02_end_to_end_ml_learning

September 14, 2019

In [1]:
```
# To support both python 2 and python 3
from __future__ import division, print_function, unicode_literals

# Common imports
import numpy as np
import os

# to make this notebook's output stable across runs
np.random.seed(42)

# To plot pretty figures
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)

# Where to save the figures
PROJECT_ROOT_DIR = "."
CHAPTER_ID = "end_to_end_project"
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, "images", CHAPTER_ID)


def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
    os.makedirs(os.path.join(IMAGES_PATH), exist_ok=True)
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)

# Ignore useless warnings (see SciPy issue #5998)
import warnings
warnings.filterwarnings(action="ignore", message="^internal gelsd")
```

In [2]:
```
import tarfile
from six.moves import urllib
```

```
DOWNLOAD_ROOT = "https://raw.githubusercontent.com/ageron/handson-ml/master/"
HOUSING_PATH = os.path.join("datasets", "housing")
HOUSING_URL = DOWNLOAD_ROOT + "datasets/housing/housing.tgz"

def fetch_housing_data(housing_url=HOUSING_URL, housing_path=HOUSING_PATH):
    '''Downloads the file at the given path'''
    os.makedirs(housing_path, exist_ok=True)
    tgz_path = os.path.join(housing_path, "housing.tgz")
    urllib.request.urlretrieve(housing_url, tgz_path)
    housing_tgz = tarfile.open(tgz_path)
    housing_tgz.extractall(path=housing_path)
    housing_tgz.close()
```

In [3]:
```
fetch_housing_data()
```

In [4]:
```
import pandas as pd
def load_housing_data(housing_path = HOUSING_PATH):
    csv_path = os.path.join(housing_path, 'housing.csv')
    return pd.read_csv(csv_path)
```

In [5]:
```
housing = load_housing_data()
housing.head()
```

Out[5]:

	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

In [6]:
```
housing.info()
#take note of these numbers to get an idea of what the data is like
#for example, notice that total bedrooms is missing some info for some subjects.
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
longitude             20640 non-null float64
latitude              20640 non-null float64
housing_median_age    20640 non-null float64
total_rooms           20640 non-null float64
```

```
total_bedrooms        20433 non-null float64
population            20640 non-null float64
households            20640 non-null float64
median_income         20640 non-null float64
median_house_value    20640 non-null float64
ocean_proximity       20640 non-null object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

In [7]: housing['ocean_proximity'].value_counts()

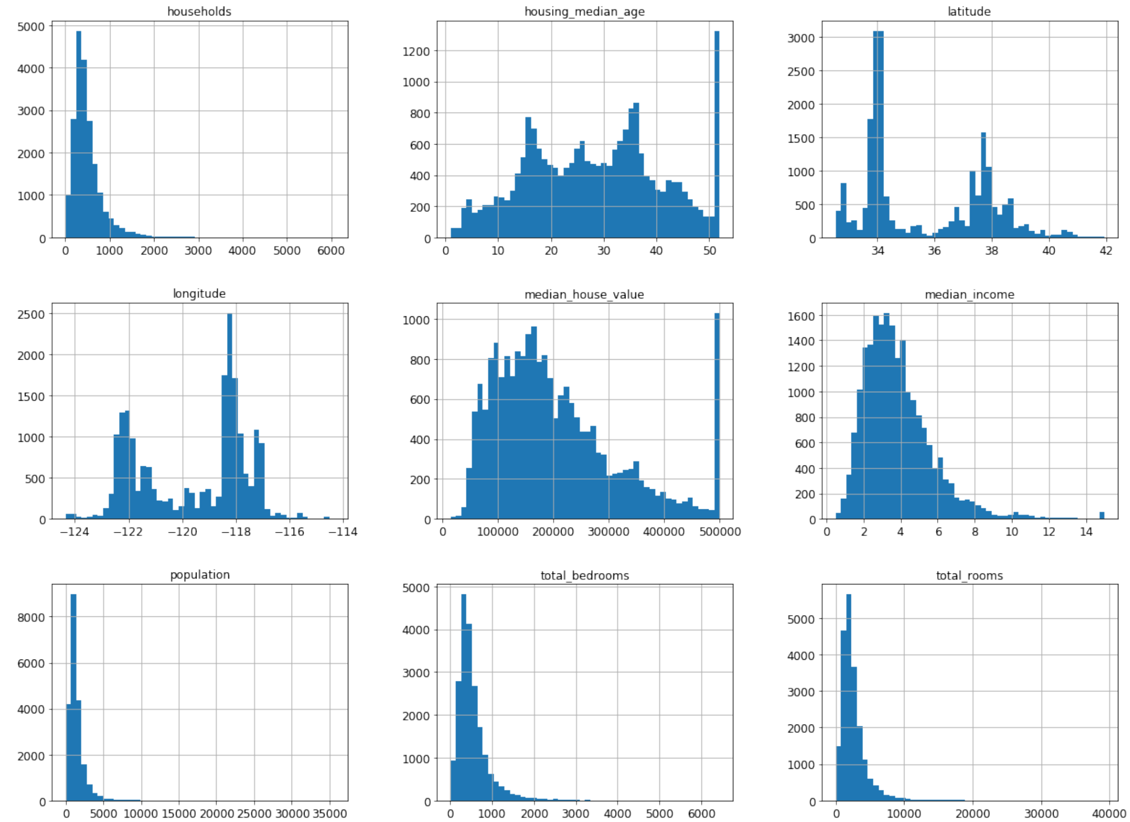
```
Out[7]: <1H OCEAN     9136
        INLAND        6551
        NEAR OCEAN    2658
        NEAR BAY      2290
        ISLAND           5
        Name: ocean_proximity, dtype: int64
```

In [8]: housing.describe()

```
Out[8]:           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
```

In [9]:
```
%matplotlib inline
import matplotlib.pyplot as plt
housing.hist(bins=50, figsize=(20,15))
plt.show()
```



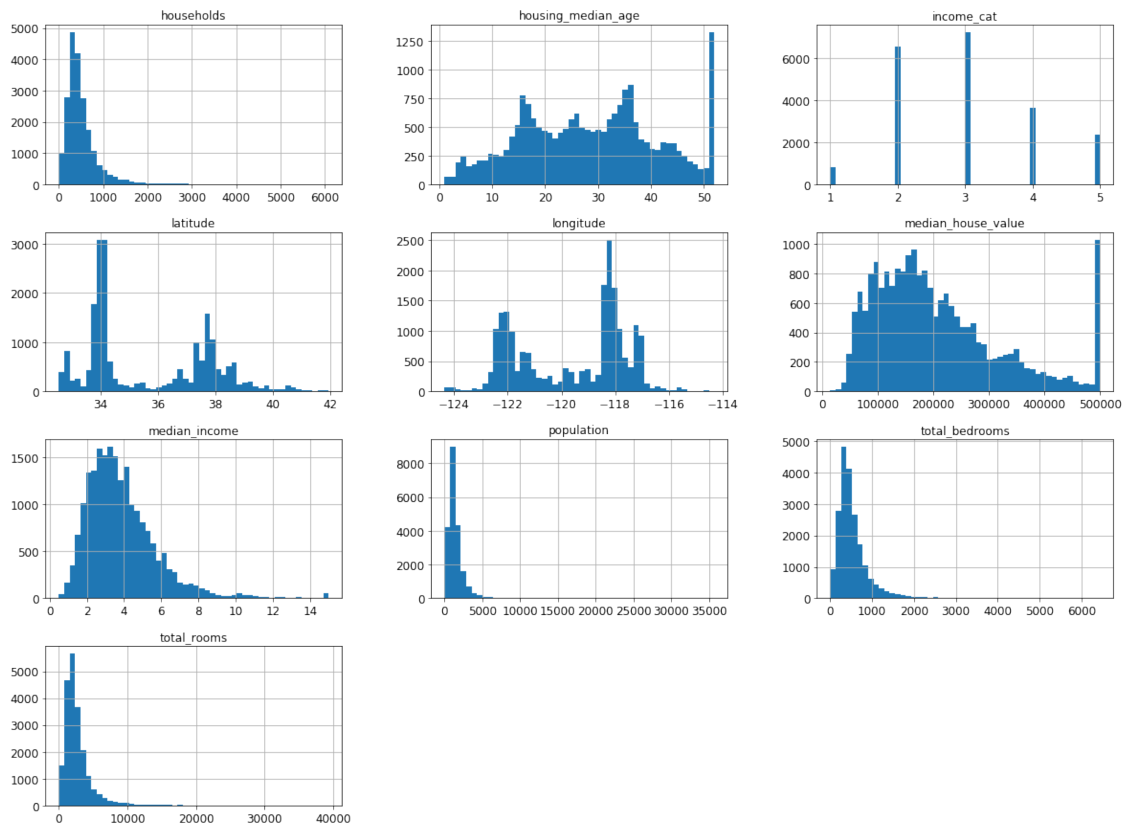
In [10]:
```
np.random.seed(42)
# you must now create a test set
'''import numpy as np
def split_train_test(data, test_ratio):
    \'''This function takes a data set and a ratio and splits your data into a test a
    shuffled_indices = np.random.permutation(len(data))
    test_set_size = int(len(data)* test_ratio)
    test_indices = shuffled_indices[:test_set_size]
    train_indices = shuffled_indeces[test_set_size:]
    return data.iloc[train_indices], data.iloc[test_indices]'''
```

Out[10]:
```
"import numpy as np\ndef split_train_test(data, test_ratio):\n    '''This function ta
```

In [11]:
```
from sklearn.model_selection import train_test_split

train_set, test_set = train_test_split(housing, test_size = 0.2, random_state = 42)
```

In [12]:
```
import numpy as np
housing['income_cat'] = np.ceil(housing['median_income'] / 1.5)
housing['income_cat'].where(housing['income_cat'] < 5, 5.0, inplace = True)
```

In [13]:
```
housing.hist(bins=50, figsize = (20, 15))
plt.show()
```



In [14]:
```
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_cat']):
    strat_train_set = housing.loc[train_index]
    strat_test_set = housing.loc[test_index]
```

In [15]:
```
housing['income_cat'].value_counts() / len(housing)
```

Out[15]:
```
3.0    0.350581
2.0    0.318847
4.0    0.176308
```

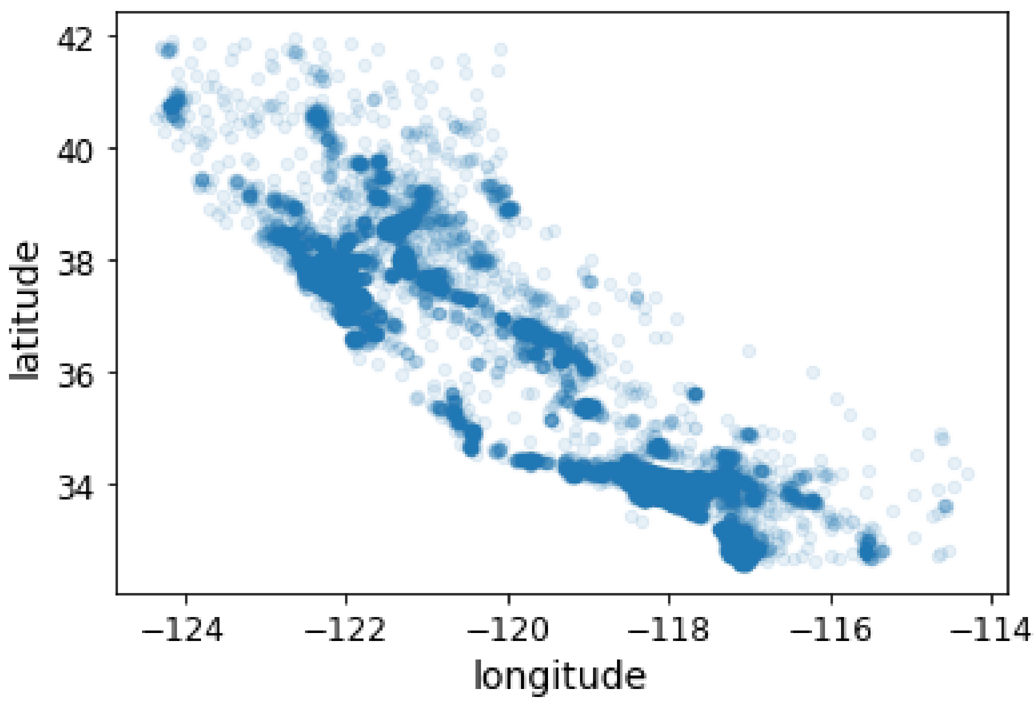
```
5.0    0.114438
1.0    0.039826
Name: income_cat, dtype: float64
```

In [16]:
```python
for set_ in (strat_train_set, strat_test_set):
    set_.drop('income_cat', axis=1, inplace=True)
```

In [17]:
```python
housing = strat_train_set.copy()
```

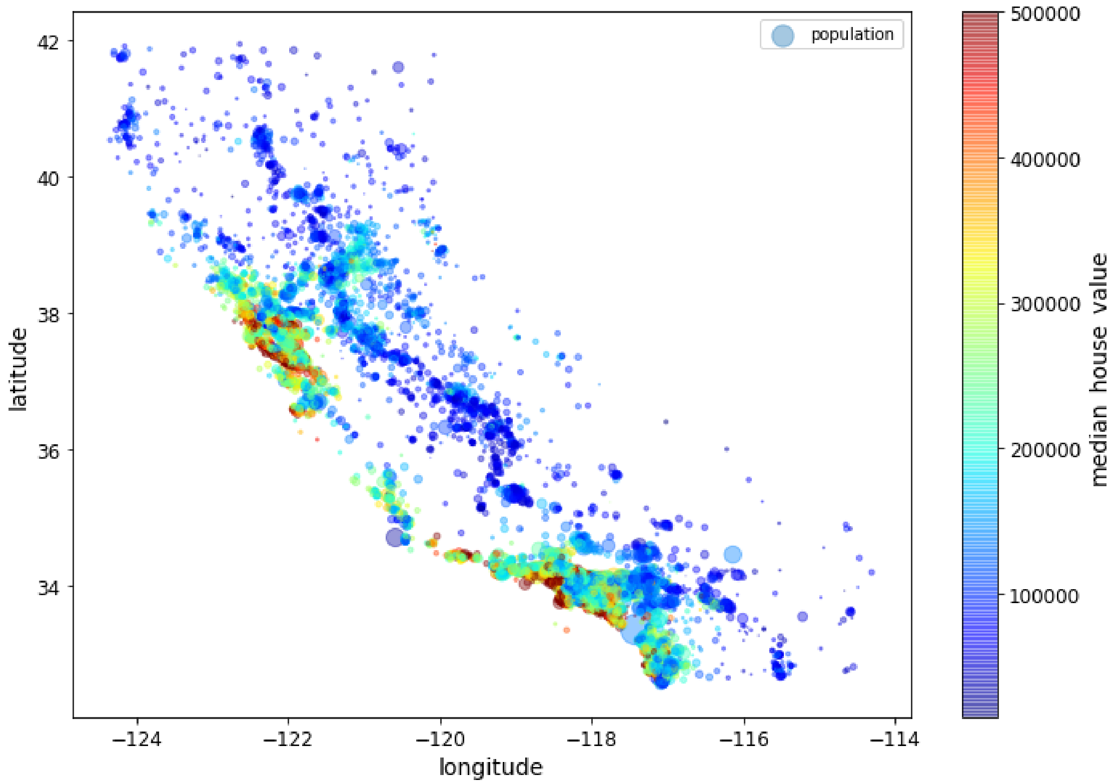
In [18]:
```python
housing.plot(kind = 'scatter', x='longitude', y = 'latitude', alpha = 0.1)
```

Out[18]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1e6449d7d08>
```

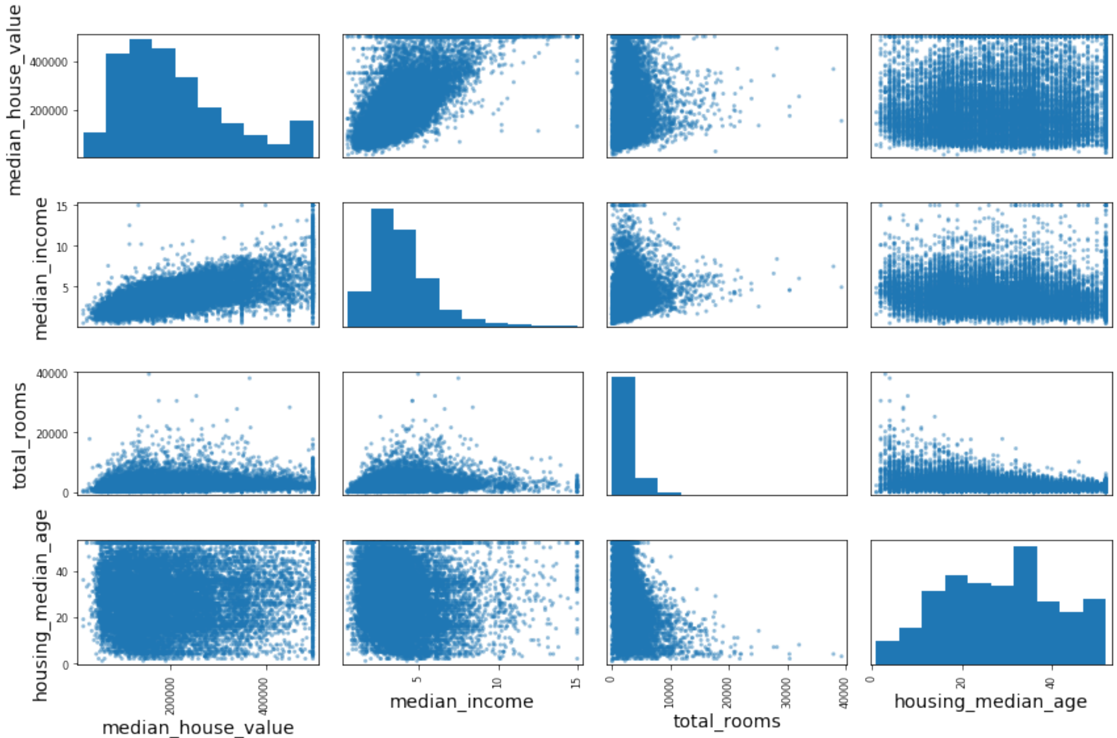


In [19]:
```python
housing.plot(kind='scatter',x='longitude',y='latitude',alpha=0.4,
    s=housing['population']/100,label='population',figsize=(10,7),
    c='median_house_value', cmap=plt.get_cmap('jet'), colorbar=True, sharex=False)
plt.legend()
save_fig('housing_prices_scatterplot')
```

```
Saving figure housing_prices_scatterplot
```

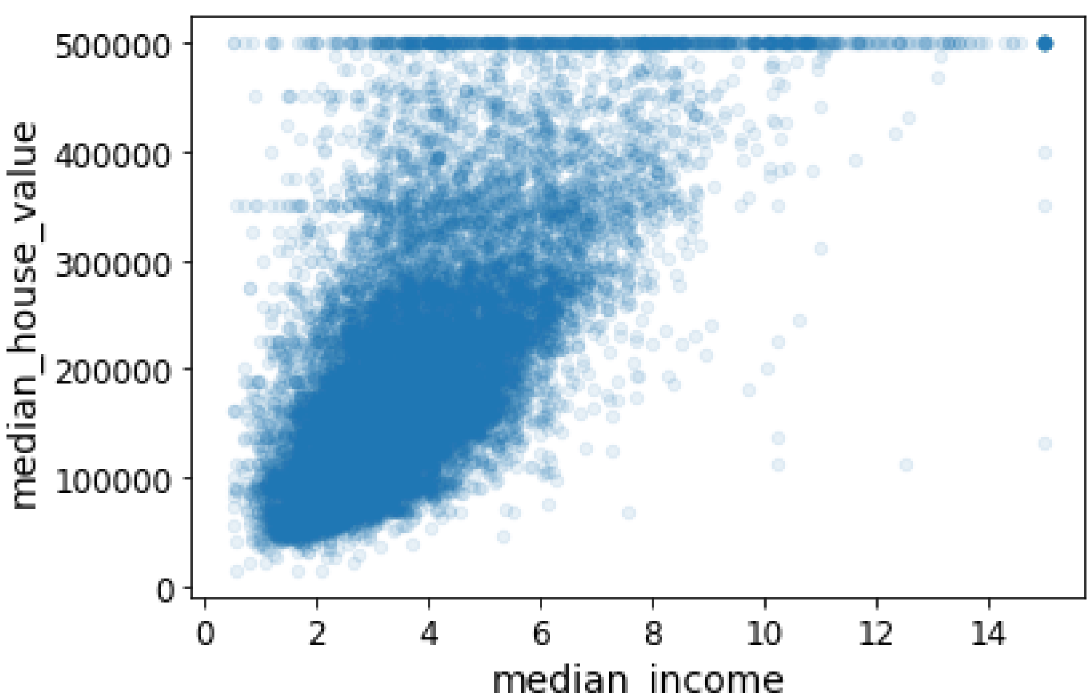



```
In [20]: #LOOKING FOR CORRELATIONS
         corr_matrix = housing.corr()

In [21]: corr_matrix['median_house_value'].sort_values(ascending = False)

Out[21]: 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 [22]: from pandas.plotting import scatter_matrix
         attributes = ['median_house_value', 'median_income', 'total_rooms',
                       'housing_median_age']
         scatter_matrix(housing[attributes], figsize = (12, 8))
         save_fig('scatter_matrix_plot')

Saving figure scatter_matrix_plot
```




```
In [23]: housing.plot(kind='scatter', x = 'median_income', y = 'median_house_value',
                      alpha = 0.1)
```

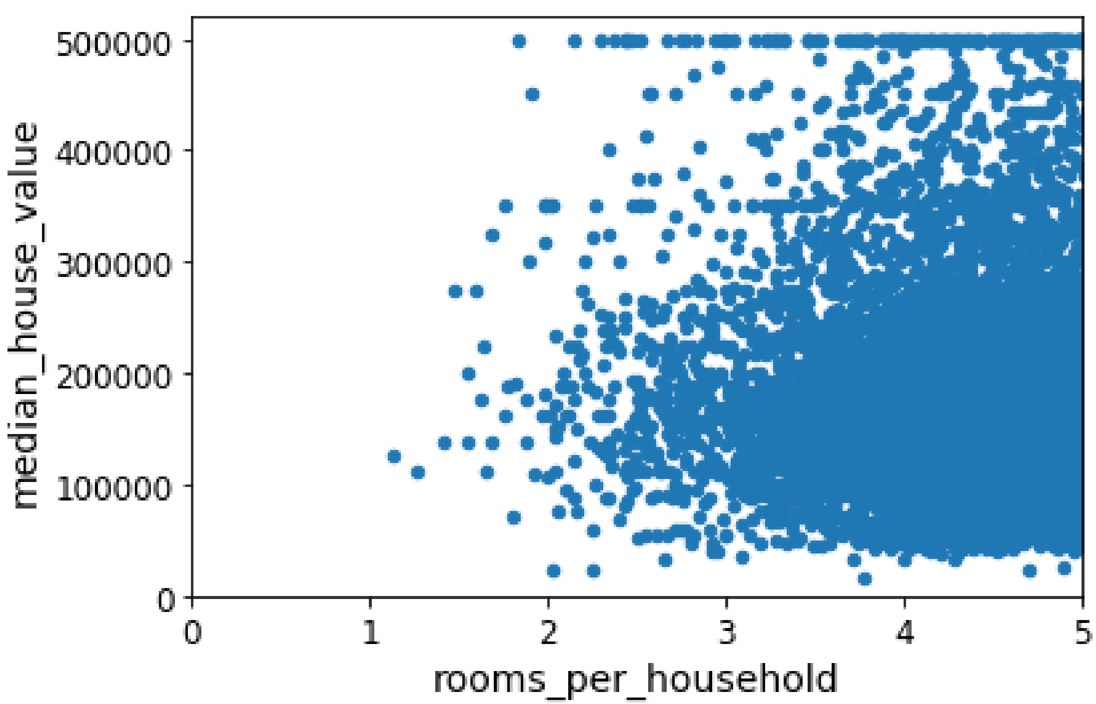
```
Out[23]: <matplotlib.axes._subplots.AxesSubplot at 0x1e6440b6548>
```



In [24]:
```
housing['rooms_per_household'] = housing['total_rooms']/housing['households']
housing["bedrooms_per_room"] = housing["total_bedrooms"]/housing["total_rooms"]
housing["population_per_household"]=housing["population"]/housing["households"]

corr_matrix = housing.corr()
corr_matrix['median_house_value'].sort_values(ascending=False)
```

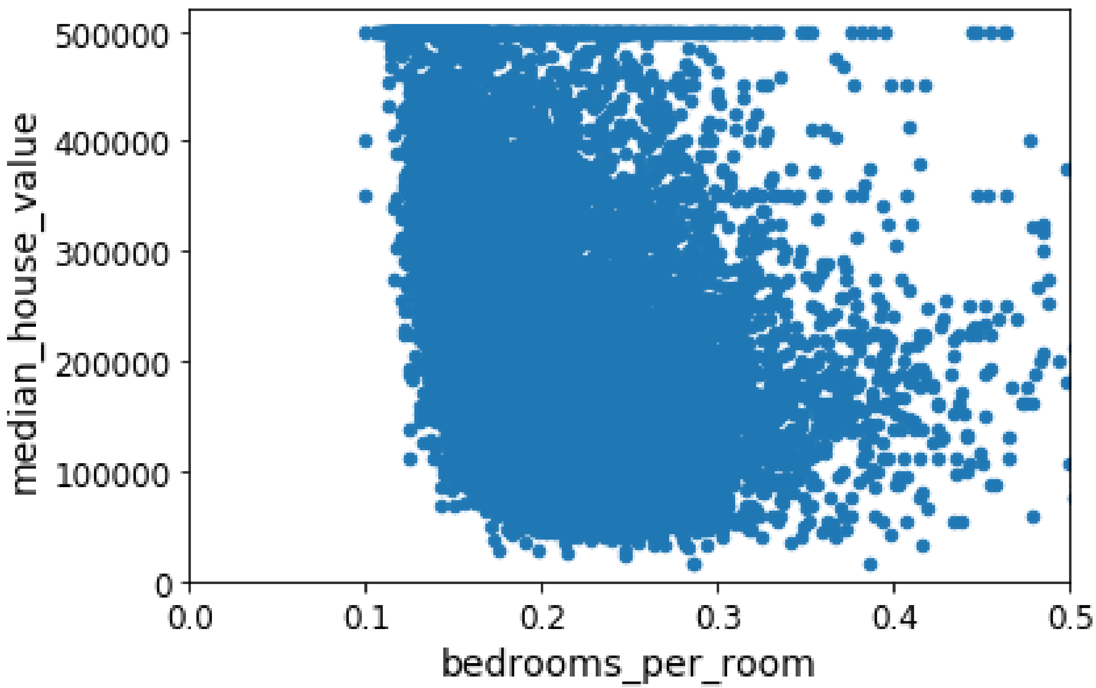
Out[24]:
```
median_house_value           1.000000
median_income                0.687160
rooms_per_household          0.146285
total_rooms                  0.135097
housing_median_age           0.114110
households                   0.064506
total_bedrooms               0.047689
population_per_household    -0.021985
population                  -0.026920
longitude                   -0.047432
latitude                    -0.142724
bedrooms_per_room           -0.259984
Name: median_house_value, dtype: float64
```

In [25]:
```
housing.plot(kind = 'scatter', x = 'rooms_per_household', y = 'median_house_value')
plt.axis([0, 5, 0, 520000])
plt.show()
```



In [26]:
```
housing.plot(kind = 'scatter', x = 'bedrooms_per_room', y = 'median_house_value')
plt.axis([0, 0.5, 0, 520000])
```

Out[26]: [0, 0.5, 0, 520000]



In [27]:
```
housing.describe()
```

Out[27]:

	longitude	latitude	housing_median_age	total_rooms	\
count	16512.000000	16512.000000	16512.000000	16512.000000	
mean	-119.575834	35.639577	28.653101	2622.728319	
std	2.001860	2.138058	12.574726	2138.458419	
min	-124.350000	32.540000	1.000000	6.000000	
25%	-121.800000	33.940000	18.000000	1443.000000	
50%	-118.510000	34.260000	29.000000	2119.500000	
75%	-118.010000	37.720000	37.000000	3141.000000	
max	-114.310000	41.950000	52.000000	39320.000000	

	total_bedrooms	population	households	median_income	\
count	16354.000000	16512.000000	16512.000000	16512.000000	
mean	534.973890	1419.790819	497.060380	3.875589	
std	412.699041	1115.686241	375.720845	1.904950	
min	2.000000	3.000000	2.000000	0.499900	

```
25%        295.000000    784.000000    279.000000        2.566775
50%        433.000000   1164.000000    408.000000        3.540900
75%        644.000000   1719.250000    602.000000        4.744475
max       6210.000000  35682.000000   5358.000000       15.000100

       median_house_value  rooms_per_household  bedrooms_per_room  \
count        16512.000000         16512.000000       16354.000000
mean        206990.920724             5.440341           0.212878
std         115703.014830             2.611712           0.057379
min          14999.000000             1.130435           0.100000
25%         119800.000000             4.442040           0.175304
50%         179500.000000             5.232284           0.203031
75%         263900.000000             6.056361           0.239831
max         500001.000000           141.909091           1.000000

       population_per_household
count              16512.000000
mean                   3.096437
std                   11.584826
min                    0.692308
25%                    2.431287
50%                    2.817653
75%                    3.281420
max                 1243.333333
```

In [28]:
```python
housing = strat_train_set.drop('median_house_value', axis = 1)
housing_labels = strat_train_set['median_house_value'].copy()
```

In [29]:
```python
try:
    from sklearn.impute import SimpleImputer # Scikit-Learn 0.20+
except ImportError:
    from sklearn.preprocessing import Imputer as SimpleImputer

imputer = SimpleImputer(strategy="median")
```

In [30]:
```python
housing_num = housing.drop('ocean_proximity', axis=1)
```

In [31]:
```python
imputer.fit(housing_num)
```

Out[31]:
```
SimpleImputer(add_indicator=False, copy=True, fill_value=None,
              missing_values=nan, strategy='median', verbose=0)
```

In [32]:
```python
imputer.statistics_
```

Out[32]:
```
array([-118.51  ,   34.26  ,   29.    , 2119.5   ,  433.    , 1164.    ,
        408.    ,    3.5409])
```

In [33]:
```python
housing_num.median().values
```

Out[33]: array([-118.51 ,   34.26 ,   29.   , 2119.5   , 433.   , 1164.   ,
         408.   ,   3.5409])

In [34]:
```
X = imputer.transform(housing_num)
```

In [35]:
```
housing_tr = pd.DataFrame(X, columns=housing_num.columns,
                          index=housing.index)
```

In [36]:
```
imputer.strategy
```

Out[36]: 'median'

In [37]:
```
housing_tr = pd.DataFrame(X, columns=housing_num.columns,
                         index=housing_num.index)
housing_tr.head()
```

Out[37]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	\
17606	-121.89	37.29	38.0	1568.0	351.0	
18632	-121.93	37.05	14.0	679.0	108.0	
14650	-117.20	32.77	31.0	1952.0	471.0	
3230	-119.61	36.31	25.0	1847.0	371.0	
3555	-118.59	34.23	17.0	6592.0	1525.0	

	population	households	median_income
17606	710.0	339.0	2.7042
18632	306.0	113.0	6.4214
14650	936.0	462.0	2.8621
3230	1460.0	353.0	1.8839
3555	4459.0	1463.0	3.0347

In [38]:
```
housing_cat = housing[['ocean_proximity']]
housing_cat.head(10)
```

Out[38]:

	ocean_proximity
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

In [ ]:

In [39]:
```
try:
    from sklearn.preprocessing import OrdinalEncoder
except ImportError:
    from future_encoders import OrdinalEncoder # Scikit-Learn < 0.20
```

In [40]:
```
ordinal_encoder = OrdinalEncoder()
housing_cat_encoded = ordinal_encoder.fit_transform(housing_cat)
housing_cat_encoded
```

Out[40]:
```
array([[0.],
       [0.],
       [4.],
       ...,
       [1.],
       [0.],
       [3.]])
```

In [41]:
```
print(ordinal_encoder.categories_)
```

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

In [42]:
```
try:
    from sklearn.preprocessing import OrdinalEncoder # just to raise an ImportError i
    from sklearn.preprocessing import OneHotEncoder
except ImportError:
    from future_encoders import OneHotEncoder # Scikit-Learn < 0.20

cat_encoder = OneHotEncoder()
housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
housing_cat_1hot
```

Out[42]:
```
<16512x5 sparse matrix of type '<class 'numpy.float64'>'
        with 16512 stored elements in Compressed Sparse Row format>
```

In [43]:
```
housing_cat_1hot.toarray()
```

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

In [44]:
```
housing.columns
```

Out[44]:
```
Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'ocean_proximity'],
      dtype='object')
```

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

Out[45]:

```
       longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
17606    -121.89     37.29                38.0       1568.0           351.0
18632    -121.93     37.05                14.0        679.0           108.0
14650    -117.20     32.77                31.0       1952.0           471.0
3230     -119.61     36.31                25.0       1847.0           371.0
3555     -118.59     34.23                17.0       6592.0          1525.0

       population  households  median_income ocean_proximity
17606       710.0       339.0         2.7042       <1H OCEAN
18632       306.0       113.0         6.4214       <1H OCEAN
14650       936.0       462.0         2.8621      NEAR OCEAN
3230       1460.0       353.0         1.8839          INLAND
3555       4459.0      1463.0         3.0347       <1H OCEAN
```

In [49]:

```python
from sklearn.base import BaseEstimator, TransformerMixin

# get the right column indices: safer than hard-coding indices 3, 4, 5, 6
rooms_ix, bedrooms_ix, population_ix, household_ix = [
    list(housing.columns).index(col)
    for col in ("total_rooms", "total_bedrooms", "population", "households")]

class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self, add_bedrooms_per_room = True): # no *args or **kwargs
        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)
housing_extra_attribs = attr_adder.transform(housing.values)
```

In [50]:

```python
housing_extra_attribs = pd.DataFrame(housing_extra_attribs,
                                     columns = list(housing.columns) + ['rooms_per_hous
                                     index = housing.index)
housing_extra_attribs.head()
```

Out[50]:

```
      longitude latitude housing_median_age total_rooms total_bedrooms  \
17606   -121.89    37.29                 38        1568            351
18632   -121.93    37.05                 14         679            108
14650    -117.2    32.77                 31        1952            471
```

```
3230    -119.61    36.31                  25         1847            371
3555    -118.59    34.23                  17         6592           1525

      population households median_income ocean_proximity rooms_per_household  \
17606        710        339        2.7042       <1H OCEAN             4.62537
18632        306        113        6.4214       <1H OCEAN             6.00885
14650        936        462        2.8621      NEAR OCEAN             4.22511
3230        1460        353        1.8839          INLAND             5.23229
3555        4459       1463        3.0347       <1H OCEAN             4.50581

      population_per_household
17606                   2.0944
18632                  2.70796
14650                  2.02597
3230                   4.13598
3555                   3.04785
```

In [51]:
```python
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

num_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('attribs_adder', FunctionTransformer(add_extra_features, validate=False)),
    ('std_scaler', StandardScaler())
])

housing_num_tr = num_pipeline.fit_transform(housing_num)
```

In [52]:
```python
housing_num_tr
```

Out[52]:
```
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 [53]:
```python
from sklearn.compose import ColumnTransformer
num_attribs = list(housing_num)
cat_attribs = ['ocean_proximity']

full_pipeline = ColumnTransformer([
```

```
            ('num', num_pipeline, num_attribs),
            ('cat', OneHotEncoder(), cat_attribs),
        ])
        housing_prepared = full_pipeline.fit_transform(housing)
        housing_prepared
```

Out[53]:
```
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 [54]:
```
housing_prepared.shape
```

Out[54]:
```
(16512, 16)
```

In [55]:
```
'''Training and Evaluating on the Training Set'''
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(housing_prepared, housing_labels)
```

Out[55]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [56]:
```
some_data = housing.iloc[:5]
some_labels = housing_labels.iloc[:5]
some_data_prepared = full_pipeline.transform(some_data)

print('Predictions:', lin_reg.predict(some_data_prepared))
```

```
Predictions: [210644.60459286 317768.80697211 210956.43331178  59218.98886849
 189747.55849879]
```

In [57]:
```
print('Labels:', list(some_labels))
```

```
Labels: [286600.0, 340600.0, 196900.0, 46300.0, 254500.0]
```

In [58]:
```
some_data_prepared
```

Out[58]:
```
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.        ]])
```

In [59]:
```python
from sklearn.metrics import mean_squared_error
housing_predictions = lin_reg.predict(housing_prepared)
lin_mse = mean_squared_error(housing_labels, housing_predictions)
lin_rmse = np.sqrt(lin_mse)
lin_rmse
```

Out[59]:
```
68628.19819848923
```

In [60]:
```python
from sklearn.tree import DecisionTreeRegressor

tree_reg = DecisionTreeRegressor()
tree_reg.fit(housing_prepared, housing_labels)
```

Out[60]:
```
DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None,
                      max_leaf_nodes=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      presort=False, random_state=None, splitter='best')
```

In [61]:
```python
housing_predictions = tree_reg.predict(housing_prepared)
tree_mse = mean_squared_error(housing_labels, housing_predictions)
tree_rmse = np.sqrt(tree_mse)
tree_rmse
```

Out[61]:
```
0.0
```

In [69]:
```python
'''
K-fold cross-validation. Randomly splits training set into 10 subsets (folds), then t
```

```
this results in an array containg the 10 evaluation scores
'''
from sklearn.model_selection import cross_val_score
scores = cross_val_score(tree_reg, housing_prepared, housing_labels,
                        scoring = 'neg_mean_squared_error', cv=10)
tree_rmse_scores = np.sqrt(-scores)
```

```
In [70]: def display_scores(scores):
             '''Shows the scores from the tree_rmse'''
             print('Scores:', scores)
             print('Mean:', scores.mean())
             print('Standard Deviation:', scores.std())
         display_scores(tree_rmse_scores)
```

```
Scores: [68752.80710562 65419.96626693 70703.77915353 70720.00059467
 71082.4189958  74858.85614429 70050.01429698 70392.31143864
 75454.60417214 69882.11508769]
Mean: 70731.68732562935
Standard Deviation: 2699.914235224334
```

```
In [73]: #Compute same scores using linear regression model
         lin_scores = cross_val_score(lin_reg, housing_prepared, housing_labels,
                                      scoring = 'neg_mean_squared_error', cv = 10)
         lin_rmse_scores = np.sqrt(-lin_scores)
         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: 2731.6740017983425
```

```
In [75]: from sklearn.ensemble import RandomForestRegressor

         forest_reg = RandomForestRegressor(n_estimators=10, random_state=42)
         forest_reg.fit(housing_prepared, housing_labels)
```

```
Out[75]: RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                               max_features='auto', max_leaf_nodes=None,
                               min_impurity_decrease=0.0, min_impurity_split=None,
                               min_samples_leaf=1, min_samples_split=2,
                               min_weight_fraction_leaf=0.0, n_estimators=10,
                               n_jobs=None, oob_score=False, random_state=42, verbose=0,
                               warm_start=False)
```

```
In [76]: housing_predictions = forest_reg.predict(housing_prepared)
         forest_mse = mean_squared_error(housing_labels, housing_predictions)
         forest_rmse = np.sqrt(forest_mse)
         forest_rmse
```

Out[76]: 21933.31414779769

In [77]:
```
forest_scores = cross_val_score(forest_reg, housing_prepared, housing_labels,
                                scoring="neg_mean_squared_error", cv=10)
forest_rmse_scores = np.sqrt(-forest_scores)
display_scores(forest_rmse_scores)
```

```
Scores: [51646.44545909 48940.60114882 53050.86323649 54408.98730149
 50922.14870785 56482.50703987 51864.52025526 49760.85037653
 55434.21627933 53326.10093303]
Mean: 52583.72407377466
Standard Deviation: 2298.353351147122
```

In [85]:
```
'''
FINE TUNE YOUR MODEL
'''
#Grid Search
from sklearn.model_selection import GridSearchCV
param_grid = [
    #try 12 (3x4) combinations of hyperparameters
    {'n_estimators': [3,10,30], 'max_features': [2,4,6,8]},
    #then try 6 (2x3) combinations with bootstrap set to false
    {'bootstrap': [False], 'n_estimators': [3,10], 'max_features': [2,3,4]}
]
forest_reg = RandomForestRegressor(random_state=42)
#train across 5 folds, thats a total of (12+6)*5=90 rounds of training
grid_search = GridSearchCV(forest_reg, param_grid, cv=5,
                          scoring = 'neg_mean_squared_error', return_train_score=True)
grid_search.fit(housing_prepared, housing_labels)
```

Out[85]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=RandomForestRegressor(bootstrap=True, criterion='mse',
                                             max_depth=None,
                                             max_features='auto',
                                             max_leaf_nodes=None,
                                             min_impurity_decrease=0.0,
                                             min_impurity_split=None,
                                             min_samples_leaf=1,
                                             min_samples_split=2,
                                             min_weight_fraction_leaf=0.0,
                                             n_estimators='warn', n_jobs=None,
                                             oob_score=False, random_state=42,
                                             verbose=0, warm_start=False),
             iid='warn', n_jobs=None,
             param_grid=[{'max_features': [2, 4, 6, 8],
                          'n_estimators': [3, 10, 30]},
                        {'bootstrap': [False], 'max_features': [2, 3, 4],
                         'n_estimators': [3, 10]}],
```

```
                 pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
                 scoring='neg_mean_squared_error', verbose=0)
```

In [86]:
```python
grid_search.best_params_
```

Out[86]:
```
{'max_features': 8, 'n_estimators': 30}
```

In [87]:
```python
grid_search.best_estimator_
```

Out[87]:
```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                      max_features=8, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=30,
                      n_jobs=None, oob_score=False, random_state=42, verbose=0,
                      warm_start=False)
```

In [88]:
```python
#hyperparameter scores combinations
cvres = grid_search.cv_results_
for mean_score, params in zip(cvres['mean_test_score'], cvres['params']):
    print(np.sqrt(-mean_score), params)
#notice the best result (lowest number) is with max feature of 8 and n_estimators as
```

```
63669.05791727153 {'max_features': 2, 'n_estimators': 3}
55627.16171305252 {'max_features': 2, 'n_estimators': 10}
53384.57867637289 {'max_features': 2, 'n_estimators': 30}
60965.99185930139 {'max_features': 4, 'n_estimators': 3}
52740.98248528835 {'max_features': 4, 'n_estimators': 10}
50377.344409590376 {'max_features': 4, 'n_estimators': 30}
58663.84733372485 {'max_features': 6, 'n_estimators': 3}
52006.15355973719 {'max_features': 6, 'n_estimators': 10}
50146.465964159885 {'max_features': 6, 'n_estimators': 30}
57869.25504027614 {'max_features': 8, 'n_estimators': 3}
51711.09443660957 {'max_features': 8, 'n_estimators': 10}
49682.25345942335 {'max_features': 8, 'n_estimators': 30}
62895.088889905004 {'bootstrap': False, 'max_features': 2, 'n_estimators': 3}
54658.14484390074 {'bootstrap': False, 'max_features': 2, 'n_estimators': 10}
59470.399594730654 {'bootstrap': False, 'max_features': 3, 'n_estimators': 3}
52725.01091081235 {'bootstrap': False, 'max_features': 3, 'n_estimators': 10}
57490.612956065226 {'bootstrap': False, 'max_features': 4, 'n_estimators': 3}
51009.51445842374 {'bootstrap': False, 'max_features': 4, 'n_estimators': 10}
```

In [89]:
```python
'''Alternatively, you can use randomized search instead of GridSearch'''
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', ranc
        rnd_search.fit(housing_prepared, housing_labels)
```

```
Out[89]: RandomizedSearchCV(cv=5, error_score='raise-deprecating',
                            estimator=RandomForestRegressor(bootstrap=True,
                                                            criterion='mse',
                                                            max_depth=None,
                                                            max_features='auto',
                                                            max_leaf_nodes=None,
                                                            min_impurity_decrease=0.0,
                                                            min_impurity_split=None,
                                                            min_samples_leaf=1,
                                                            min_samples_split=2,
                                                            min_weight_fraction_leaf=0.0,
                                                            n_estimators='warn',
                                                            n_jobs=None, oob_score=False,
                                                            random_sta...
                                                            warm_start=False),
                            iid='warn', n_iter=10, n_jobs=None,
                            param_distributions={'max_features': <scipy.stats._distn_infrastruc
                                                 'n_estimators': <scipy.stats._distn_infrastruc
                            pre_dispatch='2*n_jobs', random_state=42, refit=True,
                            return_train_score=False, scoring='neg_mean_squared_error',
                            verbose=0)
```

```
In [90]: cvres = rnd_search.cv_results_
         for mean_score, params in zip(cvres['mean_test_score'], cvres['params']):
             print(np.sqrt(-mean_score), params)
```

```
49150.657232934034 {'max_features': 7, 'n_estimators': 180}
51389.85295710133 {'max_features': 5, 'n_estimators': 15}
50796.12045980556 {'max_features': 3, 'n_estimators': 72}
50835.09932039744 {'max_features': 5, 'n_estimators': 21}
49280.90117886215 {'max_features': 7, 'n_estimators': 122}
50774.86679035961 {'max_features': 3, 'n_estimators': 75}
50682.75001237282 {'max_features': 3, 'n_estimators': 88}
49608.94061293652 {'max_features': 5, 'n_estimators': 100}
50473.57642831875 {'max_features': 3, 'n_estimators': 150}
64429.763804893395 {'max_features': 5, 'n_estimators': 2}
```

```
In [91]: '''Analyze the best models and their errors'''
         #you can gain good insight by inspecting the best models
         feature_importances = grid_search.best_estimator_.feature_importances_
         feature_importances
```

Out[91]: array([7.33442355e-02, 6.29090705e-02, 4.11437985e-02, 1.46726854e-02,
        1.41064835e-02, 1.48742809e-02, 1.42575993e-02, 3.66158981e-01,
        5.64191792e-02, 1.08792957e-01, 5.33510773e-02, 1.03114883e-02,
        1.64780994e-01, 6.02803867e-05, 1.96041560e-03, 2.85647464e-03])

In [93]:
```python
#display importance scores next to attribute names
extra_attribs = ['rooms_per_hhold', 'pop_per_hhold', 'bedrooms_per_room']
cat_encoder = full_pipeline.named_transformers_['cat']
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[93]:
```
[(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 [95]:
```python
'''Evaluate your system 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)
final_predictions = final_model.predict(X_test_prepared)

final_mse = mean_squared_error(y_test, final_predictions)
final_rmse = np.sqrt(final_mse)
```

In [96]:
```python
final_rmse
```

Out[96]: 47730.22690385927

In [ ]: