Final_Project_Template

April 6, 2024

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

The California Housing Data set from 1990 is a popular dataset often used for machine learning projects, particularly in regression analysis and geographic data visualization. It includes information about the housing in the California region, as recorded during the 1990 U.S. census.

Features:

- Median Income: The median income for households within a block of houses (measured in tens of thousands of USD).
- Housing Median Age: he median age of a house within a block; a lower number is a newer building.
- Total Rooms: The total number of rooms in the houses per block.
- Total Bedrooms: The total number of bedrooms in the houses per block.
- Population: The total number of people residing within a block.
- Households: The total number of households, a group of people residing within a home unit, for a block.
- Latitude: The block's latitude.
- Longitude: The block's longitude.
- Ocean Proximity: distance from the house to the ocean

Target Variable:

- Median House Value: This variable represents the median value of houses within a specific block, and it's what machine learning models aim to predict based on the other features in the dataset. Predicting the Median House Value helps in understanding housing market trends, assessing affordability, and planning economic and housing policies.

[2]:
```
filename_data = 'data/housing.csv'
data = pd.read_csv(filename_data)
```

[3]:
```
data.head(15)
```

[3]:
```
    longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
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
5   -122.25   37.85                52.0      919.0           213.0
6   -122.25   37.84                52.0     2535.0           489.0
7   -122.25   37.84                52.0     3104.0           687.0
8   -122.26   37.84                42.0     2555.0           665.0
9   -122.25   37.84                52.0     3549.0           707.0
10  -122.26   37.85                52.0     2202.0           434.0
11  -122.26   37.85                52.0     3503.0           752.0
12  -122.26   37.85                52.0     2491.0           474.0
13  -122.26   37.84                52.0      696.0           191.0
14  -122.26   37.85                52.0     2643.0           626.0

    population  households  median_income  median_house_value ocean_proximity
0        322.0       126.0         8.3252            452600.0        NEAR BAY
1       2401.0      1138.0         8.3014            358500.0        NEAR BAY
2        496.0       177.0         7.2574            352100.0        NEAR BAY
3        558.0       219.0         5.6431            341300.0        NEAR BAY
4        565.0       259.0         3.8462            342200.0        NEAR BAY
5        413.0       193.0         4.0368            269700.0        NEAR BAY
6       1094.0       514.0         3.6591            299200.0        NEAR BAY
7       1157.0       647.0         3.1200            241400.0        NEAR BAY
8       1206.0       595.0         2.0804            226700.0        NEAR BAY
9       1551.0       714.0         3.6912            261100.0        NEAR BAY
10       910.0       402.0         3.2031            281500.0        NEAR BAY
11      1504.0       734.0         3.2705            241800.0        NEAR BAY
12      1098.0       468.0         3.0750            213500.0        NEAR BAY
13       345.0       174.0         2.6736            191300.0        NEAR BAY
14      1212.0       620.0         1.9167            159200.0        NEAR BAY
```

0.0.1 Getting to know the data

[4]:
```
data.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
```

[5]:
```python
data.describe()
```

[5]:
```
          longitude      latitude  housing_median_age   total_rooms  \
count  20640.000000  20640.000000        20640.000000  20640.000000
mean    -119.569704     35.631861           28.639486   2635.763081
std        2.003532      2.135952           12.585558   2181.615252
min     -124.350000     32.540000            1.000000      2.000000
25%     -121.800000     33.930000           18.000000   1447.750000
50%     -118.490000     34.260000           29.000000   2127.000000
75%     -118.010000     37.710000           37.000000   3148.000000
max     -114.310000     41.950000           52.000000  39320.000000

       total_bedrooms    population    households  median_income  \
count    20433.000000  20640.000000  20640.000000   20640.000000
mean       537.870553   1425.476744    499.539680       3.870671
std        421.385070   1132.462122    382.329753       1.899822
min          1.000000      3.000000      1.000000       0.499900
25%        296.000000    787.000000    280.000000       2.563400
50%        435.000000   1166.000000    409.000000       3.534800
75%        647.000000   1725.000000    605.000000       4.743250
max       6445.000000  35682.000000   6082.000000      15.000100

       median_house_value
count        20640.000000
mean        206855.816909
std         115395.615874
min          14999.000000
25%         119600.000000
50%         179700.000000
75%         264725.000000
max         500001.000000
```

[6]:
```python
data.columns[data.isnull().any()]
data.isnull().sum()
```

[6]:
```
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
```

Since there are null values in total bedrooms, we must perform operations on total bedrooms.

[7]:
```
data[data["total_bedrooms"].isnull()]
```

[7]:
```
       longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
290      -122.16     37.77                47.0       1256.0             NaN
341      -122.17     37.75                38.0        992.0             NaN
538      -122.28     37.78                29.0       5154.0             NaN
563      -122.24     37.75                45.0        891.0             NaN
696      -122.10     37.69                41.0        746.0             NaN
…            …         …                   …           …               …
20267    -119.19     34.20                18.0       3620.0             NaN
20268    -119.18     34.19                19.0       2393.0             NaN
20372    -118.88     34.17                15.0       4260.0             NaN
20460    -118.75     34.29                17.0       5512.0             NaN
20484    -118.72     34.28                17.0       3051.0             NaN

       population  households  median_income  median_house_value  \
290         570.0       218.0         4.3750            161900.0
341         732.0       259.0         1.6196             85100.0
538        3741.0      1273.0         2.5762            173400.0
563         384.0       146.0         4.9489            247100.0
696         387.0       161.0         3.9063            178400.0
…             …           …             …                  …
20267      3171.0       779.0         3.3409            220500.0
20268      1938.0       762.0         1.6953            167400.0
20372      1701.0       669.0         5.1033            410700.0
20460      2734.0       814.0         6.6073            258100.0
20484      1705.0       495.0         5.7376            218600.0

      ocean_proximity
290          NEAR BAY
341          NEAR BAY
538          NEAR BAY
563          NEAR BAY
696          NEAR BAY
…                 …
20267      NEAR OCEAN
20268      NEAR OCEAN
20372       <1H OCEAN
20460       <1H OCEAN
20484       <1H OCEAN
```

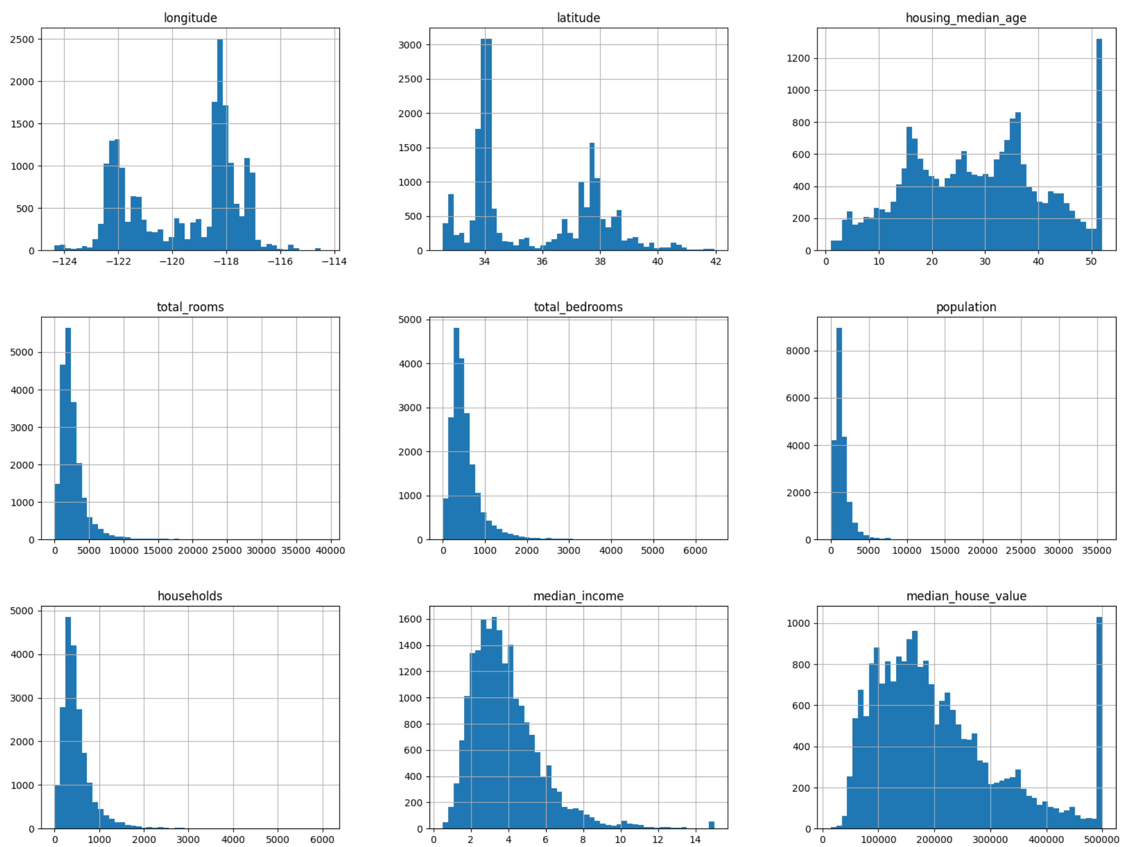
```
[207 rows x 10 columns]
```

[8]:
```python
data["total_bedrooms"] = data["total_bedrooms"].fillna(data["total_bedrooms"].
 ↪mean())
missing_total_bedrooms_values = data[data["total_bedrooms"].isnull()]
data[data["total_bedrooms"].isnull()]
```

[8]:
```
Empty DataFrame
Columns: [longitude, latitude, housing_median_age, total_rooms, total_bedrooms,
population, households, median_income, median_house_value, ocean_proximity]
Index: []
```

In total bedrooms, the average value of the existing values was assigned instead of the NaN values.

[9]:
```python
data.hist(bins=50, figsize=(20,15))
plt.show()
```



Since the ocean_proximity variable is an object, it was removed from the data because it caused problems in calculating the correlation matrix and did not have any effect on the target variable.

[10]:
```python
data = data.drop('ocean_proximity', axis=1)
```

One way of observing the strength of relationship between a feature and the target variable is to compute the correlation coefficient. The correlation coefficient ranges from -1 to 1. When it is close to 1, it means that there is a strong positive correlation. Fore exmple, the target tends to go up when the median_income goes up. When the coefficient is close to -1, it means that there is a strong negative corelation. Coefficients close to zero mean that there is no linear correlation.

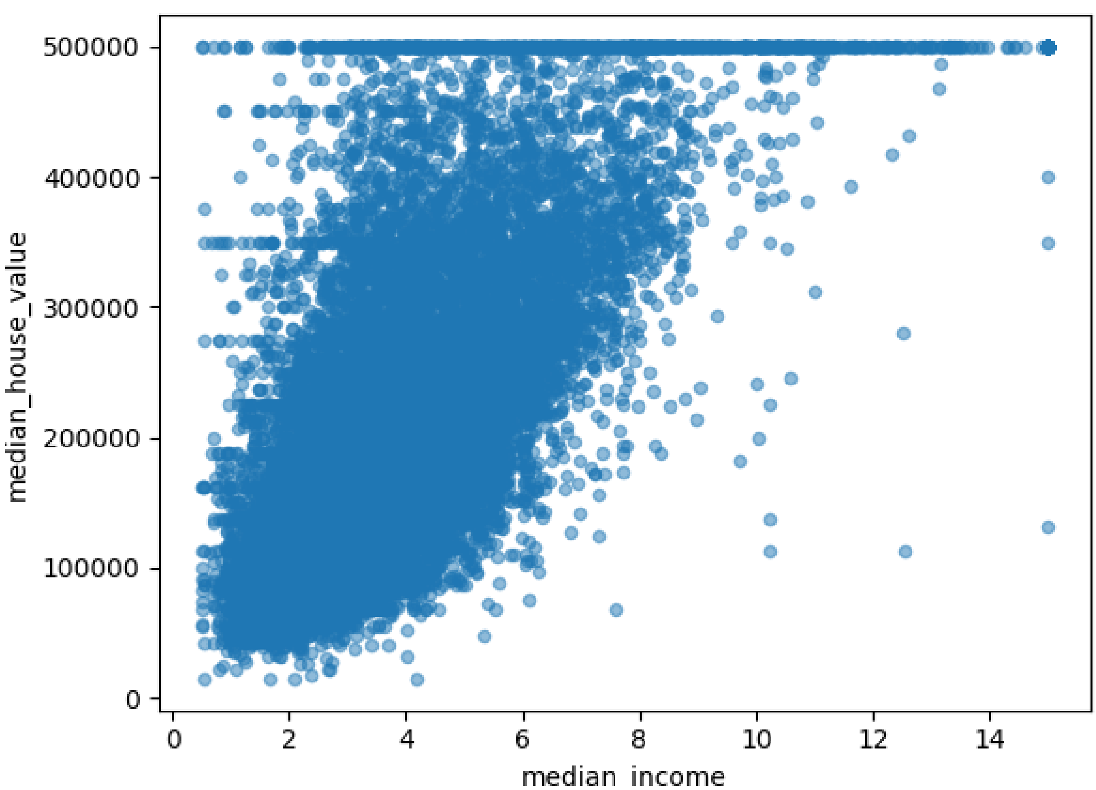
[11]:
```
corr_matrix=data.corr()
corr_matrix['median_house_value'].sort_values(ascending=False)
```

[11]:
```
median_house_value    1.000000
median_income         0.688075
total_rooms           0.134153
housing_median_age    0.105623
households            0.065843
total_bedrooms        0.049454
population           -0.024650
longitude            -0.045967
latitude             -0.144160
Name: median_house_value, dtype: float64
```

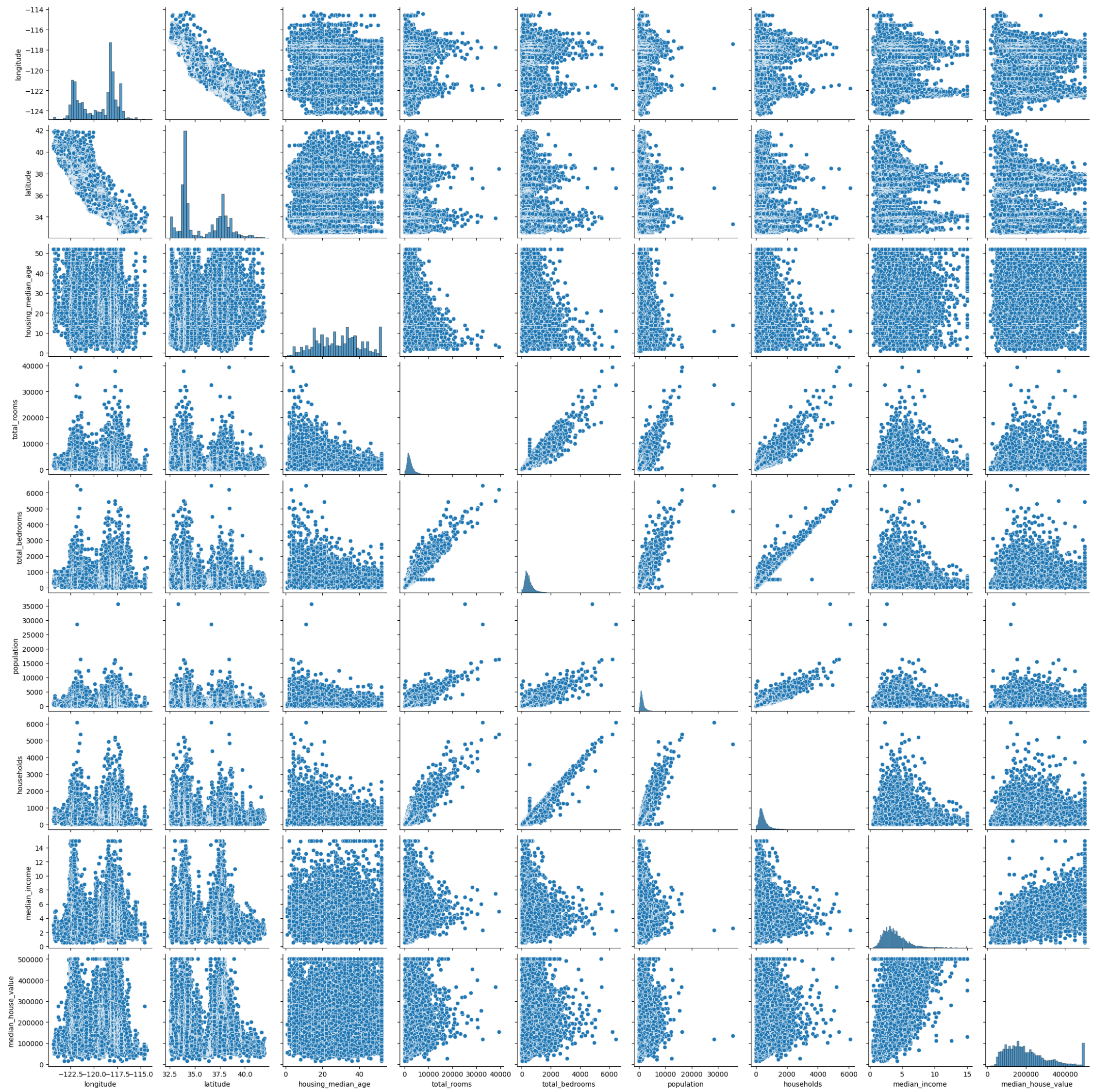
The relationship between the variables can also be observed visually, by plotting the scatter plot.

[12]:
```
data.plot(kind='scatter', x='median_income', y='median_house_value', alpha=0.5)
```

[12]:
```
<Axes: xlabel='median_income', ylabel='median_house_value'>
```



[13]:
```
sns.pairplot(data)
plt.show()
```



All remaining features were observed to have a connection with the target variable.

0.0.2 Train/ Test Split

[14]:
```
from sklearn.model_selection import train_test_split
train_set, test_set= train_test_split(data, test_size=0.2, random_state=42)
```

[15]:
```
print("Number of examples in train set:",train_set.shape[0])
print("Number of examples in test set:",test_set.shape[0])
```

```
Number of examples in train set: 16512
Number of examples in test set: 4128
```

[16]:
```
trainX=train_set.drop('median_house_value', axis=1)
trainY=train_set['median_house_value'].copy()

testX=test_set.drop('median_house_value', axis=1)
testY=test_set['median_house_value'].copy()
```

[17]:
```
print(trainX.shape)
print(trainY.shape)
print(testX.shape)
print(testY.shape)
```

```
(16512, 8)
(16512,)
(4128, 8)
(4128,)
```

0.0.3 Linear Regression

[18]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
trainX_scaled = scaler.fit_transform(trainX)
```

[19]:
```
print(trainX_scaled)
```

```
[[ 1.27258656 -1.3728112   0.34849025 …  0.76827628  0.32290591
  -0.326196  ]
 [ 0.70916212 -0.87669601  1.61811813 … -0.09890135  0.6720272
  -0.03584338]
 [-0.44760309 -0.46014647 -1.95271028 … -0.44981806 -0.43046109
   0.14470145]
 …
 [ 0.59946887 -0.75500738  0.58654547 …  0.28983345  0.07090859
  -0.49697313]
 [-1.18553953  0.90651045 -1.07984112 …  0.30830275  0.15490769
   0.96545045]
 [-1.41489815  0.99543676  1.85617335 …  1.04883375  1.94776365
  -0.68544764]]
```

[20]:
```
from sklearn.linear_model import LinearRegression
model=LinearRegression()

model.fit(trainX_scaled, trainY)
```

[20]: LinearRegression()

[21]:
```
from sklearn.metrics import mean_squared_error, r2_score

testX_scaled = scaler.transform(testX)
y_pred = model.predict(testX_scaled)
mse = mean_squared_error(testY, y_pred)
r2=r2_score(testY, y_pred)
print("Mean Squared Error on Test Set:", mse)
print("r2 score on Test Set:", r2)


y_pred = model.predict(trainX_scaled)
mse = mean_squared_error(trainY, y_pred)
r2=r2_score(trainY, y_pred)
print("Mean Squared Error on Train Set:", mse)
print("r2 score on Train Set:", r2)
```

```
Mean Squared Error on Test Set: 5052953703.90163
r2 score on Test Set: 0.6143987268246023
Mean Squared Error on Train Set: 4811134397.884197
r2 score on Train Set: 0.6400947924305294
```

0.0.4 Polynomial Regression

[22]:
```
from sklearn.preprocessing import PolynomialFeatures
poly_features = PolynomialFeatures(degree=2, include_bias=False)
trainX_poly = poly_features.fit_transform(trainX_scaled)
print(trainX_poly.shape)
```

```
(16512, 44)
```

[23]:
```
# Fit the polynomial regression model
model_poly = LinearRegression()
model_poly.fit(trainX_poly, trainY)
```

[23]: LinearRegression()

[24]:
```
testX_poly = poly_features.transform(testX_scaled)
y_pred = model_poly.predict(testX_poly)
mse = mean_squared_error(testY, y_pred)
r2=r2_score(testY, y_pred)
print("Mean Squared Error on Test Set:", mse)
print("r2 score on Test Set:", r2)


y_pred = model_poly.predict(trainX_poly)
mse = mean_squared_error(trainY, y_pred)
```

```
r2=r2_score(trainY, y_pred)
print("Mean Squared Error on Train Set:", mse)
print("r2 score on Train Set:", r2)
```

```
Mean Squared Error on Test Set: 4577486232.73672
r2 score on Test Set: 0.6506826259019103
Mean Squared Error on Train Set: 3988584883.9903984
r2 score on Train Set: 0.7016270276689182
```

0.0.5 Ridge Regression

[25]:
```
from sklearn.linear_model import Ridge
ridge_reg = Ridge(alpha=1.0)
ridge_reg.fit(trainX_scaled, trainY)
```

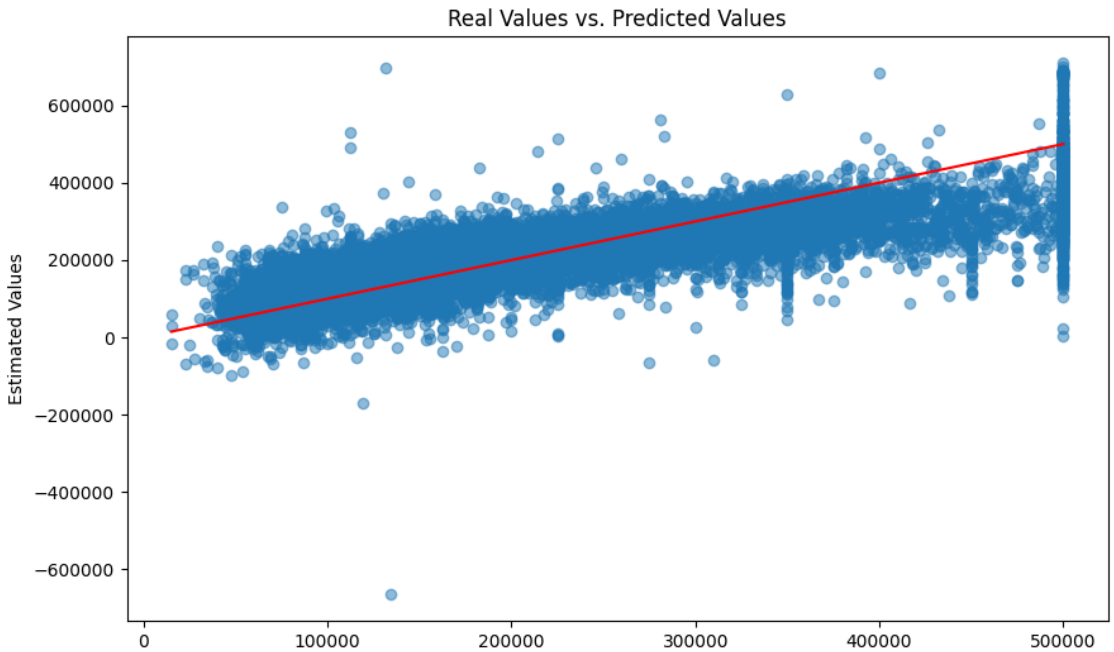
[25]:
```
Ridge()
```

[26]:
```
testX_scaled = scaler.transform(testX)
y_pred = ridge_reg.predict(testX_scaled)
mse = mean_squared_error(testY, y_pred)
r2=r2_score(testY, y_pred)
print("Mean Squared Error on Test Set:", mse)
print("r2 score on Test Set:", r2)


y_pred = ridge_reg.predict(trainX_scaled)
mse = mean_squared_error(trainY, y_pred)
r2=r2_score(trainY, y_pred)
print("Mean Squared Error on Train Set:", mse)
print("r2 score on Train Set:", r2)
```

```
Mean Squared Error on Test Set: 5052475906.315732
r2 score on Test Set: 0.6144351885395197
Mean Squared Error on Train Set: 4811135535.758371
r2 score on Train Set: 0.6400947073098889
```

[27]:
```
plt.figure(figsize=(10, 6))
plt.scatter(trainY, y_pred, alpha=0.5)
plt.plot([min(trainY), max(trainY)], [min(trainY), max(trainY)], color='red')
plt.ylabel('Estimated Values')
plt.title('Real Values vs. Predicted Values')
plt.show()
```



Real Values vs. Predicted Values
Estimated Values
600000
400000
200000
0
−200000
−400000
−600000
0
100000
200000
300000
400000
500000