1.加载数据集

In [ ]:
```
import os
import tarfile
import numpy as np
import pandas as pd
from six.moves import urllib
path = "D:\Desktop\机器学习平台\实验3\housing.csv"
housing = pd.read_csv(path)
```

快速查看数据结构

In [2]:
```
#显示数据集前五行
housing.head()
```

Out[2]:

	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 [3]:
```
#使用 info() 获取数据集的简单描述。包括总行数、每个属性的类型和非空值的数量
housing.info()
```

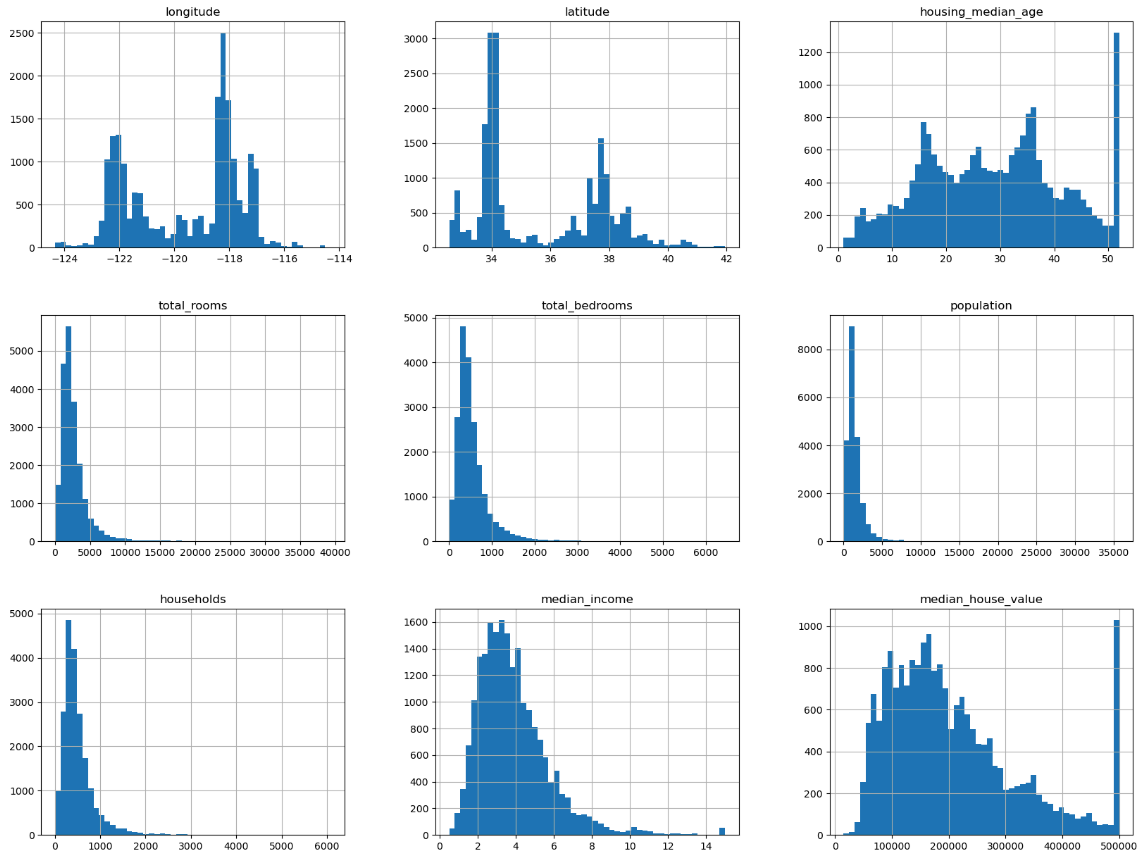
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 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]:
```
#使用 describe() 获取数值属性的描述，注意统计时的空值会被忽略。
housing.describe()
```

Out[4]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	p
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 [5]:
```
#绘制每个数值的直方图。直方图横轴表示数值范围，纵轴表示实例数量。
import matplotlib.pyplot as plt
housing.hist(bins=50, figsize=(20,15))
plt.show()
```



2. 划分测试集

In [6]:
```
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]
```

In [7]:
```
import hashlib
def test_set_check(identifier, test_ratio, hash):
    return hash(np.int64(identifier)).digest()[-1] < 256 * test_ratio
def split_train_test_by_id(data, test_ratio, id_column, hash=hashlib.md5):
    ids = data[id_column]
    in_test_set = ids.apply(lambda id_: test_set_check(id_, test_ratio, hash))
    return data.loc[~in_test_set], data.loc[in_test_set]
```

In [8]:
```
#使用索引行作为ID
housing_with_id = housing.reset_index() # adds an `index` column
train_set, test_set = split_train_test_by_id(housing_with_id, 0.2, "index")
```

In [9]:
```
#将纬度、经度结合成一个ID
housing_with_id["id"] = housing["longitude"] * 1000 + housing["latitude"]
train_set, test_set = split_train_test_by_id(housing_with_id, 0.2, "id")
```

In [10]:
```
from sklearn.model_selection import train_test_split
train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)
```

In [11]:
```
#收入数据分层
housing["income_cat"] = np.ceil(housing["median_income"] / 1.5)
housing["income_cat"].where(housing["income_cat"] < 5, 5.0, inplace=True)
```

In [12]:
```
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 [13]:
```
#查看收入分类比例:
housing["income_cat"].value_counts() / len(housing)
```

Out[13]:
```
income_cat
3.0     0.350581
2.0     0.318847
4.0     0.176308
5.0     0.114438
1.0     0.039826
Name: count, dtype: float64
```

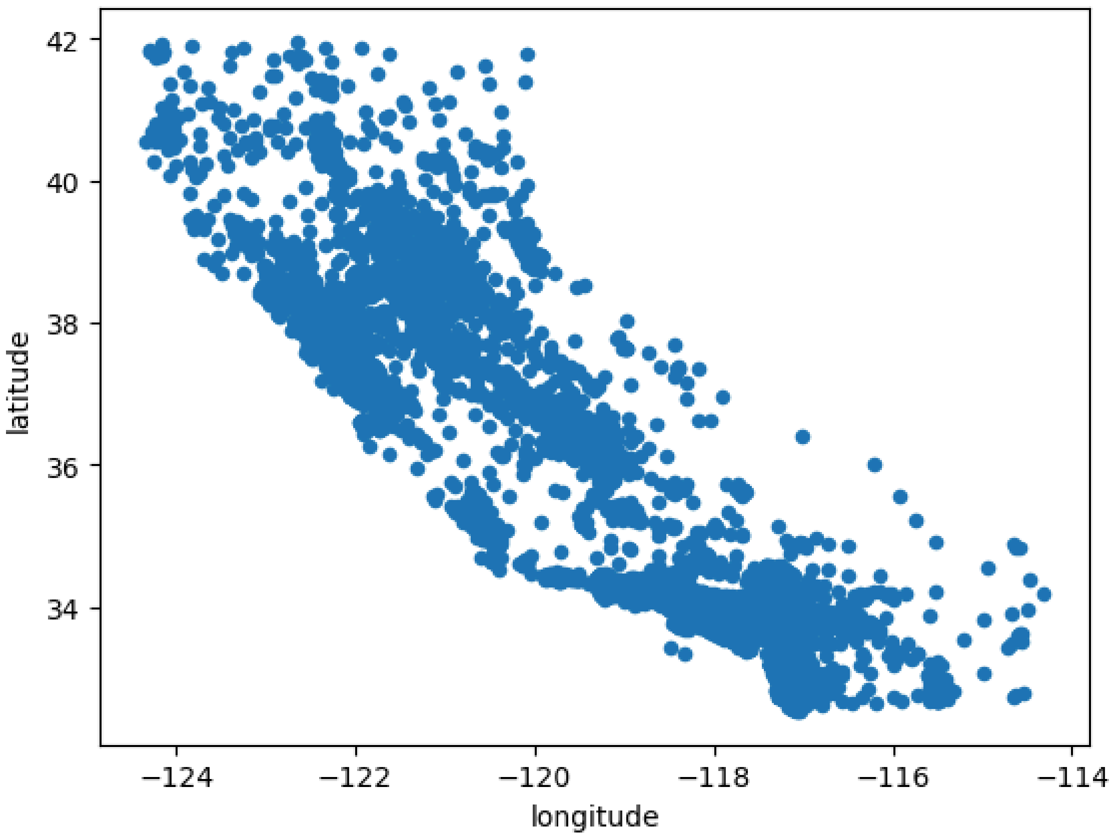
In [14]:
```
#删除 income_cat 属性，使数据回到初始状态
for set in (strat_train_set, strat_test_set):
    set.drop(["income_cat"], axis=1, inplace=True)
```

3.数据探索和可视化

In [15]:
```
housing = strat_train_set.copy()
```

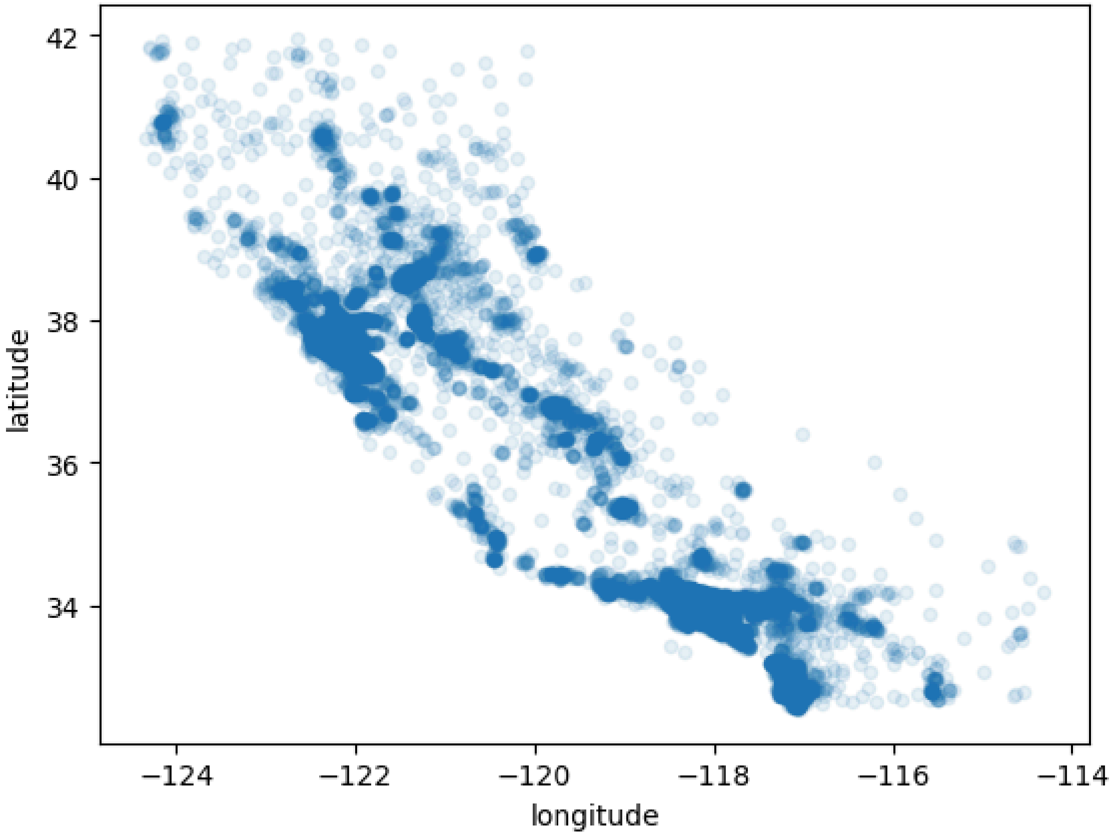
In [16]:
```
#地理可视化
housing.plot(kind="scatter", x="longitude", y="latitude")
```

Out[16]: <Axes: xlabel='longitude', ylabel='latitude'>



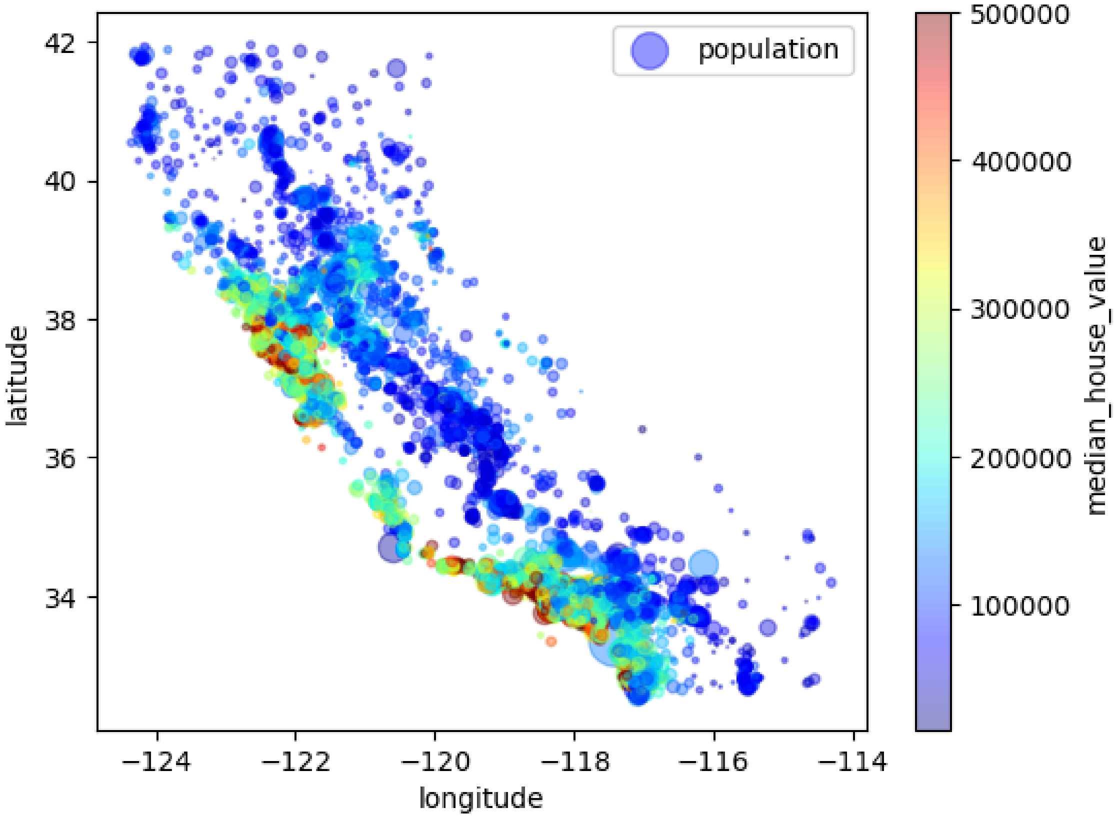
In [17]:
```
housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.1)
```

Out[17]: <Axes: xlabel='longitude', ylabel='latitude'>



In [18]:
```
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,
)
plt.legend()
```

Out[18]: <matplotlib.legend.Legend at 0x1a6eb35d910>



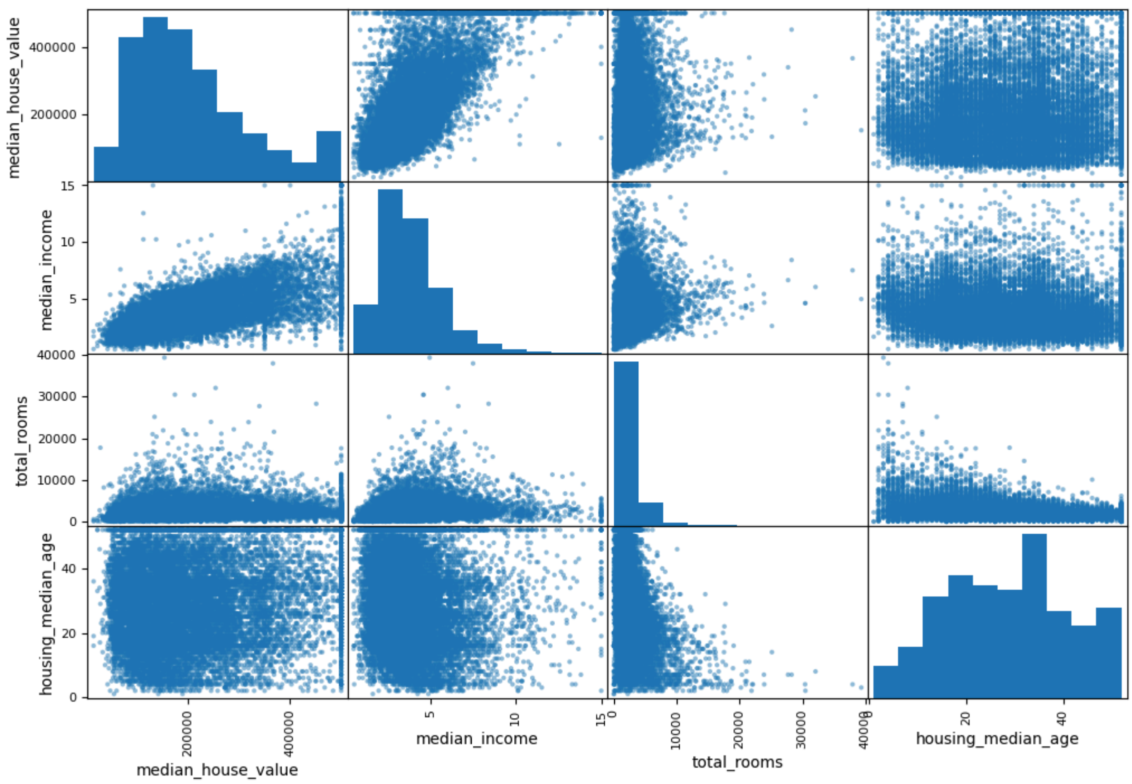
In [19]:
```
#直接对信息进行归一化、标准化或机器学习
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
knn =  KNeighborsClassifier()
housing = pd.get_dummies(housing)
#再进行查找关联
corr_matrix = housing.corr()
```

In [20]:
```
corr_matrix["median_house_value"].sort_values(ascending=False)
```

Out[20]:
```
median_house_value            1.000000
median_income                 0.687151
ocean_proximity_<1H OCEAN     0.259521
ocean_proximity_NEAR BAY      0.158691
ocean_proximity_NEAR OCEAN    0.137332
total_rooms                   0.135140
housing_median_age            0.114146
households                    0.064590
total_bedrooms                0.047781
ocean_proximity_ISLAND        0.013708
population                   -0.026882
longitude                    -0.047466
latitude                     -0.142673
ocean_proximity_INLAND       -0.482853
Name: median_house_value, dtype: float64
```

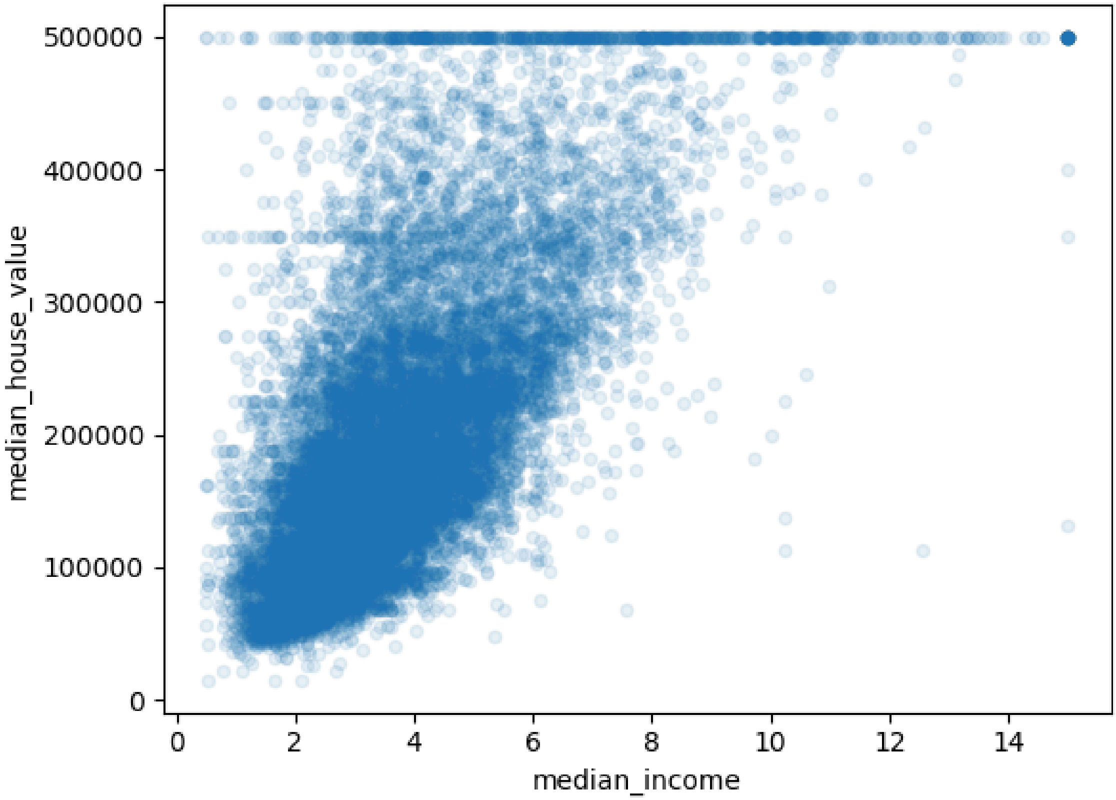
In [21]:
```
from pandas.plotting import scatter_matrix
attributes = ["median_house_value", "median_income", "total_rooms",
"housing_median_age"]
scatter_matrix(housing[attributes], figsize=(12, 8))
```

Out[21]:
```
array([[<Axes: xlabel='median_house_value', ylabel='median_house_value'>,
        <Axes: xlabel='median_income', ylabel='median_house_value'>,
        <Axes: xlabel='total_rooms', ylabel='median_house_value'>,
        <Axes: xlabel='housing_median_age', ylabel='median_house_value'>],
       [<Axes: xlabel='median_house_value', ylabel='median_income'>,
        <Axes: xlabel='median_income', ylabel='median_income'>,
        <Axes: xlabel='total_rooms', ylabel='median_income'>,
        <Axes: xlabel='housing_median_age', ylabel='median_income'>],
       [<Axes: xlabel='median_house_value', ylabel='total_rooms'>,
        <Axes: xlabel='median_income', ylabel='total_rooms'>,
        <Axes: xlabel='total_rooms', ylabel='total_rooms'>,
        <Axes: xlabel='housing_median_age', ylabel='total_rooms'>],
       [<Axes: xlabel='median_house_value', ylabel='housing_median_age'>,
        <Axes: xlabel='median_income', ylabel='housing_median_age'>,
        <Axes: xlabel='total_rooms', ylabel='housing_median_age'>,
        <Axes: xlabel='housing_median_age', ylabel='housing_median_age'>]],
      dtype=object)
```



In [22]:
```
#锁定收入中位数
housing.plot(kind="scatter", x="median_income",y="median_house_value",alpha=0.1)
```

Out[22]: <Axes: xlabel='median_income', ylabel='median_house_value'>



In [23]:
```
#属性组合试验
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 [24]:
```
corr_matrix = housing.corr()
```

In [25]:
```
corr_matrix["median_house_value"].sort_values(ascending=False)
```

Out[25]:
```
median_house_value            1.000000
median_income                 0.687151
ocean_proximity_<1H OCEAN     0.259521
ocean_proximity_NEAR BAY      0.158691
rooms_per_household           0.146255
ocean_proximity_NEAR OCEAN    0.137332
total_rooms                   0.135140
housing_median_age            0.114146
households                    0.064590
total_bedrooms                0.047781
ocean_proximity_ISLAND        0.013708
population_per_household     -0.021991
population                   -0.026882
longitude                    -0.047466
latitude                     -0.142673
bedrooms_per_room            -0.259952
ocean_proximity_INLAND       -0.482853
Name: median_house_value, dtype: float64
```

In [26]:
```
housing = strat_train_set.drop("median_house_value", axis=1)
housing_labels = strat_train_set["median_house_value"].copy()
```

数据清洗

In [27]:
```
median = housing["total_bedrooms"].median()
housing["total_bedrooms"].fillna(median)
```

Out[27]:
```
12655     797.0
15502     855.0
2908      310.0
14053     519.0
20496     646.0
          ...
15174    1231.0
12661    1422.0
19263     166.0
19140     580.0
19773     222.0
Name: total_bedrooms, Length: 16512, dtype: float64
```

In [28]:
```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy="median")
```

In [29]:
```
housing_num = housing.drop("ocean_proximity", axis=1)
```

In [30]:
```
#处理缺失值
imputer.fit(housing_num)
```

Out[30]:
```
SimpleImputer(strategy='median')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [31]:
```
imputer.statistics_
```

Out[31]:
```
array([-118.51    ,   34.26    ,   29.     , 2119.      ,  433.      ,
        1164.     ,  408.      ,    3.54155])
```

In [32]:
```
housing_num.median().values
```

Out[32]:
```
array([-118.51    ,   34.26    ,   29.     , 2119.      ,  433.      ,
        1164.     ,  408.      ,    3.54155])
```

In [33]:
```
X = imputer.transform(housing_num)
```

In [34]:
```
housing_tr = pd.DataFrame(X, columns=housing_num.columns)
```

处理文本和类别属性

In [35]:
```
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
housing_cat = housing["ocean_proximity"]
housing_cat_encoded = encoder.fit_transform(housing_cat)
housing_cat_encoded
```

Out[35]:
```
array([1, 4, 1, ..., 0, 0, 1])
```

In [36]:
```
print(encoder.classes_)
```

```
['<1H OCEAN' 'INLAND' 'ISLAND' 'NEAR BAY' 'NEAR OCEAN']
```

In [37]:
```
from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder()
housing_cat_1hot = encoder.fit_transform(housing_cat_encoded.reshape(-1,1))
housing_cat_1hot
```

Out[37]:
```
<16512x5 sparse matrix of type '<class 'numpy.float64'>'
	with 16512 stored elements in Compressed Sparse Row format>
```

In [38]:
```
housing_cat_1hot.toarray()
```

Out[38]:
```
array([[0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 1.],
       [0., 1., 0., 0., 0.],
       ...,
       [1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.]])
```

In [39]:
```
from sklearn.preprocessing import LabelBinarizer
encoder = LabelBinarizer()
housing_cat_1hot = encoder.fit_transform(housing_cat)
housing_cat_1hot
```

Out[39]:
```
array([[0, 1, 0, 0, 0],
       [0, 0, 0, 0, 1],
       [0, 1, 0, 0, 0],
       ...,
       [1, 0, 0, 0, 0],
       [1, 0, 0, 0, 0],
       [0, 1, 0, 0, 0]])
```

In [40]:
```
#添加组合后的属性
from sklearn.base import BaseEstimator,TransformerMixin
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
    def transform(self , X):
        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_
        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 [41]:
```
col_names = "total_rooms", "total_bedrooms", "population", "households"
rooms_ix, bedrooms_ix, population_ix, households_ix = [
    housing.columns.get_loc(c) for c in col_names] # get the column indices

housing_extra_attribs = pd.DataFrame(
    housing_extra_attribs,
    columns=list(housing.columns)+["rooms_per_household", "population_per_household"
    index=housing.index)
housing_extra_attribs.head()
```

Out[41]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	ho
12655	-121.46	38.52	29.0	3873.0	797.0	2237.0	
15502	-117.23	33.09	7.0	5320.0	855.0	2015.0	
2908	-119.04	35.37	44.0	1618.0	310.0	667.0	
14053	-117.13	32.75	24.0	1877.0	519.0	898.0	
20496	-118.7	34.28	27.0	3536.0	646.0	1837.0	

In [42]:
```
from sklearn.pipeline import Pipeline
from sklearn.pipeline import Pipeline
num_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy="median")),
    ('attribs_adder', CombinedAttributesAdder()),
    ('std_scaler', StandardScaler()),
    ])
housing_num_tr = num_pipeline.fit_transform(housing_num)
```

In [43]:
```
#将所有转换应用到房屋数据
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
housing_prepared.shape
```

Out[43]: (16512, 16)

In [44]:
```
#训练线性回归模型。
from sklearn.linear_model import LinearRegression

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

#训练集实例预测
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))

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
```

```
Predictions: [ 85657.90192014 305492.60737488 152056.46122456 186095.70946094
 244550.67966089]
```

Out[44]: 68627.87390018745

In [45]:
```
from sklearn.tree  import DecisionTreeRegressor
tree_reg = DecisionTreeRegressor()
tree_reg.fit(housing_prepared, housing_labels)
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[45]: 0.0

In [46]:
```
#交叉验证 过拟合
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)

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

display_scores(tree_rmse_scores)

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: [72883.37009138 70946.13064024 67844.15113188 71555.45110584
 69966.79721016 78100.55559151 71276.92856324 72536.38187821
 68843.70207303 71310.68074398]
Mean: 71526.4149029461
Standard deviation: 2640.3686487006007
Scores: [71762.76364394 64114.99166359 67771.17124356 68635.19072082
 66846.14089488 72528.03725385 73997.08050233 68802.33629334
 66443.28836884 70139.79923956]
Mean: 69104.07998247063
Standard deviation: 2880.328209818069
```

In [47]:
```
#随机森林
from sklearn.ensemble import RandomForestRegressor
forest_reg = RandomForestRegressor()
forest_reg.fit(housing_prepared,housing_labels)

housing_predictions = forest_reg.predict(housing_prepared)
forest_mse = mean_squared_error(housing_labels, housing_predictions)
forest_rmse = np.sqrt(forest_mse)
forest_rmse
from sklearn.model_selection import cross_val_score

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: [51697.47042494 49047.56653652 46938.05405788 52091.96077422
 46889.21594106 51496.65060002 52470.85183526 49714.67986553
 48325.39294755 53814.88108894]
Mean: 50248.67240719276
Standard deviation: 2288.7344073084732
```

In [48]:
```
from sklearn.model_selection import GridSearchCV

param_grid = [
    # try 12 (3×4) combinations of hyperparameters
    {'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]},
  ]

forest_reg = RandomForestRegressor(random_state=42)
# train across 5 folds, that's 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)

grid_search.best_params_

grid_search.best_estimator_

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

```
63895.161577951665 {'max_features': 2, 'n_estimators': 3}
54916.32386349543 {'max_features': 2, 'n_estimators': 10}
52885.86715332332 {'max_features': 2, 'n_estimators': 30}
60075.3680329983 {'max_features': 4, 'n_estimators': 3}
52495.01284985185 {'max_features': 4, 'n_estimators': 10}
50187.24324926565 {'max_features': 4, 'n_estimators': 30}
58064.73529982314 {'max_features': 6, 'n_estimators': 3}
51519.32062366315 {'max_features': 6, 'n_estimators': 10}
49969.80441627874 {'max_features': 6, 'n_estimators': 30}
58895.824998155826 {'max_features': 8, 'n_estimators': 3}
52459.79624724529 {'max_features': 8, 'n_estimators': 10}
49898.98913455217 {'max_features': 8, 'n_estimators': 30}
62381.765106921855 {'bootstrap': False, 'max_features': 2, 'n_estimators': 3}
54476.57050944266 {'bootstrap': False, 'max_features': 2, 'n_estimators': 10}
59974.60028085155 {'bootstrap': False, 'max_features': 3, 'n_estimators': 3}
52754.5632813202 {'bootstrap': False, 'max_features': 3, 'n_estimators': 10}
57831.136061214274 {'bootstrap': False, 'max_features': 4, 'n_estimators': 3}
51278.37877140253 {'bootstrap': False, 'max_features': 4, 'n_estimators': 10}
```

In [49]:
```
#随机搜索。
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', r
rnd_search.fit(housing_prepared, housing_labels)

rnd_search.best_params_

#所有结果:
cvres = rnd_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(np.sqrt(-mean_score), params)
```

```
49117.55344336652 {'max_features': 7, 'n_estimators': 180}
51450.63202856348 {'max_features': 5, 'n_estimators': 15}
50692.53588182537 {'max_features': 3, 'n_estimators': 72}
50783.614493515 {'max_features': 5, 'n_estimators': 21}
49162.89877456354 {'max_features': 7, 'n_estimators': 122}
50655.798471042704 {'max_features': 3, 'n_estimators': 75}
50513.856319990606 {'max_features': 3, 'n_estimators': 88}
49521.17201976928 {'max_features': 5, 'n_estimators': 100}
50302.90440763418 {'max_features': 3, 'n_estimators': 150}
65167.02018649492 {'max_features': 5, 'n_estimators': 2}
```

In [50]:
```
feature_importances = grid_search.best_estimator_.feature_importances_
feature_importances
```

Out[50]:
```
array([6.96542523e-02, 6.04213840e-02, 4.21882202e-02, 1.52450557e-02,
       1.55545295e-02, 1.58491147e-02, 1.49346552e-02, 3.79009225e-01,
       5.47789150e-02, 1.07031322e-01, 4.82031213e-02, 6.79266007e-03,
       1.65706303e-01, 7.83480660e-05, 1.52473276e-03, 3.02816106e-03])
```

In [51]:
```
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[51]:
```
[(0.3790092248170967, 'median_income'),
 (0.16570630316895876, 'INLAND'),
 (0.10703132208204355, 'pop_per_hhold'),
 (0.06965425227942929, 'longitude'),
 (0.0604213840080722, 'latitude'),
 (0.054778915018283726, 'rooms_per_hhold'),
 (0.048203121338269206, 'bedrooms_per_room'),
 (0.04218822024391753, 'housing_median_age'),
 (0.015849114744428634, 'population'),
 (0.015554529490469328, 'total_bedrooms'),
 (0.01524505568840977, 'total_rooms'),
 (0.014934655161887772, 'households'),
 (0.006792660074259966, '<1H OCEAN'),
 (0.0030281610628962747, 'NEAR OCEAN'),
 (0.0015247327555504937, 'NEAR BAY'),
 (7.834806602687504e-05, 'ISLAND')]
```

In [52]:
```
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)

final_rmse
```

Out[52]:
```
47873.26095812988
```

In [53]:
```
from scipy import stats
confidence = 0.95
squared_errors = (final_predictions - y_test) ** 2
np.sqrt(stats.t.interval(confidence, len(squared_errors) - 1,
                         loc=squared_errors.mean(),
                         scale=stats.sem(squared_errors)))
```

Out[53]:
```
array([45893.36082829, 49774.46796717])
```

In [54]:
```
full_pipeline_with_predictor = Pipeline([
        ("preparation", full_pipeline),
        ("linear", LinearRegression())
    ])

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

Out[54]:
```
array([ 85657.90192014, 305492.60737488, 152056.46122456, 186095.70946094,
       244550.67966089])
```

In [55]:
```
#保存训练好的模型
my_model = full_pipeline_with_predictor
import joblib
joblib.dump(my_model, "my_model.pkl")
my_model_loaded = joblib.load("my_model.pkl")
```

In [56]:
```
#第一题
housing_labels_log = np.log(housing_labels)
```

In [57]:
```
#交叉验证 过拟合
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)

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

display_scores(tree_rmse_scores)

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)
#线性回归
display_scores(lin_rmse_scores)
```

```
Scores: [71816.82864366 70413.94880788 67862.06771329 71875.1625928
 71157.48859112 78619.87969591 70154.34479635 73760.24692096
 68603.77073613 71051.08481462]
Mean: 71531.48233127293
Standard deviation: 2845.4252461810097
Scores: [71762.76364394 64114.99166359 67771.17124356 68635.19072082
 66846.14089488 72528.03725385 73997.08050233 68802.33629334
 66443.28836884 70139.79923956]
Mean: 69104.07998247063
Standard deviation: 2880.328209818069
```

In [58]:
```
#随机森林
from sklearn.ensemble import RandomForestRegressor
forest_reg = RandomForestRegressor()
forest_reg.fit(housing_prepared,housing_labels)

housing_predictions = forest_reg.predict(housing_prepared)
forest_mse = mean_squared_error(housing_labels, housing_predictions)
forest_rmse = np.sqrt(forest_mse)
forest_rmse

from sklearn.model_selection import cross_val_score

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)
#结果有改善!
```

```
Scores: [51300.74038384 48878.35063372 46751.16089361 52074.36696323
 47106.51743953 51476.63695219 52583.13232104 50071.81516679
 48449.2404587  53956.89479337]
Mean: 50264.88560060251
Standard deviation: 2289.074757484492
```

In [59]:
```
#第二题
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVR

param_grid = [
        {'kernel': ['linear'], 'C': [10., 30., 100., 300., 1000., 3000., 10000., 300
        {'kernel': ['rbf'], 'C': [1.0, 3.0, 10., 30., 100., 300., 1000.0],
         'gamma': [0.01, 0.03, 0.1, 0.3, 1.0, 3.0]},
    ]

svm_reg = SVR()
grid_search = GridSearchCV(svm_reg, param_grid, cv=5, scoring='neg_mean_squared_erro
grid_search.fit(housing_prepared, housing_labels)
```

```
9s
[CV] END .......................C=1.0, gamma=0.3, kernel=rbf; total time=   8.
9s
[CV] END .......................C=1.0, gamma=1.0, kernel=rbf; total time=   8.
6s
[CV] END .......................C=1.0, gamma=1.0, kernel=rbf; total time=   9.
0s
[CV] END .......................C=1.0, gamma=1.0, kernel=rbf; total time=   8.
7s
[CV] END .......................C=1.0, gamma=1.0, kernel=rbf; total time=  11.
9s
[CV] END .......................C=1.0, gamma=1.0, kernel=rbf; total time=  11.
6s
[CV] END .......................C=1.0, gamma=3.0, kernel=rbf; total time=  11.
4s
[CV] END .......................C=1.0, gamma=3.0, kernel=rbf; total time=   8.
5s
[CV] END .......................C=1.0, gamma=3.0, kernel=rbf; total time=   8.
6s
[CV] END .......................C=1.0, gamma=3.0, kernel=rbf; total time=   8.
```

In [60]:
```
negative_mse = grid_search.best_score_
rmse = np.sqrt(-negative_mse)
rmse
grid_search.best_params_
```

Out[60]:
```
{'C': 30000.0, 'kernel': 'linear'}
```

In [ ]:
```
```