California Housing Price Prediction

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

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

In [ ]:

In [4]:
```
housing = pd.read_csv("housing.csv")
housing.head()
```

Out[4]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0

In [5]:
```
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
```

there are 20640 instances & 'total_bedrooms' has only 20433 non-null values (207 values missing)

In [6]:
```
housing['ocean_proximity'].value_counts()
```

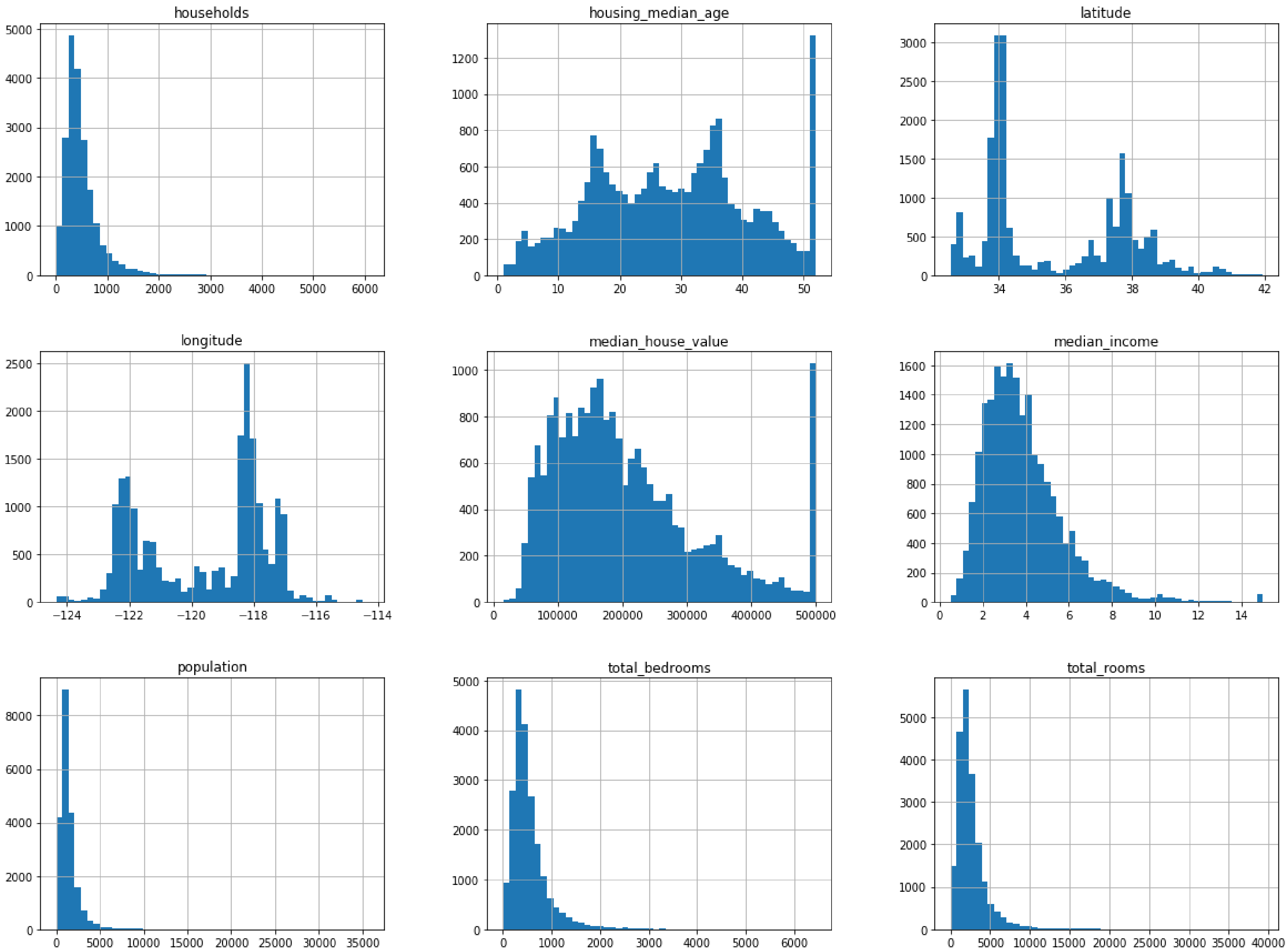
Out[6]:
```
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: ocean_proximity, dtype: int64
```

In [7]:
```
housing.describe()
```

Out[7]:

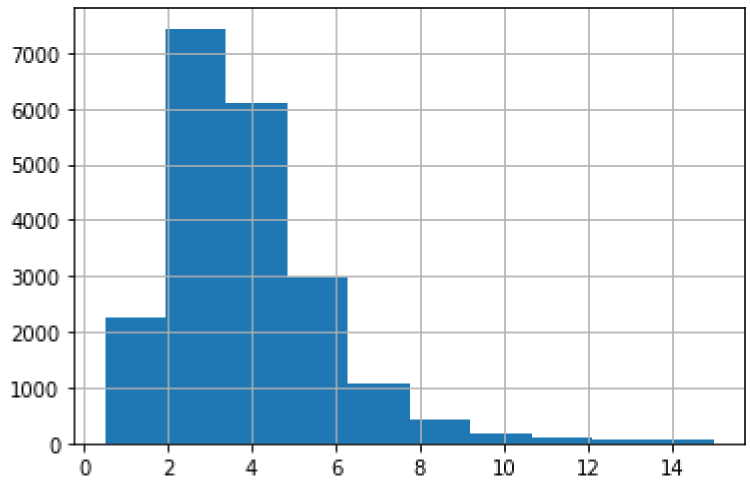
	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	mec
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.000000	
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.870671	
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.899822	
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.563400	
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.534800	
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.743250	
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.000100	

In [8]:
```
housing.hist(bins=50, figsize=(20, 15))
plt.show()
```



In [9]:
```
# median income looks like an imp feature

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

Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x14b595c96d8>



In [10]:
```
# dividing the income category to limit the number income category
housing['income_cat'] = np.ceil(housing['median_income'] / 1.5)
# putting everything above 5th category as 5th category
housing['income_cat'].where(housing['income_cat'] < 5, other=5.0, inplace=True)
```

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

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

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 [12]:
```
housing["income_cat"].value_counts() / len(housing)
```

Out[12]:
```
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 [13]:
```
strat_test_set['income_cat'].value_counts() / len(strat_test_set)
```

Out[13]:
```
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
```

as seen above the proportions are maintained in the test set using stratified sampling

[why stratified?] : because the feature-space are less and also because its a mid-sized dataset & we don't want to miss out any class

In [14]:
```
# experimenting: with random sampling now

from sklearn.model_selection import train_test_split

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

In [15]:
```
def income_cat_proportions(data):
    return data['income_cat'].value_counts() / len(data)


comparing_props = pd.DataFrame({
    "Overall Props": income_cat_proportions(housing),
    "Random": income_cat_proportions(test_set),
    "Stratified": income_cat_proportions(strat_test_set)
}).sort_index()

comparing_props["random %error"] = 100 * comparing_props["Random"] / comparing_props["Overall Props"] - 100
comparing_props["strat. %error"] = 100 * comparing_props["Stratified"] / comparing_props["Overall Props"] - 100
comparing_props
```

Out[15]:

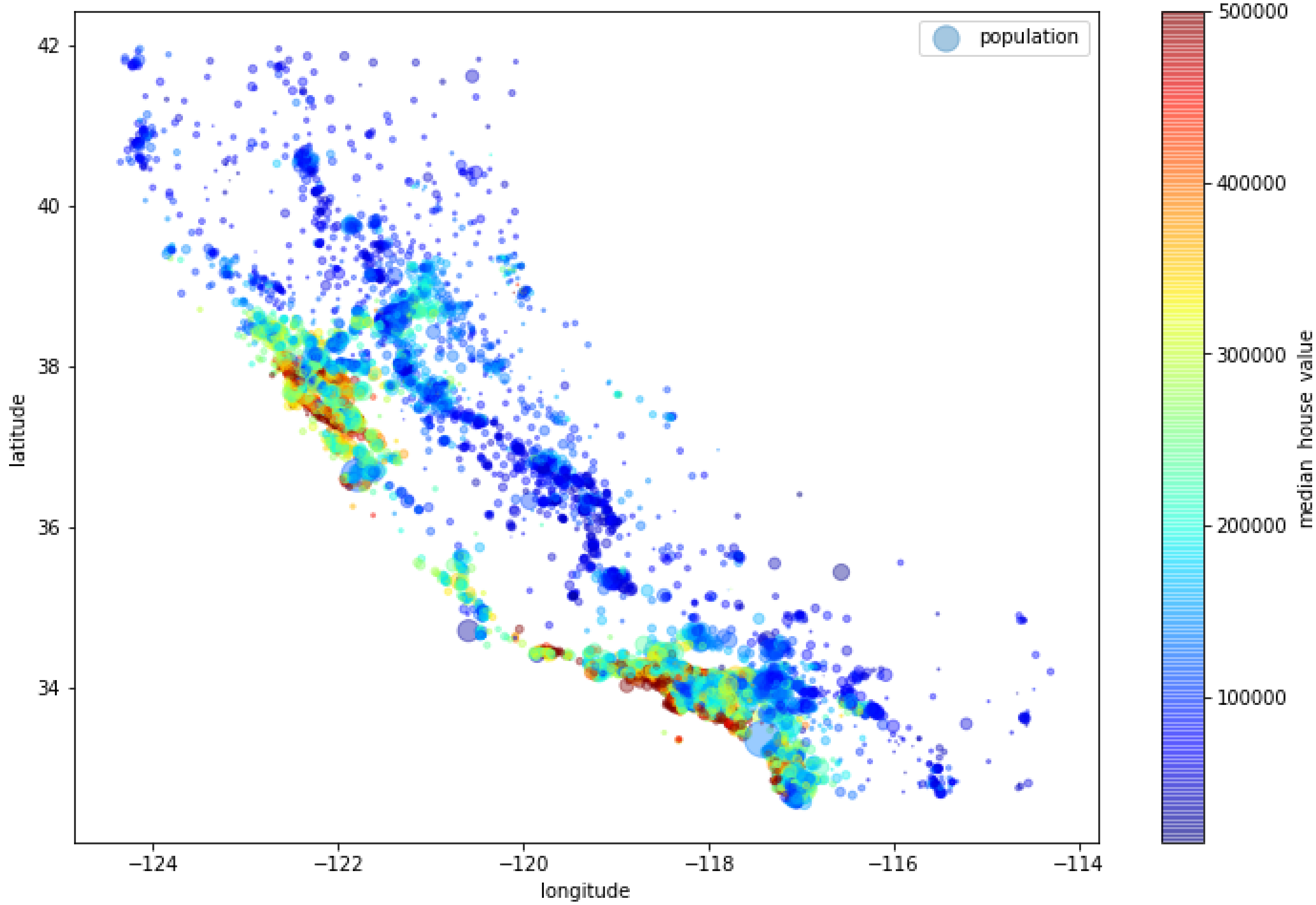
	Overall Props	Random	Stratified	random %error	strat. %error
1.0	0.039826	0.042636	0.039729	7.055961	-0.243309
2.0	0.318847	0.311531	0.318798	-2.294484	-0.015195
3.0	0.350581	0.344719	0.350533	-1.672195	-0.013820
4.0	0.176308	0.181686	0.176357	3.050289	0.027480
5.0	0.114438	0.119428	0.114583	4.360711	0.127011

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

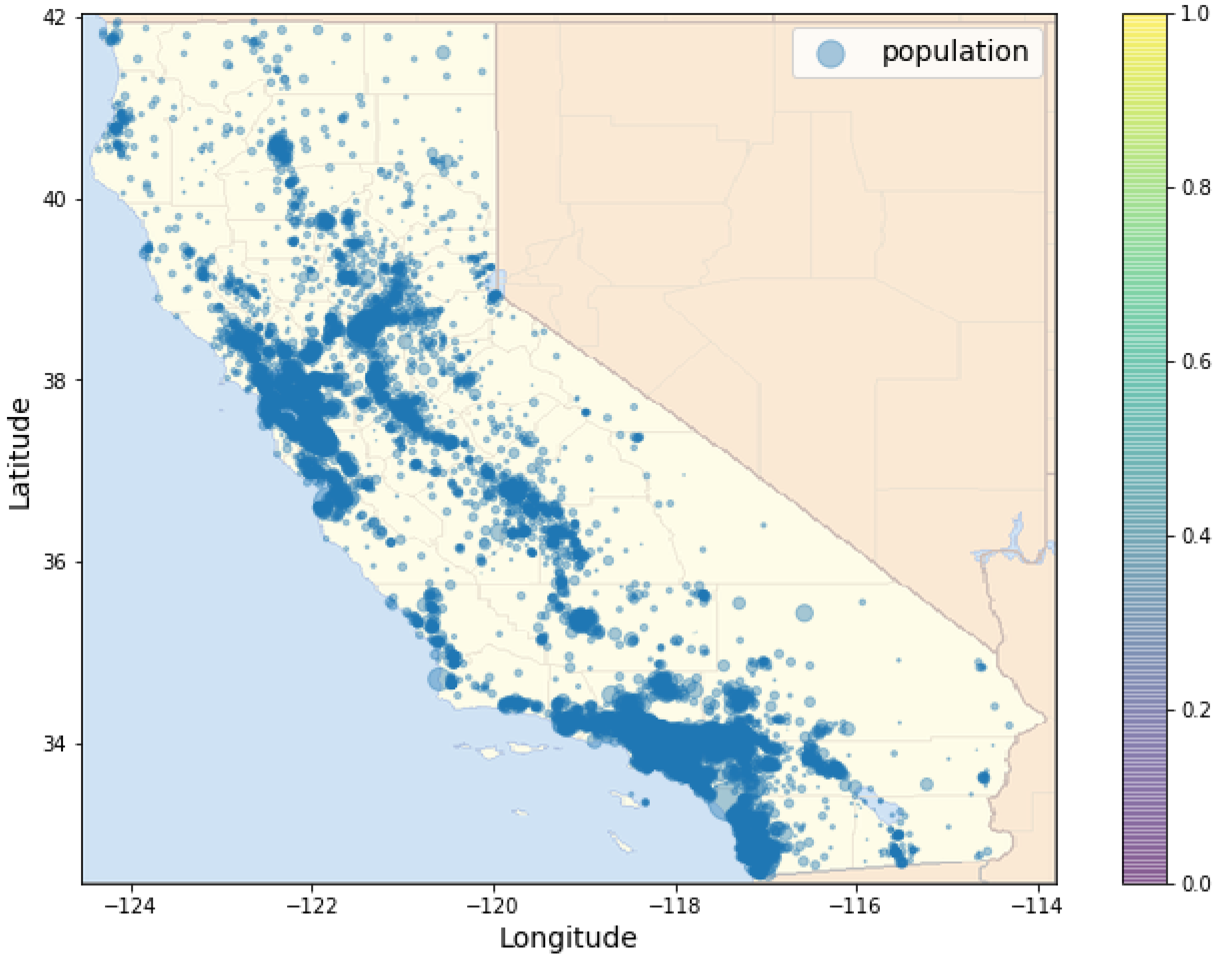
In [17]:
```
housing = strat_train_set.copy()
```

In [18]:
```
housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.4,
             s=housing['population']/100, label="population", figsize=(12,8),
             c="median_house_value", cmap=plt.get_cmap("jet"), sharex=False)

plt.legend()
```

Out[18]: <matplotlib.legend.Legend at 0x14b59f677f0>



In [79]:
```
import matplotlib.image as mpimg

ax = housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.4,
            s=housing['population']/100, label="population", figsize=(12,8),
            cmap=plt.get_cmap("jet"), sharex=False)

plt.legend()
# Load the png image
california_img = mpimg.imread("california.png")

plt.imshow(california_img, extent=[-124.55, -113.8, 32.45, 42.05], alpha=0.5, cmap=plt.get_cmap("jet"))

plt.xlabel("Longitude", fontsize=14)
plt.ylabel("Latitude", fontsize=14)

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



Looking for Correlations

(Pearson's Distance Correlation equation)

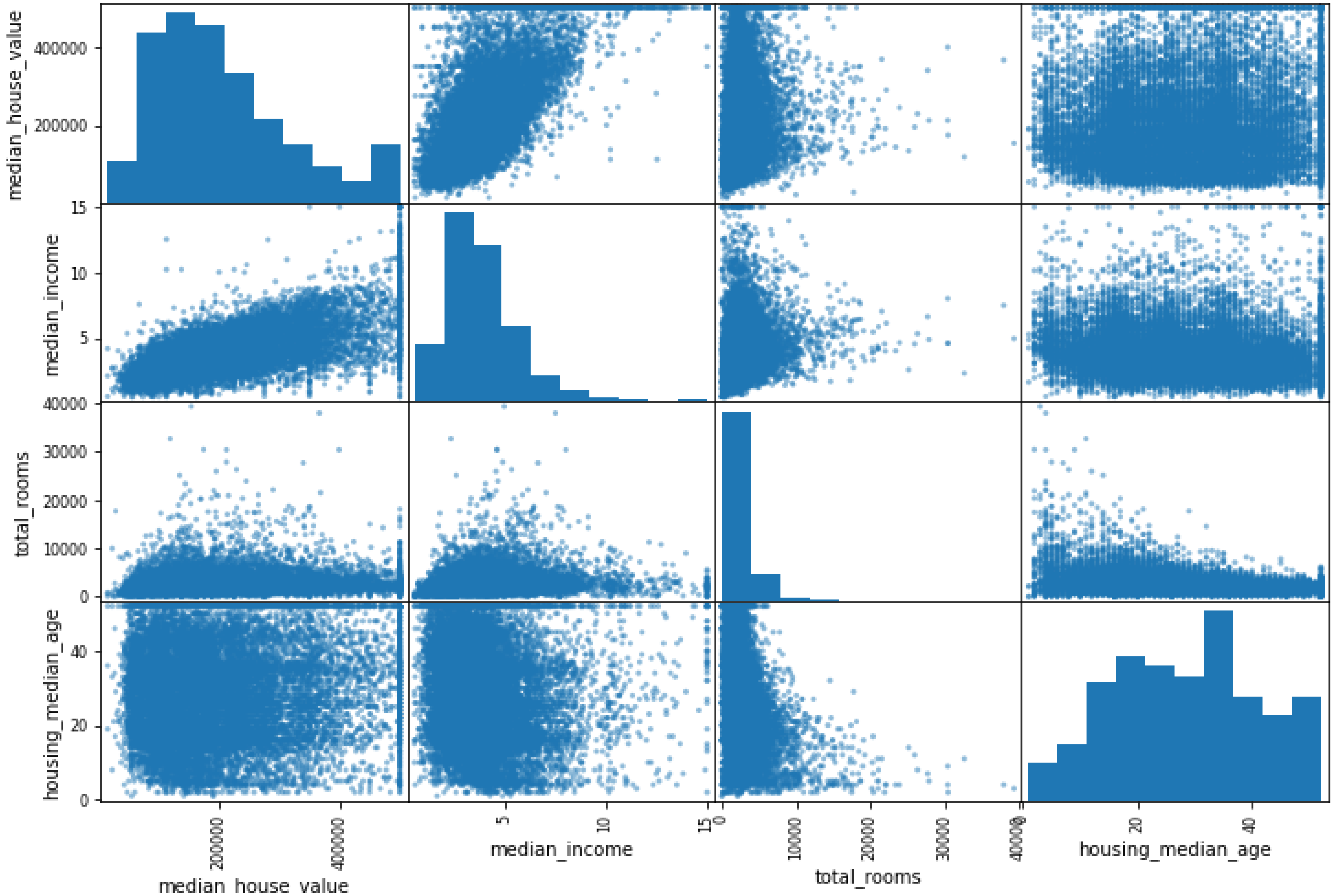
In [20]:
```
# pandas has corr method for calculating correlations
corr_matrix = housing.corr()

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

Out[20]:
```
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)

In [21]:
```
# other approach it to use the scatter plot in a A vs B fashion
# problem with this is that (for N features, there will be N^2 plots)

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))
```

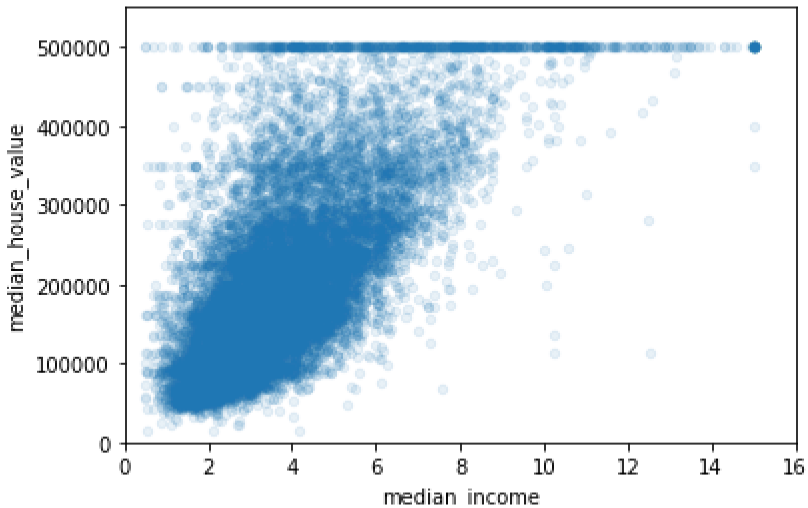
Out[21]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x0000014B5A010B38>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000014B6199AB38>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000014B619BEC88>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000014B619E4DD8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000014B61A0BF28>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000014B61A3A0B8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000014B61A61208>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000014B61A89390>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000014B61A893C8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000014B623D05F8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000014B62403748>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000014B62439898>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000014B59DD4BE0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000014B6248A278>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000014B624BA828>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000014B624EFDD8>]],
      dtype=object)
```



median_house_value
median_income
total_rooms
housing_median_age
400000
200000
15
10
5
40000
30000
20000
10000
0
40
20
0
200000
400000
5
10
15
0
10000
20000
30000
40000
20
40
median_house_value
median_income
total_rooms
housing_median_age

In [22]:
```
housing.plot(kind="scatter", x="median_income", y="median_house_value", alpha=0.1)
plt.axis([0, 16, 0, 550000])
```

Out[22]: [0, 16, 0, 550000]



Feature Engineering

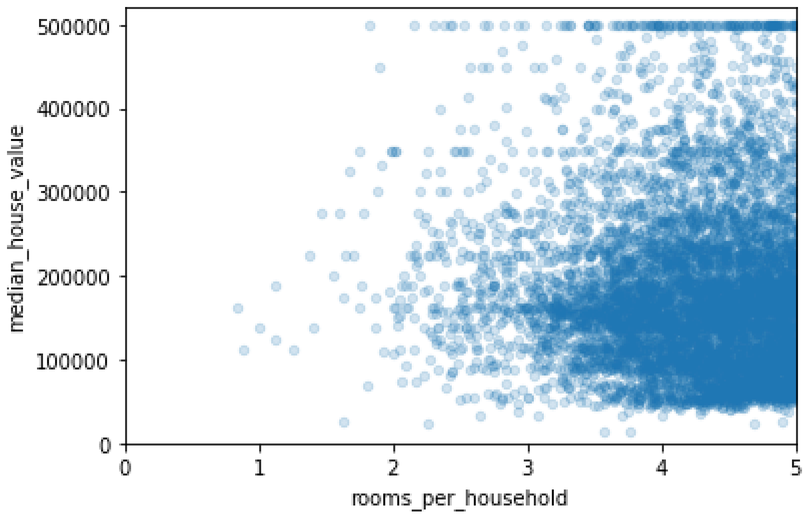
In [23]:
```
housing["bedrooms_per_room"] = housing["total_bedrooms"]/housing["total_rooms"]
housing["population_per_household"] = housing["population"]/housing["households"]
housing["rooms_per_household"] = housing["total_rooms"]/housing["households"]
```

In [24]:
```
corr_matrix = housing.corr()
corr_matrix["median_house_value"].sort_values(ascending=False)
```

Out[24]:
```
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
```

[observation]: the new `bedrooms_per_room` is highly correlated but in a reciprocative way to the `median_house_value` , So the houses with lesser bedroom/room ratio will tend to be more expensive.

In [25]:
```
housing.plot(kind="scatter", x="rooms_per_household", y="median_house_value", alpha=0.2)
plt.axis([0, 5, 0, 520000])
plt.show()
```



In [26]:
```
housing.describe()
```

Out[26]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	mec
count	16512.000000	16512.000000	16512.000000	16512.000000	16349.000000	16512.000000	16512.000000	16512.000000	
mean	-119.574691	35.642798	28.655220	2622.124879	535.192672	1418.447372	496.865492	3.870355	
std	2.005064	2.142773	12.535491	2171.310387	421.124910	1137.484934	382.194550	1.903633	
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	
25%	-121.800000	33.930000	18.000000	1446.000000	295.000000	785.000000	279.000000	2.559725	
50%	-118.500000	34.260000	29.000000	2123.000000	433.000000	1159.000000	407.000000	3.532750	
75%	-118.010000	37.720000	37.000000	3121.250000	641.000000	1715.000000	599.000000	4.739375	
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.000100	

Preparing the data for ML algos

In [27]:
```
housing = strat_train_set.drop("median_house_value", axis=1)
housing_labels = strat_train_set["median_house_value"].copy()
```

some data cleansing

In [28]:
```
# when calculating imputng value on your own
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[28]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity
5654	-118.30	33.73	42.0	1731.0	433.0	866.0	403.0	2.7451	NEAR OCEAN
14930	-117.02	32.66	19.0	771.0	433.0	376.0	108.0	6.6272	NEAR OCEAN
9814	-121.93	36.62	34.0	2351.0	433.0	1063.0	428.0	3.7250	NEAR OCEAN
14986	-117.03	32.73	34.0	2061.0	433.0	1169.0	400.0	3.5096	NEAR OCEAN
4767	-118.37	34.03	37.0	1236.0	433.0	966.0	292.0	3.0694	<1H OCEAN

In [29]:
```
# when using Scikit-Learn's Imputer class
from sklearn.impute import SimpleImputer

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

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

imputer.fit(housing_num)
```

Out[30]:
```
SimpleImputer(add_indicator=False, copy=True, fill_value=None,
              missing_values=nan, strategy='median', verbose=0)
```

In [31]:
```
# Imputer basically computes across all the attributes, so if you wanna see this across all the attributes, just
imputer.statistics_
```

Out[31]:
```
array([-118.5    ,   34.26   ,   29.     , 2123.     ,  433.     ,
       1159.     ,  407.     ,    3.53275])
```

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

Out[32]:
```
array([-118.5    ,   34.26   ,   29.     , 2123.     ,  433.     ,
       1159.     ,  407.     ,    3.53275])
```

using the imputer we created above, transforming the training set by replacing the missing values by the learned medians

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

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

In [35]:
```
# cross check for missing value
housing_tr[housing_tr.isnull().any(axis=1)]
```

Out[35]:

longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income

In [36]:
```
housing_tr.head()
```

Out[36]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
0	-118.09	33.92	35.0	1994.0	419.0	1491.0	428.0	3.7383
1	-122.57	37.96	52.0	3458.0	468.0	1449.0	471.0	9.1834
2	-121.96	36.97	23.0	4324.0	1034.0	1844.0	875.0	3.0777
3	-118.28	34.02	52.0	281.0	103.0	470.0	96.0	1.9375
4	-116.50	33.81	26.0	5032.0	1229.0	3086.0	1183.0	2.5399

handling categorical values

In [37]:
```
housing_cat = housing["ocean_proximity"]
housing_cat.head(10)
```

Out[37]:
```
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 [38]:
```
# using pandas's own factorize() method to convert them into categorical features
housing_cat_encoded, housing_categories = housing_cat.factorize()
```

In [39]:
```
housing_cat_encoded[:10]
```

Out[39]:
```
array([0, 1, 2, 0, 3, 2, 0, 2, 0, 2], dtype=int64)
```

In [40]:
```
housing_categories
```

Out[40]:
```
Index(['<1H OCEAN', 'NEAR BAY', 'NEAR OCEAN', 'INLAND', 'ISLAND'], dtype='object')
```

In [41]:
```
# 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:\Users\SUNNY\Anaconda3\lib\site-packages\sklearn\preprocessing\_encoders.py:415: FutureWarning: The handling
of integer data will change in version 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 unique 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 categories to integers, then you can
now use the OneHotEncoder directly.
  warnings.warn(msg, FutureWarning)
```

In [42]:
```
housing_cat_1hot
```

Out[42]:
```
<1x16512 sparse matrix of type '<class 'numpy.float64'>'
	with 16512 stored elements in Compressed Sparse Row format>
```

In [43]:
```
# since 1 hot encoder returns a sparse matrix, need to change it to a dense array
housing_cat_1hot.toarray()
```

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

Custom Transformations

In [44]:
```
from sklearn.base import BaseEstimator, TransformerMixin

#column indexes
rooms_ix, bedrooms_ix, population_ix, household_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 to do here

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

        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:, bedrooms_ix] / X[:, rooms_ix]
            return np.c_[X, rooms_per_household, population_per_household, bedrooms_per_room]
        else:
            return np.c_[X, rooms_per_household, population_per_household]
```

In [45]:
```
attr_adder = CombinedAttributesAdder(add_bedrooms_per_room=False)
housing_extra_attribs = attr_adder.transform(housing.values)
```

In [46]:
```
housing_extra_attribs = pd.DataFrame(housing_extra_attribs, columns=list(housing.columns)+["rooms_per_household'
                                                                                          "population_per_house

housing_extra_attribs.head()
```

Out[46]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity	room
0	-118.09	33.92	35	1994	419	1491	428	3.7383	<1H OCEAN	
1	-122.57	37.96	52	3458	468	1449	471	9.1834	NEAR BAY	
2	-121.96	36.97	23	4324	1034	1844	875	3.0777	NEAR OCEAN	
3	-118.28	34.02	52	281	103	470	96	1.9375	<1H OCEAN	
4	-116.5	33.81	26	5032	1229	3086	1183	2.5399	INLAND	

Setting up Pipeline for all the preprocessings

In [47]:
```
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[47]:
```
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 [48]:
```
class DataFrameSelector(BaseEstimator, TransformerMixin):

    def __init__(self, attribute_names):
        self.attibute_names = attribute_names

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

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

In [49]:
```
# complete Pipeline

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

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

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

In [50]:
```
from sklearn.pipeline import FeatureUnion

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

In [51]:
```
housing_prepared = full_pipeline.fit_transform(housing)
housing_prepared
```

Out[51]:
```
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 [52]:
```
from sklearn.linear_model import LinearRegression

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

Out[52]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [53]:
```
# trying the full pipeline on a few training instances

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

some_data_prepared = full_pipeline.transform(some_data)
```

In [54]:
```
print("Prediction: ", lin_reg.predict(some_data_prepared))
print("Actual Labels: ", list(some_labels))
```

```
Prediction:  [209526.30110297 455497.76141409 252936.22210586 173615.33127943
 114294.56522481]
Actual Labels:  [166200.0, 500001.0, 263800.0, 38800.0, 94800.0]
```

In [55]:
```
from sklearn.metrics import mean_squared_error
housing_predictions = lin_reg.predict(housing_prepared)

lin_mse = mean_squared_error(housing_labels, housing_predictions)
lin_rmse = np.sqrt(lin_mse)
lin_rmse
```

Out[55]:
```
67949.91466225038
```

In [56]:
```
from sklearn.tree import DecisionTreeRegressor

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

Out[56]:
```
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 [57]:
```
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[57]:
```
0.0
```

Cross Validation:

In [58]:
```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(tree_reg, housing_prepared, housing_labels, cv=10, scoring="neg_mean_squared_error")

tree_rmse_scores = np.sqrt(-scores)
```

In [59]:
```
def display_scores(scores):
    print("scores: ", scores)
    print("mean: ", scores.mean())
    print("std deviation: ", scores.std())


display_scores(tree_rmse_scores)
```

```
scores:  [71226.79952277 69435.61534003 67609.26793452 71107.39038735
 69151.95011626 67960.45688827 71096.76335259 69863.56579159
 67629.18175783 70711.71224163]
mean:  69579.27033328432
std deviation:  1386.473518494819
```

In [60]:
```
lin_scores = cross_val_score(lin_reg, housing_prepared, housing_labels, cv=10, scoring="neg_mean_squared_error")

lin_rmse_scores = np.sqrt(-lin_scores)

display_scores(lin_rmse_scores)
```

```
scores:  [67641.22210761 69245.155892   65690.83401976 67581.651926
 66586.04760743 66937.30771561 67397.33645629 69807.64170261
 66660.63451034 74883.89423608]
mean:  68243.17261737352
std deviation:  2500.7262162919783
```

In [61]:
```
from sklearn.ensemble import RandomForestRegressor

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

```
C:\Users\SUNNY\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:245: FutureWarning: The default value of
n_estimators will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

Out[61]:
```
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 [62]:
```
housing_pred = forest_reg.predict(housing_prepared)

forest_scores = cross_val_score(lin_reg, housing_prepared, housing_labels, cv=10, scoring="neg_mean_squared_erro

forest_rmse_scores = np.sqrt(-forest_scores)

display_scores(forest_rmse_scores)
```

```
scores:  [67641.22210761 69245.155892   65690.83401976 67581.651926
 66586.04760743 66937.30771561 67397.33645629 69807.64170261
 66660.63451034 74883.89423608]
mean:  68243.17261737352
std deviation:  2500.7262162919783
```

Fine Tuning Model:

In [63]:
```
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_reg = RandomForestRegressor()

grid_search = GridSearchCV(rf_reg, param_grid, cv=5, scoring="neg_mean_squared_error")

grid_search.fit(housing_prepared, housing_labels)
```

Out[63]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=RandomForestRegressor(bootstrap=True, criterion='mse',
                                             max_depth=None,
                                             max_features='auto',
                                             max_leaf_nodes=None,
                                             min_impurity_decrease=0.0,
                                             min_impurity_split=None,
                                             min_samples_leaf=1,
                                             min_samples_split=2,
                                             min_weight_fraction_leaf=0.0,
                                             n_estimators='warn', n_jobs=None,
                                             oob_score=False, random_state=None,
                                             verbose=0, warm_start=False),
             iid='warn', n_jobs=None,
             param_grid=[{'max_features': [2, 4, 6, 8],
                          'n_estimators': [3, 10, 30]},
                         {'bootstrap': [False], 'max_features': [2, 3, 4],
                          'n_estimators': [3, 10]}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='neg_mean_squared_error', verbose=0)
```

In [64]:
```
# to get the best combination of hyperparameters
grid_search.best_params_
```

Out[64]:
```
{'max_features': 6, 'n_estimators': 30}
```

In [65]:
```
# to get the best estimators directly
grid_search.best_estimator_
```

Out[65]:
```
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 [66]:
```
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)
```

```
63498.91381984168 {'max_features': 2, 'n_estimators': 3}
54800.75545492814 {'max_features': 2, 'n_estimators': 10}
52081.795518990315 {'max_features': 2, 'n_estimators': 30}
59792.96367173375 {'max_features': 4, 'n_estimators': 3}
52263.453456263975 {'max_features': 4, 'n_estimators': 10}
49890.61143971104 {'max_features': 4, 'n_estimators': 30}
57820.899823920554 {'max_features': 6, 'n_estimators': 3}
51229.02293941957 {'max_features': 6, 'n_estimators': 10}
49374.85238709492 {'max_features': 6, 'n_estimators': 30}
58104.07010180327 {'max_features': 8, 'n_estimators': 3}
51699.49555865867 {'max_features': 8, 'n_estimators': 10}
49403.986641015465 {'max_features': 8, 'n_estimators': 30}
61765.72922854422 {'bootstrap': False, 'max_features': 2, 'n_estimators': 3}
53256.639893301406 {'bootstrap': False, 'max_features': 2, 'n_estimators': 10}
59604.42372070559 {'bootstrap': False, 'max_features': 3, 'n_estimators': 3}
51156.98722333856 {'bootstrap': False, 'max_features': 3, 'n_estimators': 10}
57428.54324039999 {'bootstrap': False, 'max_features': 4, 'n_estimators': 3}
51367.46773300634 {'bootstrap': False, 'max_features': 4, 'n_estimators': 10}
```

In [67]:
```
pd.DataFrame(grid_search.cv_results_)
```

Out[67]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_features	param_n_estimators	param_bootstrap	para
0	0.090237	0.007437	0.004190	0.000400	2	3	NaN	{'max_featur 'n_estimato
1	0.308077	0.010644	0.013364	0.001196	2	10	NaN	{'max_featur 'n_estimato
2	0.784717	0.041622	0.031825	0.002666	2	30	NaN	{'max_featur 'n_estimato
3	0.137646	0.007027	0.004191	0.000399	4	3	NaN	{'max_featur 'n_estimato
4	0.434982	0.033517	0.012767	0.003051	4	10	NaN	{'max_featur 'n_estimato
5	1.242009	0.074312	0.033413	0.007246	4	30	NaN	{'max_featur 'n_estimato
6	0.187911	0.024383	0.003997	0.000641	6	3	NaN	{'max_featur 'n_estimato
7	0.617071	0.021437	0.010965	0.000015	6	10	NaN	{'max_featur 'n_estimato
8	1.823105	0.152211	0.035118	0.003530	6	30	NaN	{'max_featur 'n_estimato

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_features	param_n_estimators	param_bootstrap	para
9	0.220548	0.001245	0.003990	0.000620	8	3	NaN	{'max_featur 'n_estimato
10	0.771724	0.027090	0.011570	0.000798	8	10	NaN	{'max_featur 'n_estimato
11	2.552393	0.227175	0.038099	0.003477	8	30	NaN	{'max_featur 'n_estimato
12	0.204653	0.021598	0.005786	0.001323	2	3	False	{'bootstr Fa 'max_featur 2, 'n_e
13	0.479447	0.065911	0.015156	0.000404	2	10	False	{'bootstr Fa 'max_featur 2, 'n_e
14	0.177957	0.010908	0.005181	0.000748	3	3	False	{'bootstr Fa 'max_featur 3, 'n_e
15	0.592120	0.039514	0.013777	0.001161	3	10	False	{'bootstr Fa 'max_featur 3, 'n_e
16	0.213499	0.005090	0.004603	0.000474	4	3	False	{'bootstr Fa 'max_featur 4, 'n_e
17	0.695077	0.013857	0.014149	0.001128	4	10	False	{'bootstr Fa 'max_featur 4, 'n_e

In [68]:
```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

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

rf_reg = RandomForestRegressor(random_state=29)

rnd_search = RandomizedSearchCV(rf_reg, param_distributions=params_distibs, n_iter=10,
                                cv=5, scoring="neg_mean_squared_error", random_state=29)

rnd_search.fit(housing_prepared, housing_labels)
```

Out[68]:
```
RandomizedSearchCV(cv=5, error_score='raise-deprecating',
                   estimator=RandomForestRegressor(bootstrap=True,
                                                   criterion='mse',
                                                   max_depth=None,
                                                   max_features='auto',
                                                   max_leaf_nodes=None,
                                                   min_impurity_decrease=0.0,
                                                   min_impurity_split=None,
                                                   min_samples_leaf=1,
                                                   min_samples_split=2,
                                                   min_weight_fraction_leaf=0.0,
                                                   n_estimators='warn',
                                                   n_jobs=None, oob_score=False,
                                                   random_sta...
                                                   warm_start=False),
                   iid='warn', n_iter=10, n_jobs=None,
                   param_distributions={'max_features': <scipy.stats._distn_infrastructure.rv_frozen object at
0x0000014B646D2B00>,
                                        'n_estimators': <scipy.stats._distn_infrastructure.rv_frozen object at
0x0000014B646D23C8>},
                   pre_dispatch='2*n_jobs', random_state=29, refit=True,
                   return_train_score=False, scoring='neg_mean_squared_error',
                   verbose=0)
```

In [69]:
```
cvres = rnd_search.cv_results_

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

```
48554.82808753229 {'max_features': 6, 'n_estimators': 116}
49443.37797311243 {'max_features': 5, 'n_estimators': 35}
53570.23831992085 {'max_features': 1, 'n_estimators': 97}
50902.01282842812 {'max_features': 2, 'n_estimators': 114}
53580.16560373702 {'max_features': 1, 'n_estimators': 98}
48632.80337452587 {'max_features': 7, 'n_estimators': 95}
53454.896572172365 {'max_features': 1, 'n_estimators': 156}
48524.00649818885 {'max_features': 6, 'n_estimators': 149}
48560.27239481039 {'max_features': 7, 'n_estimators': 152}
53446.573293355774 {'max_features': 1, 'n_estimators': 165}
```

In [70]:
```
feature_importances = grid_search.best_estimator_.feature_importances_
feature_importances
```

Out[70]:
```
array([7.15708614e-02, 6.67976684e-02, 4.35429693e-02, 1.78455574e-02,
       1.55788897e-02, 1.78536251e-02, 1.53464854e-02, 3.24182235e-01,
       5.57806856e-02, 1.03380022e-01, 8.55612508e-02, 1.03386819e-02,
       1.65253175e-01, 1.43882022e-04, 2.45132811e-03, 4.37268296e-03])
```

In [71]:
```
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), reverse=True)
```

Out[71]:
```
[(0.3241822348473802, 'median_income'),
 (0.1652531749723918, 'INLAND'),
 (0.10338002205223705, 'pop_per_hhold'),
 (0.08556125084061317, 'bedrooms_per_room'),
 (0.0715708613619816, 'longitude'),
 (0.06679766843212846, 'latitude'),
 (0.05578068560123286, 'rooms_per_hhold'),
 (0.043542969306187874, 'housing_median_age'),
 (0.017853625060891214, 'population'),
 (0.017845557410279773, 'total_rooms'),
 (0.015578889740177322, 'total_bedrooms'),
 (0.015346485394769422, 'households'),
 (0.010338681892642873, '<1H OCEAN'),
 (0.0043726829553654405, 'NEAR OCEAN'),
 (0.002451328109242547, 'NEAR BAY'),
 (0.0001438820224784505, 'ISLAND')]
```

In [75]:
```
final_model = grid_search.best_estimator_

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

X_test_prepared = full_pipeline.transform(X_test)
final_predictions = final_model.predict(X_test_prepared)

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

In [ ]:

In [ ]:

In [ ]:

In [ ]:

```

```

In [ ]:

```

```