In [18]:
```
import sklearn
import numpy as np
import pandas as pd
```

In [19]:
```
np.random.seed(42)
```

In [20]:
```
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
```

In [21]:
```
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)
```

In [22]:
```
import warnings
warnings.filterwarnings(action="ignore", message="^internal gelsd")
```

In [23]:
```
!tail /cxldata/datasets/project/housing/housing.csv
```

```
-121.32,39.29,11.0,2640.0,505.0,1257.0,445.0,3.5673,112000.0,INLAND
-121.4,39.33,15.0,2655.0,493.0,1200.0,432.0,3.5179,107200.0,INLAND
-121.45,39.26,15.0,2319.0,416.0,1047.0,385.0,3.125,115600.0,INLAND
-121.53,39.19,27.0,2080.0,412.0,1082.0,382.0,2.5495,98300.0,INLAND
-121.56,39.27,28.0,2332.0,395.0,1041.0,344.0,3.7125,116800.0,INLAND
-121.09,39.48,25.0,1665.0,374.0,845.0,330.0,1.5603,78100.0,INLAND
-121.21,39.49,18.0,697.0,150.0,356.0,114.0,2.5568,77100.0,INLAND
-121.22,39.43,17.0,2254.0,485.0,1007.0,433.0,1.7,92300.0,INLAND
-121.32,39.43,18.0,1860.0,409.0,741.0,349.0,1.8672,84700.0,INLAND
-121.24,39.37,16.0,2785.0,616.0,1387.0,530.0,2.3886,89400.0,INLAND
```

In [24]:
```
HOUSING_PATH = "/cxldata/datasets/project/housing/housing.csv"
```

In [25]:
```
HOUSING_PATH
```

Out[25]: '/cxldata/datasets/project/housing/housing.csv'

In [26]:
```
housing = pd.read_csv(HOUSING_PATH)
housing.head()
```

Out[26]:

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

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

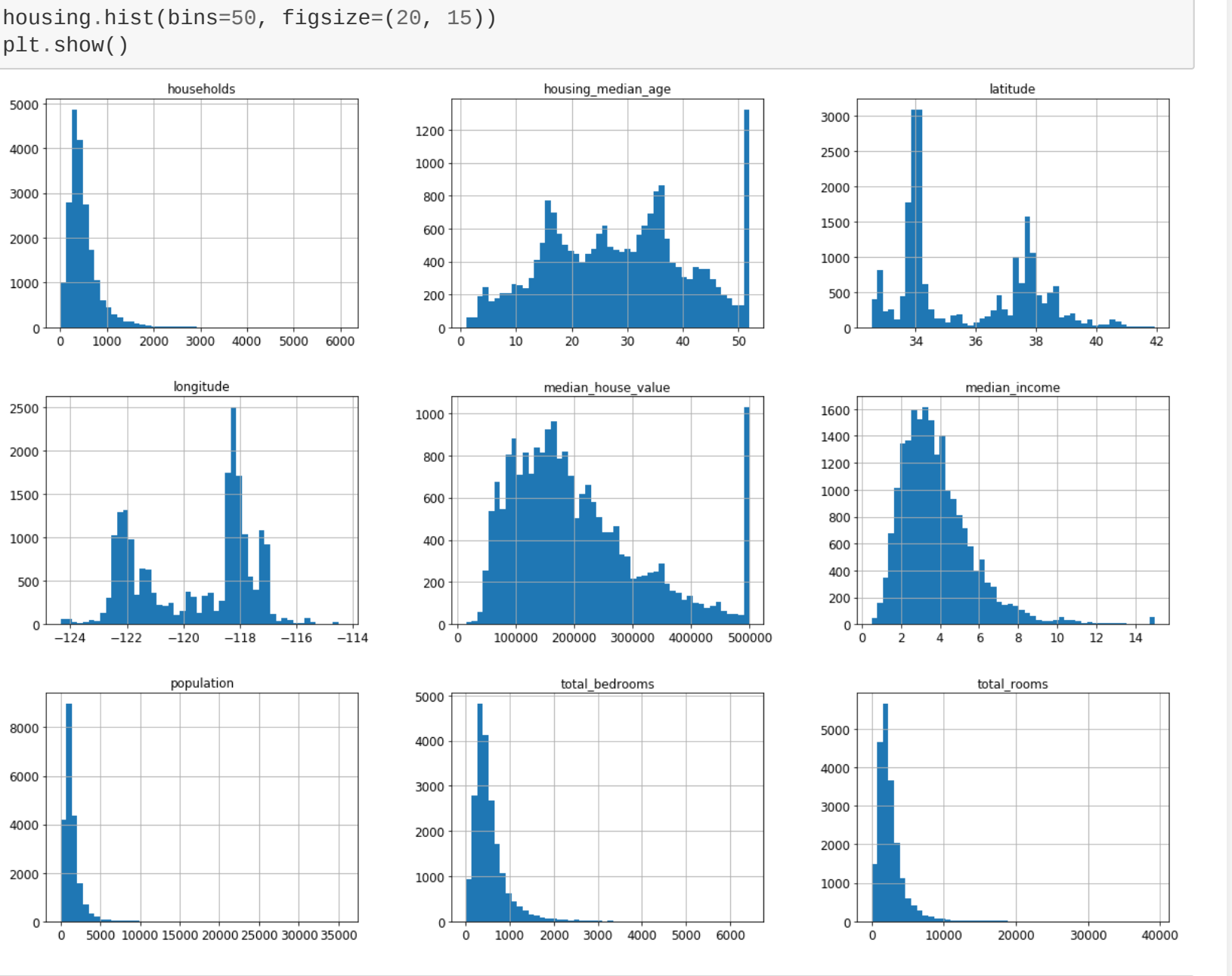
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 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 [28]:
```
housing.describe()
```

Out[28]:

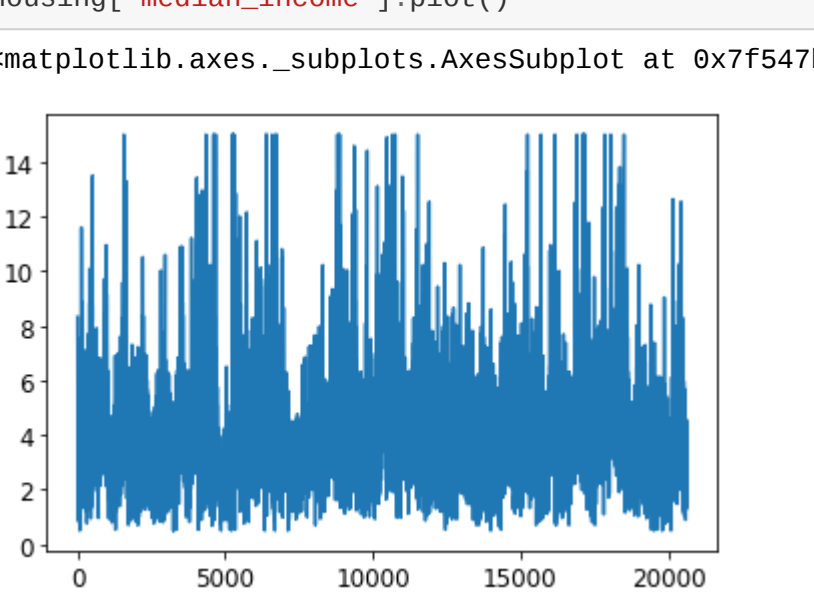
	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_i
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.(
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.8
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.8
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.4
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.5
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.5
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.7
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.(

In [29]:
```
housing.hist(bins=50, figsize=(20, 15))
plt.show()
```



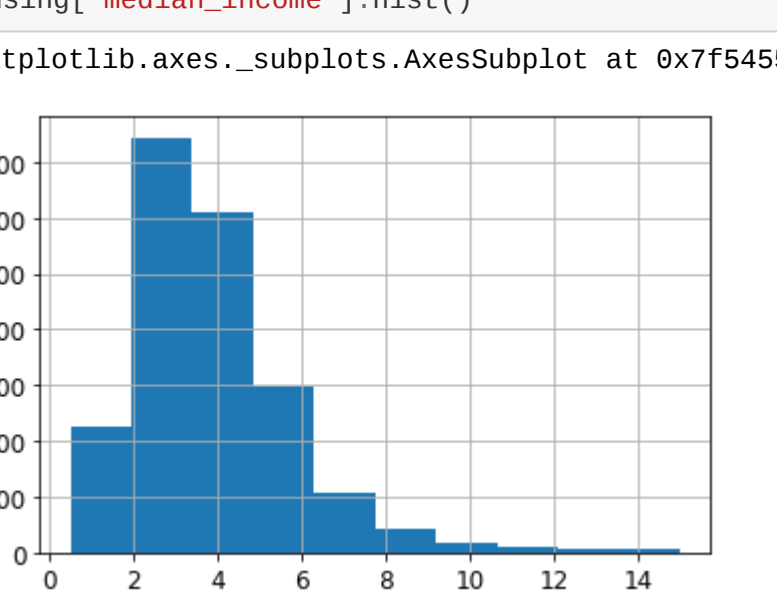
In [30]:
```
housing["median_income"].plot()
```

Out[30]: <matplotlib.axes._subplots.AxesSubplot at 0x7f547bd6f3c8>



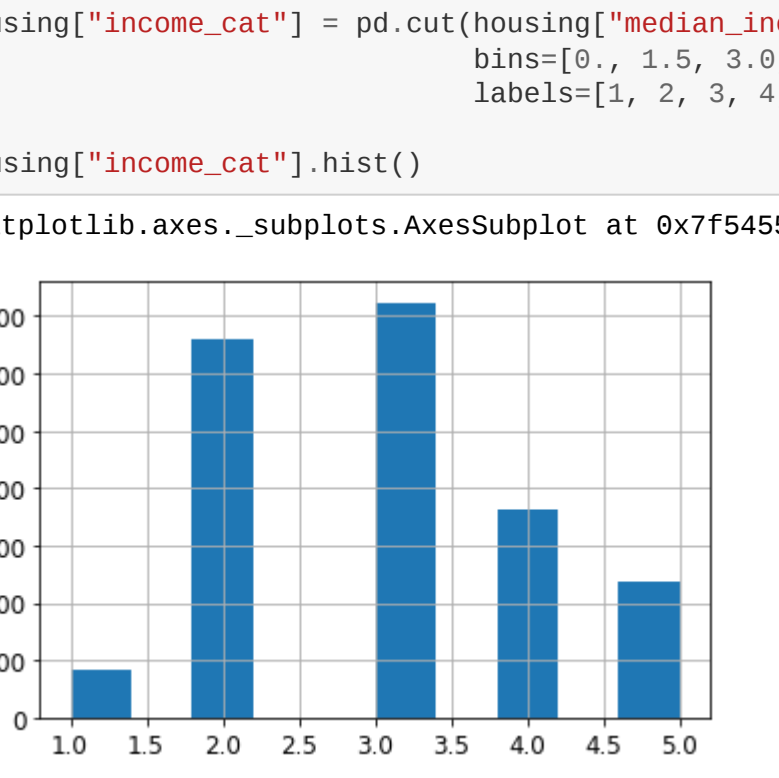
In [31]:
```
housing["median_income"].hist()
```

Out[31]: <matplotlib.axes._subplots.AxesSubplot at 0x7f5455a14668>



In [32]:
```
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()
```

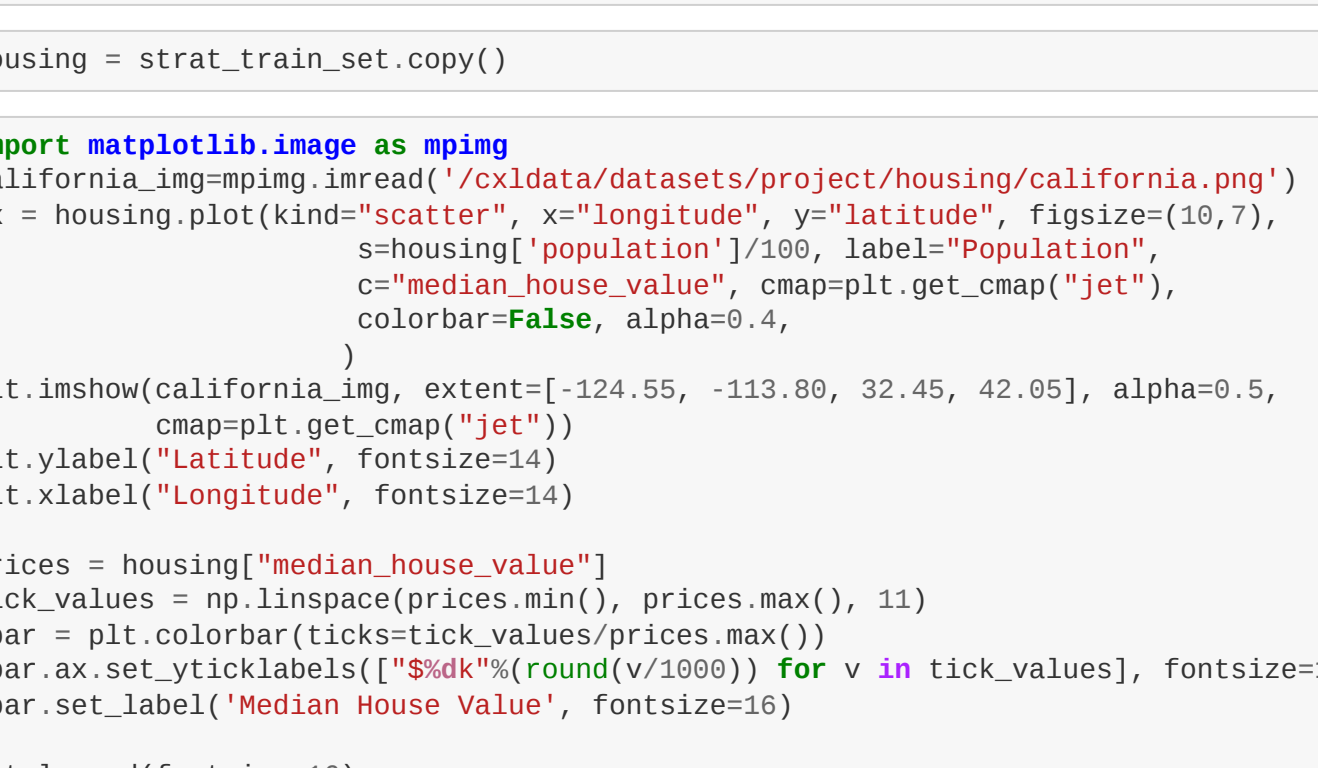
Out[32]: <matplotlib.axes._subplots.AxesSubplot at 0x7f5455994cc0>



In [33]:
```
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]
for set_ in (strat_train_set, strat_test_set):
    set_.drop("income_cat", axis=1, inplace=True)
```

In [34]:
```
housing = strat_train_set.copy()
```

In [35]:
```
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()
```

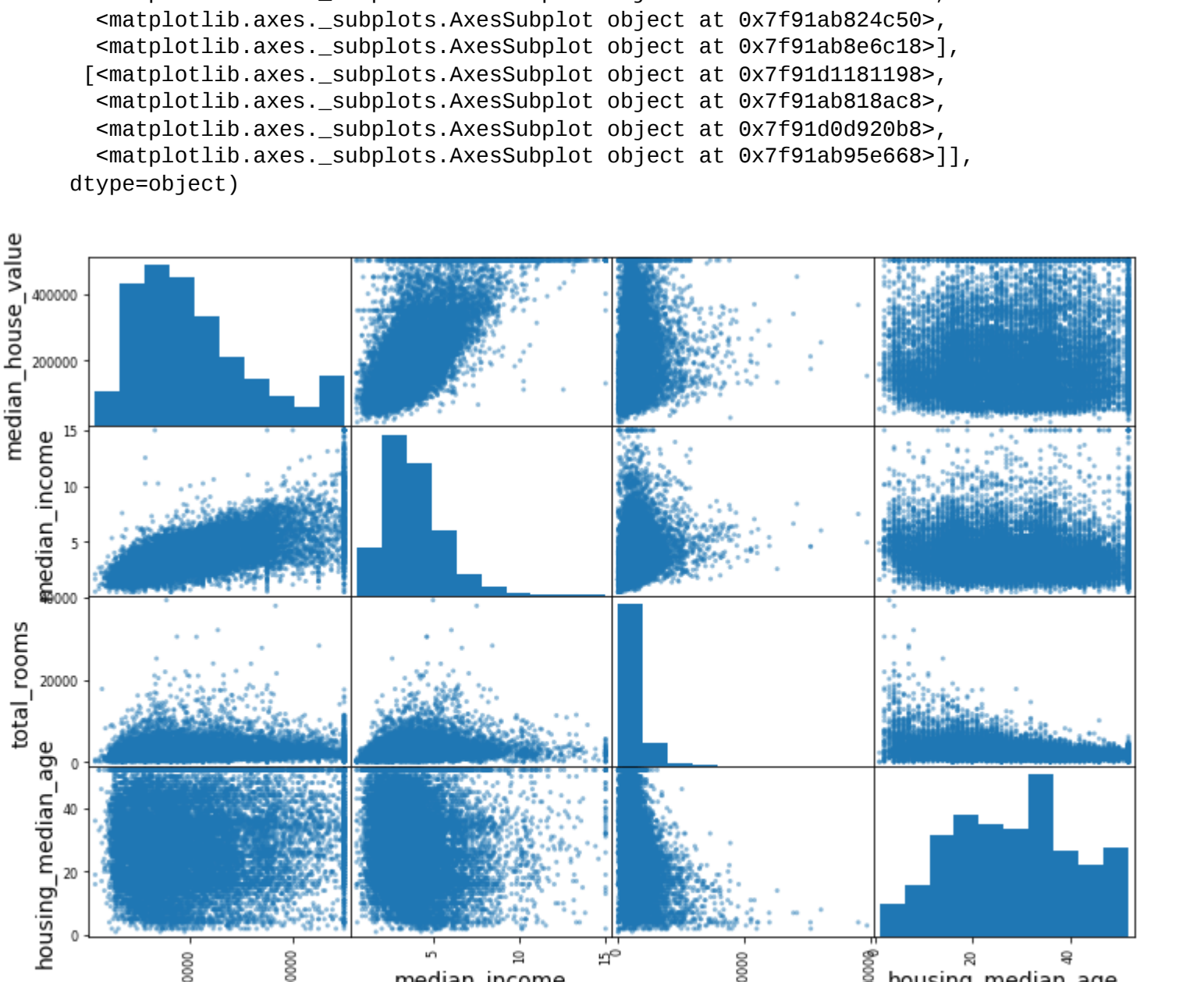


In [36]:
```
# creating 3 new features from the existing features
housing["rooms_per_household"] = housing["total_rooms"]/housing["households"]
housing["bedrooms_per_room"] = housing["total_bedrooms"]/housing["total_rooms"]
housing["population_per_household"]=housing["population"]/housing["households"]
```

In [37]:
```
corr_matrix = housing.corr()
```

In [46]:
```
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[46]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f91ab1565c0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f91aad80f98>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f91ab84f978>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f91aad6bd68>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f91aac9e860>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f91d0a3e748>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f91d0a033c8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f91d0af10f0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f91d0af1080>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f91d0c77d68>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f91ab824c50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f91ab8e6c18>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f91d1181198>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f91ab818ac8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f91d0d920b8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f91ab95e668>]],
      dtype=object)
```



In [38]:
```
housing.describe()
```

Out[38]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_i
count	16512.000000	16512.000000	16512.000000	16512.000000	16354.000000	16512.000000	16512.000000	16512.(
mean	-119.575834	35.639577	28.653101	2622.728319	534.973890	1419.790819	497.060380	3.8
std	2.001860	2.138058	12.574726	2138.458419	412.699041	1115.686241	375.720845	1.9
min	-124.350000	32.540000	1.000000	6.000000	2.000000	3.000000	2.000000	0.4
25%	-121.800000	33.940000	18.000000	1443.000000	295.000000	784.000000	279.000000	2.5
50%	-118.510000	34.260000	29.000000	2119.500000	433.000000	1164.000000	408.000000	3.5
75%	-118.010000	37.720000	37.000000	3141.000000	644.000000	1719.250000	602.000000	4.7
max	-114.310000	41.950000	52.000000	39320.000000	6210.000000	35682.000000	5358.000000	15.(

In [39]:
```
housing = strat_train_set.drop("median_house_value", axis=1)
housing_labels = strat_train_set["median_house_value"].copy()
```

In [40]:
```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy ="median")
```

In [41]:
```
housing_num = housing.drop("ocean_proximity", axis=1)
```

In [42]:
```
imputer.fit(housing_num)
```

Out[42]:
```
SimpleImputer(add_indicator=False, copy=True, fill_value=None,
              missing_values=nan, strategy='median', verbose=0)
```

In [43]:
```
X = imputer.transform(housing_num)
housing_tr = pd.DataFrame(X, columns=housing_num.columns,
                          index=housing.index)
```

In [44]:
```
housing_cat = housing[["ocean_proximity"]]
housing_cat.head(10)
```

Out[44]:

	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 [45]:
```
from sklearn.preprocessing import OneHotEncoder
cat_encoder = OneHotEncoder()
housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
housing_cat_1hot
```

Out[45]:
```
<16512x5 sparse matrix of type '<class 'numpy.float64'>'
	with 16512 stored elements in Compressed Sparse Row format>
```

In [46]:
```
housing_cat_1hot.toarray()
```

Out[46]:
```
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 [47]:
```
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_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 [48]:
```
housing.head()
```

Out[48]:

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

In [49]:
```
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_household"],
    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)
cat_attribs = ["ocean_proximity"]

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

In [50]:
```
housing_prepared = full_pipeline.fit_transform(housing)
```

In [51]:
```
from sklearn.tree import DecisionTreeRegressor
```

In [52]:
```
tree_reg = DecisionTreeRegressor(random_state=42)
tree_reg.fit(housing_prepared, housing_labels)
```

Out[52]:
```
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 [53]:
```
from sklearn.metrics import mean_squared_error
```

In [54]:
```
housing_predictions = tree_reg.predict(housing_prepared)
```

In [55]:
```
tree_mse = mean_squared_error(housing_labels, housing_predictions)
tree_rmse = np.sqrt(tree_mse)
tree_rmse
```

Out[55]: 0.0

In [56]:
```
from sklearn.ensemble import RandomForestRegressor
```

In [57]:
```
forest_reg = RandomForestRegressor(n_estimators=30, random_state=42)
forest_reg.fit(housing_prepared, housing_labels)
```

Out[57]:
```
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 [58]:
```
housing_predictions = forest_reg.predict(housing_prepared)
```

In [59]:
```
forest_mse = mean_squared_error(housing_labels, housing_predictions)
forest_rmse = np.sqrt(forest_mse)
forest_rmse
```

Out[59]: 19561.601906818396

In [60]:
```
def display_scores(scores):
    print("Scores:", scores)
    print("Mean:", scores.mean())
    print("Standard deviation:", scores.std())
```

In [61]:
```
from sklearn.model_selection import cross_val_score
```

In [62]:
```
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 [63]:
```
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 [67]:
```
from sklearn.model_selection import GridSearchCV
```

In [68]:
```
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 [69]:
```
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[69]:
```
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 [70]:
```
grid_search.best_params_
```

Out[70]: {'max_features': 8, 'n_estimators': 30}

In [71]:
```
grid_search.best_estimator_
```

Out[71]:
```
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 [72]:
```
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 [81]:
```
final_model = grid_search.best_estimator_
```

In [82]:
```
X_test = strat_test_set.drop("median_house_value", axis=1)
y_test = strat_test_set["median_house_value"].copy()
```

In [83]:
```
X_test_prepared = full_pipeline.transform(X_test)
```

In [84]:
```
final_predictions = final_model.predict(X_test_prepared)
```

In [85]:
```
final_mse = mean_squared_error(y_test, final_predictions)
final_rmse = np.sqrt(final_mse)
```

In [86]:
```
print(final_rmse)
```

```
47730.22690385927
```

In [ ]: