California Housing Price Prediction

download the dataset from here: https://www.kaggle.com/camnugent/california-housing-prices (https://www.kaggle.com/camnugent/california-housing-prices)

Loading the libs

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [3]:

```
housing = pd.read_csv('dataset/housing.csv')
housing.head()
```

Out[3]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	househ
0	-122.23	37.88	41.0	880.0	129.0	322.0	1
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	11
2	-122.24	37.85	52.0	1467.0	190.0	496.0	1
3	-122.25	37.85	52.0	1274.0	235.0	558.0	2
4	-122.25	37.85	52.0	1627.0	280.0	565.0	2

In [4]:

```
housing.info()
```

```
<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 [5]:

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

Out[5]:

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

In [6]:

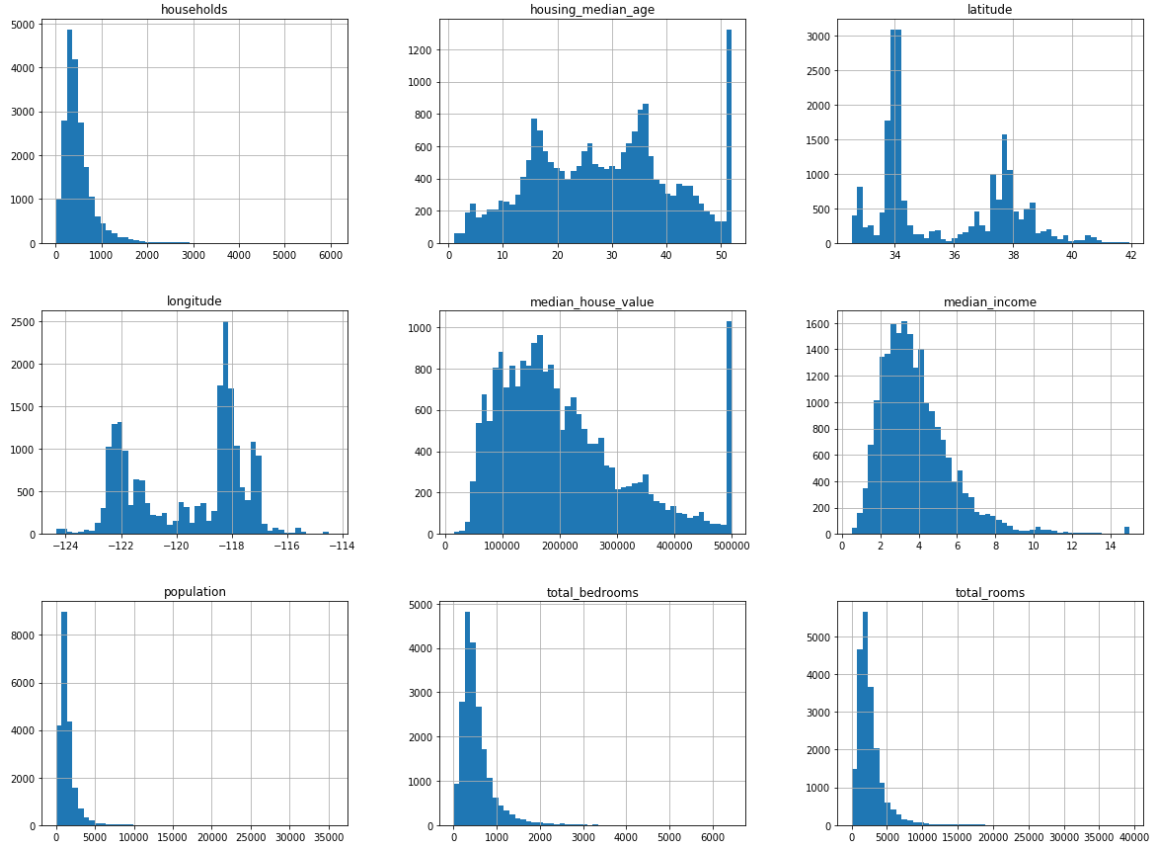
```
housing.describe()
```

Out[6]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	pc
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	2064
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	142
std	2.003532	2.135952	12.585558	2181.615252	421.385070	113
min	-124.350000	32.540000	1.000000	2.000000	1.000000	
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	78
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	116
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	172
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	3568

In [11]:

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

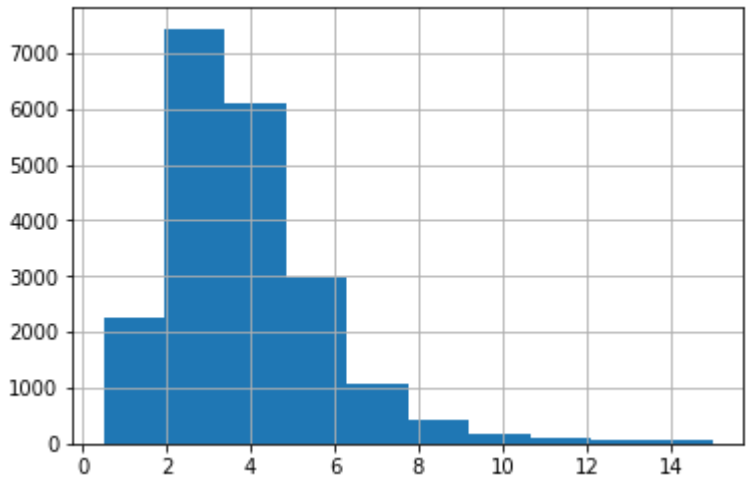


In [14]:

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

Out[14]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2129dc300b8>
```



In [23]:

```
# dividing the 1.5 to limit the number of income categories
housing['income_cat'] = np.ceil(housing['median_income'] / 1.5)

# labelling everything above 5 as 5
housing['income_cat'].where(housing['income_cat'] < 5, other=5.0, inplace=True)
```

In [25]:

```
housing.income_cat.value_counts()
```

Out[25]:

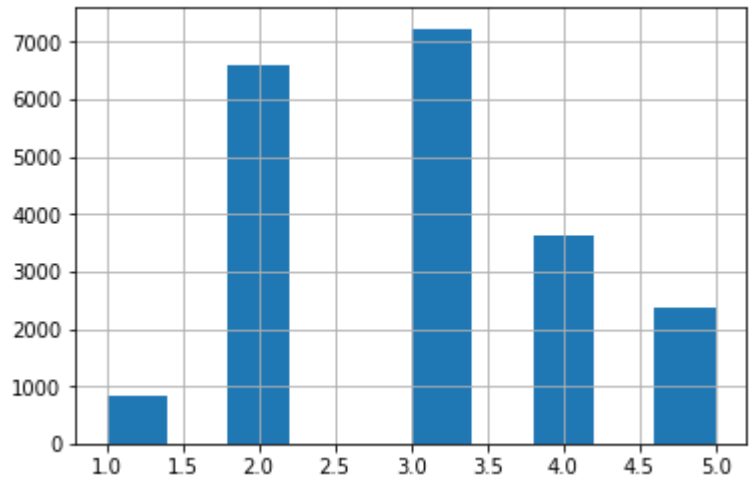
```
3.0    7236
2.0    6581
4.0    3639
5.0    2362
1.0     822
Name: income_cat, dtype: int64
```

In [26]:

```
housing['income_cat'].hist()
```

Out[26]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x212a778c6a0>
```



In [29]:

```
from sklearn.model_selection import StratifiedShuffleSplit

split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=29)

for train_idx, test_idx in split.split(housing, housing['income_cat']):
    strat_train_set = housing.loc[train_idx]
    strat_test_set = housing.loc[test_idx]
```

In [30]:

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

Out[30]:

```
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 [31]:

```
strat_train_set['income_cat'].value_counts() / len(strat_train_set)
```

Out[31]:

```
3.0    0.350594
2.0    0.318859
4.0    0.176296
5.0    0.114402
1.0    0.039850
Name: income_cat, dtype: float64
```

In [32]:

```
strat_test_set['income_cat'].value_counts() / len(strat_test_set)
```

Out[32]:

```
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
```

In [33]:

```
# lets also look at random shuffle split
from sklearn.model_selection import train_test_split

train_set, test_set = train_test_split(housing, test_size=0.2, random_state=29)
```

In [35]:

```
def income_category_proportions(data):
    return data['income_cat'].value_counts() / len(data)


compare_props = pd.DataFrame({
    "overall": income_category_proportions(housing),
    "stratified": income_category_proportions(strat_train_set),
    "random": income_category_proportions(train_set)
}).sort_index()

compare_props['rand. % err'] = 100 * compare_props['random']/compare_props['overall'] -
100
compare_props['strat. % err'] = 100 * compare_props['stratified']/compare_props['overal
l'] - 100

compare_props
```

Out[35]:

	overall	stratified	random	rand. % err	strat. % err
1.0	0.039826	0.039850	0.039123	-1.763990	0.060827
2.0	0.318847	0.318859	0.320676	0.573621	0.003799
3.0	0.350581	0.350594	0.352047	0.418049	0.003455
4.0	0.176308	0.176296	0.174964	-0.762572	-0.006870
5.0	0.114438	0.114402	0.113190	-1.090178	-0.031753

In [36]:

```
for item_set in (strat_train_set, strat_test_set):
    item_set.drop('income_cat', axis=1, inplace=True)
```

Visualizing the data

In [38]:

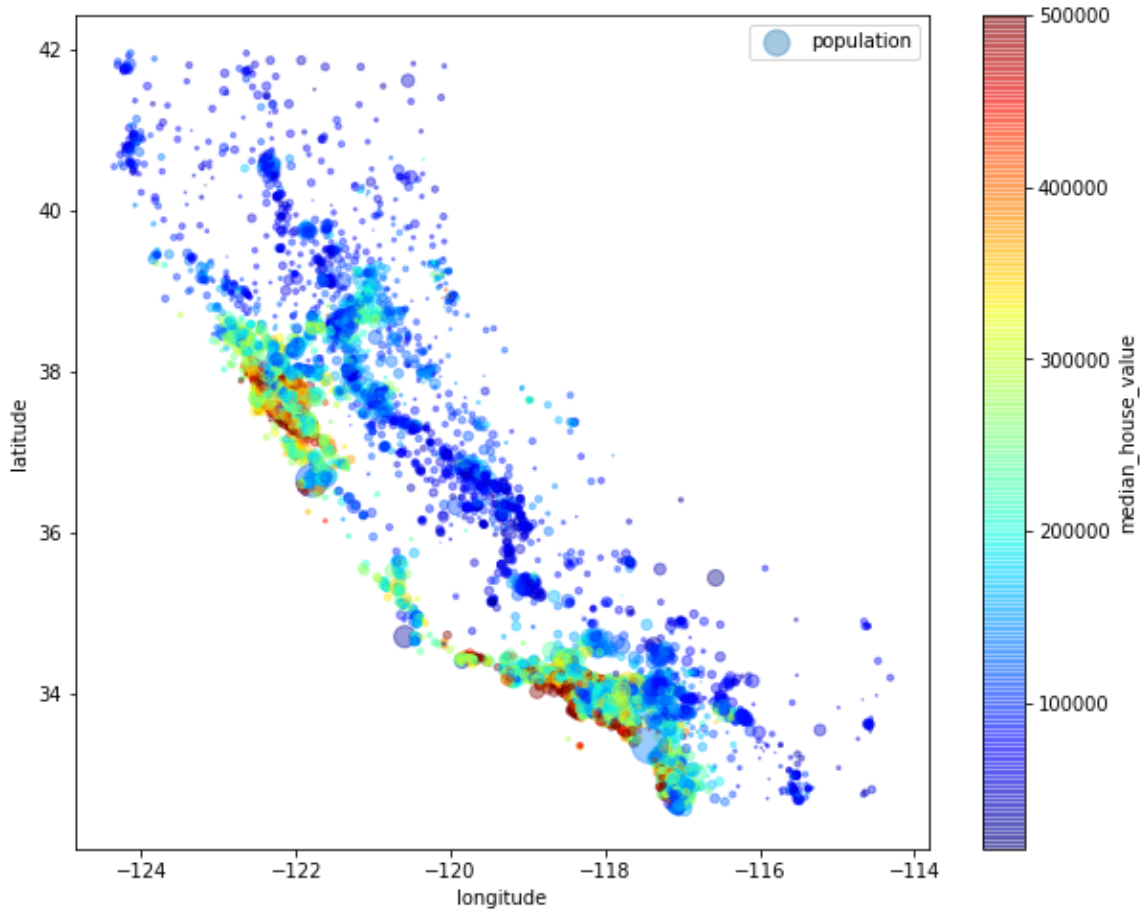
```
housing = strat_train_set.copy()
```

In [52]:

```
# 'c' is for adding coclor, 's' is for adding circles; the value of 's' represents the
 radius of the circle

housing.plot(kind='scatter', x='longitude', y='latitude', alpha=0.4,
             s=housing['population']/100, label='population', c='median_house_value',
             cmap=plt.get_cmap('jet'), colorbar=True, figsize=(10,8), sharex=False)
plt.legend()
```

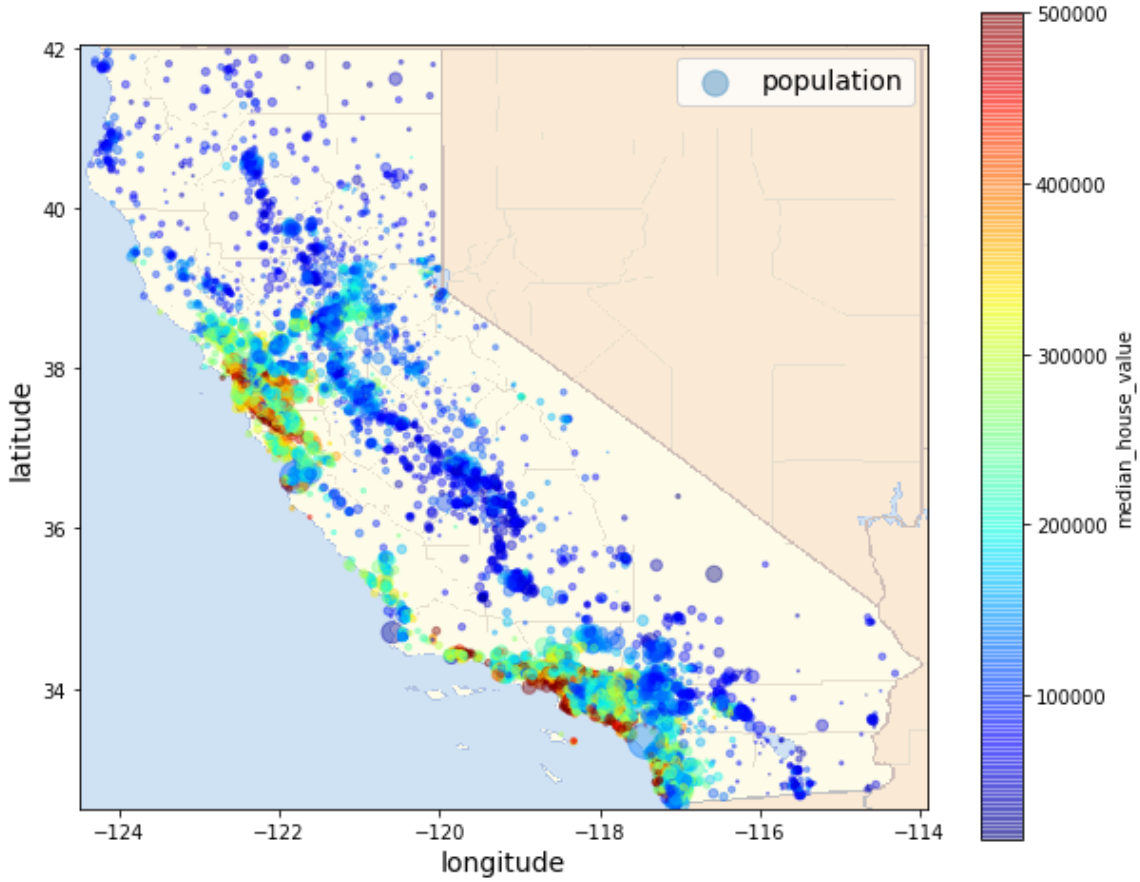
Out[52]:

```
<matplotlib.legend.Legend at 0x212aadd40b8>
```



In [50]:

```
import matplotlib.image as mpimg

housing.plot(kind='scatter', x='longitude', y='latitude', alpha=0.4,
             s=housing['population']/100, label='population', c='median_house_value',
             cmap=plt.get_cmap('jet'), colorbar=True, figsize=(10,8), sharex=False)

# # Load the png image here
california_img = mpimg.imread('images/california.png')

plt.imshow(california_img, extent= [-124.5, -113.9, 32.5, 42.05], alpha=0.5, cmap=plt.g
et_cmap('jet'))

plt.xlabel("longitude", fontsize=14)
plt.ylabel("latitude", fontsize=14)

plt.legend(fontsize=14)
plt.show()
```



Looking for Correlations

(using Pandas's Pearson's Distance Correlation equation)

In [59]:

```
corr_matrix = housing.corr()

# correlation of every other attribute with median_house_value in desc order
corr_matrix['median_house_value'].sort_values(ascending=False)
```

Out[59]:

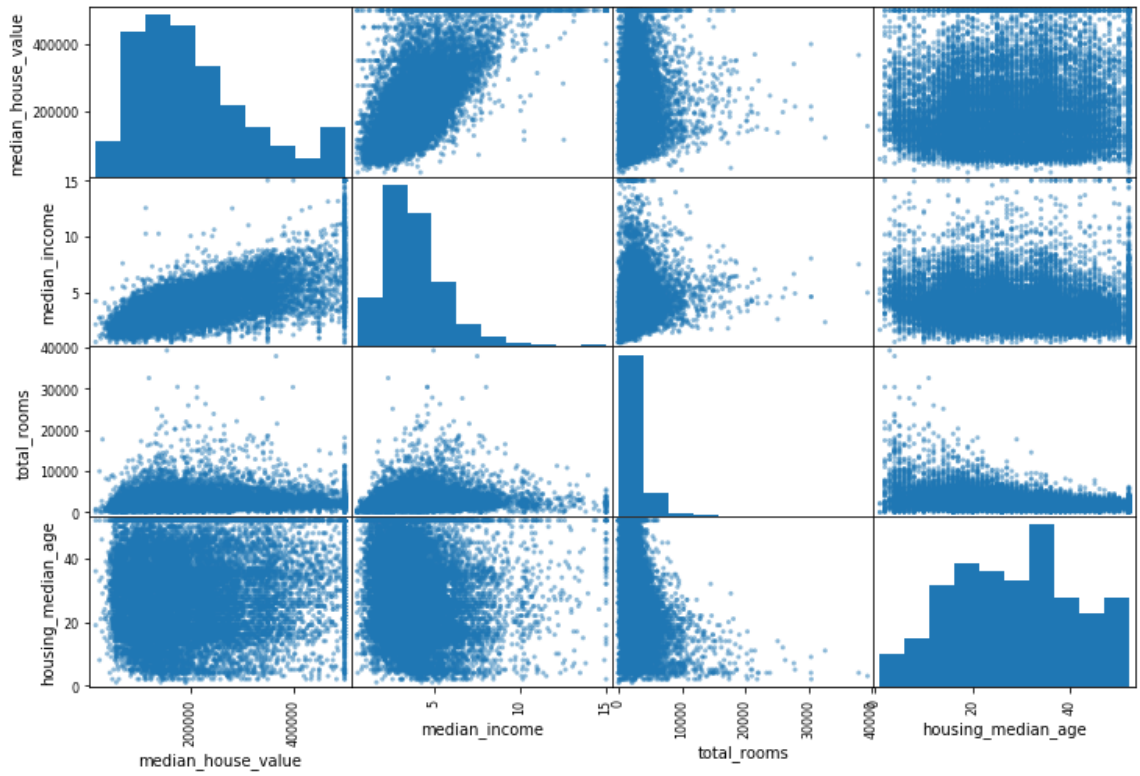
```
median_house_value    1.000000
median_income         0.691071
total_rooms           0.127306
housing_median_age    0.108483
households            0.060084
total_bedrooms        0.043921
population           -0.028341
longitude            -0.043780
latitude             -0.146422
Name: median_house_value, dtype: float64
```

its always between -1 (less correlated) and 1 (highly correlated)

other approach is to use the scatter plot in a proper 'A vs B fashion' the challenge with plotting scatter plot across all attributes is that, if you have N attributes, you'll end up creating N^2 plots

In [64]:

```
imp_attributes = ['median_house_value', 'median_income', 'total_rooms', 'housing_median
_age']

from pandas.plotting import scatter_matrix

scatter_matrix(housing[imp_attributes], figsize=(12,8))
plt.show()
```

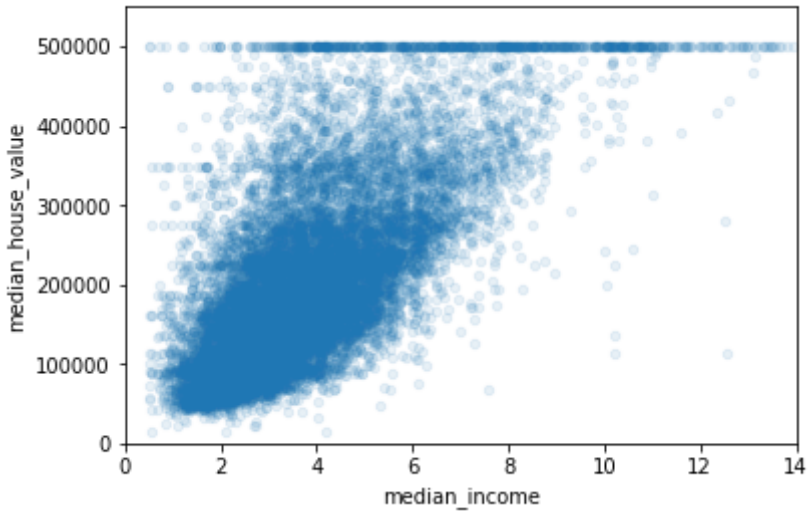


In [62]:

```
housing.plot(kind='scatter', x='median_income', y='median_house_value', alpha=0.1)
plt.axis([0, 14, 0, 550000])
```

Out[62]:

```
[0, 14, 0, 550000]
```



Feature Engineering

In [66]:

```
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']
```

In [68]:

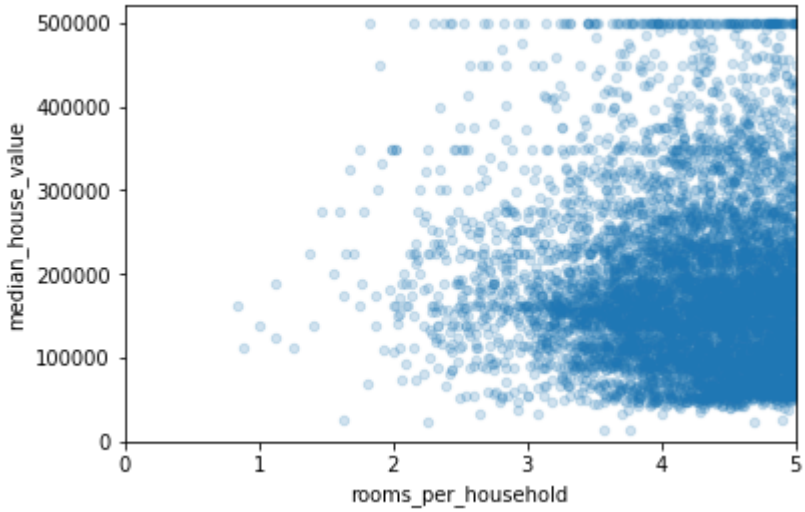
```
# looking at the corr matrix again with new features
corr_matrix = housing.corr()
corr_matrix['median_house_value'].sort_values(ascending=False)
```

Out[68]:

```
median_house_value          1.000000
median_income               0.691071
rooms_per_household         0.151804
total_rooms                 0.127306
housing_median_age          0.108483
households                  0.060084
total_bedrooms              0.043921
population_per_household   -0.021688
population                 -0.028341
longitude                  -0.043780
latitude                   -0.146422
bedrooms_per_room          -0.253572
Name: median_house_value, dtype: float64
```

In [71]:

```
housing.plot(kind='scatter', x='rooms_per_household', y='median_house_value', alpha=0.2
)
plt.axis([0, 5, 0, 520000])
plt.show()
```



Preparing the data for ML algos

In [72]:

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

In [74]:

```
housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 16512 entries, 7771 to 20194
Data columns (total 9 columns):
longitude             16512 non-null float64
latitude              16512 non-null float64
housing_median_age    16512 non-null float64
total_rooms           16512 non-null float64
total_bedrooms        16349 non-null float64
population            16512 non-null float64
households            16512 non-null float64
median_income         16512 non-null float64
ocean_proximity       16512 non-null object
dtypes: float64(8), object(1)
memory usage: 1.3+ MB
```

3 common ways of calculating averages in statistics

mean = summing up all items / total no of items
median = exact middle value when arranged in order
mode = most frequently occuring item

In [85]:

```
# when imputing your own logic of mean/median/mode
sample_incomplete_rows = housing[housing.isnull().any(axis=1)].head()

median_ = housing['total_bedrooms'].median()
sample_incomplete_rows['total_bedrooms'].fillna(median_, inplace=True)
sample_incomplete_rows
```

Out[85]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	ho
5654	-118.30	33.73	42.0	1731.0	433.0	866.0	
14930	-117.02	32.66	19.0	771.0	433.0	376.0	
9814	-121.93	36.62	34.0	2351.0	433.0	1063.0	
14986	-117.03	32.73	34.0	2061.0	433.0	1169.0	
4767	-118.37	34.03	37.0	1236.0	433.0	966.0	

In [84]:

```
# Levarging the Scikit Learn's SimpleImputer
from sklearn.impute import SimpleImputer

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

In [86]:

```
housing_num = housing.drop('ocean_proximity', axis=1)

imputer.fit(housing_num)
```

Out[86]:

```
SimpleImputer(copy=True, fill_value=None, missing_values=nan,
       strategy='median', verbose=0)
```

In [87]:

```
imputer.statistics_
```

Out[87]:

```
array([-118.5    ,   34.26   ,   29.     , 2123.     ,  433.     ,
       1159.     ,  407.     ,    3.53275])
```

In [88]:

```
housing_num.median().values
```

Out[88]:

```
array([-118.5    ,   34.26   ,   29.     , 2123.     ,  433.     ,
       1159.     ,  407.     ,    3.53275])
```

In [89]:

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

In [98]:

```
housing_tr = pd.DataFrame(X, columns=housing_num.columns)
```

In [100]:

```
# cross checking for null values
housing_tr.isnull().any()
```

Out[100]:

```
longitude             False
latitude              False
housing_median_age    False
total_rooms           False
total_bedrooms        False
population            False
households            False
median_income         False
dtype: bool
```

In [101]:

```
housing_tr.head(2)
```

Out[101]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	househ
0	-118.09	33.92	35.0	1994.0	419.0	1491.0	4
1	-122.57	37.96	52.0	3458.0	468.0	1449.0	4

handling categorical values

In [102]:

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

Out[102]:

```
7771      <1H OCEAN
9352       NEAR BAY
18657    NEAR OCEAN
4873      <1H OCEAN
12350        INLAND
18621    NEAR OCEAN
15543     <1H OCEAN
14129    NEAR OCEAN
18136     <1H OCEAN
14418    NEAR OCEAN
Name: ocean_proximity, dtype: object
```

In [103]:

```
# using Pandas factorize() method
housing_cat_encoded, housing_categories = housing_cat.factorize()
housing_cat_encoded[:10]
```

Out[103]:

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

In [104]:

```
housing_categories
```

Out[104]:

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

In [105]:

```
housing_cat_encoded.shape
```

Out[105]:

```
(16512,)
```

In [107]:

```
# using Scikit Learn's OneHotEncoder

from sklearn.preprocessing import OneHotEncoder

encoder = OneHotEncoder()
housing_cat_1hot = encoder.fit_transform(housing_cat_encoded.reshape(1, -1))
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\preprocessing\_encoder
s.py:368: FutureWarning: The handling of integer data will change in versi
on 0.22. Currently, the categories are determined based on the range [0, m
ax(values)], while in the future they will be determined based on the uniq
ue values.
If you want the future behaviour and silence this warning, you can specify
"categories='auto'".
In case you used a LabelEncoder before this OneHotEncoder to convert the c
ategories to integers, then you can now use the OneHotEncoder directly.
  warnings.warn(msg, FutureWarning)
```

In [111]:

```
housing_cat_1hot
```

Out[111]:

```
<1x16512 sparse matrix of type '<class 'numpy.float64'>'
        with 16512 stored elements in Compressed Sparse Row format>
```

In [112]:

```
# now to represent a sparse matrix call the 'toarray()' method

housing_cat_1hot.toarray()
```

Out[112]:

```
array([[1., 1., 1., ..., 1., 1., 1.]])
```

Custom Transformations

In [114]:

```
from sklearn.base import BaseEstimator, TransformerMixin

# grab columns indexes
rooms_ix, bedrooms_ix, population_ix, households_ix = 3,4,5,6

class CombinedAttributesAdder(BaseEstimator, TransformerMixin):

    def __init__(self, add_bedrooms_per_room = True):
        self.add_bedrooms_per_room = add_bedrooms_per_room

    def fit(self, X, y=None):
        return self # nothing doing here

    def transform(self, X, y=None):
        rooms_per_household = X[:, rooms_ix] / X[:, households_ix]
        population_per_household = X[:, population_ix] / X[:, households_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]
```

In [116]:

```
attr_adder = CombinedAttributesAdder(add_bedrooms_per_room=False)
housing_extra_attribs = attr_adder.transform(housing.values)
```

In [117]:

```
housing_extra_attribs = pd.DataFrame(housing_extra_attribs,
                                     columns=list(housing.columns)+['rooms_per_househol
d', 'population_per_household'])
housing_extra_attribs.head()
```

Out[117]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	househ
0	-118.09	33.92	35	1994	419	1491	
1	-122.57	37.96	52	3458	468	1449	
2	-121.96	36.97	23	4324	1034	1844	
3	-118.28	34.02	52	281	103	470	
4	-116.5	33.81	26	5032	1229	3086	

Setting up the Pipeline for all the data preprocessing

In [118]:

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

num_pipeline = Pipeline([
    ("imputer", SimpleImputer(strategy="median")),
    ("attribs_adder", CombinedAttributesAdder()),
    ("std_scaler", StandardScaler())
])

housing_num_tr = num_pipeline.fit_transform(housing_num)
housing_num_tr
```

Out[118]:

```
array([[ 0.74049299, -0.80402818,  0.50616062, ..., -0.30771122,
         0.03273077, -0.05512278],
       [-1.49391785,  1.081436  ,  1.86235125, ...,  0.75666902,
        -0.0023651 , -1.17763788],
       [-1.18967887,  0.61940394, -0.45115041, ..., -0.19550447,
        -0.08587951,  0.38012387],
       ...,
       [-1.18967887,  0.79208259,  0.58593654, ..., -0.06328319,
        -0.06658929, -0.48812906],
       [-0.09741107,  0.51673015,  1.22414389, ..., -0.43053438,
         0.07888273,  0.19240118],
       [ 0.17690276, -0.64535051, -1.00958184, ..., -0.32344572,
        -0.05235215,  0.40450624]])
```

In [119]:

```
from sklearn.base import BaseEstimator, TransformerMixin

class DFSelector(BaseEstimator, TransformerMixin):

    def __init__(self, attribute_name):
        self.attribute_name = attribute_name

    def fit(self, X, y=None):
        return self # no training at all

    def transform(self, X, y=None):
        return X[self.attribute_name].values
```

In [121]:

```
num_attribs = list(housing_num.columns)
cat_attribs = ['ocean_proximity']

num_pipeline = Pipeline([
    ("selector", DFSelector(num_attribs)),
    ("imputer", SimpleImputer(strategy="median")),
    ("attribs_adder", CombinedAttributesAdder()),
    ("std_scaler", StandardScaler())
])

cat_pipeline = Pipeline([
    ("selector", DFSelector(cat_attribs)),
    ("cat_encoder", OneHotEncoder(sparse=False))
])
```

Complete Pipeline

In [123]:

```
from sklearn.pipeline import FeatureUnion

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

In [124]:

```
housing_prepared = full_pipeline.fit_transform(housing)
housing_prepared
```

Out[124]:

```
array([[ 0.74049299, -0.80402818,  0.50616062, ...,  0.        ,
         0.        ,  0.        ],
       [-1.49391785,  1.081436  ,  1.86235125, ...,  0.        ,
         1.        ,  0.        ],
       [-1.18967887,  0.61940394, -0.45115041, ...,  0.        ,
         0.        ,  1.        ],
       ...,
       [-1.18967887,  0.79208259,  0.58593654, ...,  0.        ,
         0.        ,  0.        ],
       [-0.09741107,  0.51673015,  1.22414389, ...,  0.        ,
         0.        ,  0.        ],
       [ 0.17690276, -0.64535051, -1.00958184, ...,  0.        ,
         0.        ,  1.        ]])
```

Selecting & Training Models

In [126]:

```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(housing_prepared, housing_labels)
```

Out[126]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

In [127]:

```
# trying the full pipeline on some instances

some_data = housing.iloc[:5]
some_labels = housing_labels.iloc[:5]

some_data_prepared = full_pipeline.transform(some_data)
```

In [130]:

```
print("Prediction:", lin_reg.predict(some_data_prepared))
print("Actual Labels:", list(some_labels))
```

```
Prediction: [209526.30110297 455497.76141409 252936.22210586 173615.331279
43
 114294.56522481]
Actual Labels: [166200.0, 500001.0, 263800.0, 38800.0, 94800.0]
```

In [131]:

```
from sklearn.metrics import mean_squared_error

housing_pred = lin_reg.predict(housing_prepared)
lin_mse = mean_squared_error(housing_labels, housing_pred)
lin_rmse = np.sqrt(lin_mse)
lin_rmse
```

Out[131]:

```
67949.91466225038
```

In [133]:

```
from sklearn.tree import DecisionTreeRegressor

tree_reg = DecisionTreeRegressor()

tree_reg.fit(housing_prepared, housing_labels)
```

Out[133]:

```
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 [134]:

```
housing_pred = tree_reg.predict(housing_prepared)

tree_mse = mean_squared_error(housing_labels, housing_pred)
tree_rmse = np.sqrt(tree_mse)
tree_rmse
```

Out[134]:

```
0.0
```

Cross Validation

In [135]:

```
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 [141]:

```
def display_scores(scores):
    print("scores:", scores)
    print("mean:", scores.mean())
    print("std:", scores.std())

display_scores(tree_rmse_scores)
```

```
scores: [69581.71068363 67499.52474621 68269.93815558 71367.77580037
 69229.88525304 68379.44678941 72008.51452729 70056.71151996
 65565.56962466 71337.83670182]
mean: 69329.69138019742
std: 1884.375934997888
```

much worse than are our Linear Regression !!!!!!!

In [143]:

```
lin_scores = cross_val_score(lin_reg, housing_prepared, housing_labels, scoring="neg_mean_squared_error", cv=10)

lin_rmse = np.sqrt(-lin_scores)

display_scores(lin_rmse)
```

```
scores: [67641.22210761 69245.155892   65690.83401976 67581.651926
 66586.04760743 66937.30771561 67397.33645629 69807.64170261
 66660.63451034 74883.89423608]
mean: 68243.17261737355
std: 2500.726216291981
```

In [144]:

```
from sklearn.ensemble import RandomForestRegressor

rf_reg = RandomForestRegressor(random_state=29)
rf_reg.fit(housing_prepared, housing_labels)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:246:
FutureWarning: The default value of n_estimators will change from 10 in ve
rsion 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

Out[144]:

```
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=29, verbose=0, warm_start=False)
```

In [145]:

```
housing_pred = rf_reg.predict(housing_prepared)
rf_mse = mean_squared_error(housing_labels, housing_pred)
rf_rmse = np.sqrt(rf_mse)
rf_rmse
```

Out[145]:

```
21989.607999546064
```

In [147]:

```
rf_scores = cross_val_score(rf_reg, housing_prepared, housing_labels, scoring='neg_mean
_squared_error', cv=10)

display_scores(np.sqrt(-rf_scores))
```

```
scores: [50727.85369442 53777.59672741 50196.9006331  52163.3145675
 51773.76520866 52548.65794101 51587.29054027 51131.27756361
 51084.53618231 55110.13075575]
mean: 52010.13238140342
std: 1408.6596990564344
```

Fine Tuning our Model

In [148]:

```
from sklearn.model_selection import GridSearchCV

param_grid = [
    {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},
    {'bootstrap':[False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]}
]

rf = RandomForestRegressor()

grid_search = GridSearchCV(rf, param_grid, cv=5, scoring='neg_mean_squared_error')

grid_search.fit(housing_prepared, housing_labels)
```

Out[148]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=RandomForestRegressor(bootstrap=True, criterion='mse', ma
x_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=None, verbose=0, warm_start=Fals
e),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid=[{'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6,
8]}, {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2,
3, 4]}],
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='neg_mean_squared_error', verbose=0)
```

In [149]:

```
grid_search.best_params_
```

Out[149]:

```
{'max_features': 6, 'n_estimators': 30}
```

In [150]:

```
grid_search.best_estimator_
```

Out[150]:

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features=6, 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=None, verbose=0, warm_start=False)
```

In [152]:

```
cv_res = grid_search.cv_results_

for mean_score, params in zip(cv_res["mean_test_score"], cv_res["params"]):
    print(np.sqrt(-mean_score), params)
```

```
63406.3459390377 {'max_features': 2, 'n_estimators': 3}
54777.67088014668 {'max_features': 2, 'n_estimators': 10}
52106.0929538405 {'max_features': 2, 'n_estimators': 30}
59254.40818328368 {'max_features': 4, 'n_estimators': 3}
52274.75641438521 {'max_features': 4, 'n_estimators': 10}
49676.24596778542 {'max_features': 4, 'n_estimators': 30}
58549.205590405036 {'max_features': 6, 'n_estimators': 3}
51238.588641423754 {'max_features': 6, 'n_estimators': 10}
49435.73570498595 {'max_features': 6, 'n_estimators': 30}
58178.63258928186 {'max_features': 8, 'n_estimators': 3}
51469.50540383875 {'max_features': 8, 'n_estimators': 10}
49588.739191315115 {'max_features': 8, 'n_estimators': 30}
61282.16010333715 {'bootstrap': False, 'max_features': 2, 'n_estimators':
3}
53023.028456924345 {'bootstrap': False, 'max_features': 2, 'n_estimators':
10}
59585.81916036896 {'bootstrap': False, 'max_features': 3, 'n_estimators':
3}
51693.4924144118 {'bootstrap': False, 'max_features': 3, 'n_estimators': 1
0}
59044.766503957006 {'bootstrap': False, 'max_features': 4, 'n_estimators':
3}
51104.210360892495 {'bootstrap': False, 'max_features': 4, 'n_estimators':
10}
```

In [153]:

```
pd.DataFrame(grid_search.cv_results_)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:12
5: FutureWarning: You are accessing a training score ('split0_train_scor
e'), which will not be available by default any more in 0.21. If you need
training scores, please set return_train_score=True
  warnings.warn(*warn_args, **warn_kwargs)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:12
5: FutureWarning: You are accessing a training score ('split1_train_scor
e'), which will not be available by default any more in 0.21. If you need
training scores, please set return_train_score=True
  warnings.warn(*warn_args, **warn_kwargs)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:12
5: FutureWarning: You are accessing a training score ('split2_train_scor
e'), which will not be available by default any more in 0.21. If you need
training scores, please set return_train_score=True
  warnings.warn(*warn_args, **warn_kwargs)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:12
5: FutureWarning: You are accessing a training score ('split3_train_scor
e'), which will not be available by default any more in 0.21. If you need
training scores, please set return_train_score=True
  warnings.warn(*warn_args, **warn_kwargs)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:12
5: FutureWarning: You are accessing a training score ('split4_train_scor
e'), which will not be available by default any more in 0.21. If you need
training scores, please set return_train_score=True
  warnings.warn(*warn_args, **warn_kwargs)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:12
5: FutureWarning: You are accessing a training score ('mean_train_score'),
which will not be available by default any more in 0.21. If you need train
ing scores, please set return_train_score=True
  warnings.warn(*warn_args, **warn_kwargs)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:12
5: FutureWarning: You are accessing a training score ('std_train_score'),
which will not be available by default any more in 0.21. If you need train
ing scores, please set return_train_score=True
  warnings.warn(*warn_args, **warn_kwargs)
```

Out[153]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_features	para
0	0.078965	0.009122	0.004986	0.002090	2	
1	0.241132	0.006901	0.010985	0.001667	2	
2	0.692937	0.007394	0.030575	0.000908	2	
3	0.113710	0.001673	0.004387	0.001001	4	
4	0.414832	0.025314	0.011371	0.001018	4	
5	1.232270	0.114117	0.033507	0.004791	4	
6	0.170362	0.023262	0.004189	0.000971	6	
7	0.544535	0.011405	0.011167	0.000395	6	
8	1.570847	0.015851	0.031318	0.001345	6	
9	0.216080	0.007894	0.003589	0.000487	8	
10	0.709922	0.042854	0.011383	0.001484	8	
11	2.028893	0.043719	0.035714	0.007115	8	
12	0.114112	0.005418	0.004194	0.000750	2	
13	0.371105	0.010501	0.013023	0.001058	2	

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_features	parai
14	0.150450	0.004653	0.004014	0.000029	3	
15	0.499582	0.009120	0.012962	0.000896	3	
16	0.185389	0.004302	0.004199	0.000751	4	
17	0.610506	0.012819	0.013771	0.001705	4	

18 rows × 23 columns

In [154]:

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

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

rf_reg1 = RandomForestRegressor(random_state=29)
rf_search = RandomizedSearchCV(rf_reg1, params_distribs, cv=5, scoring='neg_mean_square
d_error', random_state=29, n_iter=10)

rf_search.fit(housing_prepared, housing_labels)
```

Out[154]:

```
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_state=29, verbose=0, warm_start=False),
          fit_params=None, iid='warn', n_iter=10, n_jobs=None,
          param_distributions={'n_estimators': <scipy.stats._distn_infrast
ructure.rv_frozen object at 0x00000212A20393C8>, 'max_features': <scipy.st
ats._distn_infrastructure.rv_frozen object at 0x00000212A5F373C8>},
          pre_dispatch='2*n_jobs', random_state=29, refit=True,
          return_train_score='warn', scoring='neg_mean_squared_error',
          verbose=0)
```

In [155]:

```
cvres = rf_search.cv_results_

for mean_score, params in zip(cvres["mean_test_score"], cv_res["params"]):
    print(np.sqrt(-mean_score), params)
```

```
48554.82808753229 {'max_features': 2, 'n_estimators': 3}
49443.37797311243 {'max_features': 2, 'n_estimators': 10}
53570.23831992085 {'max_features': 2, 'n_estimators': 30}
50902.01282842812 {'max_features': 4, 'n_estimators': 3}
53580.16560373702 {'max_features': 4, 'n_estimators': 10}
48632.80337452587 {'max_features': 4, 'n_estimators': 30}
53454.896572172365 {'max_features': 6, 'n_estimators': 3}
48524.00649818885 {'max_features': 6, 'n_estimators': 10}
48560.27239481039 {'max_features': 6, 'n_estimators': 30}
53446.573293355774 {'max_features': 8, 'n_estimators': 3}
```

In [158]:

```
rf_search.best_estimator_
```

Out[158]:

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features=6, 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=149, n_jobs=None, oob_score=False, random_state=2
9,
           verbose=0, warm_start=False)
```

In [160]:

```
feature_importances = grid_search.best_estimator_.feature_importances_
feature_importances
```

Out[160]:

```
array([7.93846578e-02, 7.39902354e-02, 4.35281067e-02, 1.70253082e-02,
       1.60008084e-02, 1.70912157e-02, 1.55047916e-02, 2.96980722e-01,
       7.05839376e-02, 1.04626185e-01, 8.37315187e-02, 1.41915600e-02,
       1.54529742e-01, 2.14093410e-04, 4.38252209e-03, 8.23459448e-03])
```

In [164]:

```
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))
```

Out[164]:

```
[(0.0002140934097777317, 'ISLAND'),
 (0.004382522087665158, 'NEAR BAY'),
 (0.008234594475386834, 'NEAR OCEAN'),
 (0.014191559987363773, '<1H OCEAN'),
 (0.015504791606575622, 'households'),
 (0.016000808427194247, 'total_bedrooms'),
 (0.017025308227107107, 'total_rooms'),
 (0.017091215736068994, 'population'),
 (0.043528106686719616, 'housing_median_age'),
 (0.07058393759434926, 'rooms_per_hhold'),
 (0.07399023538269749, 'latitude'),
 (0.07938465778607579, 'longitude'),
 (0.0837315187041208, 'bedrooms_per_room'),
 (0.10462618541413587, 'pop_per_hhold'),
 (0.1545297424523785, 'INLAND'),
 (0.2969807220223833, 'median_income')]
```

In [165]:

```
full_pipeline_with_predictions = Pipeline([
    ("data_prep", full_pipeline),
    ("linear", LinearRegression())
])

full_pipeline_with_predictions.fit(housing, housing_labels)
full_pipeline_with_predictions.predict(some_data)
```

Out[165]:

```
array([209526.30110297, 455497.76141409, 252936.22210586, 173615.33127943,
       114294.56522481])
```

In [166]:

```
my_model = full_pipeline_with_predictions

# saving the model
from sklearn.externals import joblib
# saving
joblib.dump(my_model, "my_model.pkl")

# loading the saved model
my_model_loaded = joblib.load("my_model.pkl")
```

In [ ]:

```

```