HW-4

Name: GOVIND AJITH KUMAR

UID: 116699488

Date: 12/8/2020

① Problem 3.6 LFD

Ans To do: derive a linear programming algorithm to fit a linear problem for classification. We know that a linear problem is an optimization problem of the following form:

$$\min_{z} \; c^T z$$
$$\text{subject to } Az \leq b$$

Here:
$A, b, c \rightarrow$ parameters of the linear programme
$z \rightarrow$ optimization variable

@ to prove: for a linearly separable data, show that for some w, $y_n(w^T x_n) \geq 1$ for $n = 1, 2, \ldots N$.

For linearly separable data, we know:

$$y_n (w^*)^T x_n > 0 \quad \text{for } n = 1, 2 \ldots N$$

We also know that

$$y_n (w^*)^T x_n \geq \epsilon \text{ and } \epsilon > 0$$

If we assume $w^* = w\epsilon$

$$\Rightarrow y_n (w\epsilon)^T x_n \geq \epsilon$$

$$\Rightarrow y_n (w^T) x_n \geq 1, \quad \underline{\text{Hence proved.}}$$

(b) To find a separating w for a linear program, we can write the linear programme as

$$\begin{cases} \min_w & c^T w \\ \text{subject to} & Aw \leq b \end{cases}$$

Here:

$$A = -\begin{pmatrix} — & y_1 x_1^T & — \\ — & y_2 x_2^T & — \\ \vdots & \vdots & \vdots \\ — & y_N x_1^T & — \end{pmatrix}, \quad c^T = (0, \ldots 0),$$

$$b^T = (-1, \ldots -1)$$

(c) Deriving from the previous problem, we can formulate the minimization problem in linear fashion as follows:

$$\begin{cases} \min(w,\xi) & c^T(w,\xi)^T \\ & A(w,\xi)^T \leq b \end{cases}$$

Here: $c^T = (0, \ldots, 0, 1, \ldots, 1)$, and

$b^T = (-1, \ldots, -1, 0, \ldots, 0)$, and

$$A = \left(\begin{array}{c|c} -y_1 x_1^T - & 1 \cdots 1 \\ \vdots & \\ -y_n x_n^T - & \\ \hline 0 \cdots 0 & 1 \cdots 1 \end{array}\right)$$

(d) Problem 3.5 was tasked with minimizing $\frac{1}{N}\sum_{n=1}^{N} e_n(w)$. As per the problem: $e_n(w) = \begin{cases} 0 & \text{if } y_n w^T x_n > 1 \\ 1 - y_n w^T x_n & \text{if } y_n w^T x_n < 1 \end{cases}$

We have to make a case for the above equation and the derived answer in the previous subsection (c) to be

equivalent. We can see that atleast at one of the margin of the linear seperators $y_n w^T x_n \geq 1$, x_n is correctly classified. At this particular stage $e_n(w) = 0$, which means that x_n, does not contribute to the error. But if x_n is on the other side then $e_n(w) = (-y_n w^T x_n)$, which clearly contributes to the error. From these 2 instances we can see that the violation on the margin is very crucial indetermining the contribution to the error. Therefore we can draw similarities between 3.5 and above to determine that they are equivalent.

Reference: LFD text book, internet

④ Problem 8.2

Ans

$$X = \begin{bmatrix} 0 & 0 \\ 0 & -1 \\ -2 & 0 \end{bmatrix}, \quad y = \begin{bmatrix} -1 \\ -1 \\ +1 \end{bmatrix}$$

We have to manually solve to get the optimal hyperplane (b^*, w^*) and its margin.

Calculating the constraints:

$$-(0 \cdot w_1 + 0 \cdot w_2 + b) \geq 1$$

$$\Rightarrow -b \geq 1 \quad \text{——— (a)}$$

$$-(0 \cdot w_1 - 1 \cdot w_2 + b) \geq 1$$

$$\Rightarrow -(-w_2 + b) \geq 1 \quad \text{——— (b)}$$

and,

$$-2 \cdot w_1 + 0 \cdot w_2 + b \geq 1$$

$$\Rightarrow -2w_1 + b \geq 1 \quad \text{——— (c)}$$

Now, we have the minimize: $\frac{1}{2} w^T w$

Now, we know that $w^T w =$

$$w_1^2 + w_2^2$$

$\therefore \frac{1}{2} w^T w = \frac{1}{2}(w_1^2 + w_2^2)$

Now, from (a) and (c):

$-w_1 \geq 1 \Rightarrow w_1 \leq -1$

So: $\frac{1}{2}((+0) \geq \frac{1}{2}$ when $w_1 = -1$, $w_2 = 0$.

$\therefore w^* = (-1, 0)$ — (d)

Using this we are able to find the constraint (c) that:

$-2 w_1^* + b \geq 1$

from (d):

$1 \leq 2 + b = -2 w_1^* + b$

$\Rightarrow b + 2 \geq 1$

$\Rightarrow \boxed{b \geq -1}$

$\therefore b^* = -1$. Hence (w^*, b^*) satisfies our constraints and also minimizes $\|w\|$. Hence, this is our optimal hyperplane

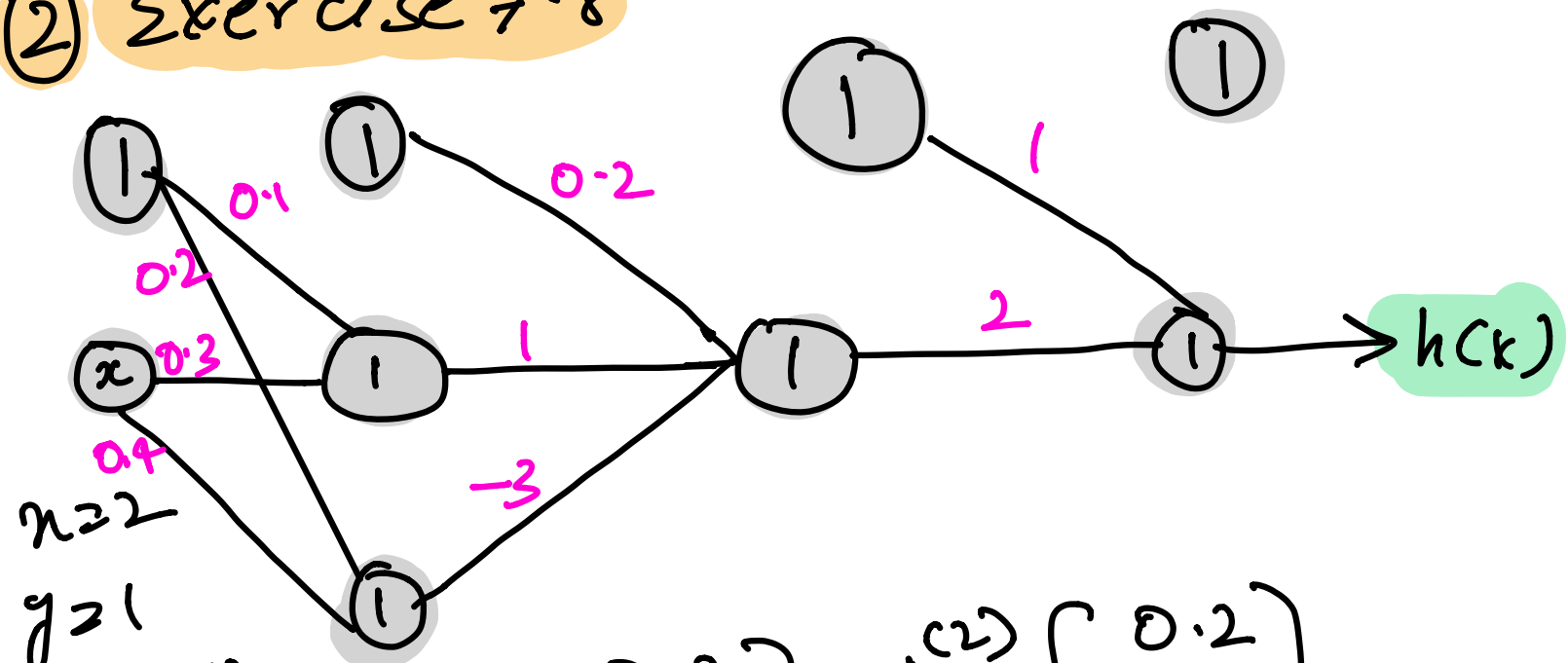
Finding margin:

$$\frac{1}{\|w^*\|} = \frac{1}{1} = 1.$$ This is the margin.

Reference: Page 7-8, chapter 8, LFD. Class Lecture.

---

(2) Exercise 7.8



$n = 2$

$y = 1$

$$W^{(1)} = \begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{bmatrix}, \quad W^{(2)} = \begin{bmatrix} 0.2 \\ 1 \\ -3 \end{bmatrix},$$

and $W^{(2)} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$

Now, $s^{(l)} = (W^{(l)})^T x^{(l-1)}$.

Hence: $s^{(1)} = (W^{(1)})^T x^{(0)}$,

$s^{(2)} = (W^{(2)})^T x^{(1)}$,

and $s^{(3)} = (W^{(3)})^T x^{(2)}$

Now,

$x^{(0)} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$

$s^{(1)} = \begin{bmatrix} 0.1 & 0.3 \\ 0.2 & 0.4 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 0.7 \\ 1 \end{bmatrix}$

$x^{(1)} = \begin{bmatrix} 1 \\ 0.7 \\ 1 \end{bmatrix}$

$$s^{(2)} = \begin{bmatrix} 1 & 0.7 & 1 \end{bmatrix} \begin{bmatrix} 0.2 \\ 1 \\ -3 \end{bmatrix}$$

$$= \begin{bmatrix} 0.2 + 0.7 - 3 \end{bmatrix}$$

$$= \begin{bmatrix} -2.1 \end{bmatrix}$$

$$x^{(2)} = \begin{bmatrix} 1 \\ -2.1 \end{bmatrix}$$

$$s^{(3)} = \begin{bmatrix} 1 & -2.1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} -3.2 \end{bmatrix}$$

$$x^{(3)} = \underline{\underline{-3.2}}$$

Now, calculating the back propogation:

for identity $\theta'(z^{(l)}) = 1$

Now doing back propagation $\longrightarrow$

$$\delta^{(3)} = 2(x^{(l)} - y)(\theta'(z^{(l)}))$$

$$\delta^{(3)} = 2(-3.2 - 1)(1 - (-3.2)^2)$$

$$\delta^{(3)} = 2(-4.2)(-9.24)$$

$$\delta^{(3)} = 77.616$$

$$\delta^{(2)} = (1 - (-2.1)^2) \cdot 2 \cdot (77.616)$$

$$= -3.41 \cdot 2 \cdot (77.616)$$

$$= -529.34$$

$$\delta^{(1)} = \begin{bmatrix} 1 - (0.7)^2 & \cdot & 1 & \cdot & -529.34 \\ 1 - 1)^2 & & \cdot -3 & \cdot & -529.34 \end{bmatrix}$$

$$= \begin{bmatrix} -269.96 \\ 0 \end{bmatrix}$$

$$\frac{\partial e}{\partial W^{(1)}} = X^{(0)} (\delta^{(1)})^T = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \begin{bmatrix} -269.96 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} -269.96 & 0 \\ -539.9 & 0 \end{bmatrix}$$

$$\frac{\partial e}{\partial W^{(2)}} = X^{(1)} (\delta^{(2)})^T = \begin{bmatrix} 1 \\ 0.7 \\ 1 \end{bmatrix} \begin{bmatrix} -529.34 \end{bmatrix}$$

$$= \begin{bmatrix} -529.34 \\ -370.538 \\ -529.34 \end{bmatrix}$$

$$\frac{\partial e}{\partial W^{(3)}} = X^{(2)} (\delta^{(3)})^T = \begin{bmatrix} 1 \\ -2.1 \end{bmatrix} \begin{bmatrix} 77.616 \end{bmatrix}$$

$$= \begin{bmatrix} 77.616 \\ -162.993 \end{bmatrix}$$

Exercise_8_6

November 25, 2020

1 Exercise 8.6

2 Importing all the necessary libraries

[1]:
```
import matplotlib.pyplot as plt
from multiprocessing import Process
import seaborn as sns
import numpy as np
from cvxopt import matrix, solvers
import random
import warnings
warnings.filterwarnings("ignore")
```

3 Generate the training samples

[2]:
```
def generateRandomPoints(N=20):
    x = np.zeros((N, 2), dtype=np.float)
    x[:, 0] = np.random.rand(N)
    x[:, 1] = 2*np.random.rand(N)- 1
    y = np.sign(x[:, 1])
    return x, y
```

4 Calculate the error

[3]:
```
def calculateErrorout(w_hat, n_samples=5000):
    x, y = generateRandomPoints(n_samples)
    N = x.shape[0]
    concatenated_x = np.concatenate((np.ones((N, 1)), x), axis=1)
    prediction = np.matmul(concatenated_x, w_hat).reshape(-1)
    prediction = np.sign(prediction) - (prediction == 0)
    error_out = 1 - (np.count_nonzero(prediction == y) / N)
    return error_out
```

5 Declaring SVM

[4]:
```
class SVM():
    def __init__(self):
        self.sp_v_threshold = 0.0000001
    def fit_data(self, x, y):
        support_vector_indices = []
        N = len(x)
        DIMENSION = len(x[0])
        P = matrix(np.identity(DIMENSION + 1))
        P[0, 0] = 0
        G = matrix(np.zeros((N, DIMENSION + 1)))
        for i in range(N):
            G[i, 0] = -y[i]
            G[i, 1:] = -x[i, :] * y[i]
        q = matrix(np.zeros((DIMENSION + 1,)))
        h = -matrix(np.ones((N,)))
        solv = solvers.qp(P, q, G, h)
        weights= np.zeros(DIMENSION,)
        b = solv["x"][0]
        for i in range(1, DIMENSION + 1):
            weights[i - 1] = solv["x"][i]
        for k in range(N):
            val = y[k] * (np.dot(weights, x[k]) + b)
            if val < (1 + self.sp_v_threshold):
                support_vector_indices.append(k)
        self.intercept = np.array([b])
        self.coefficient = weights.reshape(-1, 1)
def returnConcatenatedSVM(X, y):
    svm = SVM()
    svm.fit_data(X, y)
    concatenated_svm = np.concatenate((svm.intercept.reshape(1,1), svm.
 ↪coefficient.reshape(-1, 1)),0)
    return concatenated_svm
```

6 Declaring PLA

[5]:
```
def PLA(x, y):
    N = x.shape[0]
    concatenated_x = np.concatenate((np.ones((N, 1)), x), axis=1)
    w = np.zeros((concatenated_x.shape[1], 1))
    maximumupdates=1500
    for updates in range(maximumupdates):
        prediction = np.matmul(concatenated_x, w)
```

```
        prediction = np.sign(prediction) - (prediction == 0)
        prediction = prediction.reshape(-1)
        for i, j in enumerate(prediction != y):
            if j == 1:
                break
            if i == x.shape[0] - 1:
                return w
        w += y[i] * concatenated_x[i, :].reshape(-1, 1)
    return w
```

7 Comparing the results

[6]:
```
def compareSVMandPLA():
    x, y = generateRandomPoints()
    SVM_error_out_of_sample = calculateErrorout(returnConcatenatedSVM(x, y))
    PLA_error_out_of_sample = []
    iterations = 5000
    for iters in range(iterations):
        i = np.random.permutation(x.shape[0])
        x = x[i]
        y = y[i]
        PLA_error_out_of_sample.append(calculateErrorout(PLA(x, y)))
    PLA_error_out_of_sample = np.array(PLA_error_out_of_sample)
    return SVM_error_out_of_sample, PLA_error_out_of_sample
```

8 Generating samples and running everything

[7]:
```
x, y = generateRandomPoints()
w_hat_svm = returnConcatenatedSVM(x, y)
w_hat_pla = PLA(x, y)
```

```
     pcost       dcost       gap    pres   dres
 0:  7.9613e-01  1.0459e+01  5e+01  2e+00  2e+01
 1:  5.0087e+00 -2.1621e-01  2e+01  7e-01  6e+00
 2:  8.0068e+00  5.4934e+00  8e+00  2e-01  2e+00
 3:  1.1625e+01  9.3348e+00  3e+00  5e-02  4e-01
 4:  1.2221e+01  1.2126e+01  1e-01  7e-04  6e-03
 5:  1.2193e+01  1.2192e+01  1e-03  7e-06  6e-05
 6:  1.2192e+01  1.2192e+01  1e-05  7e-08  6e-07
 7:  1.2192e+01  1.2192e+01  1e-07  7e-10  6e-09
Optimal solution found.
```

9 Calling the comparison function

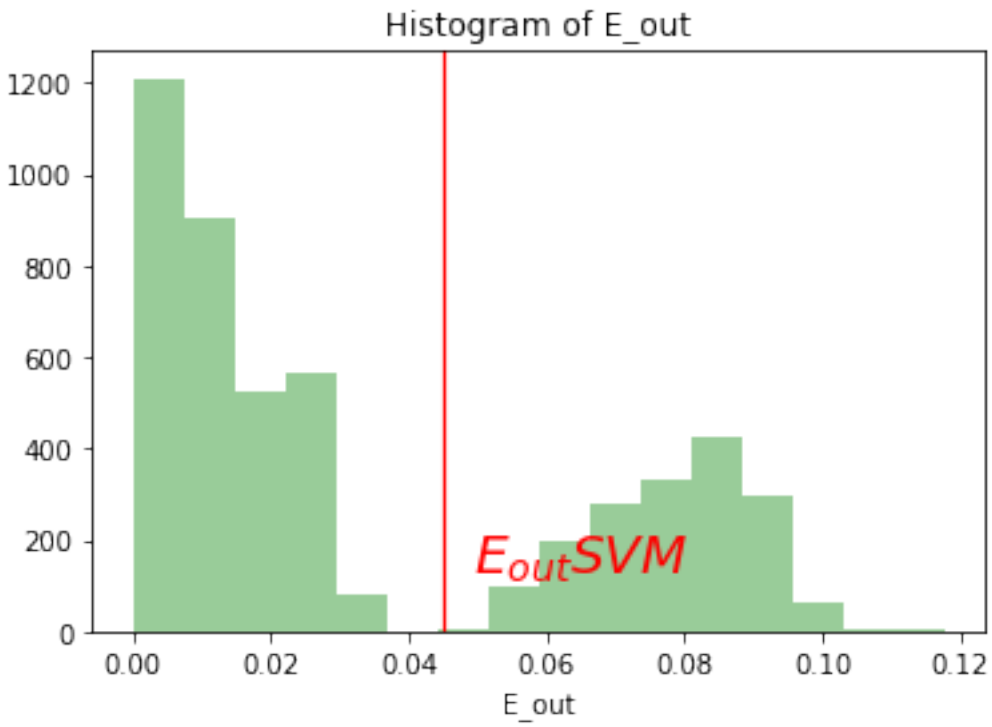
[8]:
```
E_out_SVM, E_outs_PLA = compareSVMandPLA()
```

```
     pcost       dcost       gap    pres   dres
 0:  1.0999e+00  1.5064e+01  5e+01  2e+00  1e+01
 1:  5.6889e+00  1.2563e+01  2e+01  8e-01  4e+00
 2:  1.7337e+01  5.1226e+01  2e+01  6e-01  3e+00
 3:  4.9645e+01  6.1561e+01  1e+01  2e-01  1e+00
 4:  6.5607e+01  6.5910e+01  5e-01  6e-03  3e-02
 5:  6.6064e+01  6.6067e+01  5e-03  6e-05  3e-04
 6:  6.6068e+01  6.6069e+01  5e-05  6e-07  3e-06
 7:  6.6069e+01  6.6069e+01  5e-07  6e-09  3e-08
Optimal solution found.
```

10 Plotting all the data

[9]:
```
axes = sns.distplot(E_outs_PLA, kde=False, color= 'g')
plt.axvline(E_out_SVM, color="red", lw=1.2)
plt.title('Histogram of E_out')
plt.xlabel('E_out')
plt.text(E_out_SVM + (axes.axis()[1] - axes.axis()[0]) / 30, 0.1 * axes.
 ↪axis()[3], r'$E_{out}SVM$',fontsize=20, color = 'r')
```

[9]:
```
Text(0.049319333333333375, 126.94500000000001, '$E_{out}SVM$')
```


Histogram of E_out
1200
1000
800
600
400
200
0
0.00
0.02
0.04
0.06
0.08
0.10
0.12
E_out
$E_{out}SVM$

[ ]:

Q8_13_HW4

November 26, 2020

1 Problem 8.13

2 All necessary imports

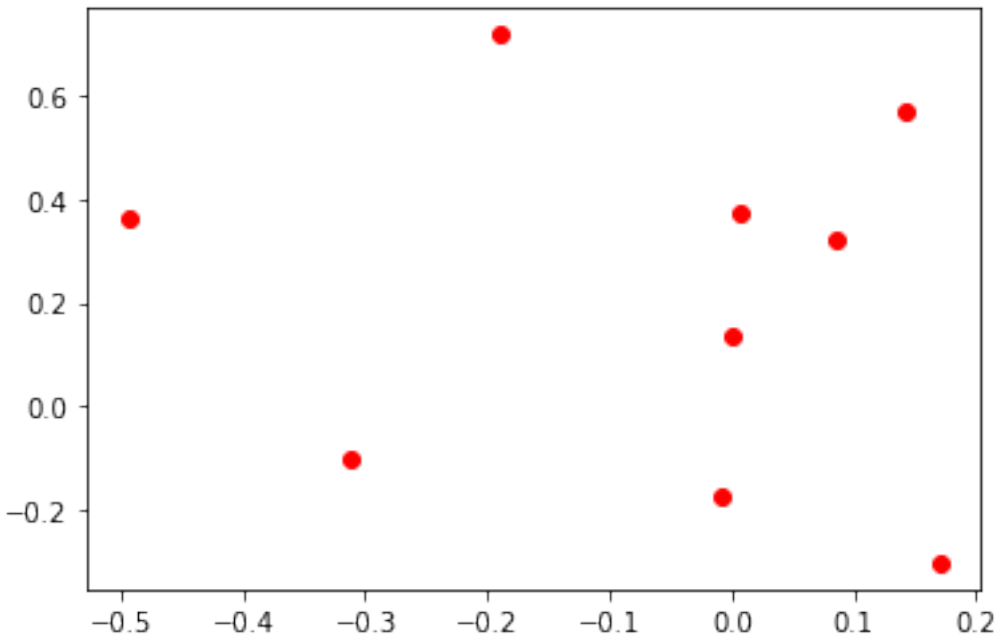
[1]:
```
import numpy as np
from numpy.linalg import inv
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

3 Question 8.13(a)

[2]:
```
x_coords_1 = [-0.494, -0.311, 0.0064, -0.0089, 0.0014, -0.189,0.085, 0.171, 0.
↪142]
y_coords_1 = [0.363, -0.101, 0.374, -0.173, 0.138, 0.718, 0.32208, -0.302, 0.
↪568]
```

[3]:
```
plt.scatter(x_coords_1,y_coords_1, color = 'red', marker = 'o')
```

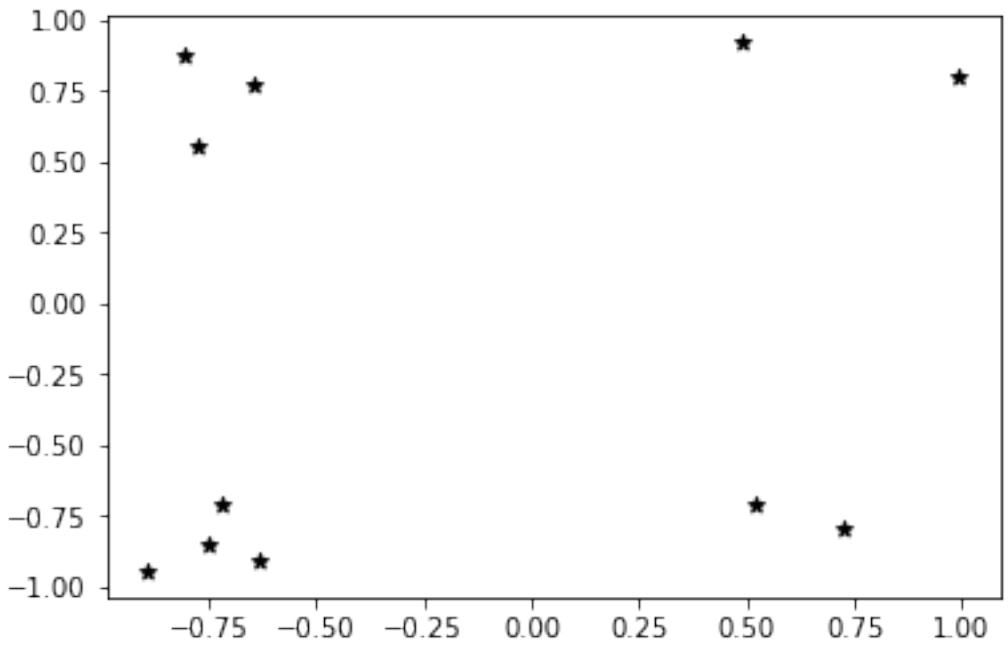
[3]:
```
<matplotlib.collections.PathCollection at 0x7f66a7396ca0>
```



[4]:
```
x_coords_2 = [0.491, -0.892, -0.721, 0.519, -0.775, -0.646,-0.803, 0.994, 0.
↪724, -0.748,-0.635]
y_coords_2= [0.920, -0.946, -0.710, -0.715, 0.551, 0.773, 0.875, 0.801, -0.
↪795,-0.853, -0.905]
```

[5]:
```
plt.scatter(x_coords_2,y_coords_2,color='k', marker = '*')
```

[5]:
```
<matplotlib.collections.PathCollection at 0x7f66a528ffd0>
```

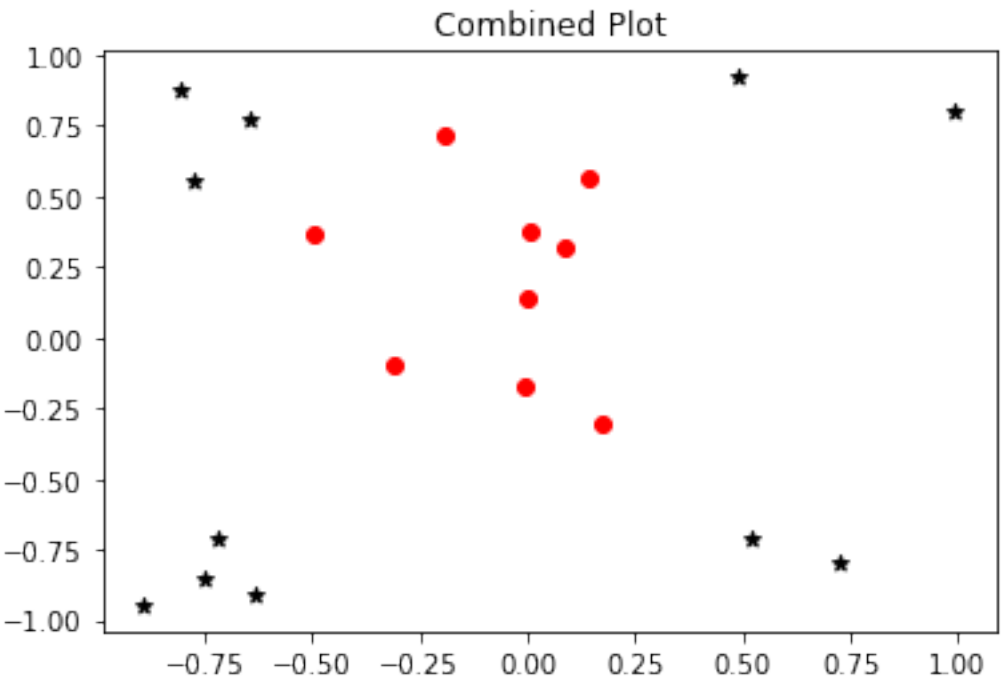


[6]:
```
all_datapoints = np.array([[1,-0.494,0.363, 1], [1,-0.311,-0.101,1], [1,0.
↪0064,0.374,1], [1,-0.0089,-0.173,1],
                [1,0.0014,0.138,1], [1,-0.189,0.718,1], [1,0.085,0.32208,1],␣
↪[1,0.171,-0.302,1],
                [1,0.142,0.568,1], [1,0.491,  0.920,-1], [1,-0.892, -0.946,-1],␣
↪[1,-0.721, -0.710,-1],
                [1,0.519, -0.715,-1], [1,-0.775, 0.551,-1],[1,-0.646,0.773,-1],␣
↪[1,-0.803, 0.875,-1],
                [1,0.994, 0.801,-1], [1,0.724,-0.795,-1],[1,-0.748,-0.
↪853-1],[1,-0.635,-0.905,-1]])
```

4 Combined plotting of all the datapoints for visualization purposes

[7]:
```
plt.title('Combined Plot')
plt.scatter(x_coords_2,y_coords_2,color='k', marker = '*')
plt.scatter(x_coords_1,y_coords_1, color = 'red', marker = 'o')
```

[7]:
```
<matplotlib.collections.PathCollection at 0x7f66a520e460>
```



5 Doing feature Transformation in X space

[8]:
```
x_w1 = []
for i in range(len(all_datapoints)):
    x_w1.append(all_datapoints[i][0:3])
x_w1 = np.array(x_w1,dtype = float)
y_w1_l = [1,1,1,1,1,1,1,1,1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1,-1]
```

[9]:
```
W_1 = inv(x_w1.T.dot(x_w1)).dot(x_w1.T).dot(y_w1_l)
```

[10]:
```
N = 2000
x = np.linspace(-1, 1, N)
y = np.linspace(-1, 1, N)
X_MESHGRID, Y_MESHGRID = np.meshgrid(x, y)
```

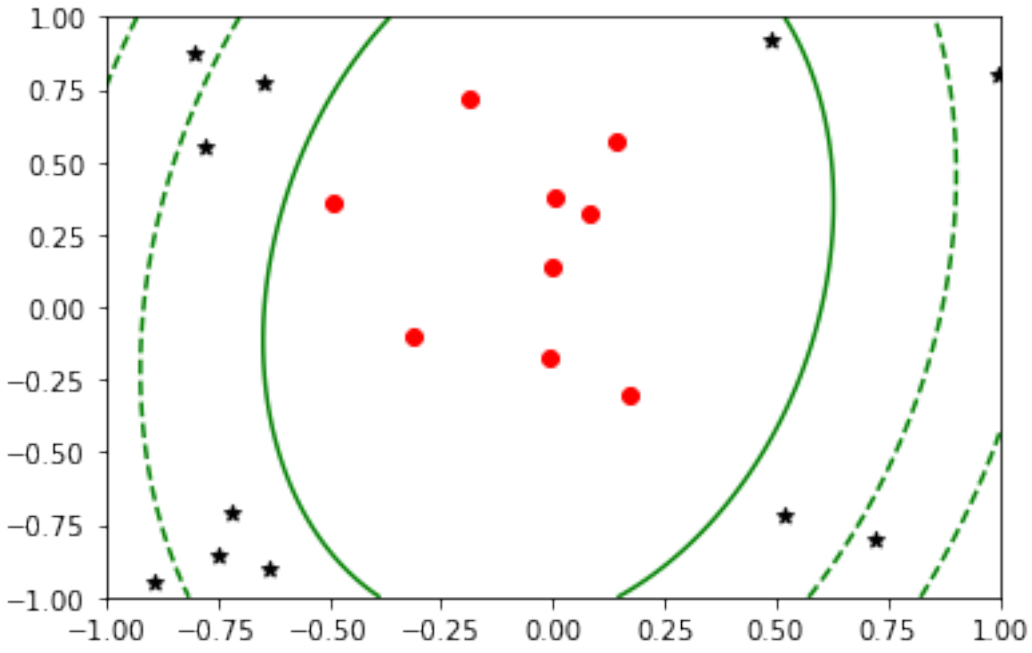
[11]:
```
def feature_transform_2(datapoints):
    result = []
    for i in datapoints:
        x_1 = i[1]
        to_append = i[-1]
        x_2 = i[2]
        x = [1, x_1, x_2, x_1**2, x_1*x_2, x_2**2, to_append]
```

```
            result.append(x)
        result_array = np.array(result)
        return result_array

def contour_func_2(x,y,w):
    return w[0] + w[1]*x + w[2]*y + w[3]*x**2 + w[4]*(x*y) + w[5]*(y**2)
```

[12]:
```
transformed = feature_transform_2(all_datapoints)
x_transformed = transformed[:,:-1]
y_transformed = transformed[:,-1]
W_2 = inv(x_transformed.T.dot(x_transformed)).dot(x_transformed.T).
 ↪dot(y_transformed)
```

[13]:
```
plt.scatter(x_coords_1,y_coords_1, color = 'r', marker = 'o')
plt.scatter(x_coords_2,y_coords_2,color='k', marker = '*')
plt.contour(X_MESHGRID, Y_MESHGRID, contour_func_2(X_MESHGRID, Y_MESHGRID,W_2),␣
 ↪3, colors = 'g')
```

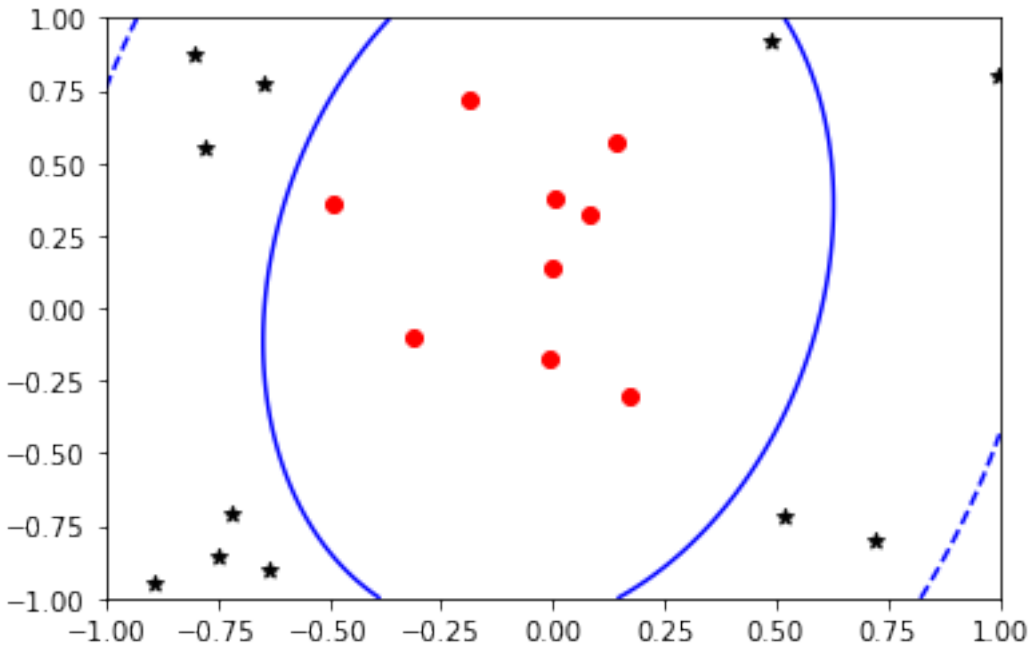
[13]: <matplotlib.contour.QuadContourSet at 0x7f66a1474460>



6 Plotting just the main contour

[14]:
```
plt.scatter(x_coords_1,y_coords_1, color = 'r', marker = 'o')
plt.scatter(x_coords_2,y_coords_2,color='k', marker = '*')
plt.contour(X_MESHGRID, Y_MESHGRID, contour_func_2(X_MESHGRID, Y_MESHGRID,W_2),␣
↪1, colors = 'b')
```

[14]:
```
<matplotlib.contour.QuadContourSet at 0x7f669f3c28e0>
```



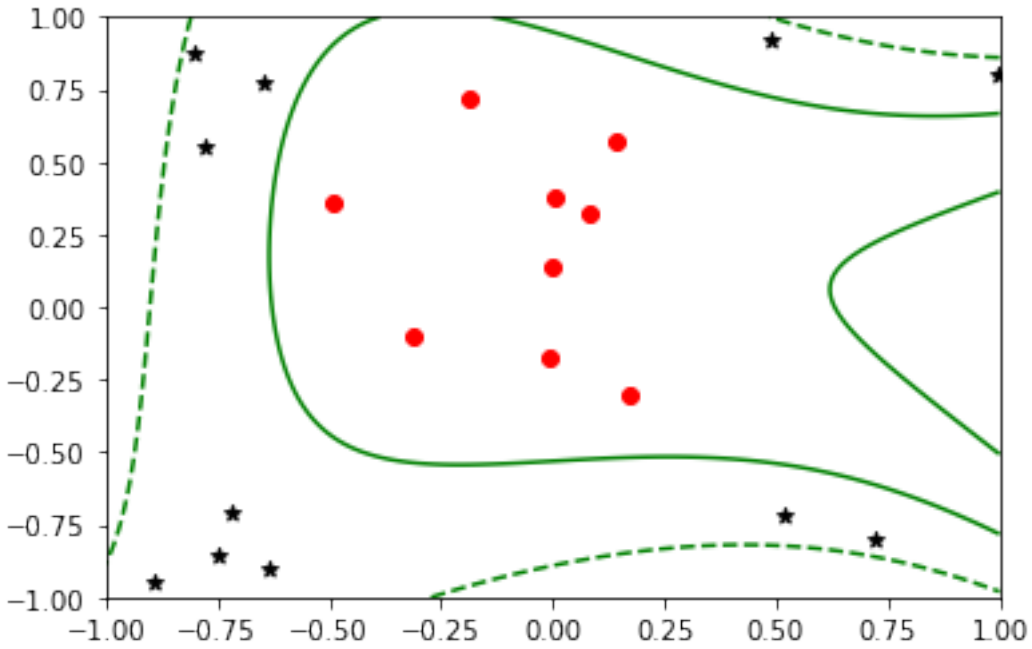
[15]:
```
def feature_transformation_3(datapoints):
    result = []
    for i in datapoints:
        x_1 = i[1]
        to_append = i[-1]
        x_2 = i[2]
        x = [1, x_1, x_2, x_1 * x_2, x_1**2, x_2**2, x_1**3, (x_1**2)*x_2,␣
↪x_1*(x_2**2), x_2**3, to_append]
        result.append(x)
    result_array = np.array(result)
    return result_array

def contour_func_3(x,y,w):
    return w[0] + w[1]*x + w[2]*y + w[3]*x*y + w[4]*(x**2) + w[5]*(y**2) +␣
↪w[6]*(x**3) + w[7]*(x**2)*y + w[8]*x*(y**2) + w[9]*y**3
```

[16]:
```
transformed = feature_transformation_3(all_datapoints)
x_transformed = transformed[:,:-1]
y_transformed = transformed[:,-1]
```

[17]:
```
W_3 = inv(x_transformed.T.dot(x_transformed)).dot(x_transformed.T).
↪dot(y_transformed)
```

[18]:
```
plt.scatter(x_coords_1,y_coords_1, color = 'r', marker = 'o')
plt.scatter(x_coords_2,y_coords_2,color='k', marker = '*')
plt.contour(X_MESHGRID, Y_MESHGRID, contour_func_3(X_MESHGRID, Y_MESHGRID,W_3),␣
↪3, colors = 'g')
```

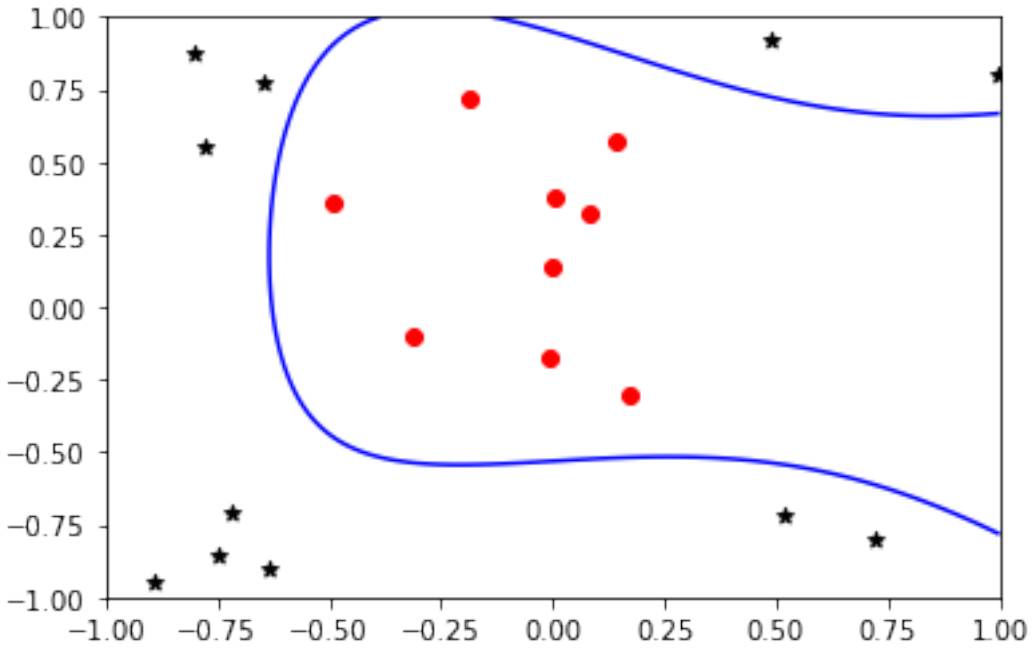
[18]: <matplotlib.contour.QuadContourSet at 0x7f669f331d30>



7 Plotting just the main contour

[19]:
```
plt.scatter(x_coords_1,y_coords_1, color = 'r', marker = 'o')
plt.scatter(x_coords_2,y_coords_2,color='k', marker = '*')
plt.contour(X_MESHGRID, Y_MESHGRID, contour_func_3(X_MESHGRID, Y_MESHGRID,W_3),␣
↪1, colors = 'b')
```

[19]: <matplotlib.contour.QuadContourSet at 0x7f669f2a8190>



8 Question 8.13(b)

Looking at the second and the third order transformations, the third order transformation seems to be overfitted.

9 Question 8.13(c)

10 Using Pseudo-Inverse Algorithm

11 Second order feature transformation

[20]:
```
def feature_transformation_pseudo_2(datapoints):
    result = []
    for i in datapoints:
        x_1 = i[1]
        to_append = i[-1]
        x_2 = i[2]
        x = [1, x_1, x_2, x_1**2, x_1*x_2, x_2**2, to_append]
        result.append(x)
    result_array = np.array(result)
    return result_array
```

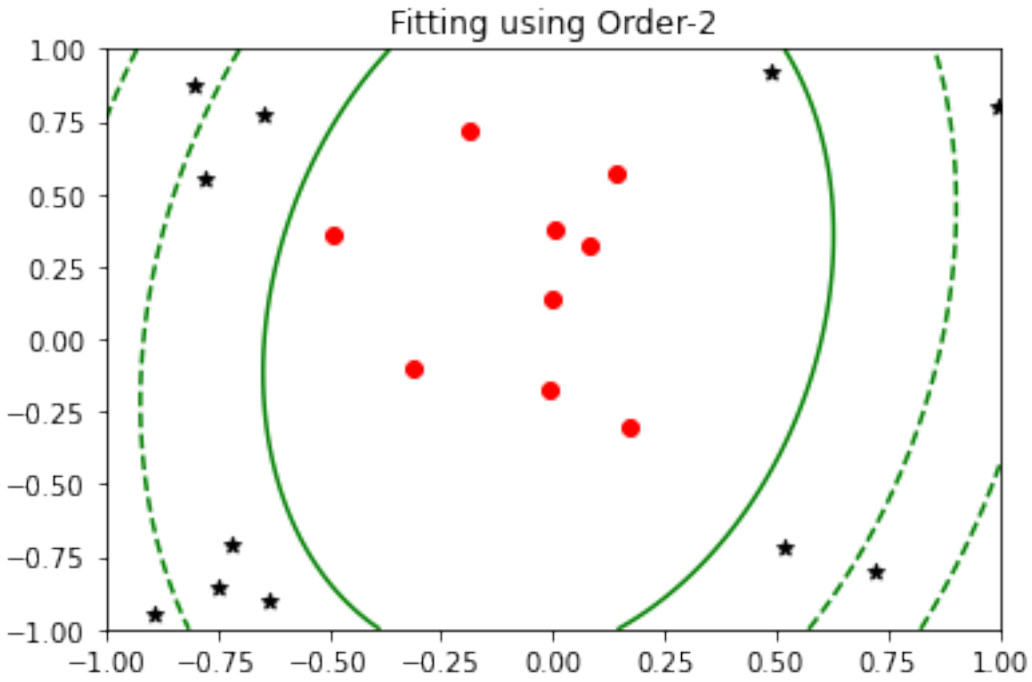
```
def contour_func_pseudo_2(x,y,w):
    return w[0] + w[1]*x + w[2]*y + w[3]*x**2 + w[4]*(x*y) + w[5]*(y**2)
```

[21]:
```
data_pseudo = feature_transformation_pseudo_2(all_datapoints)
x_transformed = data_pseudo[:,:-1]
y_transformed = data_pseudo[:,-1]
```

[22]:
```
regularization_lambda = 1
W_P = inv(x_transformed.T.dot(x_transformed) +␣
 ↪regularization_lambda*(x_transformed.T.dot(x_transformed))).
 ↪dot(x_transformed.T).dot(y_transformed)
```

[23]:
```
plt.scatter(x_coords_1,y_coords_1, color = 'r', marker = 'o')
plt.scatter(x_coords_2,y_coords_2,color='k', marker = '*')
plt.contour(X_MESHGRID, Y_MESHGRID, contour_func_pseudo_2(X_MESHGRID,␣
 ↪Y_MESHGRID,W_P), 3, colors = 'g')
plt.title('Fitting using Order-2')
```

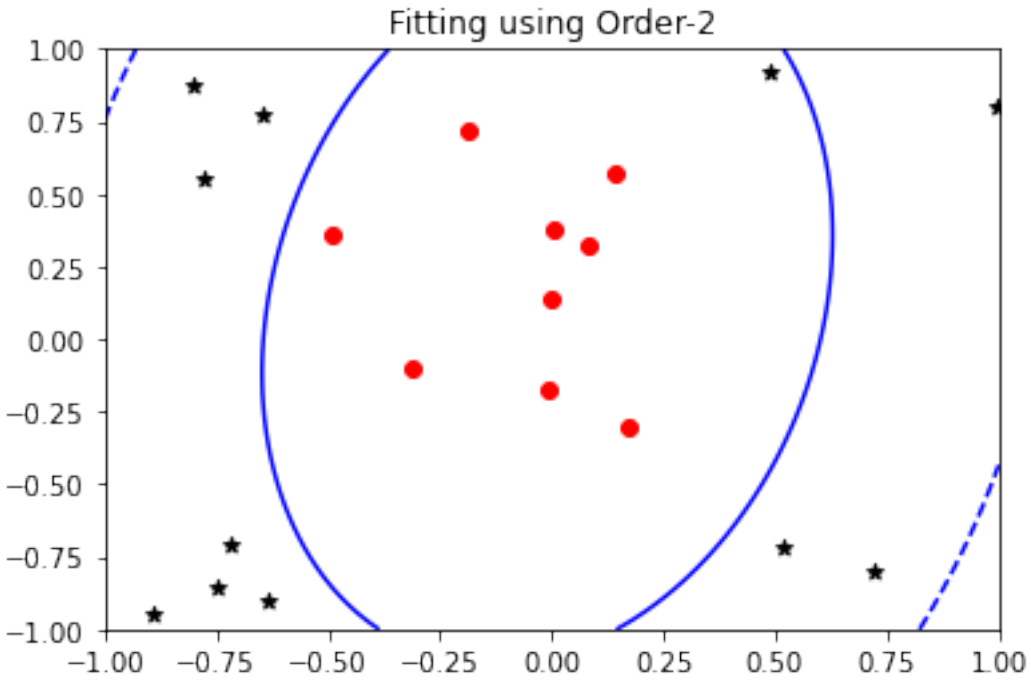
[23]:
```
Text(0.5, 1.0, 'Fitting using Order-2')
```



12 Plotting just the main contour

[24]:
```
plt.scatter(x_coords_1,y_coords_1, color = 'r', marker = 'o')
plt.scatter(x_coords_2,y_coords_2,color='k', marker = '*')
plt.contour(X_MESHGRID, Y_MESHGRID, contour_func_pseudo_2(X_MESHGRID,␣
 ↪Y_MESHGRID,W_P), 1, colors = 'b')
plt.title('Fitting using Order-2')
```

[24]:
```
Text(0.5, 1.0, 'Fitting using Order-2')
```



13 Third order feature transformation

[25]:
```
def feature_transformation_pseudo_3(datapoints):
    result = []
    for i in datapoints:
        x_1 = i[1]
        to_append = i[-1]
        x_2 = i[2]
        x = [1, x_1, x_2, x_1 * x_2, x_1**2, x_2**2, x_1**3, (x_1**2)*x_2,␣
 ↪x_1*(x_2**2), x_2**3, to_append]
        result.append(x)
    result_array = np.array(result)
```

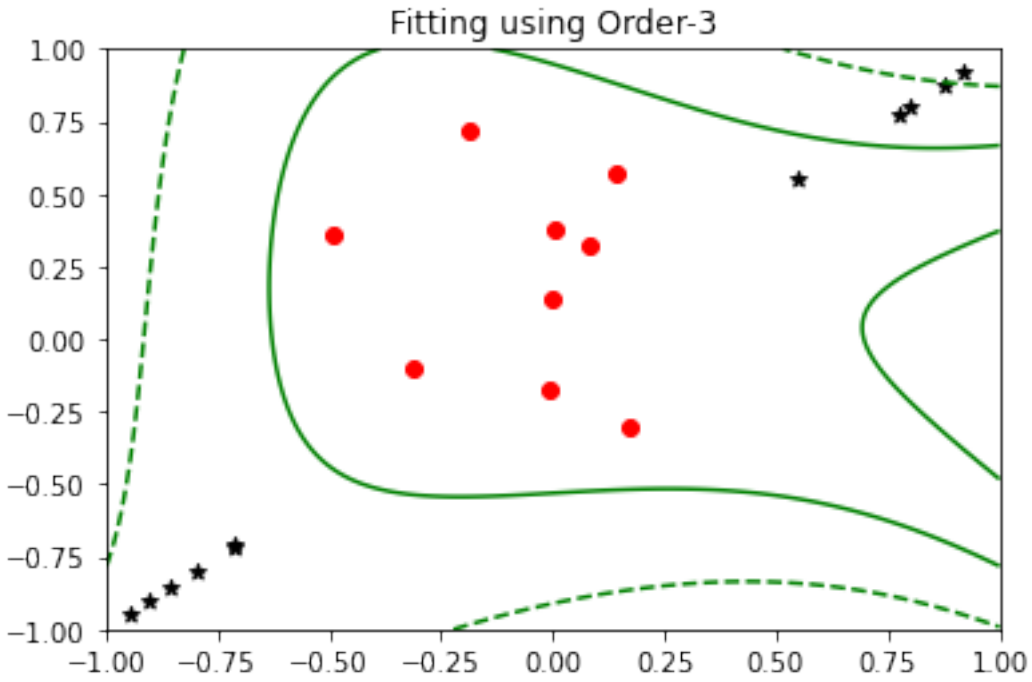
```
    return result_array
def contour_func_pseudo_3(x,y,w):
    return w[0] + w[1]*x + w[2]*y + w[3]*x*y + w[4]*(x**2) + w[5]*(y**2) +␣
 ↪w[6]*(x**3) + w[7]*(x**2)*y + w[8]*x*(y**2) + w[9]*y**3
```

[26]:
```
data_pseudo = feature_transformation_pseudo_3(all_datapoints)
x_transformed = data_pseudo[:,:-1]
y_transformed = data_pseudo[:,-1]
```

[27]:
```
regularization_lambda = 1
W_P = inv(x_transformed.T.dot(x_transformed) +␣
 ↪regularization_lambda*(x_transformed.T.dot(x_transformed))).
 ↪dot(x_transformed.T).dot(y_transformed)
```

[28]:
```
plt.scatter(x_coords_1,y_coords_1, color = 'r', marker = 'o')
plt.scatter(y_coords_2,y_coords_2,color='k', marker = '*')
plt.contour(X_MESHGRID, Y_MESHGRID, contour_func_pseudo_3(X_MESHGRID,␣
 ↪Y_MESHGRID,W_P), 3, colors = 'g')
plt.title('Fitting using Order-3')
```

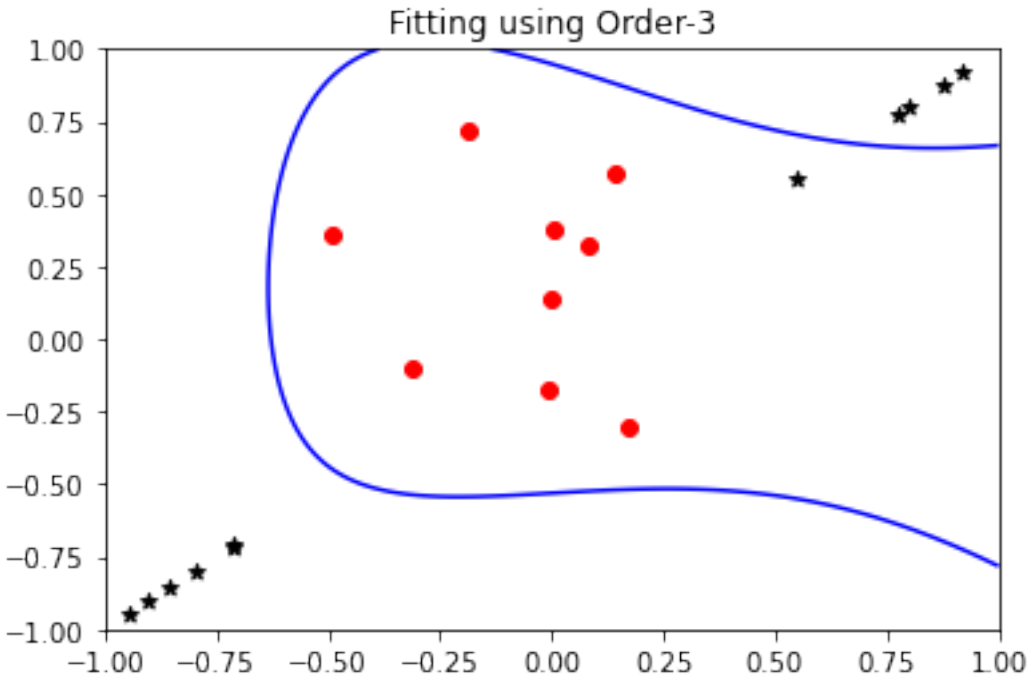
[28]:
```
Text(0.5, 1.0, 'Fitting using Order-3')
```



14 Plotting just the main contour

[29]:
```
plt.scatter(x_coords_1,y_coords_1, color = 'r', marker = 'o')
plt.scatter(y_coords_2,y_coords_2,color='k', marker = '*')
plt.contour(X_MESHGRID, Y_MESHGRID, contour_func_pseudo_3(X_MESHGRID,␣
 ↪Y_MESHGRID,W_P), 1, colors = 'b')
plt.title('Fitting using Order-3')
```

[29]:
```
Text(0.5, 1.0, 'Fitting using Order-3')
```



[ ]:
```

```

Q6_a_HW4

November 26, 2020

1 Question 6 (a)

End-to-end Machine Learning project

[1]:
```
# Python 3.5 is required
import sys
assert sys.version_info >= (3, 5)

# Scikit-Learn 0.20 is required
import sklearn
assert sklearn.__version__ >= "0.20"

# Common imports
import numpy as np
import os

# To plot pretty figures
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)

# Where to save the figures
PROJECT_ROOT_DIR = "."
CHAPTER_ID = "end_to_end_project"
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, "images", CHAPTER_ID)
os.makedirs(IMAGES_PATH, exist_ok=True)

def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)
```

```
# Ignore useless warnings (see SciPy issue #5998)
import warnings
warnings.filterwarnings(action="ignore", message="^internal gelsd")
```

2 Get the data

[2]:
```
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()
```

[3]:
```
fetch_housing_data()
```

[4]:
```
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)
```

[5]:
```
housing = load_housing_data()
housing.head()
```

[5]:

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

```
3       558.0       219.0         5.6431            341300.0        NEAR BAY
4       565.0       259.0         3.8462            342200.0        NEAR BAY
```

[6]:
```
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
```

[7]:
```
housing["ocean_proximity"].value_counts()
```

[7]:
```
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: ocean_proximity, dtype: int64
```

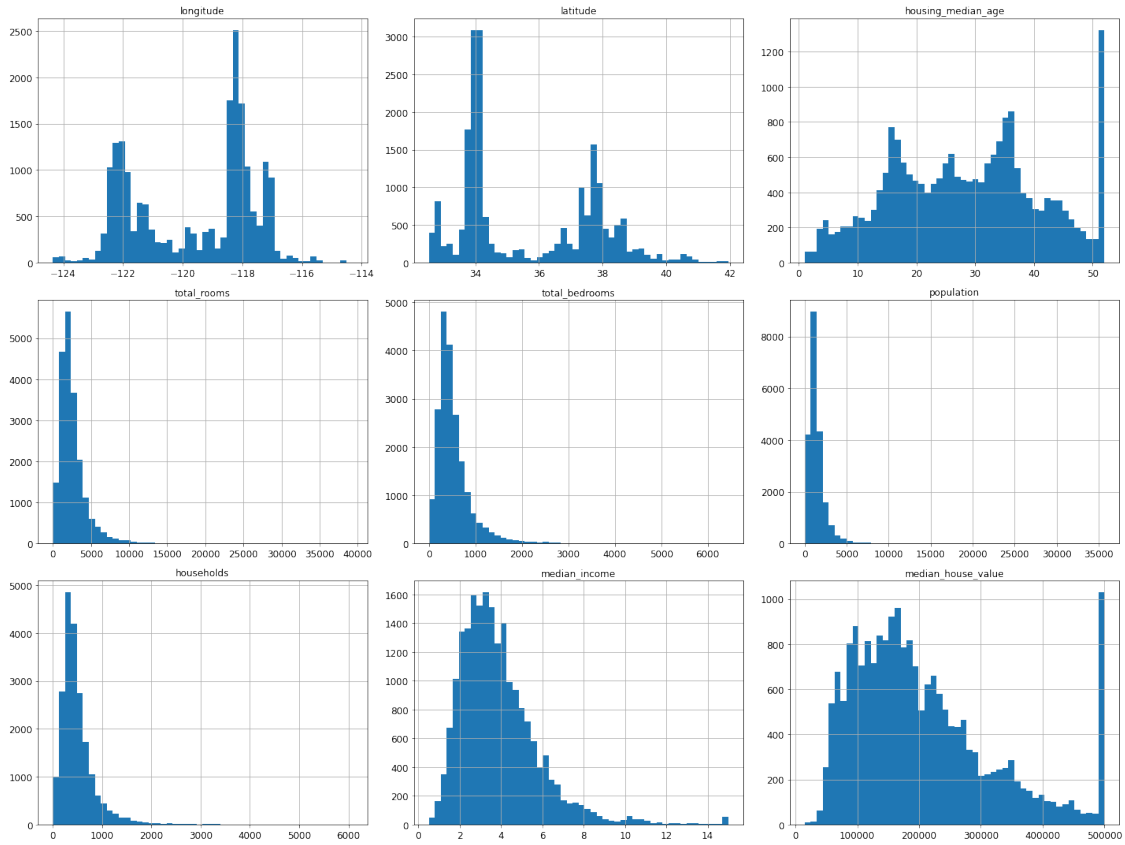
[8]:
```
housing.describe()
```

[8]:
```
          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
```

[9]:
```
%matplotlib inline
import matplotlib.pyplot as plt
housing.hist(bins=50, figsize=(20,15))
save_fig("attribute_histogram_plots")
plt.show()
```

```
Saving figure attribute_histogram_plots
```



[10]:
```
# to make this notebook's output identical at every run
np.random.seed(42)
```

[11]:
```
import numpy as np

# For illustration only. Sklearn has train_test_split()
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]
```

[12]:
```
train_set, test_set = split_train_test(housing, 0.2)
len(train_set)
```

[12]: 16512

[13]:
```
len(test_set)
```

[13]: 4128

[14]:
```
from zlib import crc32

def test_set_check(identifier, test_ratio):
    return crc32(np.int64(identifier)) & 0xffffffff < test_ratio * 2**32

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]
```

The implementation of `test_set_check()` above works fine in both Python 2 and Python 3. In earlier releases, the following implementation was proposed, which supported any hash function, but was much slower and did not support Python 2:

[15]:
```
import hashlib

def test_set_check(identifier, test_ratio, hash=hashlib.md5):
    return hash(np.int64(identifier)).digest()[-1] < 256 * test_ratio
```

If you want an implementation that supports any hash function and is compatible with both Python 2 and Python 3, here is one:

[16]:
```
def test_set_check(identifier, test_ratio, hash=hashlib.md5):
    return bytearray(hash(np.int64(identifier)).digest())[-1] < 256 * test_ratio
```

[17]:
```
housing_with_id = housing.reset_index()   # adds an `index` column
train_set, test_set = split_train_test_by_id(housing_with_id, 0.2, "index")
```

[18]:
```
housing_with_id["id"] = housing["longitude"] * 1000 + housing["latitude"]
train_set, test_set = split_train_test_by_id(housing_with_id, 0.2, "id")
```

[19]:
```
test_set.head()
```

[19]:

	index	longitude	latitude	housing_median_age	total_rooms	\
8	8	-122.26	37.84	42.0	2555.0	
10	10	-122.26	37.85	52.0	2202.0	
11	11	-122.26	37.85	52.0	3503.0	
12	12	-122.26	37.85	52.0	2491.0	
13	13	-122.26	37.84	52.0	696.0	

	total_bedrooms	population	households	median_income	median_house_value	\
8	665.0	1206.0	595.0	2.0804	226700.0	
10	434.0	910.0	402.0	3.2031	281500.0	
11	752.0	1504.0	734.0	3.2705	241800.0	
12	474.0	1098.0	468.0	3.0750	213500.0	
13	191.0	345.0	174.0	2.6736	191300.0	

	ocean_proximity	id

```
8          NEAR BAY -122222.16
10         NEAR BAY -122222.15
11         NEAR BAY -122222.15
12         NEAR BAY -122222.15
13         NEAR BAY -122222.16
```

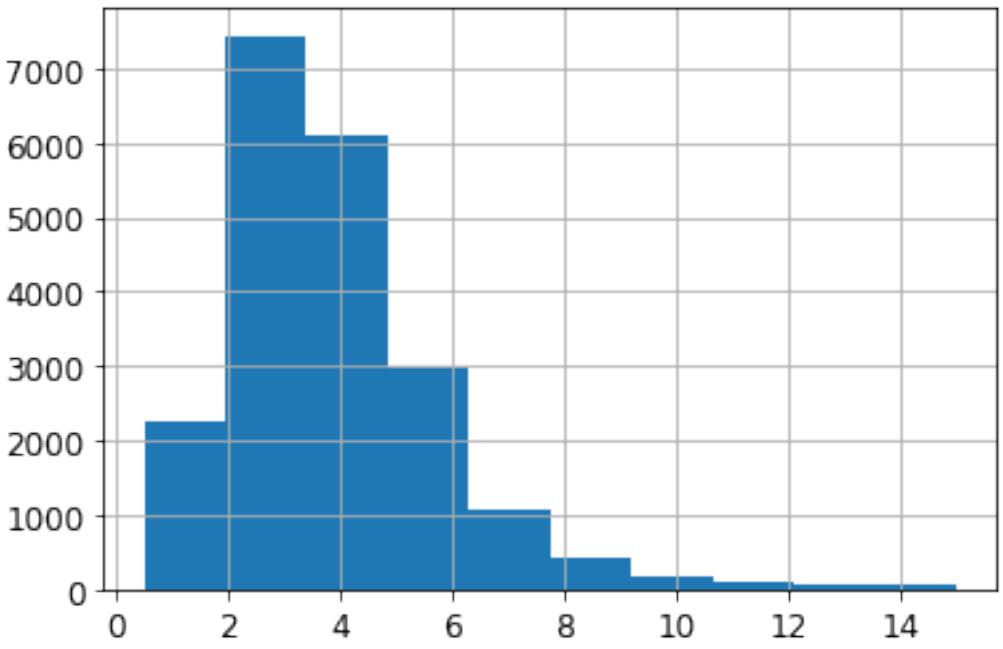
[20]:
```
from sklearn.model_selection import train_test_split

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

[21]:
```
test_set.head()
```

[21]:
```
       longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
20046    -119.01     36.06                25.0       1505.0             NaN
3024     -119.46     35.14                30.0       2943.0             NaN
15663    -122.44     37.80                52.0       3830.0             NaN
20484    -118.72     34.28                17.0       3051.0             NaN
9814     -121.93     36.62                34.0       2351.0             NaN

       population  households  median_income  median_house_value  \
20046      1392.0       359.0         1.6812             47700.0
3024       1565.0       584.0         2.5313             45800.0
15663      1310.0       963.0         3.4801            500001.0
20484      1705.0       495.0         5.7376            218600.0
9814       1063.0       428.0         3.7250            278000.0

      ocean_proximity
20046          INLAND
3024           INLAND
15663        NEAR BAY
20484       <1H OCEAN
9814       NEAR OCEAN
```

[22]:
```
housing["median_income"].hist()
```

[22]:
```
<AxesSubplot:>
```



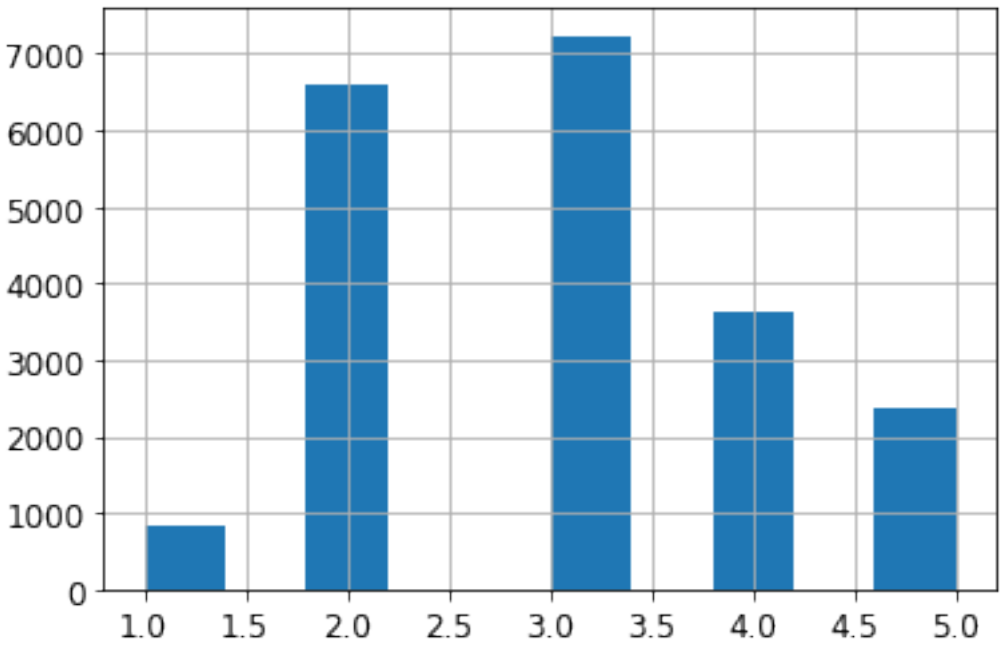
[23]:
```
housing["income_cat"] = pd.cut(housing["median_income"],
                               bins=[0., 1.5, 3.0, 4.5, 6., np.inf],
                               labels=[1, 2, 3, 4, 5])
```

[24]:
```
housing["income_cat"].value_counts()
```

[24]:
```
3    7236
2    6581
4    3639
5    2362
1     822
Name: income_cat, dtype: int64
```

[25]:
```
housing["income_cat"].hist()
```

[25]:
```
<AxesSubplot:>
```



[26]:
```
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]
```

[27]:
```
strat_test_set["income_cat"].value_counts() / len(strat_test_set)
```

[27]:
```
3    0.350533
2    0.318798
4    0.176357
5    0.114583
1    0.039729
Name: income_cat, dtype: float64
```

[28]:
```
housing["income_cat"].value_counts() / len(housing)
```

[28]:
```
3    0.350581
2    0.318847
4    0.176308
5    0.114438
1    0.039826
Name: income_cat, dtype: float64
```

[29]:
```
def income_cat_proportions(data):
    return data["income_cat"].value_counts() / len(data)

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

compare_props = pd.DataFrame({
    "Overall": income_cat_proportions(housing),
    "Stratified": income_cat_proportions(strat_test_set),
    "Random": income_cat_proportions(test_set),
}).sort_index()
compare_props["Rand. %error"] = 100 * compare_props["Random"] /␣
 ↪compare_props["Overall"] - 100
compare_props["Strat. %error"] = 100 * compare_props["Stratified"] /␣
 ↪compare_props["Overall"] - 100
```

[30]:
```
compare_props
```

[30]:

	Overall	Stratified	Random	Rand. %error	Strat. %error
1	0.039826	0.039729	0.040213	0.973236	-0.243309
2	0.318847	0.318798	0.324370	1.732260	-0.015195
3	0.350581	0.350533	0.358527	2.266446	-0.013820
4	0.176308	0.176357	0.167393	-5.056334	0.027480
5	0.114438	0.114583	0.109496	-4.318374	0.127011

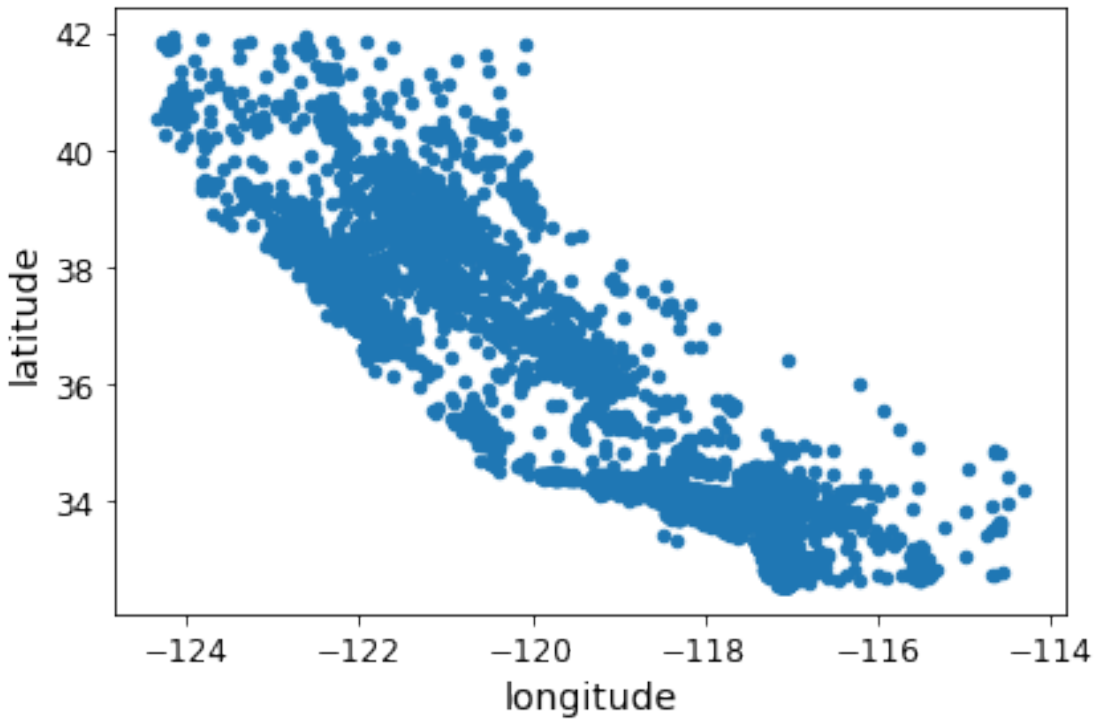
[31]:
```
for set_ in (strat_train_set, strat_test_set):
    set_.drop("income_cat", axis=1, inplace=True)
```

3 Discover and visualize the data to gain insights

[32]:
```
housing = strat_train_set.copy()
```

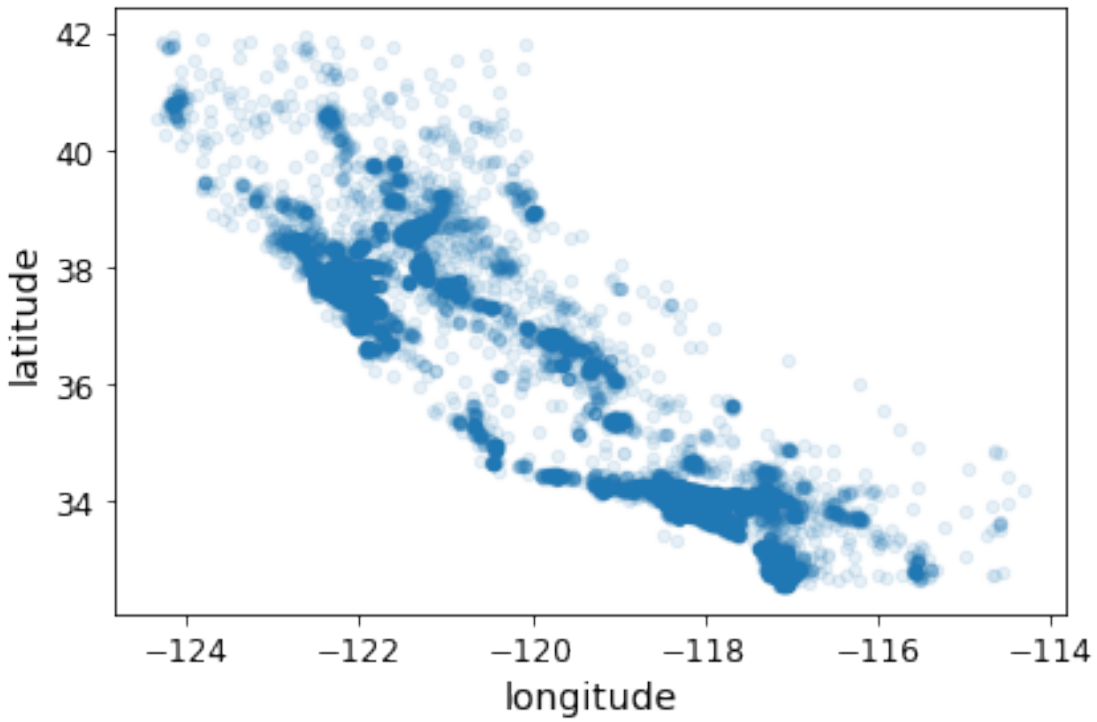
[33]:
```
housing.plot(kind="scatter", x="longitude", y="latitude")
save_fig("bad_visualization_plot")
```

```
Saving figure bad_visualization_plot
```



[34]:
```
housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.1)
save_fig("better_visualization_plot")
```

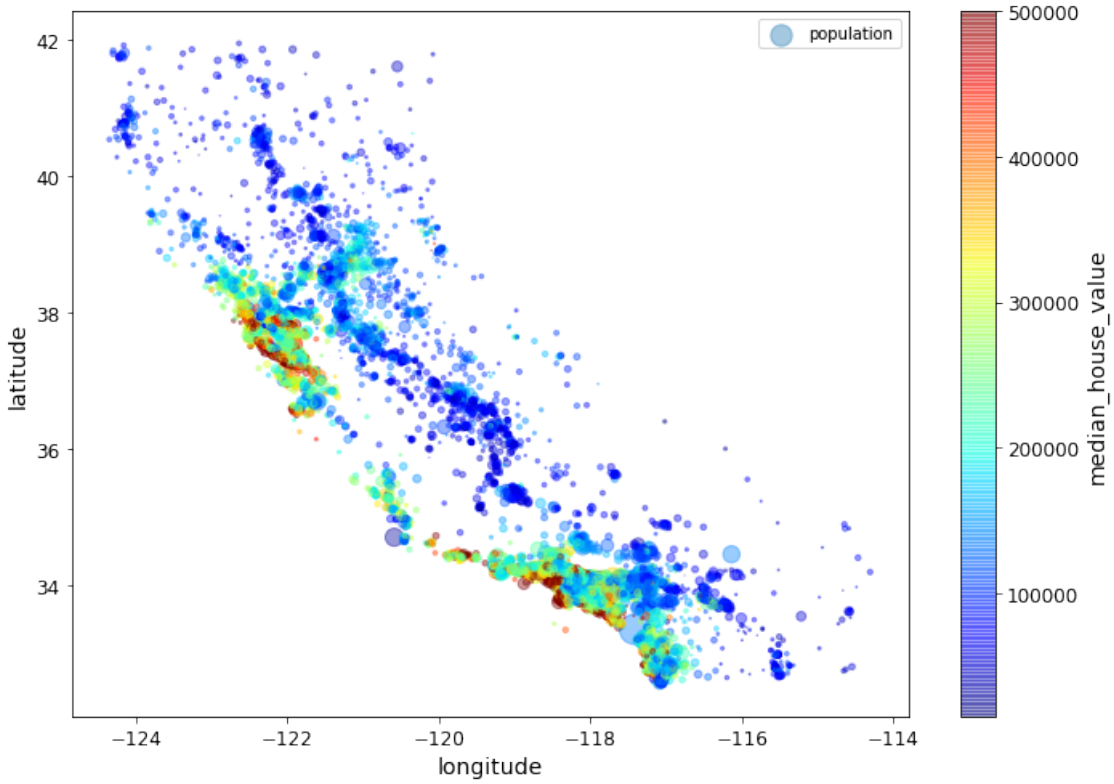
```
Saving figure better_visualization_plot
```



The argument `sharex=False` fixes a display bug (the x-axis values and legend were not displayed). This is a temporary fix (see: https://github.com/pandas-dev/pandas/issues/10611). Thanks to Wilmer Arellano for pointing it out.

[35]:
```
housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.4,
    s=housing["population"]/100, label="population", figsize=(10,7),
    c="median_house_value", cmap=plt.get_cmap("jet"), colorbar=True,
    sharex=False)
plt.legend()
save_fig("housing_prices_scatterplot")
```

```
Saving figure housing_prices_scatterplot
```



[36]:
```
# Download the California image
images_path = os.path.join(PROJECT_ROOT_DIR, "images", "end_to_end_project")
os.makedirs(images_path, exist_ok=True)
DOWNLOAD_ROOT = "https://raw.githubusercontent.com/ageron/handson-ml2/master/"
filename = "california.png"
print("Downloading", filename)
url = DOWNLOAD_ROOT + "images/end_to_end_project/" + filename
urllib.request.urlretrieve(url, os.path.join(images_path, filename))
```

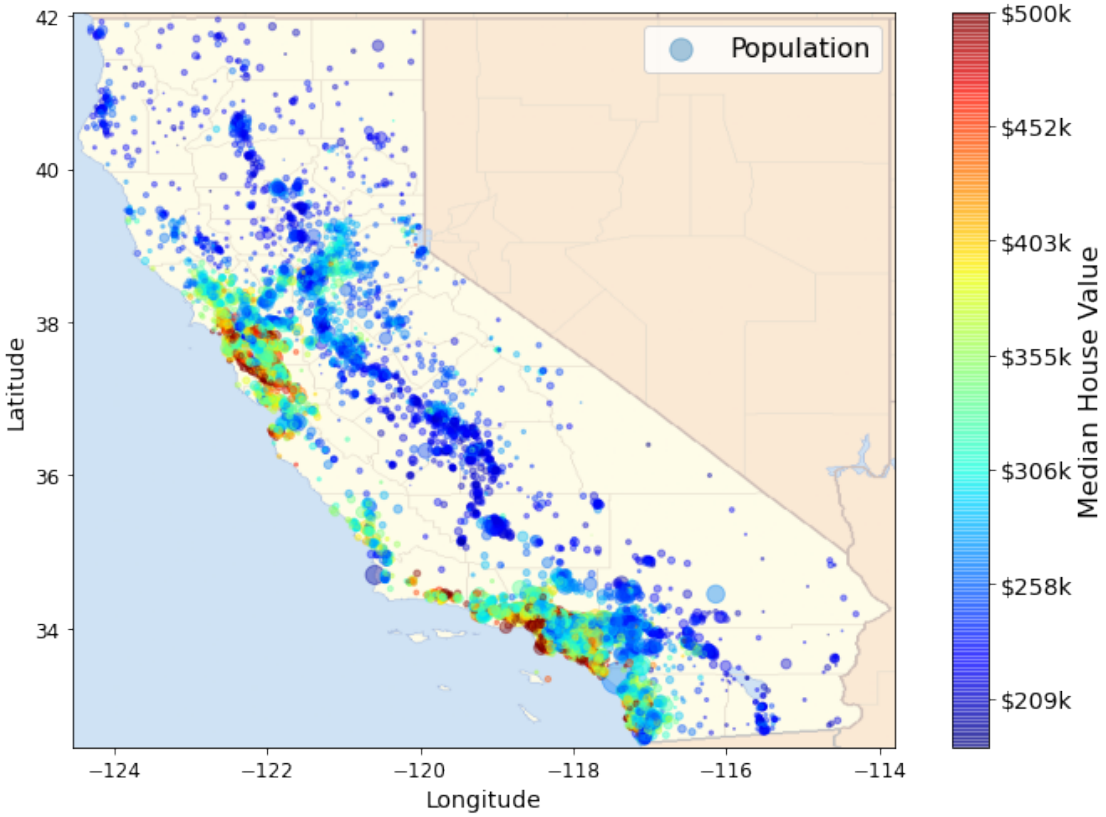
```
Downloading california.png
```

[36]:
```
('./images/end_to_end_project/california.png',
 <http.client.HTTPMessage at 0x7fe7a3b18ac0>)
```

[37]:
```
import matplotlib.image as mpimg
california_img=mpimg.imread(os.path.join(images_path, filename))
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)
save_fig("california_housing_prices_plot")
plt.show()
```

```
Saving figure california_housing_prices_plot
```



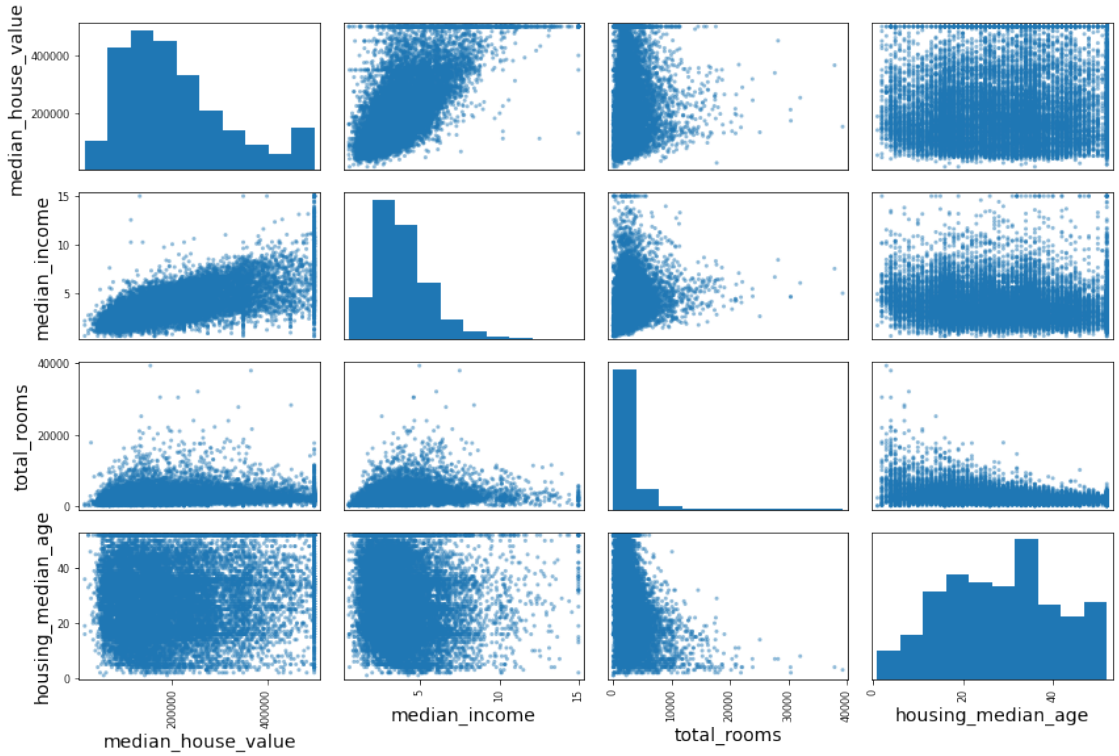
[38]:
```
corr_matrix = housing.corr()
```

[39]:
```
corr_matrix["median_house_value"].sort_values(ascending=False)
```

[39]:
```
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
```

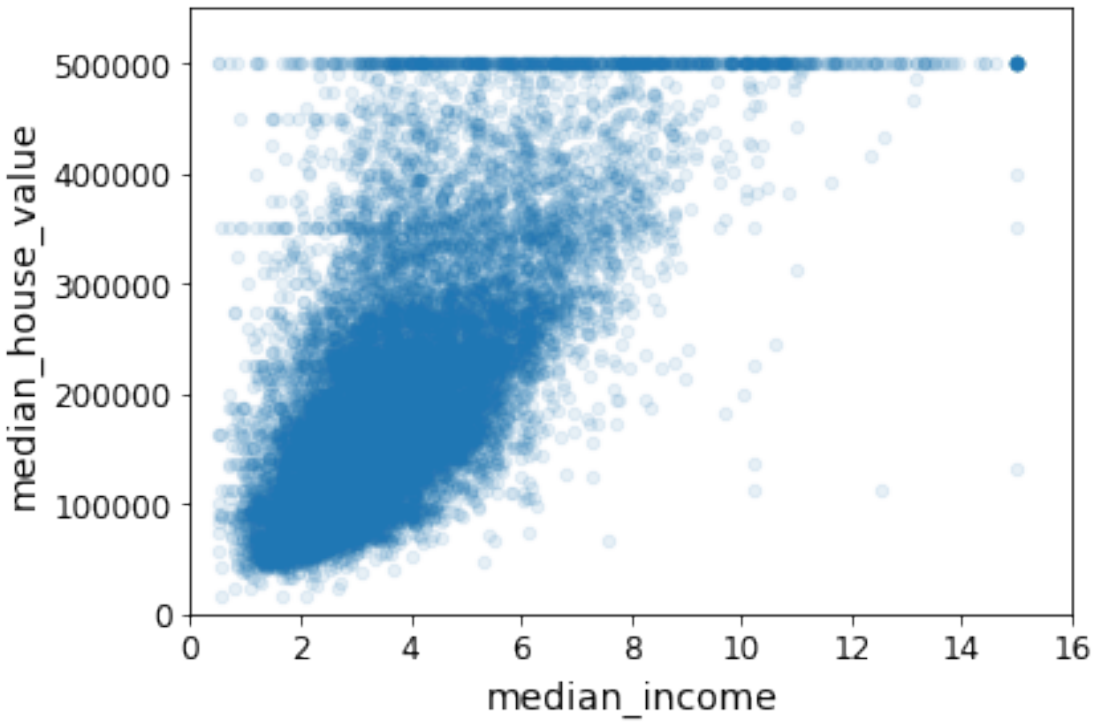
[40]:
```
# from pandas.tools.plotting import scatter_matrix # For older versions of
 ↪Pandas
from pandas.plotting import scatter_matrix

attributes = ["median_house_value", "median_income", "total_rooms",
              "housing_median_age"]
scatter_matrix(housing[attributes], figsize=(12, 8))
save_fig("scatter_matrix_plot")
```

```
Saving figure scatter_matrix_plot
```



[41]:
```
housing.plot(kind="scatter", x="median_income", y="median_house_value",
             alpha=0.1)
plt.axis([0, 16, 0, 550000])
save_fig("income_vs_house_value_scatterplot")
```

```
Saving figure income_vs_house_value_scatterplot
```



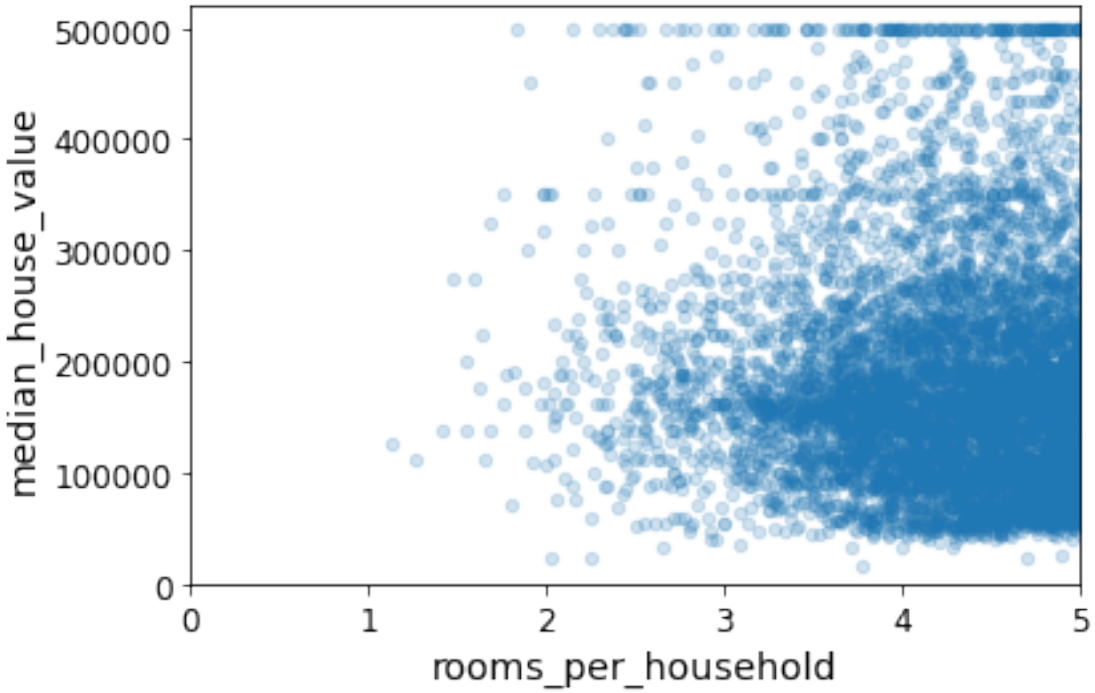
[42]:
```
housing["rooms_per_household"] = housing["total_rooms"]/housing["households"]
housing["bedrooms_per_room"] = housing["total_bedrooms"]/housing["total_rooms"]
housing["population_per_household"]=housing["population"]/housing["households"]
```

[43]:
```
corr_matrix = housing.corr()
corr_matrix["median_house_value"].sort_values(ascending=False)
```

[43]:
```
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
```

[44]:
```
housing.plot(kind="scatter", x="rooms_per_household", y="median_house_value",
             alpha=0.2)
plt.axis([0, 5, 0, 520000])
plt.show()
```



[45]:
```
housing.describe()
```

[45]:

	longitude	latitude	housing_median_age	total_rooms
count	16512.000000	16512.000000	16512.000000	16512.000000
mean	-119.575834	35.639577	28.653101	2622.728319
std	2.001860	2.138058	12.574726	2138.458419
min	-124.350000	32.540000	1.000000	6.000000
25%	-121.800000	33.940000	18.000000	1443.000000
50%	-118.510000	34.260000	29.000000	2119.500000
75%	-118.010000	37.720000	37.000000	3141.000000
max	-114.310000	41.950000	52.000000	39320.000000

	total_bedrooms	population	households	median_income
count	16354.000000	16512.000000	16512.000000	16512.000000
mean	534.973890	1419.790819	497.060380	3.875589

```
std        412.699041   1115.686241    375.720845       1.904950
min          2.000000      3.000000      2.000000       0.499900
25%        295.000000    784.000000    279.000000       2.566775
50%        433.000000   1164.000000    408.000000       3.540900
75%        644.000000   1719.250000    602.000000       4.744475
max       6210.000000  35682.000000   5358.000000      15.000100

       median_house_value  rooms_per_household  bedrooms_per_room  \
count        16512.000000         16512.000000       16354.000000
mean        206990.920724             5.440341           0.212878
std         115703.014830             2.611712           0.057379
min          14999.000000             1.130435           0.100000
25%         119800.000000             4.442040           0.175304
50%         179500.000000             5.232284           0.203031
75%         263900.000000             6.056361           0.239831
max         500001.000000           141.909091           1.000000

       population_per_household
count               16512.000000
mean                    3.096437
std                    11.584826
min                     0.692308
25%                     2.431287
50%                     2.817653
75%                     3.281420
max                  1243.333333
```

4 Prepare the data for Machine Learning algorithms

[46]:
```
housing = strat_train_set.drop("median_house_value", axis=1) # drop labels for␣
↪training set
housing_labels = strat_train_set["median_house_value"].copy()
```

[47]:
```
sample_incomplete_rows = housing[housing.isnull().any(axis=1)].head()
sample_incomplete_rows
```

[47]:
```
       longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
4629     -118.30     34.07                18.0       3759.0             NaN
6068     -117.86     34.01                16.0       4632.0             NaN
17923    -121.97     37.35                30.0       1955.0             NaN
13656    -117.30     34.05                 6.0       2155.0             NaN
19252    -122.79     38.48                 7.0       6837.0             NaN

       population  households  median_income ocean_proximity
4629       3296.0      1462.0         2.2708       <1H OCEAN
```

```
6068       3038.0       727.0         5.1762       <1H OCEAN
17923       999.0       386.0         4.6328       <1H OCEAN
13656      1039.0       391.0         1.6675          INLAND
19252      3468.0      1405.0         3.1662       <1H OCEAN
```

[48]:
```
sample_incomplete_rows.dropna(subset=["total_bedrooms"])    # option 1
```

[48]:
```
Empty DataFrame
Columns: [longitude, latitude, housing_median_age, total_rooms, total_bedrooms,
population, households, median_income, ocean_proximity]
Index: []
```

[49]:
```
sample_incomplete_rows.drop("total_bedrooms", axis=1)       # option 2
```

[49]:
```
       longitude  latitude  housing_median_age  total_rooms  population  \
4629     -118.30     34.07                18.0       3759.0      3296.0
6068     -117.86     34.01                16.0       4632.0      3038.0
17923    -121.97     37.35                30.0       1955.0       999.0
13656    -117.30     34.05                 6.0       2155.0      1039.0
19252    -122.79     38.48                 7.0       6837.0      3468.0

       households  median_income ocean_proximity
4629       1462.0         2.2708       <1H OCEAN
6068        727.0         5.1762       <1H OCEAN
17923       386.0         4.6328       <1H OCEAN
13656       391.0         1.6675          INLAND
19252      1405.0         3.1662       <1H OCEAN
```

[50]:
```
median = housing["total_bedrooms"].median()
sample_incomplete_rows["total_bedrooms"].fillna(median, inplace=True) # option 3
```

[51]:
```
sample_incomplete_rows
```

[51]:
```
       longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
4629     -118.30     34.07                18.0       3759.0           433.0
6068     -117.86     34.01                16.0       4632.0           433.0
17923    -121.97     37.35                30.0       1955.0           433.0
13656    -117.30     34.05                 6.0       2155.0           433.0
19252    -122.79     38.48                 7.0       6837.0           433.0

       population  households  median_income ocean_proximity
4629       3296.0      1462.0         2.2708       <1H OCEAN
6068       3038.0       727.0         5.1762       <1H OCEAN
17923       999.0       386.0         4.6328       <1H OCEAN
13656      1039.0       391.0         1.6675          INLAND
19252      3468.0      1405.0         3.1662       <1H OCEAN
```

[52]:
```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy="median")
```

Remove the text attribute because median can only be calculated on numerical attributes:

[53]:
```
housing_num = housing.drop("ocean_proximity", axis=1)
# alternatively: housing_num = housing.select_dtypes(include=[np.number])
```

[54]:
```
imputer.fit(housing_num)
```

[54]:
```
SimpleImputer(strategy='median')
```

[55]:
```
imputer.statistics_
```

[55]:
```
array([-118.51  ,   34.26  ,   29.    , 2119.5   ,  433.    , 1164.    ,
        408.    ,    3.5409])
```

Check that this is the same as manually computing the median of each attribute:

[56]:
```
housing_num.median().values
```

[56]:
```
array([-118.51  ,   34.26  ,   29.    , 2119.5   ,  433.    , 1164.    ,
        408.    ,    3.5409])
```

Transform the training set:

[57]:
```
X = imputer.transform(housing_num)
```

[58]:
```
housing_tr = pd.DataFrame(X, columns=housing_num.columns,
                          index=housing.index)
```

[59]:
```
housing_tr.loc[sample_incomplete_rows.index.values]
```

[59]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
4629	-118.30	34.07	18.0	3759.0	433.0	3296.0	1462.0	2.2708
6068	-117.86	34.01	16.0	4632.0	433.0	3038.0	727.0	5.1762
17923	-121.97	37.35	30.0	1955.0	433.0	999.0	386.0	4.6328
13656	-117.30	34.05	6.0	2155.0	433.0	1039.0	391.0	1.6675
19252	-122.79	38.48	7.0	6837.0	433.0	3468.0	1405.0	3.1662

[60]:
```
imputer.strategy
```

[60]:
```
'median'
```

[61]:
```
housing_tr = pd.DataFrame(X, columns=housing_num.columns,
                          index=housing_num.index)
```

[62]:
```
housing_tr.head()
```

[62]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	\
17606	-121.89	37.29	38.0	1568.0	351.0	
18632	-121.93	37.05	14.0	679.0	108.0	
14650	-117.20	32.77	31.0	1952.0	471.0	
3230	-119.61	36.31	25.0	1847.0	371.0	
3555	-118.59	34.23	17.0	6592.0	1525.0	

	population	households	median_income
17606	710.0	339.0	2.7042
18632	306.0	113.0	6.4214
14650	936.0	462.0	2.8621
3230	1460.0	353.0	1.8839
3555	4459.0	1463.0	3.0347

Now let's preprocess the categorical input feature, `ocean_proximity`:

[63]:
```
housing_cat = housing[["ocean_proximity"]]
housing_cat.head(10)
```

[63]:

	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

[64]:
```
from sklearn.preprocessing import OrdinalEncoder

ordinal_encoder = OrdinalEncoder()
housing_cat_encoded = ordinal_encoder.fit_transform(housing_cat)
housing_cat_encoded[:10]
```

[64]:
```
array([[0.],
       [0.],
       [4.],
```

```
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.]])
```

[65]:
```
ordinal_encoder.categories_
```

[65]:
```
[array(['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'],
       dtype=object)]
```

[66]:
```
from sklearn.preprocessing import OneHotEncoder

cat_encoder = OneHotEncoder()
housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
housing_cat_1hot
```

[66]:
```
<16512x5 sparse matrix of type '<class 'numpy.float64'>'
        with 16512 stored elements in Compressed Sparse Row format>
```

By default, the `OneHotEncoder` class returns a sparse array, but we can convert it to a dense array if needed by calling the `toarray()` method:

[67]:
```
housing_cat_1hot.toarray()
```

[67]:
```
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.]])
```

Alternatively, you can set `sparse=False` when creating the `OneHotEncoder`:

[68]:
```
cat_encoder = OneHotEncoder(sparse=False)
housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
housing_cat_1hot
```

[68]:
```
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.]])
```

[69]:
```
cat_encoder.categories_
```

[69]:
```
[array(['<1H OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'],
       dtype=object)]
```

Let's create a custom transformer to add extra attributes:

[70]:
```
from sklearn.base import BaseEstimator, TransformerMixin

# column index
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]

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

Note that I hard coded the indices (3, 4, 5, 6) for concision and clarity in the book, but it would be much cleaner to get them dynamically, like this:

[71]:
```
col_names = "total_rooms", "total_bedrooms", "population", "households"
rooms_ix, bedrooms_ix, population_ix, households_ix = [
    housing.columns.get_loc(c) for c in col_names] # get the column indices
```

Also, `housing_extra_attribs` is a NumPy array, we've lost the column names (unfortunately, that's a problem with Scikit-Learn). To recover a `DataFrame`, you could run this:

[72]:
```
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()
```

[72]:
```
       longitude latitude housing_median_age total_rooms total_bedrooms  \
17606   -121.89    37.29                 38        1568            351
18632   -121.93    37.05                 14         679            108
14650    -117.2    32.77                 31        1952            471
3230    -119.61    36.31                 25        1847            371
3555    -118.59    34.23                 17        6592           1525

       population households median_income ocean_proximity rooms_per_household  \
17606         710        339        2.7042       <1H OCEAN              4.62537
18632         306        113        6.4214       <1H OCEAN              6.00885
14650         936        462        2.8621      NEAR OCEAN              4.22511
3230         1460        353        1.8839          INLAND              5.23229
3555         4459       1463        3.0347       <1H OCEAN              4.50581

       population_per_household
17606                    2.0944
18632                   2.70796
14650                   2.02597
3230                    4.13598
3555                    3.04785
```

Now let's build a pipeline for preprocessing the numerical attributes:

[73]:
```
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)
```

[74]:
```
housing_num_tr
```

[74]:
```
array([[-1.15604281,  0.77194962,  0.74333089, …, -0.31205452,
        -0.08649871,  0.15531753],
       [-1.17602483,  0.6596948 , -1.1653172 , …,  0.21768338,
        -0.03353391, -0.83628902],
       [ 1.18684903, -1.34218285,  0.18664186, …, -0.46531516,
        -0.09240499,  0.4222004 ],
       …,
       [ 1.58648943, -0.72478134, -1.56295222, …,  0.3469342 ,
        -0.03055414, -0.52177644],
       [ 0.78221312, -0.85106801,  0.18664186, …,  0.02499488,
         0.06150916, -0.30340741],
```

```
        [-1.43579109,  0.99645926,  1.85670895, …, -0.22852947,
         -0.09586294,  0.10180567]])
```

[75]:
```
from sklearn.compose import ColumnTransformer

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

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

housing_prepared = full_pipeline.fit_transform(housing)
```

[76]:
```
housing_prepared
```

[76]:
```
array([[-1.15604281,  0.77194962,  0.74333089, …,  0.        ,
         0.        ,  0.        ],
       [-1.17602483,  0.6596948 , -1.1653172 , …,  0.        ,
         0.        ,  0.        ],
       [ 1.18684903, -1.34218285,  0.18664186, …,  0.        ,
         0.        ,  1.        ],
       …,
       [ 1.58648943, -0.72478134, -1.56295222, …,  0.        ,
         0.        ,  0.        ],
       [ 0.78221312, -0.85106801,  0.18664186, …,  0.        ,
         0.        ,  0.        ],
       [-1.43579109,  0.99645926,  1.85670895, …,  0.        ,
         1.        ,  0.        ]])
```

[77]:
```
housing_prepared.shape
```

[77]:
```
(16512, 16)
```

For reference, here is the old solution based on a `DataFrameSelector` transformer (to just select a subset of the Pandas `DataFrame` columns), and a `FeatureUnion`:

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

# Create a class to select numerical or categorical columns
class OldDataFrameSelector(BaseEstimator, TransformerMixin):
    def __init__(self, attribute_names):
        self.attribute_names = attribute_names
    def fit(self, X, y=None):
        return self
    def transform(self, X):
```

```
        return X[self.attribute_names].values
```

Now let's join all these components into a big pipeline that will preprocess both the numerical and the categorical features:

[79]:
```
num_attribs = list(housing_num)
cat_attribs = ["ocean_proximity"]

old_num_pipeline = Pipeline([
        ('selector', OldDataFrameSelector(num_attribs)),
        ('imputer', SimpleImputer(strategy="median")),
        ('attribs_adder', CombinedAttributesAdder()),
        ('std_scaler', StandardScaler()),
    ])

old_cat_pipeline = Pipeline([
        ('selector', OldDataFrameSelector(cat_attribs)),
        ('cat_encoder', OneHotEncoder(sparse=False)),
    ])
```

[80]:
```
from sklearn.pipeline import FeatureUnion

old_full_pipeline = FeatureUnion(transformer_list=[
        ("num_pipeline", old_num_pipeline),
        ("cat_pipeline", old_cat_pipeline),
    ])
```

[81]:
```
old_housing_prepared = old_full_pipeline.fit_transform(housing)
old_housing_prepared
```

[81]:
```
array([[-1.15604281,  0.77194962,  0.74333089, …,  0.        ,
         0.        ,  0.        ],
       [-1.17602483,  0.6596948 , -1.1653172 , …,  0.        ,
         0.        ,  0.        ],
       [ 1.18684903, -1.34218285,  0.18664186, …,  0.        ,
         0.        ,  1.        ],
       …,
       [ 1.58648943, -0.72478134, -1.56295222, …,  0.        ,
         0.        ,  0.        ],
       [ 0.78221312, -0.85106801,  0.18664186, …,  0.        ,
         0.        ,  0.        ],
       [-1.43579109,  0.99645926,  1.85670895, …,  0.        ,
         1.        ,  0.        ]])
```

The result is the same as with the `ColumnTransformer`:

[82]:
```
np.allclose(housing_prepared, old_housing_prepared)
```

[82]:
```
True
```

5 Select and train a model

[83]:
```
from sklearn.linear_model import LinearRegression

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

[83]:
```
LinearRegression()
```

[84]:
```
# let's try the full preprocessing 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)

print("Predictions:", lin_reg.predict(some_data_prepared))
```

```
Predictions: [210644.60459286 317768.80697211 210956.43331178  59218.98886849
 189747.55849879]
```

Compare against the actual values:

[85]:
```
print("Labels:", list(some_labels))
```

```
Labels: [286600.0, 340600.0, 196900.0, 46300.0, 254500.0]
```

[86]:
```
some_data_prepared
```

[86]:
```
array([[-1.15604281,  0.77194962,  0.74333089, -0.49323393, -0.44543821,
        -0.63621141, -0.42069842, -0.61493744, -0.31205452, -0.08649871,
         0.15531753,  1.        ,  0.        ,  0.        ,  0.        ,
         0.        ],
       [-1.17602483,  0.6596948 , -1.1653172 , -0.90896655, -1.0369278 ,
        -0.99833135, -1.02222705,  1.33645936,  0.21768338, -0.03353391,
        -0.83628902,  1.        ,  0.        ,  0.        ,  0.        ,
         0.        ],
       [ 1.18684903, -1.34218285,  0.18664186, -0.31365989, -0.15334458,
        -0.43363936, -0.0933178 , -0.5320456 , -0.46531516, -0.09240499,
         0.4222004 ,  0.        ,  0.        ,  0.        ,  0.        ,
         1.        ],
       [-0.01706767,  0.31357576, -0.29052016, -0.36276217, -0.39675594,
         0.03604096, -0.38343559, -1.04556555, -0.07966124,  0.08973561,
        -0.19645314,  0.        ,  1.        ,  0.        ,  0.        ,
         0.        ],
       [ 0.49247384, -0.65929936, -0.92673619,  1.85619316,  2.41221109,
```

```
        2.72415407,  2.57097492, -0.44143679, -0.35783383, -0.00419445,
        0.2699277 ,  1.        ,  0.        ,  0.        ,  0.        ,
        0.        ]])
```

[87]:
```
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
```

[87]:
```
68628.19819848922
```

[88]:
```
from sklearn.metrics import mean_absolute_error

lin_mae = mean_absolute_error(housing_labels, housing_predictions)
lin_mae
```

[88]:
```
49439.89599001897
```

[89]:
```
from sklearn.tree import DecisionTreeRegressor

tree_reg = DecisionTreeRegressor(random_state=42)
tree_reg.fit(housing_prepared, housing_labels)
```

[89]:
```
DecisionTreeRegressor(random_state=42)
```

[90]:
```
housing_predictions = tree_reg.predict(housing_prepared)
tree_mse = mean_squared_error(housing_labels, housing_predictions)
tree_rmse = np.sqrt(tree_mse)
tree_rmse
```

[90]:
```
0.0
```

6 Fine-tune your model

[91]:
```
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)
```

[92]:
```
def display_scores(scores):
    print("Scores:", scores)
    print("Mean:", scores.mean())
```

```
    print("Standard deviation:", scores.std())

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
```

[93]:
```
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: 2731.674001798348
```

Note: we specify `n_estimators=100` to be future-proof since the default value is going to change to 100 in Scikit-Learn 0.22 (for simplicity, this is not shown in the book).

[94]:
```
from sklearn.ensemble import RandomForestRegressor

forest_reg = RandomForestRegressor(n_estimators=100, random_state=42)
forest_reg.fit(housing_prepared, housing_labels)
```

[94]:
```
RandomForestRegressor(random_state=42)
```

[95]:
```
housing_predictions = forest_reg.predict(housing_prepared)
forest_mse = mean_squared_error(housing_labels, housing_predictions)
forest_rmse = np.sqrt(forest_mse)
forest_rmse
```

[95]:
```
18603.515021376355
```

[96]:
```
from sklearn.model_selection import cross_val_score

forest_scores = cross_val_score(forest_reg, housing_prepared, housing_labels,
                                scoring="neg_mean_squared_error", cv=10)
forest_rmse_scores = np.sqrt(-forest_scores)
display_scores(forest_rmse_scores)
```

```
Scores: [49519.80364233 47461.9115823  50029.02762854 52325.28068953
 49308.39426421 53446.37892622 48634.8036574  47585.73832311
 53490.10699751 50021.5852922 ]
```

```
Mean: 50182.303100336096
Standard deviation: 2097.0810550985693
```

[97]:
```
scores = cross_val_score(lin_reg, housing_prepared, housing_labels,␣
 ↪scoring="neg_mean_squared_error", cv=10)
pd.Series(np.sqrt(-scores)).describe()
```

[97]:
```
count       10.000000
mean     69052.461363
std       2879.437224
min      64969.630564
25%      67136.363758
50%      68156.372635
75%      70982.369487
max      74739.570526
dtype: float64
```

[98]:
```
from sklearn.svm import SVR

svm_reg = SVR(kernel="linear")
svm_reg.fit(housing_prepared, housing_labels)
housing_predictions = svm_reg.predict(housing_prepared)
svm_mse = mean_squared_error(housing_labels, housing_predictions)
svm_rmse = np.sqrt(svm_mse)
svm_rmse
```

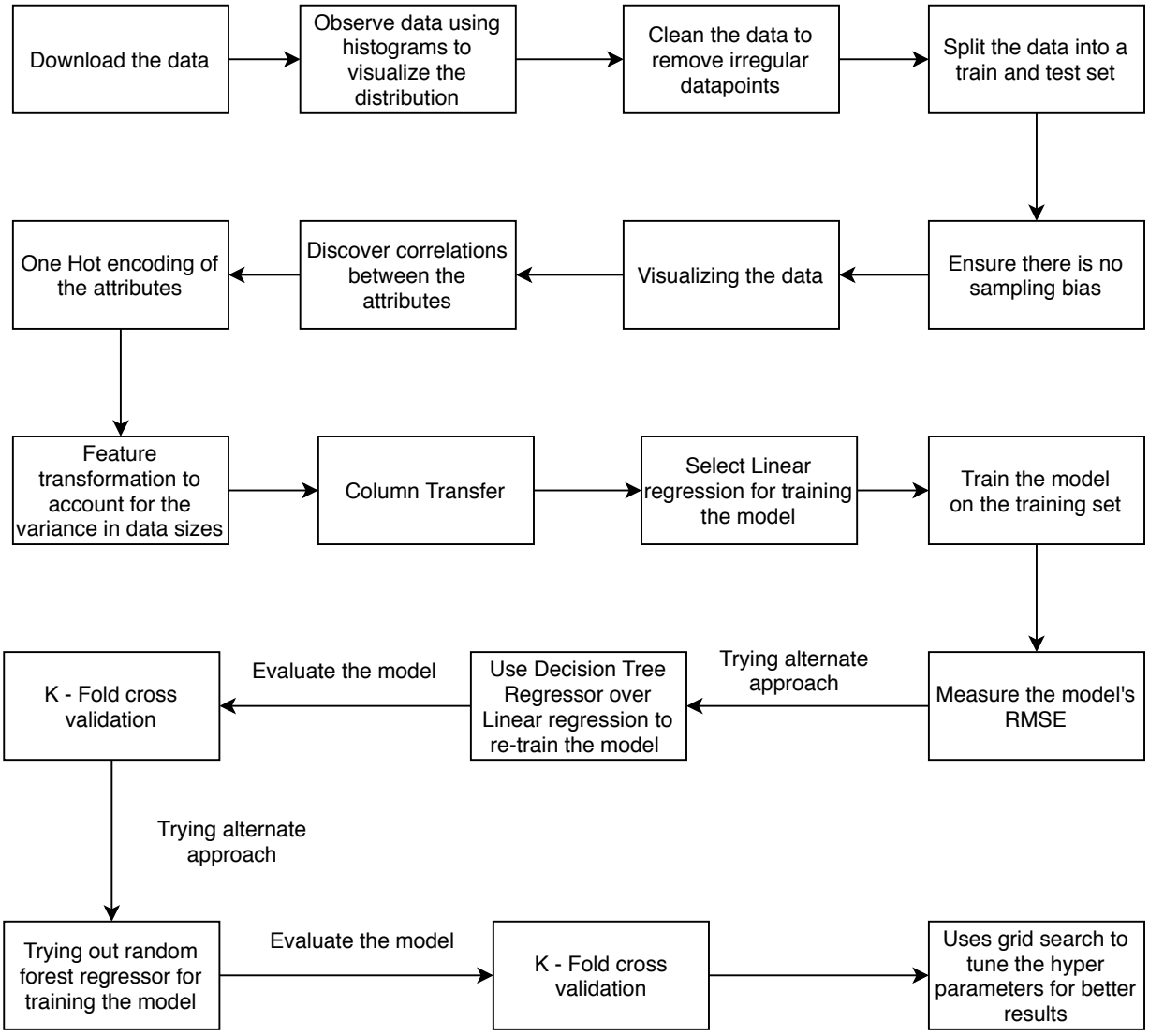
[98]:
```
111094.6308539982
```

[99]:
```
from sklearn.model_selection import GridSearchCV

param_grid = [
    # try 12 (3×4) combinations of hyperparameters
    {'n_estimators': [3, 10, 30], 'max_features': [2, 4, 6, 8]},
    # then try 6 (2×3) combinations with bootstrap set as False
    {'bootstrap': [False], 'n_estimators': [3, 10], 'max_features': [2, 3, 4]},
  ]

forest_reg = RandomForestRegressor(random_state=42)
# train across 5 folds, that's a total of (12+6)*5=90 rounds of training
grid_search = GridSearchCV(forest_reg, param_grid, cv=5,
                           scoring='neg_mean_squared_error',
                           return_train_score=True)
grid_search.fit(housing_prepared, housing_labels)
```

[99]:
```
GridSearchCV(cv=5, estimator=RandomForestRegressor(random_state=42),
             param_grid=[{'max_features': [2, 4, 6, 8],
                          'n_estimators': [3, 10, 30]},
```

```
            {'bootstrap': [False], 'max_features': [2, 3, 4],
             'n_estimators': [3, 10]}],
return_train_score=True, scoring='neg_mean_squared_error')
```

Question 6 (b)


Download the data
Observe data using histograms to visualize the distribution
Clean the data to remove irregular datapoints
Split the data into a train and test set
Ensure there is no sampling bias
Visualizing the data
Discover correlations between the attributes
One Hot encoding of the attributes
Feature transformation to account for the variance in data sizes
Column Transfer
Select Linear regression for training the model
Train the model on the training set
Measure the model's RMSE
Trying alternate approach
Use Decision Tree Regressor over Linear regression to re-train the model
Evaluate the model
K - Fold cross validation
Trying alternate approach
Trying out random forest regressor for training the model
Evaluate the model
K - Fold cross validation
Uses grid search to tune the hyper parameters for better results

Q 6 (c)

Some of the inadequacies of the pipeline can be addressed by suggesting certain improvements. These improvements can be seen as below:

1. Some of the ways in which the ML Model can be improved is by swapping out the grid search method for randomized search. This can help when the search space is really large.

2. Finding better combinations of the different parameters of the system will certainly help with fine-tuning the model.

3. Using a test set to evaluate the model is a good practice to strengthen every ML model. Be sure to evaluate with an appropriately sized test set with a good variety of data so that the model doesn't over fit and generalizes.

Q 6 (d)

Not suggesting any changes to this pipeline as this works efficiently with certain possible inadequacies addressed above.