2203A51530 ML ASSIGNMENT

```
import pandas as pd
import seaborn as sns
import os
import numpy as np
import matplotlib.pyplot as plt

housing_df = pd.read_csv('/content/housing.csv')

# Use .info() to show the features (i.e. columns) in your dataset
along with a count and datatype
housing_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
```

```
housing_df.shape
```

```
(20640, 10)
```

```
housing_df.head()
```


{"summary":"{\n \"name\": \"housing_df\",\n \"rows\": 20640,\n
\"fields\": [\n   {\n   \"column\": \"longitude\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
2.0035317235025882,\n   \"min\": -124.35,\n   \"max\": -
114.31,\n   \"num_unique_values\": 844,\n   \"samples\": [\n
-118.63,\n   -119.86,\n   -121.26\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"latitude\",\n   \"properties\":
{\n   \"dtype\": \"number\",\n   \"std\":
2.1359523974571153,\n   \"min\": 32.54,\n   \"max\": 41.95,\
n   \"num_unique_values\": 862,\n   \"samples\": [\n

33.7,\n   34.41,\n   38.24\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"housing_median_age\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
12.58555761211165,\n   \"min\": 1.0,\n   \"max\": 52.0,\n
\"num_unique_values\": 52,\n   \"samples\": [\n   35.0,\n
25.0,\n   7.0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"total_rooms\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 2181.615251582795,\n   \"min\":
2.0,\n   \"max\": 39320.0,\n   \"num_unique_values\": 5926,\
n   \"samples\": [\n   699.0,\n   1544.0,\n
3966.0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"total_bedrooms\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 421.3850700740323,\n   \"min\":
1.0,\n   \"max\": 6445.0,\n   \"num_unique_values\": 1923,\n
\"samples\": [\n   1538.0,\n   1298.0,\n
1578.0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"population\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 1132.462121765341,\n   \"min\":
3.0,\n   \"max\": 35682.0,\n   \"num_unique_values\": 3888,\
n   \"samples\": [\n   4169.0,\n   636.0,\n
3367.0\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"households\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 382.32975283161073,\n   \"min\":
1.0,\n   \"max\": 6082.0,\n   \"num_unique_values\": 1815,\n
\"samples\": [\n   21.0,\n   750.0,\n   1447.0\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"median_income\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
1.8998217179452688,\n   \"min\": 0.4999,\n   \"max\":
15.0001,\n   \"num_unique_values\": 12928,\n   \"samples\":
[\n   5.0286,\n   2.0433,\n   6.1228\n   ],\
n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"median_house_value\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
115395.61587441387,\n   \"min\": 14999.0,\n   \"max\":
500001.0,\n   \"num_unique_values\": 3842,\n   \"samples\":
[\n   194300.0,\n   379000.0,\n   230100.0\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"ocean_proximity\",\n
\"properties\": {\n   \"dtype\": \"category\",\n
\"num_unique_values\": 5,\n   \"samples\": [\n   \"<1H
OCEAN\",\n   \"ISLAND\",\n   \"INLAND\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe","variable_name":"housing_df"}

```
housing_df.tail()
```

{"summary":"{\n \"name\": \"housing_df\",\n \"rows\": 5,\n \"fields\": [\n   {\n   \"column\": \"longitude\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.08264381404557382,\n   \"min\": -121.32,\n   \"max\": -121.09,\n   \"num_unique_values\": 5,\n   \"samples\": [\n   -121.21,\n   -121.24,\n   -121.22\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"latitude\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.047958315233128025,\n   \"min\": 39.37,\n   \"max\": 39.49,\n   \"num_unique_values\": 4,\n   \"samples\": [\n   39.49,\n   39.37,\n   39.48\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"housing_median_age\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 3.5637059362410923,\n   \"min\": 16.0,\n   \"max\": 25.0,\n   \"num_unique_values\": 4,\n   \"samples\": [\n   18.0,\n   16.0,\n   25.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"total_rooms\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 774.7823565363373,\n   \"min\": 697.0,\n   \"max\": 2785.0,\n   \"num_unique_values\": 5,\n   \"samples\": [\n   697.0,\n   2785.0,\n   2254.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"total_bedrooms\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 170.95818202121828,\n   \"min\": 150.0,\n   \"max\": 616.0,\n   \"num_unique_values\": 5,\n   \"samples\": [\n   150.0,\n   616.0,\n   485.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"population\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 376.6566075352987,\n   \"min\": 356.0,\n   \"max\": 1387.0,\n   \"num_unique_values\": 5,\n   \"samples\": [\n   356.0,\n   1387.0,\n   1007.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"households\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 154.41729177783168,\n   \"min\": 114.0,\n   \"max\": 530.0,\n   \"num_unique_values\": 5,\n   \"samples\": [\n   114.0,\n   530.0,\n   433.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"median_income\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.43616087857578417,\n   \"min\": 1.5603,\n   \"max\": 2.5568,\n   \"num_unique_values\": 5,\n   \"samples\": [\n   2.5568,\n   2.3886,\n   1.7\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\

n   },\n   {\n   \"column\": \"median_house_value\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
6716.546731766258,\n   \"min\": 77100.0,\n   \"max\":
92300.0,\n   \"num_unique_values\": 5,\n   \"samples\": [\n
77100.0,\n   89400.0,\n   92300.0\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"ocean_proximity\",\n
\"properties\": {\n   \"dtype\": \"category\",\n
\"num_unique_values\": 1,\n   \"samples\": [\n
\"INLAND\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}


```
housing_df.describe()
```


{"summary":"{\n \"name\": \"housing_df\",\n \"rows\": 8,\n
\"fields\": [\n   {\n   \"column\": \"longitude\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
7333.554670164394,\n   \"min\": -124.35,\n   \"max\":
20640.0,\n   \"num_unique_values\": 8,\n   \"samples\": [\n
-119.56970445736432,\n   -118.49,\n   20640.0\
n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"latitude\",\n   \"properties\": {\n   \"dtype\":
\"number\",\n   \"std\": 7286.333552413666,\n   \"min\":
2.1359523974571153,\n   \"max\": 20640.0,\n
\"num_unique_values\": 8,\n   \"samples\": [\n
35.63186143410853,\n   34.26,\n   20640.0\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"housing_median_age\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
7288.35672120143,\n   \"min\": 1.0,\n   \"max\": 20640.0,\n
\"num_unique_values\": 8,\n   \"samples\": [\n
28.639486434108527,\n   29.0,\n   20640.0\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"total_rooms\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
13944.990983306392,\n   \"min\": 2.0,\n   \"max\": 39320.0,\
n   \"num_unique_values\": 8,\n   \"samples\": [\n
2635.7630813953488,\n   2127.0,\n   20640.0\n   ],\
n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"total_bedrooms\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
7106.427031043753,\n   \"min\": 1.0,\n   \"max\": 20433.0,\n
\"num_unique_values\": 8,\n   \"samples\": [\n
537.8705525375618,\n   435.0,\n   20433.0\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"population\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
13192.258841737372,\n   \"min\": 3.0,\n   \"max\": 35682.0,\
n   \"num_unique_values\": 8,\n   \"samples\": [\n

```
1425.4767441860465,\n          1166.0,\n          20640.0\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"households\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
7167.532601135343,\n        \"min\": 1.0,\n        \"max\": 20640.0,\n
\"num_unique_values\": 8,\n        \"samples\": [\n
499.5396802325581,\n          409.0,\n          20640.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"median_income\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
7295.7214358536385,\n        \"min\": 0.4999,\n        \"max\":
20640.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\n
3.8706710029069766,\n          3.5347999999999997,\n          20640.0\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"median_house_value\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 156160.28379826446,\n        \"min\":
14999.0,\n        \"max\": 500001.0,\n        \"num_unique_values\":
8,\n        \"samples\": [\n          206855.81690891474,\n
179700.0,\n          20640.0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe"}
```


```
housing_df.isnull().sum()
```

```
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
```

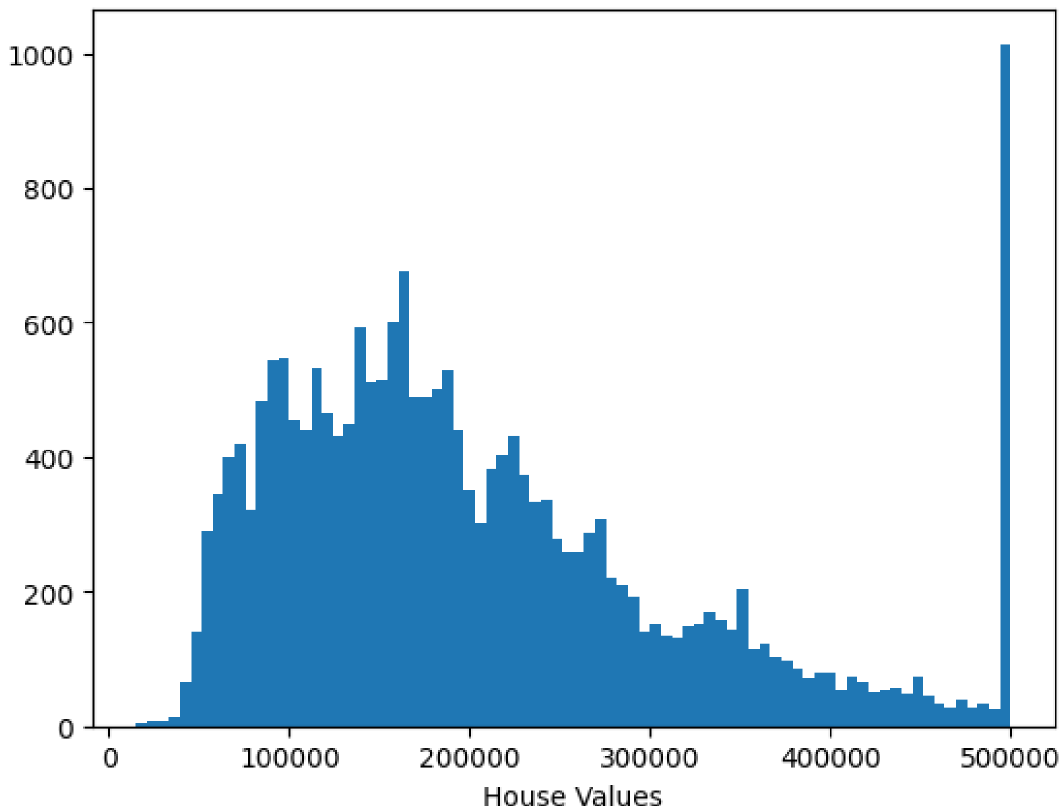
```
# Calculate the % of missing data
housing_df['total_bedrooms'].isnull().sum()/housing_df.shape[0] * 100
```

```
1.002906976744186
```

```
from sklearn.impute import KNNImputer

# create a temporary copy of the dataset
housing_df_temp = housing_df.copy()

# retrieve columns with numerical data; will exclude the
ocean_proximity column since the datatype is object; other columns are
float64
columns_list = [col for col in housing_df_temp.columns if
```

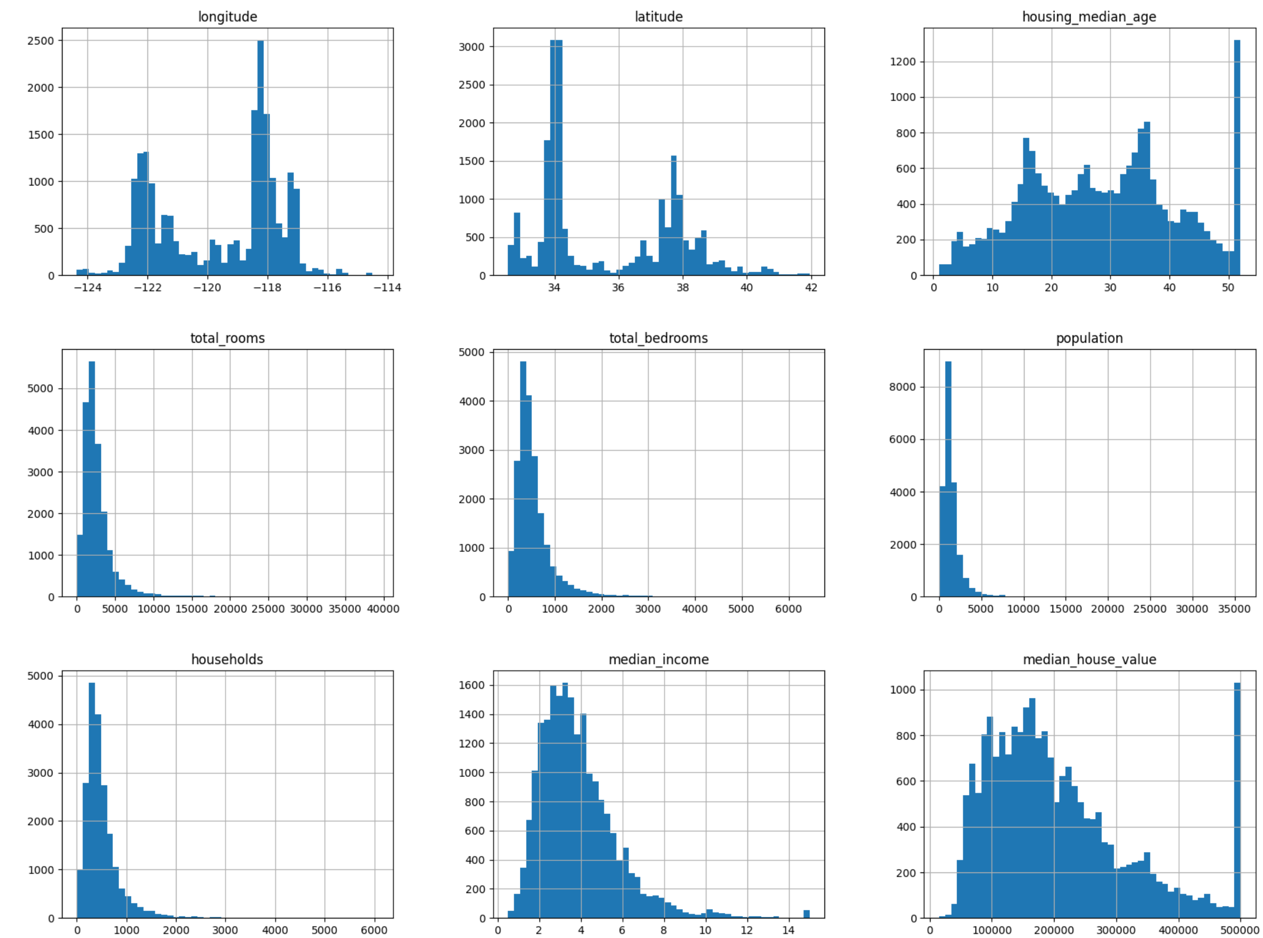
```
housing_df_temp[col].dtype != 'object']

# extract columns that contain at least one missing value
new_column_list = [col for col in housing_df_temp.loc[:,
housing_df_temp.isnull().any()]]

# update temp dataframe with numeric columns that have empty values
housing_df_temp = housing_df_temp[new_column_list]

# initialize KNNImputer to impute missing data using machine learning
knn = KNNImputer(n_neighbors = 3)

# fit function trains the model
knn.fit(housing_df_temp)

# transform the data using the model
# applies the transformation model (ie knn) to data
array_Values = knn.transform(housing_df_temp)

# convert the array values to a dataframe with the appropriate column
names
housing_df_temp = pd.DataFrame(array_Values, columns =
new_column_list)

# confirm there are no columns with missing values
housing_df_temp.isnull().sum()

total_bedrooms    0
dtype: int64

# overlay the imputed column over the old column with missing values

# loop through the list of columns and overlay each one
for column_name in new_column_list:
    housing_df[column_name] =
housing_df_temp.replace(housing_df[column_name],housing_df[column_name
])

# confirm columns no longer contain null data
housing_df.isnull().sum()

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

# Plot the distribution of the target variable (median_house_value)
using a histogram

# bins->amount of columns
plt.hist(housing_df['median_house_value'], bins=80)
plt.xlabel("House Values")


# We can see from the plot that the values of Median House Value are
distributed normally with few outliers.
# Most of the house are around 100,000-200,000 range

Text(0.5, 0, 'House Values')
```




```
# let's do histograms for the all the features to understand the data
distributions
# using housing_df as to not plot the encoded values for
OCEAN_PROXIMITY
housing_df.hist(bins=50, figsize=(20,15))
```

```
array([[<Axes: title={'center': 'longitude'}>,
        <Axes: title={'center': 'latitude'}>,
        <Axes: title={'center': 'housing_median_age'}>],
       [<Axes: title={'center': 'total_rooms'}>,
        <Axes: title={'center': 'total_bedrooms'}>,
        <Axes: title={'center': 'population'}>],
       [<Axes: title={'center': 'households'}>,
        <Axes: title={'center': 'median_income'}>,
        <Axes: title={'center': 'median_house_value'}>]],
dtype=object)
```

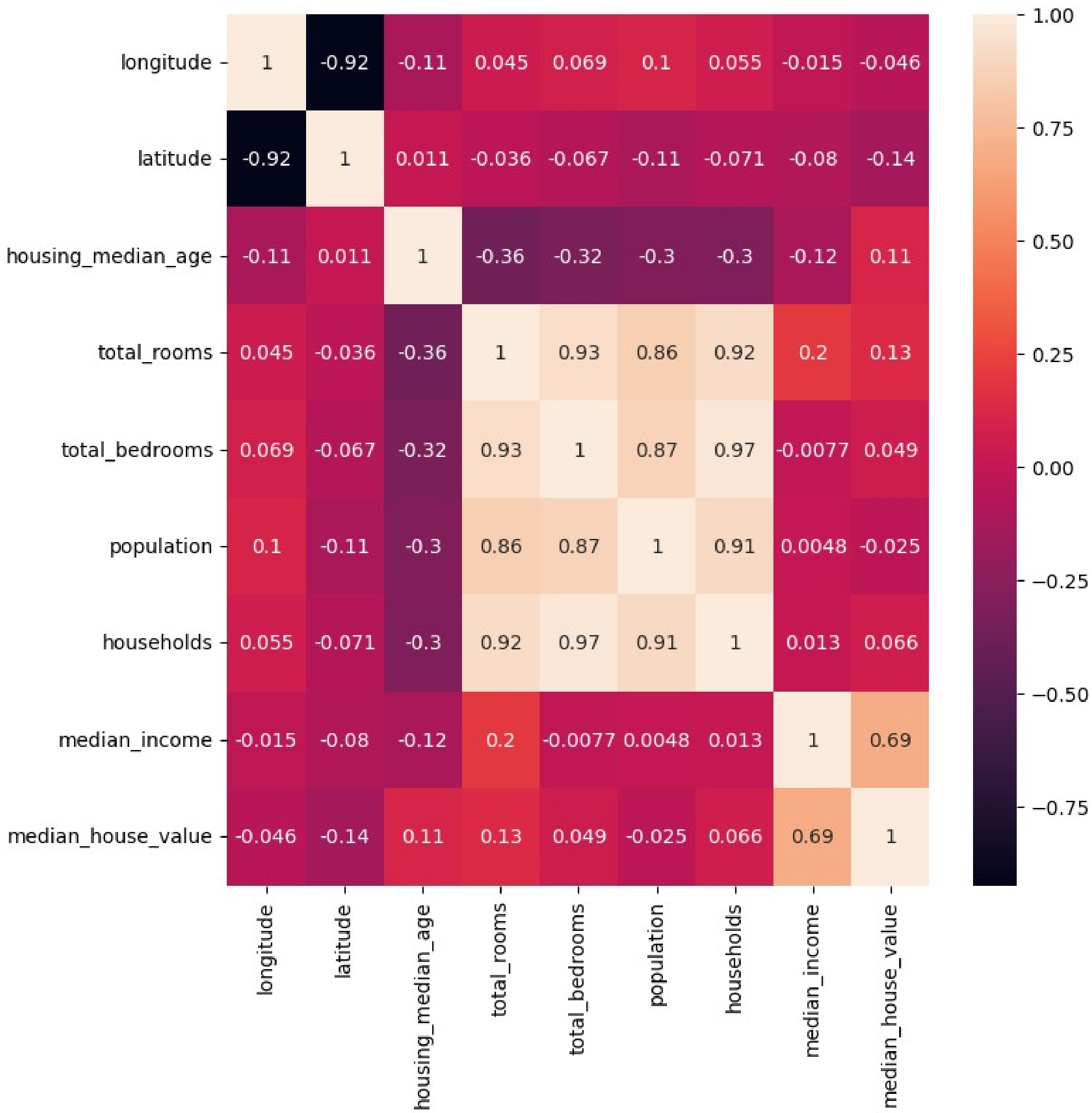



```python
# Plot a graphical correlation matrix for each pair of columns in the
dataframe
corr = housing_df.corr() # data frame correlation function
print(corr)
```

```
                    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

                    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

                    median_house_value
longitude                    -0.045967
latitude                     -0.144160
housing_median_age            0.105623
total_rooms                   0.134153
total_bedrooms                0.049454
population                   -0.024650
households                    0.065843
median_income                 0.688075
median_house_value            1.000000

<ipython-input-15-3abd71ce2464>:2: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it
```

```
will default to False. Select only valid columns or specify the value
of numeric_only to silence this warning.
  corr = housing_df.corr() # data frame correlation function

# make the heatmap larger in size
plt.figure(figsize = (8,8))

sns.heatmap(corr, annot=True)
plt.show()
```



```
# Additionally we noted that several features (total_rooms,total_bedrooms,population,households) have very high correlation to one another,
# so it's interesting to find out if a removal of a few of them would have any affect on the model performance

#  a new feature that is a ratio of the total rooms to households
housing_df['rooms_per_household'] = housing_df['total_rooms']/housing_df['households']

# a new feature that is a ratio of the total bedrooms to the total rooms
housing_df['bedrooms_per_room'] = housing_df['total_bedrooms']/housing_df['total_rooms']

# a new feature that is a ratio of the population to the households
housing_df['population_per_household']= housing_df['population']/housing_df['households']

# let's combine the latitude and longitude into 1
housing_df['coords'] = housing_df['longitude']/housing_df['latitude']

housing_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 14 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  object
 10  rooms_per_household       20640 non-null  float64
 11  bedrooms_per_room         20640 non-null  float64
 12  population_per_household  20640 non-null  float64
 13  coords                    20640 non-null  float64
dtypes: float64(13), object(1)
memory usage: 2.2+ MB
```

```
# remove total_rooms, households, total bedrooms, popluation, longitude, latitude
housing_df = housing_df.drop('total_rooms', axis=1)
housing_df = housing_df.drop('households', axis=1)
```

```
housing_df = housing_df.drop('total_bedrooms', axis=1)
housing_df = housing_df.drop('population', axis=1)
housing_df = housing_df.drop('longitude', axis=1)
housing_df = housing_df.drop('latitude', axis=1)

housing_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 8 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   housing_median_age        20640 non-null  float64
 1   median_income             20640 non-null  float64
 2   median_house_value        20640 non-null  float64
 3   ocean_proximity           20640 non-null  object
 4   rooms_per_household       20640 non-null  float64
 5   bedrooms_per_room         20640 non-null  float64
 6   population_per_household  20640 non-null  float64
 7   coords                    20640 non-null  float64
dtypes: float64(7), object(1)
memory usage: 1.3+ MB
```

```
#Heatmap after removing correlation

corr = housing_df.corr()

#make the heatmap larger in size
plt.figure(figsize = (7,7))

sns.heatmap(corr, annot=True)
plt.show()
```

```
<ipython-input-19-1264607259b1>:3: FutureWarning: The default value of
numeric_only in DataFrame.corr is deprecated. In a future version, it
will default to False. Select only valid columns or specify the value
of numeric_only to silence this warning.
  corr = housing_df.corr()
```


```
#Encoding categorical data
# Most ML algorithms can only learn from numeric data (it's all Math)
so categorical data must be encoded (i.e. converted) to numeric data

# Let's review our data types again; showing that ocean_proximity is
the only categorical data
housing_df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 8 columns):
 #   Column                    Non-Null Count  Dtype
```

```
---  ------                    --------------  -----
 0   housing_median_age        20640 non-null  float64
 1   median_income             20640 non-null  float64
 2   median_house_value        20640 non-null  float64
 3   ocean_proximity           20640 non-null  object
 4   rooms_per_household       20640 non-null  float64
 5   bedrooms_per_room         20640 non-null  float64
 6   population_per_household  20640 non-null  float64
 7   coords                    20640 non-null  float64
dtypes: float64(7), object(1)
memory usage: 1.3+ MB

# let's see the unique categories for OCEAN_PROXIMITY
housing_df.ocean_proximity.unique()

array(['NEAR BAY', '<1H OCEAN', 'INLAND', 'NEAR OCEAN', 'ISLAND'],
      dtype=object)

# let's count
housing_df["ocean_proximity"].value_counts()

<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: ocean_proximity, dtype: int64

# Let's see how the Panda's get_dummies() function works (generates
new columns based on the possible options)
print(pd.get_dummies(housing_df['ocean_proximity']))

       <1H OCEAN  INLAND  ISLAND  NEAR BAY  NEAR OCEAN
0              0       0       0         1           0
1              0       0       0         1           0
2              0       0       0         1           0
3              0       0       0         1           0
4              0       0       0         1           0
...          ...     ...     ...       ...         ...
20635          0       1       0         0           0
20636          0       1       0         0           0
20637          0       1       0         0           0
20638          0       1       0         0           0
20639          0       1       0         0           0

[20640 rows x 5 columns]

# let's replace the OCEAN_PROXIMITY column using get_dummies()
housing_df_encoded = pd.get_dummies(data=housing_df,
columns=['ocean_proximity'])
```

```python
# print the first few observations; notice the old OCEAN_PROXIMITY column is gone
housing_df_encoded.head()
```

{"summary":"{\n \"name\": \"housing_df_encoded\",\n \"rows\": 20640,\n \"fields\": [\n   {\n   \"column\": \"housing_median_age\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 12.58555761211165,\n   \"min\": 1.0,\n   \"max\": 52.0,\n   \"num_unique_values\": 52,\n   \"samples\": [\n   35.0,\n   25.0,\n   7.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"median_income\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 1.8998217179452688,\n   \"min\": 0.4999,\n   \"max\": 15.0001,\n   \"num_unique_values\": 12928,\n   \"samples\": [\n   5.0286,\n   2.0433,\n   6.1228\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"median_house_value\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 115395.61587441387,\n   \"min\": 14999.0,\n   \"max\": 500001.0,\n   \"num_unique_values\": 3842,\n   \"samples\": [\n   194300.0,\n   379000.0,\n   230100.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"rooms_per_household\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 2.4741731394243187,\n   \"min\": 0.8461538461538461,\n   \"max\": 141.9090909090909,\n   \"num_unique_values\": 19392,\n   \"samples\": [\n   6.111269614835948,\n   5.912820512820513,\n   5.7924528301886795\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"bedrooms_per_room\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.06976426445426125,\n   \"min\": 0.045936506323132786,\n   \"max\": 3.4926659255685832,\n   \"num_unique_values\": 19473,\n   \"samples\": [\n   0.19223760932944606,\n   0.2103494623655914,\n   0.20042417815482502\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"population_per_household\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 10.386049562213618,\n   \"min\": 0.6923076923076923,\n   \"max\": 1243.3333333333333,\n   \"num_unique_values\": 18841,\n   \"samples\": [\n   2.6939799331103678,\n   3.559375,\n   3.297082228116711\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"coords\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.1468967022257624,\n   \"min\": -3.597235023041475,\n   \"max\": -2.87341469251017,\n   \"num_unique_values\": 12575,\n   \"samples\": [\n   -3.1434118560704114,\n   -3.297632158590308,\n   -3.475270705297044\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\

```
n    },\n    {\n      \"column\": \"ocean_proximity_<1H OCEAN\",\n
\"properties\": {\n        \"dtype\": \"uint8\",\n
\"num_unique_values\": 2,\n        \"samples\": [\n          1,\n
0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ocean_proximity_INLAND\",\n      \"properties\": {\n
\"dtype\": \"uint8\",\n        \"num_unique_values\": 2,\n
\"samples\": [\n          1,\n          0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"ocean_proximity_ISLAND\",\n
\"properties\": {\n        \"dtype\": \"uint8\",\n
\"num_unique_values\": 2,\n        \"samples\": [\n          1,\n
0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"ocean_proximity_NEAR BAY\",\n      \"properties\": {\n
\"dtype\": \"uint8\",\n        \"num_unique_values\": 2,\n
\"samples\": [\n          0,\n          1\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"ocean_proximity_NEAR OCEAN\",\n
\"properties\": {\n        \"dtype\": \"uint8\",\n
\"num_unique_values\": 2,\n        \"samples\": [\n          1,\n
0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"housing_df_encoded"}
```


```
#Train the model
import sklearn
from sklearn.model_selection import train_test_split

# remove spaces from column names and convert all to lowercase and
remove special characters as it could cause issues in the future
housing_df_encoded.columns = [c.lower().replace(' ', '_').replace('<',
'_') for c in housing_df_encoded.columns]

# Split target variable and feature variables
X = housing_df_encoded[['housing_median_age',
'median_income','bedrooms_per_room','population_per_household','coords
','ocean_proximity__1h_ocean',

'ocean_proximity_inland','ocean_proximity_island','ocean_proximity_nea
r_bay','ocean_proximity_near_ocean']]
y = housing_df_encoded['median_house_value']

print(X)
```

```
       housing_median_age  median_income  bedrooms_per_room  \
0                    41.0         8.3252           0.146591
1                    21.0         8.3014           0.155797
2                    52.0         7.2574           0.129516
3                    52.0         5.6431           0.184458
```

```
4                    52.0        3.8462         0.172096
...                   ...           ...              ...
20635                25.0        1.5603         0.224625
20636                18.0        2.5568         0.215208
20637                17.0        1.7000         0.215173
20638                18.0        1.8672         0.219892
20639                16.0        2.3886         0.221185

       population_per_household    coords  
ocean_proximity__1h_ocean  \
0                      2.555556 -3.226769                          0   

1                      2.109842 -3.228209                          0   

2                      2.802260 -3.229590                          0   

3                      2.547945 -3.229855                          0   

4                      2.181467 -3.229855                          0   

...                         ...       ...                        ...   

20635                  2.560606 -3.067123                          0   

20636                  3.122807 -3.069385                          0   

20637                  2.325635 -3.074309                          0   

20638                  2.123209 -3.076845                          0   

20639                  2.616981 -3.079502                          0   


       ocean_proximity_inland  ocean_proximity_island  \
0                           0                       0   
1                           0                       0   
2                           0                       0   
3                           0                       0   
4                           0                       0   
...                       ...                     ...   
20635                       1                       0   
20636                       1                       0   
20637                       1                       0   
20638                       1                       0   
20639                       1                       0   

       ocean_proximity_near_bay  ocean_proximity_near_ocean
0                             1                           0
1                             1                           0
2                             1                           0
3                             1                           0
```

```
4                              1                            0
...                          ...                          ...
20635                          0                            0
20636                          0                            0
20637                          0                            0
20638                          0                            0
20639                          0                            0

[20640 rows x 10 columns]
```

```
# Split training & test data¶
# Splitting the data into training and testing sets in numpy arrays
# We train the model with 70% of the samples and test with the
remaining 30%
# X -> array with the inputs; y -> array of the outputs
X_train, X_test, y_train, y_test = train_test_split(X, y,
random_state=42, shuffle=True, test_size=0.3)

# Confirm how the data was split
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(14448, 10)
(6192, 10)
(14448,)
(6192,)
```

```
#Linear Regression - Model Training¶
# Use scikit-learn's LinearRegression to train the model on both the
training and evaluate it on the test sets
from sklearn.linear_model import LinearRegression

# Create a Linear regressor using all the feature variables
reg_model = LinearRegression()

# Train the model using the training sets
reg_model.fit(X_train, y_train)
```

```
LinearRegression()
```

```
#run the predictions on the training and testing data
y_pred_test = reg_model.predict(X_test)

#compare the actual values (ie, target) with the values predicted by
the model
pred_test_df = pd.DataFrame({'Actual': y_test, 'Predicted':
y_pred_test})

pred_test_df
```

{"summary":"{\n \"name\": \"pred_test_df\",\n \"rows\": 6192,\n \"fields\": [\n   {\n   \"column\": \"Actual\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 114575.39507173728,\n   \"min\": 14999.0,\n   \"max\": 500001.0,\n   \"num_unique_values\": 2689,\n   \"samples\": [\n   203300.0,\n   202200.0,\n   271100.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Predicted\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 92431.02486873654,\n   \"min\": -7939.178494427237,\n   \"max\": 1461440.4077402034,\n   \"num_unique_values\": 6192,\n   \"samples\": [\n   122657.7869513371,\n   72978.69492618677,\n   207207.3225332344\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"pred_test_df"}

```
# Determine accuracy uisng R^2
# R^2 : R squared is another way to evaluate the performance of a regression model.
# 1, means that the model is perfect and 0 means the the model will perform poorly.
r2_reg_model_test = round(reg_model.score(X_test, y_test),2)

print("R^2 Test: {}".format(r2_reg_model_test))
```

```
R^2 Test: 0.56
```

```
# try another machine learning algorithm : Randorm Forest
# Use scikit-learn’s Randorm Forest to train the model on both the training and evaluate it on the test sets
from sklearn.ensemble import RandomForestRegressor

# Create a  regressor using all the feature variables
rf_model = RandomForestRegressor(n_estimators=10,random_state=10)

# Train the model using the training sets
rf_model.fit(X_train, y_train)
```

```
RandomForestRegressor(n_estimators=10, random_state=10)
```

```
#run the predictions on the training and testing data
y_rf_pred_test = rf_model.predict(X_test)

#compare the actual values (ie, target) with the values predicted by the model
rf_pred_test_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_rf_pred_test})

rf_pred_test_df
```

{"summary":"{\n \"name\": \"rf_pred_test_df\",\n \"rows\": 6192,\n \"fields\": [\n   {\n   \"column\": \"Actual\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 114575.39507173728,\n   \"min\": 14999.0,\n   \"max\": 500001.0,\n   \"num_unique_values\": 2689,\n   \"samples\": [\n   203300.0,\n   202200.0,\n   271100.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Predicted\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 102994.30885837866,\n   \"min\": 46270.0,\n   \"max\": 500001.0,\n   \"num_unique_values\": 5615,\n   \"samples\": [\n   58930.0,\n   390480.5,\n   85400.0\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"rf_pred_test_df"}

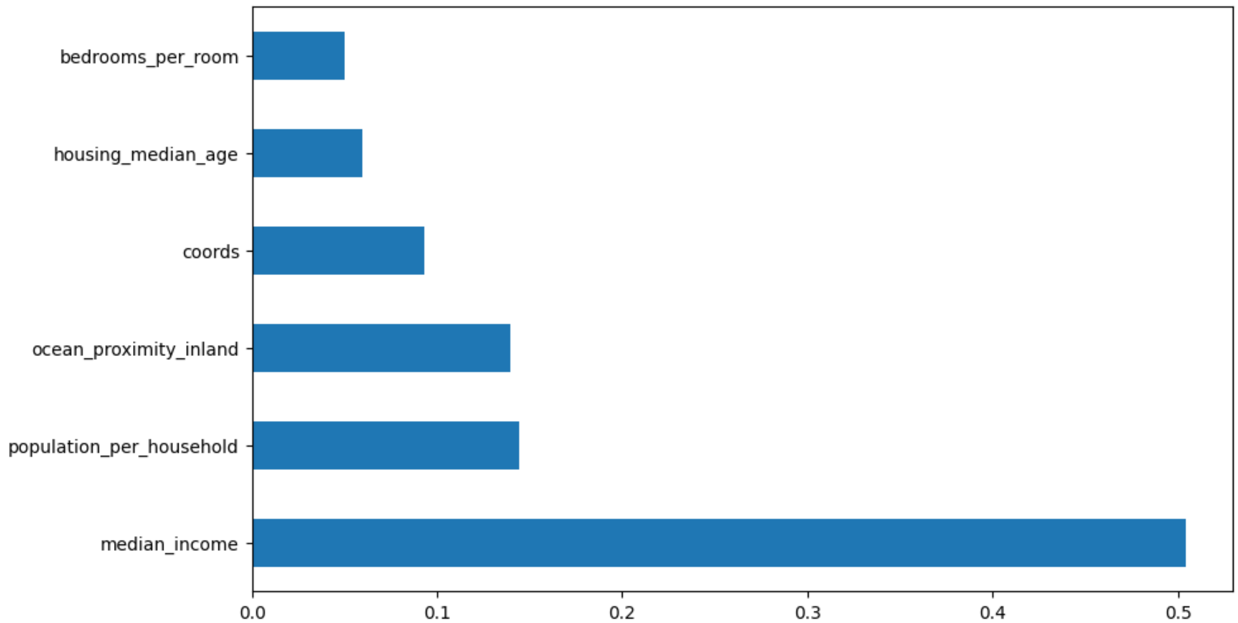
```
# Determine accuracy uisng R^2
from sklearn.metrics import r2_score, mean_squared_error

score = r2_score(y_test, y_rf_pred_test)

print("R^2 - {}%".format(round(score, 2) *100))
```

```
R^2 - 75.0%
```

```
# Determine RMSE - Root Mean Squared Error on the test data
print('RMSE on test data: ',  mean_squared_error(y_test,
y_rf_pred_test)**(0.5))
```

```
RMSE on test data:  57289.11495447338
```

```
# Determine feature importance - random forest algorithm is that it
gives you the ‘feature importance’ for all the variables in the data
# plot the 6 most important features
plt.figure(figsize=(10,6))
feat_importances = pd.Series(rf_model.feature_importances_, index =
X_train.columns)
feat_importances.nlargest(6).plot(kind='barh');
```




```
# training data with 5 most important features
train_x_if = X_train[['bedrooms_per_room', 'housing_median_age',
'coords',
'ocean_proximity_inland','population_per_household','median_income']]
test_x_if = X_test[['bedrooms_per_room', 'housing_median_age',
'coords',
'ocean_proximity_inland','population_per_household','median_income']]

# create an object of the RandfomForestRegressor Model
rf_model_if = RandomForestRegressor(n_estimators=10,random_state=10)

# fit the model with the training data
rf_model_if.fit(train_x_if, y_train)

# predict the target on the test data
predict_test_with_if = rf_model_if.predict(test_x_if)

# Root Mean Squared Error on the train and test data
print('RMSE on test data: ',  mean_squared_error(y_test,
predict_test_with_if)**(0.5))

RMSE on test data:  57366.910692045196

pip install xgboost

Requirement already satisfied: xgboost in
/usr/local/lib/python3.10/dist-packages (2.0.3)
Requirement already satisfied: numpy in
/usr/local/lib/python3.10/dist-packages (from xgboost) (1.25.2)
Requirement already satisfied: scipy in
/usr/local/lib/python3.10/dist-packages (from xgboost) (1.11.4)
```

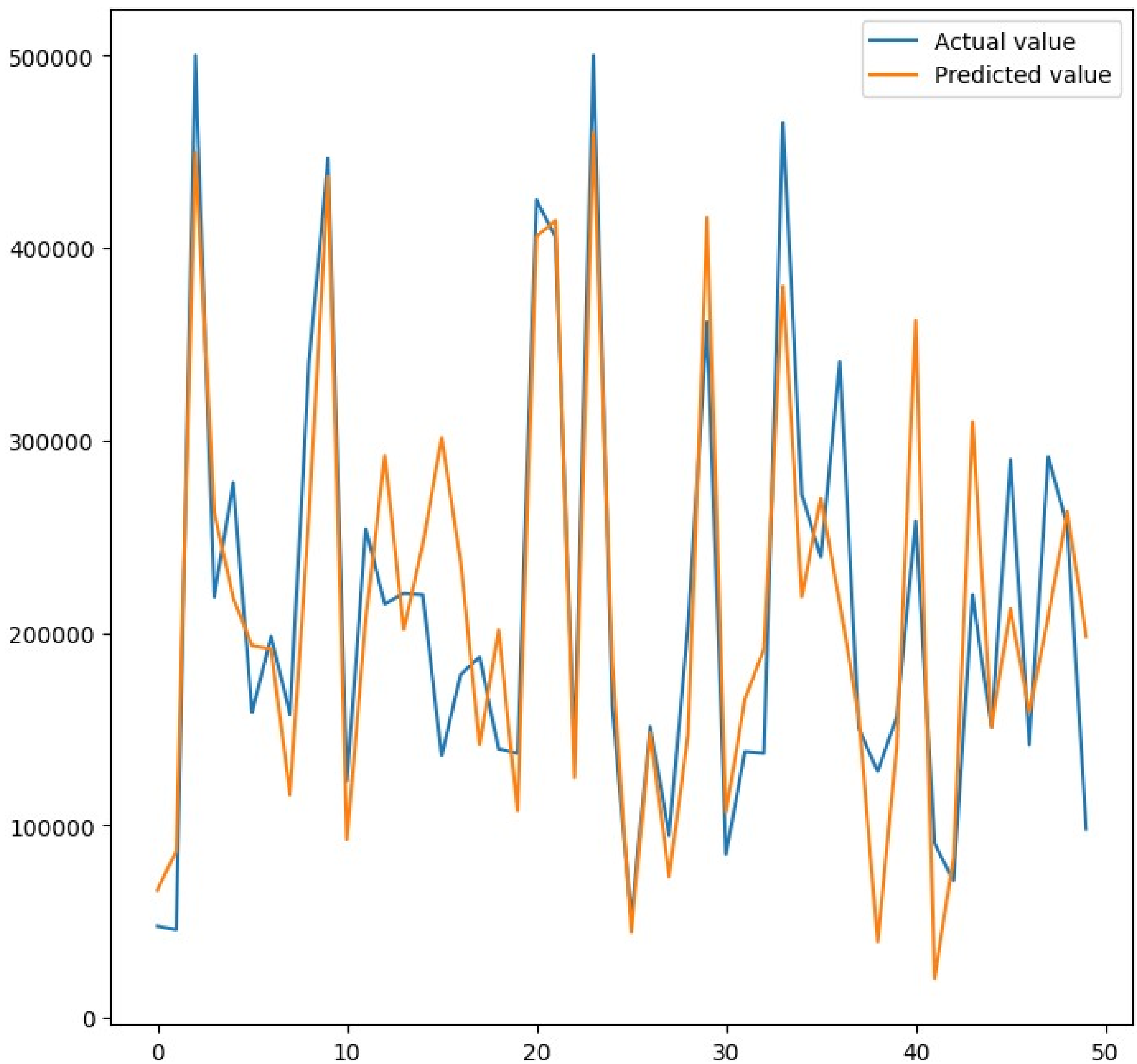
```
# Extreme Gradient Boosting (XGBoost) is an open-source library that
provides an efficient and effective implementation of the gradient
boosting algorithm.
# Use the scikit-learn wrapper classes: XGBRegressor and
XGBClassifier.

# try another machine learning algorithm : XGBoost
from xgboost import XGBRegressor

xgb_model = XGBRegressor()

# Train the model using the training sets
xgb_model.fit(X_train, y_train)

XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None,
early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None,
feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None,
max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan,
monotone_constraints=None,
             multi_strategy=None, n_estimators=None, n_jobs=None,
             num_parallel_tree=None, random_state=None, ...)

#run the predictions on the training and testing data
y_xgb_pred_test = xgb_model.predict(X_test)

#compare the actual values (ie, target) with the values predicted by
the model
xgb_pred_test_df = pd.DataFrame({'Actual': y_test, 'Predicted':
y_xgb_pred_test})

xgb_pred_test_df
```

{"summary":"{\n \"name\": \"xgb_pred_test_df\",\n \"rows\": 6192,\n
\"fields\": [\n   {\n   \"column\": \"Actual\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
114575.39507173728,\n   \"min\": 14999.0,\n   \"max\":
500001.0,\n   \"num_unique_values\": 2689,\n   \"samples\":
[\n   203300.0,\n   202200.0,\n   271100.0\n
],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   },\n   {\n   \"column\": \"Predicted\",\n
\"properties\": {\n   \"dtype\": \"float32\",\n
\"num_unique_values\": 6189,\n   \"samples\": [\n
107010.84375,\n   119321.6640625,\n   67871.625\n

```
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\
n}","type":"dataframe","variable_name":"xgb_pred_test_df"}
```


```
fig= plt.figure(figsize=(8,8))
xgb_pred_test_df = xgb_pred_test_df.reset_index()
xgb_pred_test_df = xgb_pred_test_df.drop(['index'],axis=1)
plt.plot(xgb_pred_test_df[:50])
plt.legend(['Actual value','Predicted value'])
```

```
<matplotlib.legend.Legend at 0x78b3029be8c0>
```




```
from sklearn.metrics import r2_score
```

```
score = r2_score(y_test, y_xgb_pred_test)

print("R^2 - {}%".format(round(score, 2) *100))
```

```
R^2 - 78.0%
```

```
# Determine mean square error and root mean square error
from sklearn.metrics import mean_squared_error
import math

mse = mean_squared_error(y_test, y_xgb_pred_test)
rmse = math.sqrt(mean_squared_error(y_test, y_xgb_pred_test))

print(mse)
print(rmse)
```

```
2939759040.9080276
54219.5448238735
```

```
# Calculate mean absolute error(any large error)
from sklearn.metrics import mean_absolute_error

print(mean_absolute_error(y_test, y_xgb_pred_test))
```

```
36285.050324826894
```

```
# We can build and score a model on multiple folds using cross-
validation
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import cross_val_score


# define model evaluation method
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)

scores = cross_val_score(xgb_model, X, y, scoring='r2',
error_score='raise', cv=cv, n_jobs=-1, verbose=1)

#average of all the r2 scores across runs
print(scores.mean())
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent
workers.
```

```
0.7850403811484551
```

```
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:    7.0s finished
```

```
# determine hyperparameter available for tuning
xgb_model.get_params()
```

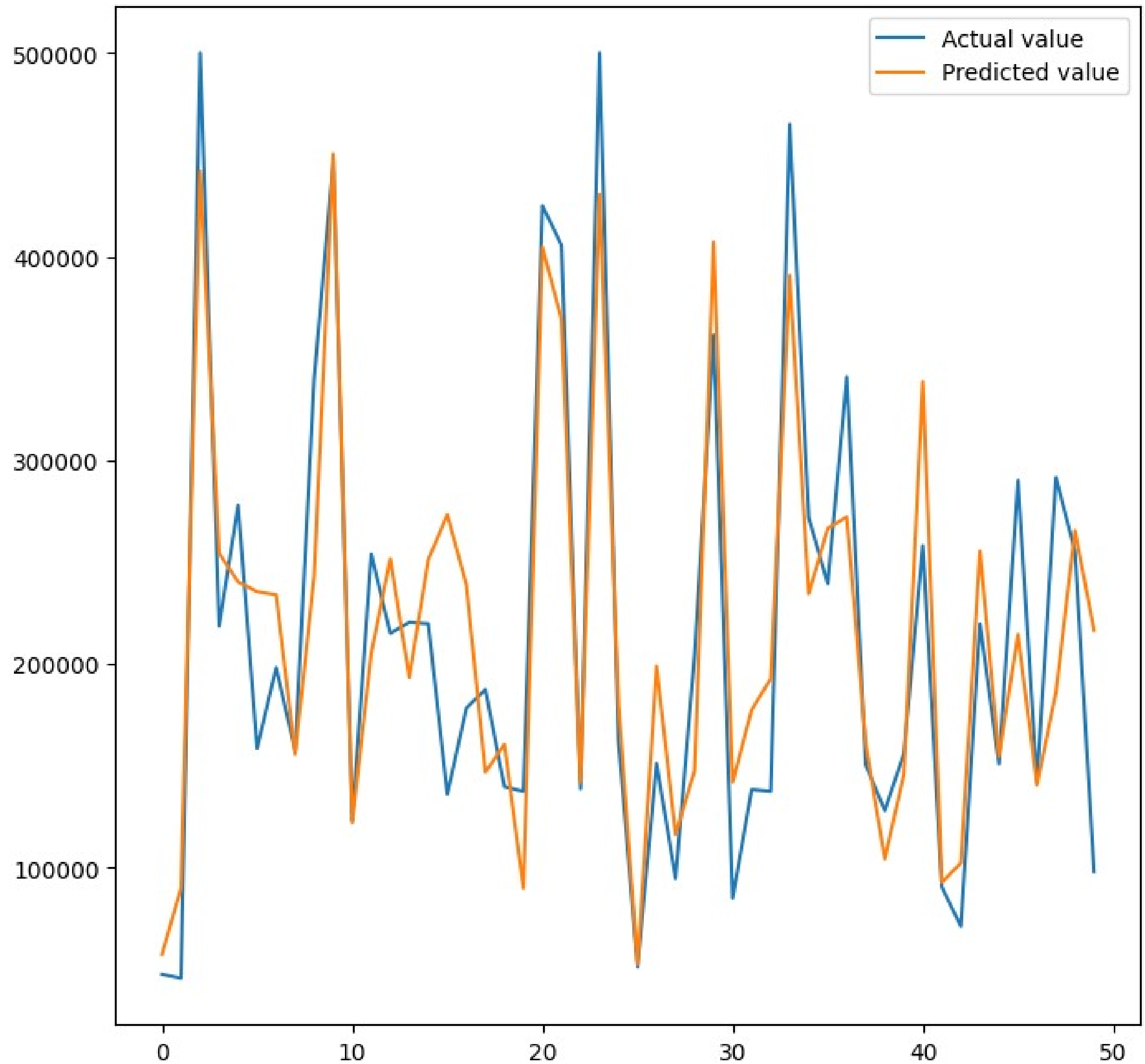
```
{'objective': 'reg:squarederror',
 'base_score': None,
 'booster': None,
 'callbacks': None,
 'colsample_bylevel': None,
 'colsample_bynode': None,
 'colsample_bytree': None,
 'device': None,
 'early_stopping_rounds': None,
 'enable_categorical': False,
 'eval_metric': None,
 'feature_types': None,
 'gamma': None,
 'grow_policy': None,
 'importance_type': None,
 'interaction_constraints': None,
 'learning_rate': None,
 'max_bin': None,
 'max_cat_threshold': None,
 'max_cat_to_onehot': None,
 'max_delta_step': None,
 'max_depth': None,
 'max_leaves': None,
 'min_child_weight': None,
 'missing': nan,
 'monotone_constraints': None,
 'multi_strategy': None,
 'n_estimators': None,
 'n_jobs': None,
 'num_parallel_tree': None,
 'random_state': None,
 'reg_alpha': None,
 'reg_lambda': None,
 'sampling_method': None,
 'scale_pos_weight': None,
 'subsample': None,
 'tree_method': None,
 'validate_parameters': None,
 'verbosity': None}
```

```
xgb_model_2 = XGBRegressor(
    gamma=0.05,
    learning_rate=0.01,
    max_depth=6,
    n_estimators=1000,
    n_jobs=16,
    objective='reg:squarederror',
    subsample=0.8,
    scale_pos_weight=0,
    reg_alpha=0,
```

```
    reg_lambda=1,
    verbosity=1)

xgb_model_2.fit(X_train, y_train)


#run the predictions on the training and testing data
y_xgb_2_pred_test = xgb_model_2.predict(X_test)

# compare the actual values (ie, target) with the values predicted by
the model
xgb_2_pred_test_df = pd.DataFrame({'Actual': y_test, 'Predicted':
y_xgb_2_pred_test})

xgb_2_pred_test_df
```


```
{"summary":"{\n  \"name\": \"xgb_2_pred_test_df\",\n  \"rows\": 6192,\
n  \"fields\": [\n    {\n      \"column\": \"Actual\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
114575.39507173728,\n        \"min\": 14999.0,\n        \"max\":
500001.0,\n        \"num_unique_values\": 2689,\n        \"samples\":
[\n          203300.0,\n          202200.0,\n          271100.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Predicted\",\n
\"properties\": {\n        \"dtype\": \"float32\",\n
\"num_unique_values\": 6190,\n        \"samples\": [\n
99311.6015625,\n          103087.6640625,\n          345059.84375\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\
n}","type":"dataframe","variable_name":"xgb_2_pred_test_df"}
```


```
fig= plt.figure(figsize=(8,8))
xgb_2_pred_test_df = xgb_2_pred_test_df.reset_index()
xgb_2_pred_test_df = xgb_2_pred_test_df.drop(['index'],axis=1)
plt.plot(xgb_2_pred_test_df[:50])
plt.legend(['Actual value','Predicted value'])

<matplotlib.legend.Legend at 0x78b3029bf0d0>
```




```
from sklearn.metrics import mean_squared_error

mse = np.sqrt(mean_squared_error(y_test, y_xgb_2_pred_test))
print("RMSE: %.2f" % (mse**(1/2.0)))

RMSE: 230.63

# Determine accuracy uisng R^2
r2_xgb_model_2_test = round(xgb_model_2.score(X_test, y_test),2)

print("R^2 Test: {}".format(r2_xgb_model_2_test))

R^2 Test: 0.78
```