import urllib

In [1]:
```
import urllib
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
housing = pd.read_csv('C:/articles/datasets/ml-master/ml-master/machine_learning/datasets/housing/housing.csv')
```

In [3]:
```
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
```

In [4]:
```
housing.describe()
```

Out[4]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_h
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.000000	20
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.870671	206
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.899822	115
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	14
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.563400	119
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.534800	179
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.743250	264
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.000100	500

In [5]:
```
housing['total_bedrooms'].value_counts()
#Thus the below output shows the distribution of the total bedrooms in the different blocks of california
```

Out[5]:
```
280.0     55
331.0     51
345.0     50
393.0     49
343.0     49
          ..
2111.0     1
1852.0     1
1663.0     1
1652.0     1
2479.0     1
Name: total_bedrooms, Length: 1923, dtype: int64
```

In [6]:
```
a = housing['housing_median_age'].value_counts()
#Thus the below output shows the distribution of the total bedrooms in the different blocks of california
```

In [7]:
```
type(a)
```

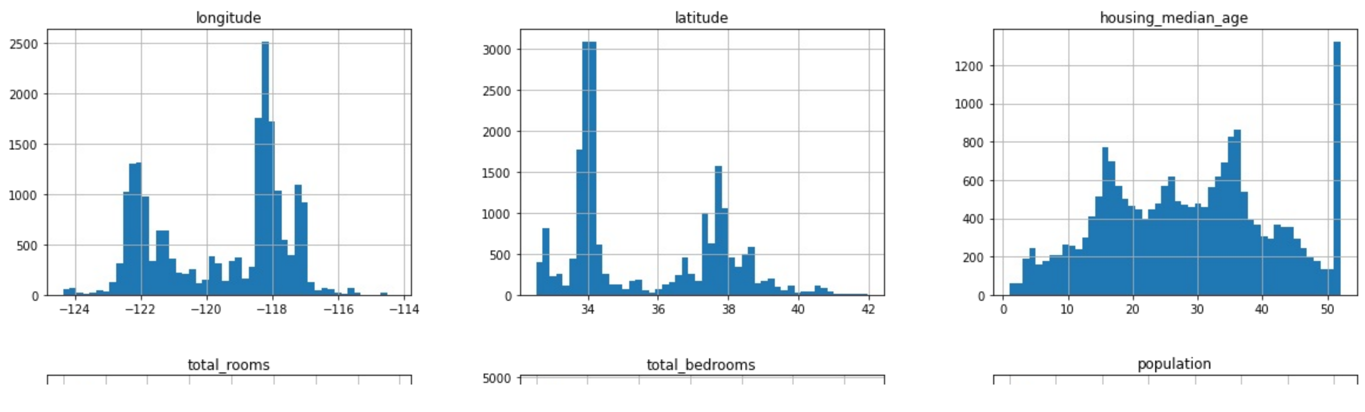
Out[7]:
```
pandas.core.series.Series
```

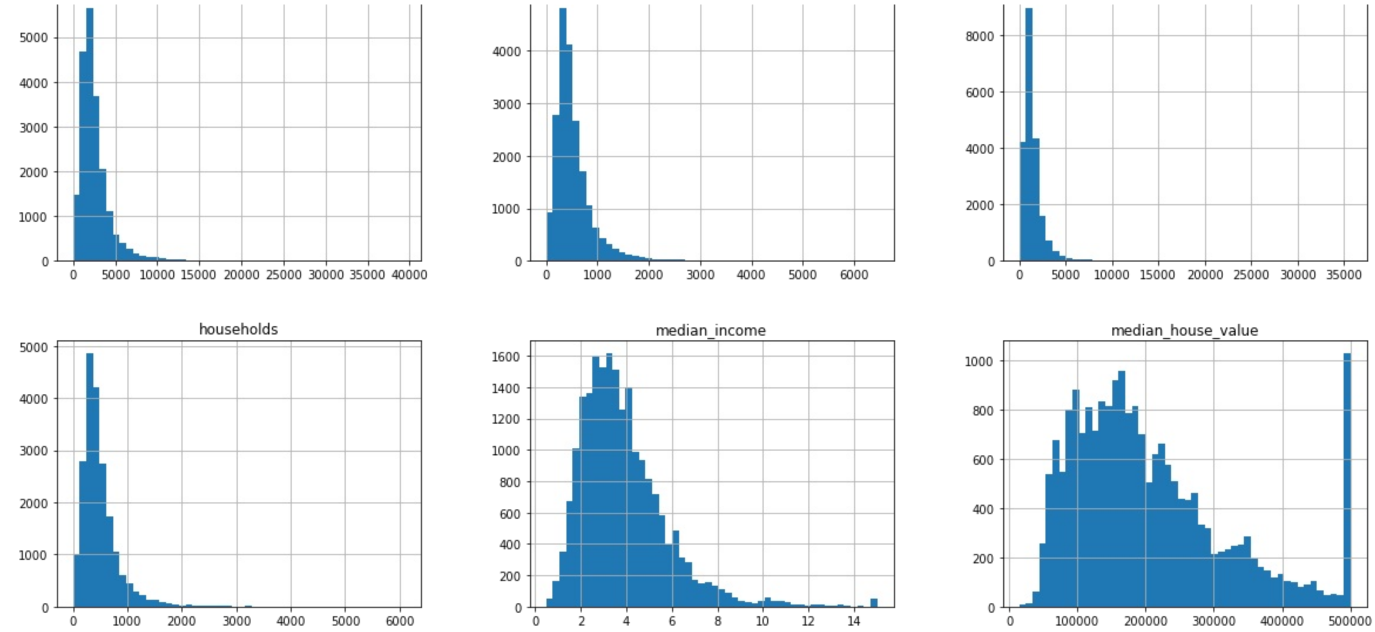
Out[7]: pandas.core.series.Series

In [8]:
```
a
```

Out[8]:
```
52.0    1273
36.0     862
35.0     824
16.0     771
17.0     698
34.0     689
26.0     619
33.0     615
18.0     570
25.0     566
32.0     565
37.0     537
15.0     512
19.0     502
27.0     488
24.0     478
30.0     476
28.0     471
20.0     465
29.0     461
31.0     458
23.0     448
21.0     446
14.0     412
22.0     399
38.0     394
39.0     369
42.0     368
44.0     356
43.0     353
40.0     304
13.0     302
41.0     296
45.0     294
10.0     264
11.0     254
46.0     245
5.0      244
12.0     238
8.0      206
9.0      205
47.0     198
4.0      191
48.0     177
7.0      175
6.0      160
50.0     136
49.0     134
3.0       62
2.0       58
51.0      48
1.0        4
Name: housing_median_age, dtype: int64
```

In [9]:
```
#plotting the cloumn of each data frame..
%matplotlib inline
import matplotlib.pyplot as plt
housing.hist(bins = 50, figsize =(20,15))
plt.show()
```





In [10]:
```
housing.head(5)
```

Out[10]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	N
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	N
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	N
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	N
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	N

In [11]:
```
housing.describe()
```

Out[11]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_h
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.000000	20
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.870671	206
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.899822	115
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	14
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.563400	119
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.534800	179
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.743250	264
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.000100	500

In [12]:
```
housing['median_income'].value_counts()
```

Out[12]:
```
3.1250     49
15.0001    49
2.8750     46
2.6250     44
4.1250     44
           ..
5.0476      1
4.6379      1
2.9402      1
6.0256      1
5.5647      1
Name: median_income, Length: 12928, dtype: int64
```

In [13]:
```
import random
random.random()
```

Out[13]: 0.25964032328767395

In [14]:
```
b = np.array([1,2,7,4,6])
c =b[:3]
```

In [15]:
```
c
```

Out[15]: array([1, 2, 7])

In [16]:
```
np.random.rand(2,3)
```

Out[16]:
```
array([[0.86431321, 0.23921777, 0.24903931],
       [0.8429882 , 0.96657521, 0.87542146]])
```

In [17]:
```
np.random.permutation(5)
```

Out[17]: array([4, 2, 3, 0, 1])

In [18]:
```
np.random.permutation(5)
```

Out[18]: array([3, 0, 1, 4, 2])

In [19]:
```
#splitting the dataset into training and test set
np.random.seed(42)
def split_train_test(data, test_ratio):
    shuffled_indices = np.random.permutation(len(data))
    test_set_size =int(len(data)*test_ratio)
    test_indices = shuffled_indices[:test_set_size]
    train_indices = shuffled_indices[test_set_size:]
    return data.iloc[train_indices] ,data.iloc[test_indices]
train_set , test_set = split_train_test(housing, 0.2)
print(len(train_set) , "train +" , len(test_set) , "test")
```

16512 train + 4128 test

In [20]:
```
train_set.head()
```

Out[20]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	oc
14196	-117.03	32.71	33.0	3126.0	627.0	2300.0	623.0	3.2596	103000.0	
8267	-118.16	33.77	49.0	3382.0	787.0	1314.0	756.0	3.8125	382100.0	
17445	-120.48	34.66	4.0	1897.0	331.0	915.0	336.0	4.1563	172600.0	
14265	-117.11	32.69	36.0	1421.0	367.0	1418.0	355.0	1.9425	93400.0	
2271	-119.80	36.78	43.0	2382.0	431.0	874.0	380.0	3.5542	96500.0	

In [21]:
```
test_set.head()
```

Out[21]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	oc
20046	-119.01	36.06	25.0	1505.0	NaN	1392.0	359.0	1.6812	47700.0	
3024	-119.46	35.14	30.0	2943.0	NaN	1565.0	584.0	2.5313	45800.0	
15663	-122.44	37.80	52.0	3830.0	NaN	1310.0	963.0	3.4801	500001.0	
20484	-118.72	34.28	17.0	3051.0	NaN	1705.0	495.0	5.7376	218600.0	
9814	-121.93	36.62	34.0	2351.0	NaN	1063.0	428.0	3.7250	278000.0	

In [22]:
```python
from sklearn.model_selection import train_test_split
```

In [23]:
```python
train_set , test_set = train_test_split(housing, test_size =0.2, random_state =42)
```

In [24]:
```python
print(len(train_set), 'train +' ,len(test_set), 'test')
```

```
16512 train + 4128 test
```

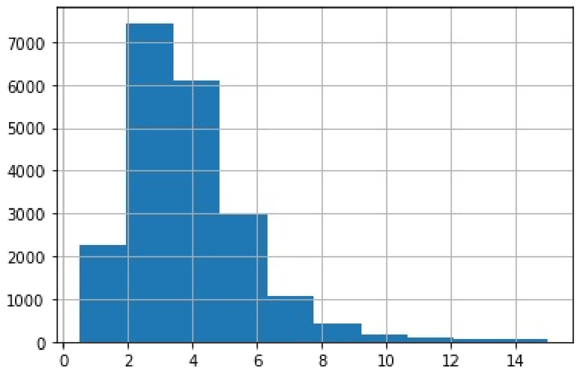
In [25]:
```python
test_set.head()
```

Out[25]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	oc
20046	-119.01	36.06	25.0	1505.0	NaN	1392.0	359.0	1.6812	47700.0	
3024	-119.46	35.14	30.0	2943.0	NaN	1565.0	584.0	2.5313	45800.0	
15663	-122.44	37.80	52.0	3830.0	NaN	1310.0	963.0	3.4801	500001.0	
20484	-118.72	34.28	17.0	3051.0	NaN	1705.0	495.0	5.7376	218600.0	
9814	-121.93	36.62	34.0	2351.0	NaN	1063.0	428.0	3.7250	278000.0	

In [26]:
```python
housing['median_income'].hist()
```

Out[26]: <AxesSubplot:>



**from the graph of the median income, it is evident that most of the population lies in the range 2 to 5. Thus we need to make our strata from this portion of the distribution.

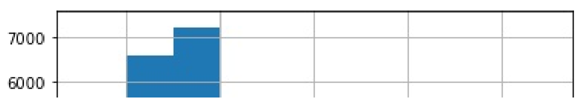
In [27]:
```python
housing['income_cat'] = np.ceil(housing['median_income']/1.5)
```

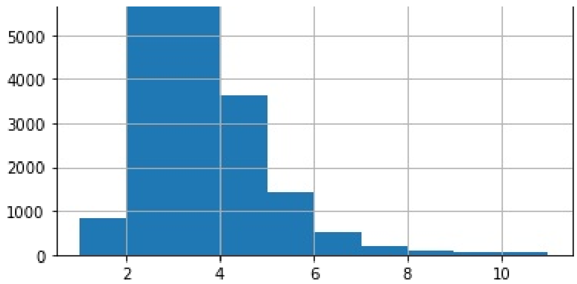
In [28]:
```python
housing['income_cat'].value_counts()
```

Out[28]:
```
3.0     7236
2.0     6581
4.0     3639
5.0     1423
1.0      822
6.0      532
7.0      189
8.0      105
9.0       50
11.0      49
10.0      14
Name: income_cat, dtype: int64
```

In [29]:
```python
housing['income_cat'].hist()
```

Out[29]: <AxesSubplot:>





In [30]:
```
housing['median_income'].value_counts()
```

Out[30]:
```
3.1250     49
15.0001    49
2.8750     46
2.6250     44
4.1250     44
           ..
5.0476      1
4.6379      1
2.9402      1
6.0256      1
5.5647      1
Name: median_income, Length: 12928, dtype: int64
```

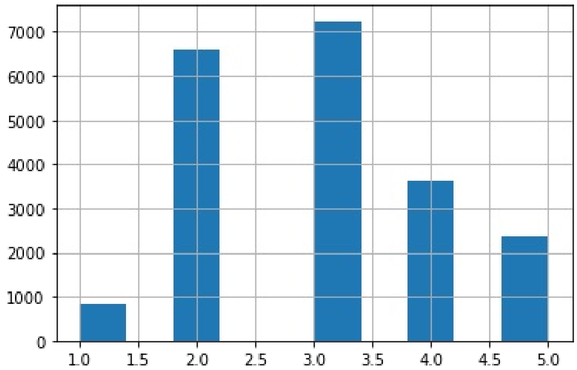
In [31]:
```
housing['income_cat'].where(housing['income_cat']<5 , 5.0, inplace = True)
```

In [32]:
```
housing['income_cat'].value_counts()
```

Out[32]:
```
3.0    7236
2.0    6581
4.0    3639
5.0    2362
1.0     822
Name: income_cat, dtype: int64
```

In [33]:
```
housing['income_cat'].hist()
```

Out[33]: <AxesSubplot:>



In [34]:
```
housing['income_cat'].describe()
```

Out[34]:
```
count    20640.000000
mean         3.006686
std          1.054618
min          1.000000
25%          2.000000
50%          3.000000
75%          4.000000
max          5.000000
Name: income_cat, dtype: float64
```

In [35]:

```python
housing['median_income'].describe()
```

Out[35]:

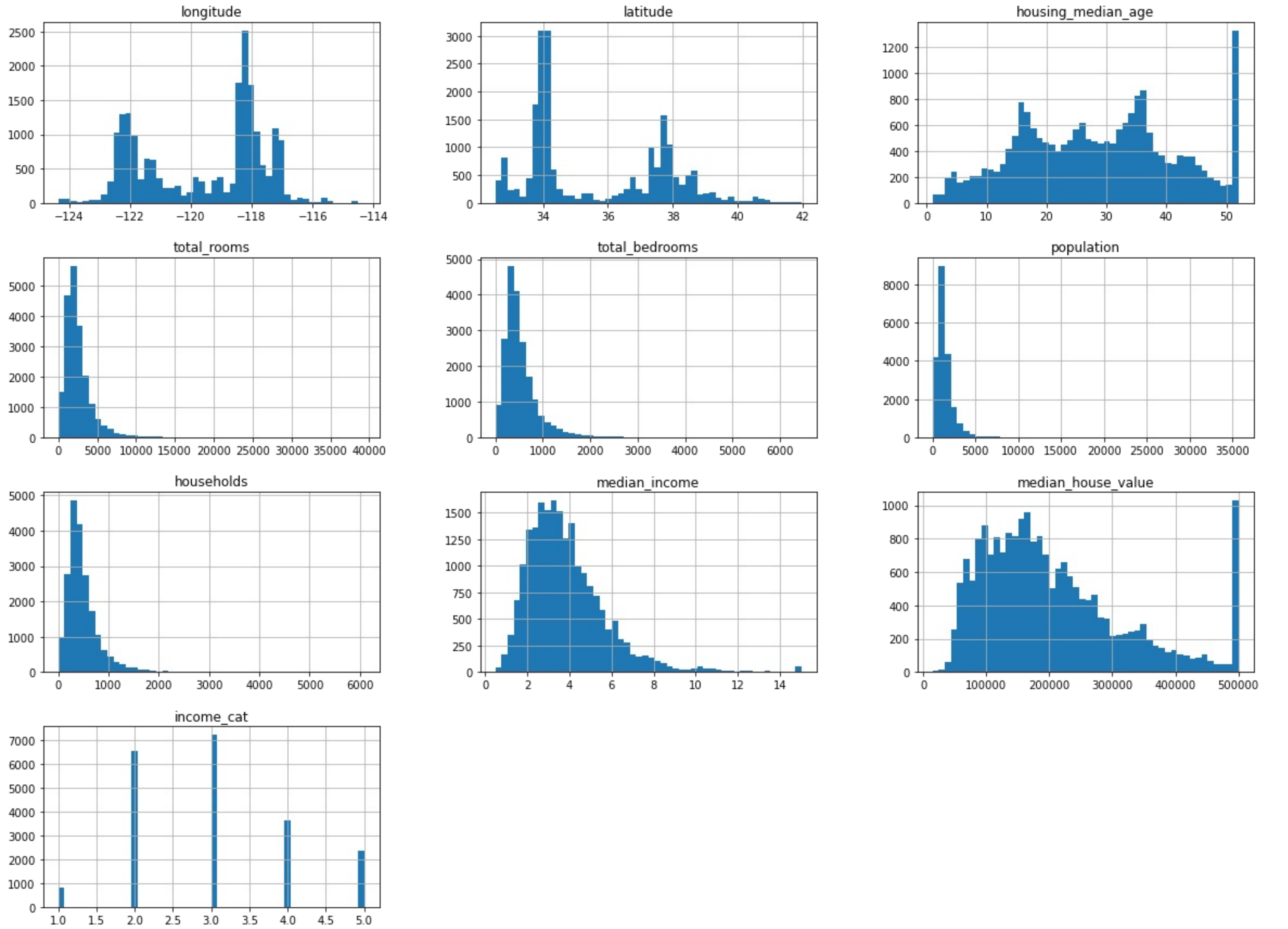
```
count    20640.000000
mean         3.870671
std          1.899822
min          0.499900
25%          2.563400
50%          3.534800
75%          4.743250
max         15.000100
Name: median_income, dtype: float64
```

In [36]:

```python
housing.hist(bins =50 ,figsize =(20,15))
```

Out[36]:

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

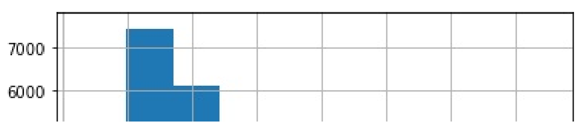


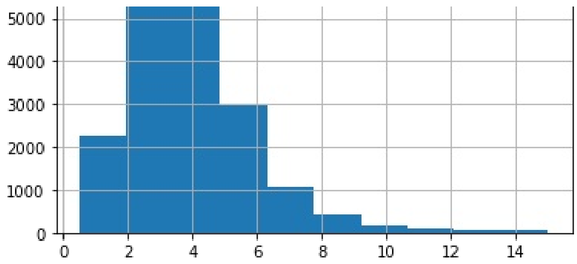
In [37]:

```python
housing['median_income'].hist()
```

Out[37]:

```
<AxesSubplot:>
```





In [38]:
```
housing['median_income'].value_counts()
```

Out[38]:
```
3.1250     49
15.0001    49
2.8750     46
2.6250     44
4.1250     44
           ..
5.0476      1
4.6379      1
2.9402      1
6.0256      1
5.5647      1
Name: median_income, Length: 12928, dtype: int64
```

In [39]:
```
housing['income_cat'].value_counts()
```

Out[39]:
```
3.0    7236
2.0    6581
4.0    3639
5.0    2362
1.0     822
Name: income_cat, dtype: int64
```

In [40]:
```
len(housing)
```

Out[40]:
```
20640
```

In [41]:
```
from sklearn.model_selection import StratifiedShuffleSplit
```

In [42]:
```
split= StratifiedShuffleSplit(n_splits=1, test_size =0.2,random_state=42)
```

In [43]:
```
split
```

Out[43]:
```
StratifiedShuffleSplit(n_splits=1, random_state=42, test_size=0.2,
            train_size=None)
```

In [44]:
```
for train_index , test_index in split.split(housing, housing['income_cat']):
    strat_train_set = housing.loc[train_index]
    strat_test_set = housing.loc[test_index]
```

In [45]:
```
strat_train_set.head()
```

Out[45]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	oc
17606	-121.89	37.29	38.0	1568.0	351.0	710.0	339.0	2.7042	286600.0	
18632	-121.93	37.05	14.0	679.0	108.0	306.0	113.0	6.4214	340600.0	
14650	-117.20	32.77	31.0	1952.0	471.0	936.0	462.0	2.8621	196900.0	
3230	-119.61	36.31	25.0	1847.0	371.0	1460.0	353.0	1.8839	46300.0	
3555	-118.59	34.23	17.0	6592.0	1525.0	4459.0	1463.0	3.0347	254500.0	

In [46]:
```
strat_train_set['income_cat'].value_counts()
```

Out[46]:
```
3.0    5789
2.0    5265
4.0    2911
5.0    1889
1.0     658
Name: income_cat, dtype: int64
```

In [47]:
```
strat_test_set['income_cat'].value_counts()
```

Out[47]:
```
3.0    1447
2.0    1316
4.0     728
5.0     473
1.0     164
Name: income_cat, dtype: int64
```

In [48]:
```
strat_test_set.head()
```

Out[48]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	oc
5241	-118.39	34.12	29.0	6447.0	1012.0	2184.0	960.0	8.2816	500001.0	
10970	-117.86	33.77	39.0	4159.0	655.0	1669.0	651.0	4.6111	240300.0	
20351	-119.05	34.21	27.0	4357.0	926.0	2110.0	876.0	3.0119	218200.0	
6568	-118.15	34.20	52.0	1786.0	306.0	1018.0	322.0	4.1518	182100.0	
13285	-117.68	34.07	32.0	1775.0	314.0	1067.0	302.0	4.0375	121300.0	

In [49]:
```
strat_test_set.head()
```

Out[49]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	oc
5241	-118.39	34.12	29.0	6447.0	1012.0	2184.0	960.0	8.2816	500001.0	
10970	-117.86	33.77	39.0	4159.0	655.0	1669.0	651.0	4.6111	240300.0	
20351	-119.05	34.21	27.0	4357.0	926.0	2110.0	876.0	3.0119	218200.0	
6568	-118.15	34.20	52.0	1786.0	306.0	1018.0	322.0	4.1518	182100.0	
13285	-117.68	34.07	32.0	1775.0	314.0	1067.0	302.0	4.0375	121300.0	

In [50]:
```
# Income category proportion in test set generated with stratified sampling
strat_test_set["income_cat"].value_counts() / len(strat_test_set)
```

Out[50]:
```
3.0    0.350533
2.0    0.318798
4.0    0.176357
5.0    0.114583
1.0    0.039729
Name: income_cat, dtype: float64
```

**This shows that 35% of the people have income upto 3000 dollars and 31% of them have income upto 2000 dollars

In [51]:
```
#this piece of code removes the extra column we created before
for set_ in (strat_train_set , strat_test_set):
    set_.drop('income_cat' , axis =1 , inplace=True)
```

In [52]:
```
strat_test_set.head()
```

Out[52]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	oc
5241	-118.39	34.12	29.0	6447.0	1012.0	2184.0	960.0	8.2816	500001.0	

10970	-117.86	33.77	39.0	4159.0	655.0	1669.0	651.0	4.6111	240300.0
20351	-119.05	34.21	27.0	4357.0	926.0	2110.0	876.0	3.0119	218200.0
6568	-118.15	34.20	52.0	1786.0	306.0	1018.0	322.0	4.1518	182100.0
13285	-117.68	34.07	32.0	1775.0	314.0	1067.0	302.0	4.0375	121300.0

In [53]:

```
strat_train_set.head()
```

Out[53]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	oc
17606	-121.89	37.29	38.0	1568.0	351.0	710.0	339.0	2.7042	286600.0	
18632	-121.93	37.05	14.0	679.0	108.0	306.0	113.0	6.4214	340600.0	
14650	-117.20	32.77	31.0	1952.0	471.0	936.0	462.0	2.8621	196900.0	
3230	-119.61	36.31	25.0	1847.0	371.0	1460.0	353.0	1.8839	46300.0	
3555	-118.59	34.23	17.0	6592.0	1525.0	4459.0	1463.0	3.0347	254500.0	

In [54]:

```
traindata = strat_train_set.copy()
```

In [55]:

```
traindata
```

Out[55]:

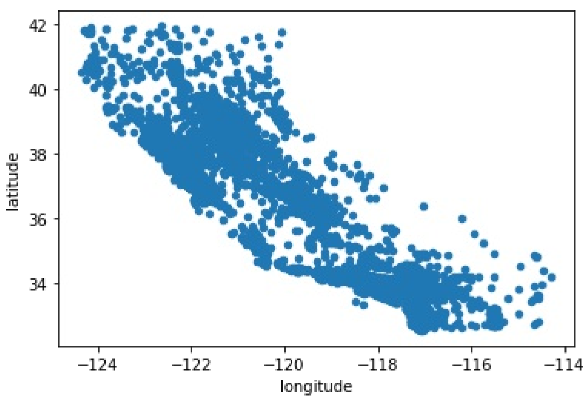
	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	oc
17606	-121.89	37.29	38.0	1568.0	351.0	710.0	339.0	2.7042	286600.0	
18632	-121.93	37.05	14.0	679.0	108.0	306.0	113.0	6.4214	340600.0	
14650	-117.20	32.77	31.0	1952.0	471.0	936.0	462.0	2.8621	196900.0	
3230	-119.61	36.31	25.0	1847.0	371.0	1460.0	353.0	1.8839	46300.0	
3555	-118.59	34.23	17.0	6592.0	1525.0	4459.0	1463.0	3.0347	254500.0	
...	...	...	...	...	...	...	...	...	...	
6563	-118.13	34.20	46.0	1271.0	236.0	573.0	210.0	4.9312	240200.0	
12053	-117.56	33.88	40.0	1196.0	294.0	1052.0	258.0	2.0682	113000.0	
13908	-116.40	34.09	9.0	4855.0	872.0	2098.0	765.0	3.2723	97800.0	
11159	-118.01	33.82	31.0	1960.0	380.0	1356.0	356.0	4.0625	225900.0	
15775	-122.45	37.77	52.0	3095.0	682.0	1269.0	639.0	3.5750	500001.0	

16512 rows × 10 columns

In [56]:

```
#visualisation of the data
traindata.plot(kind='scatter', x='longitude', y='latitude')
```

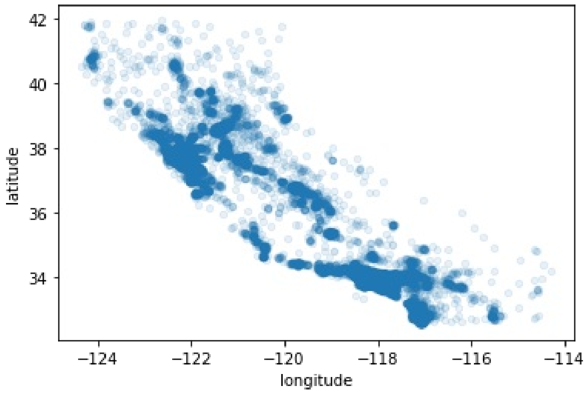
Out[56]: <AxesSubplot:xlabel='longitude', ylabel='latitude'>



In [57]:

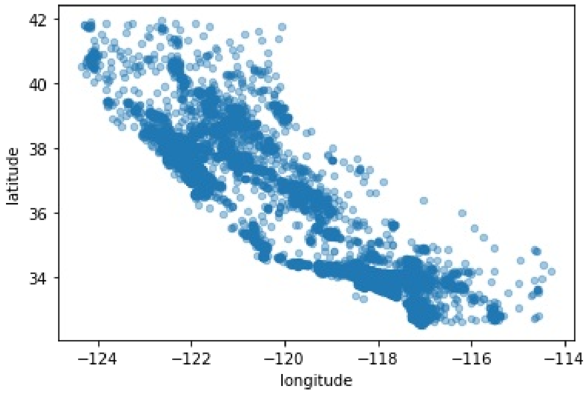
```
#visualisation of the data
traindata.plot(kind='scatter', x='longitude', y='latitude', alpha =0.1)
```

Out[57]: <AxesSubplot:xlabel='longitude', ylabel='latitude'>



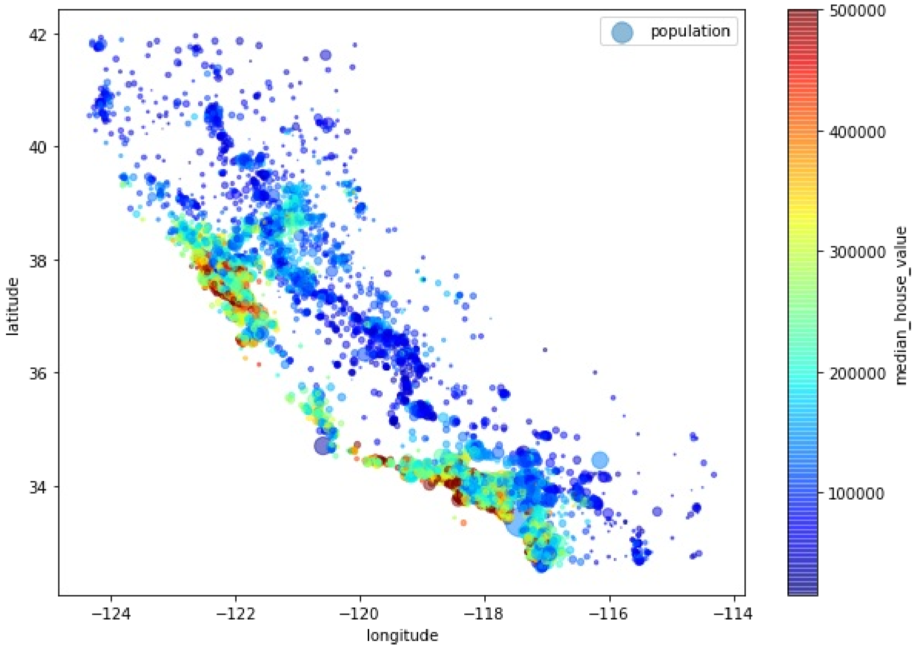
In [58]:
```
#visualisation of the data
traindata.plot(kind='scatter', x='longitude', y='latitude', alpha=.4)
```

Out[58]: <AxesSubplot:xlabel='longitude', ylabel='latitude'>



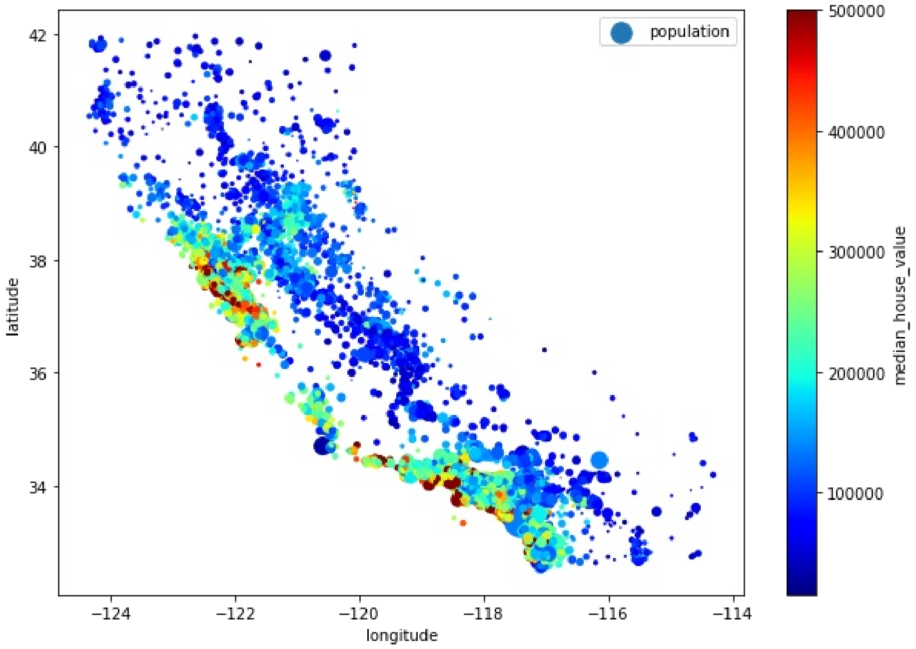
In [59]:
```
traindata.plot(kind ='scatter', x='longitude', y='latitude', alpha =0.5, s =traindata['population']/100, label='p
plt.legend()
```

Out[59]: <matplotlib.legend.Legend at 0x1a68aface20>



In [60]:
```
traindata.plot(kind ='scatter', x='longitude', y='latitude',  s =traindata['population']/100, label='population',
plt.legend()
```

Out[60]: <matplotlib.legend.Legend at 0x1a68ae92ca0>



In [61]:
```
#finding correlation between the diffrent variables at play
corr_matrix = traindata.corr()
corr_matrix
```

Out[61]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_
longitude	1.000000	-0.924478	-0.105848	0.048871	0.076598	0.108030	0.063070	-0.019583	
latitude	-0.924478	1.000000	0.005766	-0.039184	-0.072419	-0.115222	-0.077647	-0.075205	
housing_median_age	-0.105848	0.005766	1.000000	-0.364509	-0.325047	-0.298710	-0.306428	-0.111360	
total_rooms	0.048871	-0.039184	-0.364509	1.000000	0.929379	0.855109	0.918392	0.200087	
total_bedrooms	0.076598	-0.072419	-0.325047	0.929379	1.000000	0.876320	0.980170	-0.009740	
population	0.108030	-0.115222	-0.298710	0.855109	0.876320	1.000000	0.904637	0.002380	
households	0.063070	-0.077647	-0.306428	0.918392	0.980170	0.904637	1.000000	0.010781	
median_income	-0.019583	-0.075205	-0.111360	0.200087	-0.009740	0.002380	0.010781	1.000000	
median_house_value	-0.047432	-0.142724	0.114110	0.135097	0.047689	-0.026920	0.064506	0.687160	

In [62]:
```
corr_matrix['median_house_value']
```

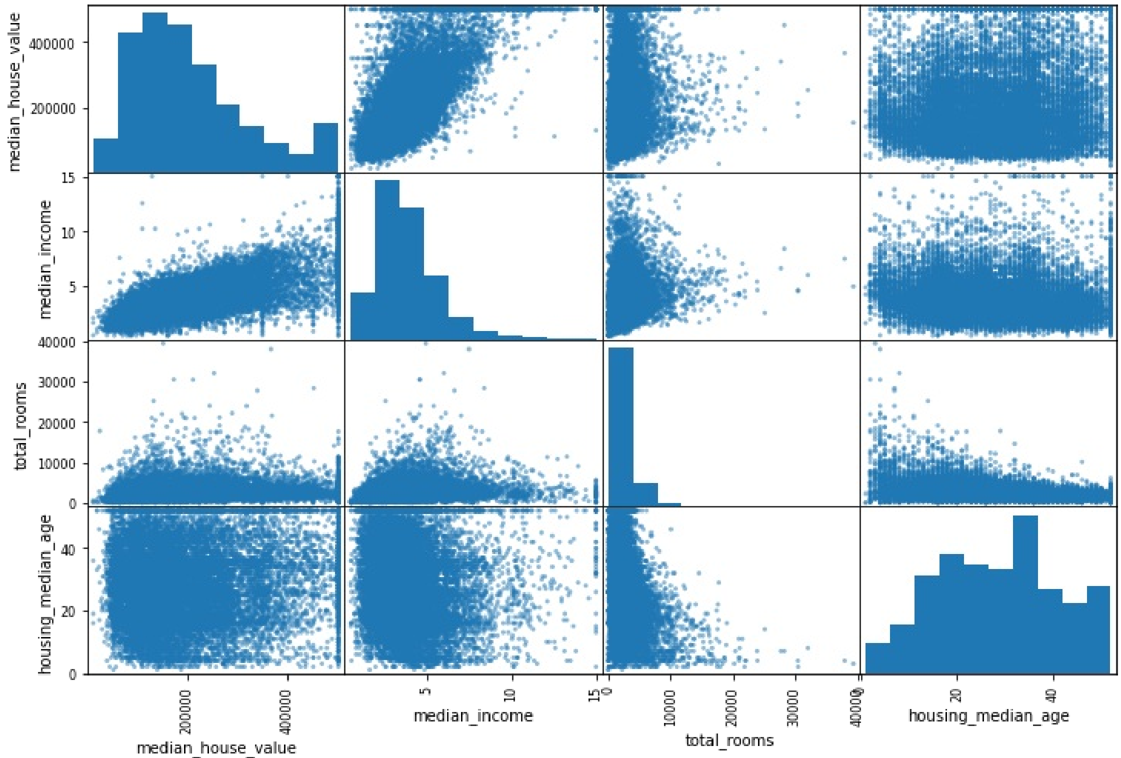
Out[62]:
```
longitude            -0.047432
latitude             -0.142724
housing_median_age    0.114110
total_rooms           0.135097
total_bedrooms        0.047689
population           -0.026920
households            0.064506
median_income         0.687160
median_house_value    1.000000
Name: median_house_value, dtype: float64
```

In [63]:
```
#to sort the corr_matrix in descending order
corr_matrix['median_house_value'].sort_values(ascending =False)
```

Out[63]:
```
median_house_value    1.000000
median_income         0.687160
total_rooms           0.135097
housing_median_age    0.114110
households            0.064506
total_bedrooms        0.047689
population           -0.026920
longitude            -0.047432
latitude             -0.142724
Name: median_house_value, dtype: float64
```

In [64]:
```python
#visualisation
from pandas.plotting import scatter_matrix
attributes =['median_house_value','median_income','total_rooms','housing_median_age']
scatter_matrix(traindata[attributes],figsize=(12,8))
```

Out[64]:
```
array([[<AxesSubplot:xlabel='median_house_value', ylabel='median_house_value'>,
        <AxesSubplot:xlabel='median_income', ylabel='median_house_value'>,
        <AxesSubplot:xlabel='total_rooms', ylabel='median_house_value'>,
        <AxesSubplot:xlabel='housing_median_age', ylabel='median_house_value'>],
       [<AxesSubplot:xlabel='median_house_value', ylabel='median_income'>,
        <AxesSubplot:xlabel='median_income', ylabel='median_income'>,
        <AxesSubplot:xlabel='total_rooms', ylabel='median_income'>,
        <AxesSubplot:xlabel='housing_median_age', ylabel='median_income'>],
       [<AxesSubplot:xlabel='median_house_value', ylabel='total_rooms'>,
        <AxesSubplot:xlabel='median_income', ylabel='total_rooms'>,
        <AxesSubplot:xlabel='total_rooms', ylabel='total_rooms'>,
        <AxesSubplot:xlabel='housing_median_age', ylabel='total_rooms'>],
       [<AxesSubplot:xlabel='median_house_value', ylabel='housing_median_age'>,
        <AxesSubplot:xlabel='median_income', ylabel='housing_median_age'>,
        <AxesSubplot:xlabel='total_rooms', ylabel='housing_median_age'>,
        <AxesSubplot:xlabel='housing_median_age', ylabel='housing_median_age'>]],
      dtype=object)
```



In [65]:
```python
traindata[['median_income', 'median_house_value']].head()
```

Out[65]:

	median_income	median_house_value
17606	2.7042	286600.0
18632	6.4214	340600.0
14650	2.8621	196900.0
3230	1.8839	46300.0
3555	3.0347	254500.0

In [66]:
```python
traindata.describe()
```

Out[66]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_h
count	16512.000000	16512.000000	16512.000000	16512.000000	16354.000000	16512.000000	16512.000000	16512.000000	16
mean	-119.575834	35.639577	28.653101	2622.728319	534.973890	1419.790819	497.060380	3.875589	206
std	2.001860	2.138058	12.574726	2138.458419	412.699041	1115.686241	375.720845	1.904950	115
min	-124.350000	32.540000	1.000000	6.000000	2.000000	3.000000	2.000000	0.499900	14

25%	-121.800000	33.940000	18.000000	1443.000000	295.000000	784.000000	279.000000	2.566775	119
50%	-118.510000	34.260000	29.000000	2119.500000	433.000000	1164.000000	408.000000	3.540900	179
75%	-118.010000	37.720000	37.000000	3141.000000	644.000000	1719.250000	602.000000	4.744475	263
max	-114.310000	41.950000	52.000000	39320.000000	6210.000000	35682.000000	5358.000000	15.000100	500

In [67]:

```
traindata['median_income'].value_counts()
```

Out[67]:

```
15.0001    38
2.8750     38
2.6250     37
3.1250     37
3.3750     33
           ..
1.6410      1
6.0074      1
3.8421      1
2.6941      1
5.5647      1
Name: median_income, Length: 10905, dtype: int64
```

In [68]:

```
traindata.describe()
```

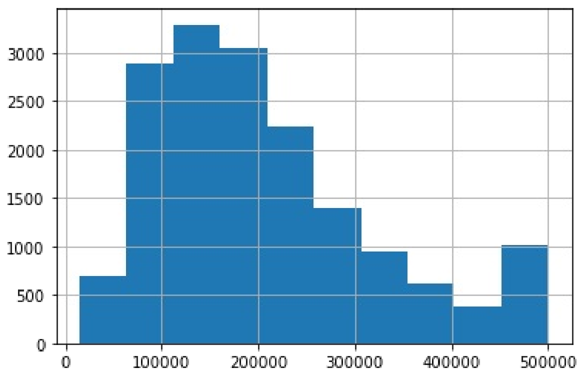
Out[68]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_h
count	16512.000000	16512.000000	16512.000000	16512.000000	16354.000000	16512.000000	16512.000000	16512.000000	16
mean	-119.575834	35.639577	28.653101	2622.728319	534.973890	1419.790819	497.060380	3.875589	206
std	2.001860	2.138058	12.574726	2138.458419	412.699041	1115.686241	375.720845	1.904950	115
min	-124.350000	32.540000	1.000000	6.000000	2.000000	3.000000	2.000000	0.499900	14
25%	-121.800000	33.940000	18.000000	1443.000000	295.000000	784.000000	279.000000	2.566775	119
50%	-118.510000	34.260000	29.000000	2119.500000	433.000000	1164.000000	408.000000	3.540900	179
75%	-118.010000	37.720000	37.000000	3141.000000	644.000000	1719.250000	602.000000	4.744475	263
max	-114.310000	41.950000	52.000000	39320.000000	6210.000000	35682.000000	5358.000000	15.000100	500

In [69]:

```
traindata['median_house_value'].hist()
```

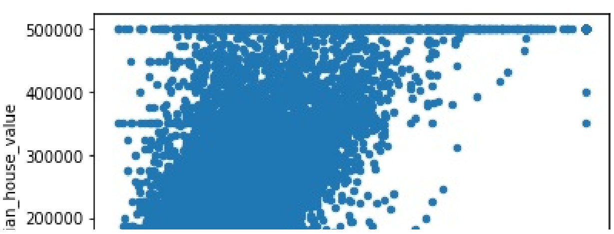
Out[69]: <AxesSubplot:>

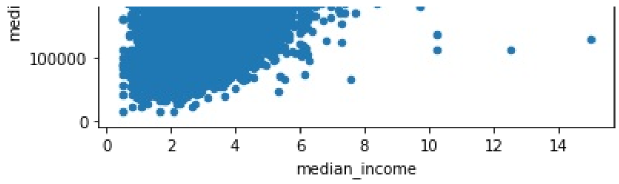


In [70]:

```
traindata.plot(kind='scatter', x='median_income', y='median_house_value')
```

Out[70]: <AxesSubplot:xlabel='median_income', ylabel='median_house_value'>





***The graph above shows the following observation:

1) There exist a positive correlation between the median income and the median house values.

2) The median house value above 50K has been capped at 50K.

In [71]:
```
#creating new attributes
traindata['rooms_per_households']= traindata['total_rooms']/traindata['households']
traindata['bedrooms_per_room']= traindata['total_bedrooms']/traindata['total_rooms']
traindata['population_per_houselholds']= traindata['population']/traindata['households']
```

In [72]:
```
traindata
```

Out[72]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	oc
17606	-121.89	37.29	38.0	1568.0	351.0	710.0	339.0	2.7042	286600.0	
18632	-121.93	37.05	14.0	679.0	108.0	306.0	113.0	6.4214	340600.0	
14650	-117.20	32.77	31.0	1952.0	471.0	936.0	462.0	2.8621	196900.0	
3230	-119.61	36.31	25.0	1847.0	371.0	1460.0	353.0	1.8839	46300.0	
3555	-118.59	34.23	17.0	6592.0	1525.0	4459.0	1463.0	3.0347	254500.0	
...	...	...	...	...	...	...	...	...	...	
6563	-118.13	34.20	46.0	1271.0	236.0	573.0	210.0	4.9312	240200.0	
12053	-117.56	33.88	40.0	1196.0	294.0	1052.0	258.0	2.0682	113000.0	
13908	-116.40	34.09	9.0	4855.0	872.0	2098.0	765.0	3.2723	97800.0	
11159	-118.01	33.82	31.0	1960.0	380.0	1356.0	356.0	4.0625	225900.0	
15775	-122.45	37.77	52.0	3095.0	682.0	1269.0	639.0	3.5750	500001.0	

16512 rows × 13 columns

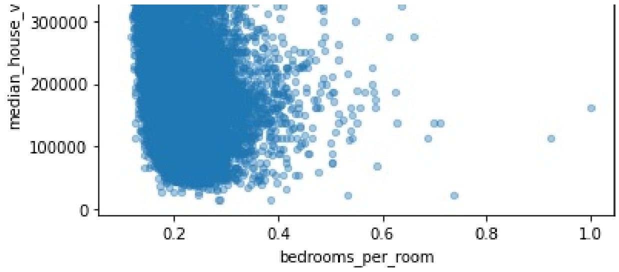
In [73]:
```
#finding correlation between the diffrent variables at play
new_corr_matrix = traindata.corr()
```

In [74]:
```
#to sort the corr_matrix in descending order
new_corr_matrix['median_house_value'].sort_values(ascending =False)
```

Out[74]:
```
median_house_value            1.000000
median_income                 0.687160
rooms_per_households          0.146285
total_rooms                   0.135097
housing_median_age            0.114110
households                    0.064506
total_bedrooms                0.047689
population_per_houselholds   -0.021985
population                   -0.026920
longitude                    -0.047432
latitude                     -0.142724
bedrooms_per_room            -0.259984
Name: median_house_value, dtype: float64
```

In [75]:
```
traindata.plot(kind='scatter', x='bedrooms_per_room', y ='median_house_value', alpha=.4)
```

Out[75]: <AxesSubplot:xlabel='bedrooms_per_room', ylabel='median_house_value'>



In [76]:
```
# Let’s revert to a clean training set

traindata = strat_train_set.drop("median_house_value", axis=1) # drop labels for training set
traindata_labels = strat_train_set["median_house_value"].copy()

# Note drop() creates a copy of the data and does not affect strat_train_set
```

In [77]:
```
traindata.head(5)
```

Out[77]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity
17606	-121.89	37.29	38.0	1568.0	351.0	710.0	339.0	2.7042	<1H OCEAN
18632	-121.93	37.05	14.0	679.0	108.0	306.0	113.0	6.4214	<1H OCEAN
14650	-117.20	32.77	31.0	1952.0	471.0	936.0	462.0	2.8621	NEAR OCEAN
3230	-119.61	36.31	25.0	1847.0	371.0	1460.0	353.0	1.8839	INLAND
3555	-118.59	34.23	17.0	6592.0	1525.0	4459.0	1463.0	3.0347	<1H OCEAN

In [78]:
```
traindata_labels.head()
```

Out[78]:
```
17606    286600.0
18632    340600.0
14650    196900.0
3230      46300.0
3555     254500.0
Name: median_house_value, dtype: float64
```

In [79]:
```
traindata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16512 entries, 17606 to 15775
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      16354 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
```

In [80]:
```
housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 11 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 
 10  income_cat          20640 non-null  float64
dtypes: float64(10), object(1)
memory usage: 1.7+ MB
```

In [81]:

```
traindata['total_bedrooms'].isna()
```

Out[81]:

```
17606    False
18632    False
14650    False
3230     False
3555     False
         ...  
6563     False
12053    False
13908    False
11159    False
15775    False
Name: total_bedrooms, Length: 16512, dtype: bool
```

In [82]:

```
isn = traindata.isna()
isn.any(axis=1)
```

Out[82]:

```
17606    False
18632    False
14650    False
3230     False
3555     False
         ...  
6563     False
12053    False
13908    False
11159    False
15775    False
Length: 16512, dtype: bool
```

In [83]:

```
type(isn)
```

Out[83]:

```
pandas.core.frame.DataFrame
```

In [84]:

```
sample_incomplete_rows = traindata[traindata.isna().any(axis =1)]
sample_incomplete_rows
```

Out[84]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity
4629	-118.30	34.07	18.0	3759.0	NaN	3296.0	1462.0	2.2708	<1H OCEAN
6068	-117.86	34.01	16.0	4632.0	NaN	3038.0	727.0	5.1762	<1H OCEAN
17923	-121.97	37.35	30.0	1955.0	NaN	999.0	386.0	4.6328	<1H OCEAN
13656	-117.30	34.05	6.0	2155.0	NaN	1039.0	391.0	1.6675	INLAND
19252	-122.79	38.48	7.0	6837.0	NaN	3468.0	1405.0	3.1662	<1H OCEAN
...	...	...	...	...	...	...	...	...	...
3376	-118.28	34.25	29.0	2559.0	NaN	1886.0	769.0	2.6036	<1H OCEAN
4691	-118.37	34.07	50.0	2519.0	NaN	1117.0	516.0	4.3667	<1H OCEAN
6052	-117.76	34.04	34.0	1914.0	NaN	1564.0	328.0	2.8347	INLAND
17198	-119.75	34.45	6.0	2864.0	NaN	1404.0	603.0	5.5073	NEAR OCEAN
4738	-118.38	34.05	49.0	702.0	NaN	458.0	187.0	4.8958	<1H OCEAN

158 rows × 9 columns

In [85]:

In [85]:
```
sample_incomplete_rows.dropna(subset=['total_bedrooms'])
```

Out[85]:

longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity

In [86]:
```
## Or let's drop the column of the missing values
sample_incomplete_rows.drop('total_bedrooms' ,axis =1)
```

Out[86]:

	longitude	latitude	housing_median_age	total_rooms	population	households	median_income	ocean_proximity
4629	-118.30	34.07	18.0	3759.0	3296.0	1462.0	2.2708	<1H OCEAN
6068	-117.86	34.01	16.0	4632.0	3038.0	727.0	5.1762	<1H OCEAN
17923	-121.97	37.35	30.0	1955.0	999.0	386.0	4.6328	<1H OCEAN
13656	-117.30	34.05	6.0	2155.0	1039.0	391.0	1.6675	INLAND
19252	-122.79	38.48	7.0	6837.0	3468.0	1405.0	3.1662	<1H OCEAN
...	...	...	...	...	...	...	...	...
3376	-118.28	34.25	29.0	2559.0	1886.0	769.0	2.6036	<1H OCEAN
4691	-118.37	34.07	50.0	2519.0	1117.0	516.0	4.3667	<1H OCEAN
6052	-117.76	34.04	34.0	1914.0	1564.0	328.0	2.8347	INLAND
17198	-119.75	34.45	6.0	2864.0	1404.0	603.0	5.5073	NEAR OCEAN
4738	-118.38	34.05	49.0	702.0	458.0	187.0	4.8958	<1H OCEAN

158 rows × 8 columns

In [87]:
```
median = traindata['total_bedrooms'].median()
median
```

Out[87]: 433.0

In [88]:
```
sample_incomplete_rows['total_bedrooms'].fillna(median, inplace=True)
```

```
C:\Users\NIYAZ AHMED\anaconda3\lib\site-packages\pandas\core\series.py:4463: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  return super().fillna(
```

In [89]:
```
sample_incomplete_rows
```

Out[89]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity
4629	-118.30	34.07	18.0	3759.0	433.0	3296.0	1462.0	2.2708	<1H OCEAN
6068	-117.86	34.01	16.0	4632.0	433.0	3038.0	727.0	5.1762	<1H OCEAN
17923	-121.97	37.35	30.0	1955.0	433.0	999.0	386.0	4.6328	<1H OCEAN
13656	-117.30	34.05	6.0	2155.0	433.0	1039.0	391.0	1.6675	INLAND
19252	-122.79	38.48	7.0	6837.0	433.0	3468.0	1405.0	3.1662	<1H OCEAN
...	...	...	...	...	...	...	...	...	...
3376	-118.28	34.25	29.0	2559.0	433.0	1886.0	769.0	2.6036	<1H OCEAN
4691	-118.37	34.07	50.0	2519.0	433.0	1117.0	516.0	4.3667	<1H OCEAN
6052	-117.76	34.04	34.0	1914.0	433.0	1564.0	328.0	2.8347	INLAND
17198	-119.75	34.45	6.0	2864.0	433.0	1404.0	603.0	5.5073	NEAR OCEAN
4738	-118.38	34.05	49.0	702.0	433.0	458.0	187.0	4.8958	<1H OCEAN

158 rows × 9 columns

In [90]:
```
traindata.head()
```

Out[90]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity
17606	-121.89	37.29	38.0	1568.0	351.0	710.0	339.0	2.7042	<1H OCEAN
18632	-121.93	37.05	14.0	679.0	108.0	306.0	113.0	6.4214	<1H OCEAN
14650	-117.20	32.77	31.0	1952.0	471.0	936.0	462.0	2.8621	NEAR OCEAN
3230	-119.61	36.31	25.0	1847.0	371.0	1460.0	353.0	1.8839	INLAND
3555	-118.59	34.23	17.0	6592.0	1525.0	4459.0	1463.0	3.0347	<1H OCEAN

In [91]:

```
sample_incomplete_rows.iloc[1]
```

Out[91]:

```
longitude              -117.86
latitude                 34.01
housing_median_age        16.0
total_rooms             4632.0
total_bedrooms           433.0
population              3038.0
households               727.0
median_income           5.1762
ocean_proximity      <1H OCEAN
Name: 6068, dtype: object
```

In [92]:

```
traindata.iloc[4629]
```

Out[92]:

```
longitude              -119.81
latitude                 36.73
housing_median_age        47.0
total_rooms             1314.0
total_bedrooms           416.0
population              1155.0
households               326.0
median_income            1.372
ocean_proximity         INLAND
Name: 1987, dtype: object
```

In [93]:

```
# Let's use Scikit-Learn Imputer class to fill missing values

from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy='median')
```

In [94]:

```
traindata_num = traindata.drop('ocean_proximity', axis =1)
```

In [95]:

```
traindata_num
```

Out[95]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
17606	-121.89	37.29	38.0	1568.0	351.0	710.0	339.0	2.7042
18632	-121.93	37.05	14.0	679.0	108.0	306.0	113.0	6.4214
14650	-117.20	32.77	31.0	1952.0	471.0	936.0	462.0	2.8621
3230	-119.61	36.31	25.0	1847.0	371.0	1460.0	353.0	1.8839
3555	-118.59	34.23	17.0	6592.0	1525.0	4459.0	1463.0	3.0347
...	...	...	...	...	...	...	...	...
6563	-118.13	34.20	46.0	1271.0	236.0	573.0	210.0	4.9312
12053	-117.56	33.88	40.0	1196.0	294.0	1052.0	258.0	2.0682
13908	-116.40	34.09	9.0	4855.0	872.0	2098.0	765.0	3.2723
11159	-118.01	33.82	31.0	1960.0	380.0	1356.0	356.0	4.0625
15775	-122.45	37.77	52.0	3095.0	682.0	1269.0	639.0	3.5750

16512 rows × 8 columns

In [96]:

```
traindata_num[traindata_num.isna().any(axis = 1)].head()
```

Out[96]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income

4629	-118.30	34.07	18.0	3759.0	NaN	3296.0	1462.0	2.2708
6068	-117.86	34.01	16.0	4632.0	NaN	3038.0	727.0	5.1762
17923	-121.97	37.35	30.0	1955.0	NaN	999.0	386.0	4.6328
13656	-117.30	34.05	6.0	2155.0	NaN	1039.0	391.0	1.6675
19252	-122.79	38.48	7.0	6837.0	NaN	3468.0	1405.0	3.1662

In [97]:

```
traindata_num.head()
```

Out[97]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
17606	-121.89	37.29	38.0	1568.0	351.0	710.0	339.0	2.7042
18632	-121.93	37.05	14.0	679.0	108.0	306.0	113.0	6.4214
14650	-117.20	32.77	31.0	1952.0	471.0	936.0	462.0	2.8621
3230	-119.61	36.31	25.0	1847.0	371.0	1460.0	353.0	1.8839
3555	-118.59	34.23	17.0	6592.0	1525.0	4459.0	1463.0	3.0347

In [98]:

```
# creating an object of the SimpleImputer class
imputer = SimpleImputer(strategy = 'median')
```

In [99]:

```
imputer.fit(traindata_num)
```

Out[99]:

```
SimpleImputer(strategy='median')
```

In [100…

```
traindata.median()
```

Out[100…

```
longitude              -118.5100
latitude                 34.2600
housing_median_age       29.0000
total_rooms            2119.5000
total_bedrooms          433.0000
population             1164.0000
households              408.0000
median_income             3.5409
dtype: float64
```

In [101…

```
imputer.statistics_
```

Out[101…

```
array([-118.51  ,   34.26  ,   29.    , 2119.5   ,  433.    , 1164.    ,
        408.    ,    3.5409])
```

In [102…

```
X = imputer.transform(traindata_num)
```

In [103…

```
X
```

Out[103…

```
array([[-121.89  ,   37.29  ,   38.    , ...,  710.    ,  339.    ,
           2.7042],
       [-121.93  ,   37.05  ,   14.    , ...,  306.    ,  113.    ,
           6.4214],
       [-117.2   ,   32.77  ,   31.    , ...,  936.    ,  462.    ,
           2.8621],
       ...,
       [-116.4   ,   34.09  ,    9.    , ..., 2098.    ,  765.    ,
           3.2723],
       [-118.01  ,   33.82  ,   31.    , ..., 1356.    ,  356.    ,
           4.0625],
       [-122.45  ,   37.77  ,   52.    , ..., 1269.    ,  639.    ,
           3.575 ]])
```

This array doesn't have any missing values now. We need to convert in into a dataframe again

In [104…
```
type(X)
```

Out[104… numpy.ndarray

In [105…
```
traindata_tr = pd.DataFrame(X, columns = traindata_num.columns)
```

In [106…
```
traindata_tr
```

Out[106…

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
0	-121.89	37.29	38.0	1568.0	351.0	710.0	339.0	2.7042
1	-121.93	37.05	14.0	679.0	108.0	306.0	113.0	6.4214
2	-117.20	32.77	31.0	1952.0	471.0	936.0	462.0	2.8621
3	-119.61	36.31	25.0	1847.0	371.0	1460.0	353.0	1.8839
4	-118.59	34.23	17.0	6592.0	1525.0	4459.0	1463.0	3.0347
...	...	...	...	...	...	...	...	...
16507	-118.13	34.20	46.0	1271.0	236.0	573.0	210.0	4.9312
16508	-117.56	33.88	40.0	1196.0	294.0	1052.0	258.0	2.0682
16509	-116.40	34.09	9.0	4855.0	872.0	2098.0	765.0	3.2723
16510	-118.01	33.82	31.0	1960.0	380.0	1356.0	356.0	4.0625
16511	-122.45	37.77	52.0	3095.0	682.0	1269.0	639.0	3.5750

16512 rows × 8 columns

This dataframe doesn't have any missing values. Let;s check

In [107…
```
traindata_tr[traindata_tr.isna().any(axis = 1)].head()
```

Out[107…

longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income

This shows that there's no missing data now

In [108…
```
traindata_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
```

All the columns now have same number of data in the data fame.

In [109…
```
traindata.head()
```

Out[109

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity
17606	-121.89	37.29	38.0	1568.0	351.0	710.0	339.0	2.7042	<1H OCEAN
18632	-121.93	37.05	14.0	679.0	108.0	306.0	113.0	6.4214	<1H OCEAN
14650	-117.20	32.77	31.0	1952.0	471.0	936.0	462.0	2.8621	NEAR OCEAN
3230	-119.61	36.31	25.0	1847.0	371.0	1460.0	353.0	1.8839	INLAND
3555	-118.59	34.23	17.0	6592.0	1525.0	4459.0	1463.0	3.0347	<1H OCEAN

In [110…

```
#understanding the categorical data in the dataframe

traindata['ocean_proximity'].value_counts()
```

Out[110…

```
<1H OCEAN     7276
INLAND        5263
NEAR OCEAN    2124
NEAR BAY      1847
ISLAND           2
Name: ocean_proximity, dtype: int64
```

In [111…

```
traindata_cat = traindata['ocean_proximity']
```

In [112…

```
# converting oceanproximity to nos.
traindata_cat_encoded,traindata_categories = traindata_cat.factorize()
```

In [113…

```
traindata_cat_encoded[:10]
```

Out[113…

```
array([0, 0, 1, 2, 0, 2, 0, 2, 0, 0], dtype=int64)
```

In [114…

```
traindata_categories
```

Out[114…

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

In [115…

```
# Pandas factorize() example

df = pd.DataFrame({
        'A':['type1','type3','type3', 'type2', 'type0']
    })
df['A'].factorize()
```

Out[115…

```
(array([0, 1, 1, 2, 3], dtype=int64),
 Index(['type1', 'type3', 'type2', 'type0'], dtype='object'))
```

In [116…

```
traindata_cat.head(10)
```

Out[116…

```
17606     <1H OCEAN
18632     <1H OCEAN
14650    NEAR OCEAN
3230         INLAND
3555      <1H OCEAN
19480        INLAND
8879      <1H OCEAN
13685        INLAND
4937      <1H OCEAN
4861      <1H OCEAN
Name: ocean_proximity, dtype: object
```

In [117…

```
traindata_cat_encoded[:20]
```

Out[117…

```
array([0, 0, 1, 2, 0, 2, 0, 2, 0, 0, 2, 2, 0, 2, 2, 0, 3, 2, 2, 2],
      dtype=int64)
```

In [118…
```
df1 =pd.DataFrame({'B':['okay', 'good', 'better', 'best']})
df1['B'].factorize()
```

Out[118…
```
(array([0, 1, 2, 3], dtype=int64),
 Index(['okay', 'good', 'better', 'best'], dtype='object'))
```

In [119…
```
from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder()
```

In [120…
```
traindata_cat_1hot = encoder.fit_transform(traindata_cat_encoded.reshape(-1,1))
```

In [121…
```
traindata_cat_1hot
```

Out[121…
```
<16512x5 sparse matrix of type '<class 'numpy.float64'>'
	with 16512 stored elements in Compressed Sparse Row format>
```

In [122…
```
type(traindata_cat_1hot)
```

Out[122…
```
scipy.sparse.csr.csr_matrix
```

In [123…
```
traindata_cat_1hot.toarray()
```

Out[123…
```
array([[1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       ...,
       [0., 0., 1., 0., 0.],
       [1., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0.]])
```

In [124…
```
traindata_cat_encoded.reshape(-1,1)
```

Out[124…
```
array([[0],
       [0],
       [1],
       ...,
       [2],
       [0],
       [3]], dtype=int64)
```

In [125…
```
# Just run this cell, or copy it to your code, do not try to understand it (yet).
# Definition of the CategoricalEncoder class, copied from PR #9151.

from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.utils import check_array
from sklearn.preprocessing import LabelEncoder
from scipy import sparse

class CategoricalEncoder(BaseEstimator, TransformerMixin):
    """Encode categorical features as a numeric array.
    The input to this transformer should be a matrix of integers or strings,
    denoting the values taken on by categorical (discrete) features.
    The features can be encoded using a one-hot aka one-of-K scheme
    (``encoding='onehot'``, the default) or converted to ordinal integers
    (``encoding='ordinal'``).
    This encoding is needed for feeding categorical data to many scikit-learn
    estimators, notably linear models and SVMs with the standard kernels.
    Read more in the :ref:`User Guide <preprocessing_categorical_features>`.
    Parameters
    ----------
    encoding : str, 'onehot', 'onehot-dense' or 'ordinal'
        The type of encoding to use (default is 'onehot'):
        - 'onehot': encode the features using a one-hot aka one-of-K scheme
```

```
          (or also called 'dummy' encoding). This creates a binary column for
          each category and returns a sparse matrix.
        - 'onehot-dense': the same as 'onehot' but returns a dense array
          instead of a sparse matrix.
        - 'ordinal': encode the features as ordinal integers. This results in
          a single column of integers (0 to n_categories - 1) per feature.
    categories : 'auto' or a list of lists/arrays of values.
        Categories (unique values) per feature:
        - 'auto' : Determine categories automatically from the training data.
        - list : ``categories[i]`` holds the categories expected in the ith
          column. The passed categories are sorted before encoding the data
          (used categories can be found in the ``categories_`` attribute).
    dtype : number type, default np.float64
        Desired dtype of output.
    handle_unknown : 'error' (default) or 'ignore'
        Whether to raise an error or ignore if a unknown categorical feature is
        present during transform (default is to raise). When this is parameter
        is set to 'ignore' and an unknown category is encountered during
        transform, the resulting one-hot encoded columns for this feature
        will be all zeros.
        Ignoring unknown categories is not supported for
        ``encoding='ordinal'``.
    Attributes
    ----------
    categories_ : list of arrays
        The categories of each feature determined during fitting. When
        categories were specified manually, this holds the sorted categories
        (in order corresponding with output of `transform`).
    Examples
    --------
    Given a dataset with three features and two samples, we let the encoder
    find the maximum value per feature and transform the data to a binary
    one-hot encoding.
    >>> from sklearn.preprocessing import CategoricalEncoder
    >>> enc = CategoricalEncoder(handle_unknown='ignore')
    >>> enc.fit([[0, 0, 3], [1, 1, 0], [0, 2, 1], [1, 0, 2]])
    ... # doctest: +ELLIPSIS
    CategoricalEncoder(categories='auto', dtype=<... 'numpy.float64'>,
              encoding='onehot', handle_unknown='ignore')
    >>> enc.transform([[0, 1, 1], [1, 0, 4]]).toarray()
    array([[ 1.,  0.,  0.,  1.,  0.,  0.,  1.,  0.,  0.],
           [ 0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.,  0.]])
    See also
    --------
    sklearn.preprocessing.OneHotEncoder : performs a one-hot encoding of
      integer ordinal features. The ``OneHotEncoder assumes`` that input
      features take on values in the range ``[0, max(feature)]`` instead of
      using the unique values.
    sklearn.feature_extraction.DictVectorizer : performs a one-hot encoding of
      dictionary items (also handles string-valued features).
    sklearn.feature_extraction.FeatureHasher : performs an approximate one-hot
      encoding of dictionary items or strings.
    """

    def __init__(self, encoding='onehot', categories='auto', dtype=np.float64,
                 handle_unknown='error'):
        self.encoding = encoding
        self.categories = categories
        self.dtype = dtype
        self.handle_unknown = handle_unknown

    def fit(self, X, y=None):
        """Fit the CategoricalEncoder to X.
        Parameters
        ----------
        X : array-like, shape [n_samples, n_feature]
            The data to determine the categories of each feature.
        Returns
        -------
        self
        """

        if self.encoding not in ['onehot', 'onehot-dense', 'ordinal']:
            template = ("encoding should be either 'onehot', 'onehot-dense' "
                        "or 'ordinal', got %s")
            raise ValueError(template % self.handle_unknown)

        if self.handle_unknown not in ['error', 'ignore']:
            template = ("handle_unknown should be either 'error' or "
                        "'ignore', got %s")
            raise ValueError(template % self.handle_unknown)

        if self.encoding == 'ordinal' and self.handle_unknown == 'ignore':
            raise ValueError("handle_unknown='ignore' is not supported for"
                             " encoding='ordinal'")

        X = check_array(X, dtype=np.object, accept_sparse='csc', copy=True)
        n_samples, n_features = X.shape
```

```
        self._label_encoders_ = [LabelEncoder() for _ in range(n_features)]

        for i in range(n_features):
            le = self._label_encoders_[i]
            Xi = X[:, i]
            if self.categories == 'auto':
                le.fit(Xi)
            else:
                valid_mask = np.in1d(Xi, self.categories[i])
                if not np.all(valid_mask):
                    if self.handle_unknown == 'error':
                        diff = np.unique(Xi[~valid_mask])
                        msg = ("Found unknown categories {0} in column {1}"
                               " during fit".format(diff, i))
                        raise ValueError(msg)
                le.classes_ = np.array(np.sort(self.categories[i]))

        self.categories_ = [le.classes_ for le in self._label_encoders_]

        return self

    def transform(self, X):
        """Transform X using one-hot encoding.
        Parameters
        ----------
        X : array-like, shape [n_samples, n_features]
            The data to encode.
        Returns
        -------
        X_out : sparse matrix or a 2-d array
            Transformed input.
        """
        X = check_array(X, accept_sparse='csc', dtype=np.object, copy=True)
        n_samples, n_features = X.shape
        X_int = np.zeros_like(X, dtype=np.int)
        X_mask = np.ones_like(X, dtype=np.bool)

        for i in range(n_features):
            valid_mask = np.in1d(X[:, i], self.categories_[i])

            if not np.all(valid_mask):
                if self.handle_unknown == 'error':
                    diff = np.unique(X[~valid_mask, i])
                    msg = ("Found unknown categories {0} in column {1}"
                           " during transform".format(diff, i))
                    raise ValueError(msg)
                else:
                    # Set the problematic rows to an acceptable value and
                    # continue `The rows are marked `X_mask` and will be
                    # removed later.
                    X_mask[:, i] = valid_mask
                    X[:, i][~valid_mask] = self.categories_[i][0]
            X_int[:, i] = self._label_encoders_[i].transform(X[:, i])

        if self.encoding == 'ordinal':
            return X_int.astype(self.dtype, copy=False)

        mask = X_mask.ravel()
        n_values = [cats.shape[0] for cats in self.categories_]
        n_values = np.array([0] + n_values)
        indices = np.cumsum(n_values)

        column_indices = (X_int + indices[:-1]).ravel()[mask]
        row_indices = np.repeat(np.arange(n_samples, dtype=np.int32),
                                n_features)[mask]
        data = np.ones(n_samples * n_features)[mask]

        out = sparse.csc_matrix((data, (row_indices, column_indices)),
                                shape=(n_samples, indices[-1]),
                                dtype=self.dtype).tocsr()
        if self.encoding == 'onehot-dense':
            return out.toarray()
        else:
            return out
```

In [126…

```
# Creating custom Transformer
from sklearn.base import BaseEstimator, TransformerMixin

# column index
rooms_ix, bedrooms_ix, population_ix, household_ix = 3, 4, 5, 6

class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self, add_bedrooms_per_room = True): # no *args or **kargs
        self.add_bedrooms_per_room = add_bedrooms_per_room
    def fit(self, X, y=None):
        return self  # nothing else to do
    def transform(self, X, y=None):
```

```
        rooms_per_household = X[:, rooms_ix] / X[:, household_ix]
        population_per_household = X[:, population_ix] / X[:, household_ix]
        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:, bedrooms_ix] / X[:, rooms_ix]
            return np.c_[X, rooms_per_household, population_per_household,
                         bedrooms_per_room]
        else:
            return np.c_[X, rooms_per_household, population_per_household]

attr_adder = CombinedAttributesAdder(add_bedrooms_per_room=False)
traindata_extra_attribs = attr_adder.transform(traindata.values)
traindata_extra_attribs = pd.DataFrame(traindata_extra_attribs, columns=list(traindata.columns)+["rooms_per_house
traindata_extra_attribs.head()
```

Out[126...

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity	rooms_per_
0	-121.89	37.29	38.0	1568.0	351.0	710.0	339.0	2.7042	<1H OCEAN	
1	-121.93	37.05	14.0	679.0	108.0	306.0	113.0	6.4214	<1H OCEAN	
2	-117.2	32.77	31.0	1952.0	471.0	936.0	462.0	2.8621	NEAR OCEAN	
3	-119.61	36.31	25.0	1847.0	371.0	1460.0	353.0	1.8839	INLAND	
4	-118.59	34.23	17.0	6592.0	1525.0	4459.0	1463.0	3.0347	<1H OCEAN	

In [127...

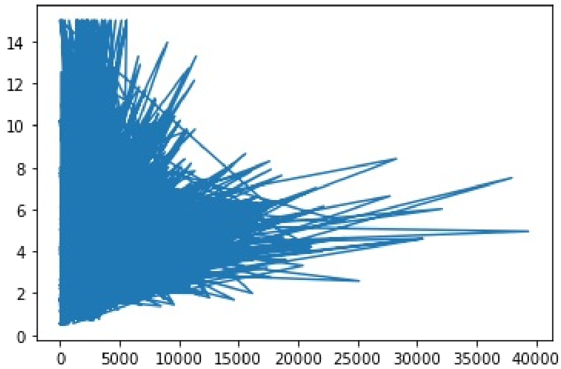
```
max(traindata['total_rooms']), min(traindata['total_rooms'])
```

Out[127... (39320.0, 6.0)

In [128...

```
plt.plot(traindata['total_rooms'], traindata['median_income'])
```

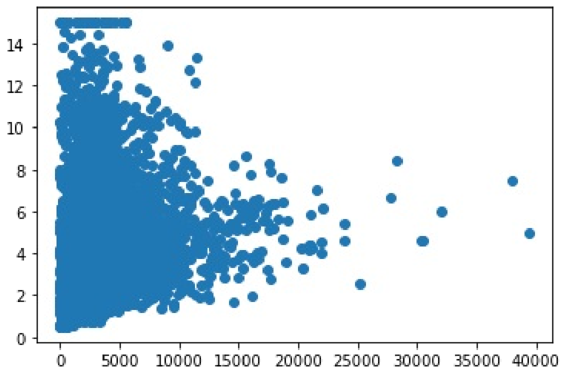
Out[128... [<matplotlib.lines.Line2D at 0x1a68ed99fd0>]



In [129...

```
plt.scatter(traindata['total_rooms'], traindata['median_income'])
```

Out[129... <matplotlib.collections.PathCollection at 0x1a68b049f10>



In [130...

```
# Feature Scaling - Min-max Scaling - Example
# Creating DataFrame first

s1 = pd.Series([1, 2, 3, 4, 5, 6], index=(range(6)))
```

```
s2 = pd.Series([10, 9, 8, 7, 6, 5], index=(range(6)))
df = pd.DataFrame(s1, columns=['s1'])
df['s2'] = s2
df
```

Out[130…

	s1	s2
0	1	10
1	2	9
2	3	8
3	4	7
4	5	6
5	6	5

In [131…

```
# Use Scikit-Learn minmax_scaling


#from mlxtend.preprocessing import minmax_scaling
#minmax_scaling(df, columns=['s1', 's2'])
```

In [132…

```
#standardisation of the data
from sklearn.preprocessing import StandardScaler
stdscaler = StandardScaler()
df_tr= stdscaler.fit_transform(df)
df_tr
```

Out[132…

```
array([[-1.46385011,  1.46385011],
       [-0.87831007,  0.87831007],
       [-0.29277002,  0.29277002],
       [ 0.29277002, -0.29277002],
       [ 0.87831007, -0.87831007],
       [ 1.46385011, -1.46385011]])
```

In [133…

```
pd.DataFrame(df_tr, columns =['s1' ,'s2'])
```

Out[133…

	s1	s2
0	-1.46385	1.46385
1	-0.87831	0.87831
2	-0.29277	0.29277
3	0.29277	-0.29277
4	0.87831	-0.87831
5	1.46385	-1.46385

In [134…

```
#performing a series of transfomation using pipeline
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
#note: we have already created a method by the name CombinedAttributesAdder() above and defined fit and transfrom
num_pipeline = Pipeline([('imputer' , SimpleImputer(strategy ='median')), ('attribs_adder', CombinedAttributesAdc
traindata_num_tr = num_pipeline.fit_transform(traindata_num)
```

In [135…

```
traindata_num_tr
# note: scikit learn does not handle dataframe
```

Out[135…

```
array([[-1.15604281,  0.77194962,  0.74333089, ..., -0.31205452,
        -0.08649871,  0.15531753],
       [-1.17602483,  0.6596948 , -1.1653172 , ...,  0.21768338,
        -0.03353391, -0.83628902],
       [ 1.18684903, -1.34218285,  0.18664186, ..., -0.46531516,
        -0.09240499,  0.4222004 ],
       ...,
       [ 1.58648943, -0.72478134, -1.56295222, ...,  0.3469342 ,
        -0.03055414, -0.52177644],
       [ 0.78221312, -0.85106801,  0.18664186, ...,  0.02499488,
         0.06150916, -0.30340741],
       [-1.43579109,  0.99645926,  1.85670895, ..., -0.22852947,
        -0.09586294,  0.10180567]])
```

In [136…

```
traindata_num
```

Out[136…

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
17606	-121.89	37.29	38.0	1568.0	351.0	710.0	339.0	2.7042
18632	-121.93	37.05	14.0	679.0	108.0	306.0	113.0	6.4214
14650	-117.20	32.77	31.0	1952.0	471.0	936.0	462.0	2.8621
3230	-119.61	36.31	25.0	1847.0	371.0	1460.0	353.0	1.8839
3555	-118.59	34.23	17.0	6592.0	1525.0	4459.0	1463.0	3.0347
...	...	...	...	...	...	...	...	...
6563	-118.13	34.20	46.0	1271.0	236.0	573.0	210.0	4.9312
12053	-117.56	33.88	40.0	1196.0	294.0	1052.0	258.0	2.0682
13908	-116.40	34.09	9.0	4855.0	872.0	2098.0	765.0	3.2723
11159	-118.01	33.82	31.0	1960.0	380.0	1356.0	356.0	4.0625
15775	-122.45	37.77	52.0	3095.0	682.0	1269.0	639.0	3.5750

16512 rows × 8 columns

In [137…

```
type(traindata_num_tr)
```

Out[137…

```
numpy.ndarray
```

In [138…

```
num_attribs = list(traindata_num)
num_attribs
```

Out[138…

```
['longitude',
 'latitude',
 'housing_median_age',
 'total_rooms',
 'total_bedrooms',
 'population',
 'households',
 'median_income']
```

In [139…

```
from sklearn.base import BaseEstimator, TransformerMixin

# Create a class to select numerical or categorical columns
# since Scikit-Learn doesn't handle DataFrames yet
class DataFrameSelector(BaseEstimator, TransformerMixin):
    def __init__(self, attribute_names):
        self.attribute_names = attribute_names
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        return X[self.attribute_names].values
```

In [140…

```
num_attribs = list(traindata_num)
num_pipeline = Pipeline([('selector', DataFrameSelector(num_attribs)),('imputer' , SimpleImputer(strategy ='media
traindata_num_tr = num_pipeline.fit_transform(traindata_num)

#we've also created a pipleline for categorical attributes
cat_attribs = ['ocean_proximity']
cat_pipeline = Pipeline([('selector', DataFrameSelector(cat_attribs)),
                         ('cat_encoder', CategoricalEncoder(encoding='onehot-dense'))])
```

In [141…

```
Y = num_pipeline.fit_transform(traindata)
# This is the command to fit and  transform the numerical attributes. since scikit learn does not yeild datafram
#retured
```

In [142…

```
type(Y)
```

Out[142…

```
numpy.ndarray
```

In [143…

```
Y
```

Out[143…

```
array([[-1.15604281,  0.77194962,  0.74333089, ..., -0.31205452,
        -0.08649871,  0.15531753],
       [-1.17602483,  0.6596948 , -1.1653172 , ...,  0.21768338,
        -0.03353391, -0.83628902],
       [ 1.18684903, -1.34218285,  0.18664186, ..., -0.46531516,
        -0.09240499,  0.4222004 ],
       ...,
       [ 1.58648943, -0.72478134, -1.56295222, ...,  0.3469342 ,
        -0.03055414, -0.52177644],
       [ 0.78221312, -0.85106801,  0.18664186, ...,  0.02499488,
         0.06150916, -0.30340741],
       [-1.43579109,  0.99645926,  1.85670895, ..., -0.22852947,
        -0.09586294,  0.10180567]])
```

In [144…

```
#converting the array into data frame
#note: we have used the columns of the dataframe that has all the extra attributed added to it. we cant use the c
# and the new extra added attributes dataframe contains 11 columns
traindata_scaled_tr = pd.DataFrame(Y, columns =traindata_extra_attribs.columns)
traindata_scaled_tr
```

Out[144…

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity	room
0	-1.156043	0.771950	0.743331	-0.493234	-0.445438	-0.636211	-0.420698	-0.614937	-0.312055	
1	-1.176025	0.659695	-1.165317	-0.908967	-1.036928	-0.998331	-1.022227	1.336459	0.217683	
2	1.186849	-1.342183	0.186642	-0.313660	-0.153345	-0.433639	-0.093318	-0.532046	-0.465315	
3	-0.017068	0.313576	-0.290520	-0.362762	-0.396756	0.036041	-0.383436	-1.045566	-0.079661	
4	0.492474	-0.659299	-0.926736	1.856193	2.412211	2.724154	2.570975	-0.441437	-0.357834	
...	...	...	...	...	...	...	...	...	...	
16507	0.722267	-0.673331	1.379547	-0.632123	-0.725361	-0.759010	-0.764049	0.554158	0.234352	
16508	1.007011	-0.823004	0.902385	-0.667196	-0.584183	-0.329664	-0.636291	-0.948815	-0.308114	
16509	1.586489	-0.724781	-1.562952	1.043901	0.822735	0.607904	0.713156	-0.316705	0.346934	
16510	0.782213	-0.851068	0.186642	-0.309919	-0.374849	-0.057178	-0.375451	0.098121	0.024995	
16511	-1.435791	0.996459	1.856709	0.220853	0.360253	-0.135159	0.377791	-0.157799	-0.228529	

16512 rows × 11 columns

In [145…

```
#we are not going to run the pipeline for the categotical data seperately. Instead we will use the FeatureUnion (
#concatenation of the pipelines
from sklearn.pipeline import FeatureUnion

full_pipeline = FeatureUnion(transformer_list=[
        ("num_pipeline", num_pipeline),
        ("cat_pipeline", cat_pipeline),
    ])
```

In [146…

```
type(full_pipeline)
```

Out[146…

```
sklearn.pipeline.FeatureUnion
```

In [147…

```
traindata_prepared_arr = full_pipeline.fit_transform(traindata)
#this is the array with numerical as well as categorical attributes
traindata_prepared_arr[0]
traindata_prepared_arr[1]
```

```
<ipython-input-125-341a9c0cd727>:110: DeprecationWarning: `np.object` is a deprecated alias for the builtin `obje
ct`. To silence this warning, use `object` by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#depr
ecations
  X = check_array(X, dtype=np.object, accept_sparse='csc', copy=True)
<ipython-input-125-341a9c0cd727>:145: DeprecationWarning: `np.object` is a deprecated alias for the builtin `obje
ct`. To silence this warning, use `object` by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#depr
ecations
```

```
  X = check_array(X, accept_sparse='csc', dtype=np.object, copy=True)
<ipython-input-125-341a9c0cd727>:147: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. T
o silence this warning, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing
`np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to review your
current use, check the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#depr
ecations
  X_int = np.zeros_like(X, dtype=np.int)
<ipython-input-125-341a9c0cd727>:148: DeprecationWarning: `np.bool` is a deprecated alias for the builtin `bool`.
To silence this warning, use `bool` by itself. Doing this will not modify any behavior and is safe. If you specif
ically wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#depr
ecations
  X_mask = np.ones_like(X, dtype=np.bool)
```

Out[147…

```
array([-1.17602483,  0.6596948 , -1.1653172 , -0.90896655, -1.0369278 ,
       -0.99833135, -1.02222705,  1.33645936,  0.21768338, -0.03353391,
       -0.83628902,  1.        ,  0.        ,  0.        ,  0.        ,
        0.        ])
```

In [148…

```
traindata_prepared_arr[1]
```

Out[148…

```
array([-1.17602483,  0.6596948 , -1.1653172 , -0.90896655, -1.0369278 ,
       -0.99833135, -1.02222705,  1.33645936,  0.21768338, -0.03353391,
       -0.83628902,  1.        ,  0.        ,  0.        ,  0.        ,
        0.        ])
```

In [149…

```
traindata_prepared_arr[52]
```

Out[149…

```
array([-1.18601584,  0.75791776,  0.42522288, -0.53345104, -0.67911311,
       -0.7491498 , -0.66823011,  0.85910944,  0.22363394, -0.06236395,
       -0.63722671,  1.        ,  0.        ,  0.        ,  0.        ,
        0.        ])
```

In [150…

```
traindata_prepared_arr
```

Out[150…

```
array([[-1.15604281,  0.77194962,  0.74333089, ...,  0.        ,
         0.        ,  0.        ],
       [-1.17602483,  0.6596948 , -1.1653172 , ...,  0.        ,
         0.        ,  0.        ],
       [ 1.18684903, -1.34218285,  0.18664186, ...,  0.        ,
         0.        ,  1.        ],
       ...,
       [ 1.58648943, -0.72478134, -1.56295222, ...,  0.        ,
         0.        ,  0.        ],
       [ 0.78221312, -0.85106801,  0.18664186, ...,  0.        ,
         0.        ,  0.        ],
       [-1.43579109,  0.99645926,  1.85670895, ...,  0.        ,
         1.        ,  0.        ]])
```

In [151…

```
traindata_labels.head()
type(traindata_labels)
```

Out[151…

```
pandas.core.series.Series
```

In [152…

```
traindata_scaled_tr
```

Out[152…

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity	room
0	-1.156043	0.771950	0.743331	-0.493234	-0.445438	-0.636211	-0.420698	-0.614937	-0.312055	
1	-1.176025	0.659695	-1.165317	-0.908967	-1.036928	-0.998331	-1.022227	1.336459	0.217683	
2	1.186849	-1.342183	0.186642	-0.313660	-0.153345	-0.433639	-0.093318	-0.532046	-0.465315	
3	-0.017068	0.313576	-0.290520	-0.362762	-0.396756	0.036041	-0.383436	-1.045566	-0.079661	
4	0.492474	-0.659299	-0.926736	1.856193	2.412211	2.724154	2.570975	-0.441437	-0.357834	

...	...	...	...	...	...	...	...	...	...
16507	0.722267	-0.673331	1.379547	-0.632123	-0.725361	-0.759010	-0.764049	0.554158	0.234352
16508	1.007011	-0.823004	0.902385	-0.667196	-0.584183	-0.329664	-0.636291	-0.948815	-0.308114
16509	1.586489	-0.724781	-1.562952	1.043901	0.822735	0.607904	0.713156	-0.316705	0.346934
16510	0.782213	-0.851068	0.186642	-0.309919	-0.374849	-0.057178	-0.375451	0.098121	0.024995
16511	-1.435791	0.996459	1.856709	0.220853	0.360253	-0.135159	0.377791	-0.157799	-0.228529

16512 rows × 11 columns

In [153…

```
traindata
```

Out[153…

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity
17606	-121.89	37.29	38.0	1568.0	351.0	710.0	339.0	2.7042	<1H OCEAN
18632	-121.93	37.05	14.0	679.0	108.0	306.0	113.0	6.4214	<1H OCEAN
14650	-117.20	32.77	31.0	1952.0	471.0	936.0	462.0	2.8621	NEAR OCEAN
3230	-119.61	36.31	25.0	1847.0	371.0	1460.0	353.0	1.8839	INLAND
3555	-118.59	34.23	17.0	6592.0	1525.0	4459.0	1463.0	3.0347	<1H OCEAN
...	...	...	...	...	...	...	...	...	...
6563	-118.13	34.20	46.0	1271.0	236.0	573.0	210.0	4.9312	INLAND
12053	-117.56	33.88	40.0	1196.0	294.0	1052.0	258.0	2.0682	INLAND
13908	-116.40	34.09	9.0	4855.0	872.0	2098.0	765.0	3.2723	INLAND
11159	-118.01	33.82	31.0	1960.0	380.0	1356.0	356.0	4.0625	<1H OCEAN
15775	-122.45	37.77	52.0	3095.0	682.0	1269.0	639.0	3.5750	NEAR BAY

16512 rows × 9 columns

In [154…

```
#training the model-Linear Regression
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()  #initializing the object
h= lin_reg.fit(traindata_prepared_arr, traindata_labels)
```

In [155…

```
type(h)
```

Out[155…

```
sklearn.linear_model._base.LinearRegression
```

In [156…

```
h
```

Out[156…

```
LinearRegression()
```

In [157…

```
#lets try the full pipeline on new instances
some_data = traindata.iloc[:5]
some_labels= traindata_labels[:5]
some_data_prepared_arr= full_pipeline.transform(some_data)
```

```
<ipython-input-125-341a9c0cd727>:145: DeprecationWarning: `np.object` is a deprecated alias for the builtin `obje
ct`. To silence this warning, use `object` by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#depr
ecations
  X = check_array(X, accept_sparse='csc', dtype=np.object, copy=True)
<ipython-input-125-341a9c0cd727>:147: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. T
o silence this warning, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing
`np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to review your
current use, check the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#depr
ecations
  X_int = np.zeros_like(X, dtype=np.int)
<ipython-input-125-341a9c0cd727>:148: DeprecationWarning: `np.bool` is a deprecated alias for the builtin `bool`.
To silence this warning, use `bool` by itself. Doing this will not modify any behavior and is safe. If you specif
ically wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#depr
ecations
```

```
X_mask = np.ones_like(X, dtype=np.bool)
```

In [158…

```
some_data_prepared_arr
```

Out[158…

```
array([[-1.15604281,  0.77194962,  0.74333089, -0.49323393, -0.44543821,
        -0.63621141, -0.42069842, -0.61493744, -0.31205452, -0.08649871,
         0.15531753,  1.        ,  0.        ,  0.        ,  0.        ,
         0.        ],
       [-1.17602483,  0.6596948 , -1.1653172 , -0.90896655, -1.0369278 ,
        -0.99833135, -1.02222705,  1.33645936,  0.21768338, -0.03353391,
        -0.83628902,  1.        ,  0.        ,  0.        ,  0.        ,
         0.        ],
       [ 1.18684903, -1.34218285,  0.18664186, -0.31365989, -0.15334458,
        -0.43363936, -0.0933178 , -0.5320456 , -0.46531516, -0.09240499,
         0.4222004 ,  0.        ,  0.        ,  0.        ,  0.        ,
         1.        ],
       [-0.01706767,  0.31357576, -0.29052016, -0.36276217, -0.39675594,
         0.03604096, -0.38343559, -1.04556555, -0.07966124,  0.08973561,
        -0.19645314,  0.        ,  1.        ,  0.        ,  0.        ,
         0.        ],
       [ 0.49247384, -0.65929936, -0.92673619,  1.85619316,  2.41221109,
         2.72415407,  2.57097492, -0.44143679, -0.35783383, -0.00419445,
         0.2699277 ,  1.        ,  0.        ,  0.        ,  0.        ,
         0.        ]])
```

Understanding the algorithm in Detail

We extracted five rows from the trainded dataframe containing all the attributes except median_house valuesome

Then we extracted five rows from the median_house values

In the third step:

We sent the dataframe 'some_data' to the 'full_pipleline' which contains the object FeatureUnion that combined the categorical and numerical pipleline together. The dataframe 'some_data' goes through all the cleaning necessary through these pipelines and then is transformed into an array containing all its attributes.

In [159…

```
#now we do the prediction
print('predictions', h.predict(some_data_prepared_arr))
```

```
predictions [210644.60459286 317768.80697211 210956.43331178  59218.98886849
 189747.55849879]
```

In [160…

```
print('labels', list(some_labels))
```

```
labels [286600.0, 340600.0, 196900.0, 46300.0, 254500.0]
```

In [161…

```
traindata_predictions = h.predict(traindata_prepared_arr) # this predicts all the data
```

In [162…

```
type(traindata_predictions)
```

Out[162…

```
numpy.ndarray
```

In [163…

```
traindata_predictions # this is an array
len(traindata_predictions)
```

Out[163…

```
16512
```

In [164…

```
type(traindata_labels) # this is a series
len(traindata_labels)
```

Out[164…

```
16512
```

In [165…
```
#calculated mean squared error
from sklearn.metrics import mean_squared_error
lin_mse = mean_squared_error(traindata_predictions, traindata_labels)
type(lin_mse)
```
Out[165… numpy.float64

In [166…
```
lin_mse
```
Out[166… 4709829587.971121

In [167…
```
lin_rmse= np.sqrt(lin_mse)
lin_rmse
```
Out[167… 68628.19819848923

The error is huge indeed. This means that the fitted has deviated from the actual value by about 68628 above or below. This is an example of underrfitting.

In [168…
```
#using decision trees on the data
#this is another model used for predictions
from sklearn.tree import DecisionTreeRegressor
tree_reg = DecisionTreeRegressor(random_state=42) # initilisation
tree_reg.fit(traindata_prepared_arr, traindata_labels) #this commands help  the algorithm learn from the data we
```
Out[168… DecisionTreeRegressor(random_state=42)

In [169…
```
#now we predict from the fitted data, it will generate an array of fitted values of housing_median_values
traindata_predictions_1 = tree_reg.predict(traindata_prepared_arr)
```
In [170…
```
traindata_predictions_1[:5]
```
Out[170… array([286600., 340600., 196900., 46300., 254500.])

In [171…
```
#cacluate the root mean squared error..
tree_mse = mean_squared_error(traindata_labels, traindata_predictions_1)
tree_rmse = np.sqrt(tree_mse)
tree_rmse
```
Out[171… 0.0

The RMSE obtained is zero. This shows that the model seems to have memorized the whole training set. This is called overfitting.So we perform the Cross-validation over the training set without touching the training set. We shall create training folds out of the training set

In [172…
```
from sklearn.model_selection import cross_val_score
tree_reg = DecisionTreeRegressor(random_state =42)
scores = cross_val_score(tree_reg, traindata_prepared_arr, traindata_labels, scoring= 'neg_mean_squared_error', c
```
In [173…
```
scores
```
Out[173…
```
array([-4.92724492e+09, -4.46961291e+09, -5.24647900e+09, -5.00679914e+09,
       -5.05746872e+09, -5.71311365e+09, -4.93686969e+09, -4.93838343e+09,
       -5.68016653e+09, -5.07394900e+09])
```
In [174…
```
tree_rmse_scores = np.sqrt(-scores)
```

```
tree_rmse_scores
```

Out[174… array([70194.33680785, 66855.16363941, 72432.58244769, 70758.73896782,
       71115.88230639, 75585.14172901, 70262.86139133, 70273.6325285 ,
       75366.87952553, 71231.65726027])

In [175…

```
 #Look at the score of cross-validation of DecisionTreeRegressor

def display_scores(scores):
    print("Scores:", scores)
    print("Mean:", scores.mean())
    print("Standard deviation:", scores.std())

display_scores(tree_rmse_scores)
```

```
Scores: [70194.33680785 66855.16363941 72432.58244769 70758.73896782
 71115.88230639 75585.14172901 70262.86139133 70273.6325285
 75366.87952553 71231.65726027]
Mean: 71407.68766037929
Standard deviation: 2439.4345041191004
```

In [176…

```
#using cros valirdation on Linear Regression
lin_reg = LinearRegression()
lin_scores = cross_val_score(lin_reg, traindata_prepared_arr, traindata_labels, scoring= 'neg_mean_squared_error'
lin_rmse_scores = np.sqrt(-lin_scores)
lin_rmse_scores
```

Out[176… array([66782.73843989, 66960.118071 , 70347.95244419, 74739.57052552,
       68031.13388938, 71193.84183426, 64969.63056405, 68281.61137997,
       71552.91566558, 67665.10082067])

In [177…

```
display_scores(lin_rmse_scores)
```

```
Scores: [66782.73843989 66960.118071   70347.95244419 74739.57052552
 68031.13388938 71193.84183426 64969.63056405 68281.61137997
 71552.91566558 67665.10082067]
Mean: 69052.46136345083
Standard deviation: 2731.674001798348
```

So we see that the mean value(69052.46136345083) for Linear regression is less than the mean value(71407.68766037929) of the Decision Trees. Which one to choose? Let's see further

In [178…

```
# Let's train one more model using Random Forests

from sklearn.ensemble import RandomForestRegressor

forest_reg = RandomForestRegressor(random_state=42)
forest_reg.fit(traindata_prepared_arr, traindata_labels)
```

Out[178… RandomForestRegressor(random_state=42)

In [179…

```
#prediction and MSE calculation
traindata_predictions_2 = forest_reg.predict(traindata_prepared_arr) #creates an array of predticted values
forest_mse = mean_squared_error(traindata_predictions_2, traindata_labels) #gives the mean squared error
forest_rmse = np.sqrt(forest_mse)
forest_rmse
```

Out[179… 18603.515021376355

In [180…

```
#cross validation for Random Forest
forest_reg= RandomForestRegressor(random_state=42)
forest_scores = cross_val_score(forest_reg, traindata_prepared_arr, traindata_labels, scoring= 'neg_mean_squared_
forest_rmse_scores = np.sqrt(-forest_scores)
```

In [181…

```
forest_rmse_scores
```

Out[181… array([49519.80364233, 47461.9115823 , 50029.02762854, 52325.28068953,
       49308.39426421, 53446.37892622, 48634.8036574 , 47585.73832311,
       53490.10699751, 50021.5852922])

In [182…
```
display_scores(forest_rmse_scores)
```

```
Scores: [49519.80364233 47461.9115823  50029.02762854 52325.28068953
 49308.39426421 53446.37892622 48634.8036574  47585.73832311
 53490.10699751 50021.5852922 ]
Mean: 50182.303100336096
Standard deviation: 2097.0810550985693
```

so we have applied three models till now.

Now that we have checked three models, it is time to fine tune the model. In the process of fine tuning, we need to combine hyperparameters and perform gridsearching to get the best model. See copy

In [183…
```
from sklearn.model_selection import GridSearchCV
 # try 12 (3×4) combinations of hyperparameters
param_grid =[{'n_estimators':[3,10,30],'max_features':[2,4,6,8]},
            # then try 6 (2×3) combinations with bootstrap set as False
            {'bootstrap':[False],'n_estimators':[3,10],'max_features':[2,3,4]}]
```

In [184…
```
forest_reg = RandomForestRegressor(random_state = 42)
grid_search = GridSearchCV(forest_reg, param_grid, cv=5, scoring= 'neg_mean_squared_error')
grid_search.fit(traindata_prepared_arr, traindata_labels)
```

Out[184…
```
GridSearchCV(cv=5, estimator=RandomForestRegressor(random_state=42),
             param_grid=[{'max_features': [2, 4, 6, 8],
                          'n_estimators': [3, 10, 30]},
                         {'bootstrap': [False], 'max_features': [2, 3, 4],
                          'n_estimators': [3, 10]}],
             scoring='neg_mean_squared_error')
```

In [185…
```
#best hyperparameter
grid_search.best_params_
```

Out[185… {'max_features': 8, 'n_estimators': 30}

In [186…
```
#best estimator
grid_search.best_estimator_
```

Out[186… RandomForestRegressor(max_features=8, n_estimators=30, random_state=42)

In [187…
```
#looking at score of each hyperparamter combination tested during the grid search
cvres = grid_search.cv_results_
for mean_score , params in zip(cvres['mean_test_score'],cvres['params']):
    print(np.sqrt(-mean_score), params)
```

```
63669.11631261028 {'max_features': 2, 'n_estimators': 3}
55627.099719926795 {'max_features': 2, 'n_estimators': 10}
53384.57275149205 {'max_features': 2, 'n_estimators': 30}
60965.950449450494 {'max_features': 4, 'n_estimators': 3}
52741.04704299915 {'max_features': 4, 'n_estimators': 10}
50377.40461678399 {'max_features': 4, 'n_estimators': 30}
58663.93866579625 {'max_features': 6, 'n_estimators': 3}
52006.19873526564 {'max_features': 6, 'n_estimators': 10}
50146.51167415009 {'max_features': 6, 'n_estimators': 30}
57869.25276169646 {'max_features': 8, 'n_estimators': 3}
51711.127883959234 {'max_features': 8, 'n_estimators': 10}
49682.273345071546 {'max_features': 8, 'n_estimators': 30}
62895.06951262424 {'bootstrap': False, 'max_features': 2, 'n_estimators': 3}
54658.176157539405 {'bootstrap': False, 'max_features': 2, 'n_estimators': 10}
59470.40652318466 {'bootstrap': False, 'max_features': 3, 'n_estimators': 3}
52724.9822587892 {'bootstrap': False, 'max_features': 3, 'n_estimators': 10}
```

```
57490.5691951261 {'bootstrap': False, 'max_features': 4, 'n_estimators': 3}
51009.495668875716 {'bootstrap': False, 'max_features': 4, 'n_estimators': 10}
```

Thus the lowest score we have is for (8,30)

49682.273345071546 {'max_features': 8, 'n_estimators': 30}

In [188…

```
# RandomizedSearchCV

from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

param_distribs = {
        'n_estimators': randint(low=1, high=200),
        'max_features': randint(low=1, high=8),
    }

forest_reg = RandomForestRegressor(random_state=42)
rnd_search = RandomizedSearchCV(forest_reg, param_distributions=param_distribs,
                                n_iter=10, cv=5, scoring='neg_mean_squared_error', random_state=42)
rnd_search.fit(traindata_prepared_arr, traindata_labels)
```

Out[188…

```
RandomizedSearchCV(cv=5, estimator=RandomForestRegressor(random_state=42),
                   param_distributions={'max_features': <scipy.stats._distn_infrastructure.rv_frozen object at 0x
000001A68EF8EC40>,
                                        'n_estimators': <scipy.stats._distn_infrastructure.rv_frozen object at 0x
000001A68EFAD700>},
                   random_state=42, scoring='neg_mean_squared_error')
```

In [189…

```
rnd_search.best_estimator_
```

Out[189…

```
RandomForestRegressor(max_features=7, n_estimators=180, random_state=42)
```

In [190…

```
rnd_search.best_params_
```

Out[190…

```
{'max_features': 7, 'n_estimators': 180}
```

In [191…

```
cvres1 = rnd_search.cv_results_
for mean_score , params in zip(cvres1['mean_test_score'],cvres1['params']):
    print(np.sqrt(-mean_score), params)
```

```
49150.70756927707 {'max_features': 7, 'n_estimators': 180}
51389.889203389284 {'max_features': 5, 'n_estimators': 15}
50796.155224308866 {'max_features': 3, 'n_estimators': 72}
50835.13360315349 {'max_features': 5, 'n_estimators': 21}
49280.9449827171 {'max_features': 7, 'n_estimators': 122}
50774.90662363929 {'max_features': 3, 'n_estimators': 75}
50682.78888164288 {'max_features': 3, 'n_estimators': 88}
49608.99608105296 {'max_features': 5, 'n_estimators': 100}
50473.61930350219 {'max_features': 3, 'n_estimators': 150}
64429.84143294435 {'max_features': 5, 'n_estimators': 2}
```

Thus the lowest score we have is for (7,180)

49150.70756927707 {'max_features': 7, 'n_estimators': 180}

In [194…

```
#see the importance score of each attibute using GridsearchCV
feature_importances = grid_search.best_estimator_.feature_importances_
extra_attribs = ["rooms_per_hhold", "pop_per_hhold", "bedrooms_per_room"]
cat_encoder = cat_pipeline.named_steps["cat_encoder"]
cat_one_hot_attribs = list(cat_encoder.categories_[0])
attributes = num_attribs + extra_attribs + cat_one_hot_attribs
sorted(zip(feature_importances, attributes), reverse=True)
```

Out[194…

```
[(0.36615898061813423, 'median_income'),
```

```
(0.16478099356159054, 'INLAND'),
(0.10879295677551575, 'pop_per_hhold'),
(0.07334423551601243, 'longitude'),
(0.06290907048262032, 'latitude'),
(0.056419179181954014, 'rooms_per_hhold'),
(0.053351077347675815, 'bedrooms_per_room'),
(0.04114379847872964, 'housing_median_age'),
(0.014874280890402769, 'population'),
(0.014672685420543239, 'total_rooms'),
(0.014257599323407808, 'households'),
(0.014106483453584104, 'total_bedrooms'),
(0.010311488326303788, '<1H OCEAN'),
(0.0028564746373201584, 'NEAR OCEAN'),
(0.0019604155994780706, 'NEAR BAY'),
(6.0280386727366e-05, 'ISLAND')]
```

In [195…

```
# Evaluate model on the Test Set

final_model = grid_search.best_estimator_
X_test = strat_test_set.drop("median_house_value", axis=1)
y_test = strat_test_set["median_house_value"].copy()
X_test_prepared = full_pipeline.transform(X_test)
```

```
<ipython-input-125-341a9c0cd727>:145: DeprecationWarning: `np.object` is a deprecated alias for the builtin `obje
ct`. To silence this warning, use `object` by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#depr
ecations
  X = check_array(X, accept_sparse='csc', dtype=np.object, copy=True)
<ipython-input-125-341a9c0cd727>:147: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. T
o silence this warning, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing
`np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to review your
current use, check the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#depr
ecations
  X_int = np.zeros_like(X, dtype=np.int)
<ipython-input-125-341a9c0cd727>:148: DeprecationWarning: `np.bool` is a deprecated alias for the builtin `bool`.
To silence this warning, use `bool` by itself. Doing this will not modify any behavior and is safe. If you specif
ically wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#depr
ecations
  X_mask = np.ones_like(X, dtype=np.bool)
```

In [196…

```
#command for data cleaning through the pipeline that was created earlier
X_test_prepared = full_pipeline.transform(X_test)
```


```
<ipython-input-125-341a9c0cd727>:145: DeprecationWarning: `np.object` is a deprecated alias for the builtin `obje
ct`. To silence this warning, use `object` by itself. Doing this will not modify any behavior and is safe.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#depr
ecations
  X = check_array(X, accept_sparse='csc', dtype=np.object, copy=True)
<ipython-input-125-341a9c0cd727>:147: DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. T
o silence this warning, use `int` by itself. Doing this will not modify any behavior and is safe. When replacing
`np.int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision. If you wish to review your
current use, check the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#depr
ecations
  X_int = np.zeros_like(X, dtype=np.int)
<ipython-input-125-341a9c0cd727>:148: DeprecationWarning: `np.bool` is a deprecated alias for the builtin `bool`.
To silence this warning, use `bool` by itself. Doing this will not modify any behavior and is safe. If you specif
ically wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#depr
ecations
  X_mask = np.ones_like(X, dtype=np.bool)
```


In [197…

```
#prediction using the RandomforestRegressor mode. will create and array of predicted values for median_house_valu
final_predictions = final_model.predict(X_test_prepared)
```

In [198…

```
#calculating the RMSE
final_mse = mean_squared_error(y_test, final_predictions)
final_rmse = np.sqrt(final_mse)
final_rmse
```

Out[198… 47730.22690385927

In [199…

```
final_model
```

Out[199…

```
RandomForestRegressor(max_features=8, n_estimators=30, random_state=42)
```

In [200…

```
X_test
```

Out[200…

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity
5241	-118.39	34.12	29.0	6447.0	1012.0	2184.0	960.0	8.2816	<1H OCEAN
10970	-117.86	33.77	39.0	4159.0	655.0	1669.0	651.0	4.6111	<1H OCEAN
20351	-119.05	34.21	27.0	4357.0	926.0	2110.0	876.0	3.0119	<1H OCEAN
6568	-118.15	34.20	52.0	1786.0	306.0	1018.0	322.0	4.1518	INLAND
13285	-117.68	34.07	32.0	1775.0	314.0	1067.0	302.0	4.0375	INLAND
...	...	...	...	...	...	...	...	...	...
20519	-121.53	38.58	33.0	4988.0	1169.0	2414.0	1075.0	1.9728	INLAND
17430	-120.44	34.65	30.0	2265.0	512.0	1402.0	471.0	1.9750	NEAR OCEAN
4019	-118.49	34.18	31.0	3073.0	674.0	1486.0	684.0	4.8984	<1H OCEAN
12107	-117.32	33.99	27.0	5464.0	850.0	2400.0	836.0	4.7110	INLAND
2398	-118.91	36.79	19.0	1616.0	324.0	187.0	80.0	3.7857	INLAND

4128 rows × 9 columns

In [201…

```
y_test
```

Out[201…

```
5241     500001.0
10970    240300.0
20351    218200.0
6568     182100.0
13285    121300.0
           ...
20519     76400.0
17430    134000.0
4019     311700.0
12107    133500.0
2398      78600.0
Name: median_house_value, Length: 4128, dtype: float64
```

In [202…

```
X_test_prepared[0:5]
```

Out[202…

```
array([[ 0.59238393, -0.71074948,  0.02758786,  1.78838525,  1.16351084,
         0.68498857,  1.23217448,  2.31299771,  0.48830927, -0.07090847,
        -0.86820063,  1.        ,  0.        ,  0.        ,  0.        ,
         0.        ],
       [ 0.8571457 , -0.87445443,  0.8228579 ,  0.71842323,  0.29453231,
         0.22337528,  0.40973048,  0.38611673,  0.36310326, -0.04598303,
        -0.86028018,  1.        ,  0.        ,  0.        ,  0.        ,
         0.        ],
       [ 0.26268061, -0.66865392, -0.13146615,  0.8110161 ,  0.95417708,
         0.61865967,  1.00859747, -0.45340597, -0.17866074, -0.05936925,
        -0.01792937,  1.        ,  0.        ,  0.        ,  0.        ,
         0.        ],
       [ 0.71227605, -0.67333121,  1.85670895, -0.39128825, -0.55497332,
        -0.36013977, -0.46594615,  0.14500069,  0.04068081,  0.00561556,
        -0.64843204,  0.        ,  1.        ,  0.        ,  0.        ,
         0.        ],
       [ 0.94706479, -0.7341359 ,  0.26616887, -0.3964323 , -0.53550041,
        -0.31621928, -0.51917877,  0.08499728,  0.167383  ,  0.03769486,
        -0.56320765,  0.        ,  1.        ,  0.        ,  0.        ,
         0.        ]])
```

In [203…

```
final_predictions[0:10]
```

Out[203… array([495467.5   , 262676.7   , 235380.   , 211883.33333333,
       135516.66666667, 147776.66666667, 63540.   , 439026.9   ,
       106323.33333333, 100293.33333333])

In [204…
```
y_test.head(10)
```

Out[204…
```
5241     500001.0
10970    240300.0
20351    218200.0
6568     182100.0
13285    121300.0
20552    120600.0
19989     72300.0
17049    500001.0
13692     98900.0
13916     82600.0
Name: median_house_value, dtype: float64
```

In [ ]: