# Assignment (1)

## Part (2): Linear regression

## Import all required libraries

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from scipy import stats
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Ridge
```

## 1.Load Data

In [2]:
```
# Load the dataset (in Pandas DataFrame)
data = pd.read_csv('California_Houses.csv')
# Show dimensions of loaded data to make sure that data are fully loaded
print("data shape:", data.shape)
# Show First 5 Examples
data.head(5)
```

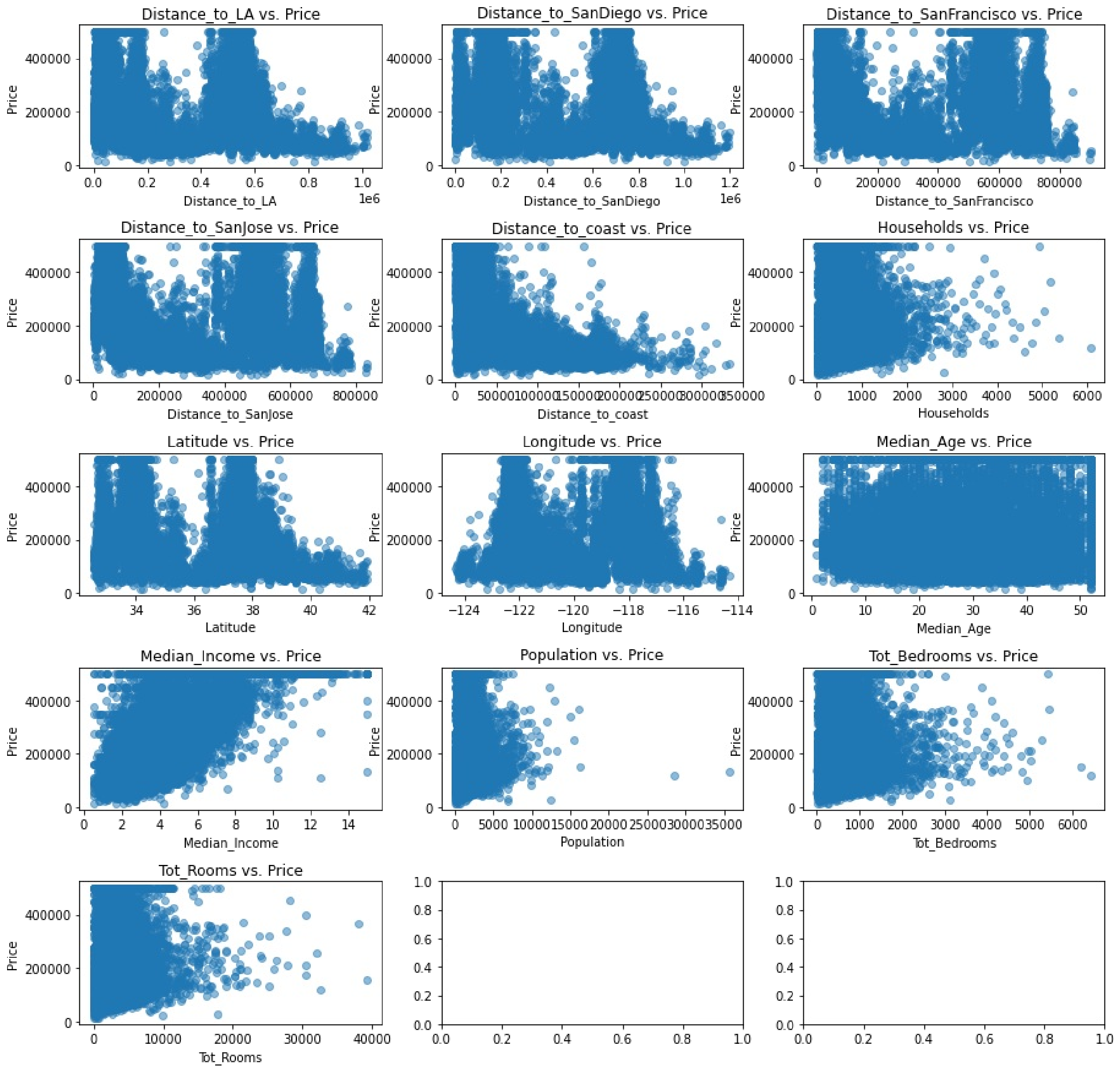
```
data shape: (20640, 14)
```

Out[2]:

| | Median_House_Value | Median_Income | Median_Age | Tot_Rooms | Tot_Bedrooms | Population | Households | Latitude | Longitude | Distance_to_ |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 452600.0 | 8.3252 | 41 | 880 | 129 | 322 | 126 | 37.88 | -122.23 | 9263.0 |
| **1** | 358500.0 | 8.3014 | 21 | 7099 | 1106 | 2401 | 1138 | 37.86 | -122.22 | 10225.7 |
| **2** | 352100.0 | 7.2574 | 52 | 1467 | 190 | 496 | 177 | 37.85 | -122.24 | 8259.0 |
| **3** | 341300.0 | 5.6431 | 52 | 1274 | 235 | 558 | 219 | 37.85 | -122.25 | 7768.0 |
| **4** | 342200.0 | 3.8462 | 52 | 1627 | 280 | 565 | 259 | 37.85 | -122.25 | 7768.0 |

## 2.Visualize Realtionship between each feature and target (price)

In [3]:
```
# Extract feature names from the DataFrame (excluding 'Median_House_Value')
features = data.columns.difference(['Median_House_Value'])

# Create basic scatter plots to visualize relationships
fig, axes = plt.subplots(nrows=5, ncols=3, figsize=(15, 15))
fig.subplots_adjust(hspace=0.5)

for i, feature in enumerate(features):
    row, col = i // 3, i % 3
    axes[row, col].scatter(data[feature], data['Median_House_Value'], alpha=0.5)
    axes[row, col].set_title(f'{feature} vs. Price')
    axes[row, col].set_xlabel(feature)
    axes[row, col].set_ylabel('Price')

plt.show()
```

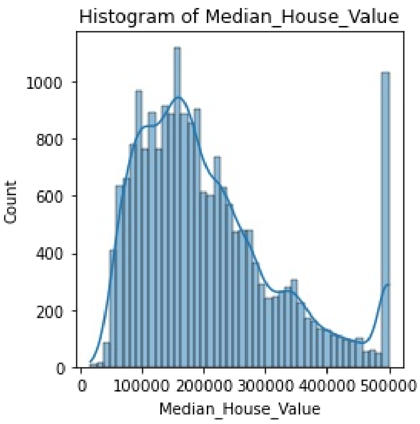


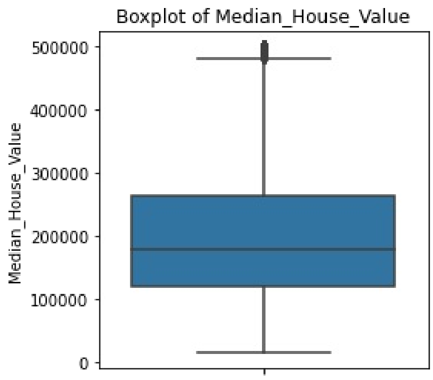

# 3.Data Preprocessing

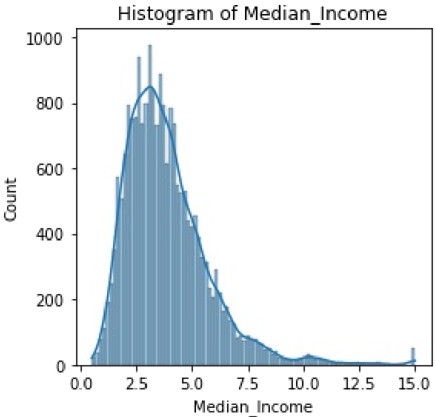
## Identify The Skewed Features

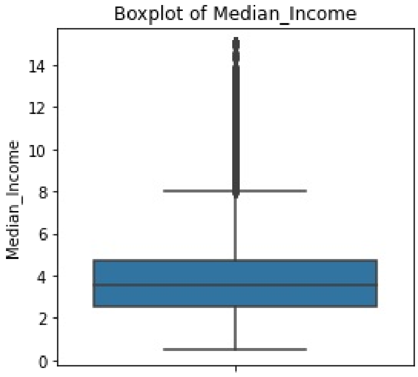
In [4]:

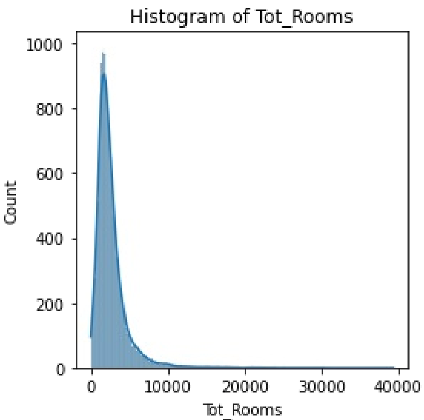
```
# Calculate skewness for all numeric features
numeric_features = data.select_dtypes(include=[np.number])
skewness = numeric_features.apply(lambda x: stats.skew(x))

# Assume threshold = 1/2
skew_threshold = 0.5

# Filter features with skewness above the threshold
skewed_features = skewness[abs(skewness) > skew_threshold]

# Create a histogram and boxplot for each skewed feature
for feature in skewed_features.index:
    plt.figure(figsize=(8, 4))

    plt.subplot(1, 2, 1)
    sns.histplot(data[feature], kde=True)
    plt.title(f'Histogram of {feature}')

    plt.subplot(1, 2, 2)
    sns.boxplot(y=data[feature])
    plt.title(f'Boxplot of {feature}')

    plt.tight_layout()
    plt.show()
```

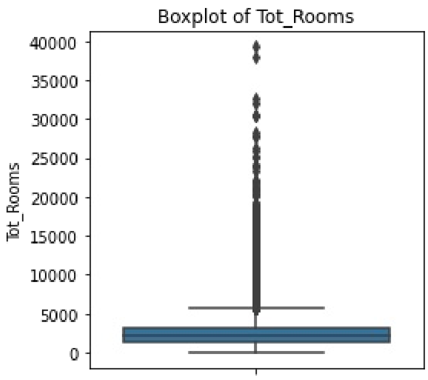

Histogram of Median_House_Value
Count
Median_House_Value
0
200
400
600
800
1000
100000 200000 300000 400000 500000

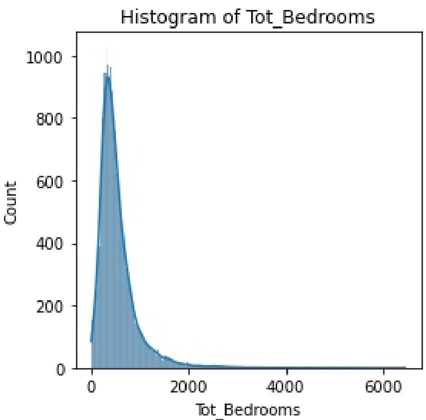

Boxplot of Median_House_Value
Median_House_Value
0
100000
200000
300000
400000
500000

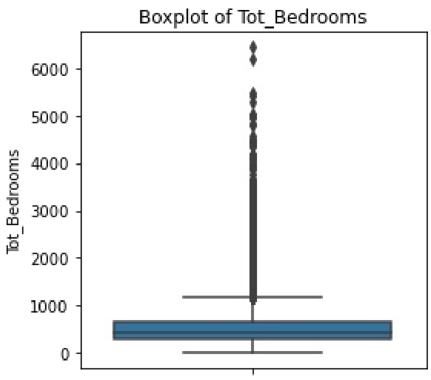

Histogram of Median_Income
Count
Median_Income
0
200
400
600
800
1000
0.0 2.5 5.0 7.5 10.0 12.5 15.0

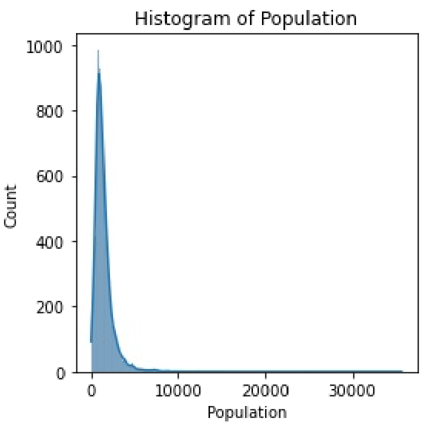

Boxplot of Median_Income
Median_Income
0
2
4
6
8
10
12
14

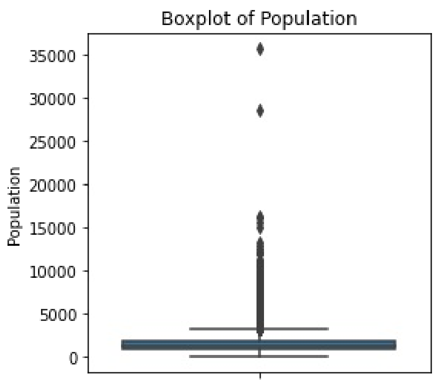

Histogram of Tot_Rooms
Count
Tot_Rooms
0
200
400
600
800
1000
0 10000 20000 30000 40000

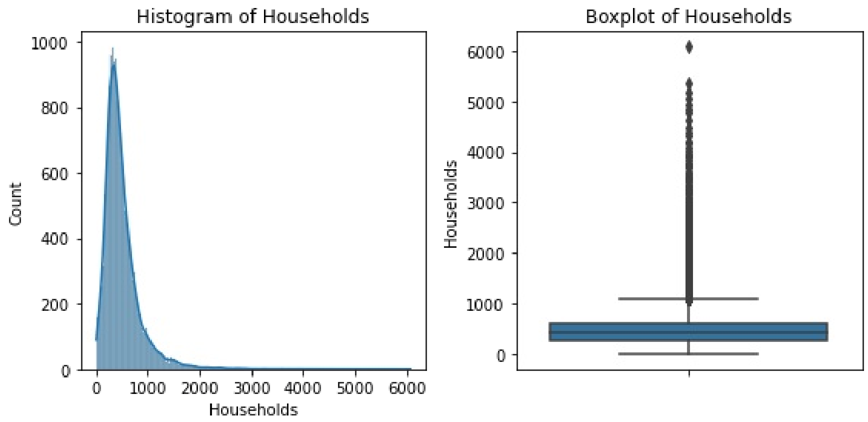

Boxplot of Tot_Rooms
Tot_Rooms
0
5000
10000
15000
20000
25000
30000
35000
40000

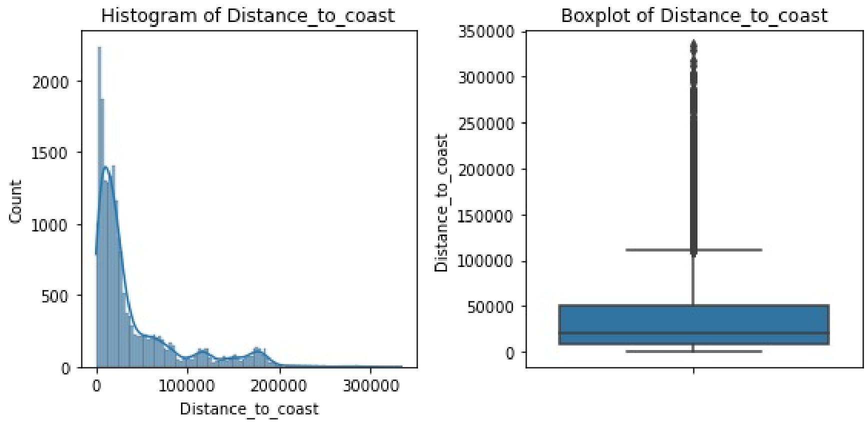

Histogram of Tot_Bedrooms
Count
Tot_Bedrooms
0
200
400
600
800
1000
0 2000 4000 6000


Boxplot of Tot_Bedrooms
Tot_Bedrooms
0
1000
2000
3000
4000
5000
6000


Histogram of Population
Count
Population
0
200
400
600
800
1000
0 10000 20000 30000

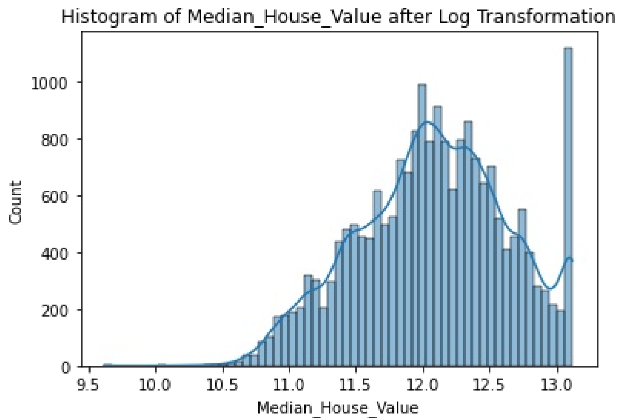

Boxplot of Population
Population
0
5000
10000
15000
20000
25000
30000
35000

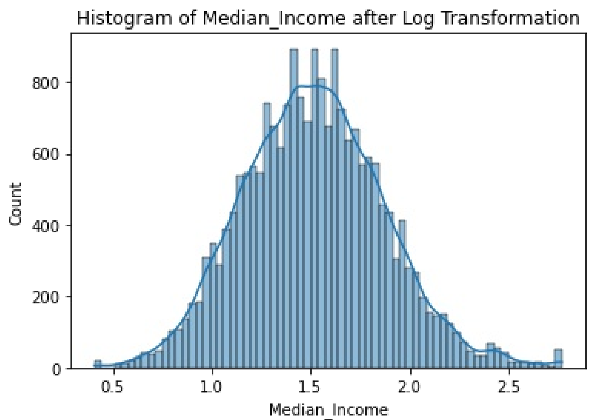


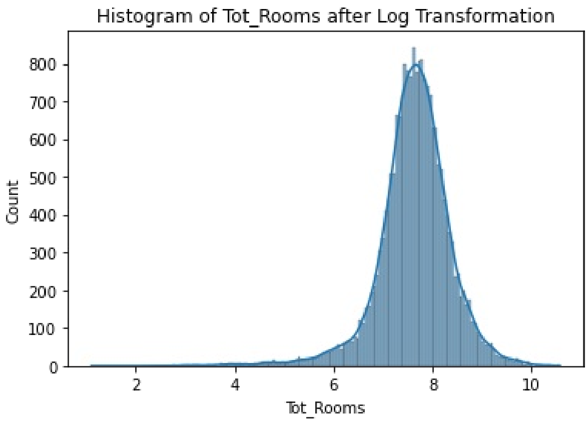



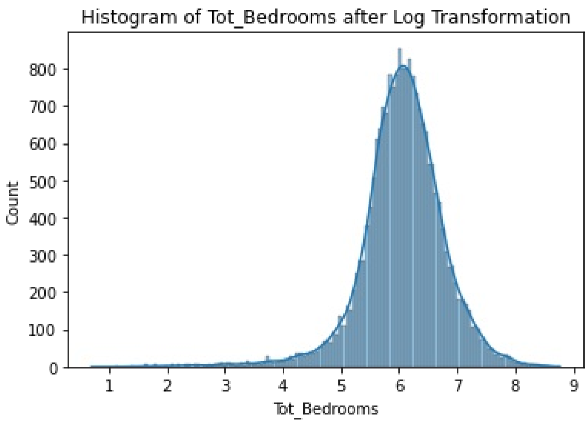

## Apply Log to this skewed data (to enhance it)

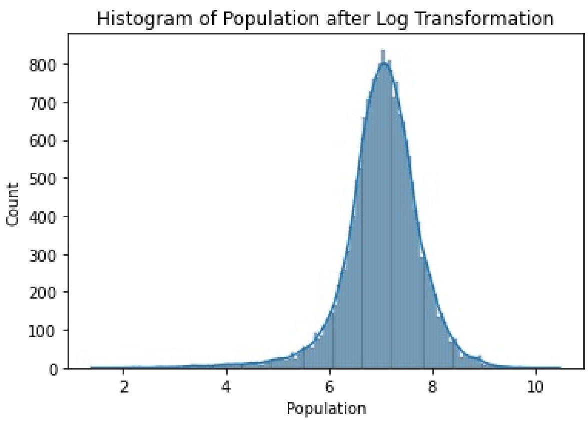
In [5]:
```python
# Apply log transformation to skewed features
for feature in skewed_features.index:
    data[feature] = np.log1p(data[feature])

# Create histograms for each transformed feature
for feature in skewed_features.index:
    plt.figure(figsize=(6, 4))
    sns.histplot(data[feature], kde=True)
    plt.title(f'Histogram of {feature} after Log Transformation')
    plt.show()
```

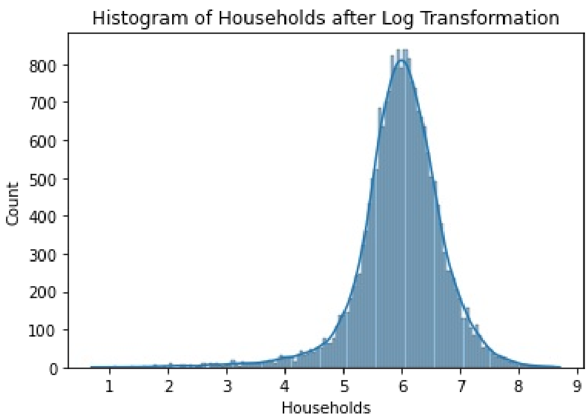


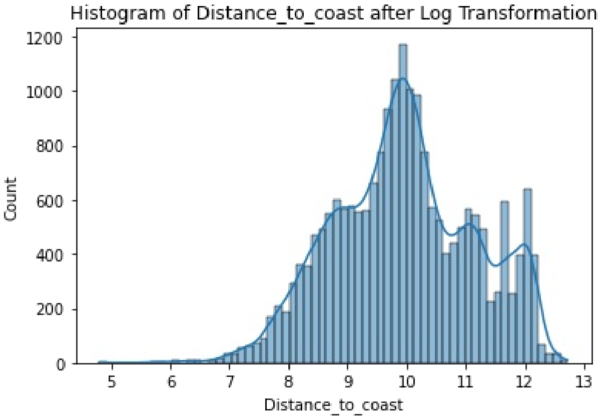




Histogram of Tot_Rooms after Log Transformation
Count
Tot_Rooms
0
100
200
300
400
500
600
700
800
2
4
6
8
10


Histogram of Tot_Bedrooms after Log Transformation
Count
Tot_Bedrooms
0
100
200
300
400
500
600
700
800
1
2
3
4
5
6
7
8
9


Histogram of Population after Log Transformation
Count
Population
0
100
200
300
400
500
600
700
800
2
4
6
8
10


Histogram of Households after Log Transformation
Count
Households
0
100
200
300
400
500
600
700
800
1
2
3
4
5
6
7
8
9




## Z-Score Normalization

In [6]:
```
# Initialize the StandardScaler
scaler = StandardScaler()

# Apply Z-score normalization to the entire dataset, including features and target
X = data.drop('Median_House_Value', axis=1)  # Extract features
y = data['Median_House_Value']               # Extract target variable

X = scaler.fit_transform(X)
# Convert X back to a Pandas DataFrame
X = pd.DataFrame(X, columns=data.columns.difference(['Median_House_Value']))

# Convert y to a Pandas DataFrame
y = pd.DataFrame(scaler.fit_transform(y.values.reshape(-1, 1)), columns=['Median_House_Value'])
```

## Handling Missing Values

In [7]:
```
# Check for missing values in the features
missing_values = X.isnull().sum()
print("Missing Values in Features:")
print(missing_values)

# Fill missing values with the mean value
X = X.fillna(X.mean())

#NO MISSING Values!
```

```
Missing Values in Features:
Distance_to_LA              0
Distance_to_SanDiego        0
Distance_to_SanFrancisco    0
Distance_to_SanJose         0
Distance_to_coast           0
Households                  0
Latitude                    0
Longitude                   0
Median_Age                  0
Median_Income               0
Population                  0
Tot_Bedrooms                0
Tot_Rooms                   0
dtype: int64
```

In [8]:
```
# Calculate the correlation matrix between X (features) and y (target)
correlation_matrix = np.corrcoef(X, y, rowvar=False)

# Sort the correlations with the target variable
correlations_with_target = correlation_matrix[:-1, -1]

# Create a Pandas Series to associate feature names with their correlations
correlations_series = pd.Series(correlations_with_target, index=X.columns)
```

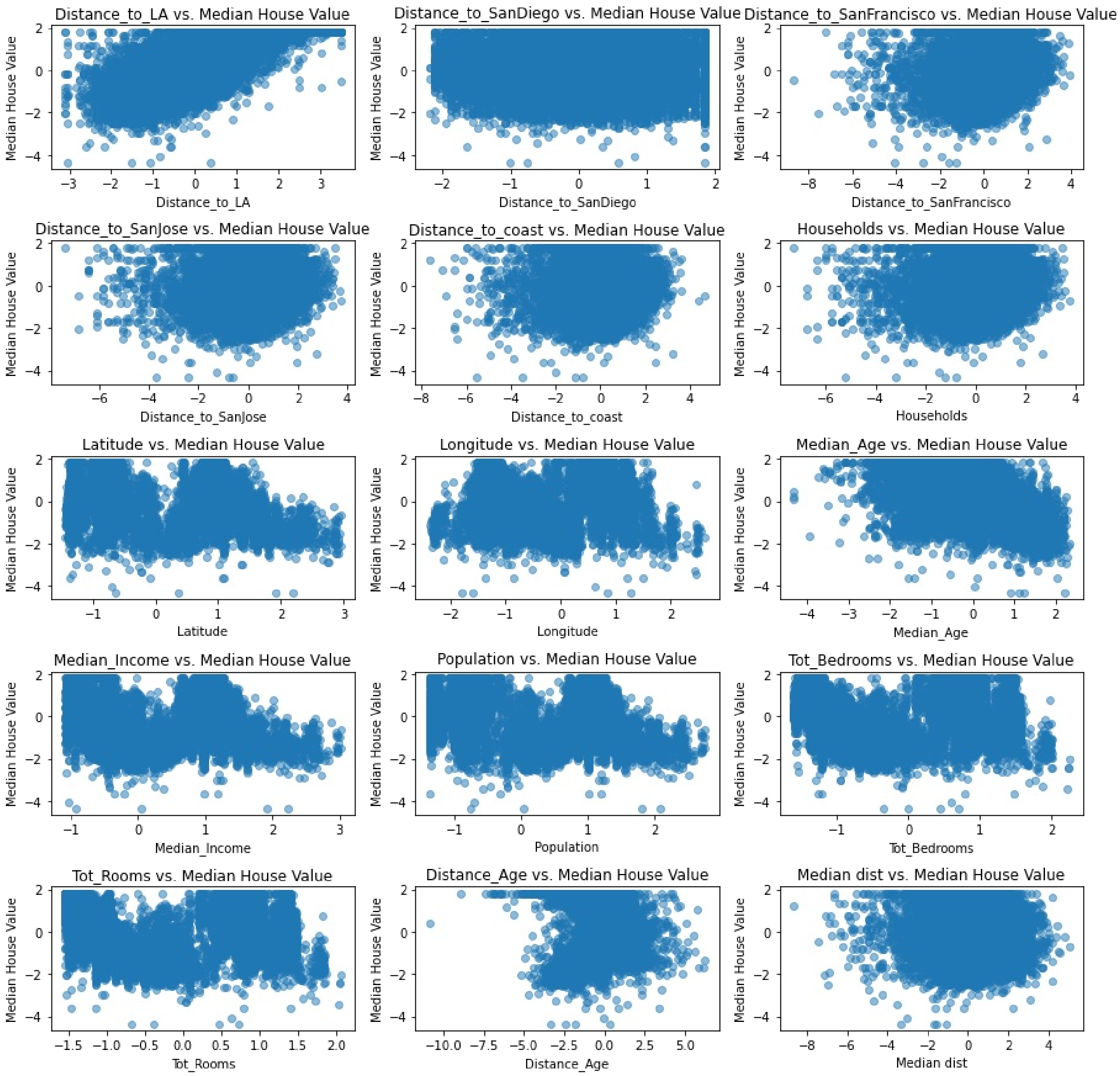
```python
# Sort the correlations in descending order
correlations_sorted = correlations_series.abs().sort_values(ascending=False)

# Display the correlations
print("Correlations with Median_House_Value:")
print(correlations_sorted)


# Combine 'Distance_to_LA' and 'Median_Age' to create a new feature
X['Distance_Age'] = X['Distance_to_LA'] * X['Median_Age']
# Combine 'Distance_to_LA' and 'Median_Age' to create another new feature
X['Median dist' ] = X['Distance_to_coast'] + X['Median_Income']
```

```
Correlations with Median_House_Value:
Distance_to_LA              0.681271
Median_Age                  0.545302
Latitude                    0.192596
Distance_to_SanFrancisco    0.186376
Median_Income               0.174126
Population                  0.136793
Households                  0.113635
Distance_to_SanJose         0.088626
Distance_to_SanDiego        0.076007
Distance_to_coast           0.026385
Longitude                   0.023209
Tot_Bedrooms                0.017556
Tot_Rooms                   0.005726
dtype: float64
```

## Visualize Realtionship between each feature and target (price) After Data Preprocessing:

In [9]:

```python
# Create basic scatter plots to visualize relationships between X features and y after Data Preprocessing:
fig, axes = plt.subplots(nrows=5, ncols=3, figsize=(15, 15))
fig.subplots_adjust(hspace=0.5)

for i, feature in enumerate(X.columns):
    row, col = i // 3, i % 3
    axes[row, col].scatter(X[feature], y, alpha=0.5)
    axes[row, col].set_title(f'{feature} vs. Median House Value')
    axes[row, col].set_xlabel(feature)
    axes[row, col].set_ylabel('Median House Value')

plt.show()
```




# 5.Split the data into training (70%), validation (15%), and testing (15%)

In [10]:

```
# Extract the target variable
y = data.iloc[:, 0]   # the first column is 'Median House Value' [target]
X = data.iloc[:, 1:]  # All other columns are features

# First, We Splited the data into training (70%) and the rest (30%)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42)

# Then We Splited the 30% into  crossvalidation set and testing set
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

# Display the shapes of the resulting sets
print("Training Data - X_train shape:", X_train.shape)
print("Training Data - y_train shape:", y_train.shape)
print("Cross Validation Data - X_val shape:", X_val.shape)
print("Cross Validation Data - y_val shape:", y_val.shape)
print("Testing Data - X_test shape:", X_test.shape)
print("Testing Data - y_test shape:", y_test.shape)
```

```
Training Data - X_train shape: (14448, 13)
Training Data - y_train shape: (14448,)
Cross Validation Data - X_val shape: (3096, 13)
Cross Validation Data - y_val shape: (3096,)
Testing Data - X_test shape: (3096, 13)
Testing Data - y_test shape: (3096,)
```

# 4.Apply Linear regression

In [19]:

```
# Initialize the model
linear_regressor = LinearRegression()
```

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

# Make predictions on the validation set
linear_regressor_prediction = linear_regressor.predict(X_val)

# Calculate and print mean squared error and mean absolute error for Linear regression
linear_mse = mean_squared_error(y_val, linear_regressor_prediction)
linear_mae = mean_absolute_error(y_val, linear_regressor_prediction)
print("Mean Squared Error Of Linear Regression =", linear_mse)
print("Mean Absolute Error Of Linear Regression =", linear_mae)
```

```
Mean Squared Error Of Linear Regression = 0.10130599074050421
Mean Absolute Error Of Linear Regression = 0.23393468937926465
```

## 5.Apply Lasso Regression

In [12]:

```
# Define hyperparameter grids for Lasso
lasso_param_grid = {'alpha': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100]}

# Initialize GridSearchCV with the respective regressors and parameter grids
lasso_regressor = GridSearchCV(Lasso(max_iter=10000), lasso_param_grid, cv=5, scoring='neg_mean_squared_error')

# Fit the grids to the training data
lasso_regressor.fit(X_train, y_train)

# Get the best hyperparameters for Lasso
best_lasso = lasso_regressor.best_estimator_

# Now, retrain the Lasso regressor with the best hyperparameters
best_lasso.fit(X_train, y_train)
print(best_lasso)

# Make predictions on the validation set using the best Lasso model
lasso_regressor_prediction = best_lasso.predict(X_val)

# Calculate and print mean squared error and mean absolute error for the best Lasso model
lasso_mse = mean_squared_error(y_val, lasso_regressor_prediction)
lasso_mae = mean_absolute_error(y_val, lasso_regressor_prediction)
print("Mean Squared Error Of Best Lasso Regression =", lasso_mse)
print("Mean Absolute Error Of Best Lasso Regression =", lasso_mae)
```

```
Lasso(alpha=0.0001, max_iter=10000)
Mean Squared Error Of Best Lasso Regression = 0.10127881291265355
Mean Absolute Error Of Best Lasso Regression = 0.23395521147021486
```

## 6.Apply Ridge Regression

In [20]:

```
# Define hyperparameter grids for Ridge
ridge_param_grid = {'alpha': [0.01, 0.1, 1, 10, 100]}

# Initialize GridSearchCV with the respective regressors and parameter grids
ridge_regressor = GridSearchCV(Ridge(), ridge_param_grid, cv=5, scoring='neg_mean_squared_error')

# Fit the grids to the training data
ridge_regressor.fit(X_train, y_train)

# Get the best hyperparameters for Ridge
best_ridge = ridge_regressor.best_estimator_

# Make predictions on the validation set using the best Ridge model
ridge_regressor_prediction = best_ridge.predict(X_val)

# Calculate and print mean squared error and mean absolute
ridge_mse = mean_squared_error(y_val, ridge_regressor_prediction)
ridge_mae = mean_absolute_error(y_val, ridge_regressor_prediction)
print("Mean Squared Error Of Ridge Regression =", ridge_mse)
print("Mean Absolute Error Of Ridge Regression =", ridge_mae)
```

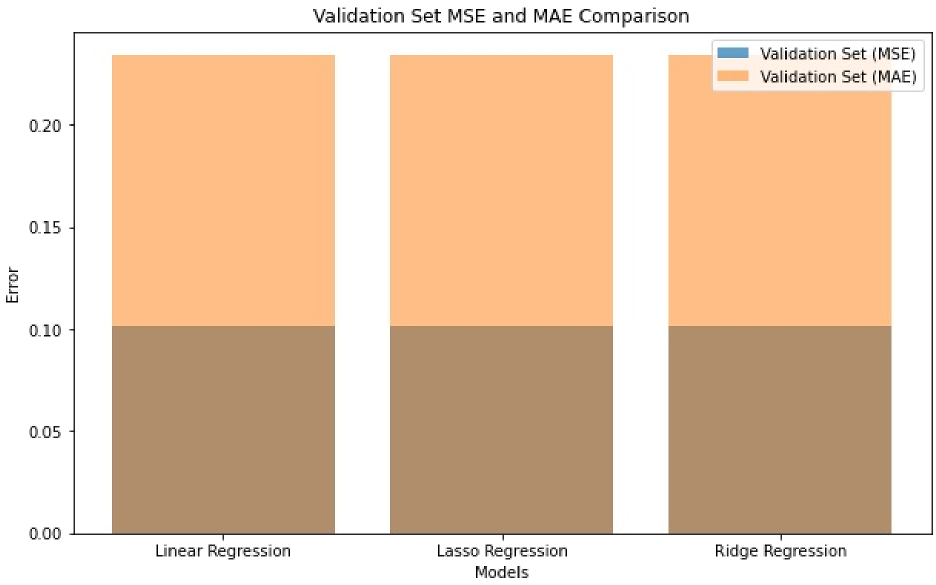
```
Mean Squared Error Of Ridge Regression = 0.10130438141390889
Mean Absolute Error Of Ridge Regression = 0.23393495832967653
```

## 7.Plot Performance

In [14]:

```
# Create a list of model names
models = ['Linear Regression', 'Lasso Regression', 'Ridge Regression']

# Create lists of MSE and MAE for validation set only
mse_validation = [linear_mse, lasso_mse, ridge_mse]
mae_validation = [linear_mae, lasso_mae, ridge_mae]

# Plot the MSE and MAE for validation set only
plt.figure(figsize=(10, 6))
plt.bar(models, mse_validation, label='Validation Set (MSE)', alpha=0.7)
plt.bar(models, mae_validation, label='Validation Set (MAE)', alpha=0.5)
```

```
plt.xlabel('Models')
plt.ylabel('Error')
plt.title('Validation Set MSE and MAE Comparison')
plt.legend()
plt.show()
```




# Compute MSE and MAE for test set

In [15]:

```
# Calculate mean squared error and mean absolute error for linear regression (test set)
linear_regressor_prediction = linear_regressor.predict(X_test)
linear_mse_test = mean_squared_error(y_test, linear_regressor_prediction)
linear_mae_test = mean_absolute_error(y_test, linear_regressor_prediction)
print("Mean Squared Error Of Linear Regression =", linear_mse_test)
print("Mean Absolute Error Of Linear Regression =", linear_mae_test)
```

```
Mean Squared Error Of Linear Regression = 0.09071771379333025
Mean Absolute Error Of Linear Regression = 0.2276109916345666
```

In [16]:

```
# Calculate mean squared error and mean absolute error for lasso resgression (test set)
lasso_regressor_prediction = lasso_regressor.predict(X_test)
lasso_mse_test = mean_squared_error(y_test, lasso_regressor_prediction)
lasso_mae_test = mean_absolute_error(y_test, lasso_regressor_prediction)
print("Mean Squared Error Of Lasso Regression =", lasso_mse_test)
print("Mean Absolute Error Of Lasso Regression =", lasso_mae_test)
```

```
Mean Squared Error Of Lasso Regression = 0.0907038110654024
Mean Absolute Error Of Lasso Regression = 0.2276920114604557
```

In [17]:

```
# Calculate mean squared error and mean absolute for ridge regression (test set)
ridge_regressor_prediction = ridge_regressor.predict(X_test)
ridge_mse_test = mean_squared_error(y_test, ridge_regressor_prediction)
ridge_mae_test = mean_absolute_error(y_test, ridge_regressor_prediction)
print("Mean Squared Error Of Ridge Regression =", ridge_mse_test)
print("Mean Absolute Error Of Ridge Regression =", ridge_mae_test)
```

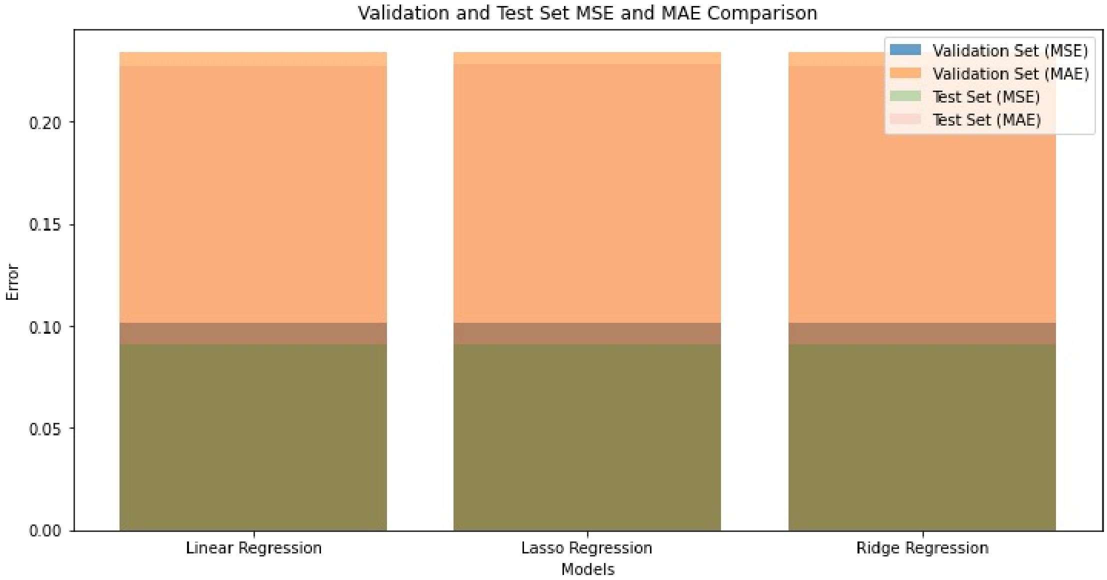
```
Mean Squared Error Of Ridge Regression = 0.09071697131615189
Mean Absolute Error Of Ridge Regression = 0.22761328573626052
```

# Residual Plot before and after testing

In [18]:

```
# Create a list of model names
models = ['Linear Regression', 'Lasso Regression', 'Ridge Regression']

# Create lists of MSE and MAE for validation set
mse_validation = [linear_mse, lasso_mse, ridge_mse]
mae_validation = [linear_mae, lasso_mae, ridge_mae]

# Create lists of MSE and MAE for the test set
mse_test = [linear_mse_test, lasso_mse_test, ridge_mse_test]
mae_test = [linear_mae_test, lasso_mae_test, ridge_mae_test]

# Plot the MSE and MAE for validation and test sets
plt.figure(figsize=(12, 6))

plt.bar(models, mse_validation, label='Validation Set (MSE)', alpha=0.7)
plt.bar(models, mae_validation, label='Validation Set (MAE)', alpha=0.5)
plt.bar(models, mse_test, label='Test Set (MSE)', alpha=0.3)
plt.bar(models, mae_test, label='Test Set (MAE)', alpha=0.1)

plt.xlabel('Models')
plt.ylabel('Error')
```

```
plt.title('Validation and Test Set MSE and MAE Comparison')
plt.legend()
plt.show()
```




Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js