Project Title: Predicting House Prices **12/09/2023**

Objective: Develop a machine learning model to forecast house prices by leveraging a range of property features.

Methodology:

1. Data Collection:

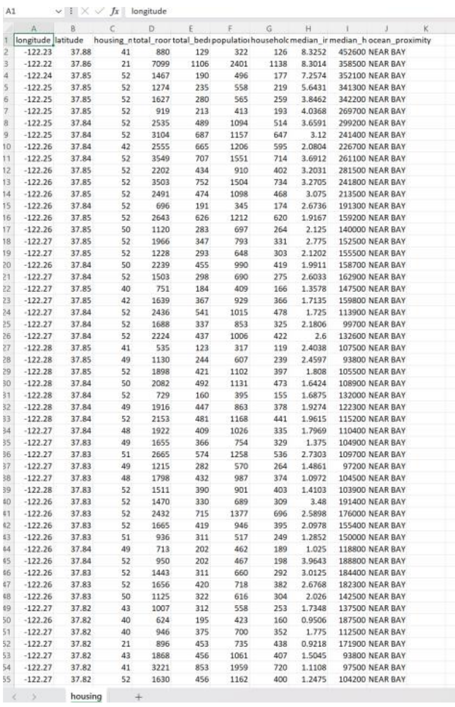
Imported California Housing Price Dataset from Kaggle.

https://www.kaggle.com/datasets/camnugent/california-housing-prices?select=housing.csv

The dataset contains information about houses within a specific California district, including summary statistics derived from the 1990 census data. It's important to note that the data has not undergone cleaning, and certain preprocessing steps are necessary. The columns are as follows, and their names are largely self-explanatory:

Longitude, Latitude, Housing Median Age, Total Rooms, Total Bedrooms, Population, Households, Median Income, Median House Value, Ocean Proximity

Raw data:

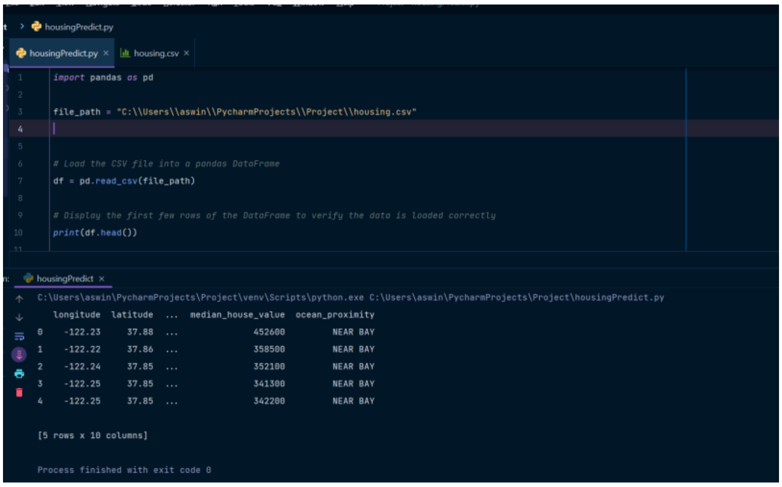


	longitude	latitude	housing_n	total_roor	total_bedı	populatioı	householc	median_ir	median_h	ocean_proximity
2	-122.23	37.88	41	880	129	322	126	8.3252	452600	NEAR BAY
3	-122.22	37.86	21	7099	1106	2401	1138	8.3014	358500	NEAR BAY
4	-122.24	37.85	52	1467	190	496	177	7.2574	352100	NEAR BAY
5	-122.25	37.85	52	1274	235	558	219	5.6431	341300	NEAR BAY
6	-122.25	37.85	52	1627	280	565	259	3.8462	342200	NEAR BAY
7	-122.25	37.85	52	919	213	413	193	4.0368	269700	NEAR BAY
8	-122.25	37.84	52	2535	489	1094	514	3.6591	299200	NEAR BAY
9	-122.25	37.84	52	3104	687	1157	647	3.12	241400	NEAR BAY
10	-122.26	37.84	42	2555	665	1206	595	2.0804	226700	NEAR BAY
11	-122.25	37.84	52	3549	707	1551	714	3.6912	261100	NEAR BAY
12	-122.26	37.85	52	2202	434	910	402	3.2031	281500	NEAR BAY
13	-122.26	37.85	52	3503	752	1504	734	3.2705	241800	NEAR BAY
14	-122.26	37.85	52	2491	474	1098	468	3.075	213500	NEAR BAY
15	-122.26	37.84	52	696	191	345	174	2.6736	191300	NEAR BAY
16	-122.26	37.85	52	2643	626	1212	620	1.9167	159200	NEAR BAY
17	-122.26	37.85	50	1120	283	697	264	2.125	140000	NEAR BAY
18	-122.27	37.85	52	1966	347	793	331	2.775	152500	NEAR BAY
19	-122.27	37.85	52	1228	293	648	303	2.1202	155500	NEAR BAY
20	-122.26	37.84	50	2239	455	990	419	1.9911	158700	NEAR BAY
21	-122.27	37.84	52	1503	298	690	275	2.6033	162900	NEAR BAY
22	-122.27	37.85	40	751	184	409	166	1.3578	147500	NEAR BAY
23	-122.27	37.85	42	1639	367	929	366	1.7135	159800	NEAR BAY
24	-122.27	37.84	52	2436	541	1015	478	1.725	113900	NEAR BAY
25	-122.27	37.84	52	1688	337	853	325	2.1806	99700	NEAR BAY
26	-122.27	37.84	52	2224	437	1006	422	2.6	132600	NEAR BAY
27	-122.28	37.85	41	535	123	317	119	2.4038	107500	NEAR BAY
28	-122.28	37.85	49	1130	244	607	239	2.4597	93800	NEAR BAY
29	-122.28	37.85	52	1898	421	1102	397	1.808	105500	NEAR BAY
30	-122.28	37.84	50	2082	492	1131	473	1.6424	108900	NEAR BAY
31	-122.28	37.84	52	729	160	395	155	1.6875	132000	NEAR BAY
32	-122.28	37.84	49	1916	447	863	378	1.9274	122300	NEAR BAY
33	-122.28	37.84	52	2153	481	1168	441	1.9615	115200	NEAR BAY
34	-122.27	37.84	48	1922	409	1026	335	1.7969	110400	NEAR BAY
35	-122.27	37.83	49	1655	366	754	329	1.375	104900	NEAR BAY
36	-122.27	37.83	51	2665	574	1258	536	2.7303	109700	NEAR BAY
37	-122.27	37.83	49	1215	282	570	264	1.4861	97200	NEAR BAY
38	-122.27	37.83	48	1798	432	987	374	1.0972	104500	NEAR BAY
39	-122.28	37.83	52	1511	390	901	403	1.4103	103900	NEAR BAY
40	-122.26	37.83	52	1470	330	689	309	3.48	191400	NEAR BAY
41	-122.26	37.83	52	2432	715	1377	696	2.5898	176000	NEAR BAY
42	-122.26	37.83	52	1665	419	946	395	2.0978	155400	NEAR BAY
43	-122.26	37.83	51	936	311	517	249	1.2852	150000	NEAR BAY
44	-122.26	37.84	49	713	202	462	189	1.025	118800	NEAR BAY
45	-122.26	37.84	52	950	202	467	198	3.9643	188800	NEAR BAY
46	-122.26	37.83	52	1443	311	660	292	3.0125	184400	NEAR BAY
47	-122.26	37.83	52	1656	420	718	382	2.6768	182300	NEAR BAY
48	-122.26	37.83	50	1125	322	616	304	2.026	142500	NEAR BAY
49	-122.27	37.82	43	1007	312	558	253	1.7348	137500	NEAR BAY
50	-122.26	37.82	40	624	195	423	160	0.9506	187500	NEAR BAY
51	-122.27	37.82	40	946	375	700	352	1.775	112500	NEAR BAY
52	-122.27	37.82	21	896	453	735	438	0.9218	171900	NEAR BAY
53	-122.27	37.82	43	1868	456	1061	407	1.5045	93800	NEAR BAY
54	-122.27	37.82	41	3221	853	1959	720	1.1108	97500	NEAR BAY
55	-122.27	37.82	52	1630	456	1162	400	1.2475	104200	NEAR BAY

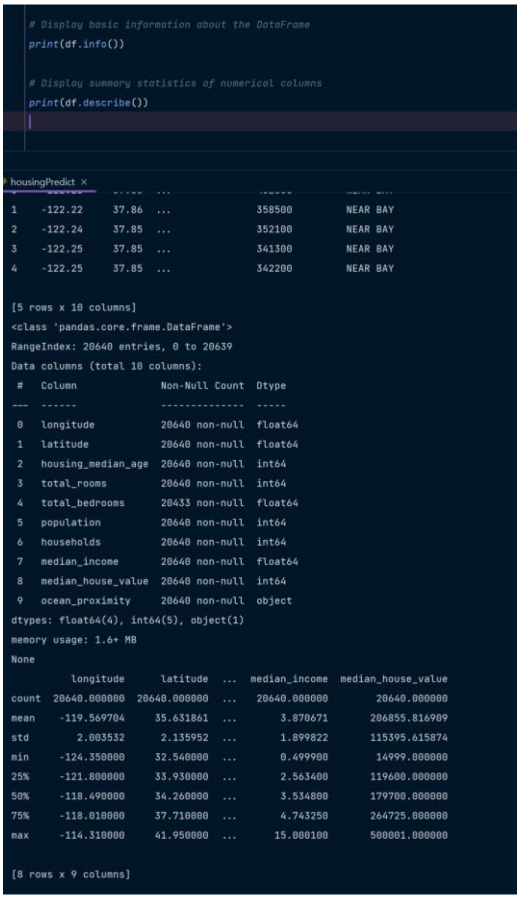
2. Data Processing:

Step 1: Glanced through the dataset for any possible errors, unnecessary variables. The dataset was pretty clean and did not need much adjustments on initially.

Step 2: Loaded the dataset into Python using pandas library.



Step 3: Used pandas functions to explore the data and calculate summary statistics to make some sense of the dataset.



Step 4:

Imported Necessary Libraries:

- pandas: A powerful data manipulation library.
- StandardScaler: A class for standardizing numerical features.
- OneHotEncoder: A class for one-hot encoding categorical variables.
- SimpleImputer: A class for imputing missing values.
- ColumnTransformer: A class for applying transformers to columns of an array or DataFrame.
- Pipeline: A class for sequentially applying a list of transformations.

```
import pandas as pd
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
```

Handling Missing Values:

- Defined lists of numerical and categorical features.
- Created a numeric_transformer pipeline:
- Imputed missing numerical values using the mean.
- Scaled the numerical features using StandardScaler.
- Created a categorical_transformer pipeline:
- Imputed missing categorical values using the most frequent value.
- Applied one-hot encoding using OneHotEncoder.
- Used ColumnTransformer to apply different transformers to different columns.

```
# Handling Missing Values
# Use SimpleImputer to fill missing values
numeric_features = ['longitude', 'latitude', 'housing_median_age', 'total_rooms', 'total_bedrooms', 'population', 'households', 'median_income']
categorical_features = ['ocean_proximity']
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())
])
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])
```

Apply the Preprocessing Steps and Display the Preprocessed DataFrame:

- Used the fit_transform method to apply the preprocessing steps to the DataFrame and create a new DataFrame (df_preprocessed).

```
# Apply the preprocessing steps
df_preprocessed = pd.DataFrame(preprocessor.fit_transform(df))
# Display the preprocessed DataFrame
print(df_preprocessed.head())
```

```
          0         1         2         3         4  ...    8    9   10   11   12
0 -1.327835  1.052548  0.982143 -0.804819 -0.975228  ...  0.0  0.0  0.0  1.0  0.0
1 -1.322844  1.043185 -0.607019  2.045890  1.355088  ...  0.0  0.0  0.0  1.0  0.0
2 -1.332827  1.038503  1.856182 -0.535746 -0.829732  ...  0.0  0.0  0.0  1.0  0.0
3 -1.337818  1.038503  1.856182 -0.624215 -0.722399  ...  0.0  0.0  0.0  1.0  0.0
4 -1.337818  1.038503  1.856182 -0.462404 -0.615066  ...  0.0  0.0  0.0  1.0  0.0

[5 rows x 13 columns]
```

3. **Exploratory Data Analysis (EDA):**

- Used some basic pandas functions to explore the dataset in previous steps.

Step 1: Imported Seaborn and matplotlib Library

```
import seaborn as sns
import matplotlib.pyplot as plt
```

Step 2: Set the style for Seaborn plots to "whitegrid" for a visually appealing background.

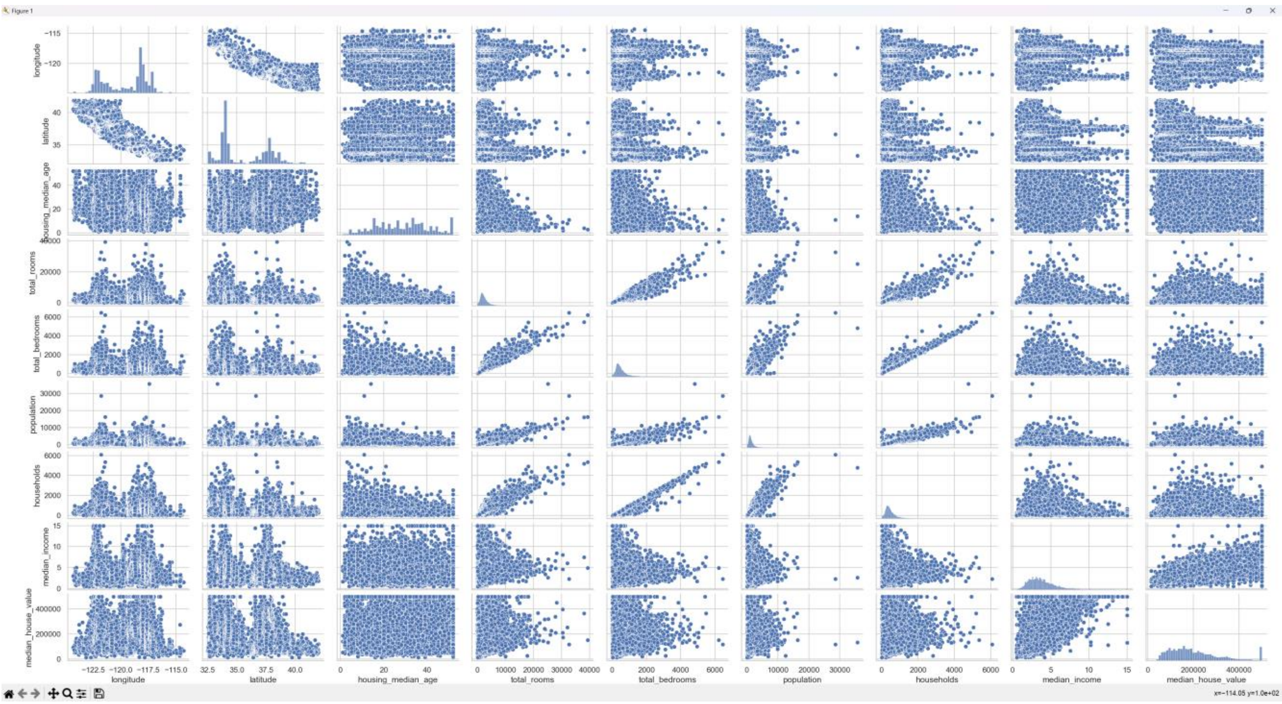
```
# Set the style for seaborn plots
sns.set(style="whitegrid")
```

Step 3: Created a list of numerical features to explore.

```
# Visualize the distribution of numerical features
numerical_features = ['longitude', 'latitude', 'housing_median_age', 'total_rooms', 'total_bedrooms', 'population', 'households', 'median_income', 'median_house_value']
```

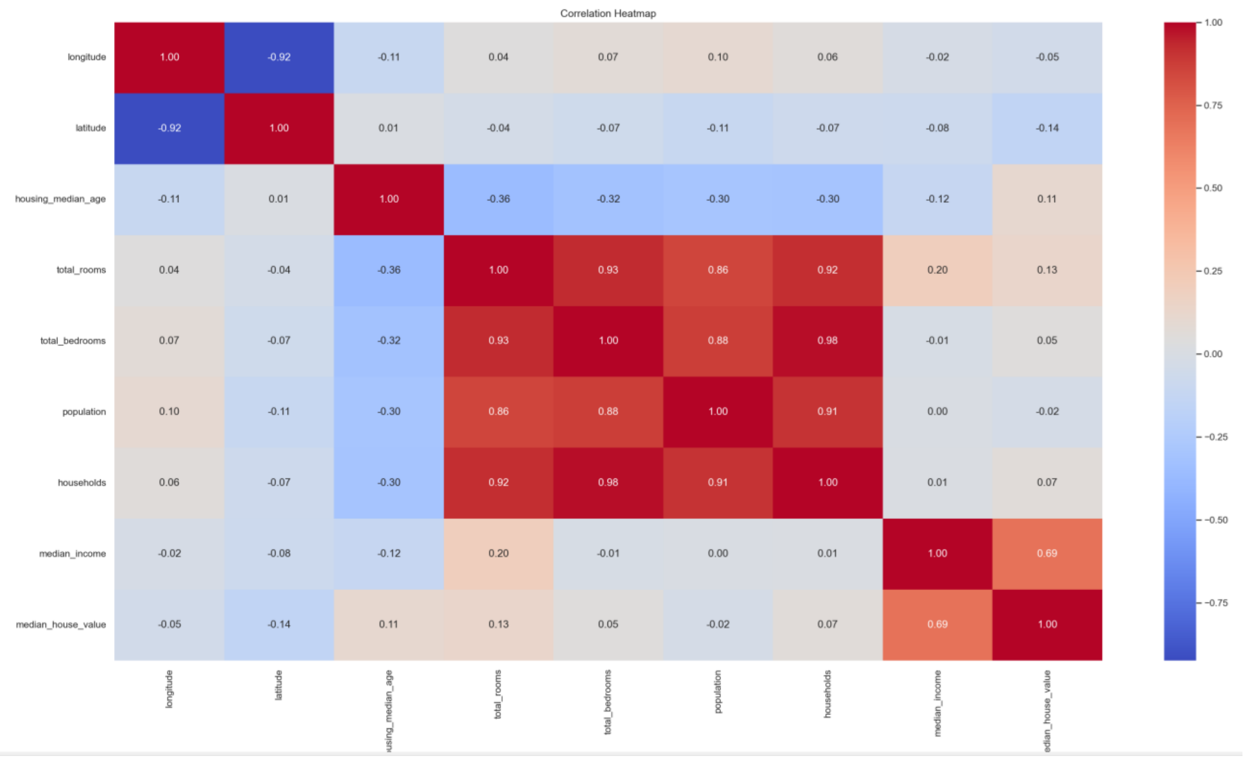
Step 4: Created a pairplot to visualize pairwise relationships between numerical features. This helps identify patterns, trends, and potential outliers.

```
# Pairplot for numerical features
sns.pairplot(df[numerical_features])
plt.show()
```



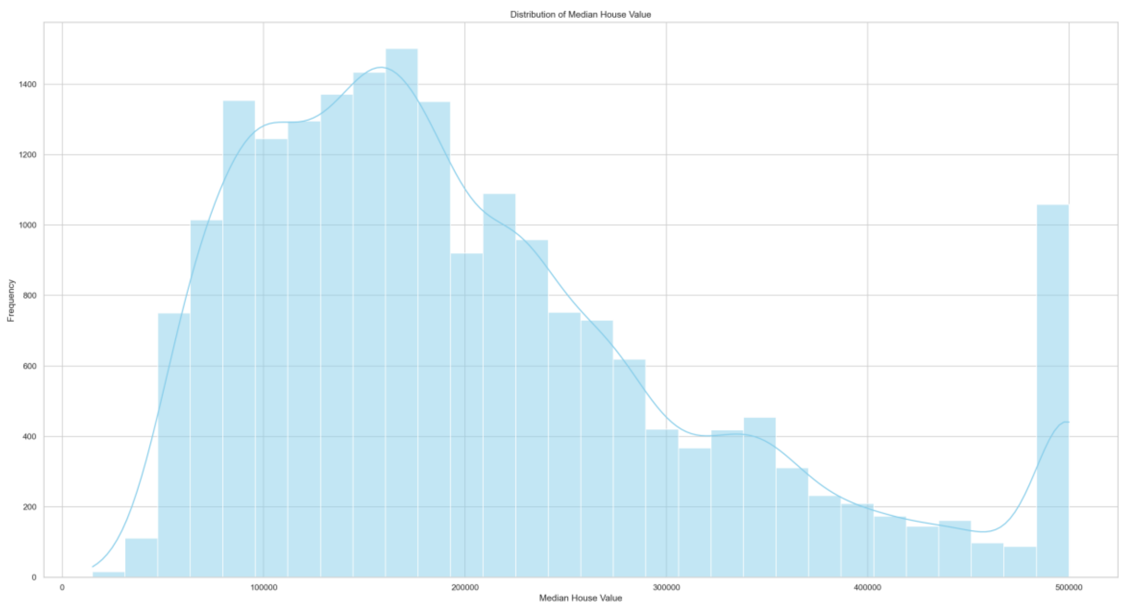
Step 5: Calculated the correlation matrix between numerical features and created a heatmap to visualize correlations. This helps identify strong correlations, which can be potential predictors.

```
# Correlation heatmap
correlation_matrix = df[numerical_features].corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation Heatmap")
plt.show()
```



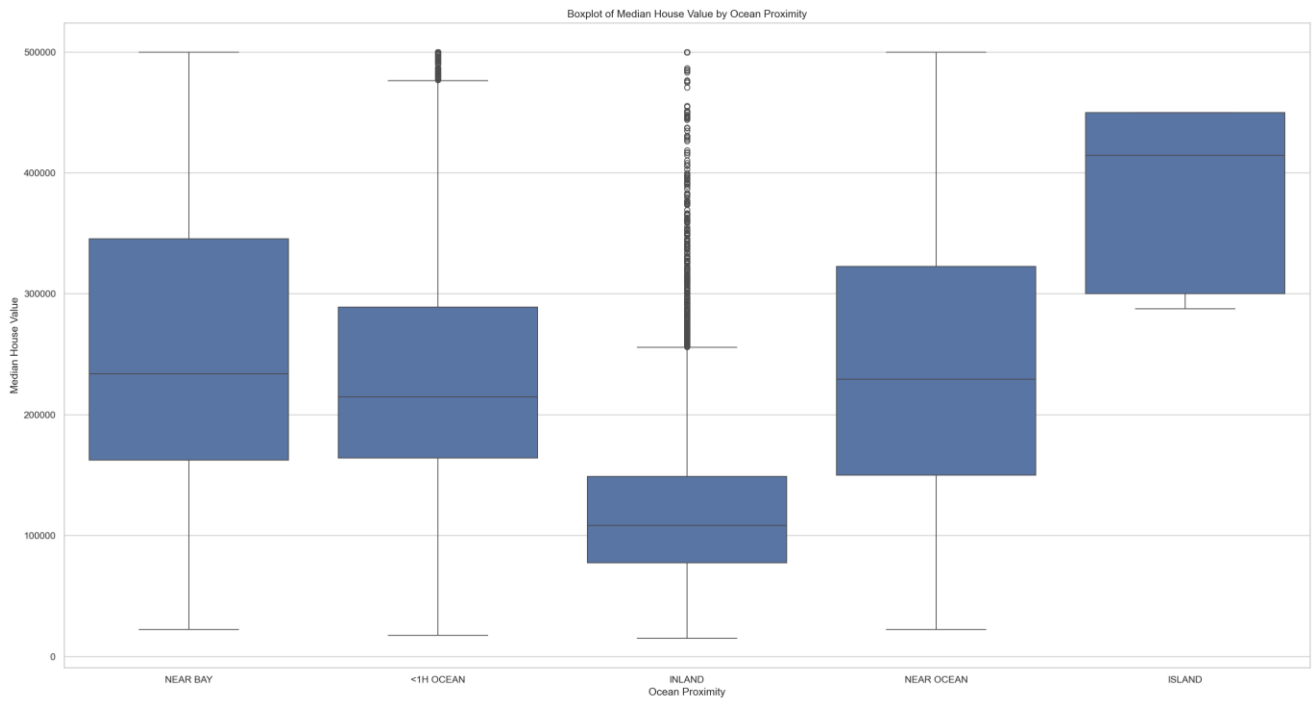
Step 6: Created a histogram to visualize the distribution of the target variable 'median_house_value'. This helps understand the spread and central tendency of house prices.

```
# Visualize the distribution of the target variable
plt.figure(figsize=(10, 6))
sns.histplot(df['median_house_value'], kde=True, bins=30, color='skyblue')
plt.title("Distribution of Median House Value")
plt.xlabel("Median House Value")
plt.ylabel("Frequency")
plt.show()
```



Step 7: Created a boxplot to show how the target variable varies with the categorical variable 'ocean_proximity'. This helps identify if certain categories have a significant impact on house prices.

```
# Boxplot for categorical variable 'ocean_proximity' vs 'median_house_value'
plt.figure(figsize=(12, 8))
sns.boxplot(x='ocean_proximity', y='median_house_value', data=df)
plt.title("Boxplot of Median House Value by Ocean Proximity")
plt.xlabel("Ocean Proximity")
plt.ylabel("Median House Value")
plt.show()
```



4. **Feature Engineering:**

Step 1: Created three new features 'Rooms per Household", "Bedrooms per room" and "Population per Household.

```
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']
```

Step 2: Updated the data preprocessing pipeline as follows to enhance the predictive power of our model by providing it with more relevant information:

```
# Update the list of numerical features
numerical_features = ['longitude', 'latitude', 'housing_median_age', 'total_rooms', 'total_bedrooms', 'population', 'households', 'median_income', 'rooms_per_household', 'bedrooms_per_room', 'population_per_household' 'median_house_value']
# Update the numeric_transformer in the ColumnTransformer
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())
])
# Update the ColumnTransformer
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)
    ])
# Apply the preprocessing steps
df_preprocessed = pd.DataFrame(preprocessor.fit_transform(df))
```

Step 3: Imported the numpy library and applied transformations to existing features.

```
#apply transformation to existing feature
df['log_median_income'] = np.log1p(df['median_income'])
```

Step 4: Examined the correlation between features and the target variable. Keep features with high correlation.

```
#correlation analysis
correlation_with_target = df_preprocessed.corr()['median_house_value'].abs().sort_values(ascending=False)
selected_features = correlation_with_target[correlation_with_target > 0.1].index.tolist()
```

Step 5: Used statistical tests "SelectKBest" from scikit-learn to select the k most important features.

```
#SelectKBest (Statistical Tests)
k_best_selector = SelectKBest(score_func=f_regression, k=5)

# Extract features from df_preprocessed
X_selected = k_best_selector.fit_transform(df_preprocessed.drop('median_house_value', axis=1), df_preprocessed['median_house_value'])

# Get the selected feature indices
selected_feature_indices = k_best_selector.get_support(indices=True)

# Get the selected feature names from the preprocessed DataFrame columns
selected_features = df_preprocessed.drop('median_house_value', axis=1).columns[selected_feature_indices].tolist()

# Ensure 'median_house_value' is included in the selected features list
selected_features.append('median_house_value')
```

Step 6: Removed the least important features recursively using RFE available in scikit-learn.

```
# Recursive Feature Elimination (RFE)
estimator = RandomForestRegressor()
rfe_selector = RFE(estimator, n_features_to_select=3, step=2)

print("Starting RFE feature selection...(please allow few minutes for this process)")
# Extract features from df_preprocessed
X_rfe = rfe_selector.fit_transform(df_preprocessed.drop('median_house_value', axis=1), df_preprocessed['median_house_value'])
print("RFE feature selection completed.")

# Get the selected feature indices
selected_feature_indices_rfe = rfe_selector.get_support(indices=True)

# RFE feature selection
selected_features_rfe = df_preprocessed.drop('median_house_value', axis=1).columns[rfe_selector.support_]
# Append the target variable to the selected features
selected_features_rfe = selected_features_rfe.append(pd.Index(['median_house_value']))
```

Step 7: Used LASSO Regression to shrink coefficients (select features with non-zero coefficients) and effectively perform feature selection.

```
# LASSO Regression (L1 Regularization) using features from RFE
lasso = Lasso(alpha=0.1)
lasso.fit(df_preprocessed[selected_features_rfe.drop('median_house_value')], df_preprocessed['median_house_value'])
selected_features_lasso = selected_features_rfe.drop('median_house_value').tolist()
```

5. **Model Building:**

Step 1: Imported the following libraries for splitting the dataset into a training set and a testing set and then implement machine learning algorithms such as Linear Regression, Decision Trees, Random Forest, and Gradient Boosting

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

Step 2: Used train_test_split from scikit-learn to split the preprocessed dataset into features (X) and the target variable (y). Then further splited these into training and testing sets. The random_state is set for reproducibility.

```
# Split the dataset into features (X) and target variable (y)
X = df_preprocessed.drop('median_house_value', axis=1)
y = df_preprocessed['median_house_value']

# Split the data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Step 3: Imported regression models from scikit-learn (Linear Regression, Decision Tree, Random Forest, Gradient Boosting) and train them using the training data.

```
# Linear Regression
linear_reg_model = LinearRegression()
linear_reg_model.fit(X_train, y_train)
linear_reg_pred = linear_reg_model.predict(X_test)

# Decision Tree
decision_tree_model = DecisionTreeRegressor()
decision_tree_model.fit(X_train, y_train)
decision_tree_pred = decision_tree_model.predict(X_test)

# Random Forest
random_forest_model = RandomForestRegressor()
random_forest_model.fit(X_train, y_train)
random_forest_pred = random_forest_model.predict(X_test)

# Gradient Boosting
gradient_boosting_model = GradientBoostingRegressor()
gradient_boosting_model.fit(X_train, y_train)
gradient_boosting_pred = gradient_boosting_model.predict(X_test)
```

6. Model Evaluation:

Defined a function evaluate_model to calculate Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R-squared for each model and print the results.

```
# Evaluate the models
def evaluate_model(name, y_true, y_pred):
    mse = mean_squared_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)
    print(f"{name} Model:")
    print(f"Mean Squared Error: {mse}")
    print(f"R-squared: {r2}")
    print()

# Evaluate Linear Regression
evaluate_model("Linear Regression", y_test, linear_reg_pred)

# Evaluate Decision Tree
evaluate_model("Decision Tree", y_test, decision_tree_pred)

# Evaluate Random Forest
evaluate_model("Random Forest", y_test, random_forest_pred)

# Evaluate Gradient Boosting
evaluate_model("Gradient Boosting", y_test, gradient_boosting_pred)
```

```
Starting RFE feature selection...(please allow few minutes for this process)
RFE feature selection completed.
Building Model...
Linear Regression Model:
Mean Absolute Error: 50701.77903132994
Root Mean Squared Error: 70031.41991955665
R-squared: 0.6257351821159705

Decision Tree Model:
Mean Absolute Error: 44051.59084302326
Root Mean Squared Error: 69699.34835099982
R-squared: 0.6292761083585198

Random Forest Model:
Mean Absolute Error: 31698.508582848834
Root Mean Squared Error: 49087.942385564325
R-squared: 0.8161164851844057

Gradient Boosting Model:
Mean Absolute Error: 38368.283348536745
Root Mean Squared Error: 55967.21074708723
R-squared: 0.7609655664131871
```

Conclusion:

In summary, this project successfully navigated the entire machine learning pipeline for predicting residential property prices. The dataset underwent thorough preprocessing, addressing missing values and incorporating feature engineering to enhance model understanding. Utilizing various visualization techniques provided valuable insights into feature relationships. Feature selection was carried out using correlation analysis, SelectKBest, Recursive Feature Elimination (RFE), and LASSO Regression. Four machine learning models—Linear Regression, Decision Tree, Random Forest, and Gradient Boosting—were trained and evaluated, demonstrating the model's flexibility. The evaluation metrics, including Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R-squared, provided a comprehensive assessment of predictive performance. The code offers adaptability for future improvements, tuning, and exploration of advanced modeling techniques to further enhance the predictive capabilities of the system.