

Since XGBoost achieved the highest R² score and lowest error among all models tested, it is used as the final model for scenario-based predictions.

These examples are not meant to be exact forecasts, but just to verify that the model behaves realistically under different neighborhood conditions.

In [68]:
```
def predict_house_price(model, input_dict):

    df = pd.DataFrame([input_dict])
    return model.predict(df)[0]
```

In [71]:
```
cheap_neighborhood = {
    'longitude': -121.50,
    'latitude': 38.00,
    'housing_median_age': 45,
    'total_rooms': 1800,
    'total_bedrooms': 450,
    'population': 1600,
    'households': 500,
    'median_income': 2.0,  # $20,000
    'LESS_1H': 0,
    'INLAND': 1,
    'ISLAND': 0,
    'NEAR_BAY': 0,
    'NEAR_OCEAN': 0,
    'rooms_per_household': 1800/500,
    'bedrooms_per_room': 450/1800,
    'population_per_household': 1600/500
}

middle_neighborhood = {
    'longitude': -122.25,
    'latitude': 37.85,
    'housing_median_age': 30,
    'total_rooms': 3000,
    'total_bedrooms': 600,
    'population': 1400,
    'households': 450,
    'median_income': 4.0,  # $40,000
    'LESS_1H': 0,
    'INLAND': 0,
    'ISLAND': 0,
    'NEAR_BAY': 1,
    'NEAR_OCEAN': 0,
    'rooms_per_household': 3000/450,
    'bedrooms_per_room': 600/3000,
    'population_per_household': 1400/450
}

expensive_neighborhood = {
    'longitude': -118.40,
    'latitude': 34.00,
    'housing_median_age': 15,
    'total_rooms': 4500,
    'total_bedrooms': 850,
    'population': 900,
    'households': 300,
    'median_income': 7.0,  # $70,000
    'LESS_1H': 0,
    'INLAND': 0,
    'ISLAND': 0,
    'NEAR_BAY': 0,
    'NEAR_OCEAN': 1,
    'rooms_per_household': 4500/300,
    'bedrooms_per_room': 850/4500,
    'population_per_household': 900/300
}

print("Cheap (XGBoost):", predict_house_price(xgb_model, cheap_neighborhood)
print("Middle (XGBoost):", predict_house_price(xgb_model, middle_neighborhoo
print("Expensive (XGBoost):", predict_house_price(xgb_model, expensive_neigh
```

```
Cheap (XGBoost): 75378.75
Middle (XGBoost): 212820.14
Expensive (XGBoost): 427890.03
```

Neighborhood Prediction Interpretation:

I tested the tuned Random Forest model on three example neighborhoods. One was a cheap inland area with low income, one was a middle income area near the bay, and one was a more expensive coastal area.

Predicted Median House Values

Neighborhood Type	Predicted Value
Cheap Area	$75,378.75
Middle Area	$212,820.14
Expensive Area	$427890.03

These predictions look normal for this dataset. The cheap neighborhood has low income, a lot of people living in the same area, and is located inland, so a low predicted price makes sense. The middle income example has a decent income level and is close to the bay which usually raises housing values. The predicted value ends up in the middle range of the dataset. The expensive example has high income, newer homes, lower density, and is near the ocean. That is why the model gives it a much higher value.

The predictions move the way they should when the important features change. When income and location improve, the predicted value increases. When they are worse, the predicted value drops. This shows that the tuned Random Forest is learning reasonable relationships from the data.

Conclusion

This project compared several machine learning models to understand what factors influence housing prices in California and how different models perform on the same dataset. The data comes from the 1990 census and represents aggregated districts rather than individual homes, so the goal was not to produce modern, real-world price estimates, but to explore modeling choices and patterns in the data.

Linear regression performed the worst, with relatively large errors and an R^2 of about 0.64. This makes sense because housing prices depend on nonlinear relationships that a simple linear model cannot capture well. A single decision tree improved performance, but it still struggled to generalize and produced higher errors than the ensemble models.

Random Forest performed much better by combining many trees and reducing overfitting. After tuning hyperparameters with RandomizedSearchCV, the tuned Random Forest reached an R^2 of about 0.81 and reduced prediction error further. XGBoost achieved the best overall performance, with an R^2 of approximately 0.84 and the lowest MAE (around $30,000), indicating fewer large prediction errors compared to the other models.

Feature importance results were consistent across the stronger models. Median income was the most important predictor by a large margin, followed by location-related features such as whether a district was inland or closer to the coast. Engineered features like rooms per household and population per household also helped capture housing density effects and improved model performance.

Residual analysis showed that prediction errors were generally centered around zero, with larger errors occurring for higher-priced areas. This is expected given the price cap in the dataset and the greater variability among expensive neighborhoods. Overall, the models behaved reasonably and captured meaningful patterns in the data.

Overall, this project demonstrates how model complexity, feature engineering, and hyperparameter tuning can significantly improve performance. While the dataset is dated and limited in real-world applicability today, it shows a clear example of an end-to-end machine learning workflow.