Assignment Objective

Build a model to predict the house prices

Import Libraries

```
# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Import the dataset

```
# Import dataset as a pandas DataFrame
df = pd.read_excel('./dataset1.xlsx')
```

Exploratory Data Analysis

View 5 random samples

```
df.sample(5)
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	po
9345	-117.65	33.65	15	4713	671.0	
8432	-122.54	37.96	33	2534	495.0	
17863	-119.32	36.33	20	1896	266.0	
689	-122.10	37.66	33	1954	464.0	
13349	-117.08	32.69	36	1571	284.0	

View the number of rows and columns in the dataset

```
df.shape
```

```
(18565, 10)
```

One-Hot Encoding for Ocean Proximity

```
from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore').set_output(
ohe_transform = encoder.fit_transform(df[['ocean_proximity']])
df = pd.concat([df,ohe_transform], axis=1).drop(columns=['ocean_proximity'])
```

```
df.dtypes
```

	0
longitude	float64
latitude	float64
housing_median_age	int64
total_rooms	int64
total_bedrooms	float64
population	int64
households	int64
median_income	float64
median_house_value	int64
ocean_proximity_<1H OCEAN	float64
ocean_proximity_INLAND	float64
ocean_proximity_ISLAND	float64
ocean_proximity_NEAR BAY	float64
ocean_proximity_NEAR OCEAN	float64

dtype: object

View number of missing values in each column

```
df.isnull().sum()
```

	0
longitude	0
latitude	0
housing_median_age	0
total_rooms	0
total_bedrooms	189
population	0
households	0
median_income	0
median_house_value	0
ocean_proximity_<1H OCEAN	0
ocean_proximity_INLAND	0
ocean_proximity_ISLAND	0
ocean_proximity_NEAR BAY	0
ocean_proximity_NEAR OCEAN	0

dtype: int64

Using KNN Imputer for missing values

```
# total_bedrooms has 189 missing values
from sklearn.impute import KNNImputer
imputer = KNNImputer(n_neighbors=2)
imputed_array =  imputer.fit_transform(df)
df = pd.DataFrame(imputed_array, columns=df.columns, index=df.index)
```

```
df.isnull().sum()
```

	0
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_<1H OCEAN	0
ocean_proximity_INLAND	0
ocean_proximity_ISLAND	0
ocean_proximity_NEAR BAY	0
ocean_proximity_NEAR OCEAN	0

dtype: int64

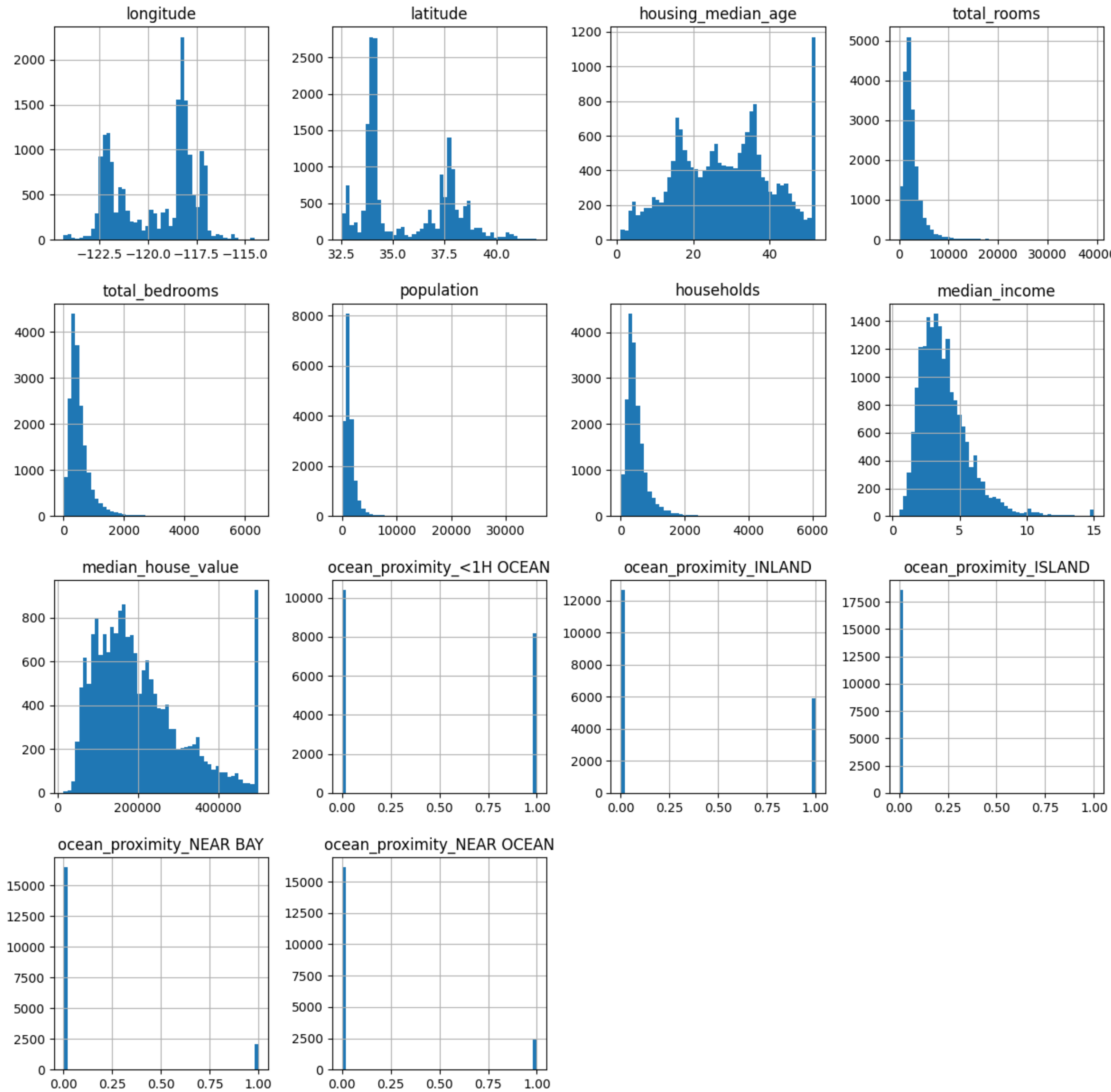
```
df.sample()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	po
11106	-116.53	33.85	16.0	10077.0	2186.0	

View the distribution of features and target variable

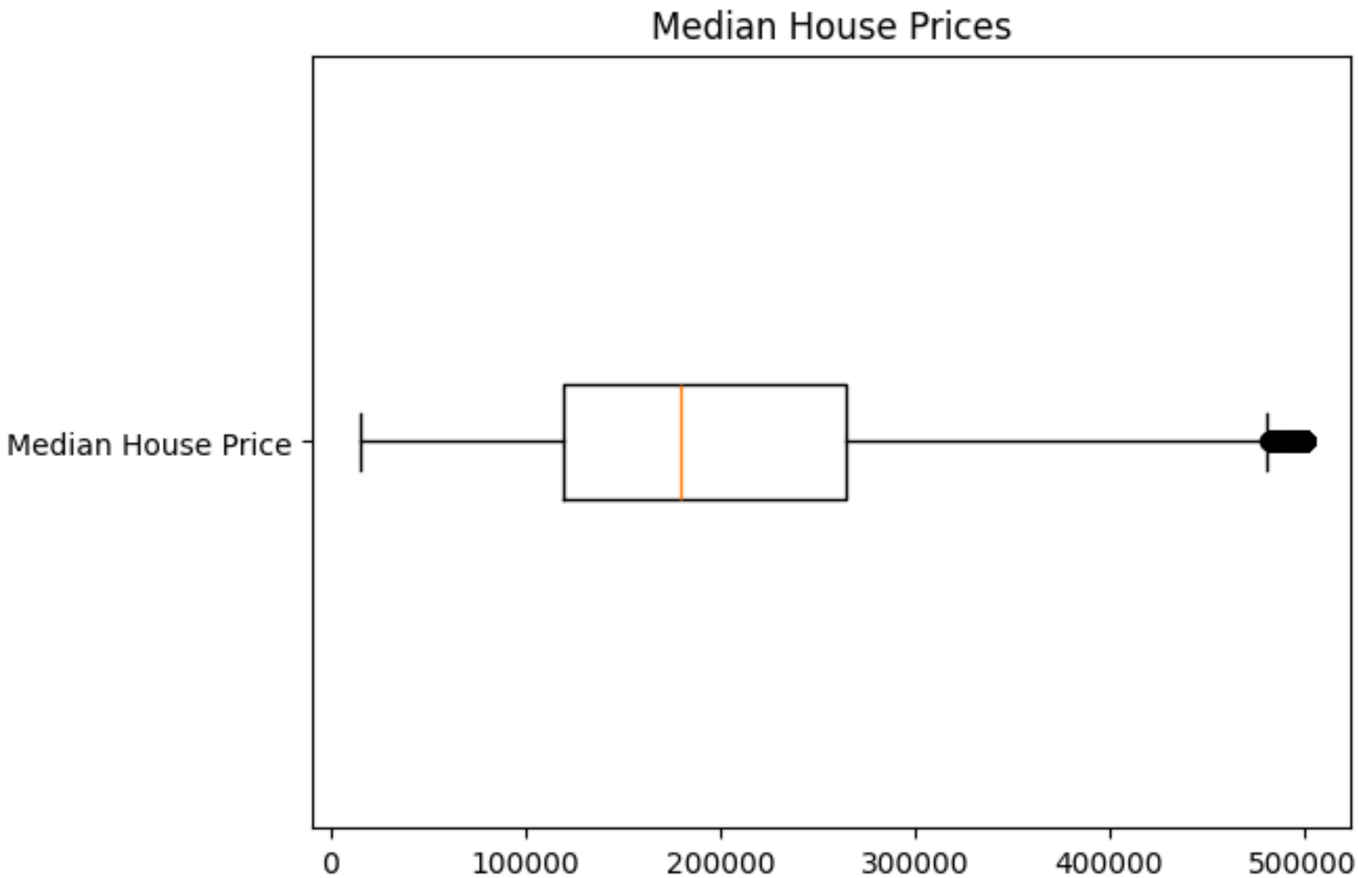
```
# View distribution of  variables
df.hist(bins=50, figsize=(15, 15))
```

```
plt.show()
```


longitude
0 500 1000 1500 2000
−122.5 −120.0 −117.5 −115.0
latitude
0 500 1000 1500 2000 2500
32.5 35.0 37.5 40.0
housing_median_age
0 200 400 600 800 1000 1200
0 20 40
total_rooms
0 1000 2000 3000 4000 5000
0 10000 20000 30000 40000
total_bedrooms
0 1000 2000 3000 4000
0 2000 4000 6000
population
0 2000 4000 6000 8000
0 10000 20000 30000
households
0 1000 2000 3000 4000
0 2000 4000 6000
median_income
0 200 400 600 800 1000 1200 1400
0 5 10 15
median_house_value
0 200 400 600 800
0 200000 400000
ocean_proximity_<1H OCEAN
0 2000 4000 6000 8000 10000
0.00 0.25 0.50 0.75 1.00
ocean_proximity_INLAND
0 2000 4000 6000 8000 10000 12000
0.00 0.25 0.50 0.75 1.00
ocean_proximity_ISLAND
0 2500 5000 7500 10000 12500 15000 17500
0.00 0.25 0.50 0.75 1.00
ocean_proximity_NEAR BAY
0 2500 5000 7500 10000 12500 15000
0.00 0.25 0.50 0.75 1.00
ocean_proximity_NEAR OCEAN
0 2500 5000 7500 10000 12500 15000
0.00 0.25 0.50 0.75 1.00

Target variable has outliers

```
plt.boxplot(df['median_house_value'], tick_labels=['Median House Price'],vert=Fal
plt.title('Median House Prices')
plt.show()
```



Remove outliers in target variable

```
# Remove outliers
min_value = df['median_house_value'].min()
q1,q3 = np.percentile(df['median_house_value'],[25,75])
median = df['median_house_value'].median()
max_value = df['median_house_value'].max()
min_value,q1,q3,median,max_value
```

```
(14999.0, np.float64(119300.0), np.float64(264400.0), 179400.0, 500001.0)
```

```
iqr = q3-q1
lower_boundary = q1 - 1.5*iqr
upper_boundary = q3 + 1.5*iqr
range = [lower_boundary,upper_boundary]
range
```

```
[np.float64(-98350.0), np.float64(482050.0)]
```

```
# outliers
outlier = df[(df['median_house_value'] < lower_boundary) | (df['median_house_valu
outlier
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	po
82	-122.27	37.80	52.0	249.0	78.0	
414	-122.25	37.87	52.0	609.0	236.0	
444	-122.25	37.86	48.0	2153.0	517.0	
448	-122.24	37.86	52.0	1668.0	225.0	
449	-122.24	37.85	52.0	3726.0	474.0	
...	...	...	...	...	...	
18366	-118.90	34.14	35.0	1503.0	263.0	
18370	-118.69	34.18	11.0	1177.0	138.0	
18371	-118.80	34.19	4.0	15572.0	2222.0	
18380	-118.69	34.21	10.0	3663.0	409.0	
18387	-118.85	34.27	50.0	187.0	33.0	

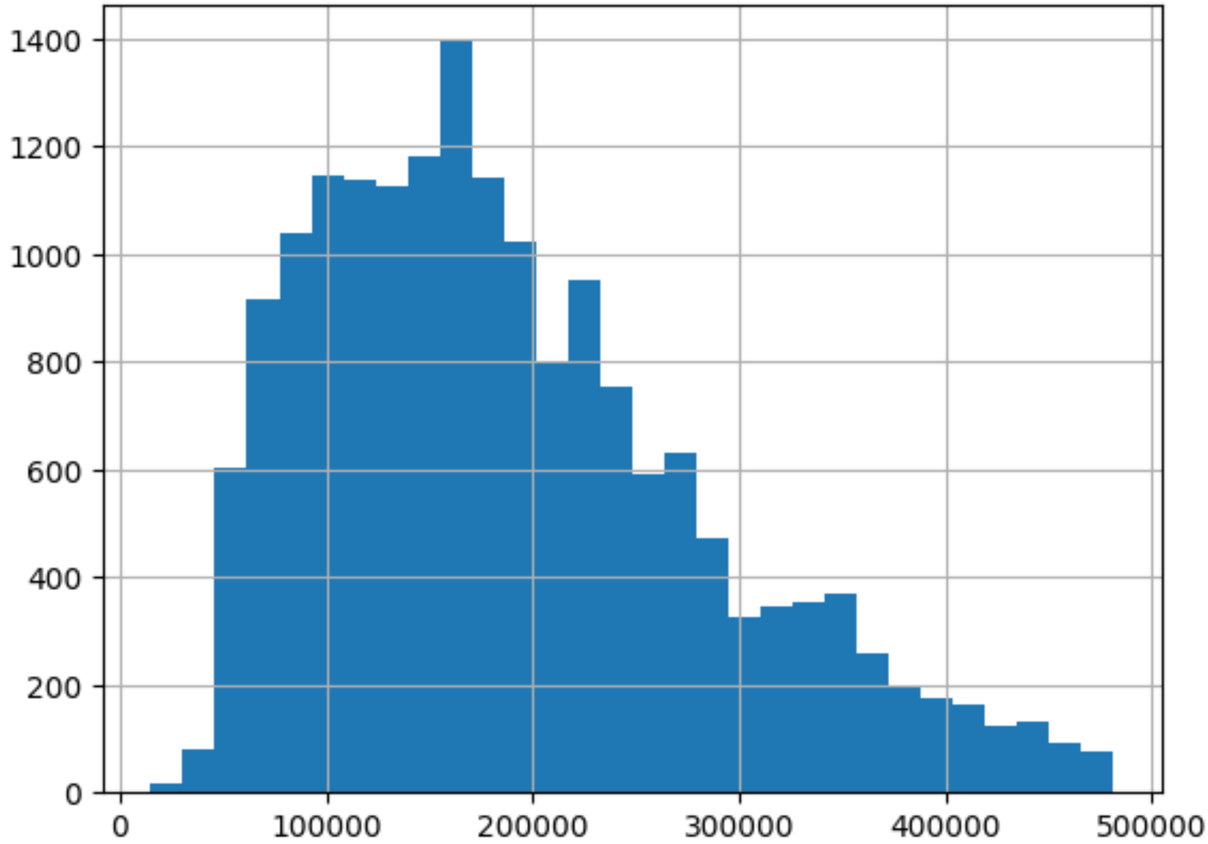
964 rows × 14 columns

Next steps: Generate code with outlier | View recommended plots | New interactive sheet

```
df = df[(df['median_house_value'] > lower_boundary) & (df['median_house_value'] <
```

```
df['median_house_value'].hist(bins=30)
```

<Axes: >




```
df.shape
```

(17601, 14)

View the datatype of each column

```
df.dtypes
```

	0
longitude	float64
latitude	float64
housing_median_age	float64
total_rooms	float64
total_bedrooms	float64
population	float64
households	float64
median_income	float64
median_house_value	float64
ocean_proximity_<1H OCEAN	float64
ocean_proximity_INLAND	float64
ocean_proximity_ISLAND	float64
ocean_proximity_NEAR BAY	float64
ocean_proximity_NEAR OCEAN	float64

dtype: object

Add new features

```
# Add new features
df['rooms_per_household'] = df['total_rooms'] / df['households']
df['bedrooms_per_room'] = df['total_bedrooms'] / df['total_rooms']
df['population_per_household'] = df['population'] / df['households']
df.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	popula
0	-122.23	37.88	41.0	880.0	129.0	
1	-122.22	37.86	21.0	7099.0	1106.0	2
2	-122.25	37.85	52.0	1627.0	280.0	
3	-122.25	37.85	52.0	919.0	213.0	
4	-122.25	37.84	52.0	2535.0	489.0	1

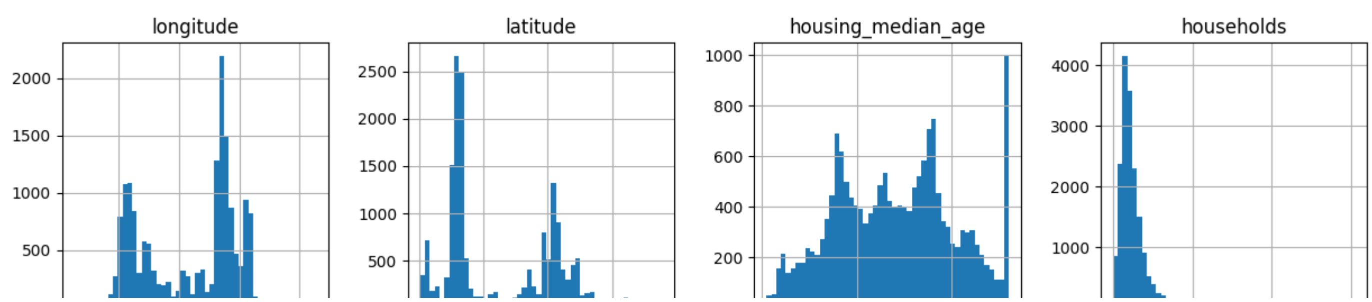
Next steps: Generate code with df | View recommended plots | New interactive sheet

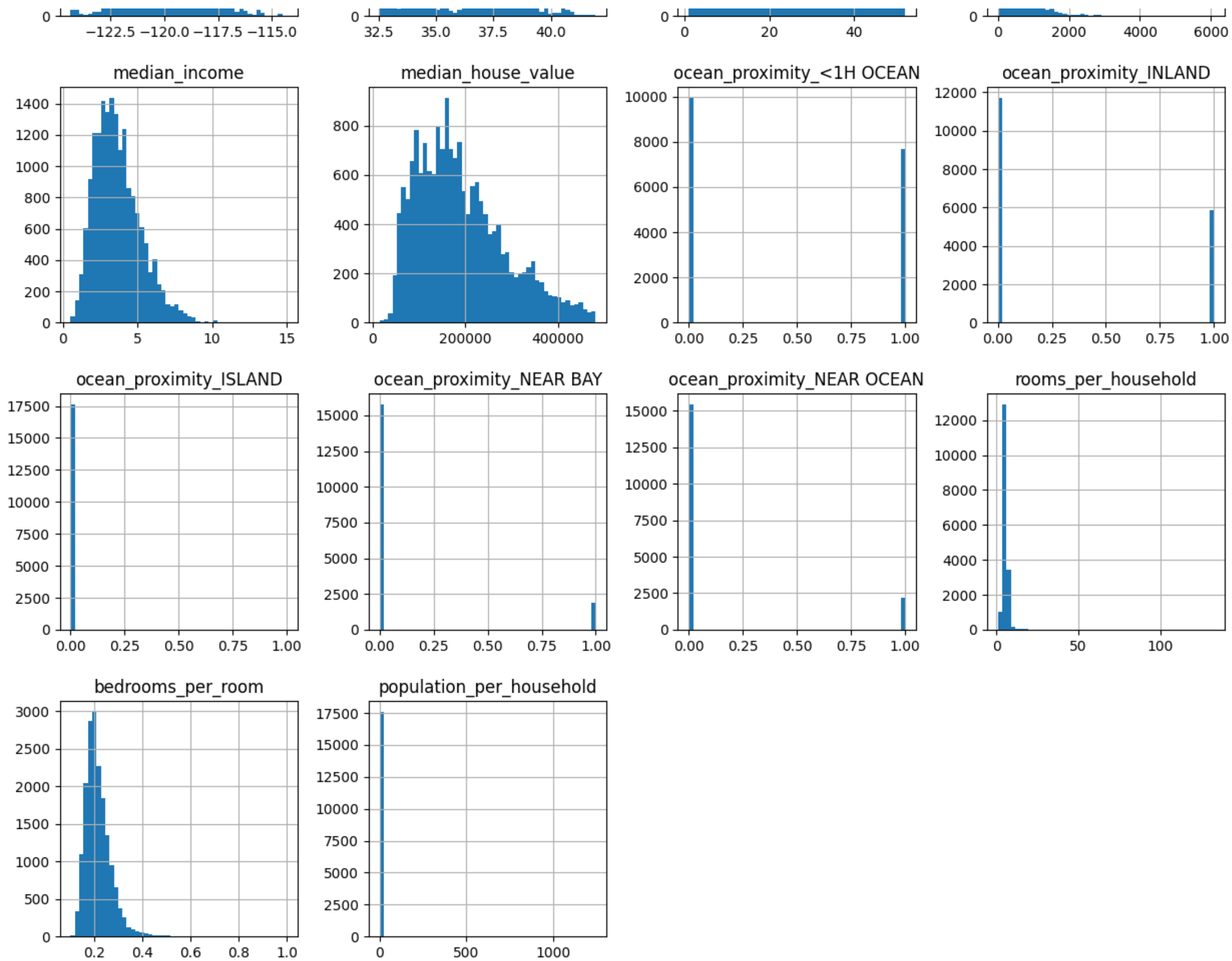
```
# Remove total_rooms, total_bedrooms, population_per_household
df = df.drop(['total_rooms','total_bedrooms','population'], axis=1)
```

```
df.sample()
```

	longitude	latitude	housing_median_age	households	median_income	media
8311	-120.11	36.96	17.0	536.0	3.8952	

```
df.hist(bins=50, figsize=(15, 15))
plt.show()
```




−122.5 −120.0 −117.5 −115.0
32.5 35.0 37.5 40.0
0 20 40
0 2000 4000 6000
median_income
median_house_value
ocean_proximity_<1H OCEAN
ocean_proximity_INLAND
ocean_proximity_ISLAND
ocean_proximity_NEAR BAY
ocean_proximity_NEAR OCEAN
rooms_per_household
bedrooms_per_room
population_per_household

```
# View correlation of median household value with other features
correlations = df.corr()
corr_house_price = correlations['median_house_value']
corr_house_price
```

	median_house_value
longitude	-0.046439
latitude	-0.150336
housing_median_age	0.062694
households	0.099881
median_income	0.643706
median_house_value	1.000000
ocean_proximity_<1H OCEAN	0.289026
ocean_proximity_INLAND	-0.503104
ocean_proximity_ISLAND	0.033530
ocean_proximity_NEAR BAY	0.156472
ocean_proximity_NEAR OCEAN	0.137837
rooms_per_household	0.106967
bedrooms_per_room	-0.215925
population_per_household	-0.020016

dtype: float64

Setup validation framework: Split data into training & test sets

```
# Split data into train, test, split
y = df['median_house_value']


features = ['latitude','median_income', 'ocean_proximity_<1H OCEAN', 'ocean_proxi
X = df[features]
```

```
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,random_state
```

```
X_train.shape
```

```
(14080, 8)
```

```
X_test.shape
```

```
(3521, 8)
```

```
y_train.shape
```

```
(14080,)
```

```
y_test.shape
```

```
(3521,)
```

Feature Scaling

```
# Standardize Features
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Train Linear Regression Model

```
# Train the Linear Regression Model
# Import LinearRegression.
from sklearn.linear_model import LinearRegression


# Instantiate linear regression model.
model = LinearRegression()


# Fit the model to the training data.
model.fit(X_train_scaled, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
# Make predictions on the testing data.
y_pred = model.predict(X_test_scaled)
```

Evaluate Model Performance

```
# Import metrics.
from sklearn.metrics import mean_squared_error, r2_score


# Calculate and print R^2 score.
r2 = r2_score(y_test, y_pred)
print(f"R-squared: {r2:.4f}")
```

```
R-squared: 0.5480
```

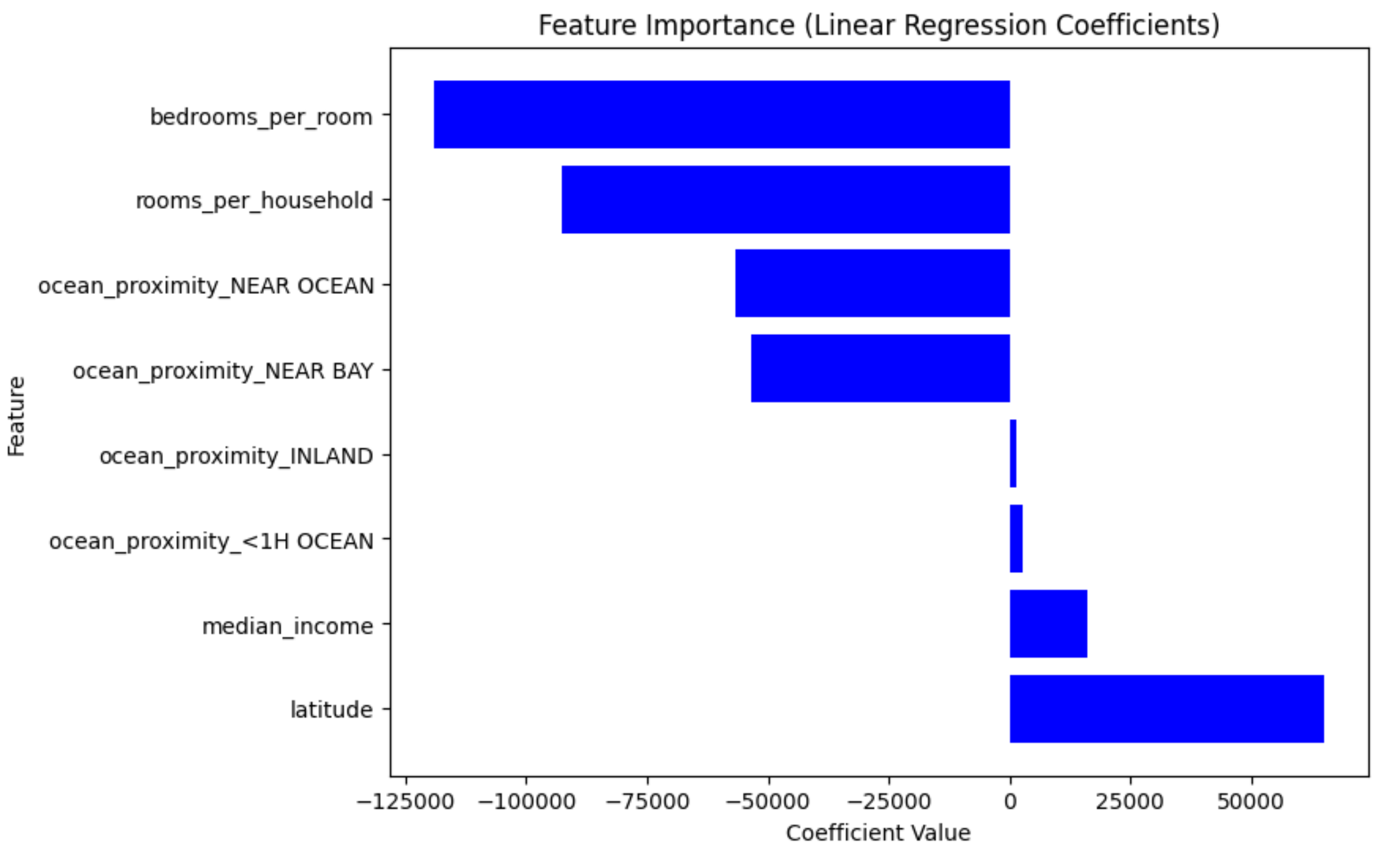
```
# Calculate and print MSE.
mse = mean_squared_error(y_test, y_pred)
print(f"Mean squared error: {mse:.4f}")


# Calculate and print RMSE.
rmse = mse ** 0.5
print(f"Root mean squared error: {rmse:.4f}")
```

```
Mean squared error: 4034851473.8891
Root mean squared error: 63520.4807
```

```
print("Intercept:", model.intercept_)


coeff_df = pd.DataFrame({"Feature": X.columns, "Coefficient": model.coef_})
print("\nFeature Coefficients:\n", coeff_df)
```

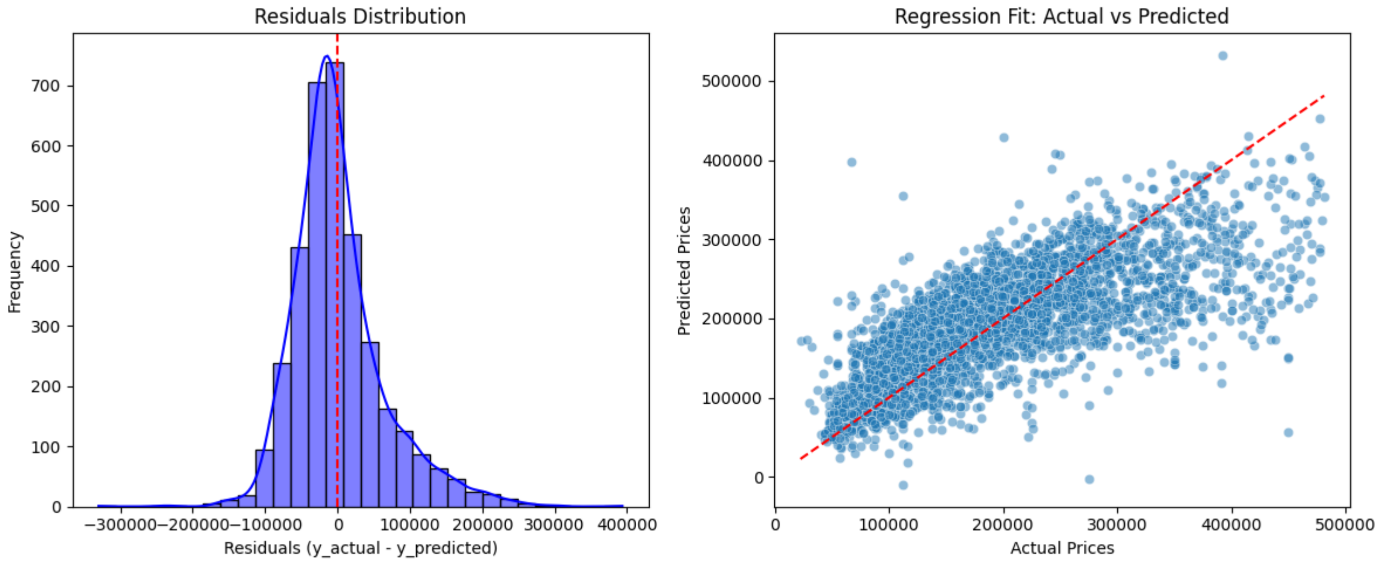
```
Intercept: 190135.3122159091

Feature Coefficients:
                       Feature    Coefficient
0                    latitude    2562.599372
1               median_income   64967.025877
2   ocean_proximity_<1H OCEAN  -92572.133647
3      ocean_proximity_INLAND -119011.501232
4    ocean_proximity_NEAR BAY  -53416.907974
5  ocean_proximity_NEAR OCEAN  -56561.160148
6         rooms_per_household    1512.709388
7            bedrooms_per_room   16111.885822
```

```
# Sort dataframe by coefficients.
coef_df_sorted = coeff_df.sort_values(by="Coefficient", ascending=False)


# Create plot.
plt.figure(figsize=(8,6))
plt.barh(coeff_df["Feature"], coef_df_sorted["Coefficient"], color="blue")
plt.xlabel("Coefficient Value")
plt.ylabel("Feature")
plt.title("Feature Importance (Linear Regression Coefficients)")
plt.show()
```




```
# Compute residuals.
residuals = y_test - y_pred
```

```
# Create plots.
plt.figure(figsize=(12,5))


# Plot 1: Residuals Distribution.
plt.subplot(1,2,1)
sns.histplot(residuals, bins=30, kde=True, color="blue")
plt.axvline(x=0, color='red', linestyle='--')
plt.title("Residuals Distribution")
plt.xlabel("Residuals (y_actual - y_predicted)")
plt.ylabel("Frequency")


# Plot 2: Regression Fit (Actual vs Predicted).
plt.subplot(1,2,2)
sns.scatterplot(x=y_test, y=y_pred, alpha=0.5)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', lin
plt.title("Regression Fit: Actual vs Predicted")
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")


# Show plots.
plt.tight_layout()
plt.show()
```




```
y_pred
```

```
array([227201.28619909, 104773.24437391,  90962.98234827, ...,
       234983.29961186,  99568.66165704, 167015.90740646])
```

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Train Random Forest Regressor Model

```
# Import RandomForestRegressor
from sklearn.ensemble import RandomForestRegressor

# Instantiate Random Forest Regressor model
forest_model = RandomForestRegressor(random_state=42)

# Fit the model to the training data
forest_model.fit(X_train_scaled, y_train)
```

```
RandomForestRegressor
RandomForestRegressor(random_state=42)
```

Evaluate Random Forest Regressor Model Performance

```
# Make predictions on the testing data
y_pred_forest = forest_model.predict(X_test_scaled)

# Calculate and print R^2 score
r2_forest = r2_score(y_test, y_pred_forest)
print(f"Random Forest R-squared: {r2_forest:.4f}")

# Calculate and print MSE
mse_forest = mean_squared_error(y_test, y_pred_forest)
print(f"Random Forest Mean squared error: {mse_forest:.4f}")

# Calculate and print RMSE
rmse_forest = mse_forest ** 0.5
print(f"Random Forest Root mean squared error: {rmse_forest:.4f}")
```

```
Random Forest R-squared: 0.6345
Random Forest Mean squared error: 3262685866.5878
Random Forest Root mean squared error: 57119.9253
```

Model Performance Comparison

Here is a comparison of the performance metrics for the Linear Regression and Random Forest Regressor models:

Metric	Linear Regression	Random Forest Regressor
R-squared	{{r2:.4f}}	{{r2_forest:.4f}}
Mean Squared Error	{{mse:.4f}}	{{mse_forest:.4f}}
Root Mean Squared Error	{{rmse:.4f}}	{{rmse_forest:.4f}}

Based on these metrics, the **Random Forest Regressor** model shows better performance with a higher R-squared value and lower Mean Squared Error and Root Mean Squared Error.

Make Predictions with Random Forest Regressor Model

```
# Simulate new data (replace with your actual new data)
# The new data should have the same features as the training data (X)
# 'latitude','median_income', 'ocean_proximity_<1H OCEAN', 'ocean_proximity_INLAN
new_data = pd.DataFrame({
    'latitude': [34.05, 33.93],
    'median_income': [5.0, 4.5],
    'ocean_proximity_<1H OCEAN': [1.0, 0.0],
    'ocean_proximity_INLAND': [0.0, 1.0],
    'ocean_proximity_ISLAND': [0.0, 0.0], # Added the missing column
    'ocean_proximity_NEAR BAY': [0.0, 0.0],
    'ocean_proximity_NEAR OCEAN': [0.0, 0.0],
    'rooms_per_household': [6.0, 5.5],
    'bedrooms_per_room': [0.15, 0.20]
})

# Scale the new data using the same scaler fitted on the training data
new_data_scaled = scaler.transform(new_data)

# Make predictions
new_predictions = forest_model.predict(new_data_scaled)

# Display the predictions
print("Predictions for new data:")
print(new_predictions)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/tmp/ipython-input-3363194403.py in <cell line: 0>()
     15 
     16 # Scale the new data using the same scaler fitted on the training
data
---> 17 new_data_scaled = scaler.transform(new_data)
     18 
     19 # Make predictions
```

3 frames

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py in
_check_feature_names(estimator, X, reset)
   2775             message += "Feature names must be in the same order as
they were in fit.\n"
   2776 
-> 2777         raise ValueError(message)
   2778 
   2779 

ValueError: The feature names should match those that were passed during fit.
Feature names unseen at fit time:
```

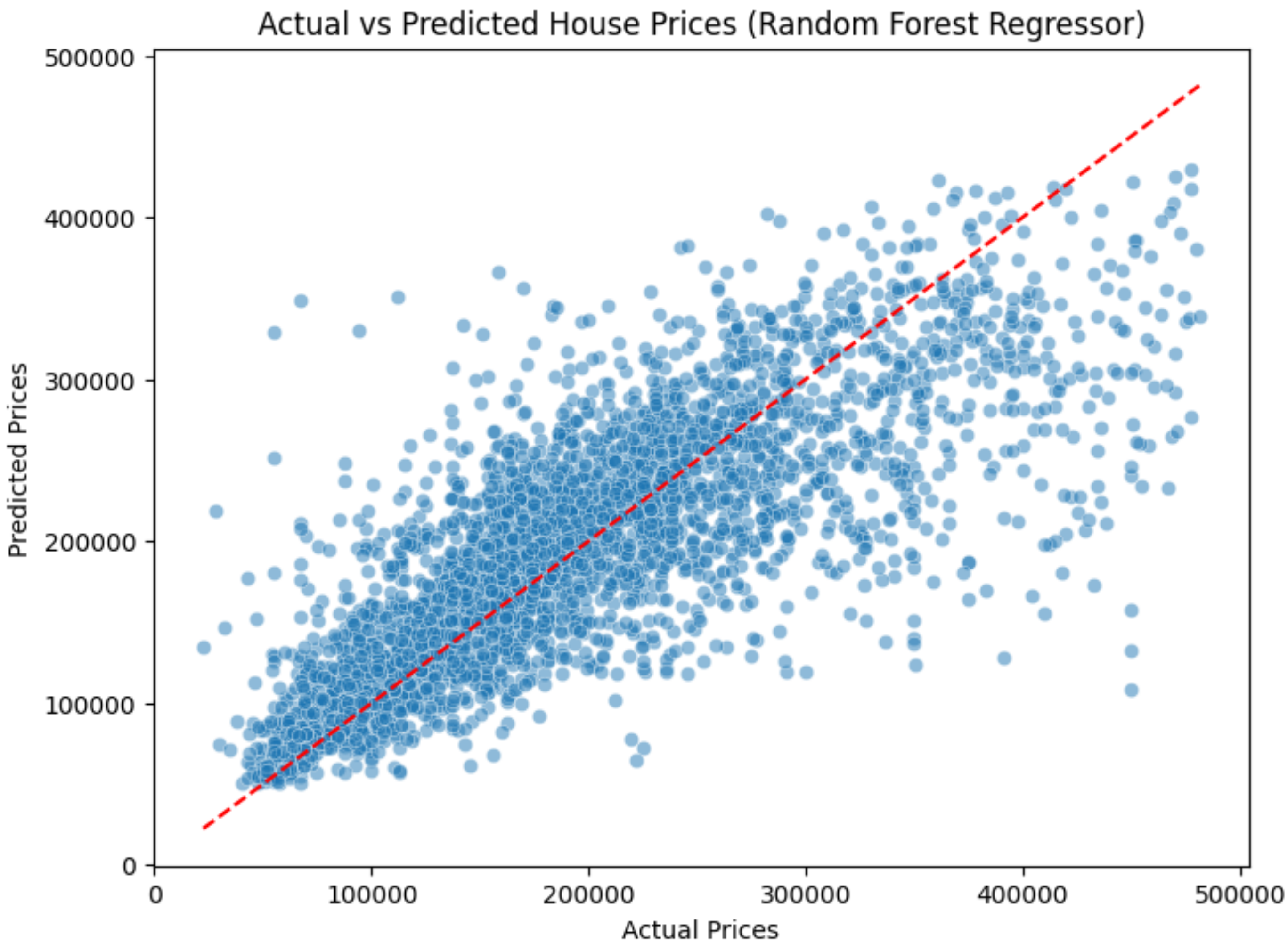
Next steps: Explain error

Compare Predictions with Actual Values (Random Forest Regressor)

```
# Make predictions on the testing data (already done in evaluation step)
# y_pred_forest = forest_model.predict(X_test_scaled)

# Compare predictions with actual values
comparison_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred_forest})
print("Comparison of Actual vs Predicted Values:")
print(comparison_df.head())

# Visualize the comparison (Actual vs Predicted)
plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_test, y=y_pred_forest, alpha=0.5)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', lin
plt.title("Actual vs Predicted House Prices (Random Forest Regressor)")
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
```

```
plt.show()
```

```
Comparison of Actual vs Predicted Values:
         Actual  Predicted
3429   325000.0   266329.0
14820  187500.0   133900.0
1843   153800.0    97091.0
12815   88600.0   124005.0
7137   194600.0   206307.0
```



Feature Importance (Random Forest Regressor)

```
# Get feature importances from the Random Forest model
feature_importances = forest_model.feature_importances_
```

```
# Create a DataFrame for feature importances
feature_importance_df = pd.DataFrame({'Feature': X_train.columns, 'Importance': f

# Sort features by importance
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascend

# Print the feature importances
print("Feature Importances (Random Forest Regressor):")
print(feature_importance_df)

# Visualize feature importances
plt.figure(figsize=(10, 6))
plt.barh(feature_importance_df['Feature'], feature_importance_df['Importance'], c
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Feature Importance (Random Forest Regressor)')
plt.gca().invert_yaxis()  # Invert y-axis to show most important features at the
plt.show()
```

```
Feature Importances (Random Forest Regressor):
                        Feature  Importance
1                 median_income    0.488776
3        ocean_proximity_INLAND    0.162334
0                      latitude    0.154167
7             bedrooms_per_room    0.095974
6           rooms_per_household    0.083266
5    ocean_proximity_NEAR OCEAN    0.007478
2     ocean_proximity_<1H OCEAN    0.005835
4      ocean_proximity_NEAR BAY    0.002170
```