Housing

April 20, 2021

1 Chapter 2. End-to-end machine learning project

The purpose of this project is to predict median house values in Californian districts, given a number of features from these districts.

1.1 Main steps

1. Look at the big picture Frame de problem Select a performance measure Check the assumptions
2. Get the data
3. Discover and visualize the data to gain insights.
4. Prepare the data for Machine Learning algorithms.
5. Select a model and train it.
6. Fine-tune the model.
7. Predict the values.

1.2 Get the data

[44]:
```
import os
import tarfile
import urllib

DOWNLOAD_ROOT = "https://raw.githubusercontent.com/ageron/handson-ml2/master/"
HOUSING_PATH = os.path.join("datasets", "housing")
HOUSING_URL = DOWNLOAD_ROOT + "datasets/housing/housing.tgz"

def fetch_housing_data(housing_url=HOUSING_URL, housing_path=HOUSING_PATH):
    if not os.path.isdir(housing_path):
        os.makedirs(housing_path)
    tgz_path = os.path.join(housing_path, "housing.tgz")
    urllib.request.urlretrieve(housing_url, tgz_path)
    housing_tgz = tarfile.open(tgz_path)
    housing_tgz.extractall(path=housing_path)
    housing_tgz.close()
```

[45]:
```
fetch_housing_data()
```

[46]:
```python
import pandas as pd

def load_housing_data(housing_path=HOUSING_PATH):
    csv_path = os.path.join(housing_path, "housing.csv")
    return pd.read_csv(csv_path)
```

[47]:
```python
housing = load_housing_data()
housing.head()
```

[47]:
```
   longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
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

   population  households  median_income  median_house_value ocean_proximity
0       322.0       126.0         8.3252            452600.0        NEAR BAY
1      2401.0      1138.0         8.3014            358500.0        NEAR BAY
2       496.0       177.0         7.2574            352100.0        NEAR BAY
3       558.0       219.0         5.6431            341300.0        NEAR BAY
4       565.0       259.0         3.8462            342200.0        NEAR BAY
```

[48]:
```python
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
```

[49]:
```python
housing["ocean_proximity"].value_counts()
```

[49]:
```
<1H OCEAN     9136
INLAND        6551
```

```
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: ocean_proximity, dtype: int64
```

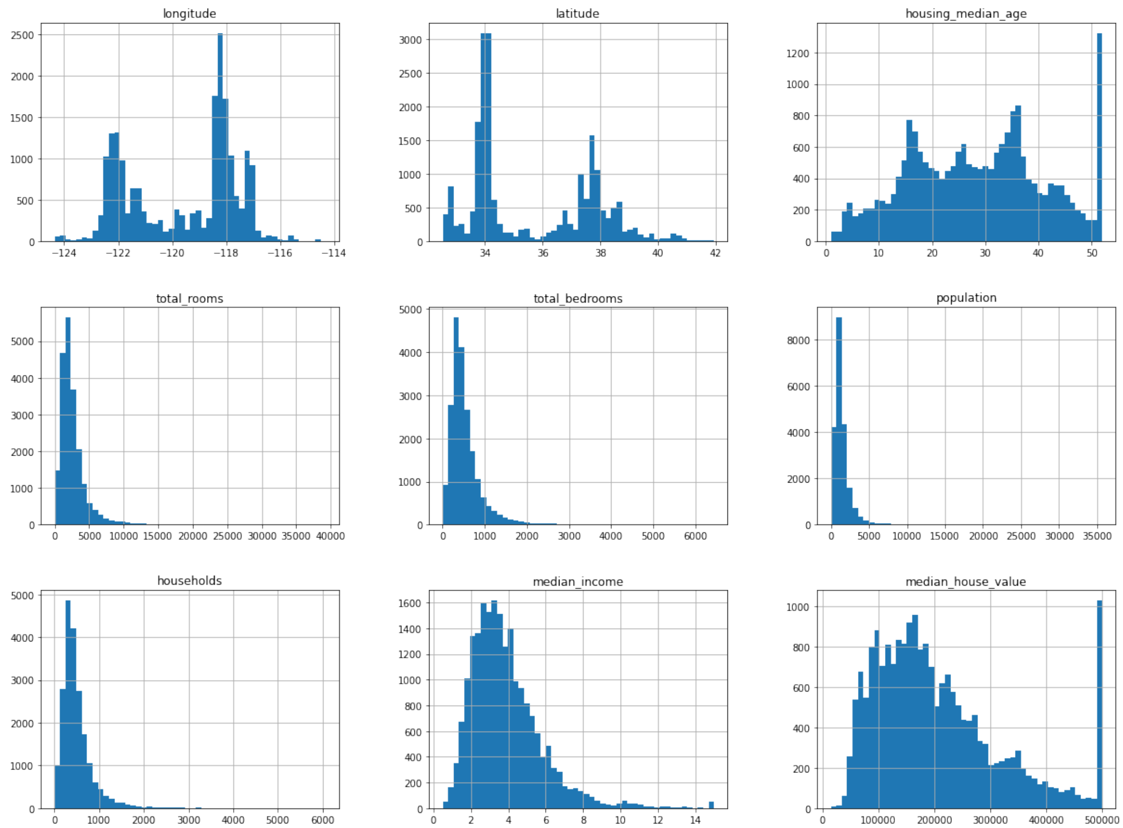
[50]:
```python
housing.groupby("ocean_proximity").size()
```

[50]:
```
ocean_proximity
<1H OCEAN     9136
INLAND        6551
ISLAND           5
NEAR BAY      2290
NEAR OCEAN    2658
dtype: int64
```

[51]:
```python
housing.describe()
```

[51]:
```
          longitude      latitude  housing_median_age   total_rooms  \
count  20640.000000  20640.000000        20640.000000  20640.000000
mean    -119.569704     35.631861           28.639486   2635.763081
std        2.003532      2.135952           12.585558   2181.615252
min     -124.350000     32.540000            1.000000      2.000000
25%     -121.800000     33.930000           18.000000   1447.750000
50%     -118.490000     34.260000           29.000000   2127.000000
75%     -118.010000     37.710000           37.000000   3148.000000
max     -114.310000     41.950000           52.000000  39320.000000

       total_bedrooms    population    households  median_income  \
count    20433.000000  20640.000000  20640.000000   20640.000000
mean       537.870553   1425.476744    499.539680       3.870671
std        421.385070   1132.462122    382.329753       1.899822
min          1.000000      3.000000      1.000000       0.499900
25%        296.000000    787.000000    280.000000       2.563400
50%        435.000000   1166.000000    409.000000       3.534800
75%        647.000000   1725.000000    605.000000       4.743250
max       6445.000000  35682.000000   6082.000000      15.000100

       median_house_value
count        20640.000000
mean        206855.816909
std         115395.615874
min          14999.000000
25%         119600.000000
50%         179700.000000
75%         264725.000000
max         500001.000000
```

[52]:
```
%matplotlib inline
import matplotlib.pyplot as plt
housing.hist(bins = 50, figsize=(20,15))
plt.show()
```



Split the data into training set and test set Example 1 of splif function

[53]:
```
import numpy as np
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]
```

Example 2 of split function

[54]:
```
from zlib import crc32
def test_set_check(identifier, test_ratio):
    return crc32(np.int64(identifier)) & 0xffffffff < test_ratio * 2**32
```

```python
def split_train_test_by_id(data, test_ratio, id_column):
    ids = data[id_column]
    in_test_set = ids.apply(lambda id_: test_set_check(id_, test_ratio))
    return data.loc[~in_test_set], data.loc[in_test_set]
```

Example with scikit learn

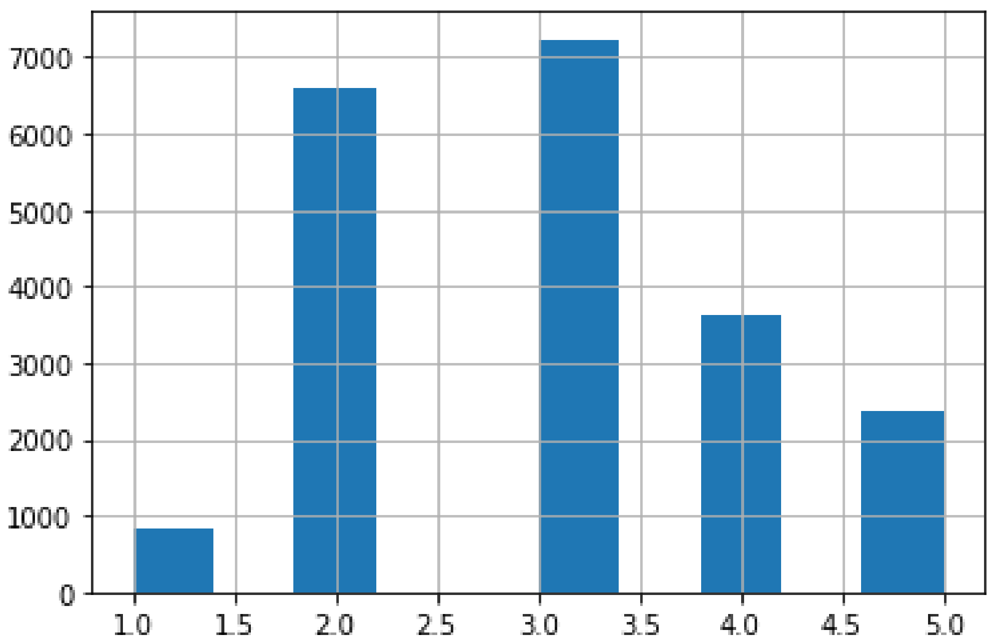
[55]:
```python
from sklearn.model_selection import train_test_split

#train_set, test_set = train_test_split(housing, test_size=0.2, random_state = 42)
```

Stratified sampling: Is a technique used to guarantee that the test set is representative of the overall population.

[56]:
```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])
```

[57]:
```python
housing["income_cat"].hist()
```

[57]: <AxesSubplot:>



[58]:
```python
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]
```

[59]:
```
strat_test_set["income_cat"].value_counts() / len(strat_test_set)
```

[59]:
```
3    0.350533
2    0.318798
4    0.176357
5    0.114583
1    0.039729
Name: income_cat, dtype: float64
```

[60]:
```
for set_ in (strat_train_set, strat_test_set):
    set_.drop("income_cat", axis = 1, inplace = True)
```

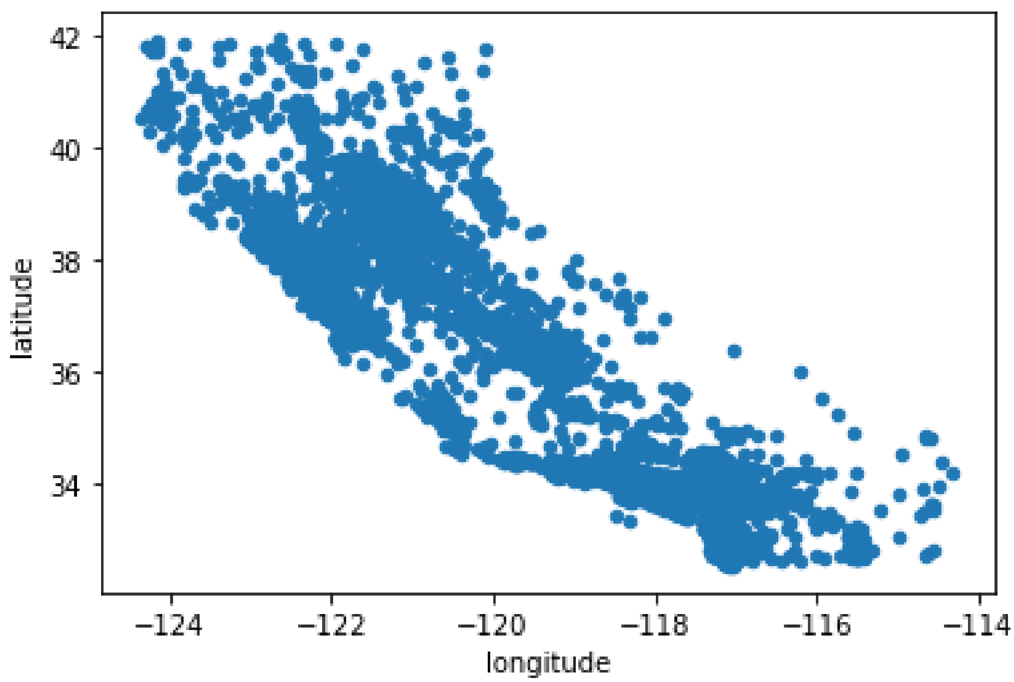
1.3 Discover and visualize the data to gain insights

[61]:
```
housing = strat_train_set.copy()
```

1.3.1 Visualizing Geographical Data

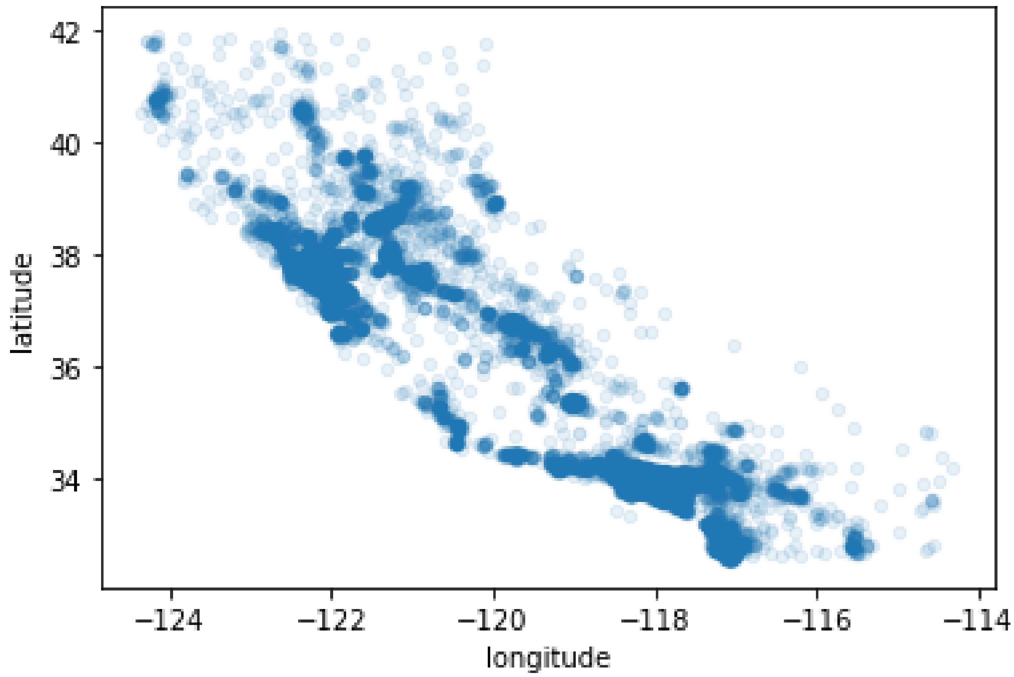
[62]:
```
housing.plot(kind = "scatter", x = "longitude", y = "latitude")
```

[62]: <AxesSubplot:xlabel='longitude', ylabel='latitude'>



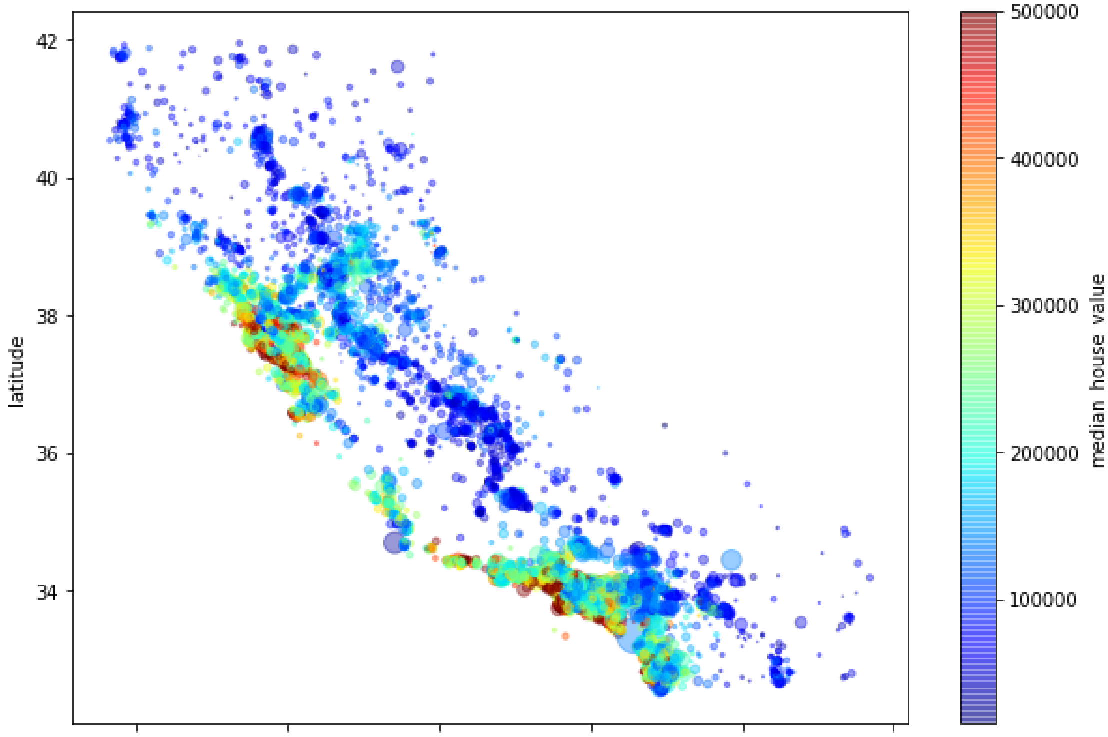
[63]:
```python
#Vishousing.plot(kind = "scatter", x = "longitude", y = "latitude")ualize the␣
↪places with high density of data points
housing.plot(kind = "scatter", x = "longitude", y = "latitude", alpha = 0.1)
```

[63]: <AxesSubplot:xlabel='longitude', ylabel='latitude'>



[64]:
```python
housing.plot(kind = "scatter", x = "longitude", y="latitude", alpha=0.4,
             s=housing["population"]/100, figsize=(10,7),␣
↪c="median_house_value",
             cmap=plt.get_cmap("jet"), colorbar=True)
```

[64]: <AxesSubplot:xlabel='longitude', ylabel='latitude'>



1.3.2 Looking for Correlations

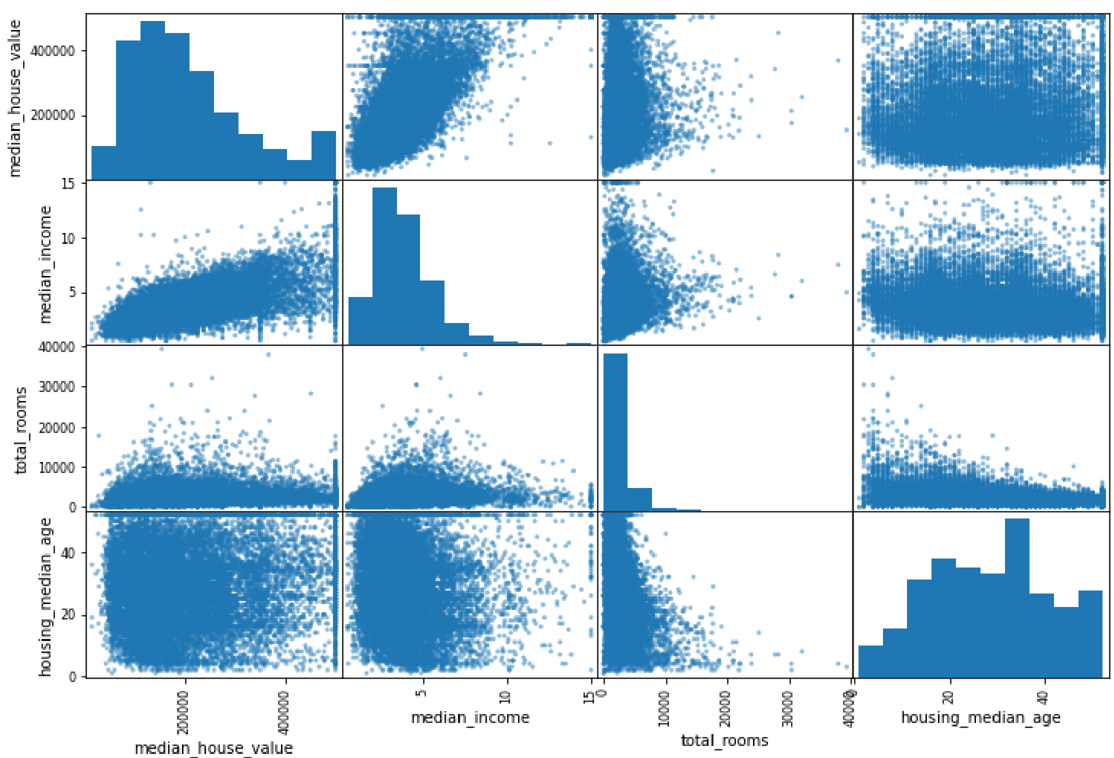
[65]:
```
corr_matrix = housing.corr()
```

[66]:
```
corr_matrix["median_house_value"].sort_values(ascending = False)
```

[66]:
```
median_house_value    1.000000
median_income         0.687160
total_rooms           0.135097
housing_median_age    0.114110
households            0.064506
total_bedrooms        0.047689
population           -0.026920
longitude            -0.047432
latitude             -0.142724
Name: median_house_value, dtype: float64
```

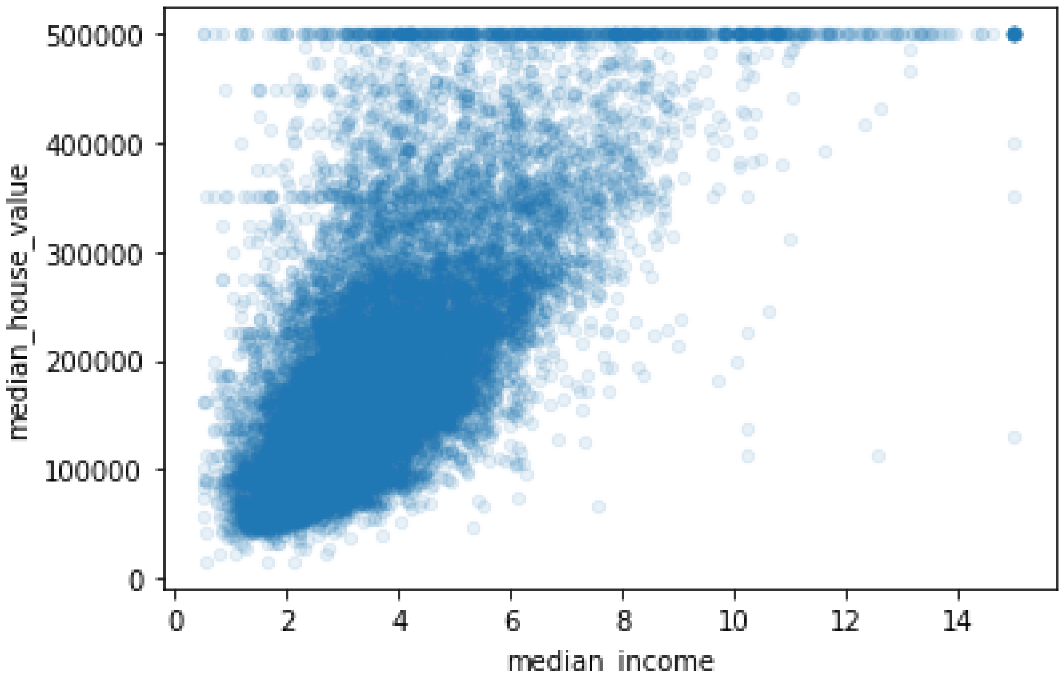
[67]:
```
from pandas.plotting import scatter_matrix

attributes = ["median_house_value", "median_income",␣
 ↪"total_rooms","housing_median_age"]

scatter_matrix(housing[attributes], figsize=(12,8))
```

[67]:
```
array([[<AxesSubplot:xlabel='median_house_value', ylabel='median_house_value'>,
        <AxesSubplot:xlabel='median_income', ylabel='median_house_value'>,
        <AxesSubplot:xlabel='total_rooms', ylabel='median_house_value'>,
        <AxesSubplot:xlabel='housing_median_age', ylabel='median_house_value'>],
       [<AxesSubplot:xlabel='median_house_value', ylabel='median_income'>,
        <AxesSubplot:xlabel='median_income', ylabel='median_income'>,
        <AxesSubplot:xlabel='total_rooms', ylabel='median_income'>,
        <AxesSubplot:xlabel='housing_median_age', ylabel='median_income'>],
       [<AxesSubplot:xlabel='median_house_value', ylabel='total_rooms'>,
        <AxesSubplot:xlabel='median_income', ylabel='total_rooms'>,
        <AxesSubplot:xlabel='total_rooms', ylabel='total_rooms'>,
        <AxesSubplot:xlabel='housing_median_age', ylabel='total_rooms'>],
       [<AxesSubplot:xlabel='median_house_value', ylabel='housing_median_age'>,
        <AxesSubplot:xlabel='median_income', ylabel='housing_median_age'>,
        <AxesSubplot:xlabel='total_rooms', ylabel='housing_median_age'>,
        <AxesSubplot:xlabel='housing_median_age',
ylabel='housing_median_age'>]],
      dtype=object)
```



[68]:
```
housing.plot(kind="scatter", x="median_income", y="median_house_value", alpha =␣
↪0.1)
```

[68]: <AxesSubplot:xlabel='median_income', ylabel='median_house_value'>



The cap in the median_house value is seen as hortizontal lines. This needs to be changed.

1.3.3 Experimenting with Attribute Combinations

New attributes

[69]:
```
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"]
```

Correlation matrix

[70]:
```
corr_matrix = housing.corr()
corr_matrix["median_house_value"].sort_values(ascending=False)
```

[70]:
```
median_house_value          1.000000
median_income               0.687160
rooms_per_household         0.146285
total_rooms                 0.135097
housing_median_age          0.114110
households                  0.064506
total_bedrooms              0.047689
population_per_household   -0.021985
population                 -0.026920
longitude                  -0.047432
```

```
latitude                     -0.142724
bedrooms_per_room            -0.259984
Name: median_house_value, dtype: float64
```

1.4 Prepare the data for machine learning algorithms

[71]:
```
#Another copy of original training set, splited in to x values and y
housing = strat_train_set.drop("median_house_value", axis=1)
housing_labels = strat_train_set["median_house_value"].copy()
```

1.4.1 Data cleaning

Handling missing numeric values

[72]:
```
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy="median")
housing_num = housing.drop("ocean_proximity", axis=1)

imputer.fit(housing_num)

#imputer.statistics_
#housing_num.median().values

X = imputer.transform(housing_num)
housing_tr = pd.DataFrame(X, columns = housing_num.columns, index=housing_num.
 ↪index)
```

Handling text and categorical values

[73]:
```
housing_cat = housing[["ocean_proximity"]]
housing_cat.head(10)
```

[73]:
```
      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
```

Ordinal Encoder -> is useful when two nearby values are more similar than two distant values. Like (“bad”,“average”,“good”,“excelent”)

```
[74]: from sklearn.preprocessing import OrdinalEncoder
ordinal_encoder = OrdinalEncoder()
housing_cat_encoded = ordinal_encoder.fit_transform(housing_cat)
housing_cat_encoded[:10]
```

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

```
[75]: ordinal_encoder.categories_
```

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

One-hot encoder

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

```
[76]: <16512x5 sparse matrix of type '<class 'numpy.float64'>'
        with 16512 stored elements in Compressed Sparse Row format>
```

```
[77]: ordinal_encoder.categories_
```

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

1.4.2 Custom Transformers

```
[78]: 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): # no *args or **kargs
        self.add_bedrooms_per_room = add_bedrooms_per_room
```

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

    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]
```

[79]:
```
attr_adder = CombinedAttributesAdder(add_bedrooms_per_room=False)
housing_extra_attribs = attr_adder.transform(housing.values)
```

1.4.3 Transformation Pipelines

Pipelines are useful to automatize process that you have to do several times. It helps you with the big quantity of data transformation steps that need to be executed in the right order.

[80]:
```
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)
```

[81]:
```
from sklearn.compose import ColumnTransformer

num_attribs = list(housing_num)
cat_attribs = ["ocean_proximity"]
k = 5

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

housing_prepared = full_pipeline.fit_transform(housing)
```

1.5 Select and Train a Model

1.5.1 Training and Evaluating on the Training Set

[82]:
```
from sklearn.linear_model import LinearRegression

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

[82]:
```
LinearRegression()
```

[83]:
```
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))
print("Labels:", list(some_labels))
```

```
Predictions: [210644.60459286 317768.80697211 210956.43331178  59218.98886849
 189747.55849879]
Labels: [286600.0, 340600.0, 196900.0, 46300.0, 254500.0]
```

[84]:
```
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
```

[84]:
```
68628.19819848923
```

[85]:
```
from sklearn.tree import DecisionTreeRegressor

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

[85]:
```
DecisionTreeRegressor()
```

[86]:
```
housing_predictions = tree_reg.predict(housing_prepared)
tree_mse = mean_squared_error(housing_labels, housing_predictions)
tree_rmse = np.sqrt(tree_mse)
tree_rmse
```

[86]:
```
0.0
```

1.5.2 Better Evaluation Using Cross-Validation

[87]:
```python
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)
```

[88]:
```python
def display_scores(scores):
    print("Scores:", scores)
    print("Mean:", scores.mean())
    print("Standard deviation:", scores.std())
```

[89]:
```python
display_scores(tree_rmse_scores)
```

```
Scores: [68431.19771345 66907.44518354 71861.17057648 69034.64312986
 71323.17211961 74842.14001017 71188.98601197 71366.8581941
 76720.70752396 71078.76539397]
Mean: 71275.50858571235
Standard deviation: 2737.9266239227704
```

[90]:
```python
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: [66782.73843989 66960.118071   70347.95244419 74739.57052552
 68031.13388938 71193.84183426 64969.63056405 68281.61137997
 71552.91566558 67665.10082067]
Mean: 69052.46136345083
Standard deviation: 2731.674001798342
```

[91]:
```python
from sklearn.ensemble import RandomForestRegressor
forest_reg = RandomForestRegressor()

forest_reg.fit(housing_prepared, housing_labels)

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: [49773.16229665 47458.65166157 50024.1330189  52278.3796685
 49816.25036768 53374.69809531 49104.10031073 48235.38836373
 52937.43739583 50259.9822007 ]
Mean: 50326.21833796121
```

```
Standard deviation: 1862.0496727576528
```

1.6 Fine-tune the model

[92]:
```python
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]},
 ↪]

forest_reg = RandomForestRegressor()

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)

print(grid_search.best_params_)
print(grid_search.best_estimator_)
```

```
{'max_features': 6, 'n_estimators': 30}
RandomForestRegressor(max_features=6, n_estimators=30)
```

[93]:
```python
feature_importances = grid_search.best_estimator_.feature_importances_
feature_importances
```

[93]:
```
array([7.52664665e-02, 7.14269143e-02, 4.17259173e-02, 1.78678192e-02,
       1.65241898e-02, 1.78175813e-02, 1.63545241e-02, 3.50061931e-01,
       5.83747194e-02, 1.09911345e-01, 5.79385128e-02, 1.39132971e-02,
       1.39906930e-01, 7.34590470e-05, 4.96050190e-03, 7.87589193e-03])
```

[94]:
```python
extra_attribs = ["rooms_per_hhold", "pop_per_hhold", "bedrooms_per_room"]
#cat_encoder = cat_pipeline.named_steps["cat_encoder"] # old solution
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)
```

[94]:
```
[(0.3500619309201084, 'median_income'),
 (0.13990692978211006, 'INLAND'),
 (0.10991134463983283, 'pop_per_hhold'),
 (0.07526646651941024, 'longitude'),
 (0.07142691426837043, 'latitude'),
 (0.05837471939441458, 'rooms_per_hhold'),
 (0.057938512835160313, 'bedrooms_per_room'),
 (0.041725917297971406, 'housing_median_age'),
 (0.017867819244104657, 'total_rooms'),
 (0.017817581250000835, 'population'),
```

```
(0.016524189814007884, 'total_bedrooms'),
(0.016354524105828543, 'households'),
(0.01391329705772381, '<1H OCEAN'),
(0.007875891925881257, 'NEAR OCEAN'),
(0.0049605018980332545, 'NEAR BAY'),
(7.345904704156673e-05, 'ISLAND')]
```

[95]:
```
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
```

[95]:
```
47576.66221096059
```

[96]:
```
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)))
```

[96]:
```
array([45632.17930057, 49444.73466965])
```

1.7 Exercises

Exercise 1: Try a Support Vector Machine regressor (sklearn.svm.SVR) with various hyper-parameters, such as kernel="linear" (with various values for the C hyperpara-meter) or kernel="rbf" (with various values for the C and gammahyperparameters).

[107]:
```
from sklearn.svm import SVR

param_grid = [{"kernel": ["linear"], "C": [1.0, 2.0, 0.35, 0.45]} ,{"kernel":␣
 ↪["rbf"], "C": [1.5, 2.5, 0.5], "gamma": ["scale", "auto"]}]
vector_reg = SVR()

grid_search = GridSearchCV(vector_reg, param_grid, cv=5,␣
 ↪scoring='neg_mean_squared_error',return_train_score=True)
grid_search.fit(housing_prepared, housing_labels)

print(grid_search.best_params_)
```

```
print(grid_search.best_estimator_)

final_model = grid_search.best_estimator_
```

```
{'C': 2.0, 'kernel': 'linear'}
SVR(C=2.0, kernel='linear')
```

Exercise 2: Try replacing GridSearchCV with RandomizedSearchCV

[105]:
```
#Exercise 2
from sklearn.model_selection import RandomizedSearchCV

random_search = RandomizedSearchCV(vector_reg, param_grid,n_iter =10, cv=5)

random_search.fit(housing_prepared, housing_labels)
print(random_search.best_params_)
print(random_search.best_estimator_)
```

```
{'kernel': 'linear', 'C': 2.0}
SVR(C=2.0, kernel='linear')
```

Exercise 3: Try adding a transformer in the preparation pipeline to select only the most important attributes.

[108]:
```
#Exercise 3
def indices_n_important(arr, n):
    return np.sort(np.argpartition(np.array(arr), -k)[-k:])

class ImportantAttributes(BaseEstimator, TransformerMixin):
    def __init__(self, importances, n):
        self.importances = importances
        self.n = n

    def fit(self, X, y = None):
        self.attribute_indices_ = indices_n_important(self.importances, n)

        return self

    def transform(self, X):
        return X[:, self.attribute_indices_]
```

[101]:
```
n = 5

complete_pipeline = Pipeline([
    ("full", full_pipeline),
    ("best_features", ImportantAttributes(feature_importances, n))
])
```

```python
housing_prepared = complete_pipeline.fit_transform(housing)
```

Exercise 4: Try creating a single pipeline that does the full data preparation plus the final prediction.

[111]:
```python
final_pipeline = Pipeline([
    ("complete_pipeline", complete_pipeline),
    ("svm_reg", SVR(**random_search.best_params_))
])
```

[115]:
```python
final_pipeline.fit(housing, housing_labels)
```

[115]:
```
Pipeline(steps=[('complete_pipeline',
                 Pipeline(steps=[('full',
                                  ColumnTransformer(transformers=[('num',
Pipeline(steps=[('imputer',
    SimpleImputer(strategy='median')),
   ('attribs_adder',
    CombinedAttributesAdder()),
   ('std_scaler',
    StandardScaler())]),
                                                                   ['longitude',
                                                                    'latitude',
'housing_median_age',
'total_rooms',
'total_bedrooms',
'population',
'households',
                                                                    'median_i…
['ocean_proximity'])])),
                                 ('best_features',
ImportantAttributes(importances=array([7.52664665e-02, 7.14269143e-02,
4.17259173e-02, 1.78678192e-02,
       1.65241898e-02, 1.78175813e-02, 1.63545241e-02, 3.50061931e-01,
       5.83747194e-02, 1.09911345e-01, 5.79385128e-02, 1.39132971e-02,
       1.39906930e-01, 7.34590470e-05, 4.96050190e-03, 7.87589193e-03]),
                                                       n=5))])),
                ('svm_reg', SVR(C=2.0, kernel='linear'))])
```

[ ]: