housing

December 27, 2024

[2]:
```
import pandas as pd
import numpy as np
```

0.0.1 Import the data and have a glance of it

[3]:
```
housing = pd.read_csv("housing.csv")
```

[4]:
```
housing.head()
```

[4]:
```
   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

   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
```

0.0.2 See some information and description of the data

[5]:
```
housing.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
```

[6]:
```
housing.describe()
```

[6]:
```
          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
```

0.0.3 For data in formats other than numbers, like some falling on particular categories, have a look at the value count to know frequency of each category

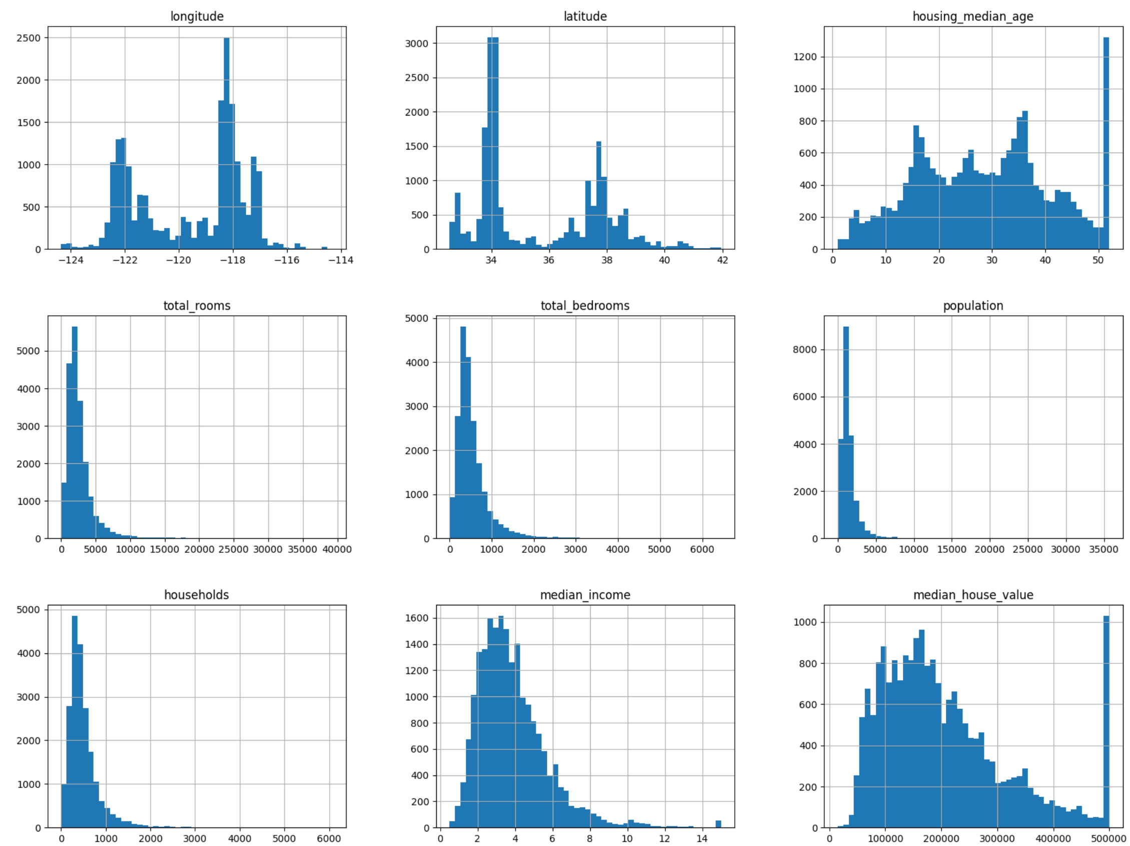
[7]:
```
housing["ocean_proximity"].value_counts()
```

[7]:
```
ocean_proximity
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
```

```
NEAR BAY      2290
ISLAND           5
Name: count, dtype: int64
```

0.0.4 Have a histogram plot for every numeric data to have a look at the frequency of different intervals

[8]:
```
%matplotlib inline
import matplotlib.pyplot as plt
housing.hist(bins=50, figsize=(20,15))
plt.show()
```



0.0.5 Splitting the data into test set and training set

Option 1: You can split the data into 2 parts randomly

- Every time you will run the notebook the test set and the train set will change.
- You can use np.random.seed({SOME NUMBER}) to make sure everytime you run the notebook same indices are selected for test and train set
- Stratified sampling can not be satisfied with this option

```
[9]: def split_test_set(data, ratio):
         shuffled_indices = np.random.permutation(len(data))
         test_indices = shuffled_indices[ : int(len(data) * ratio)]
         train_indices = shuffled_indices[int(len(data) * ratio) : ]
         test_set = data.iloc[test_indices]
         train_set = data.iloc[train_indices]
         return test_set, train_set
     test_set, train_set = split_test_set(housing, 0.2)
     print(f"Train set length: {len(train_set)}\nTest set length: {len(test_set)}")
```

```
Train set length: 16512
Test set length: 4128
```

Option 2: You can make use of the sklearn in built train_test_split function

- This is pretty similar to option 1
- If you specify the random state number then it will select same data in both the sets everytime you run the notebook
- Stratified sampling cannot be done

```
[10]: from sklearn.model_selection import train_test_split
      training_set, testing_set = train_test_split(housing, test_size = 0.2,␣
       ↪random_state = 42)
      print(f"Train Set length: {len(training_set)}\nTest Set length:␣
       ↪{len(testing_set)}")
```

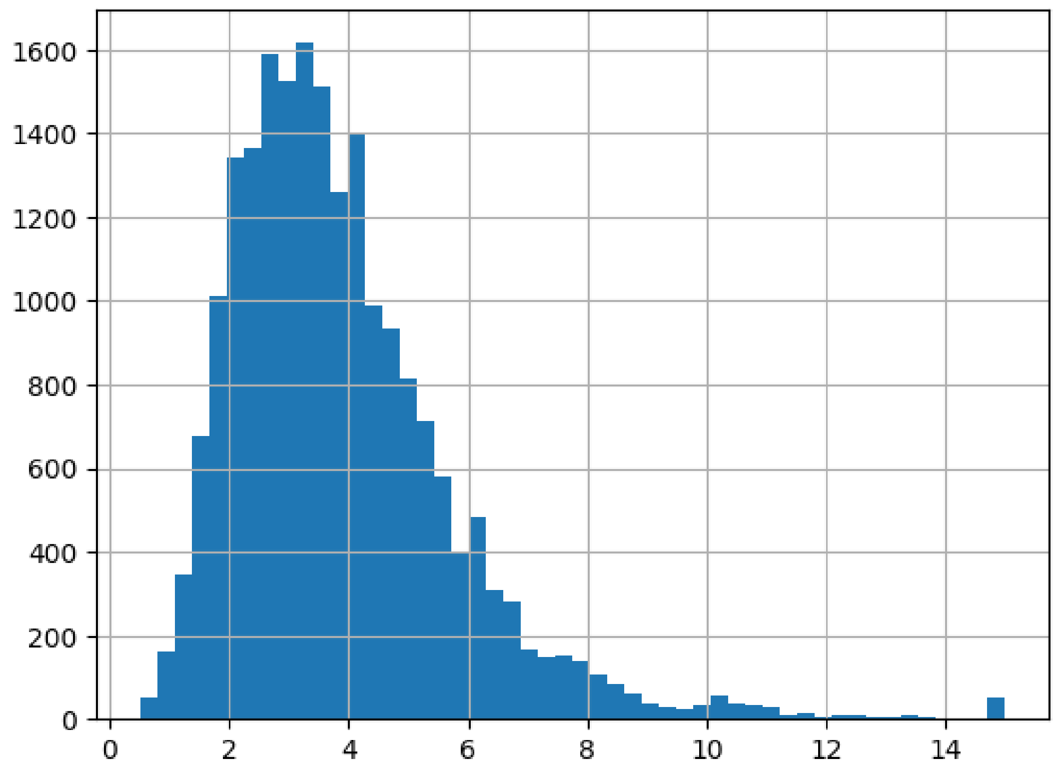
```
Train Set length: 16512
Test Set length: 4128
```

Option 3: Stratified Sampling (The best option)

- Identify the most important attribute and classify the data in some parts based on that feature
- See the frequency of data in each classification (newly created classification)
- Split the data based on that classification
  - For example:
    - If a data has 4 classes:
      - Class 1: 15%
      - Class 2: 10%
      - Class 3: 70%
      - Class 4: 5%
  - Then the test set and the train set must also have the same percentage of each class

```
[11]: housing["median_income"].hist(bins=50)
```

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




```
[12]: housing["median_income"].max()
```

[12]: 15.0001

Identify the intervals to break in and specify them in the bins section and give a name to each class in the labels section

```
[13]: housing["income_category"] = pd.cut(housing["median_income"],
                                          bins = [0, 1.5, 3, 4.5, 6, np.inf],
                                          labels = [1, 2, 3, 4, 5],
                                          right = False)
      housing[["median_income", "income_category"]].sample(10)
```

[13]:

	median_income	income_category
810	3.8125	3
11081	3.5900	3
8344	4.0625	3
5640	4.6250	4
15232	10.1560	5
11965	4.2321	3
2340	4.9432	4
2733	1.3373	1
12830	2.5000	2

```
15801          4.6053                 4
```

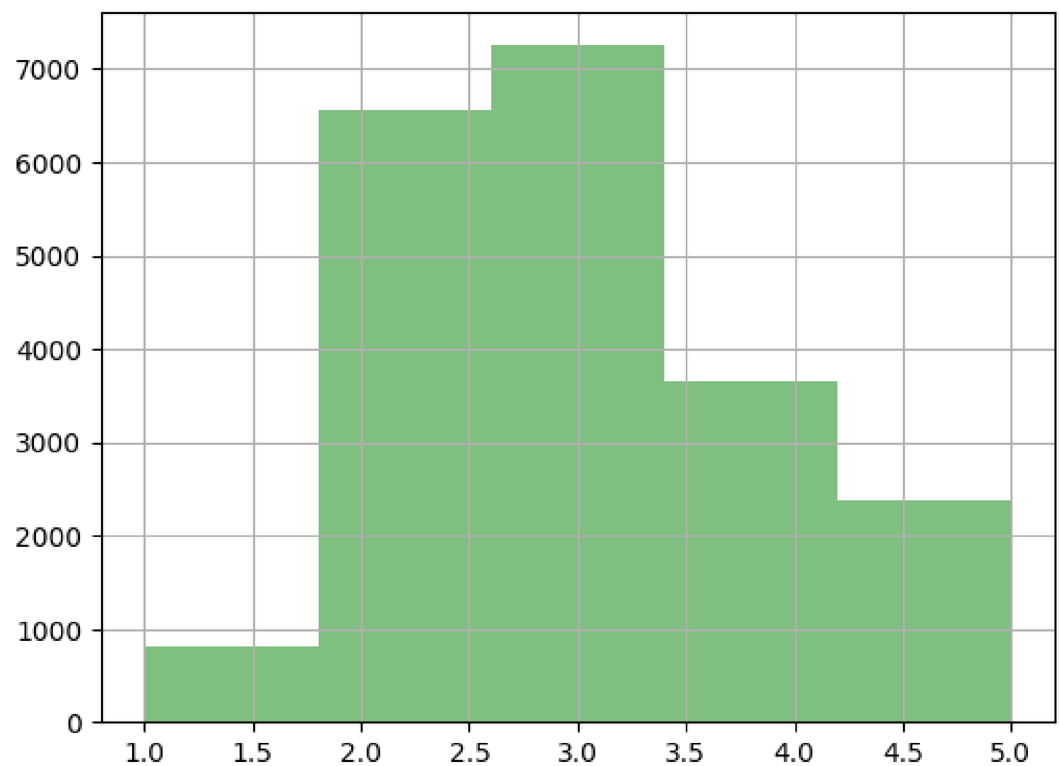
Check the frequency of each newly formed class

```
[14]: housing["income_category"].value_counts()
```

```
[14]: income_category
3    7250
2    6550
4    3652
5    2373
1     815
Name: count, dtype: int64
```

Visualize the same with the help of a histogram

```
[15]: housing["income_category"].hist(color = 'g', alpha = 0.5, bins = 5)
plt.show()
full_dataset = housing["income_category"].value_counts() / len(housing)
```



With the help of StratifiedShuffleSplit class in Sklearn split the data into test set and train set

[240]:
```python
from sklearn.model_selection import StratifiedShuffleSplit
split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in split.split(housing, housing["income_category"]):
    strat_train_set = housing.loc[train_index]
    strat_test_set = housing.loc[test_index]
```

Verify that each of the 3 data sets viz. full data set, train set and test set contains data in same proportion in accordance to the classes created by you

[17]:
```python
s_train_set = strat_train_set["income_category"].value_counts() /␣
 ↪len(strat_train_set)
```

[18]:
```python
s_test_set = strat_test_set["income_category"].value_counts() /␣
 ↪len(strat_test_set)
```

[19]:
```python
stratified_ratio_df = pd.DataFrame([full_dataset, s_train_set, s_test_set],
                                   index = ["Full Set", "Train Set", "Test␣
 ↪Set"],)
stratified_ratio_df
```

[19]:

income_category	3	2	4	5	1
Full Set	0.35126	0.317345	0.176938	0.114971	0.039486
Train Set	0.35126	0.317345	0.176962	0.114947	0.039486
Test Set	0.35126	0.317345	0.176841	0.115068	0.039486

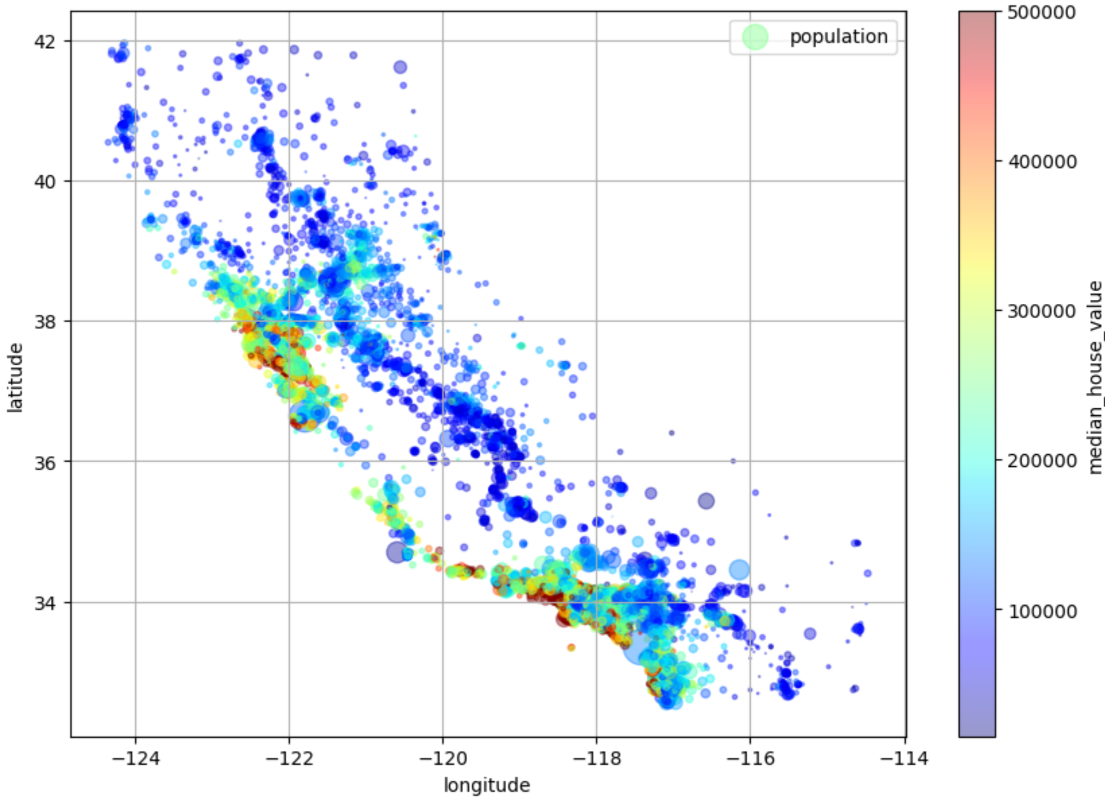
The class created by you was for our convinience so delete it once splitting into test set and train set is complete

[20]:
```python
for drop_set in (strat_train_set, strat_test_set):
    drop_set.drop("income_category", axis = 1, inplace = True)
```

0.0.6 Now keep the test set aside and start working with the train set. Make a copy of the train set to make sure any changes you perform in the set won't affect your actual dataset.

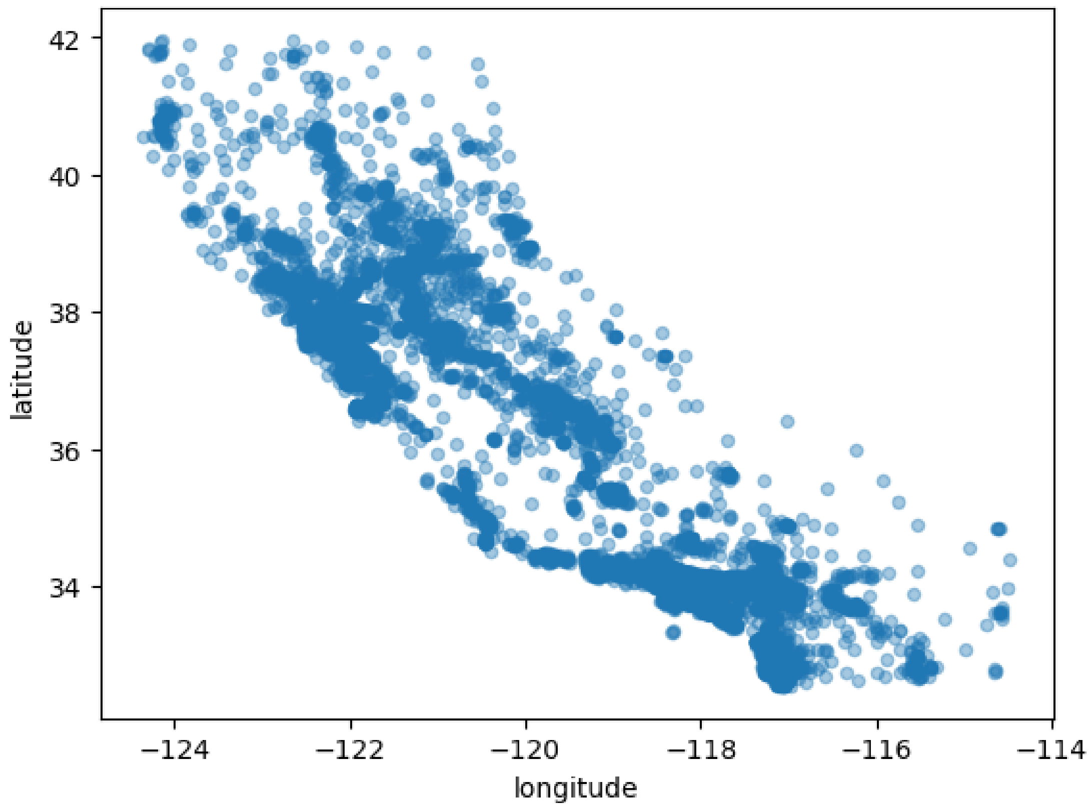
0.0.7 Explore the train set to derrive some base level conclusions

[21]:
```python
data = strat_train_set.copy()
data["population"]/=100
data.plot(kind = "scatter", x = "longitude", y = "latitude", alpha = 0.4,
             s = "population", label = "population",
             c = "median_house_value", cmap=plt.get_cmap("jet"),␣
 ↪figsize=(10,7), colorbar = True)
plt.legend()
plt.grid()
plt.show()
```

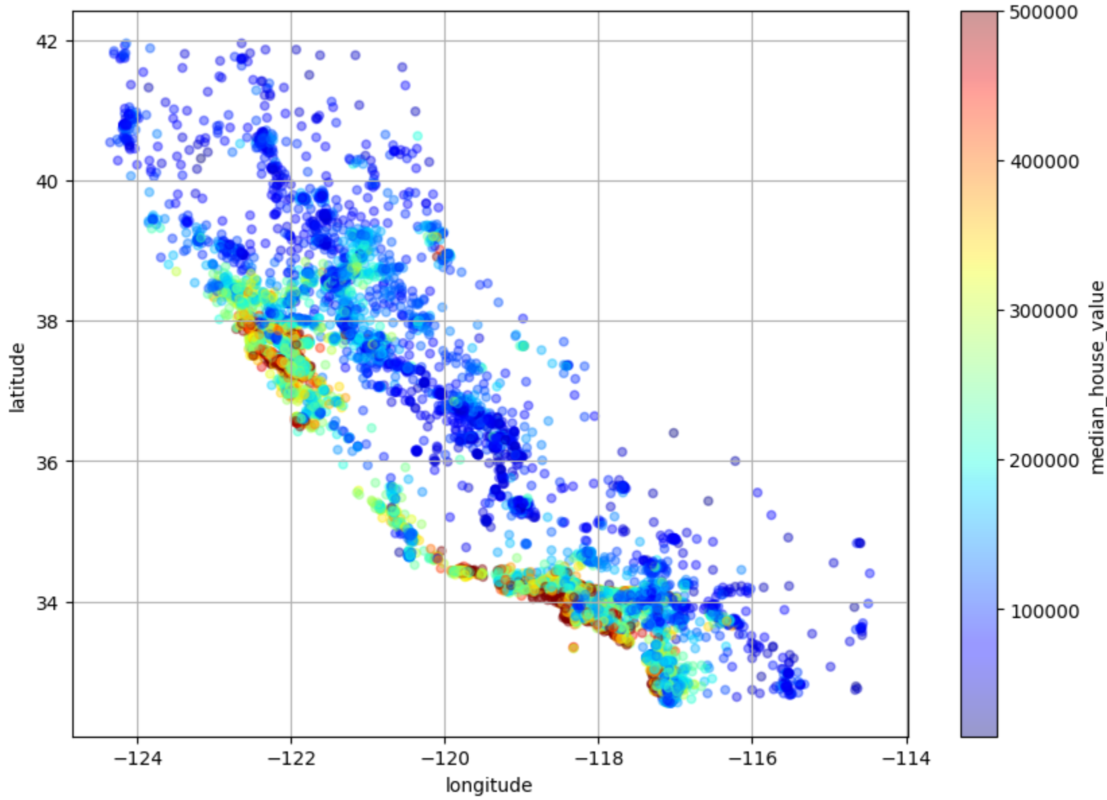


[22]:
```
data.plot(kind = "scatter", x = "longitude", y = "latitude", alpha = 0.4)
```

[22]: <Axes: xlabel='longitude', ylabel='latitude'>

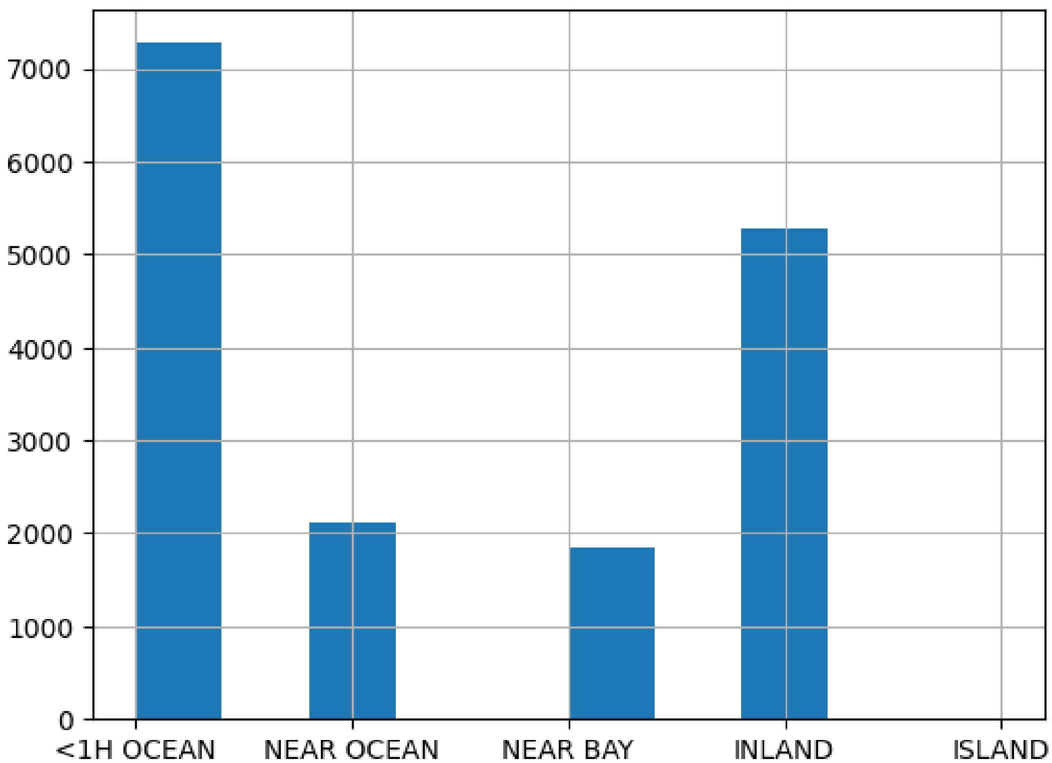


[23]:
```
data.plot(kind = "scatter", x = "longitude", y = "latitude", alpha = 0.4,
              c = "median_house_value", cmap=plt.get_cmap("jet"),␣
 ↪figsize=(10,7), colorbar = True)
plt.grid()
plt.show()
```



0.0.8 For most of the ML algorithms, it is difficult to work with string data so for the time being remove the non-numeric data

[24]:
```
data["ocean_proximity"].hist()
plt.show()
```



[25]:
```
data["ocean_proximity"].value_counts()
```

[25]:
```
ocean_proximity
<1H OCEAN     7274
INLAND        5275
NEAR OCEAN    2110
NEAR BAY      1850
ISLAND           3
Name: count, dtype: int64
```

[26]:
```
data_without_ocean = data.drop(["ocean_proximity"], axis = 1)
data_without_ocean
```

[26]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	\
17262	-119.71	34.42	52.0	1411.0	324.0	
16799	-122.44	37.67	35.0	1814.0	365.0	
16718	-120.66	35.49	17.0	4422.0	945.0	
5373	-118.38	34.04	36.0	3005.0	771.0	
20311	-119.11	34.17	37.0	470.0	105.0	
…	…	…	…	…	…	
13893	-116.31	34.13	20.0	2352.0	556.0	
20015	-119.08	36.09	25.0	1880.0	339.0	

```
18879    -122.26    38.10                 24.0       1213.0            395.0
19612    -121.11    37.47                 12.0       2263.0            410.0
4624     -118.32    34.06                 52.0        983.0            246.0

       population  households  median_income  median_house_value
17262       10.91       306.0         4.1062            252900.0
16799       10.25       384.0         4.4250            268400.0
16718       23.07       885.0         2.8285            171300.0
5373        20.54       758.0         2.0437            309100.0
20311        5.22        83.0         2.0368            243800.0
…             …          …              …                  …
13893       12.17       481.0         1.6063             55400.0
20015       10.03       315.0         2.7298            103400.0
18879        6.99       386.0         1.3007             94600.0
19612        9.13       330.0         3.5795            145600.0
4624         5.78       204.0         5.7393            500001.0

[16512 rows x 9 columns]
```

0.0.9 Create the correlation matrix

[27]:
```python
corr_matrix = data_without_ocean.corr()
corr_matrix
```

[27]:
```
                    longitude  latitude  housing_median_age  total_rooms  \
longitude            1.000000 -0.924406           -0.110271     0.046208
latitude            -0.924406  1.000000            0.013167    -0.036300
housing_median_age  -0.110271  0.013167            1.000000    -0.361847
total_rooms          0.046208 -0.036300           -0.361847     1.000000
total_bedrooms       0.070559 -0.067435           -0.319500     0.930540
population           0.101778 -0.108950           -0.294295     0.852908
households           0.058129 -0.072744           -0.302308     0.919858
median_income       -0.017381 -0.076203           -0.115598     0.194623
median_house_value  -0.045456 -0.144337            0.108851     0.131329

                    total_bedrooms  population  households  median_income  \
longitude                 0.070559    0.101778    0.058129      -0.017381
latitude                 -0.067435   -0.108950   -0.072744      -0.076203
housing_median_age       -0.319500   -0.294295   -0.302308      -0.115598
total_rooms               0.930540    0.852908    0.919858       0.194623
total_bedrooms            1.000000    0.874161    0.981022      -0.012690
population                0.874161    1.000000    0.902343      -0.001087
households                0.981022    0.902343    1.000000       0.008239
median_income            -0.012690   -0.001087    0.008239       1.000000
median_house_value        0.047040   -0.030342    0.062166       0.687446

                    median_house_value
```

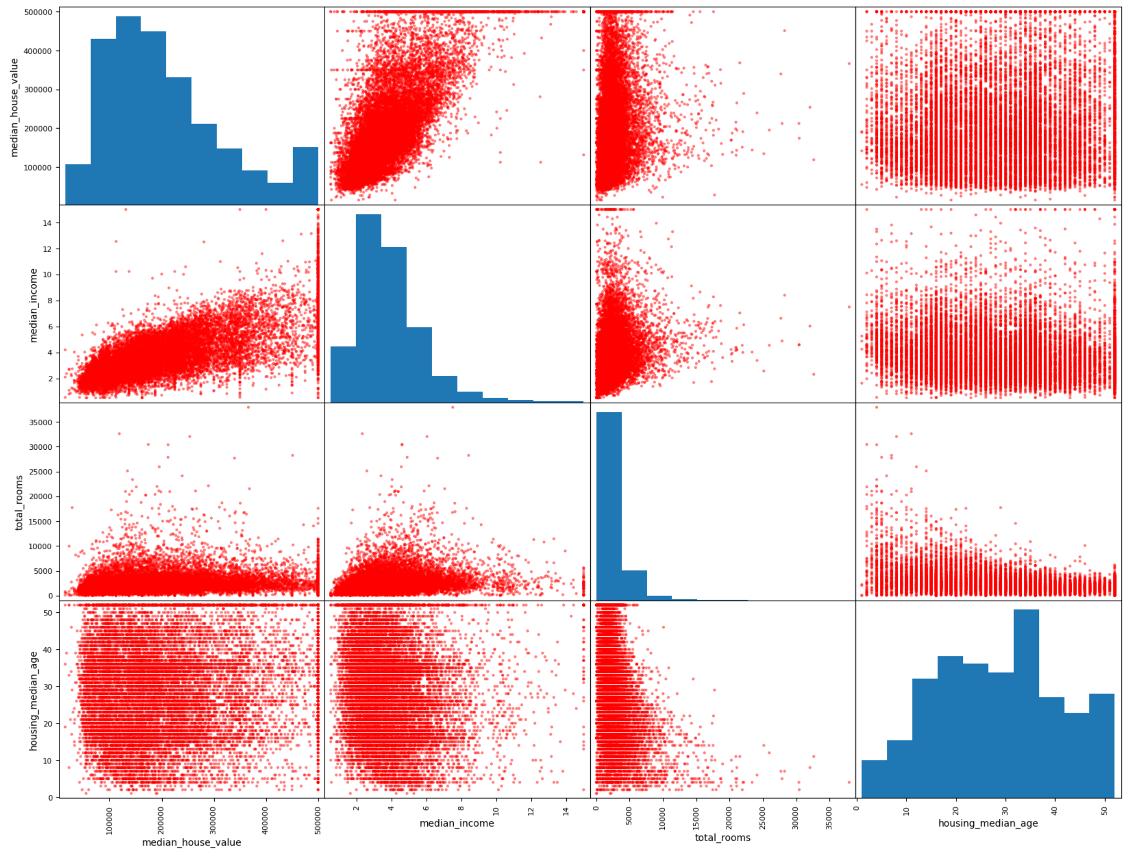
```
longitude                    -0.045456
latitude                     -0.144337
housing_median_age            0.108851
total_rooms                   0.131329
total_bedrooms                0.047040
population                   -0.030342
households                    0.062166
median_income                 0.687446
median_house_value            1.000000
```

0.0.10 Study the correlation matrix for the target attribute and see which attribute has the most amount of correlation with the target attribute.

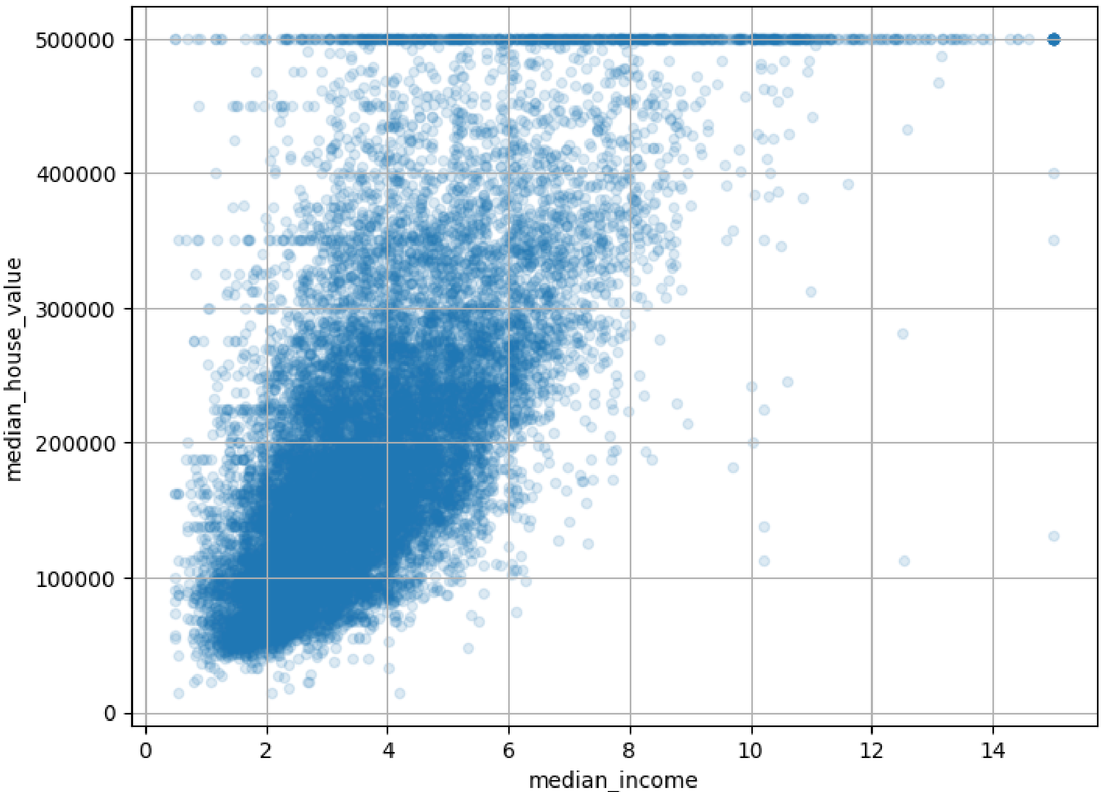
[28]:
```
corr_matrix["median_house_value"].sort_values(ascending = False)
```

[28]:
```
median_house_value    1.000000
median_income         0.687446
total_rooms           0.131329
housing_median_age    0.108851
households            0.062166
total_bedrooms        0.047040
population           -0.030342
longitude            -0.045456
latitude             -0.144337
Name: median_house_value, dtype: float64
```

[29]:
```
from pandas.plotting import scatter_matrix
attributes = ["median_house_value", "median_income", "total_rooms",␣
 ↪"housing_median_age"]
scatter_matrix(data[attributes], figsize=(20, 15), color = 'r')
plt.show()
```



[30]:
```
data.plot(kind = "scatter", x = "median_income", y = "median_house_value", figsize = (8, 6), alpha = 0.15)
plt.grid()
plt.show()
```



0.0.11 Genetate some new features as per requirement which seems to be more promising than the already existing features

[31]:
```
data["population_per_household"] = data["population"] / data["households"]
data["rooms_per_household"] = data["total_rooms"] / data["households"]
data["bedrooms_per_room"] = data["total_bedrooms"] / data["total_rooms"]
```

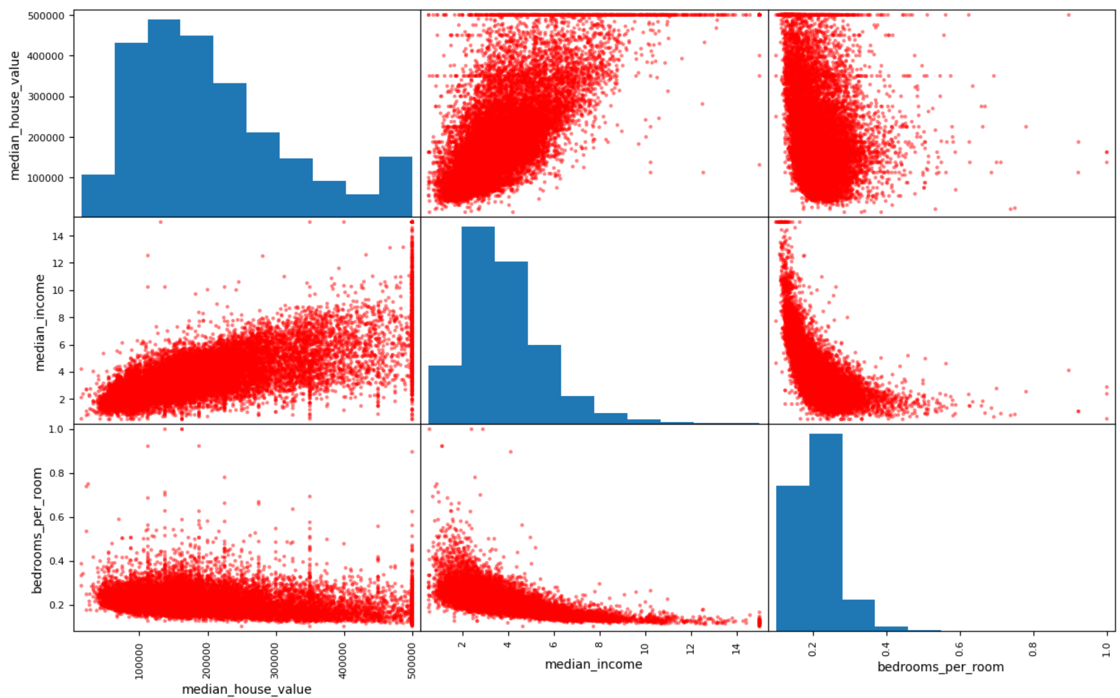
[32]:
```
data_without_ocean = data.drop(["ocean_proximity"], axis = 1)
```

[33]:
```
corr_matrix = data_without_ocean.corr()
corr_matrix["median_house_value"].sort_values(ascending = False)
```

[33]:
```
median_house_value          1.000000
median_income               0.687446
rooms_per_household         0.156215
total_rooms                 0.131329
housing_median_age          0.108851
households                  0.062166
total_bedrooms              0.047040
population_per_household   -0.022300
population                 -0.030342
```

```
longitude                    -0.045456
latitude                     -0.144337
bedrooms_per_room            -0.250534
Name: median_house_value, dtype: float64
```

[34]:
```python
attributes = ["median_house_value", "median_income", "bedrooms_per_room"]
scatter = scatter_matrix(data[attributes], color = 'r', figsize = (15, 9))
```



[35]:
```python
strat_train_set.sample(5)
```

[35]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms
8522	-118.34	33.90	37.0	542.0	105.0
13756	-117.16	34.06	17.0	2285.0	554.0
19462	-120.99	37.68	30.0	1975.0	375.0
2154	-119.80	36.78	50.0	1818.0	374.0
18898	-122.25	38.11	49.0	2365.0	504.0

	population	households	median_income	median_house_value
8522	355.0	118.0	5.5133	227300.0
13756	1412.0	541.0	1.8152	94300.0
19462	732.0	326.0	2.6932	94900.0
2154	737.0	338.0	2.2614	73000.0
18898	1131.0	458.0	2.6133	103100.0

```
      ocean_proximity
8522        <1H OCEAN
13756          INLAND
19462          INLAND
2154           INLAND
18898        NEAR BAY
```

[36]:
```
train_data = strat_train_set.copy()
train_data.drop("median_house_value", inplace = True, axis = 1)
train_data_labels = strat_train_set["median_house_value"].copy()
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 16512 entries, 17262 to 4624
Data columns (total 9 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           16512 non-null  float64
 1   latitude            16512 non-null  float64
 2   housing_median_age  16512 non-null  float64
 3   total_rooms         16512 non-null  float64
 4   total_bedrooms      16358 non-null  float64
 5   population          16512 non-null  float64
 6   households          16512 non-null  float64
 7   median_income       16512 non-null  float64
 8   ocean_proximity     16512 non-null  object
dtypes: float64(8), object(1)
memory usage: 1.3+ MB
```

0.0.12 Fill the null data and the empty data with median of the column or any other strategy like mean or mode using the SimpleImputer class of Sklearn

[37]:
```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy = "median")
train_data_num = train_data.drop("ocean_proximity", axis = 1)
imputer.fit(train_data_num)
tr_temp = imputer.transform(train_data_num)
housing_tr = pd.DataFrame(tr_temp, columns=train_data_num.columns)
```

[38]:
```
housing_tr.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16512 entries, 0 to 16511
Data columns (total 8 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           16512 non-null  float64
 1   latitude            16512 non-null  float64
```

```
 2   housing_median_age  16512 non-null  float64
 3   total_rooms         16512 non-null  float64
 4   total_bedrooms      16512 non-null  float64
 5   population          16512 non-null  float64
 6   households          16512 non-null  float64
 7   median_income       16512 non-null  float64
dtypes: float64(8)
memory usage: 1.0 MB
```

0.0.13 Convert the non-numaric data into numeric categories

[39]:
```python
housing_ocean = train_data[["ocean_proximity"]] # the double square brackets␣
 ↪returns a pandas DataFrame
                                                # a single bracket would have␣
 ↪returned a pandas Series
                                                # but the fit_transform()␣
 ↪method accepts DataFrame as its argument
from sklearn.preprocessing import OrdinalEncoder
ordinal_encoder = OrdinalEncoder()
housing_ocean_encoded = ordinal_encoder.fit_transform(housing_ocean) # a one-D␣
 ↪array is returned, pandas DataFrame is not returned
housing_ocean_encoded
```

[39]:
```
array([[0.],
       [4.],
       [0.],
       …,
       [3.],
       [1.],
       [0.]])
```

0.0.14 Numeric categorires like 1,2,3... might not be the best for every case. Use the OneHotEncoder from Sklearn to create a binary like representation of the non numeric data.

[40]:
```python
from sklearn.preprocessing import OneHotEncoder
one_hot_encoder = OneHotEncoder() # To create binary like representation for␣
 ↪each category
```

[41]:
```python
housing_ocean_1hot = one_hot_encoder.fit_transform(housing_ocean) # returns a␣
 ↪sparse matrix
```

[42]:
```python
one_hot_encoder.categories_
```

[42]:
```
[array(['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'],
       dtype=object)]
```

[43]:
```
housing_ocean_1hot.toarray()
```

[43]:
```
array([[1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1.],
       [1., 0., 0., 0., 0.],
       …,
       [0., 0., 0., 1., 0.],
       [0., 1., 0., 0., 0.],
       [1., 0., 0., 0., 0.]])
```

[44]:
```
#lsit() function extracts the column names from the pandas dataframe
num_attributes = list(housing_tr) #extracting the numerical attribute name from␣
 ↪the dataset
cat_attributes = list(housing_ocean) #extracting the categorical attribute name␣
 ↪from the dataset
cat_attributes
```

[44]:
```
['ocean_proximity']
```

[45]:
```
train_data["rooms_per_household"] = train_data["total_rooms"] /␣
 ↪train_data["households"]
train_data["population_per_household"] = train_data["population"] /␣
 ↪train_data["households"]
train_data["bedrooms_per_room"] = train_data["total_bedrooms"] /␣
 ↪train_data["total_rooms"]
train_data
```

[45]:
```
       longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
17262    -119.71     34.42                52.0       1411.0           324.0
16799    -122.44     37.67                35.0       1814.0           365.0
16718    -120.66     35.49                17.0       4422.0           945.0
5373     -118.38     34.04                36.0       3005.0           771.0
20311    -119.11     34.17                37.0        470.0           105.0
…            …         …                   …           …              …
13893    -116.31     34.13                20.0       2352.0           556.0
20015    -119.08     36.09                25.0       1880.0           339.0
18879    -122.26     38.10                24.0       1213.0           395.0
19612    -121.11     37.47                12.0       2263.0           410.0
4624     -118.32     34.06                52.0        983.0           246.0

       population  households  median_income ocean_proximity  \
17262      1091.0       306.0         4.1062       <1H OCEAN
16799      1025.0       384.0         4.4250      NEAR OCEAN
16718      2307.0       885.0         2.8285       <1H OCEAN
5373       2054.0       758.0         2.0437       <1H OCEAN
20311       522.0        83.0         2.0368      NEAR OCEAN
…             …           …             …              …
```

```
13893     1217.0      481.0         1.6063         INLAND
20015     1003.0      315.0         2.7298         INLAND
18879      699.0      386.0         1.3007       NEAR BAY
19612      913.0      330.0         3.5795         INLAND
4624       578.0      204.0         5.7393      <1H OCEAN

       rooms_per_household  population_per_household  bedrooms_per_room
17262             4.611111                  3.565359           0.229624
16799             4.723958                  2.669271           0.201213
16718             4.996610                  2.606780           0.213704
5373              3.964380                  2.709763           0.256572
20311             5.662651                  6.289157           0.223404
…                      …                         …                  …
13893             4.889813                  2.530146           0.236395
20015             5.968254                  3.184127           0.180319
18879             3.142487                  1.810881           0.325639
19612             6.857576                  2.766667           0.181175
4624              4.818627                  2.833333           0.250254

[16512 rows x 12 columns]
```

[46]:
```python
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

numerical_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy="median")),
    ('std_scaler', StandardScaler()),
])
train_data_without_ocean = train_data.drop("ocean_proximity", axis = 1)
num_attributes = list(train_data_without_ocean)
```

[47]:
```python
from sklearn.compose import ColumnTransformer
full_pipeline = ColumnTransformer([
     ("num", numerical_pipeline, num_attributes),
     ("cat", OneHotEncoder(), cat_attributes),
])
housing_prepared = full_pipeline.fit_transform(train_data)
housing_prepared
```

[47]:
```
array([[-0.06764406, -0.57094951,  1.85663063, …,  0.        ,
          0.        ,  0.        ],
       [-1.43149467,  0.95093743,  0.50408653, …,  0.        ,
          0.        ,  1.        ],
       [-0.54224409, -0.0698975 , -0.92801899, …,  0.        ,
          0.        ,  0.        ],
       …,
       [-1.34157046,  1.15229477, -0.37108906, …,  0.        ,
```

```
         1.        ,  0.        ],
       [-0.76705463,  0.85728285, -1.32582607, …,  0.        ,
         0.        ,  0.        ],
       [ 0.62677072, -0.73952775,  1.85663063, …,  0.        ,
         0.        ,  0.        ]])
```

[48]:
```
housing_prepared.shape
```

[48]:
```
(16512, 16)
```

[60]:
```
housing_pandas = pd.DataFrame(housing_prepared)
housing_pandas.info()
```

```
&lt;class 'pandas.core.frame.DataFrame'&gt;
RangeIndex: 16512 entries, 0 to 16511
Data columns (total 16 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   0       16512 non-null  float64
 1   1       16512 non-null  float64
 2   2       16512 non-null  float64
 3   3       16512 non-null  float64
 4   4       16512 non-null  float64
 5   5       16512 non-null  float64
 6   6       16512 non-null  float64
 7   7       16512 non-null  float64
 8   8       16512 non-null  float64
 9   9       16512 non-null  float64
 10  10      16512 non-null  float64
 11  11      16512 non-null  float64
 12  12      16512 non-null  float64
 13  13      16512 non-null  float64
 14  14      16512 non-null  float64
 15  15      16512 non-null  float64
dtypes: float64(16)
memory usage: 2.0 MB
```

[59]:
```
train_data
```

[59]:
```
       longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
17262    -119.71     34.42                52.0       1411.0           324.0
16799    -122.44     37.67                35.0       1814.0           365.0
16718    -120.66     35.49                17.0       4422.0           945.0
5373     -118.38     34.04                36.0       3005.0           771.0
20311    -119.11     34.17                37.0        470.0           105.0
…            …        …                  …           …               …
13893    -116.31     34.13                20.0       2352.0           556.0
20015    -119.08     36.09                25.0       1880.0           339.0
```

```
18879   -122.26   38.10                24.0      1213.0           395.0
19612   -121.11   37.47                12.0      2263.0           410.0
4624    -118.32   34.06                52.0       983.0           246.0

       population  households  median_income ocean_proximity  \
17262      1091.0       306.0         4.1062       <1H OCEAN
16799      1025.0       384.0         4.4250      NEAR OCEAN
16718      2307.0       885.0         2.8285       <1H OCEAN
5373       2054.0       758.0         2.0437       <1H OCEAN
20311       522.0        83.0         2.0368      NEAR OCEAN
…             …          …             …             …
13893      1217.0       481.0         1.6063          INLAND
20015      1003.0       315.0         2.7298          INLAND
18879       699.0       386.0         1.3007        NEAR BAY
19612       913.0       330.0         3.5795          INLAND
4624        578.0       204.0         5.7393       <1H OCEAN

       rooms_per_household  population_per_household  bedrooms_per_room
17262             4.611111                  3.565359           0.229624
16799             4.723958                  2.669271           0.201213
16718             4.996610                  2.606780           0.213704
5373              3.964380                  2.709763           0.256572
20311             5.662651                  6.289157           0.223404
…                    …                        …                 …
13893             4.889813                  2.530146           0.236395
20015             5.968254                  3.184127           0.180319
18879             3.142487                  1.810881           0.325639
19612             6.857576                  2.766667           0.181175
4624              4.818627                  2.833333           0.250254

[16512 rows x 12 columns]
```

[54]:
```
train_data_labels
```

[54]:
```
17262    252900.0
16799    268400.0
16718    171300.0
5373     309100.0
20311    243800.0
           …
13893     55400.0
20015    103400.0
18879     94600.0
19612    145600.0
4624     500001.0
Name: median_house_value, Length: 16512, dtype: float64
```

[93]:
```python
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(housing_pandas, train_data_labels)
```

[93]:
```
LinearRegression()
```

[94]:
```python
some_data = housing_pandas.iloc[:5]
some_labels = train_data_labels[:5]
predictions = lin_reg.predict(some_data)
print("Predictions: ", predictions)
print("Actual values: ", list(some_labels))
```

```
Predictions:  [278601.94507113 271523.71773396 209549.98164306 173011.51131126
 167052.31181711]
Actual values:  [252900.0, 268400.0, 171300.0, 309100.0, 243800.0]
```

[127]:
```python
from sklearn.metrics import mean_squared_error
full_predictions = lin_reg.predict(housing_pandas)
lin_mse = mean_squared_error(train_data_labels, full_predictions)
lin_rmse = np.sqrt(lin_mse)
print("The root mean squared error: ", lin_rmse)
```

```
The root mean squared error:  67985.1053709194
```

[150]:
```python
from sklearn.tree import DecisionTreeRegressor
dtree = DecisionTreeRegressor()
dtree.fit(housing_pandas, train_data_labels)
```

[150]:
```
DecisionTreeRegressor()
```

[151]:
```python
dtree_pred = dtree.predict(housing_pandas)
dtree_mse = mean_squared_error(train_data_labels, dtree_pred)
dtree_rmse = np.sqrt(dtree_mse)
print("the root mean squared error for decision trees is: ", dtree_rmse)
```

```
the root mean squared error for decision trees is:  0.0
```

[173]:
```python
from sklearn.model_selection import cross_val_score
scores = cross_val_score(dtree, housing_pandas, train_data_labels,␣
 ↪scoring="neg_mean_squared_error", cv=15)
cross_rmse_scores = np.sqrt(-scores)
print("the scores for cross validation are: ", cross_rmse_scores)
```

```
the scores for cross validation are:  [67257.43520675 71852.33667335
68977.80188973 74086.12759078
 64239.53029938 67975.2019006  73006.40212952 65445.25287954
 68275.38133543 71766.37645894 72261.75184246 67908.84147885
 77873.75267504 67872.26095493 66254.00981927]
```

[174]:
```python
def displayScores(score):
    print("The respective scores are:")
    count = int(0)
    for i in score:
        count+=1
        print(f"{count}\t{i} ")
    print("The mean of the scores are: ", score.mean())
    print("The standard deviation of the scores are: ", score.std())
displayScores(cross_rmse_scores)
```

```
The respective scores are:
1       67257.43520675147
2       71852.33667335063
3       68977.80188972827
4       74086.12759078329
5       64239.53029938324
6       67975.20190060338
7       73006.40212952126
8       65445.25287953931
9       68275.3813354252
10      71766.37645894017
11      72261.75184245536
12      67908.84147884641
13      77873.75267503632
14      67872.26095493317
15      66254.00981926918
The mean of the scores are:  69670.16420897111
The standard deviation of the scores are:  3561.533537607548
```

[172]:
```python
lin_scores = cross_val_score(lin_reg, housing_pandas, train_data_labels,␣
 ↪scoring = "neg_mean_squared_error", cv=15)
cross_rmse_lin_scores = np.sqrt(-lin_scores)
displayScores(cross_rmse_lin_scores)
```

```
The respective scores are:
1       68371.43697623984
2       67661.67533851672
3       70722.760975738
4       66954.54594761398
5       67003.3670553982
6       66006.49105720749
7       68414.63556127886
8       67705.01478977986
9       71401.59845540774
10      65663.67644721322
11      75758.90604686084
12      68083.7002127005
13      71265.99960692934
```

```
14      68625.11948894041
15      60614.03015498459
The mean of the scores are:  68283.53054098731
The standard deviation of the scores are:  3221.0378035014933
```

[176]:
```
from sklearn.ensemble import RandomForestRegressor
rforest = RandomForestRegressor()
rforest.fit(housing_pandas, train_data_labels)
rforest_pred = rforest.predict(housing_pandas)
rforest_mse = mean_squared_error(train_data_labels, rforest_pred)
rforest_rmse = np.sqrt(rforest_mse)
print("The root mean squared error for random forest is: ", rforest_rmse)
```

```
The root mean squared error for random forest is:  18596.745302281124
```

[178]:
```
rforest_scores = cross_val_score(rforest, housing_pandas, train_data_labels,␣
 ↪scoring="neg_mean_squared_error", cv=10)
cross_rmse_rforest_scores = np.sqrt(-rforest_scores)
displayScores(cross_rmse_rforest_scores)
```

```
The respective scores are:
1       48875.00570475645
2       50266.613530296076
3       48323.779871783576
4       48911.37254928006
5       51421.72862895735
6       51010.11793763163
7       50929.0677284739
8       49671.65744326685
9       54744.03722778164
10      46908.2474471355
The mean of the scores are:  50106.1628069363
The standard deviation of the scores are:  2032.1481849387492
```

[182]:
```
import joblib
joblib.dump(rforest, "random_forest.pkl")
joblib.dump(dtree, "decision_tree.pkl")
joblib.dump(lin_reg, "linear_regression.pkl")
```

[182]:
```
['linear_regression.pkl']
```

[183]:
```
rforest_rmse_scores = cross_val_score(rforest, housing_pandas,␣
 ↪train_data_labels, scoring="neg_root_mean_squared_error", cv=10)
displayScores(rforest_rmse_scores)
```

```
The respective scores are:
1       -49220.108097126016
2       -50001.45921668029
3       -48159.57480878568
```

```
4       -49087.46423286893
5       -51336.25566407327
6       -51030.0967187676
7       -51487.365010733374
8       -50013.94827167112
9       -54837.08845616376
10      -46486.57140406693
The mean of the scores are:  -50165.9931880937
The standard deviation of the scores are:  2129.6317861543453
```

[184]:
```python
from sklearn.model_selection import GridSearchCV
param_grid = [
    {
        "n_estimators":[10, 20, 40, 80, 160],
        "criterion":["squared_error", "absolute_error", "friedman_mse",␣
 ↪"poisson"],
        "max_features":["sqrt", "log2", 2, 4, 6, 8, 10, 12],

    }
]
rforest_regressor = RandomForestRegressor()
grid_search = GridSearchCV(rforest_regressor,
                           param_grid=param_grid,
                           scoring='neg_mean_squared_error',
                           cv=10, return_train_score=True)
grid_search.fit(housing_pandas, train_data_labels)
```

[184]:
```
GridSearchCV(cv=10, estimator=RandomForestRegressor(),
             param_grid=[{'criterion': ['squared_error', 'absolute_error',
                                        'friedman_mse', 'poisson'],
                          'max_features': ['sqrt', 'log2', 2, 4, 6, 8, 10, 12],
                          'n_estimators': [10, 20, 40, 80, 160]}],
             return_train_score=True, scoring='neg_mean_squared_error')
```

[188]:
```python
def displayGridSearchResults(res):
    print("The best parameters are: ", res.best_params_)
    print("The best model is: ", res.best_estimator_)
    print("Detailed results:")
    cv_res = res.cv_results_
    count = 1
    for (mean_scr, params) in zip(cv_res['mean_train_score'], cv_res['params']):
        print(f"{count}. {np.sqrt(-mean_scr)}\t{params}")
        count+=1
displayGridSearchResults(grid_search)
```

```
The best parameters are:  {'criterion': 'absolute_error', 'max_features': 6,
'n_estimators': 160}
The best model is:  RandomForestRegressor(criterion='absolute_error',
```

```
max_features=6,
                      n_estimators=160)
Detailed results:
1. 22625.847783351954   {'criterion': 'squared_error', 'max_features': 'sqrt',
'n_estimators': 10}
2. 20352.897560653953   {'criterion': 'squared_error', 'max_features': 'sqrt',
'n_estimators': 20}
3. 19223.779308037556   {'criterion': 'squared_error', 'max_features': 'sqrt',
'n_estimators': 40}
4. 18578.491355320544   {'criterion': 'squared_error', 'max_features': 'sqrt',
'n_estimators': 80}
5. 18341.13251380937    {'criterion': 'squared_error', 'max_features': 'sqrt',
'n_estimators': 160}
6. 22515.60901590808    {'criterion': 'squared_error', 'max_features': 'log2',
'n_estimators': 10}
7. 20421.908085916988   {'criterion': 'squared_error', 'max_features': 'log2',
'n_estimators': 20}
8. 19139.6804886989     {'criterion': 'squared_error', 'max_features': 'log2',
'n_estimators': 40}
9. 18656.48143962623    {'criterion': 'squared_error', 'max_features': 'log2',
'n_estimators': 80}
10. 18265.373053319254  {'criterion': 'squared_error', 'max_features': 'log2',
'n_estimators': 160}
11. 23856.91201551      {'criterion': 'squared_error', 'max_features': 2,
'n_estimators': 10}
12. 21433.68992213853   {'criterion': 'squared_error', 'max_features': 2,
'n_estimators': 20}
13. 20113.967592617184  {'criterion': 'squared_error', 'max_features': 2,
'n_estimators': 40}
14. 19463.488311133195  {'criterion': 'squared_error', 'max_features': 2,
'n_estimators': 80}
15. 19090.15722488032   {'criterion': 'squared_error', 'max_features': 2,
'n_estimators': 160}
16. 22692.577297517582  {'criterion': 'squared_error', 'max_features': 4,
'n_estimators': 10}
17. 20334.242180920795  {'criterion': 'squared_error', 'max_features': 4,
'n_estimators': 20}
18. 19215.31463224876   {'criterion': 'squared_error', 'max_features': 4,
'n_estimators': 40}
19. 18622.905819329986  {'criterion': 'squared_error', 'max_features': 4,
'n_estimators': 80}
20. 18313.61554210922   {'criterion': 'squared_error', 'max_features': 4,
'n_estimators': 160}
21. 22186.95747512457   {'criterion': 'squared_error', 'max_features': 6,
'n_estimators': 10}
22. 20205.04591129518   {'criterion': 'squared_error', 'max_features': 6,
'n_estimators': 20}
23. 19118.669481483088  {'criterion': 'squared_error', 'max_features': 6,
```

```
'n_estimators': 40}
24. 18490.92877929516   {'criterion': 'squared_error', 'max_features': 6,
'n_estimators': 80}
25. 18170.624454169458  {'criterion': 'squared_error', 'max_features': 6,
'n_estimators': 160}
26. 22170.0220212713    {'criterion': 'squared_error', 'max_features': 8,
'n_estimators': 10}
27. 20122.291602080237  {'criterion': 'squared_error', 'max_features': 8,
'n_estimators': 20}
28. 19068.382856065426  {'criterion': 'squared_error', 'max_features': 8,
'n_estimators': 40}
29. 18430.9442898605    {'criterion': 'squared_error', 'max_features': 8,
'n_estimators': 80}
30. 18193.503781115138  {'criterion': 'squared_error', 'max_features': 8,
'n_estimators': 160}
31. 21994.798691507876  {'criterion': 'squared_error', 'max_features': 10,
'n_estimators': 10}
32. 20113.07665887661   {'criterion': 'squared_error', 'max_features': 10,
'n_estimators': 20}
33. 19052.306018622865  {'criterion': 'squared_error', 'max_features': 10,
'n_estimators': 40}
34. 18534.19739052396   {'criterion': 'squared_error', 'max_features': 10,
'n_estimators': 80}
35. 18248.35143922619   {'criterion': 'squared_error', 'max_features': 10,
'n_estimators': 160}
36. 22313.986575944957  {'criterion': 'squared_error', 'max_features': 12,
'n_estimators': 10}
37. 20287.149700031918  {'criterion': 'squared_error', 'max_features': 12,
'n_estimators': 20}
38. 19243.15232888584   {'criterion': 'squared_error', 'max_features': 12,
'n_estimators': 40}
39. 18620.69448693786   {'criterion': 'squared_error', 'max_features': 12,
'n_estimators': 80}
40. 18350.75158229042   {'criterion': 'squared_error', 'max_features': 12,
'n_estimators': 160}
41. 23026.907510642155  {'criterion': 'absolute_error', 'max_features': 'sqrt',
'n_estimators': 10}
42. 20742.37608341811   {'criterion': 'absolute_error', 'max_features': 'sqrt',
'n_estimators': 20}
43. 19530.074129337703  {'criterion': 'absolute_error', 'max_features': 'sqrt',
'n_estimators': 40}
44. 18976.01588969135   {'criterion': 'absolute_error', 'max_features': 'sqrt',
'n_estimators': 80}
45. 18676.937566586595  {'criterion': 'absolute_error', 'max_features': 'sqrt',
'n_estimators': 160}
46. 22873.09078480179   {'criterion': 'absolute_error', 'max_features': 'log2',
'n_estimators': 10}
47. 20810.76521595372   {'criterion': 'absolute_error', 'max_features': 'log2',
```

```
'n_estimators': 20}
48. 19636.46984456111   {'criterion': 'absolute_error', 'max_features': 'log2',
'n_estimators': 40}
49. 18989.28522753006   {'criterion': 'absolute_error', 'max_features': 'log2',
'n_estimators': 80}
50. 18640.248012250177  {'criterion': 'absolute_error', 'max_features': 'log2',
'n_estimators': 160}
51. 24662.739944969235  {'criterion': 'absolute_error', 'max_features': 2,
'n_estimators': 10}
52. 22170.34968160643   {'criterion': 'absolute_error', 'max_features': 2,
'n_estimators': 20}
53. 20915.11673965792   {'criterion': 'absolute_error', 'max_features': 2,
'n_estimators': 40}
54. 20137.48778405992   {'criterion': 'absolute_error', 'max_features': 2,
'n_estimators': 80}
55. 19892.776366509323  {'criterion': 'absolute_error', 'max_features': 2,
'n_estimators': 160}
56. 23151.82129194066   {'criterion': 'absolute_error', 'max_features': 4,
'n_estimators': 10}
57. 20916.87817365437   {'criterion': 'absolute_error', 'max_features': 4,
'n_estimators': 20}
58. 19591.86425451383   {'criterion': 'absolute_error', 'max_features': 4,
'n_estimators': 40}
59. 18974.084413471246  {'criterion': 'absolute_error', 'max_features': 4,
'n_estimators': 80}
60. 18653.184850560763  {'criterion': 'absolute_error', 'max_features': 4,
'n_estimators': 160}
61. 22827.124222321385  {'criterion': 'absolute_error', 'max_features': 6,
'n_estimators': 10}
62. 20521.766282548368  {'criterion': 'absolute_error', 'max_features': 6,
'n_estimators': 20}
63. 19321.931081511255  {'criterion': 'absolute_error', 'max_features': 6,
'n_estimators': 40}
64. 18712.25261355186   {'criterion': 'absolute_error', 'max_features': 6,
'n_estimators': 80}
65. 18435.490630547458  {'criterion': 'absolute_error', 'max_features': 6,
'n_estimators': 160}
66. 22537.60531247283   {'criterion': 'absolute_error', 'max_features': 8,
'n_estimators': 10}
67. 20344.0634262903    {'criterion': 'absolute_error', 'max_features': 8,
'n_estimators': 20}
68. 19305.761818640214  {'criterion': 'absolute_error', 'max_features': 8,
'n_estimators': 40}
69. 18738.01797370373   {'criterion': 'absolute_error', 'max_features': 8,
'n_estimators': 80}
70. 18460.517588549836  {'criterion': 'absolute_error', 'max_features': 8,
'n_estimators': 160}
71. 22535.27035168526   {'criterion': 'absolute_error', 'max_features': 10,
```

```
'n_estimators': 10}
72. 20431.171601457216  {'criterion': 'absolute_error', 'max_features': 10,
'n_estimators': 20}
73. 19289.535114611874  {'criterion': 'absolute_error', 'max_features': 10,
'n_estimators': 40}
74. 18770.99496560422   {'criterion': 'absolute_error', 'max_features': 10,
'n_estimators': 80}
75. 18535.83067111891   {'criterion': 'absolute_error', 'max_features': 10,
'n_estimators': 160}
76. 22613.805045473513  {'criterion': 'absolute_error', 'max_features': 12,
'n_estimators': 10}
77. 20582.25139742802   {'criterion': 'absolute_error', 'max_features': 12,
'n_estimators': 20}
78. 19507.311805025023  {'criterion': 'absolute_error', 'max_features': 12,
'n_estimators': 40}
79. 18930.79205505346   {'criterion': 'absolute_error', 'max_features': 12,
'n_estimators': 80}
80. 18679.375062145864  {'criterion': 'absolute_error', 'max_features': 12,
'n_estimators': 160}
81. 22566.079853946063  {'criterion': 'friedman_mse', 'max_features': 'sqrt',
'n_estimators': 10}
82. 20515.624877954     {'criterion': 'friedman_mse', 'max_features': 'sqrt',
'n_estimators': 20}
83. 19193.990805460508  {'criterion': 'friedman_mse', 'max_features': 'sqrt',
'n_estimators': 40}
84. 18613.979408524865  {'criterion': 'friedman_mse', 'max_features': 'sqrt',
'n_estimators': 80}
85. 18311.196592576624  {'criterion': 'friedman_mse', 'max_features': 'sqrt',
'n_estimators': 160}
86. 22699.51288186796   {'criterion': 'friedman_mse', 'max_features': 'log2',
'n_estimators': 10}
87. 20404.32284676581   {'criterion': 'friedman_mse', 'max_features': 'log2',
'n_estimators': 20}
88. 19214.54302225376   {'criterion': 'friedman_mse', 'max_features': 'log2',
'n_estimators': 40}
89. 18655.09935659817   {'criterion': 'friedman_mse', 'max_features': 'log2',
'n_estimators': 80}
90. 18321.62469453262   {'criterion': 'friedman_mse', 'max_features': 'log2',
'n_estimators': 160}
91. 23895.215325600057  {'criterion': 'friedman_mse', 'max_features': 2,
'n_estimators': 10}
92. 21411.312794227284  {'criterion': 'friedman_mse', 'max_features': 2,
'n_estimators': 20}
93. 20127.144962491304  {'criterion': 'friedman_mse', 'max_features': 2,
'n_estimators': 40}
94. 19467.082088758427  {'criterion': 'friedman_mse', 'max_features': 2,
'n_estimators': 80}
95. 19098.771578653355  {'criterion': 'friedman_mse', 'max_features': 2,
```

```
'n_estimators': 160}
96. 22689.155593696392  {'criterion': 'friedman_mse', 'max_features': 4,
'n_estimators': 10}
97. 20404.436547159323  {'criterion': 'friedman_mse', 'max_features': 4,
'n_estimators': 20}
98. 19223.943644592353  {'criterion': 'friedman_mse', 'max_features': 4,
'n_estimators': 40}
99. 18596.026833570406  {'criterion': 'friedman_mse', 'max_features': 4,
'n_estimators': 80}
100. 18276.465766096073 {'criterion': 'friedman_mse', 'max_features': 4,
'n_estimators': 160}
101. 22290.18627260149  {'criterion': 'friedman_mse', 'max_features': 6,
'n_estimators': 10}
102. 20146.241040394812 {'criterion': 'friedman_mse', 'max_features': 6,
'n_estimators': 20}
103. 19046.587574603895 {'criterion': 'friedman_mse', 'max_features': 6,
'n_estimators': 40}
104. 18469.404988330767 {'criterion': 'friedman_mse', 'max_features': 6,
'n_estimators': 80}
105. 18196.595081726704 {'criterion': 'friedman_mse', 'max_features': 6,
'n_estimators': 160}
106. 22216.088056262375 {'criterion': 'friedman_mse', 'max_features': 8,
'n_estimators': 10}
107. 20115.2553023982   {'criterion': 'friedman_mse', 'max_features': 8,
'n_estimators': 20}
108. 19012.917274445805 {'criterion': 'friedman_mse', 'max_features': 8,
'n_estimators': 40}
109. 18507.332307460605 {'criterion': 'friedman_mse', 'max_features': 8,
'n_estimators': 80}
110. 18187.850245871294 {'criterion': 'friedman_mse', 'max_features': 8,
'n_estimators': 160}
111. 22027.96360786555  {'criterion': 'friedman_mse', 'max_features': 10,
'n_estimators': 10}
112. 20065.39499426595  {'criterion': 'friedman_mse', 'max_features': 10,
'n_estimators': 20}
113. 19153.004349503073 {'criterion': 'friedman_mse', 'max_features': 10,
'n_estimators': 40}
114. 18617.323265998813 {'criterion': 'friedman_mse', 'max_features': 10,
'n_estimators': 80}
115. 18268.962436281454 {'criterion': 'friedman_mse', 'max_features': 10,
'n_estimators': 160}
116. 22019.87945392713  {'criterion': 'friedman_mse', 'max_features': 12,
'n_estimators': 10}
117. 20188.37291133241  {'criterion': 'friedman_mse', 'max_features': 12,
'n_estimators': 20}
118. 19175.52895117567  {'criterion': 'friedman_mse', 'max_features': 12,
'n_estimators': 40}
119. 18616.319990412736 {'criterion': 'friedman_mse', 'max_features': 12,
```

```
'n_estimators': 80}
120. 18371.704630857817 {'criterion': 'friedman_mse', 'max_features': 12,
'n_estimators': 160}
121. 22500.31665616187  {'criterion': 'poisson', 'max_features': 'sqrt',
'n_estimators': 10}
122. 20424.641963569786 {'criterion': 'poisson', 'max_features': 'sqrt',
'n_estimators': 20}
123. 19288.617667747094 {'criterion': 'poisson', 'max_features': 'sqrt',
'n_estimators': 40}
124. 18673.271325121157 {'criterion': 'poisson', 'max_features': 'sqrt',
'n_estimators': 80}
125. 18339.461722266857 {'criterion': 'poisson', 'max_features': 'sqrt',
'n_estimators': 160}
126. 22594.596834182175 {'criterion': 'poisson', 'max_features': 'log2',
'n_estimators': 10}
127. 20416.160112474994 {'criterion': 'poisson', 'max_features': 'log2',
'n_estimators': 20}
128. 19264.972482781468 {'criterion': 'poisson', 'max_features': 'log2',
'n_estimators': 40}
129. 18660.15291762746  {'criterion': 'poisson', 'max_features': 'log2',
'n_estimators': 80}
130. 18311.486445189665 {'criterion': 'poisson', 'max_features': 'log2',
'n_estimators': 160}
131. 23947.707831175805 {'criterion': 'poisson', 'max_features': 2,
'n_estimators': 10}
132. 21504.34829814254  {'criterion': 'poisson', 'max_features': 2,
'n_estimators': 20}
133. 20276.680163367273 {'criterion': 'poisson', 'max_features': 2,
'n_estimators': 40}
134. 19561.771182870943 {'criterion': 'poisson', 'max_features': 2,
'n_estimators': 80}
135. 19167.942525848226 {'criterion': 'poisson', 'max_features': 2,
'n_estimators': 160}
136. 22746.881780368258 {'criterion': 'poisson', 'max_features': 4,
'n_estimators': 10}
137. 20404.883430119775 {'criterion': 'poisson', 'max_features': 4,
'n_estimators': 20}
138. 19297.99380539365  {'criterion': 'poisson', 'max_features': 4,
'n_estimators': 40}
139. 18678.711106290473 {'criterion': 'poisson', 'max_features': 4,
'n_estimators': 80}
140. 18317.987979142137 {'criterion': 'poisson', 'max_features': 4,
'n_estimators': 160}
141. 22246.33828261774  {'criterion': 'poisson', 'max_features': 6,
'n_estimators': 10}
142. 20101.77750881709  {'criterion': 'poisson', 'max_features': 6,
'n_estimators': 20}
143. 18990.968762663902 {'criterion': 'poisson', 'max_features': 6,
```

```
'n_estimators': 40}
144. 18446.700368477508 {'criterion': 'poisson', 'max_features': 6,
'n_estimators': 80}
145. 18228.108527198892 {'criterion': 'poisson', 'max_features': 6,
'n_estimators': 160}
146. 22250.20915180805  {'criterion': 'poisson', 'max_features': 8,
'n_estimators': 10}
147. 20147.161223091953 {'criterion': 'poisson', 'max_features': 8,
'n_estimators': 20}
148. 19006.02694446665  {'criterion': 'poisson', 'max_features': 8,
'n_estimators': 40}
149. 18454.835336861786 {'criterion': 'poisson', 'max_features': 8,
'n_estimators': 80}
150. 18180.59536326515  {'criterion': 'poisson', 'max_features': 8,
'n_estimators': 160}
151. 22055.19809167954  {'criterion': 'poisson', 'max_features': 10,
'n_estimators': 10}
152. 20129.396213315045 {'criterion': 'poisson', 'max_features': 10,
'n_estimators': 20}
153. 19159.150779553747 {'criterion': 'poisson', 'max_features': 10,
'n_estimators': 40}
154. 18560.38889576643  {'criterion': 'poisson', 'max_features': 10,
'n_estimators': 80}
155. 18297.120184063475 {'criterion': 'poisson', 'max_features': 10,
'n_estimators': 160}
156. 22076.89987152067  {'criterion': 'poisson', 'max_features': 12,
'n_estimators': 10}
157. 20241.632198015617 {'criterion': 'poisson', 'max_features': 12,
'n_estimators': 20}
158. 19159.874994388032 {'criterion': 'poisson', 'max_features': 12,
'n_estimators': 40}
159. 18631.339609351322 {'criterion': 'poisson', 'max_features': 12,
'n_estimators': 80}
160. 18350.29252309775  {'criterion': 'poisson', 'max_features': 12,
'n_estimators': 160}
```

[191]:
```
grid_search.best_estimator_.feature_importances_
```

[191]:
```
array([9.73320501e-02, 8.74341672e-02, 5.36958862e-02, 3.11174851e-02,
       2.90162558e-02, 3.05559715e-02, 2.82711219e-02, 2.53282748e-01,
       6.33132844e-02, 1.06112869e-01, 8.06153510e-02, 1.60852682e-02,
       1.13016543e-01, 1.43418212e-04, 3.32079672e-03, 6.68678390e-03])
```

[215]:
```
attributes = full_pipeline.named_transformers_["cat"]
cat_attrib = list(attributes.categories_[0])
num_attributes = list(train_data)
num_attributes.remove("ocean_proximity")
attribs = num_attributes + cat_attrib
```

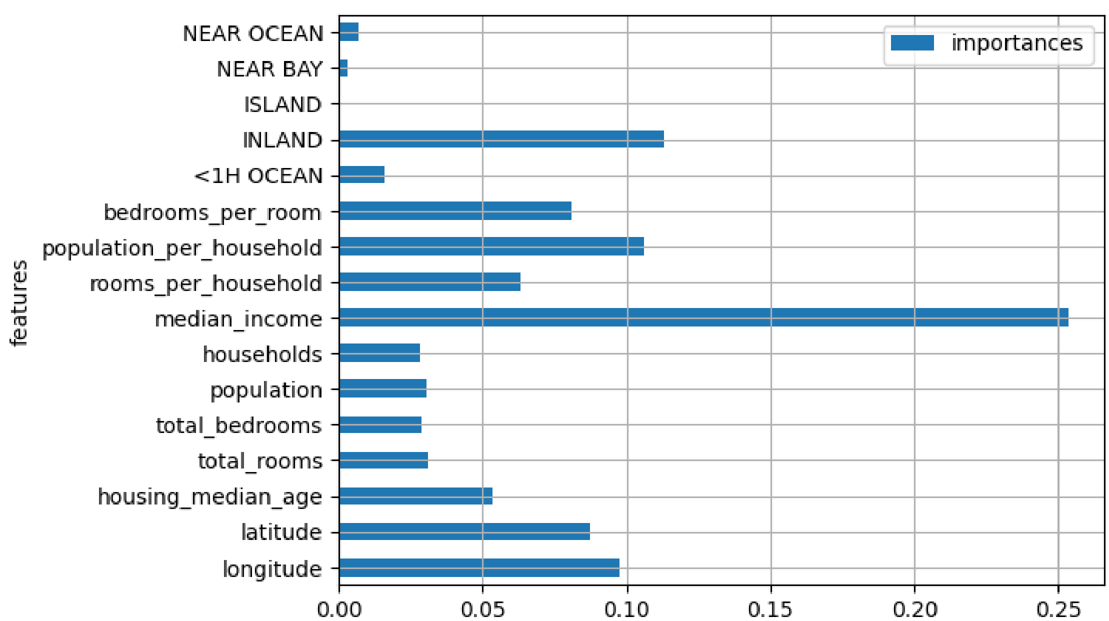
```
feature_imp = zip(attribs, grid_search.best_estimator_.feature_importances_)
feature_imp = list(feature_imp)
imp_df = pd.DataFrame({"importances" : grid_search.best_estimator_.
↪feature_importances_, "features":attribs})
imp_df
```

[215]:

	importances	features
0	0.097332	longitude
1	0.087434	latitude
2	0.053696	housing_median_age
3	0.031117	total_rooms
4	0.029016	total_bedrooms
5	0.030556	population
6	0.028271	households
7	0.253283	median_income
8	0.063313	rooms_per_household
9	0.106113	population_per_household
10	0.080615	bedrooms_per_room
11	0.016085	<1H OCEAN
12	0.113017	INLAND
13	0.000143	ISLAND
14	0.003321	NEAR BAY
15	0.006687	NEAR OCEAN

[223]:

```
imp_df.plot(kind="barh", x="features", y="importances")
plt.grid(True)
plt.show()
```



[226]:
```
final_model = RandomForestRegressor(max_features = 6, n_estimators = 160)
final_model_scores = cross_val_score(final_model, housing_pandas,␣
 ↪train_data_labels, scoring="neg_mean_squared_error", cv=10)
cross_rmse_final_scores = np.sqrt(-final_model_scores)
displayScores(cross_rmse_final_scores)
```

```
The respective scores are:
1       47896.94133514639
2       48645.64940963574
3       46873.125528126744
4       47668.40549490622
5       50545.41077827432
6       50000.348844412045
7       49240.825023719684
8       48384.67304392187
9       52751.70832967562
10      45626.28059811532
The mean of the scores are:  48763.33683859339
The standard deviation of the scores are:  1906.8786384322414
```

[228]:
```
final_model1 = RandomForestRegressor(max_features = 6, n_estimators = 160,␣
 ↪criterion="absolute_error")
final_model_scores1 = cross_val_score(final_model1, housing_pandas,␣
 ↪train_data_labels, scoring="neg_mean_squared_error", cv=10)
cross_rmse_final_scores1 = np.sqrt(-final_model_scores1)
displayScores(cross_rmse_final_scores1)
```

```
The respective scores are:
1       47825.56825660464
2       48777.464431791115
3       47132.82195933664
4       47079.74996668004
5       50378.52414103982
6       49627.90406197995
7       49816.49323753231
8       48310.103469811605
9       53113.33635843003
10      45257.67058058815
The mean of the scores are:  48731.96364637943
The standard deviation of the scores are:  2055.379949915629
```

[243]:
```
final_model.fit(housing_pandas, train_data_labels)
```

[243]:
```
RandomForestRegressor(max_features=6, n_estimators=160)
```

[245]:
```
final_model1.fit(housing_pandas, train_data_labels)
```

[245]: RandomForestRegressor(criterion='absolute_error', max_features=6,
                      n_estimators=160)

[241]:
```
test_data_labels = strat_test_set["median_house_value"].copy()
strat_test_set.drop("median_house_value", inplace = True, axis = 1)
strat_test_set["rooms_per_household"] = strat_test_set["total_rooms"] /␣
 ↪strat_test_set["households"]
strat_test_set["population_per_household"] = strat_test_set["population"] /␣
 ↪strat_test_set["households"]
strat_test_set["bedrooms_per_room"] = strat_test_set["total_bedrooms"] /␣
 ↪strat_test_set["total_rooms"]
test_data = full_pipeline.fit_transform(strat_test_set)
test_data.shape
```

[241]: (4128, 16)

[244]:
```
final_pred = final_model.predict(test_data)
final_mse = mean_squared_error(test_data_labels, final_pred)
final_rmse = np.sqrt(final_mse)
print("the root mean squared error for decision trees is: ", final_rmse)
```

```
the root mean squared error for decision trees is:  62217.57528630097
```

[246]:
```
final_pred1 = final_model1.predict(test_data)
final_mse1 = mean_squared_error(test_data_labels, final_pred1)
final_rmse1 = np.sqrt(final_mse1)
print("the root mean squared error for decision trees is: ", final_rmse1)
```

```
the root mean squared error for decision trees is:  61021.16577708002
```

[247]:
```
final_model2 = grid_search.best_estimator_
final_pred2 = final_model2.predict(test_data)
final_mse2 = mean_squared_error(test_data_labels, final_pred2)
final_rmse2 = np.sqrt(final_mse2)
print("the root mean squared error for decision trees is: ", final_rmse2)
```

```
the root mean squared error for decision trees is:  60594.260464140665
```

[248]:
```
final_pred3 = lin_reg.predict(test_data)
final_mse3 = mean_squared_error(test_data_labels, final_pred3)
final_rmse3 = np.sqrt(final_mse3)
print("the root mean squared error for decision trees is: ", final_rmse3)
```

```
the root mean squared error for decision trees is:  67621.16608333463
```

[249]:
```
final_pred4 = dtree.predict(test_data)
final_mse4 = mean_squared_error(test_data_labels, final_pred4)
final_rmse4 = np.sqrt(final_mse4)
print("the root mean squared error for decision trees is: ", final_rmse4)
```

```
the root mean squared error for decision trees is:  108241.9312110502
```

[250]:
```python
joblib.dump(final_model2, "final_model_random_forest.pkl")
```

[250]:
```
['final_model_random_forest.pkl']
```