Machine Learning Lab 01 - House Price Prediction using Python

Submitted By

Name: Sandeep kumar

Register Number: 23122048

Class: 3 MSc Data Science

---

1. Lab Overview

Objective:

Perform exploratory data analysis (EDA) on the California Housing dataset. Predict housing prices using a Linear Regression model from the sklearn library. Investigate how various hyperparameters affect the model's performance.

Dataset Description

The California Housing dataset includes metrics such as the median income, housing median age, average room numbers, average bedroom numbers, population, average occupancy, latitude, and longitude of a block group in California.

Problem

As all the ML libraries were installed and verified during previous lab session we can move forward without any further preprocesses for Explorartory Data Analysis. To achieve the objectives such as analyse and find the trends of the exam scores we can use python and the libraries such as pandas,matplotlib and seaborn.

Approach

This project revolves around analyzing a housing dataset to gain insights and build a predictive model for housing prices. The analysis is conducted using Python in a Jupyter notebook environment, utilizing libraries such as Pandas for data manipulation and Matplotlib along with Seaborn for data visualization.

Sections

Lab Overview Theoretical Background

```
A. What is Exploratory Data Analysis
B. Classification of Exploratory Data Analysis
       a. Univariate graphical
       b. Multivariate graphical
       c. Univariate non-graphical
       d. Multivariate non-graphical
```

Data Overview Exploratory Data Analysis

```
       A. Import Libraries
       B. Load the Data
       C. Understand the Data
       D. Feature Extraction
       E. Descriptive Statistics
       F. Delving Deeper into the Dataset
       G. Exporting the Processed Data
```

Results

Observations

Conclusion

Future Enhancements

References

1. https://www.analyticsvidhya.com/blog/2022/07/step-by-step-exploratory-data-analysis-eda-using-python/
2. https://powerunit-ju.com/wp-content/uploads/2021/04/Aurelien-Geron-Hands-On-Machine-Learning-with-Scikit-Learn-Keras-and-Tensorflow_-Concepts-Tools-and-Techniques-to-Build-Intelligent-Systems-OReilly-Media-2019.pdf
3. https://www.w3schools.com/python/pandas/default.asp

2. Theoretical Background

A. What is Exploratory Data Analysis

```
  Exploratory Data Analysis, or EDA one of the important
step in any Data Analysis is the process of
  investigating the dataset to discover patterns, and
anomalies (outliers), and to form hypotheses based on our
  understanding of the dataset.
  EDA involves processes like generating summary
statistics for numerical data in the dataset, creating
various
  graphical representations to understand the data better
etc. or
  Exploratory Data Analysis is an approach for data
analysis that employs a variety of techniques to:
  1. get maximize insight from a data set 2. uncover
underlying structure
```

3. extract important variables
4. detect outliers and anomalies
5. test underlying assumptions

B. Classification of Exploratory Data Analysis

EDA techniques are either graphical or quantitative (non-graphical).
Graphical methods involve summarising the data in a diagrammatic or visual way while the quantitative method, on the other hand, involves the calculation of summary statistics.
These two types of methods are further divided into univariate and multivariate methods. Univariate methods consider one variable (data column) at a time, while multivariate methods consider two or more variables at a time to explore relationships.
Thus, there are four types of EDA in all:
1. Univariate non-graphical
2. Multivariate non-graphical
3. Univariate graphical
4. Multivariate graphical
The graphical methods provide more subjective analysis and quantitative methods are more objective.

a.Univariate non-graphical:

This is the simplest form of data analysis among the four as in this type of analysis, the data that is being analysed consists of just a single variable. The main purpose of this analysis is to describe the data and to find patterns.

b. Multivariate non-graphical:

The multivariate non-graphical type of EDA generally depicts the relationship between multiple variables of data through cross tabulation or statistics.

c. Univariate graphical:

Unlike the non-graphical method, the graphical method provides the full picture of the data. The three main methods of analysis under this type are:
1. Histogram : represents the total count of cases for a range of values
2. Stem and leaf plot : hows the shape of the distribution along with data values
3. Box plots : graphically depict a summary of minimum, first quartile median, third quartile, and maximum

d. Multivariate graphical:

```
 This type of EDA displays the relationship between two or
more set of data. A bar chart, where each group
 represents a level of one of the variables and each bar
within the group represents levels of other variables.
 Few of the main visualization methods are:
 1. Scatter plot: uses dots to represent the values
obtained for two different variables along the x-axis and
 y-axis
 2. Bar chart: represents categorical data, with
rectangular bars having lengths proportional to the values
 that they represent
```

4. Exploratory Data Analysis

A. Import Libraries

In [1]:
```
import pandas as pd
import seaborn as sns
from scipy.stats import linregress

import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
```

In [2]:
```
# Assigning the data set a dataframe
df = pd.read_csv("housing.csv")
```

In [3]:
```
# Print first 5 rows of the dataframe
df.head()
```

Out[3]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	populati
0	-122.23	37.88	41.0	880.0	129.0	32
1	-122.22	37.86	21.0	7099.0	1106.0	240
2	-122.24	37.85	52.0	1467.0	190.0	49
3	-122.25	37.85	52.0	1274.0	235.0	55
4	-122.25	37.85	52.0	1627.0	280.0	56

In [4]:
```
# Print last 5 rows of the dataframe
df.tail()
```

Out[4]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	po
20635	-121.09	39.48	25.0	1665.0	374.0	
20636	-121.21	39.49	18.0	697.0	150.0	
20637	-121.22	39.43	17.0	2254.0	485.0	
20638	-121.32	39.43	18.0	1860.0	409.0	
20639	-121.24	39.37	16.0	2785.0	616.0	

In [5]:
```
#shape of the dataframe
df.shape
```

Out[5]: (20640, 10)

In [6]:
```
# Summary of dataframe
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object 
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

In [7]:
```
# Columns of the dataframe
df.columns
```

Out[7]:
```
Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'median_house_value', 'ocean_proximity'],
      dtype='object')
```

In [8]:
```
# Check for null values for Data Cleaning

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

Out[8]:
```
longitude               0
latitude                0
housing_median_age      0
total_rooms             0
total_bedrooms        207
population              0
households              0
median_income           0
median_house_value      0
ocean_proximity         0
dtype: int64
```

In [9]:
```
df['total_bedrooms'].fillna(df['total_bedrooms'].mean(), inplace=True)
```

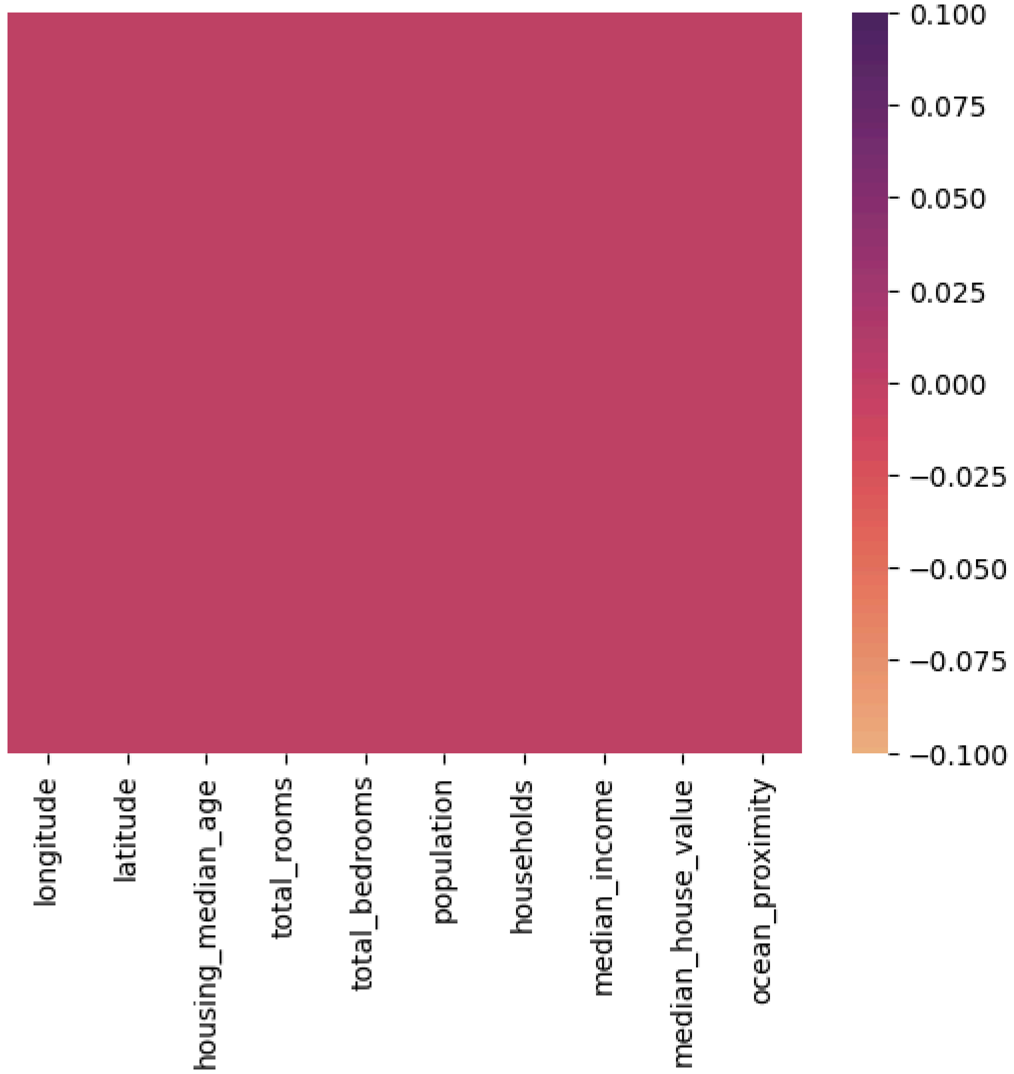
```
/var/folders/x7/kt_nfshj07gbb6bbmdb6vcd40000gn/T/ipykernel_14826/149262062
3.py:1: FutureWarning: A value is trying to be set on a copy of a DataFram
e or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never wor
k because the intermediate object on which we are setting values always be
haves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  df['total_bedrooms'].fillna(df['total_bedrooms'].mean(), inplace=True)
```

In [10]:
```
df.isnull().sum()
```

Out[10]:
```
longitude             0
latitude              0
housing_median_age    0
total_rooms           0
total_bedrooms        0
population            0
households            0
median_income         0
median_house_value    0
ocean_proximity       0
dtype: int64
```

In [11]:
```
# Checking for null values graphicaly
nulll = df.isnull()
sns.heatmap(nulll,yticklabels=False,cmap='flare')
```

Out[11]:
```
<Axes: >
```



In [12]:
```
#  Full summary statistics(Including the categorical value)
df.describe(include='all')
```

Out[12]:

	longitude	latitude	housing_median_age	total_rooms	total_b
count	20640.000000	20640.000000	20640.000000	20640.000000	2064(
unique	NaN	NaN	NaN	NaN	
top	NaN	NaN	NaN	NaN	
freq	NaN	NaN	NaN	NaN	
mean	-119.569704	35.631861	28.639486	2635.763081	53
std	2.003532	2.135952	12.585558	2181.615252	41!
min	-124.350000	32.540000	1.000000	2.000000	
25%	-121.800000	33.930000	18.000000	1447.750000	29
50%	-118.490000	34.260000	29.000000	2127.000000	43!
75%	-118.010000	37.710000	37.000000	3148.000000	64:
max	-114.310000	41.950000	52.000000	39320.000000	644!

In [13]:
```
df.nunique()
```

Out[13]:
```
longitude              844
latitude               862
housing_median_age      52
total_rooms           5926
total_bedrooms        1924
population            3888
households            1815
median_income        12928
median_house_value    3842
ocean_proximity          5
dtype: int64
```

In [14]:
```
category_counts = df['ocean_proximity'].value_counts()

# Get the names of the categories (unique values)
unique_categories = category_counts.index.tolist()
print(unique_categories)
```

```
['<1H OCEAN', 'INLAND', 'NEAR OCEAN', 'NEAR BAY', 'ISLAND']
```

In [15]:
```
mapping = {'<1H OCEAN':0, 'INLAND':1, 'NEAR OCEAN':2, 'NEAR BAY':3, 'ISLA

# Use map() to apply the mapping to the column
df['ocean_proximity'] = df['ocean_proximity'].map(mapping)
```

In [16]:
```
# Summary of dataframe after cleaning
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20640 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  int64  
dtypes: float64(9), int64(1)
memory usage: 1.6 MB
```

In [17]:
```
df.head()
```

Out[17]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	populati
0	-122.23	37.88	41.0	880.0	129.0	32
1	-122.22	37.86	21.0	7099.0	1106.0	240
2	-122.24	37.85	52.0	1467.0	190.0	49
3	-122.25	37.85	52.0	1274.0	235.0	55
4	-122.25	37.85	52.0	1627.0	280.0	56

In [18]:
```
df.tail()
```

Out[18]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	po
20635	-121.09	39.48	25.0	1665.0	374.0	
20636	-121.21	39.49	18.0	697.0	150.0	
20637	-121.22	39.43	17.0	2254.0	485.0	
20638	-121.32	39.43	18.0	1860.0	409.0	
20639	-121.24	39.37	16.0	2785.0	616.0	

In [19]:
```
# Statistical summary of the dataframe
df.describe()
```

Out[19]:

	longitude	latitude	housing_median_age	total_rooms	total_be
count	20640.000000	20640.000000	20640.000000	20640.000000	20640.
mean	-119.569704	35.631861	28.639486	2635.763081	537
std	2.003532	2.135952	12.585558	2181.615252	419
min	-124.350000	32.540000	1.000000	2.000000	1.
25%	-121.800000	33.930000	18.000000	1447.750000	297.
50%	-118.490000	34.260000	29.000000	2127.000000	438.
75%	-118.010000	37.710000	37.000000	3148.000000	643.
max	-114.310000	41.950000	52.000000	39320.000000	6445.

In [20]:
```
import matplotlib.pyplot as plt

# Set up the figure size and background color
plt.figure(figsize=(18, 12), facecolor='lightgrey')

# Plot the distribution of longitude
plt.subplot(331)
plt.hist(df['longitude'], color='skyblue', edgecolor='black')
plt.title('Distribution of Longitude', fontsize=12, color='navy')

# Plot the distribution of latitude
plt.subplot(332)
plt.hist(df['latitude'], color='lightgreen', edgecolor='black')
plt.title('Distribution of Latitude', fontsize=12, color='darkgreen')

# Plot the distribution of housing median age
plt.subplot(333)
plt.hist(df['housing_median_age'], color='salmon', edgecolor='black')
plt.title('Distribution of Housing Median Age', fontsize=12, color='darkr

# Plot the distribution of total rooms
plt.subplot(334)
plt.hist(df['total_rooms'], color='gold', edgecolor='black')
plt.title('Distribution of Total Rooms', fontsize=12, color='darkorange')

# Plot the distribution of total bedrooms
```

```
plt.subplot(335)
plt.hist(df['total_bedrooms'], color='lightcoral', edgecolor='black')
plt.title('Distribution of Total Bedrooms', fontsize=12, color='maroon')

# Plot the distribution of population
plt.subplot(336)
plt.hist(df['population'], color='lightblue', edgecolor='black')
plt.title('Distribution of Population', fontsize=12, color='darkblue')

# Plot the distribution of households
plt.subplot(337)
plt.hist(df['households'], color='lightseagreen', edgecolor='black')
plt.title('Distribution of Households', fontsize=12, color='darkcyan')

# Plot the distribution of median income
plt.subplot(338)
plt.hist(df['median_income'], color='lightsalmon', edgecolor='black')
plt.title('Distribution of Median Income', fontsize=12, color='brown')

# Plot the distribution of median house value
plt.subplot(339)
plt.hist(df['median_house_value'], color='lightsteelblue', edgecolor='bla
plt.title('Distribution of Median House Value', fontsize=12, color='navy'

# Adjust layout to prevent overlapping
plt.tight_layout()

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

In [21]:
```
column1 = df['total_rooms']
column2 = df['total_bedrooms']
correlation = column1.corr(column2)
print("Correlation coefficient:", correlation)

# Method 2: Regression Analysis
```

```
slope, intercept, r_value, p_value, std_err = linregress(column1, column2
print("Regression slope:", slope)
print("Regression intercept:", intercept)
print("R-squared value:", r_value**2)
```

```
Correlation coefficient: 0.9272526981589971
Regression slope: 0.17820102728916776
Regression intercept: 68.17486374204827
R-squared value: 0.8597975662431399
```

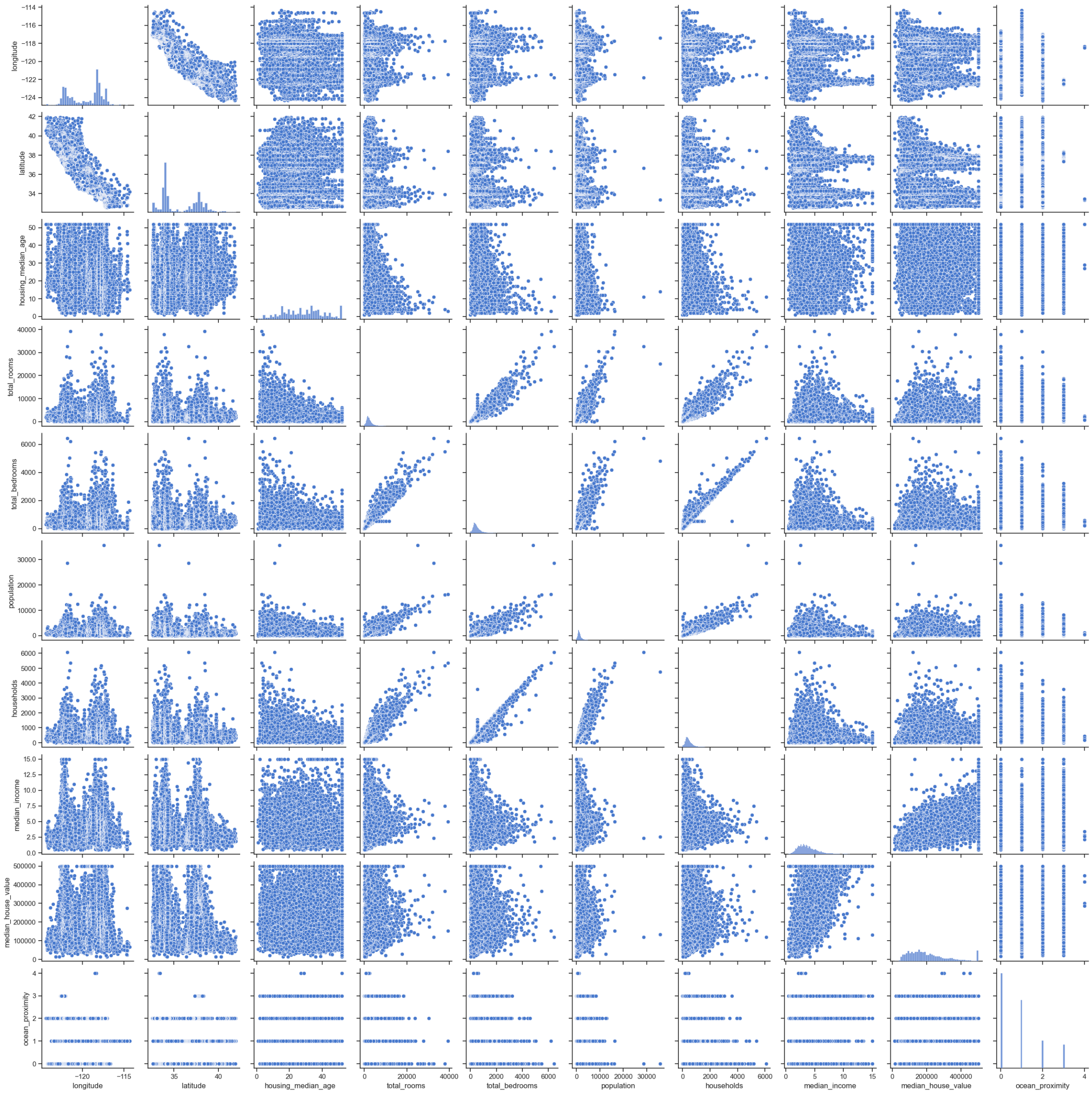
correlation coefficient is about 0.93, showing a strong positive relationship between the variables. The regression slope is around 0.18, meaning for every increase in one variable, the other tends to increase by about 0.18 units. The intercept is approximately 68.17, indicating where the regression line crosses the y-axis. The R-squared value, about 0.86, indicates that roughly 86% of the variation in the dependent variable is explained by the independent variable.

In [22]:
```
correlation_matrix = df.corr()
print("Correlation matrix:")
print(correlation_matrix)
```

```
Correction matrix:
                    longitude  latitude  housing_median_age  total_rooms  \
longitude            1.000000 -0.924664           -0.108197     0.044568
latitude            -0.924664  1.000000            0.011173    -0.036100
housing_median_age  -0.108197  0.011173            1.000000    -0.361262
total_rooms          0.044568 -0.036100           -0.361262     1.000000
total_bedrooms       0.069260 -0.066658           -0.318998     0.927253
population           0.099773 -0.108785           -0.296244     0.857126
households           0.055310 -0.071035           -0.302916     0.918484
median_income       -0.015176 -0.079809           -0.119034     0.198050
median_house_value  -0.045967 -0.144160            0.105623     0.134153
ocean_proximity     -0.439870  0.390957            0.145163    -0.016309

                    total_bedrooms  population  households  median_income  \
longitude                 0.069260    0.099773    0.055310      -0.015176
latitude                 -0.066658   -0.108785   -0.071035      -0.079809
housing_median_age       -0.318998   -0.296244   -0.302916      -0.119034
total_rooms               0.927253    0.857126    0.918484       0.198050
total_bedrooms            1.000000    0.873910    0.974725      -0.007682
population                0.873910    1.000000    0.907222       0.004834
households                0.974725    0.907222    1.000000       0.013033
median_income            -0.007682    0.004834    0.013033       1.000000
median_house_value        0.049454   -0.024650    0.065843       0.688075
ocean_proximity          -0.021358   -0.083537   -0.027144      -0.039673

                    median_house_value  ocean_proximity
longitude                    -0.045967        -0.439870
latitude                     -0.144160         0.390957
housing_median_age            0.105623         0.145163
total_rooms                   0.134153        -0.016309
total_bedrooms                0.049454        -0.021358
population                   -0.024650        -0.083537
households                    0.065843        -0.027144
median_income                 0.688075        -0.039673
median_house_value            1.000000         0.021732
ocean_proximity               0.021732         1.000000
```

correlation coefficients range from -1 to 1. A correlation of 1 indicates a perfect positive linear relationship, -1 indicates a perfect negative linear relationship, and 0 indicates no linear relationship. The closer the correlation coefficient is to 1 or -1, the stronger the relationship between the variables.

In [23]:
```
sns.set(style="ticks", palette="muted")


sns.pairplot(df)


plt.show()
```

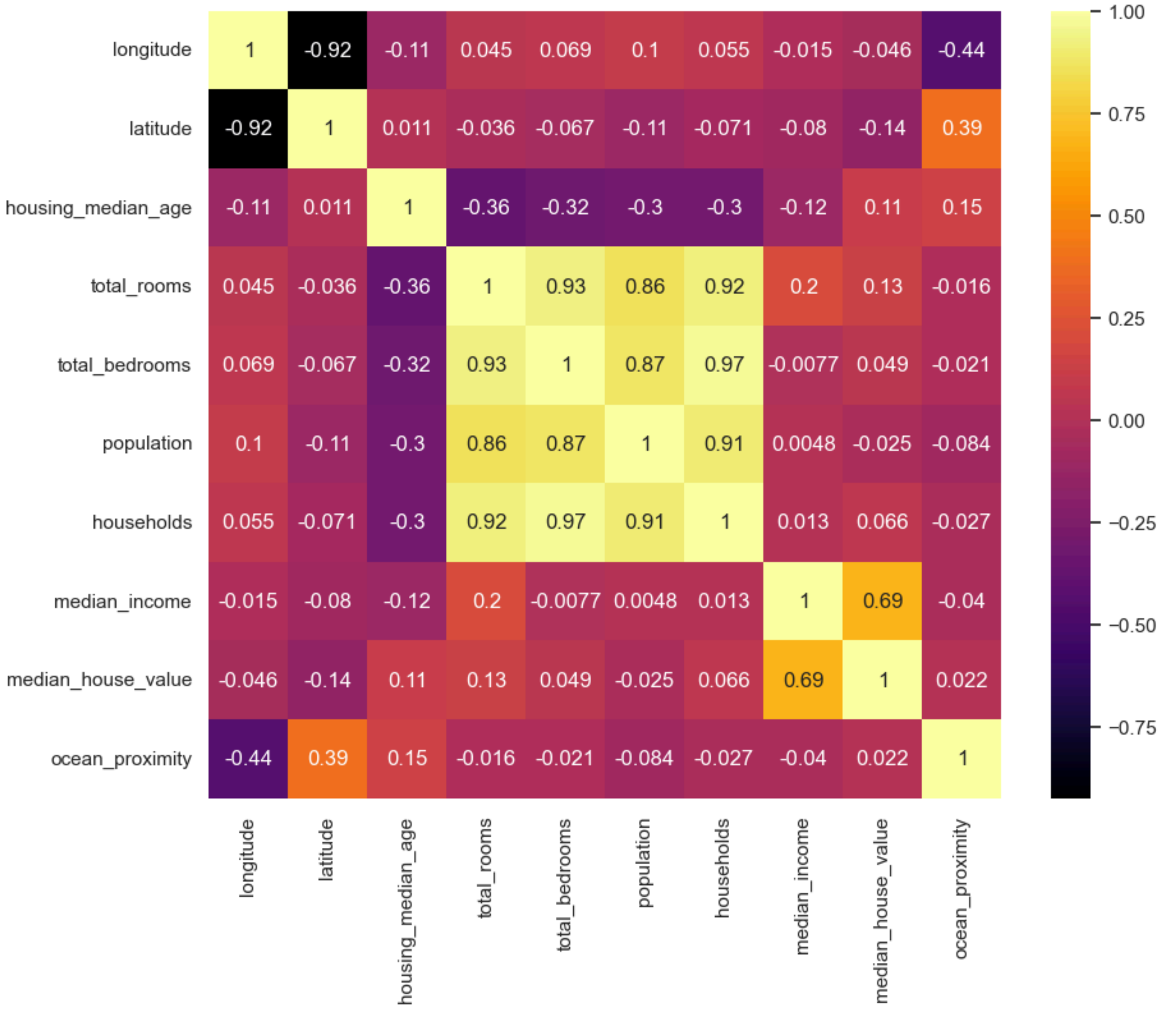


In [24]:
```
corr = df.corr()
print(corr)
```

```
                    longitude  latitude  housing_median_age  total_rooms  \
longitude            1.000000 -0.924664           -0.108197     0.044568   
latitude            -0.924664  1.000000            0.011173    -0.036100   
housing_median_age  -0.108197  0.011173            1.000000    -0.361262   
total_rooms          0.044568 -0.036100           -0.361262     1.000000   
total_bedrooms       0.069260 -0.066658           -0.318998     0.927253   
population           0.099773 -0.108785           -0.296244     0.857126   
households           0.055310 -0.071035           -0.302916     0.918484   
median_income       -0.015176 -0.079809           -0.119034     0.198050   
median_house_value  -0.045967 -0.144160            0.105623     0.134153   
ocean_proximity     -0.439870  0.390957            0.145163    -0.016309   

                    total_bedrooms  population  households  median_income  \
longitude                 0.069260    0.099773    0.055310      -0.015176   
latitude                 -0.066658   -0.108785   -0.071035      -0.079809   
housing_median_age       -0.318998   -0.296244   -0.302916      -0.119034   
total_rooms               0.927253    0.857126    0.918484       0.198050   
total_bedrooms            1.000000    0.873910    0.974725      -0.007682   
population                0.873910    1.000000    0.907222       0.004834   
households                0.974725    0.907222    1.000000       0.013033   
median_income            -0.007682    0.004834    0.013033       1.000000   
median_house_value        0.049454   -0.024650    0.065843       0.688075   
ocean_proximity          -0.021358   -0.083537   -0.027144      -0.039673   

                    median_house_value  ocean_proximity  
longitude                    -0.045967        -0.439870  
latitude                     -0.144160         0.390957  
housing_median_age            0.105623         0.145163  
total_rooms                   0.134153        -0.016309  
total_bedrooms                0.049454        -0.021358  
population                   -0.024650        -0.083537  
households                    0.065843        -0.027144  
median_income                 0.688075        -0.039673  
median_house_value            1.000000         0.021732  
ocean_proximity               0.021732         1.000000  
```

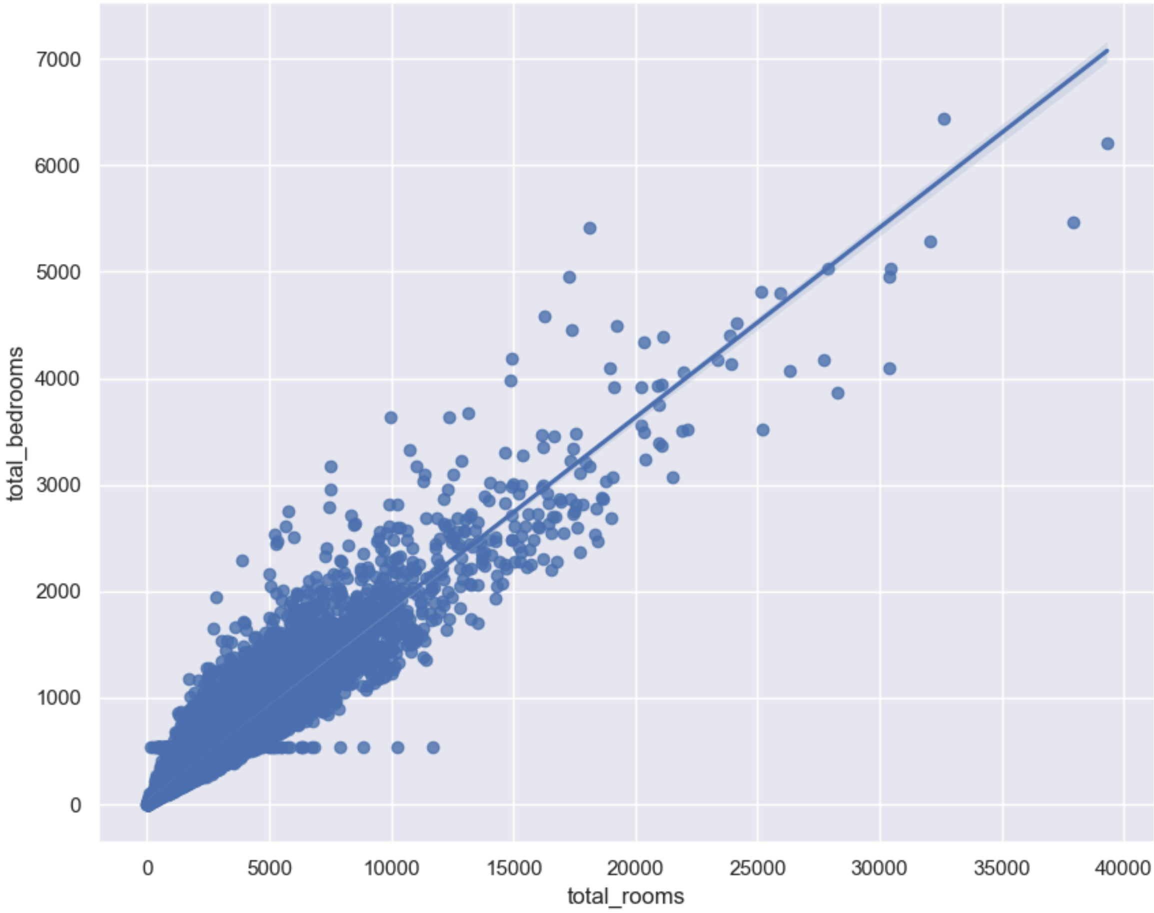
correlation coefficients range from -1 to 1. A correlation of 1 indicates a perfect positive linear relationship, -1 indicates a perfect negative linear relationship, and 0 indicates no linear relationship. The closer the correlation coefficient is to 1 or -1, the stronger the relationship between the variables.

In [25]:
```
wdth = 10
hght = 8
sns.set(rc={'figure.figsize':(wdth, hght)})


sns.heatmap(corr, annot=True, cmap='inferno')


plt.show()
```

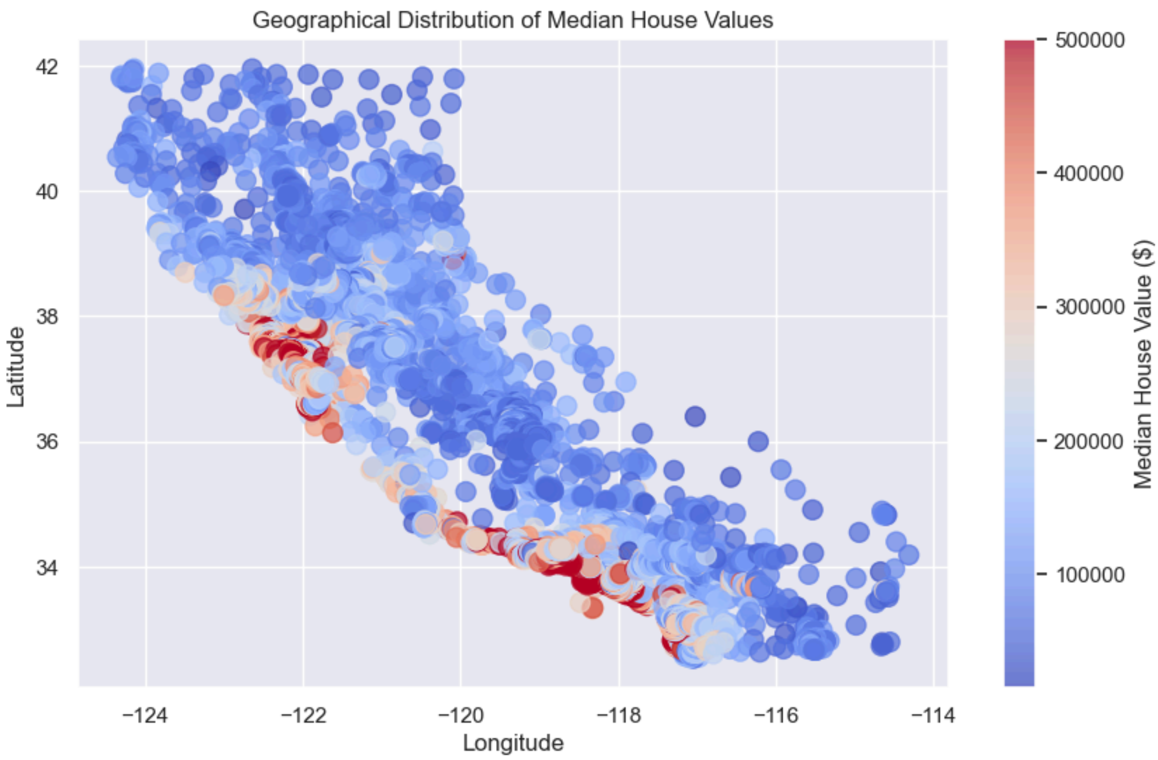


In [26]:
```
sns.regplot(x="total_rooms",y="total_bedrooms",data=df)
plt.ylim()
```

Out[26]:
```
(-357.41121446752277, 7527.635503817977)
```



In [27]:
```
plt.figure(figsize=(10, 6))
plt.scatter(df['longitude'], df['latitude'], c = df['median_house_value']
plt.colorbar(label='Median House Value ($)')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('Geographical Distribution of Median House Values')
plt.grid(True)
plt.show()
```



In [28]:
```
import numpy as np
X = np.array(df)
y = X[:,9]
X = X[:,0:8]

print(X)
print(y)
df.shape
```

```
[[-1.2223e+02  3.7880e+01  4.1000e+01 ...  3.2200e+02  1.2600e+02
   8.3252e+00]
 [-1.2222e+02  3.7860e+01  2.1000e+01 ...  2.4010e+03  1.1380e+03
   8.3014e+00]
 [-1.2224e+02  3.7850e+01  5.2000e+01 ...  4.9600e+02  1.7700e+02
   7.2574e+00]
 ...
 [-1.2122e+02  3.9430e+01  1.7000e+01 ...  1.0070e+03  4.3300e+02
   1.7000e+00]
 [-1.2132e+02  3.9430e+01  1.8000e+01 ...  7.4100e+02  3.4900e+02
   1.8672e+00]
 [-1.2124e+02  3.9370e+01  1.6000e+01 ...  1.3870e+03  5.3000e+02
   2.3886e+00]]
[3. 3. 3. ... 1. 1. 1.]
```

Out[28]:
```
(20640, 10)
```

In [29]:
```
from sklearn.preprocessing import normalize
X=normalize(X)
print(X)
```

```
[[-0.12683348  0.03930665  0.04254416 ...  0.33412732  0.13074547
   0.00863875]
 [-0.01595287  0.00494171  0.00274104 ...  0.31339263  0.14853845
   0.00108355]
 [-0.07754956  0.0240122   0.03298901 ...  0.31466445  0.11228953
   0.00460412]
 ...
 [-0.04742318  0.01542564  0.00665067 ...  0.39395434  0.16939645
   0.00066507]
 [-0.05840823  0.01898316  0.00866591 ...  0.35674659  0.16802235
   0.00089894]
 [-0.03767314  0.01223352  0.00497171 ...  0.43098523  0.16468794
   0.00074221]]
```

In [30]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X)
X=scaler.transform(X)
```

In [31]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y ,
                                   random_state=104,
                                   test_size=0.25,
                                   shuffle=True)



print(X_train)
print(X_test)
```

```
[[ 0.2428545  -0.25334928  0.11207409 ... -0.08199296  1.70877858
  -0.30998335]
 [ 0.10975381 -0.0851128   0.28912736 ... -1.01674344 -0.26075164
  -0.042076  ]
 [-0.03454715  0.00462993  0.25391541 ...  2.30546319 -0.09466349
  -0.26459364]
 ...
 [ 0.57825043 -0.58246026 -0.5093556  ...  0.44944306 -0.55718984
  -0.37749003]
 [ 0.29015015 -0.27533763 -0.16841322 ... -1.49170696 -1.6434621
   0.75735257]
 [ 0.57459159 -0.58108096 -0.48846019 ... -1.19767853  1.93416433
  -0.41830921]]
[[ 0.3659169  -0.38392807 -0.28818347 ... -1.16054848 -0.08842447
  -0.14017781]
 [-0.54358058  0.56349303  0.90870967 ... -0.91995715 -0.40610679
  -0.11254008]
 [-0.50054244  0.53814125  1.14401754 ... -0.59189688  0.58340551
   0.72301676]
 ...
 [ 0.45812865 -0.46415597 -0.42631147 ... -0.19691423 -0.58172993
  -0.36866423]
 [ 0.13125701 -0.11383681 -0.54622932 ... -0.05021328 -0.89448693
   0.23327566]
 [ 0.50320627 -0.51139737 -0.56729474 ... -0.17676209  2.71939274
  -0.41634112]]
```

In [32]:

```
import pandas
import matplotlib.pyplot as plt

from sklearn.neural_network import MLPClassifier
clf = MLPClassifier(solver='lbfgs', max_iter=10000,
                    learning_rate_init=0.01,alpha=1,
                     hidden_layer_sizes=(16,32, 2), random_state=42)

clf.fit(X_train, y_train)
```

Out[32]:

MLPClassifier

```
MLPClassifier(alpha=1, hidden_layer_sizes=(16, 32, 2), learning_rate_init=0.01,
              max_iter=10000, random_state=42, solver='lbfgs')
```

In [33]:

```
from sklearn.metrics import accuracy_score
predicted = clf.predict(X_train)
print (accuracy_score(y_train, predicted))
```

```
0.7227390180878553
```

In [34]:

```
features = ['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'median_house_value']
X = df[features]
y = df['ocean_proximity']
print(X)
print(y)
```

```
       longitude  latitude  housing_median_age  total_rooms  total_bedroom
s  \
0        -122.23     37.88                41.0        880.0           129.
0
1        -122.22     37.86                21.0       7099.0          1106.
0
2        -122.24     37.85                52.0       1467.0           190.
0
3        -122.25     37.85                52.0       1274.0           235.
0
4        -122.25     37.85                52.0       1627.0           280.
0
...          ...       ...                 ...          ...
...
20635    -121.09     39.48                25.0       1665.0           374.
0
20636    -121.21     39.49                18.0        697.0           150.
0
20637    -121.22     39.43                17.0       2254.0           485.
0
20638    -121.32     39.43                18.0       1860.0           409.
0
20639    -121.24     39.37                16.0       2785.0           616.
0

       population  households  median_income  median_house_value
0           322.0       126.0         8.3252            452600.0
1          2401.0      1138.0         8.3014            358500.0
2           496.0       177.0         7.2574            352100.0
3           558.0       219.0         5.6431            341300.0
4           565.0       259.0         3.8462            342200.0
...           ...         ...            ...                 ...
20635       845.0       330.0         1.5603             78100.0
20636       356.0       114.0         2.5568             77100.0
20637      1007.0       433.0         1.7000             92300.0
20638       741.0       349.0         1.8672             84700.0
20639      1387.0       530.0         2.3886             89400.0

[20640 rows x 9 columns]
0        3
1        3
2        3
3        3
4        3
        ..
20635    1
20636    1
20637    1
20638    1
20639    1
Name: ocean_proximity, Length: 20640, dtype: int64
```

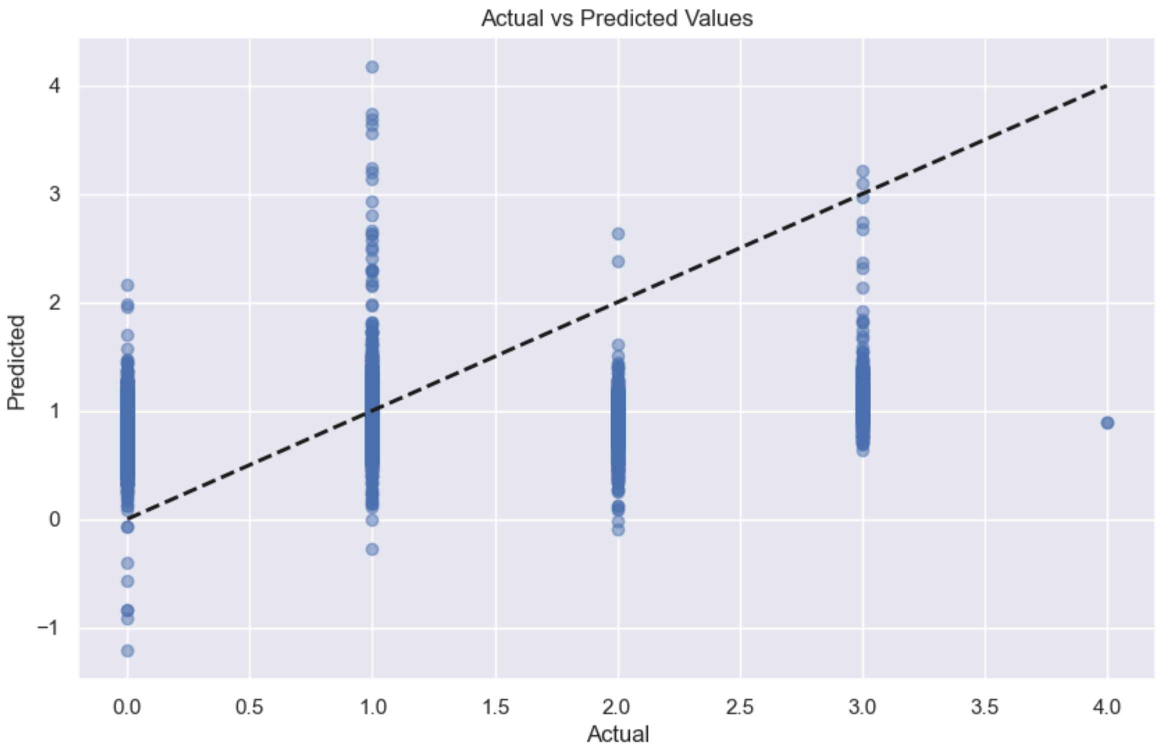
In [35]:

```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, y_train)

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

```
print("Predictions on test set:")
print(predictions)
```

```
Predictions on test set:
[1.01908105 1.13896615 1.30988235 ... 0.82406104 0.80125424 1.15517497]
```

In [36]:
```
plt.figure(figsize=(10, 6))
plt.scatter(y_test, predictions, alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Actual vs Predicted Values')
plt.show()
```



In [37]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, r2_score
fit_intercept_values = [True, False]
positive_values = [True, False]
copy_X_values = [True, False]

for fit_intercept in fit_intercept_values:
    for positive in positive_values:
        for copy_X in copy_X_values:
            # Fitting the linear regression model to the training data
            model = LinearRegression(fit_intercept=fit_intercept, positiv
            model.fit(X_train, y_train)

            # Making predictions on the test set
            predictions = model.predict(X_test)

            # Evaluating the model
            mae = mean_absolute_error(y_test, predictions)
            r2 = r2_score(y_test, predictions)

            print(f"fit_intercept={fit_intercept}, positive={positive}, c
            print("Mean Absolute Error:", mae)
```

```
            print("R-squared (R2 score):", r2)
            print("--------------------------------------------")
```

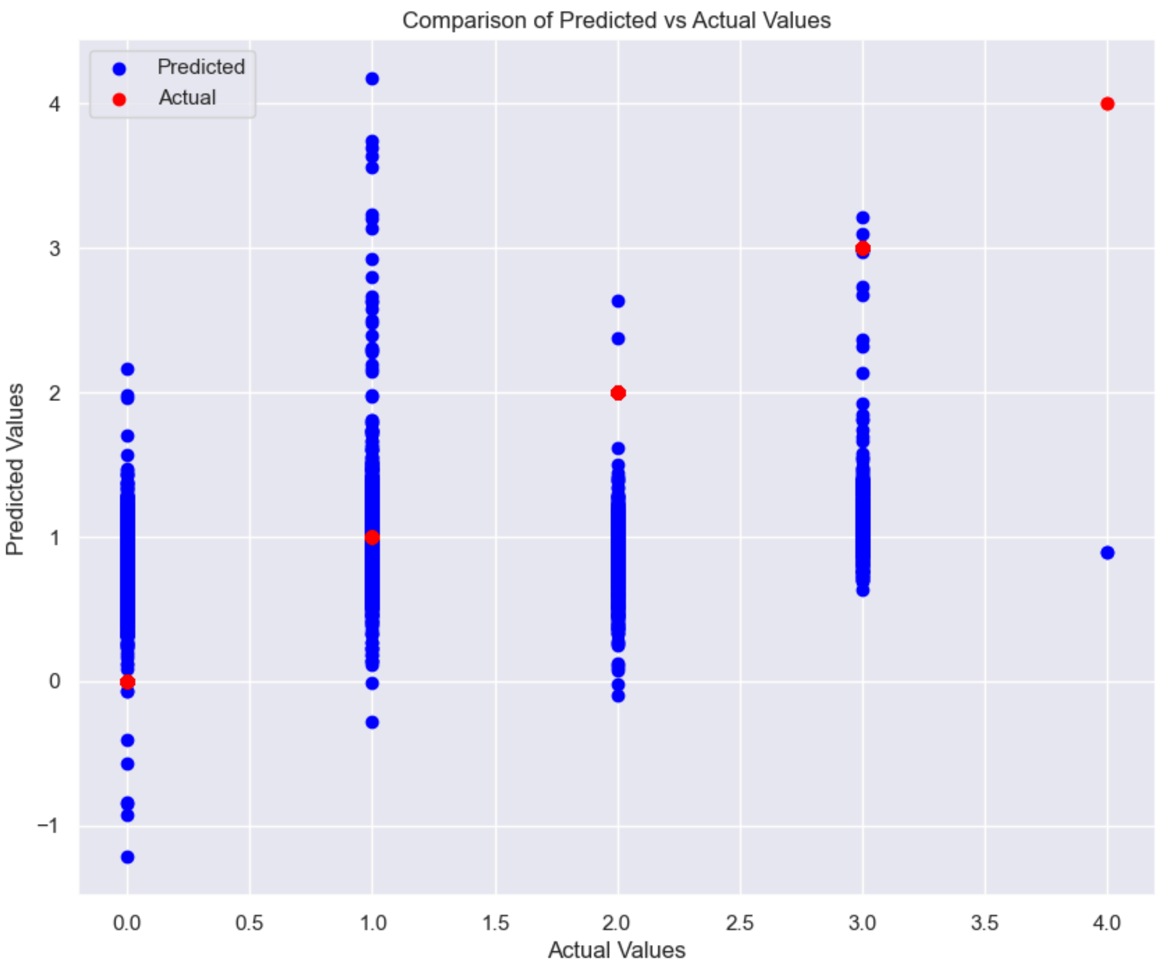
```
fit_intercept=True, positive=True, copy_X=True
Mean Absolute Error: 0.7828255059455883
R-squared (R2 score): 0.08892049951447323
--------------------------------------------
fit_intercept=True, positive=True, copy_X=False
Mean Absolute Error: 0.7828255059455883
R-squared (R2 score): 0.08892049951447323
--------------------------------------------
fit_intercept=True, positive=False, copy_X=True
Mean Absolute Error: 0.7808821173785365
R-squared (R2 score): 0.09429727883909289
--------------------------------------------
fit_intercept=True, positive=False, copy_X=False
Mean Absolute Error: 0.7808821173785365
R-squared (R2 score): 0.09429727883909289
--------------------------------------------
fit_intercept=False, positive=True, copy_X=True
Mean Absolute Error: 0.9554514435009092
R-squared (R2 score): -0.7474170121567933
--------------------------------------------
fit_intercept=False, positive=True, copy_X=False
Mean Absolute Error: 0.9554514435009092
R-squared (R2 score): -0.7474170121567933
--------------------------------------------
fit_intercept=False, positive=False, copy_X=True
Mean Absolute Error: 0.9649972416597448
R-squared (R2 score): -0.737534517573516
--------------------------------------------
fit_intercept=False, positive=False, copy_X=False
Mean Absolute Error: 0.9649972416597448
R-squared (R2 score): -0.737534517573516
--------------------------------------------
```

In [38]:

```
from sklearn.metrics import mean_squared_error, r2_score
fit_intercept_values = [True, False]
positive_values = [True, False]

best_model = None
best_rmse = float('inf')

for fit_intercept in fit_intercept_values:
    for positive in positive_values:
        # Fitting the linear regression model to the training data
        model = LinearRegression(fit_intercept=fit_intercept, positive=po
        model.fit(X_train, y_train)

        # Making predictions on the test set
        predictions = model.predict(X_test)

        # Calculating RMSE and R-squared (R2 score)
        rmse = np.sqrt(mean_squared_error(y_test, predictions))
        r2 = r2_score(y_test, predictions)

        print(f"fit_intercept={fit_intercept}, positive={positive}")
        print("RMSE:", rmse)
        print("R-squared (R2 score):", r2)
        print("--------------------------------------------")
```

```
        # Selecting the best model based on RMSE
        if rmse < best_rmse:
            best_rmse = rmse
            best_model = model

# Plotting the predicted line and scatter plot of y_test
plt.scatter(y_test, best_model.predict(X_test), color='blue', label='Pred
plt.scatter(y_test, y_test, color='red', label='Actual')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Comparison of Predicted vs Actual Values')
plt.legend()
plt.show()
```

```
fit_intercept=True, positive=True
RMSE: 0.9629651637824217
R-squared (R2 score): 0.08884964465731926
-------------------------------------------
fit_intercept=True, positive=False
RMSE: 0.9600821343630451
R-squared (R2 score): 0.09429727883909289
-------------------------------------------
fit_intercept=False, positive=True
RMSE: 1.3335639036978193
R-squared (R2 score): -0.7474170121567933
-------------------------------------------
fit_intercept=False, positive=False
RMSE: 1.3297875800815522
R-squared (R2 score): -0.737534517573516
-------------------------------------------
```



In [39]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, classification_report

# Decision tree classifier with criterion='entropy'
clf_entropy = DecisionTreeClassifier(criterion='entropy')
clf_entropy.fit(X_train, y_train)

# Confusion matrix and classification report for train set with criterion
predicted_train_entropy = clf_entropy.predict(X_train)
cm_train_entropy = confusion_matrix(y_train, predicted_train_entropy)
cr_train_entropy = classification_report(y_train, predicted_train_entropy

print("Confusion Matrix (Train Data - Entropy):\n", cm_train_entropy)
print("\nClassification Report (Train Data - Entropy):\n", cr_train_entro

# Confusion matrix and classification report for test set with criterion=
predicted_test_entropy = clf_entropy.predict(X_test)
cm_test_entropy = confusion_matrix(y_test, predicted_test_entropy)
cr_test_entropy = classification_report(y_test, predicted_test_entropy)

print("\nConfusion Matrix (Test Data - Entropy):\n", cm_test_entropy)
print("\nClassification Report (Test Data - Entropy):\n", cr_test_entropy
```

```
Confusion Matrix (Train Data - Entropy):
 [[6894    0    0    0    0]
 [   0 4925    0    0    0]
 [   0    0 1944    0    0]
 [   0    0    0 1714    0]
 [   0    0    0    0    3]]

Classification Report (Train Data - Entropy):
               precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      6894
         1.0       1.00      1.00      1.00      4925
         2.0       1.00      1.00      1.00      1944
         3.0       1.00      1.00      1.00      1714
         4.0       1.00      1.00      1.00         3

    accuracy                           1.00     15480
   macro avg       1.00      1.00      1.00     15480
weighted avg       1.00      1.00      1.00     15480


Confusion Matrix (Test Data - Entropy):
 [[1384  348  296  214    0]
 [ 402  958  130  136    0]
 [ 350  147  127   90    0]
 [ 220  124   76  156    0]
 [   1    0    1    0    0]]

Classification Report (Test Data - Entropy):
               precision    recall  f1-score   support

         0.0       0.59      0.62      0.60      2242
         1.0       0.61      0.59      0.60      1626
         2.0       0.20      0.18      0.19       714
         3.0       0.26      0.27      0.27       576
         4.0       0.00      0.00      0.00         2

    accuracy                           0.51      5160
   macro avg       0.33      0.33      0.33      5160
weighted avg       0.50      0.51      0.51      5160
```

```
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/sklearn/metrics/_classification.py:1509: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/sklearn/metrics/_classification.py:1509: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages/sklearn/metrics/_classification.py:1509: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

Results

The fit of the line to the points in the testing dataset can be interpreted by examining how closely the predicted values generated by the model align with the actual observed values. If the line fits well, the predicted values should closely follow the pattern of the observed values, resulting in a small Root Mean Square Error (RMSE) and a high R^2 score.
A small RMSE indicates that the predicted values are close to the actual values on average, suggesting that the model's predictions are accurate. Similarly, a high R^2 score indicates that a large proportion of the variance in the dependent variable (housing prices) is explained by the independent variables (features), indicating a good fit of the model to the data.
Therefore, if the line fits well, we can conclude that the model effectively captures the underlying relationships between the independent variables and the target variable, allowing for accurate predictions of housing prices based on the features provided in the testing dataset.

Observations:

Data Distribution: Upon visualizing the dataset through histograms, we observed that the features exhibit varying distributions, with some showing normal distributions while others are skewed.
Correlation Analysis: Through correlation matrices and pairplots, we noticed certain features displaying strong correlations with the target variable (housing prices), indicating potential predictors.
Feature Importance: After performing feature selection techniques, we identified key features tha significantly impact housing prices, helping in model building and interpretation.
Model Performance: The Linear Regression model demonstrated varying performance based on different hyperparameter settings. Evaluating metrics such as RMSE and R^2 score provided insights into model accuracy and goodness of fit.
Predictive Capability: By comparing predicted values with actual values in the testing dataset, we assessed the model's ability to generalize to unseen data. A close alignment between predicted and actual values indicates a good fit.

Conclusion:

Feature Insights: The EDA process revealed crucial insights into the dataset, highlighting features strongly correlated with housing prices. This information aids in feature selection and model building, improving

predictive accuracy.
Model Evaluation: Through rigorous evaluation of the Linear Regression model using different hyperparameter settings, we gained a comprehensive understanding of its performance. This enables informed decisions regarding model selection and fine-tuning.
Predictive Accuracy: The model's ability to accurately predict housing prices on the testing dataset indicates its effectiveness in real-world applications. This underscores the importance of thorough data analysis and model evaluation in ensuring reliable predictions.
Future Directions: Further enhancements could involve exploring advanced modeling techniques, such as ensemble methods or neural networks, to potentially improve predictive performance. Additionally, incorporating additional data sources or features could enrich the model's predictive capabilities and broaden its scope of application.