In [3]:
```python
import sklearn
import numpy as np
import pandas as pd
```

In [4]:
```python
np.random.seed(42)
```

In [5]:
```python
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
```

In [6]:
```python
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)
```

In [8]:
```python
import warnings
warnings.filterwarnings(action="ignore", message="^internal gelsd")
```

In [15]:
```python
HOUSING_PATH = "/cxldata/datasets/project/housing/housing.csv"
```

In [28]:
```python
housing = pd.read_csv(HOUSING_PATH)
```

In [29]:
```python
housing.head()
```

Out[29]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	me
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	

edian_income	median_house_value	ocean_proximity
8.3252	452600.0	NEAR BAY
8.3014	358500.0	NEAR BAY
7.2574	352100.0	NEAR BAY

3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0

In [33]:
```
housing.info
```

Out[33]:
```
<bound method DataFrame.info of        longitude  latitude  housing_median_age  tc
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
...          ...       ...                 ...          ...             ...
20635    -121.09     39.48                25.0       1665.0           374.0
20636    -121.21     39.49                18.0        697.0           150.0
20637    -121.22     39.43                17.0       2254.0           485.0
20638    -121.32     39.43                18.0       1860.0           409.0
20639    -121.24     39.37                16.0       2785.0           616.0

       population  households  median_income  median_house_value  \
0           322.0       126.0         8.3252            452600.0
1          2401.0      1138.0         8.3014            358500.0
2           496.0       177.0         7.2574            352100.0
3           558.0       219.0         5.6431            341300.0
4           565.0       259.0         3.8462            342200.0
...           ...         ...            ...                 ...
20635       845.0       330.0         1.5603             78100.0
20636       356.0       114.0         2.5568             77100.0
20637      1007.0       433.0         1.7000             92300.0
20638       741.0       349.0         1.8672             84700.0
20639      1387.0       530.0         2.3886             89400.0

      ocean_proximity
0            NEAR BAY
1            NEAR BAY
2            NEAR BAY
```

5.6431	341300.0	NEAR BAY
3.8462	342200.0	NEAR BAY

```
otal_rooms  total_bedrooms  \
```

```
2            NEAR BAY
3            NEAR BAY
4            NEAR BAY
...               ...
20635          INLAND
20636          INLAND
20637          INLAND
20638          INLAND
20639          INLAND

[20640 rows x 10 columns]>
```

In [34]:

```
housing.describe
```

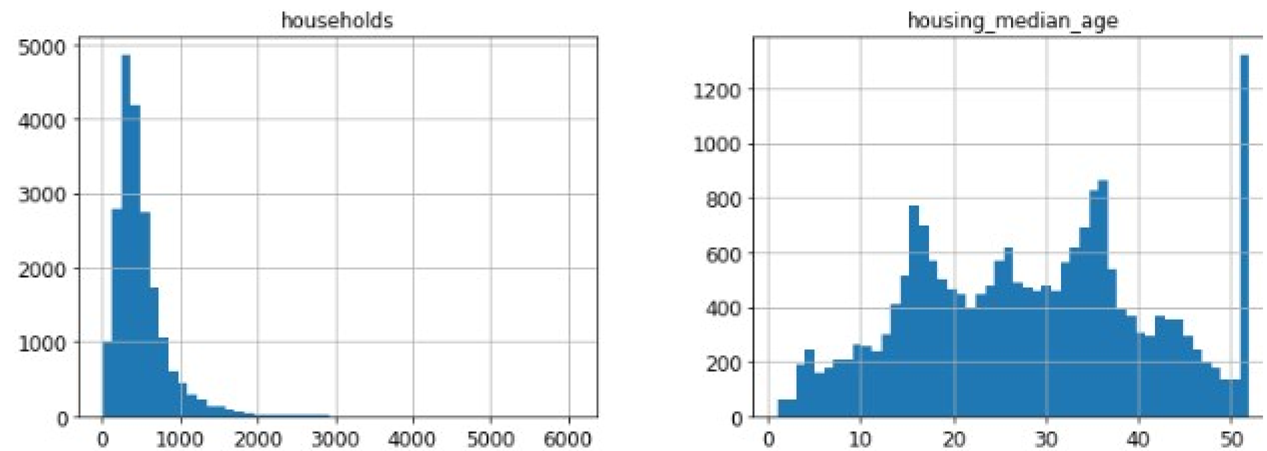
Out[34]:

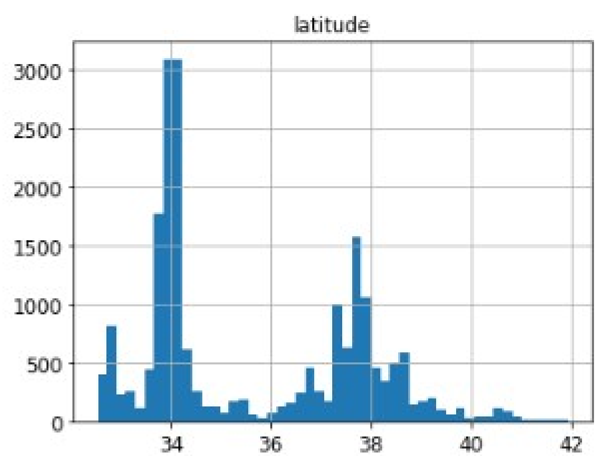
```
<bound method NDFrame.describe of        longitude  latitude  housing_median_age
0        -122.23     37.88                41.0        880.0             129.0
1        -122.22     37.86                21.0       7099.0            1106.0
2        -122.24     37.85                52.0       1467.0             190.0
3        -122.25     37.85                52.0       1274.0             235.0
4        -122.25     37.85                52.0       1627.0             280.0
...          ...       ...                 ...          ...               ...
20635    -121.09     39.48                25.0       1665.0             374.0
20636    -121.21     39.49                18.0        697.0             150.0
20637    -121.22     39.43                17.0       2254.0             485.0
20638    -121.32     39.43                18.0       1860.0             409.0
20639    -121.24     39.37                16.0       2785.0             616.0

       population  households  median_income  median_house_value  \
0           322.0       126.0         8.3252            452600.0
1          2401.0      1138.0         8.3014            358500.0
2           496.0       177.0         7.2574            352100.0
3           558.0       219.0         5.6431            341300.0
4           565.0       259.0         3.8462            342200.0
...           ...         ...            ...                 ...
20635       845.0       330.0         1.5603             78100.0
20636       356.0       114.0         2.5568             77100.0
```

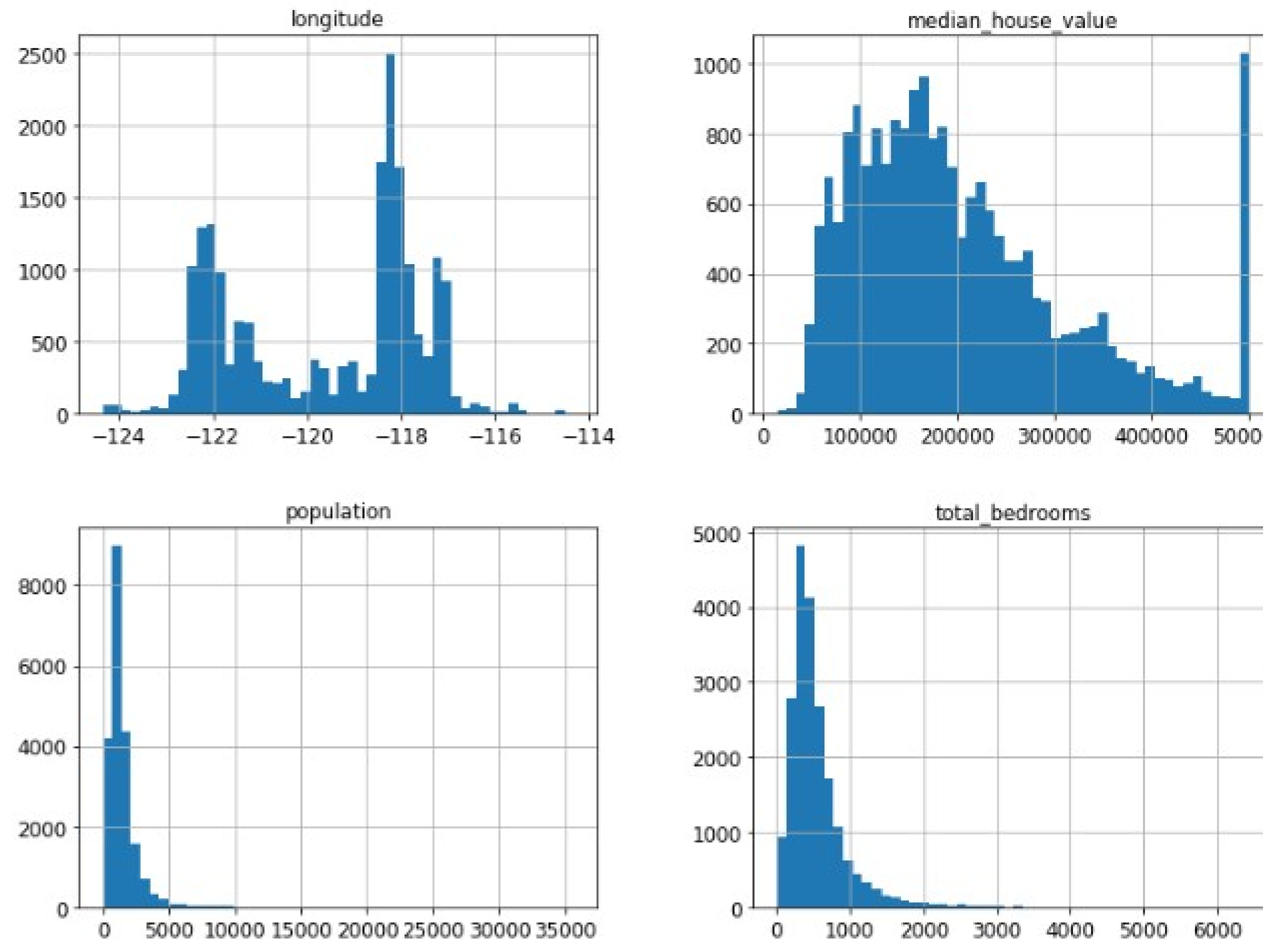
```
total_rooms  total_bedrooms  \
```

```
20637    1007.0    433.0    1.7000    92300.0
20638     741.0    349.0    1.8672    84700.0
20639    1387.0    530.0    2.3886    89400.0

      ocean_proximity
0            NEAR BAY
1            NEAR BAY
2            NEAR BAY
3            NEAR BAY
4            NEAR BAY
...               ...
20635          INLAND
20636          INLAND
20637          INLAND
20638          INLAND
20639          INLAND

[20640 rows x 10 columns]>
```

In [35]:
```
housing.hist(bins=50, figsize=(20,15))
plt.show()
```

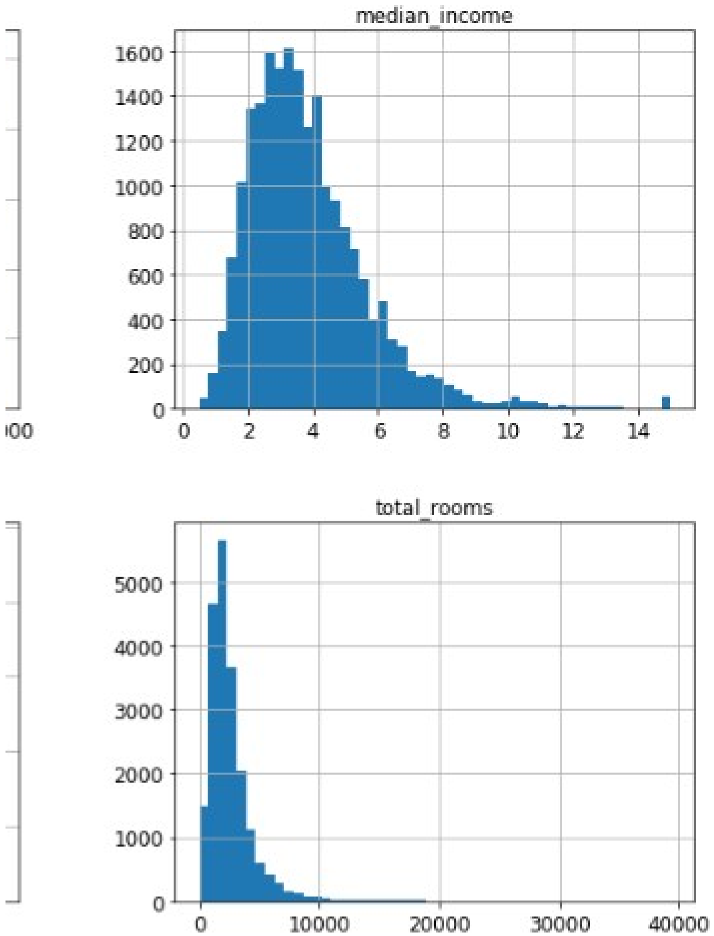



latitude
3000
2500
2000
1500
1000
500
0
34
36
38
40
42




```
In [36]: housing["median_income"].hist
```

```
Out[36]: <bound method hist_series of 0        8.3252
         1        8.3014
         2        7.2574
         3        5.6431
         4        3.8462
```

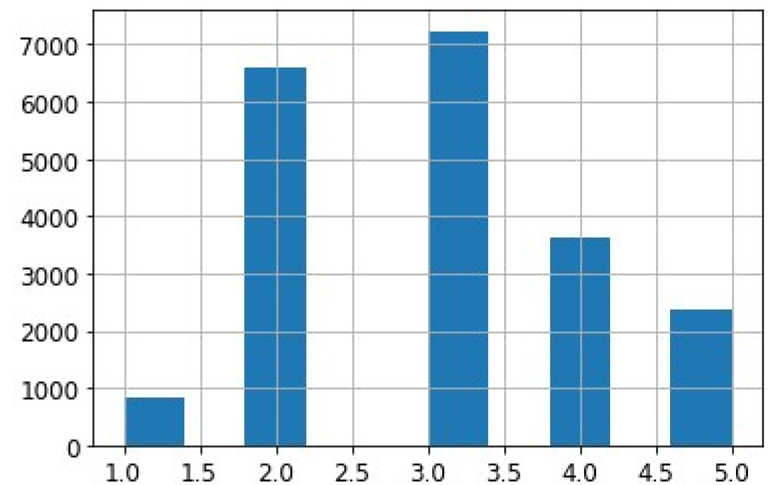


median_income
1600
1400
1200
1000
800
600
400
200
0
0
2
4
6
8
10
12
14
total_rooms
5000
4000
3000
2000
1000
0
0
10000
20000
30000
40000

```
            ...
20635    1.5603
20636    2.5568
20637    1.7000
20638    1.8672
20639    2.3886
Name: median_income, Length: 20640, dtype: float64>
```

In [37]:
```python
housing["income_cat"] = pd.cut(housing["median_income"],
                               bins=[0., 1.5, 3.0, 4.5, 6., np.inf],
                               labels=[1, 2, 3, 4, 5])

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

Out[37]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4577a3a4a8>
```



In [40]:
```python
from sklearn.model_selection import StratifiedShuffleSplit
```

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

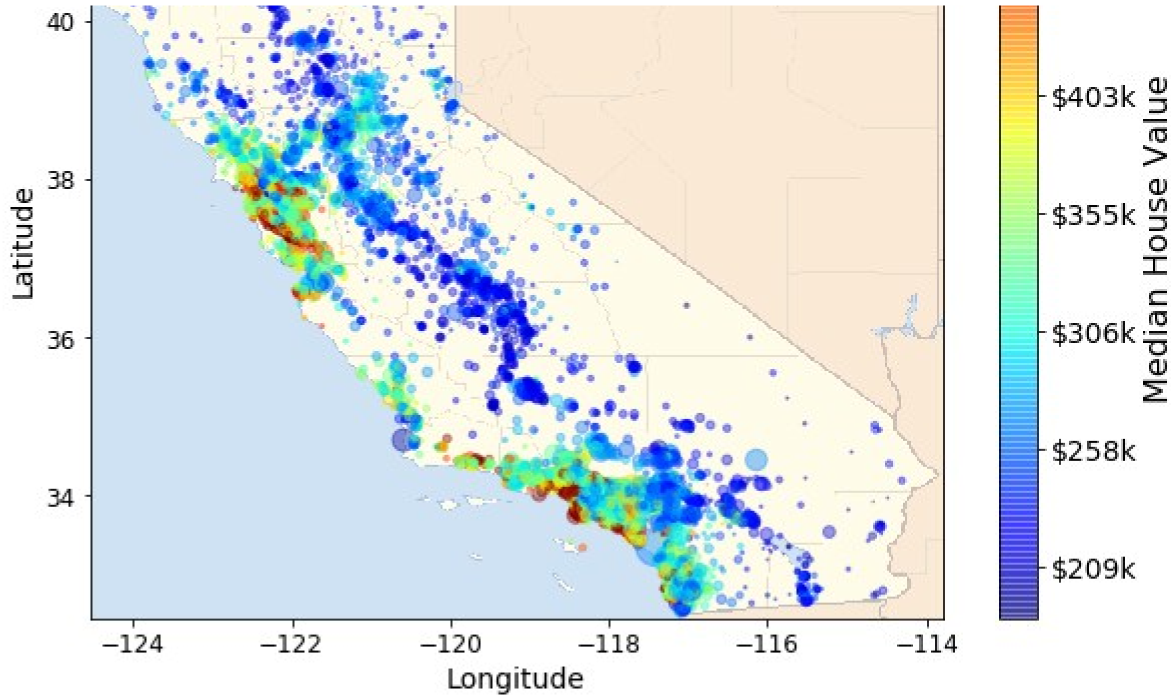
```
for train_index, test_index in split.split(housing, housing["income_cat"]):
    strat_train_set = housing.loc[train_index]
    strat_test_set = housing.loc[test_index]
```

In [44]:
```
for set_ in (strat_train_set, strat_test_set):
    set_.drop("income_cat", axis=1, inplace=True)
```

In [48]:
```
housing = strat_train_set.copy()
```

In [49]:
```
import matplotlib.image as mpimg
california_img=mpimg.imread('/cxldata/datasets/project/housing/california.png')
ax = housing.plot(kind="scatter", x="longitude", y="latitude", figsize=(10,7),
                       s=housing['population']/100, label="Population",
                       c="median_house_value", cmap=plt.get_cmap("jet"),
                       colorbar=False, alpha=0.4,
                      )
plt.imshow(california_img, extent=[-124.55, -113.80, 32.45, 42.05], alpha=0.5,
           cmap=plt.get_cmap("jet"))
plt.ylabel("Latitude", fontsize=14)
plt.xlabel("Longitude", fontsize=14)

prices = housing["median_house_value"]
tick_values = np.linspace(prices.min(), prices.max(), 11)
cbar = plt.colorbar(ticks=tick_values/prices.max())
cbar.ax.set_yticklabels(["$%dk"%(round(v/1000)) for v in tick_values], fontsize=14
cbar.set_label('Median House Value', fontsize=16)

plt.legend(fontsize=16)
plt.show()
```

```
4)
```

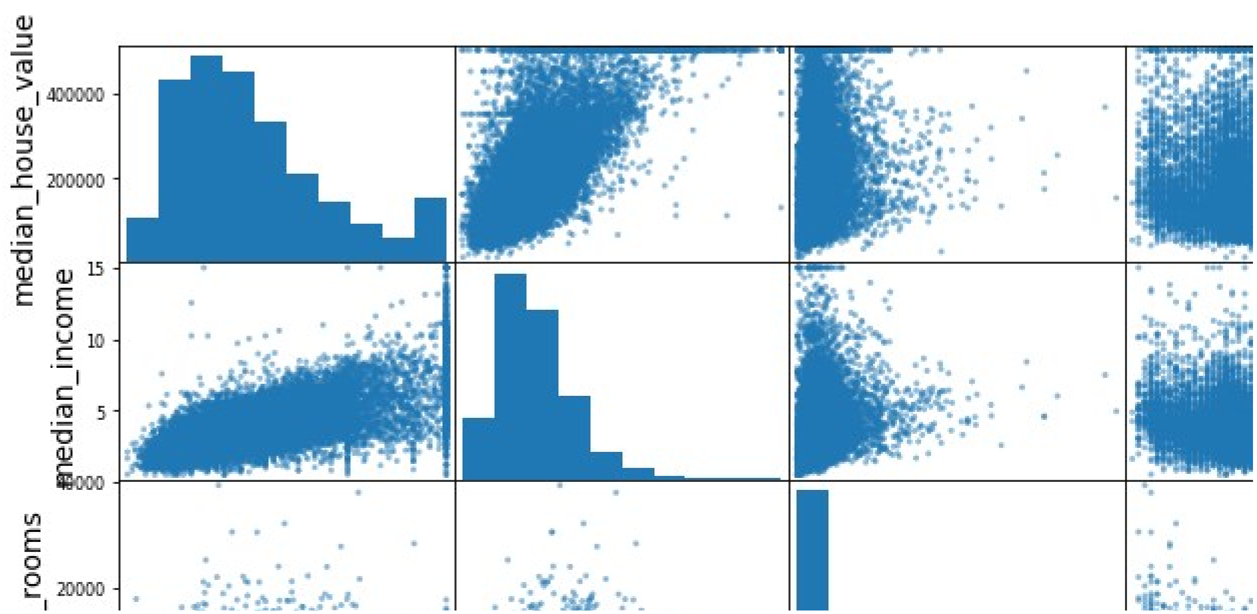


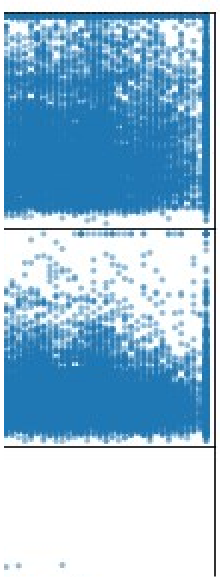
In [53]:
```
housing["rooms_per_household"] = housing["total_rooms"]/housing["households"]
housing["bedrooms_per_room"] = housing["total_bedrooms"]/housing["total_rooms"]
housing["population_per_household"]=housing["population"]/housing["households"]
```

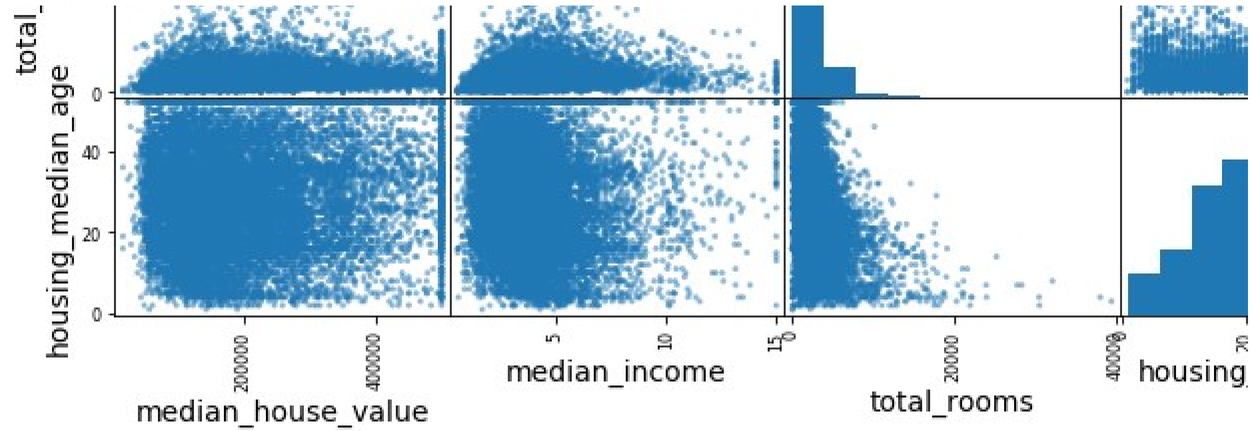
In [54]:
```
corr_matrix = housing.corr()
```

In [55]:
```
corr_matrix["median_house_value"].sort_values(ascending=False)

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[55]: array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f45776bc780>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x7f457081cd68>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x7f4577681358>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x7f4574ef5908>],
               [<matplotlib.axes._subplots.AxesSubplot object at 0x7f4574f22eb8>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x7f4574edc4a8>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x7f4574e8da58>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x7f45780314a8>],
               [<matplotlib.axes._subplots.AxesSubplot object at 0x7f45780315f8>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x7f45779d83c8>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x7f457809f588>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x7f4577a0f6d8>],
               [<matplotlib.axes._subplots.AxesSubplot object at 0x7f4577a28128>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x7f45778f1320>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x7f4578209cf8>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x7f457841c0f0>]],
              dtype=object)
```







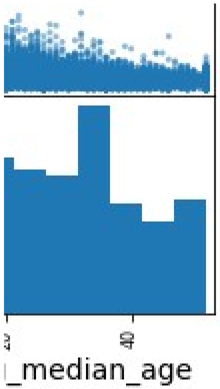
In [56]:
```
housing.describe()
```

Out[56]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population
count	16512.000000	16512.000000	16512.000000	16512.000000	16354.000000	16512.000000
mean	-119.575834	35.639577	28.653101	2622.728319	534.973890	1419.790819
std	2.001860	2.138058	12.574726	2138.458419	412.699041	1115.686241
min	-124.350000	32.540000	1.000000	6.000000	2.000000	3.000000
25%	-121.800000	33.940000	18.000000	1443.000000	295.000000	784.000000
50%	-118.510000	34.260000	29.000000	2119.500000	433.000000	1164.000000
75%	-118.010000	37.720000	37.000000	3141.000000	644.000000	1719.250000
max	-114.310000	41.950000	52.000000	39320.000000	6210.000000	35682.000000

In [61]:
```
housing = strat_train_set.drop("median_house_value", axis=1)
```

In [62]:
```
housing_labels = strat_train_set["median_house_value"].copy()
```



households	median_income	median_house_value	roor
16512.000000	16512.000000	16512.000000	
497.060380	3.875589	206990.920724	
375.720845	1.904950	115703.014830	
2.000000	0.499900	14999.000000	
279.000000	2.566775	119800.000000	
408.000000	3.540900	179500.000000	
602.000000	4.744475	263900.000000	
5358.000000	15.000100	500001.000000	

In [63]:
```python
from sklearn.impute import SimpleImputer
```

In [64]:
```python
imputer = SimpleImputer(strategy="median")
```

In [65]:
```python
housing_num = housing.drop("ocean_proximity", axis=1)
```

In [66]:
```python
imputer.fit(housing_num)
```

Out[66]:
```
SimpleImputer(add_indicator=False, copy=True, fill_value=None,
              missing_values=nan, strategy='median', verbose=0)
```

In [67]:
```python
X = imputer.transform(housing_num)
housing_tr = pd.DataFrame(X, columns=housing_num.columns,
                          index=housing.index)
```

In [70]:
```python
housing_cat = housing[["ocean_proximity"]]
```

In [71]:
```python
housing_cat.head(10)
```

Out[71]:

	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 [72]:
```
from sklearn.preprocessing import OneHotEncoder
```

In [73]:
```
cat_encoder = OneHotEncoder()
housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
housing_cat_1hot
```

Out[73]:
```
<16512x5 sparse matrix of type '<class 'numpy.float64'>'
	with 16512 stored elements in Compressed Sparse Row format>
```

In [74]:
```
housing_cat_1hot.toarray()
```

Out[74]:
```
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 [78]:
```
from sklearn.base import BaseEstimator, TransformerMixin
```

In [79]:
```
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]
```

```python
        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]

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

In [83]:

```python
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]

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

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)

from sklearn.compose import ColumnTransformer

num_attribs = list(housing_num)
```

```
ld"],
```

```python
num_attribs = list(housing_num)
cat_attribs = ["ocean_proximity"]

full_pipeline = ColumnTransformer([
        ("num", num_pipeline, num_attribs),
        ("cat", OneHotEncoder(), cat_attribs),
    ])
```

In [84]:
```python
housing_prepared = full_pipeline.fit_transform(housing)
```

In [87]:
```python
from sklearn.tree import DecisionTreeRegressor
```

In [88]:
```python
tree_reg = DecisionTreeRegressor(random_state=42)
tree_reg.fit(housing_prepared, housing_labels)
```

Out[88]:
```
DecisionTreeRegressor(ccp_alpha=0.0, 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='deprecated',
                      random_state=42, splitter='best')
```

In [89]:
```python
from sklearn.metrics import mean_squared_error
```

In [90]:
```python
housing_predictions = tree_reg.predict(housing_prepared)
```

In [91]:
```python
tree_mse = mean_squared_error(housing_labels, housing_predictions)
tree_rmse = np.sqrt(tree_mse)
tree_rmse
```

Out[91]:
```
0.0
```

In [95]:
```python
from sklearn.ensemble import RandomForestRegressor
```

In [97]:
```python
forest_reg = RandomForestRegressor(n_estimators=30, random_state=42)
forest_reg.fit(housing_prepared, housing_labels)
```

Out[97]:
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=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 [98]:
```python
housing_predictions = forest_reg.predict(housing_prepared)
```

In [99]:
```python
forest_mse = mean_squared_error(housing_labels, housing_predictions)
forest_rmse = np.sqrt(forest_mse)
forest_rmse
```

Out[99]:
```
19561.601906818396
```

In [102]:
```python
def display_scores(scores):
    print("Scores:", scores)
    print("Mean:", scores.mean())
    print("Standard deviation:", scores.std())
```

In [104]:
```python
from sklearn.model_selection import cross_val_score
```

In [105]:
```python
scores = cross_val_score(tree_reg, housing_prepared, housing_labels,
                         scoring="neg_mean_squared_error", cv=10)
tree_rmse_scores = np.sqrt(-scores)
display_scores(tree_rmse_scores)
```

```
Scores: [70194.33680785 66855.16363941 72432.58244769 70758.73896782
 71115.88230639 75585.14172901 70262.86139133 70273.6325285
```

```
 75366.87952553 71231.65726027]
Mean: 71407.68766037929
Standard deviation: 2439.4345041191004
```

In [107]:
```python
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: [50141.36385885 47640.30832627 50921.08943207 52659.54280148
 49506.38494424 54204.85834912 49214.04467119 47790.14481191
 53869.80154598 51020.52377426]
Mean: 50696.80625153554
Standard deviation: 2193.098737823643
```

In [111]:
```python
from sklearn.model_selection import GridSearchCV
```

In [112]:
```python
param_grid = [
    {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},
    {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]},
  ]
```

In [113]:
```python
forest_reg = RandomForestRegressor(random_state=42)
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[113]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=RandomForestRegressor(bootstrap=True, ccp_alpha=0.0,
                                             criterion='mse', max_depth=None,
                                             max_features='auto',
                                             max_leaf_nodes=None,
                                             max_samples=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=100, n_jobs=None,
                                          oob_score=False, random_state=42,
                                          verbose=0, warm_start=False),
             iid='deprecated', 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 [114]:
```
grid_search.best_params_
```

Out[114]:
```
{'max_features': 8, 'n_estimators': 30}
```

In [115]:
```
grid_search.best_estimator_
```

Out[115]:
```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features=8, max_leaf_nodes=None,
                      max_samples=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 [116]:
```
cvres = grid_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(np.sqrt(-mean_score), params)
```

```
63669.11631261028 {'max_features': 2, 'n_estimators': 3}
55627.099719926795 {'max_features': 2, 'n_estimators': 10}
```

```
53384.57275149205 {'max_features': 2, 'n_estimators': 30}
60965.950449450494 {'max_features': 4, 'n_estimators': 3}
52741.04704299915 {'max_features': 4, 'n_estimators': 10}
50377.40461678399 {'max_features': 4, 'n_estimators': 30}
58663.93866579625 {'max_features': 6, 'n_estimators': 3}
52006.19873526564 {'max_features': 6, 'n_estimators': 10}
50146.51167415009 {'max_features': 6, 'n_estimators': 30}
57869.25276169646 {'max_features': 8, 'n_estimators': 3}
51711.127883959234 {'max_features': 8, 'n_estimators': 10}
49682.273345071546 {'max_features': 8, 'n_estimators': 30}
62895.06951262424 {'bootstrap': False, 'max_features': 2, 'n_estimators': 3}
54658.176157539405 {'bootstrap': False, 'max_features': 2, 'n_estimators': 10}
59470.40652318466 {'bootstrap': False, 'max_features': 3, 'n_estimators': 3}
52724.9822587892 {'bootstrap': False, 'max_features': 3, 'n_estimators': 10}
57490.5691951261 {'bootstrap': False, 'max_features': 4, 'n_estimators': 3}
51009.495668875716 {'bootstrap': False, 'max_features': 4, 'n_estimators': 10}
```

In [124]:
```python
final_model = grid_search.best_estimator_
```

In [125]:
```python
X_test = strat_test_set.drop("median_house_value", axis=1)
y_test = strat_test_set["median_house_value"].copy()
```

In [126]:
```python
X_test_prepared = full_pipeline.transform(X_test)
```

In [127]:
```python
final_predictions = final_model.predict(X_test_prepared)
```

In [128]:
```python
final_mse = mean_squared_error(y_test, final_predictions)
final_rmse = np.sqrt(final_mse)
```

In [129]:
```python
final_rmse
```

Out[129]:
```
47730.22690385927
```

```
In [ ]:
```