

Linear Regression Analysis Tutorial - Polynomial Regression

Creator: Muhammad Bilal Alam

What is Polynomial Regression?

Polynomial regression is a type of regression analysis in which the relationship between the independent variable x and the dependent variable y is modeled as an n th degree polynomial. The formula for a polynomial regression model of degree n can be written as:

$$y = \beta_0 + \beta_1x + \beta_2x^2 + \dots + \beta_nx^n + \epsilon$$

where:

- y is the dependent variable
- x is the independent variable
- $\beta_0, \beta_1, \beta_2, \dots, \beta_n$ are the coefficients of the polynomial regression model
- ϵ is the error term or the residual
- n is the degree of the polynomial

The goal of polynomial regression is to find the values of the coefficients $\beta_0, \beta_1, \beta_2, \dots, \beta_n$ that minimize the sum of squared errors between the predicted values of y and the actual values of y .

The California Housing Dataset for Multiple Linear Regression

The California Housing Dataset contains information on the median income, housing age, and other features for census tracts in California. The dataset was originally published by Pace, R. Kelley and Ronald Barry in their 1997 paper "Sparse Spatial Autoregressions" and is available in the `sklearn.datasets` module.

The dataset consists of 20,640 instances, each representing a census tract in California.

There are eight features in the dataset, including:

- **MedInc:** Median income in the census tract.
- **HouseAge:** Median age of houses in the census tract.
- **AveRooms:** Average number of rooms per dwelling in the census tract.
- **AveBedrms:** Average number of bedrooms per dwelling in the census tract.
- **Population:** Total number of people living in the census tract.
- **AveOccup:** Average number of people per household in the census tract.
- **Latitude:** Latitude of the center of the census tract.
- **Longitude:** Longitude of the center of the census tract.

Step 1: Import the necessary libraries

```
In [34]: import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import GridSearchCV
import warnings
warnings.filterwarnings('ignore')
```

Step 2: Load the dataset

```
In [2]: # Load the California Housing Dataset from seaborn

california = fetch_california_housing()

# Convert the data to a pandas dataframe
california_df = pd.DataFrame(data=california.data, columns=california.feature_names)

# Add the target variable to the dataframe
california_df['MedHouseVal'] = california.target

# Print the first 5 rows of the dataframe
california_df.head()
```

Out[2]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	Medi
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	

Step 3: Do Data Preprocessing along with Data Exploratory Analysis

Step 3(a): Check Shape of Dataframe

Checking the Shape of Dataframe tell hows how many rows and columns we have in the dataset

```
In [3]: # Print the shape of the dataframe
print("Data shape:", california_df.shape)
```

Data shape: (20640, 9)

Step 3(b): Check Info of Dataframe

This is very useful to quickly get an overview of the structure and properties of a dataset, and to check for any missing or null values that may need to be addressed before performing any analysis or modeling.

In [4]: `california_df.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   MedInc       20640 non-null   float64
 1   HouseAge     20640 non-null   float64
 2   AveRooms     20640 non-null   float64
 3   AveBedrms    20640 non-null   float64
 4   Population   20640 non-null   float64
 5   AveOccup     20640 non-null   float64
 6   Latitude     20640 non-null   float64
 7   Longitude    20640 non-null   float64
 8   MedHouseVal  20640 non-null   float64
dtypes: float64(9)
memory usage: 1.4 MB
```

Step 3(c): Show Descriptive Statistics of each numerical column

Looking at descriptive statistics in machine learning is important because it gives an overview of the dataset's distribution and key characteristics. Some of the reasons why we should look at descriptive statistics include:

- **Understanding the distribution of data:** Descriptive statistics provide information about the central tendency and the spread of the data. This information is useful in determining the type of distribution and whether the data is skewed or symmetrical.
- **Identifying outliers:** Descriptive statistics help to identify any extreme values or outliers in the dataset. These outliers can have a significant impact on the analysis and should be investigated further.

From the descriptive statistics, we can observe the following:

- **Outliers:** The 'AveRooms', 'AveBedrms', 'Population', and 'AveOccup' columns have high maximum values, indicating the presence of outliers in the data. These outliers may need to be treated or removed before model selection. We will create visuals to see them more clearly.
- **Distribution:** The 'MedInc', 'HouseAge', and 'MedHouseVal' columns appear to be normally distributed, as the mean and median values are close to each other, and the standard deviation is not very high. The 'Latitude' column is skewed to the left, as the mean is less than the median. The 'Longitude' column is skewed to the right, as the mean is greater than the median.

In [5]: `california_df.describe().T`

Out[5]:

	count	mean	std	min	25%	50%	75%
MedInc	20640.0	3.870671	1.899822	0.499900	2.563400	3.534800	4.7432
HouseAge	20640.0	28.639486	12.585558	1.000000	18.000000	29.000000	37.0000
AveRooms	20640.0	5.429000	2.474173	0.846154	4.440716	5.229129	6.0523
AveBedrms	20640.0	1.096675	0.473911	0.333333	1.006079	1.048780	1.0995
Population	20640.0	1425.476744	1132.462122	3.000000	787.000000	1166.000000	1725.0000
AveOccup	20640.0	3.070655	10.386050	0.692308	2.429741	2.818116	3.2822
Latitude	20640.0	35.631861	2.135952	32.540000	33.930000	34.260000	37.7100
Longitude	20640.0	-119.569704	2.003532	-124.350000	-121.800000	-118.490000	-118.0100
MedHouseVal	20640.0	2.068558	1.153956	0.149990	1.196000	1.797000	2.6472

Step 3(d): Check for missing values in the Dataframe

This is important because most machine learning algorithms cannot handle missing data and will throw an error if missing values are present. Therefore, it is necessary to check for missing values and impute or remove them before fitting the data into a machine learning model. This helps to ensure that the model is trained on complete and accurate data, which leads to better performance and more reliable predictions.

Here we have no missing values so lets move on.

In [6]:

```
# Check for missing values
print("Missing values:\n", california_df.isnull().sum())
```

```
Missing values:
MedInc      0
HouseAge    0
AveRooms   0
AveBedrms  0
Population  0
AveOccup   0
Latitude   0
Longitude  0
MedHouseVal 0
dtype: int64
```

Step 3(e): Check for duplicate values in the Dataframe

Checking for duplicate values in machine learning is important because it can affect the accuracy of your model. Duplicate values can skew your data and lead to overfitting, where your model is too closely fit to the training data and does not generalize well to new data.

We have no duplicate values so thats good.

In [7]:

```
california_df.duplicated().sum()
```

Out[7]:

```
0
```

Step 3(f)(i): Check for Outliers in the Dataframe

We should check for outliers as they can have a negative impact on machine learning algorithms as they can skew the results of the analysis. Outliers can significantly alter the mean, standard deviation, and other statistical measures, which can misrepresent the true characteristics of the data. Linear regression models, are sensitive to outliers and can produce inaccurate results if the outliers are not properly handled or removed. Therefore, it is important to identify and handle outliers appropriately to ensure the accuracy and reliability of the models.

Here in the plots we can clearly see very high outliers on the right hand side. So we need to deal with them appropriately

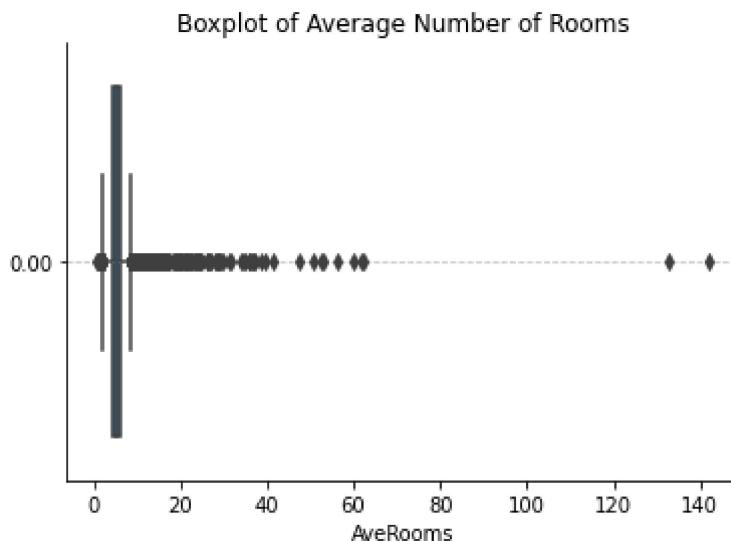
```
In [8]: # Create a boxplot of the 'AveRooms' column
ax = sns.boxplot(x=california_df['AveRooms'])

# Set the title and axes labels
ax.set_title('Boxplot of Average Number of Rooms')
ax.set_xlabel('AveRooms')
ax.set_ylabel('')

# Customize the y-axis tick Labels to display values in millions
yticks = ax.get_yticks() / 1000000
ax.set_yticklabels(['{:,.2f}'.format(ytick) for ytick in yticks])

# Add grid Lines and remove top and right spines
ax.grid(axis='y', linestyle='--', alpha=0.7)
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)

# Show the plot
plt.show()
```



```
In [9]: # Create a boxplot of the 'AveBedrms' column
ax = sns.boxplot(x=california_df['AveBedrms'])

# Set the title and axes labels
ax.set_title('Boxplot of Average Number of Bedrooms')
ax.set_xlabel('AveBedrms')
ax.set_ylabel('')

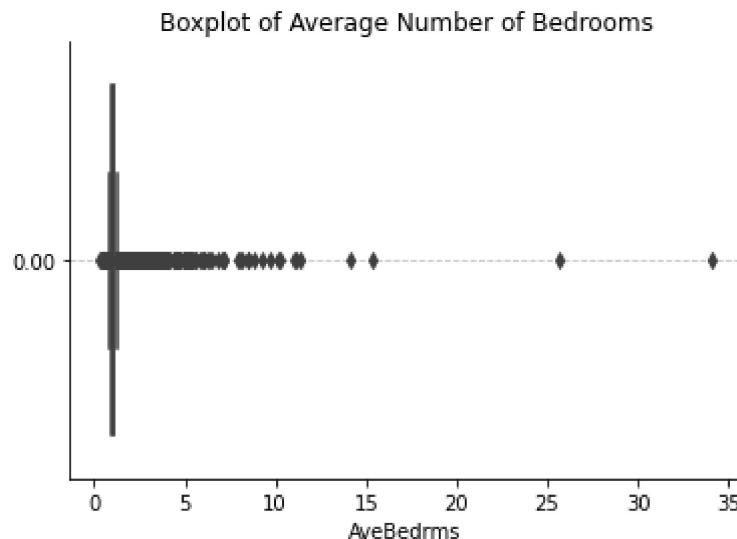
# Customize the y-axis tick Labels to display values in millions
yticks = ax.get_yticks() / 1000000
ax.set_yticklabels(['{:,.2f}'.format(ytick) for ytick in yticks])
```

```

# Add grid Lines and remove top and right spines
ax.grid(axis='y', linestyle='--', alpha=0.7)
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)

# Show the plot
plt.show()

```



In [10]:

```

# Create a boxplot of the 'Population' column
ax = sns.boxplot(x=california_df['Population'])

# Set the title and axes labels
ax.set_title('Boxplot of Populations')
ax.set_xlabel('Population')
ax.set_ylabel('')

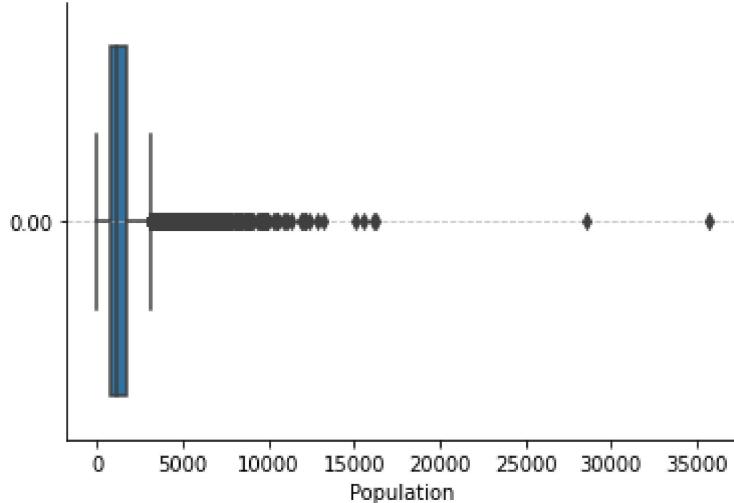
# Customize the y-axis tick Labels to display values in millions
yticks = ax.get_yticks() / 1000000
ax.set_yticklabels(['{:,.2f}'.format(ytick) for ytick in yticks])

# Add grid Lines and remove top and right spines
ax.grid(axis='y', linestyle='--', alpha=0.7)
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)

# Show the plot
plt.show()

```

Boxplot of Populations



```
In [11]: # Create a boxplot of the 'AveOccup' column
ax = sns.boxplot(x=california_df['AveOccup'])

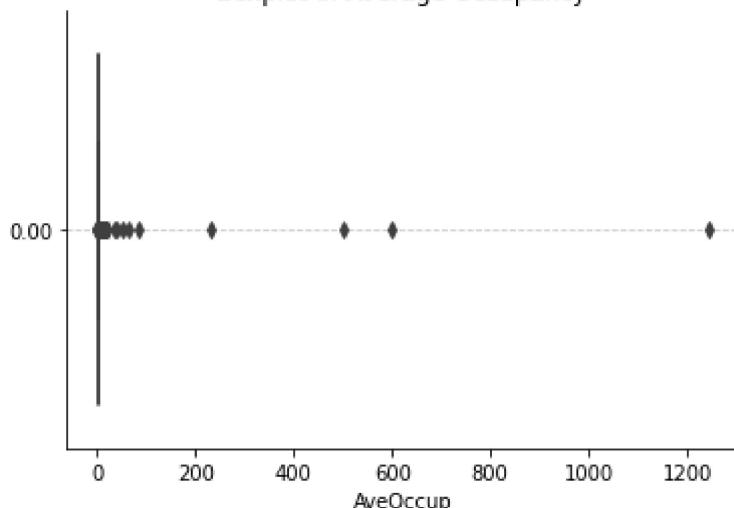
# Set the title and axes labels
ax.set_title('Boxplot of Average Occupancy')
ax.set_xlabel('AveOccup')
ax.set_ylabel('')

# Customize the y-axis tick Labels to display values in millions
yticks = ax.get_yticks() / 1000000
ax.set_yticklabels(['{:,.2f}'.format(ytick) for ytick in yticks])

# Add grid lines and remove top and right spines
ax.grid(axis='y', linestyle='--', alpha=0.7)
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)

# Show the plot
plt.show()
```

Boxplot of Average Occupancy



```
In [12]: # Create a boxplot of the 'AveOccup' column
ax = sns.boxplot(x=california_df['AveOccup'])

# Set the title and axes labels
ax.set_title('Boxplot of Average Occupancy')
ax.set_xlabel('AveOccup')
ax.set_ylabel('')
```

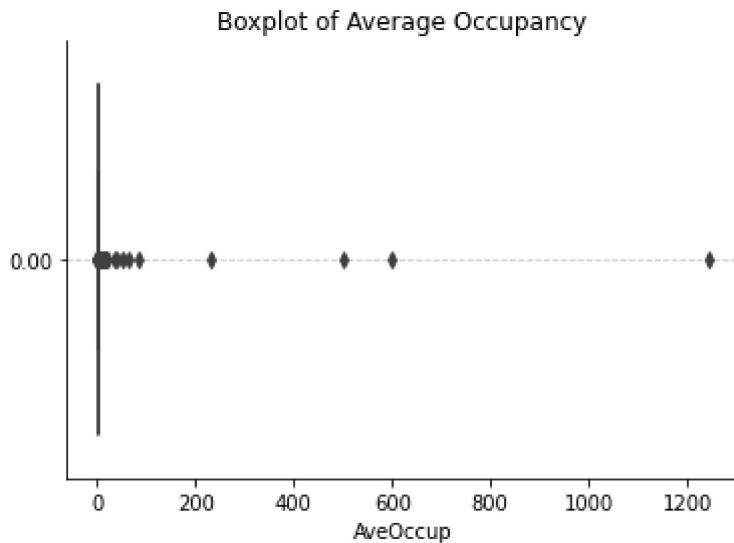
```

# Customize the y-axis tick labels to display values in millions
yticks = ax.get_yticks() / 1000000
ax.set_yticklabels(['{:.2f}'.format(ytick) for ytick in yticks])

# Add grid Lines and remove top and right spines
ax.grid(axis='y', linestyle='--', alpha=0.7)
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)

# Show the plot
plt.show()

```



```

In [13]: # Create a boxplot of the 'MedInc' column
ax = sns.boxplot(x=california_df['MedInc'])

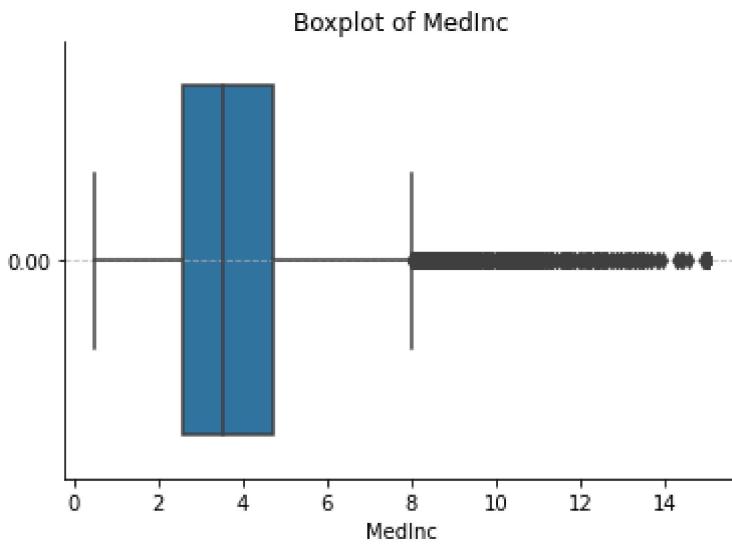
# Set the title and axes labels
ax.set_title('Boxplot of MedInc')
ax.set_xlabel('MedInc')
ax.set_ylabel('')

# Customize the y-axis tick labels to display values in millions
yticks = ax.get_yticks() / 1000000
ax.set_yticklabels(['{:.2f}'.format(ytick) for ytick in yticks])

# Add grid Lines and remove top and right spines
ax.grid(axis='y', linestyle='--', alpha=0.7)
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)

# Show the plot
plt.show()

```



Step 2(f)(ii): Deal with Outliers in the Dataframe using Winsorization:

This method involves replacing extreme values with the nearest values that are within a certain percentile range. For example, we replace values above the 95th percentile with the value at the 95th percentile and values below the 1st percentile with the value at the 1st percentile. From the visuals we can clearly see that the data is way more normally distributed now

```
In [14]: # Define the percentile limits for winsorization
pct_lower = 0.01
pct_upper = 0.95

# Apply winsorization to the five columns
california_df['AveRooms'] = np.clip(california_df['AveRooms'],
                                      california_df['AveRooms'].quantile(pct_lower),
                                      california_df['AveRooms'].quantile(pct_upper))
california_df['AveBedrms'] = np.clip(california_df['AveBedrms'],
                                      california_df['AveBedrms'].quantile(pct_lower),
                                      california_df['AveBedrms'].quantile(pct_upper))
california_df['Population'] = np.clip(california_df['Population'],
                                       california_df['Population'].quantile(pct_lower),
                                       california_df['Population'].quantile(pct_upper))
california_df['AveOccup'] = np.clip(california_df['AveOccup'],
                                     california_df['AveOccup'].quantile(pct_lower),
                                     california_df['AveOccup'].quantile(pct_upper))
california_df['MedInc'] = np.clip(california_df['MedInc'],
                                   california_df['MedInc'].quantile(pct_lower),
                                   california_df['MedInc'].quantile(pct_upper))
```

```
In [15]: # Create a boxplot of the 'AveRooms' column
ax = sns.boxplot(x=california_df['AveRooms'])

# Set the title and axes labels
ax.set_title('Boxplot of Average Number of Rooms')
ax.set_xlabel('AveRooms')
ax.set_ylabel('')

# Customize the y-axis tick labels to display values in millions
yticks = ax.get_yticks() / 1000000
ax.set_yticklabels(['{:,.2f}'.format(ytick) for ytick in yticks])

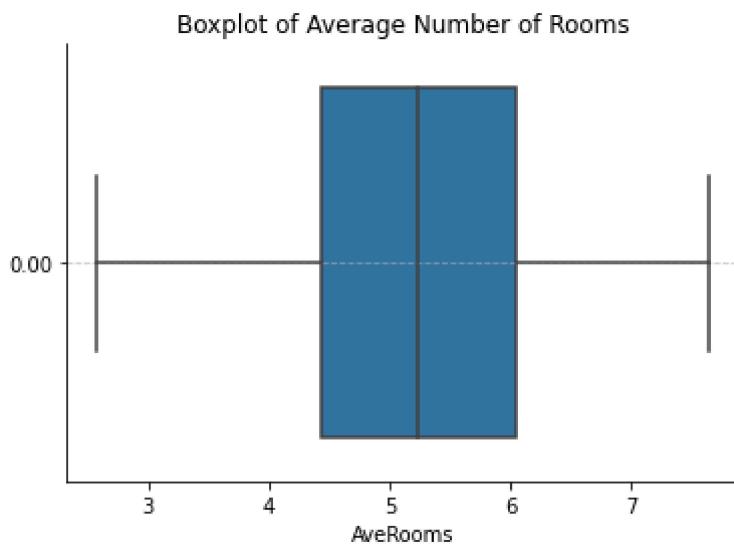
# Add grid lines and remove top and right spines
ax.grid(axis='y', linestyle='--', alpha=0.7)
```

```

ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)

# Show the plot
plt.show()

```



```

In [16]: # Create a boxplot of the 'AveBedrms' column
ax = sns.boxplot(x=california_df['AveBedrms'])

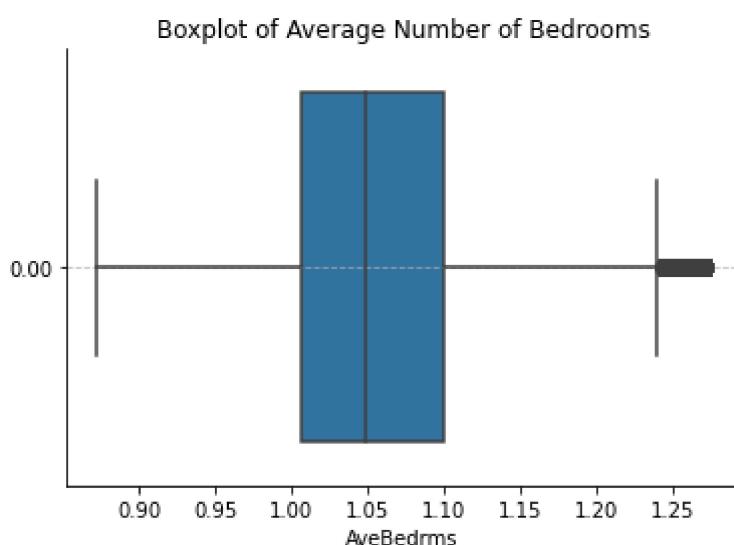
# Set the title and axes Labels
ax.set_title('Boxplot of Average Number of Bedrooms')
ax.set_xlabel('AveBedrms')
ax.set_ylabel('')

# Customize the y-axis tick Labels to display values in millions
yticks = ax.get_yticks() / 1000000
ax.set_yticklabels(['{:.2f}'.format(ytick) for ytick in yticks])

# Add grid Lines and remove top and right spines
ax.grid(axis='y', linestyle='--', alpha=0.7)
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)

# Show the plot
plt.show()

```



```

In [17]: # Create a boxplot of the 'Population' column
ax = sns.boxplot(x=california_df['Population'])

```

```

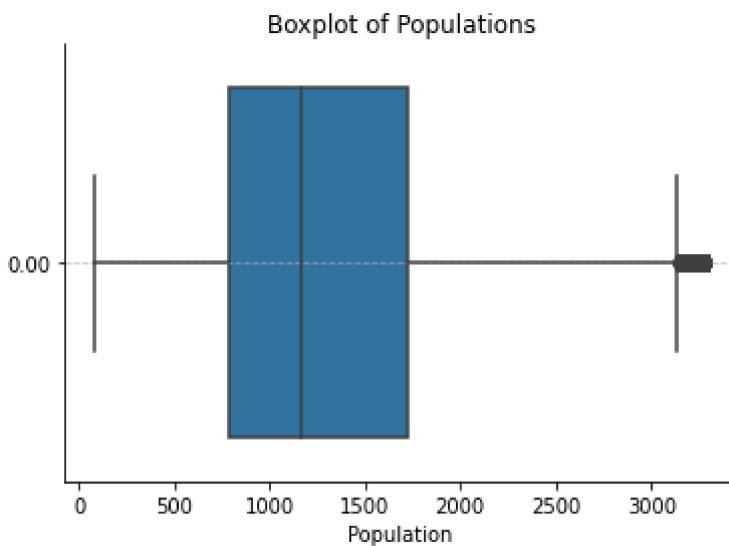
# Set the title and axes labels
ax.set_title('Boxplot of Populations')
ax.set_xlabel('Population')
ax.set_ylabel('')

# Customize the y-axis tick labels to display values in millions
yticks = ax.get_yticks() / 1000000
ax.set_yticklabels(['{:,.2f}'.format(ytick) for ytick in yticks])

# Add grid lines and remove top and right spines
ax.grid(axis='y', linestyle='--', alpha=0.7)
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)

# Show the plot
plt.show()

```



```

In [18]: # Create a boxplot of the 'MedInc' column
ax = sns.boxplot(x=california_df['MedInc'])

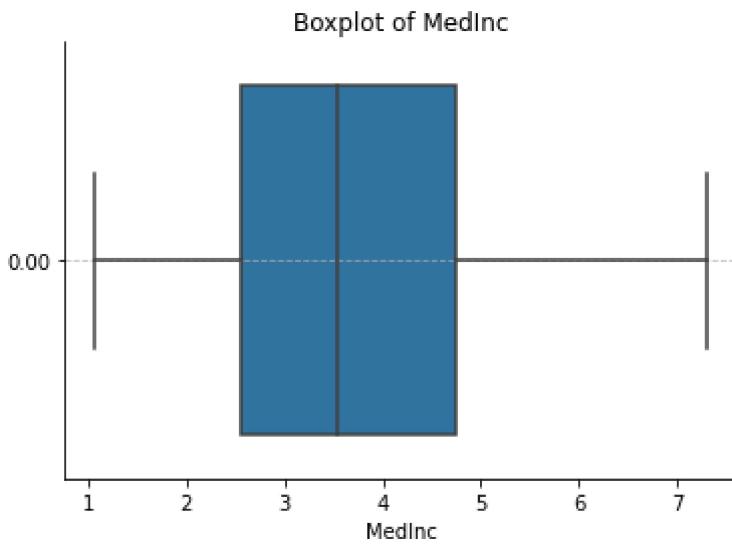
# Set the title and axes labels
ax.set_title('Boxplot of MedInc')
ax.set_xlabel('MedInc')
ax.set_ylabel('')

# Customize the y-axis tick labels to display values in millions
yticks = ax.get_yticks() / 1000000
ax.set_yticklabels(['{:,.2f}'.format(ytick) for ytick in yticks])

# Add grid lines and remove top and right spines
ax.grid(axis='y', linestyle='--', alpha=0.7)
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)

# Show the plot
plt.show()

```



Step 3(g): Check for Skewness using a Histogram

Skewed data can result in biased estimates of model parameters and reduce the accuracy of predictions. Therefore, it is important to assess the distribution of features and target variables to identify any potential issues and take appropriate measures to address them.

Here almost all the features and target look normally distributed. There is some Skewness In MedHouseVal but not enough to do Transformation on it.

Note: For learning purposes I have shown how to do MedHouseVal transformation for skewness in my previous tutorial of Simple Linear Regression. Feel free to check that out!

```
In [19]: # Set figure size and font scale
plt.figure(figsize=(8, 6))
sns.set(font_scale=1.5)

# Create histogram
sns.histplot(data=california_df, x='MedHouseVal', kde=True, bins=50, color="#7fc97f")

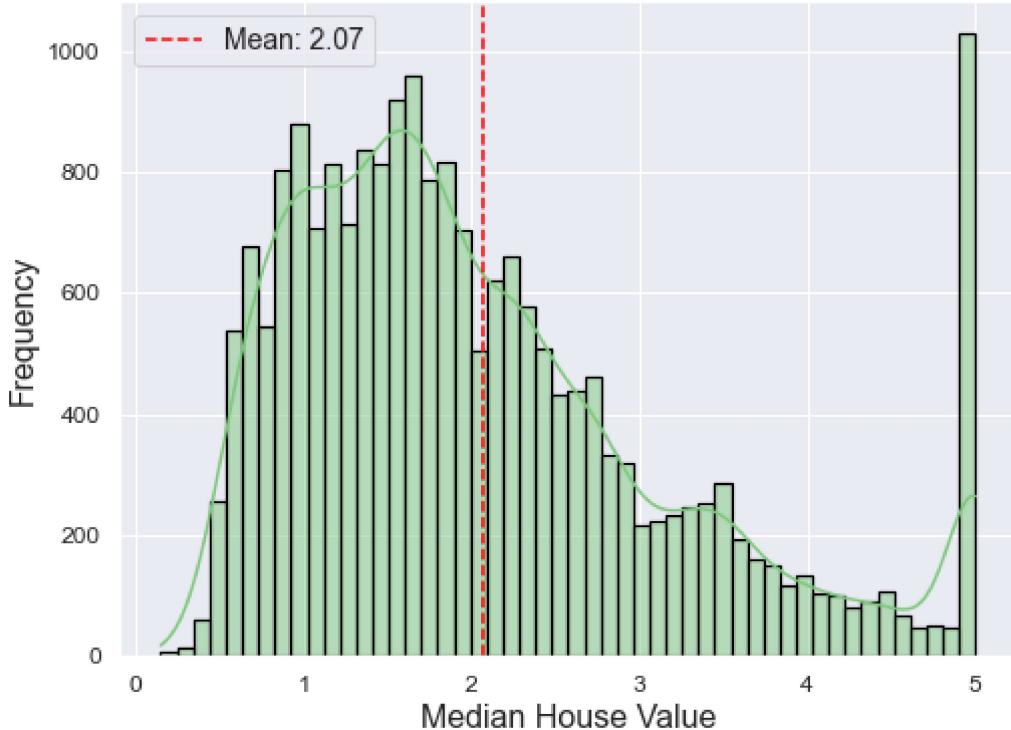
# Set x and y axis labels and title
plt.xlabel('Median House Value', fontsize=16)
plt.ylabel('Frequency', fontsize=16)
plt.title('Distribution of Median House Value in California', fontsize=20)

# Customize x and y axis tick marks and labels
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

# Add vertical Line for mean
mean = california_df['MedHouseVal'].mean()
plt.axvline(mean, color='red', linestyle='--', label=f'Mean: {mean:.2f}')
plt.legend(fontsize=14)

# Show the plot
plt.show()
```

Distribution of Median House Value in California



```
In [20]: # Set figure size and font scale
plt.figure(figsize=(8, 6))
sns.set(font_scale=1.5)

# Create histogram
sns.histplot(data=california_df, x='MedInc', kde=True, bins=50, color='#beaed4', edgecolor='black')

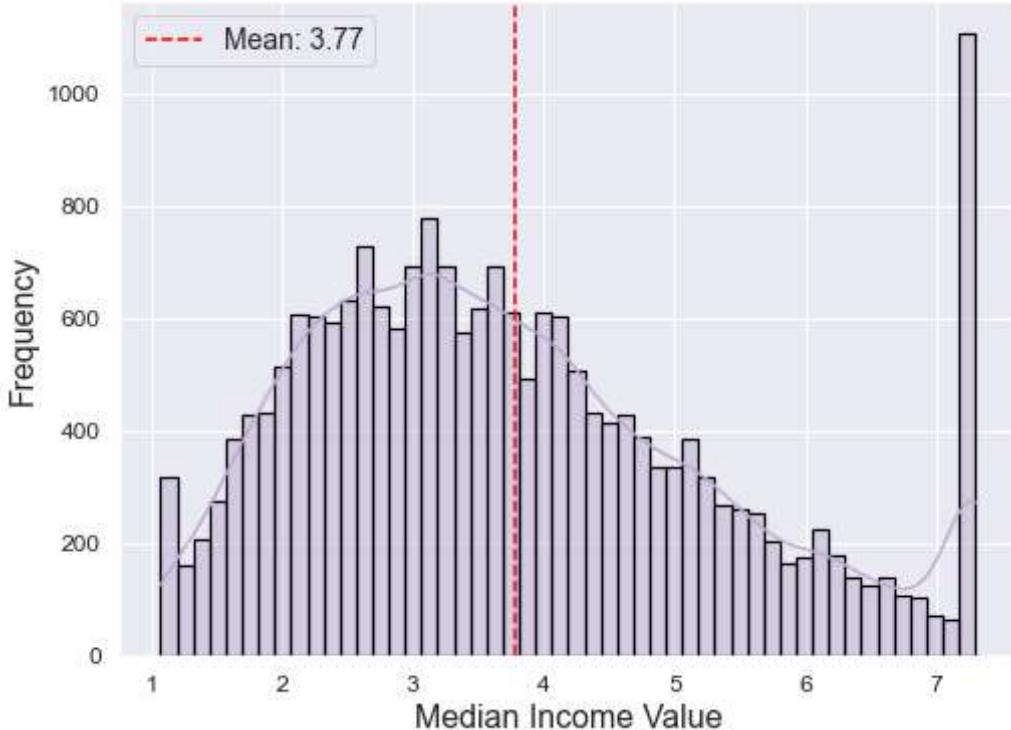
# Set x and y axis labels and title
plt.xlabel('Median Income Value', fontsize=16)
plt.ylabel('Frequency', fontsize=16)
plt.title('Distribution of Median MedInc Value in California', fontsize=20)

# Customize x and y axis tick marks and labels
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

# Add vertical line for MedInc
mean = california_df['MedInc'].mean()
plt.axvline(mean, color='red', linestyle='--', label=f'Mean: {mean:.2f}')
plt.legend(fontsize=14)

# Show the plot
plt.show()
```

Distribution of Median MedInc Value in California



```
In [21]: # Set figure size and font scale
plt.figure(figsize=(8, 6))
sns.set(font_scale=1.5)

# Create histogram
sns.histplot(data=california_df, x='HouseAge', kde=True, bins=50, color='#fdc086',

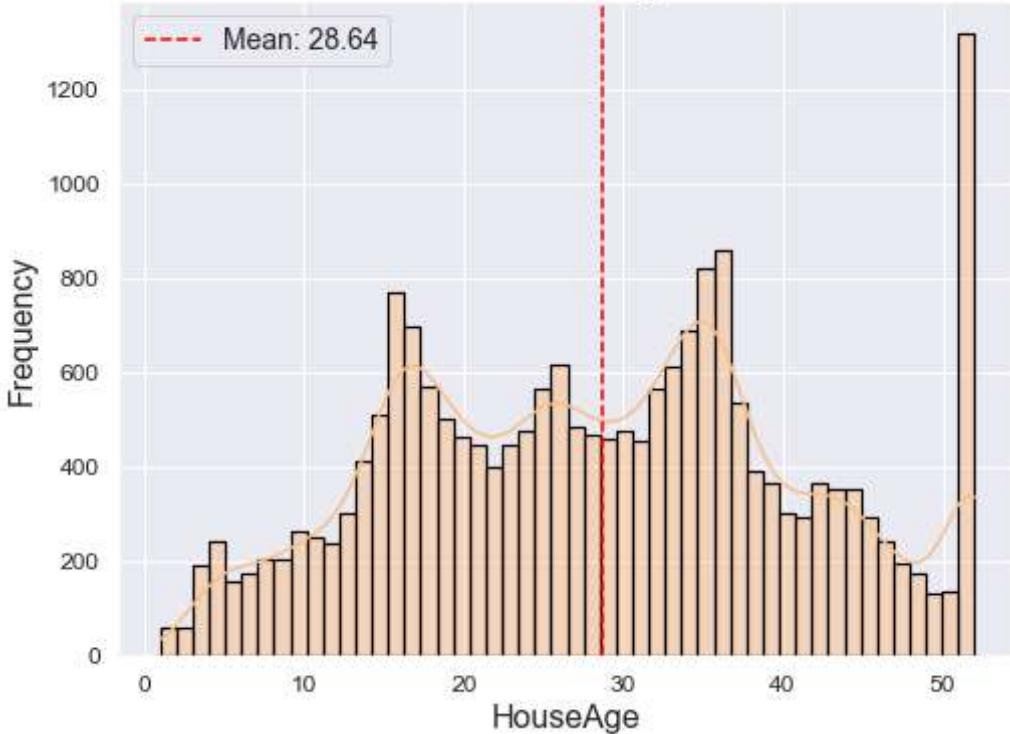
# Set x and y axis labels and title
plt.xlabel('HouseAge', fontsize=16)
plt.ylabel('Frequency', fontsize=16)
plt.title('Distribution of House Age in California', fontsize=20)

# Customize x and y axis tick marks and labels
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

# Add vertical line for mean
mean = california_df['HouseAge'].mean()
plt.axvline(mean, color='red', linestyle='--', label=f'Mean: {mean:.2f}')
plt.legend(fontsize=14)

# Show the plot
plt.show()
```

Distribution of House Age in California



```
In [22]: # Set figure size and font scale
plt.figure(figsize=(8, 6))
sns.set(font_scale=1.5)

# Create histogram
sns.histplot(data=california_df, x='AveRooms', kde=True, bins=50, color='#ffff99',

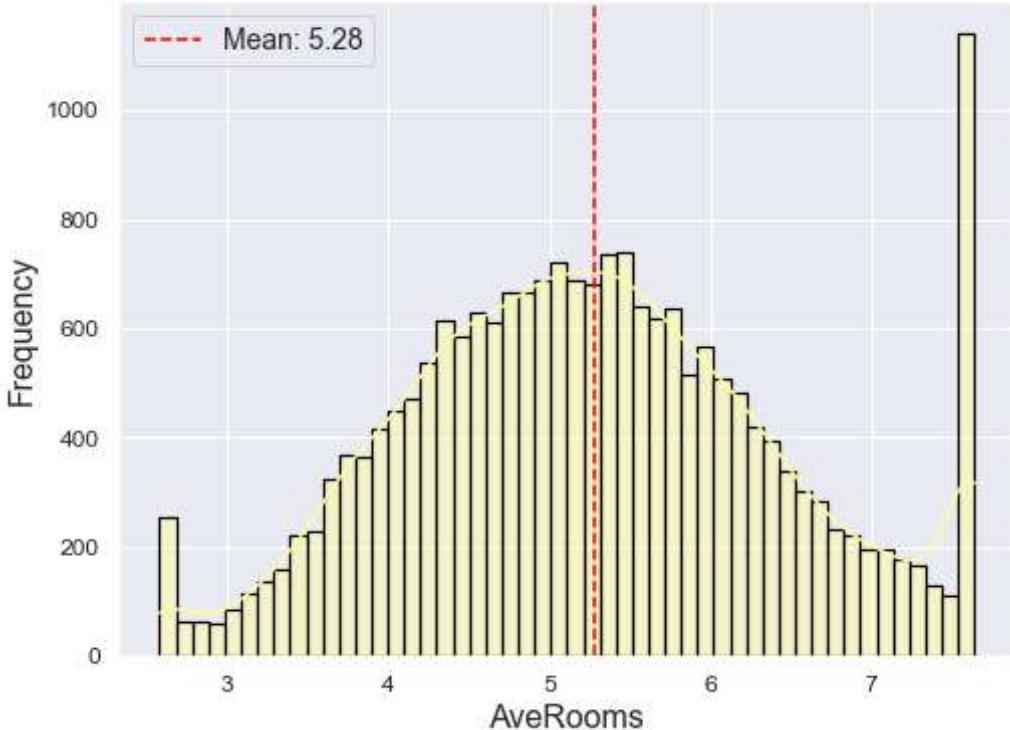
# Set x and y axis labels and title
plt.xlabel('AveRooms', fontsize=16)
plt.ylabel('Frequency', fontsize=16)
plt.title('Distribution of AveRooms in California', fontsize=20)

# Customize x and y axis tick marks and labels
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

# Add vertical line for mean
mean = california_df['AveRooms'].mean()
plt.axvline(mean, color='red', linestyle='--', label=f'Mean: {mean:.2f}')
plt.legend(fontsize=14)

# Show the plot
plt.show()
```

Distribution of AveRooms in California



```
In [23]: # Set figure size and font scale
plt.figure(figsize=(8, 6))
sns.set(font_scale=1.5)

# Create histogram
sns.histplot(data=california_df, x='AveBedrms', kde=True, bins=50, color="#386cb0")

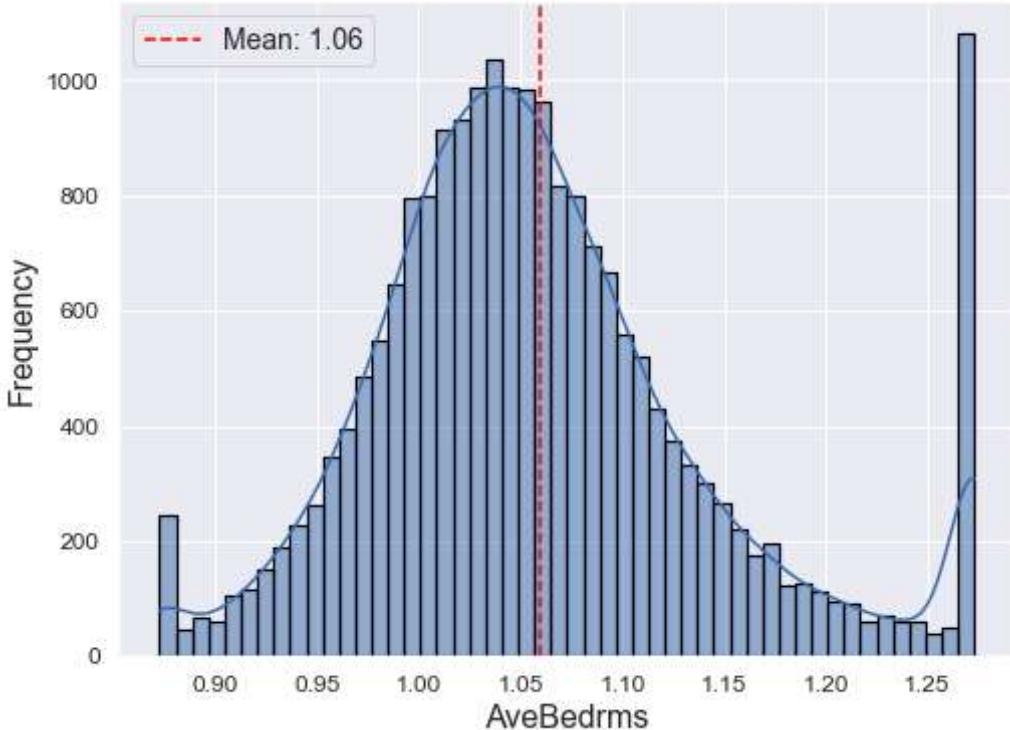
# Set x and y axis labels and title
plt.xlabel('AveBedrms', fontsize=16)
plt.ylabel('Frequency', fontsize=16)
plt.title('Distribution of AveBedrms in California', fontsize=20)

# Customize x and y axis tick marks and labels
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

# Add vertical line for mean
mean = california_df['AveBedrms'].mean()
plt.axvline(mean, color='red', linestyle='--', label=f'Mean: {mean:.2f}')
plt.legend(fontsize=14)

# Show the plot
plt.show()
```

Distribution of AveBedrms in California



```
In [24]: # Set figure size and font scale
plt.figure(figsize=(8, 6))
sns.set(font_scale=1.5)

# Create histogram
sns.histplot(data=california_df, x='Population', kde=True, bins=50, color='#f0027f'

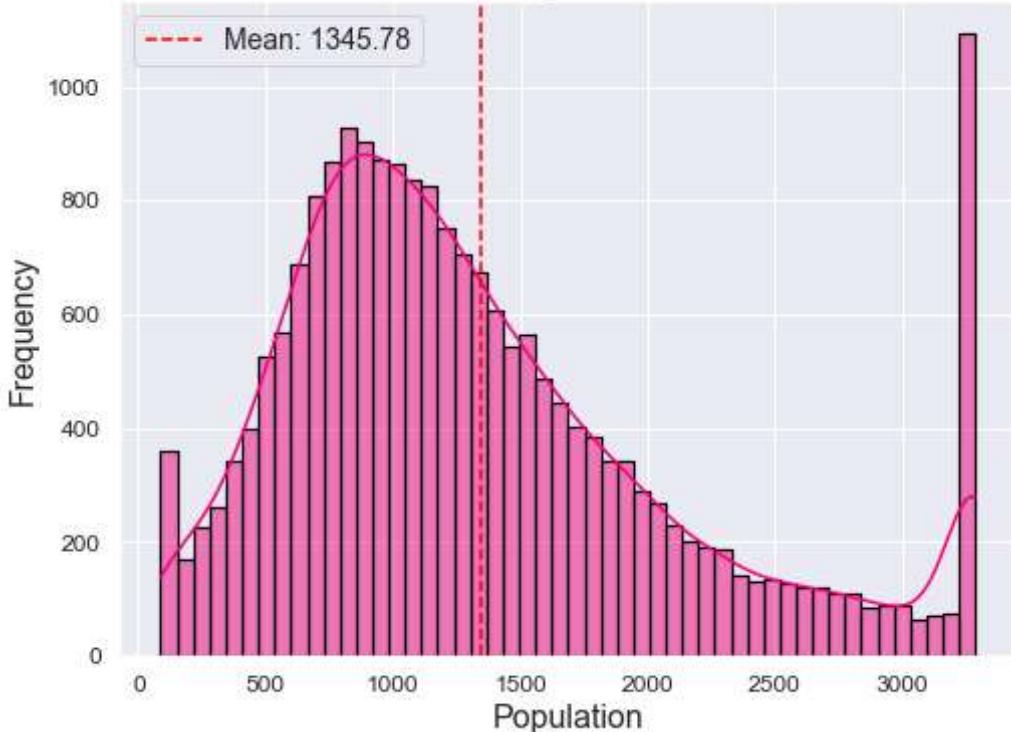
# Set x and y axis labels and title
plt.xlabel('Population', fontsize=16)
plt.ylabel('Frequency', fontsize=16)
plt.title('Distribution of Population in California', fontsize=20)

# Customize x and y axis tick marks and labels
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

# Add vertical line for mean
mean = california_df['Population'].mean()
plt.axvline(mean, color='red', linestyle='--', label=f'Mean: {mean:.2f}')
plt.legend(fontsize=14)

# Show the plot
plt.show()
```

Distribution of Population in California



```
In [25]: # Set figure size and font scale
plt.figure(figsize=(8, 6))
sns.set(font_scale=1.5)

# Create histogram
sns.histplot(data=california_df, x='AveOccup', kde=True, bins=50, color='#bf5b17',

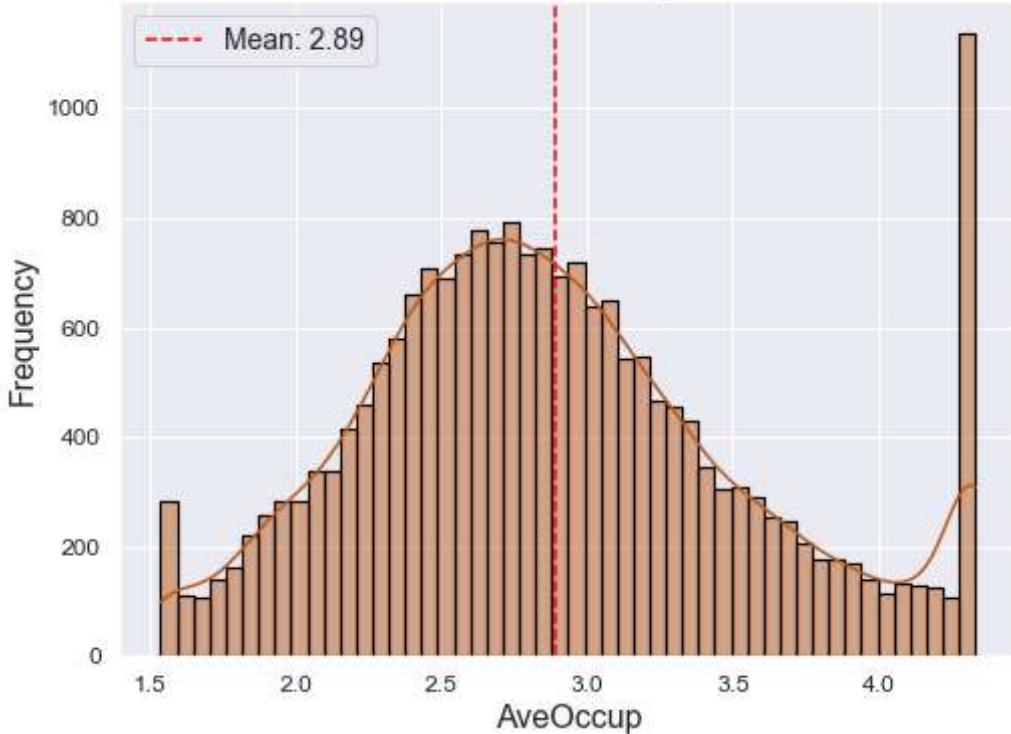
# Set x and y axis labels and title
plt.xlabel('AveOccup', fontsize=16)
plt.ylabel('Frequency', fontsize=16)
plt.title('Distribution of AveOccup in California', fontsize=20)

# Customize x and y axis tick marks and labels
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

# Add vertical line for mean
mean = california_df['AveOccup'].mean()
plt.axvline(mean, color='red', linestyle='--', label=f'Mean: {mean:.2f}')
plt.legend(fontsize=14)

# Show the plot
plt.show()
```

Distribution of AveOccup in California



```
In [26]: # Set figure size and font scale
plt.figure(figsize=(8, 6))
sns.set(font_scale=1.5)

# Create histogram
sns.histplot(data=california_df, x='Latitude', kde=True, bins=50, color='#F4CCCC',

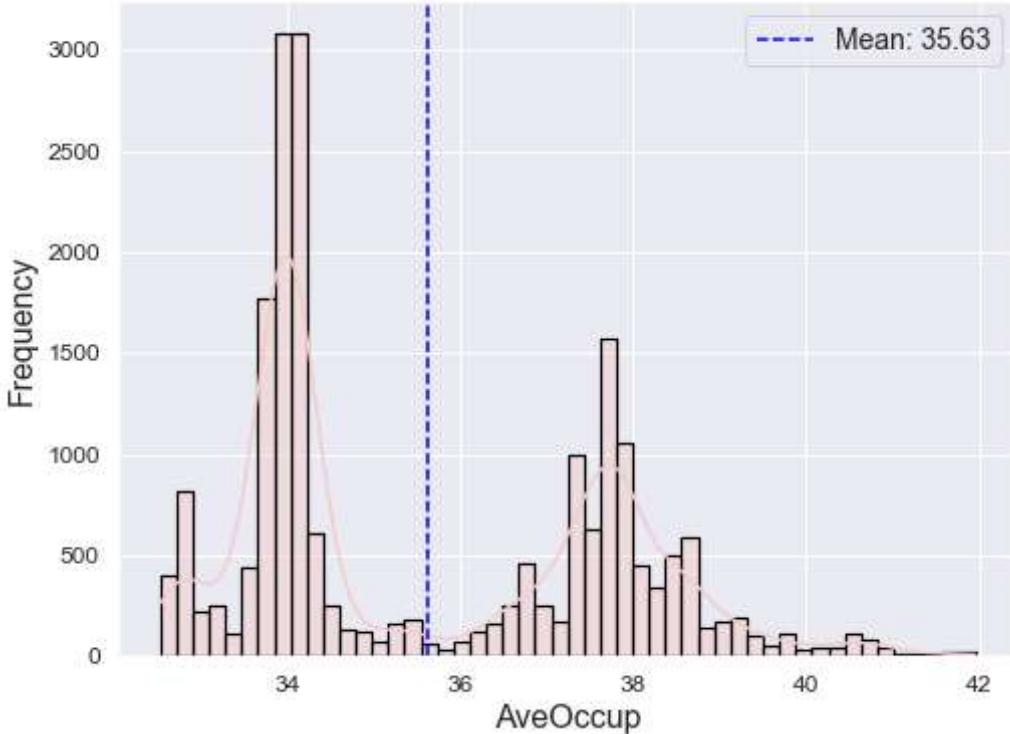
# Set x and y axis labels and title
plt.xlabel('AveOccup', fontsize=16)
plt.ylabel('Frequency', fontsize=16)
plt.title('Distribution of Latitude', fontsize=20)

# Customize x and y axis tick marks and labels
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

# Add vertical line for mean
mean = california_df['Latitude'].mean()
plt.axvline(mean, color='blue', linestyle='--', label=f'Mean: {mean:.2f}')
plt.legend(fontsize=14)

# Show the plot
plt.show()
```

Distribution of Latitude



```
In [27]: # Set figure size and font scale
plt.figure(figsize=(8, 6))
sns.set(font_scale=1.5)

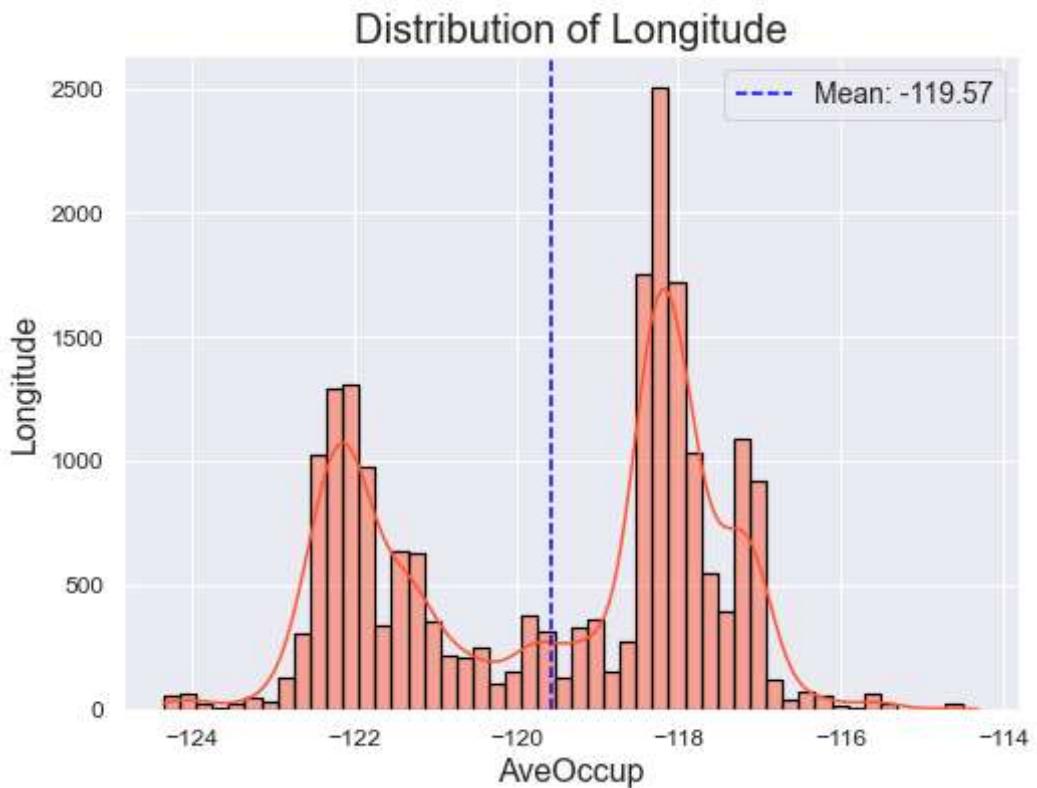
# Create histogram
sns.histplot(data=california_df, x='Longitude', kde=True, bins=50, color="#FF5733")

# Set x and y axis labels and title
plt.xlabel('AveOccup', fontsize=16)
plt.ylabel('Longitude', fontsize=16)
plt.title('Distribution of Longitude', fontsize=20)

# Customize x and y axis tick marks and labels
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

# Add vertical line for mean
mean = california_df['Longitude'].mean()
plt.axvline(mean, color='blue', linestyle='--', label=f'Mean: {mean:.2f}')
plt.legend(fontsize=14)

# Show the plot
plt.show()
```



Step 3(h): Create a Vertical Correlation Heatmap

The correlation matrix shows the correlation coefficients between every pair of variables in the dataset. A correlation coefficient ranges from -1 to 1 and measures the strength and direction of the linear relationship between two variables. A coefficient of -1 indicates a perfect negative correlation, a coefficient of 0 indicates no correlation, and a coefficient of 1 indicates a perfect positive correlation.

```
In [28]: # Calculate the correlation matrix
corr_matrix = california_df.corr()

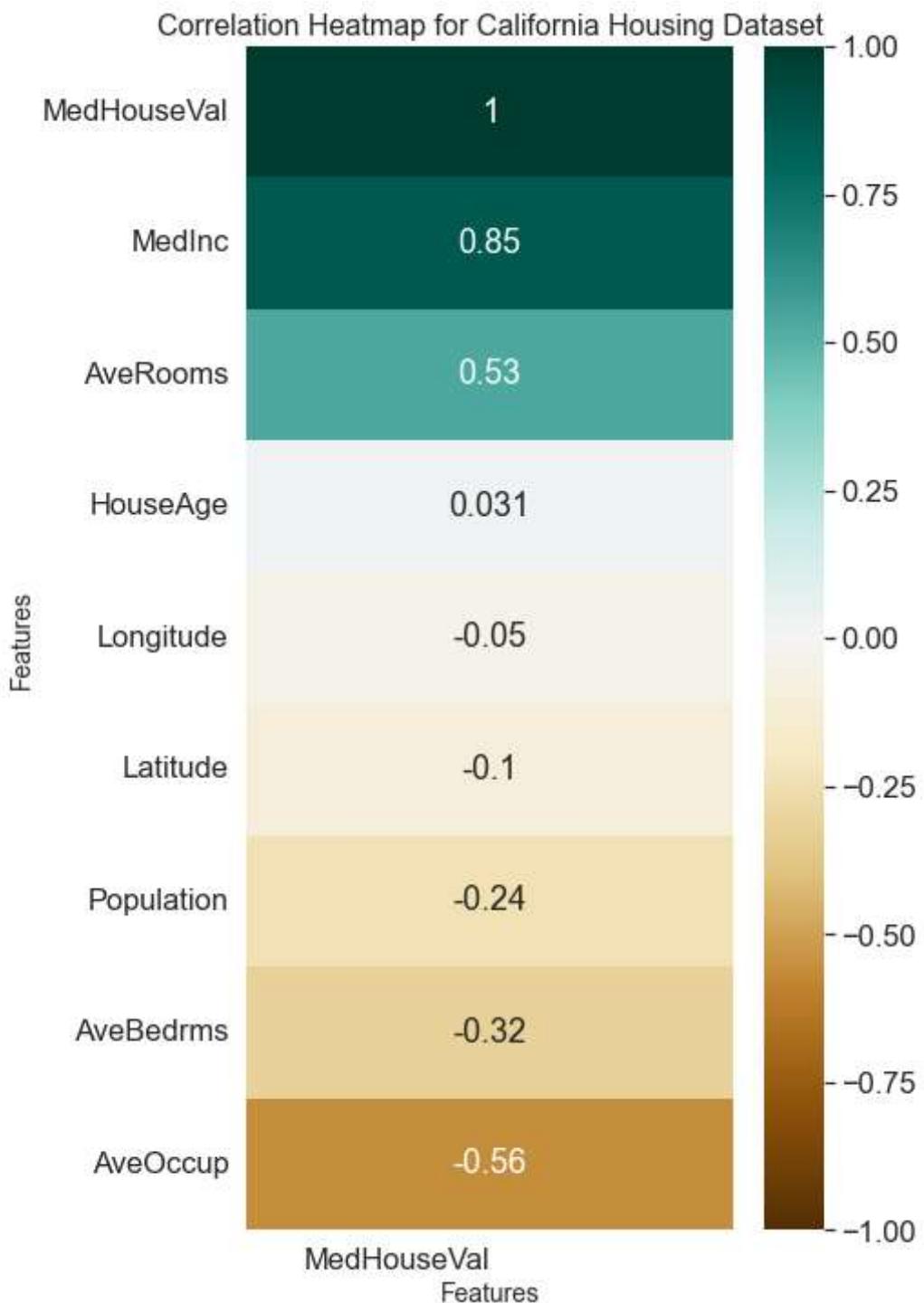
# Set up the matplotlib figure
fig, ax = plt.subplots(figsize=(6, 12))

# Create the heatmap
sns.heatmap(corr_matrix, cmap='BrBG', annot=True, fmt='.2f', linewidths=.5, ax=ax,
sns.heatmap(corr_matrix.corr()[[ 'MedHouseVal']].sort_values(by='MedHouseVal', ascending=False), ax=ax)

# Set the title and axis labels
ax.set_title('Correlation Heatmap for California Housing Dataset', fontsize=16)
ax.set_xlabel('Features', fontsize=14)
ax.set_ylabel('Features', fontsize=14)

# Rotate the x-axis labels for readability
plt.xticks(rotation=0, ha='right')

# Show the plot
plt.show()
```



Step 3(i): Perform Feature Scaling

Feature scaling is the process of transforming numerical features in a dataset to have similar scales or ranges of values. The purpose of feature scaling is to ensure that all features have the same level of impact on the model and to prevent certain features from dominating the model simply because they have larger values. In linear regression, feature scaling is particularly important because the coefficients of the model represent the change in the dependent variable associated with a one-unit change in the independent variable. Scaling the features to have similar ranges can result in a more accurate and reliable model with more accurate representations of the relationships between the independent variables and the dependent variable.

```
In [47]: scaler = StandardScaler()
california_df_scaled = scaler.fit_transform(california_df)
```

```
california_df_scaled = pd.DataFrame(california_df_scaled, columns=california_df.co
```

Step 3(j) Check for Assumptions using Scatter Plots

From the scatter plots, we can observe that there is a linear relationship between the dependent variable (Median House Value) and some of the independent variables like Median Income and Total Rooms. However, we can also see that some of the independent variables like Longitude, Latitude, and Housing Median Age do not have a clear linear relationship with the dependent variable. This suggests that a linear regression model might not be the best fit for predicting the Median House Value based on these variables.

In [44]:

```
# Create scatter plots
fig, axs = plt.subplots(nrows=2, ncols=4, figsize=(30,15))

axs[0,0].scatter(california_df_scaled['Latitude'], california_df_scaled['MedHouseValue'])
axs[0,0].set_xlabel('Latitude')
axs[0,0].set_ylabel('Median House Value')
axs[0,0].set_title('Latitude vs Median House Value')

axs[0,1].scatter(california_df_scaled['Longitude'], california_df_scaled['MedHouseValue'])
axs[0,1].set_xlabel('Longitude')
axs[0,1].set_ylabel('Median House Value')
axs[0,1].set_title('Longitude vs Median House Value')

axs[0,2].scatter(california_df_scaled['HouseAge'], california_df_scaled['MedHouseValue'])
axs[0,2].set_xlabel('Housing Median Age')
axs[0,2].set_ylabel('Median House Value')
axs[0,2].set_title('Housing Median Age vs Median House Value')

axs[0,3].scatter(california_df_scaled['AveRooms'], california_df_scaled['MedHouseValue'])
axs[0,3].set_xlabel('Total Rooms')
axs[0,3].set_ylabel('Median House Value')
axs[0,3].set_title('Total Rooms vs Median House Value')

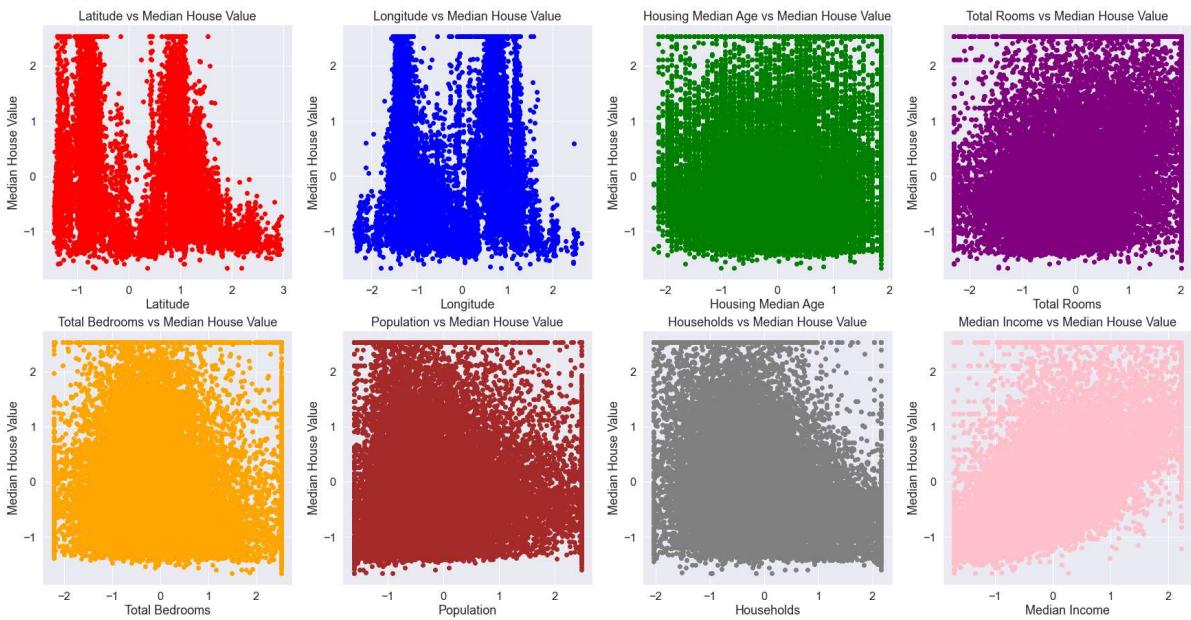
axs[1,0].scatter(california_df_scaled['AveBedrms'], california_df_scaled['MedHouseValue'])
axs[1,0].set_xlabel('Total Bedrooms')
axs[1,0].set_ylabel('Median House Value')
axs[1,0].set_title('Total Bedrooms vs Median House Value')

axs[1,1].scatter(california_df_scaled['Population'], california_df_scaled['MedHouseValue'])
axs[1,1].set_xlabel('Population')
axs[1,1].set_ylabel('Median House Value')
axs[1,1].set_title('Population vs Median House Value')

axs[1,2].scatter(california_df_scaled['AveOccup'], california_df_scaled['MedHouseValue'])
axs[1,2].set_xlabel('Households')
axs[1,2].set_ylabel('Median House Value')
axs[1,2].set_title('Households vs Median House Value')

axs[1,3].scatter(california_df_scaled['MedInc'], california_df_scaled['MedHouseValue'])
axs[1,3].set_xlabel('Median Income')
axs[1,3].set_ylabel('Median House Value')
axs[1,3].set_title('Median Income vs Median House Value')

plt.show()
```



Step 4: Define Dependant and Independant Variable

```
In [50]: X = california_df_scaled.drop(['MedHouseVal'], axis=1)
y = california_df_scaled['MedHouseVal']
```

Step 5: Do Train-Test Split in the ratio 70-30

```
In [51]: # Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Step 6: Searching for Best Polynomial Degeree Based on R2

```
In [63]: # initialize variables for storing best model and score
best_score = 0
best_degree = 1

# Loop through degrees of polynomial from 1 to 10
for degree in range(1, 11):
    # create polynomial features
    poly_features = PolynomialFeatures(degree=degree)
    X_poly_train = poly_features.fit_transform(X_train)
    X_poly_test = poly_features.transform(X_test)

    # fit linear regression model to polynomial features
    poly_reg = LinearRegression()
    poly_reg.fit(X_poly_train, y_train)

    # evaluate model on test set
    y_pred = poly_reg.predict(X_poly_test)
    score = r2_score(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)

    # check if this is the best model so far
    if score > best_score:
        best_score = score
        best_mse = mse
        best_degree = degree
        best_model = poly_reg
```

```
# print best degree and score
print("Best degree:", best_degree)
print("R^2 score:", best_score)
print("Mean Squared Error:", best_mse)
```

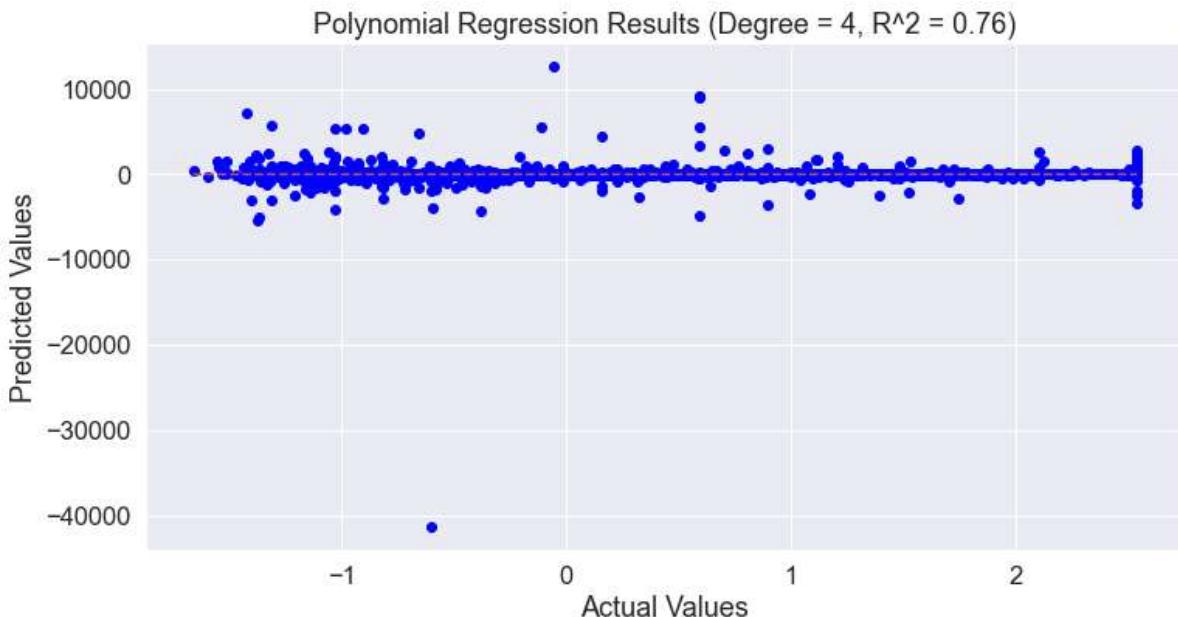
```
Best degree: 4
R^2 score: 0.7564759864738175
Mean Squared Error: 0.2438324527757844
```

```
In [61]: # plot actual vs predicted values
plt.figure(figsize=(12, 6))
plt.scatter(y_test, y_pred, color='blue')

# add labels and title
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Polynomial Regression Results (Degree = ' + str(best_degree) + ', R^2 = ' + str(best_score) + ')')

# add diagonal line for reference
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')

# display the plot
plt.show()
```



Conclusion:

In this polynomial regression tutorial, we explored how to use polynomial regression to model the relationship between variables with a non-linear pattern with the target feature, specifically the median house value in California. We started by loading and cleaning the dataset, and then visualizing the relationship between the variables using scatter plots.

Next, we split the data into training and testing sets and used polynomial regression to fit a model to the training data. We gradually increased the degree of the polynomial until we achieved a good balance between bias and variance. We then used the model to make predictions on the test data and evaluated the model's performance using mean squared error and R-squared score.