DS-210: Programming for Data Science

Lecture 10: Measuring errors for regression. Loss functions.

Reminders

Homework:

- Please start your homework from

  `Collaborators: ???`
- `Collaborators: none` if no collaborators

Reminders

Homework:

- Please start your homework from `Collaborators: ???`
- `Collaborators: none` if no collaborators

Typical predictive data analysis pipeline:

- **Very important:** split your data into a training and test part
- Train your model on the training part
- Use the testing part to evaluate accuracy

Importing libraries we will use today

In [1]:
```
import numpy as np
from scipy.optimize import least_squares
import matplotlib.pyplot as plt
import math
```

Measuring errors for regression

- Usually, the predictor is not ideal
- How do I evaluate different options and choose the best one?

Definition of an outlier

A point or small set of points that are *"different"*

Definition of an outlier

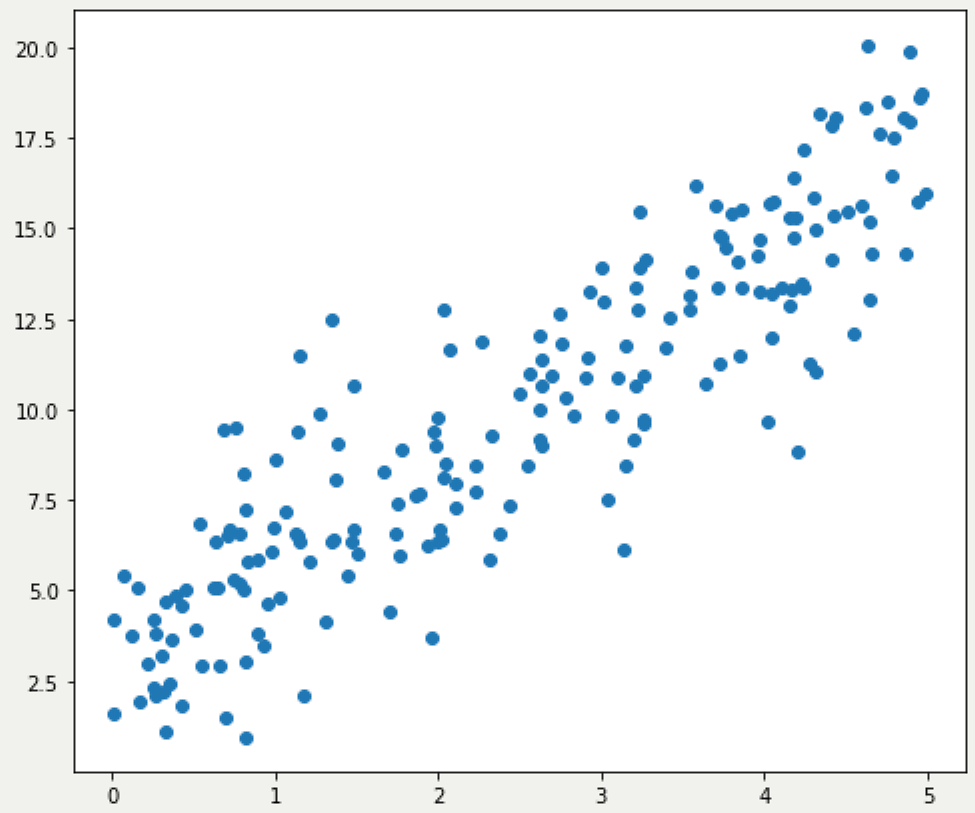
A point or small set of points that are *"different"*

Important difference between error measures: different attention to outliers

Constructing an example with outliers

In [3]:

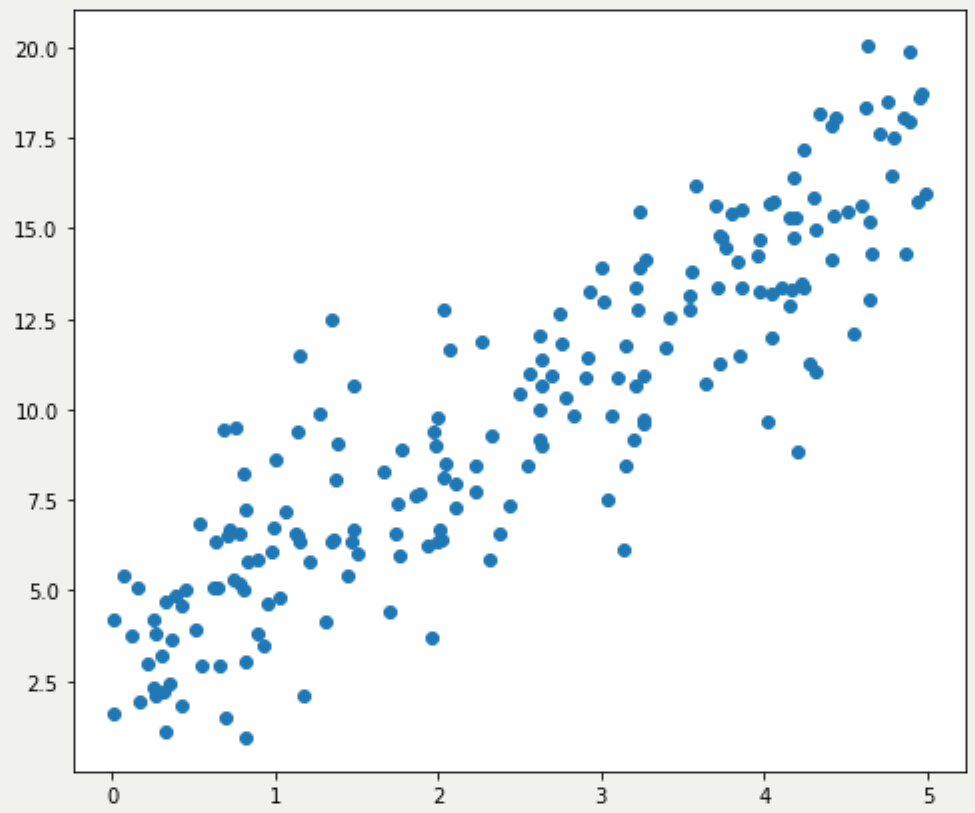
```
A,B = 0.0,5.0
RANGE = np.array([A,B])

SAMPLES = 200
x = np.random.uniform(low=A,high=B,size=SAMPLES)
y = x * math.e + math.pi \
    + np.random.normal(scale=2, size=SAMPLES)
plt.figure(figsize=(8,7))
plt.plot(x,y,"o");
```



Constructing an example with outliers

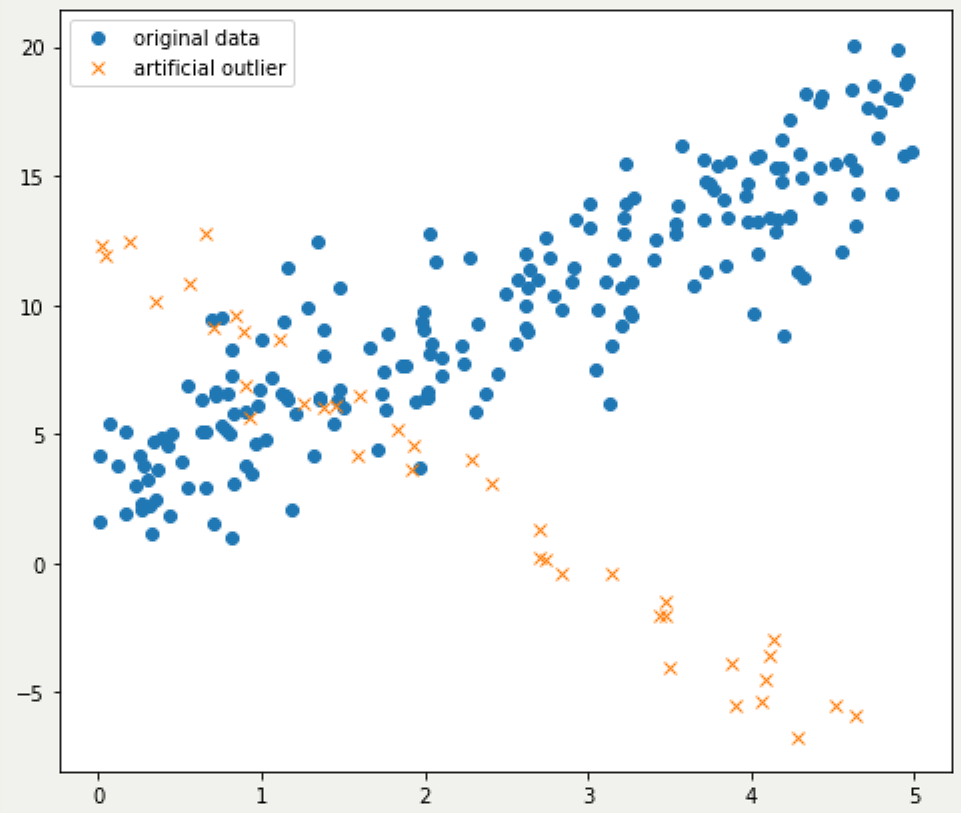
In [3]:

```
A,B = 0.0,5.0
RANGE = np.array([A,B])

SAMPLES = 200
x = np.random.uniform(low=A,high=B,size=SAMPLES)
y = x * math.e + math.pi \
    + np.random.normal(scale=2, size=SAMPLES)
plt.figure(figsize=(8,7))
plt.plot(x,y,"o");
```



In [4]:

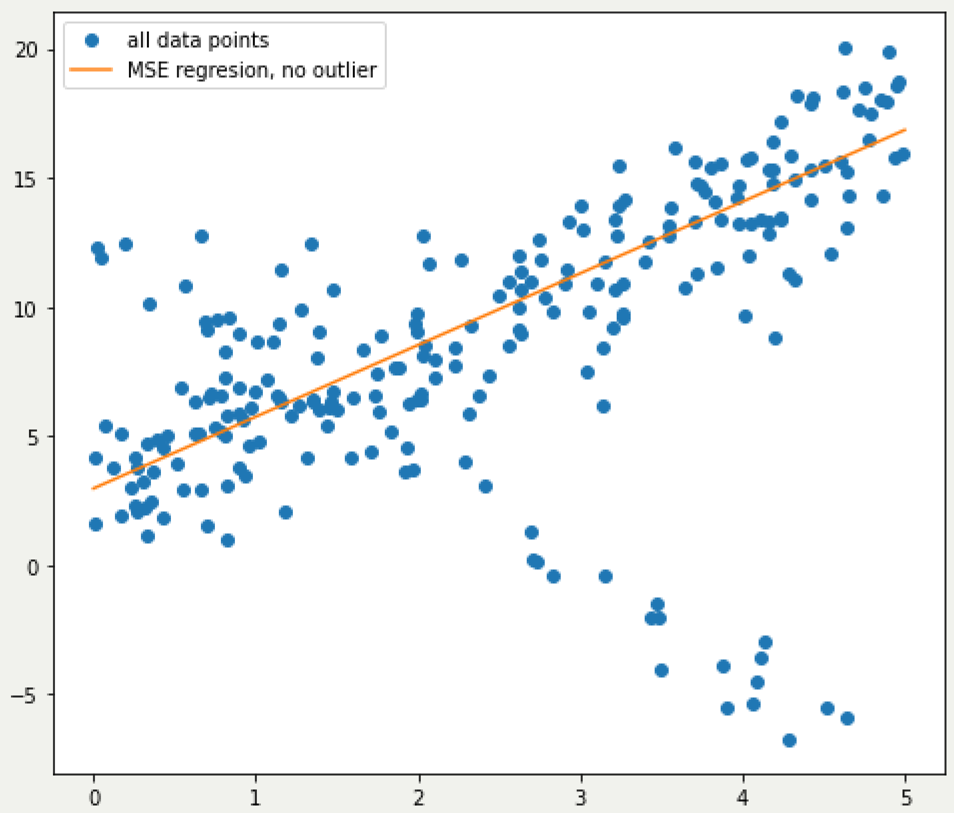
```
SAMPLES_2 = 40
x_2 = np.random.uniform(low=A,high=B,size=SAMPLES_2)
y_2 = x_2 * - 4 + 12 \
    + np.random.normal(scale=1, size=SAMPLES_2)
x_c = np.concatenate((x,x_2))
y_c = np.concatenate((y,y_2))
plt.figure(figsize=(8,7))
plt.plot(x,y,"o",x_2,y_2,"x")
plt.legend(["original data","artificial outlier"]);
```



Linear Regression with Mean Squared Error

In [5]:
```
# apply a linear transform
def F(x,c):
    return x * c[0] + c[1]

def error(c):
    return F(x,c) - y

sol_mse = least_squares(error,[0.0,0.0]).x
print(sol_mse)
plt.figure(figsize=(8,7))
plt.plot(x_c,y_c,"o",
         RANGE,F(RANGE,sol_mse),"-");
plt.legend(["all data points",
            "MSE regresion, no outlier"]);
```

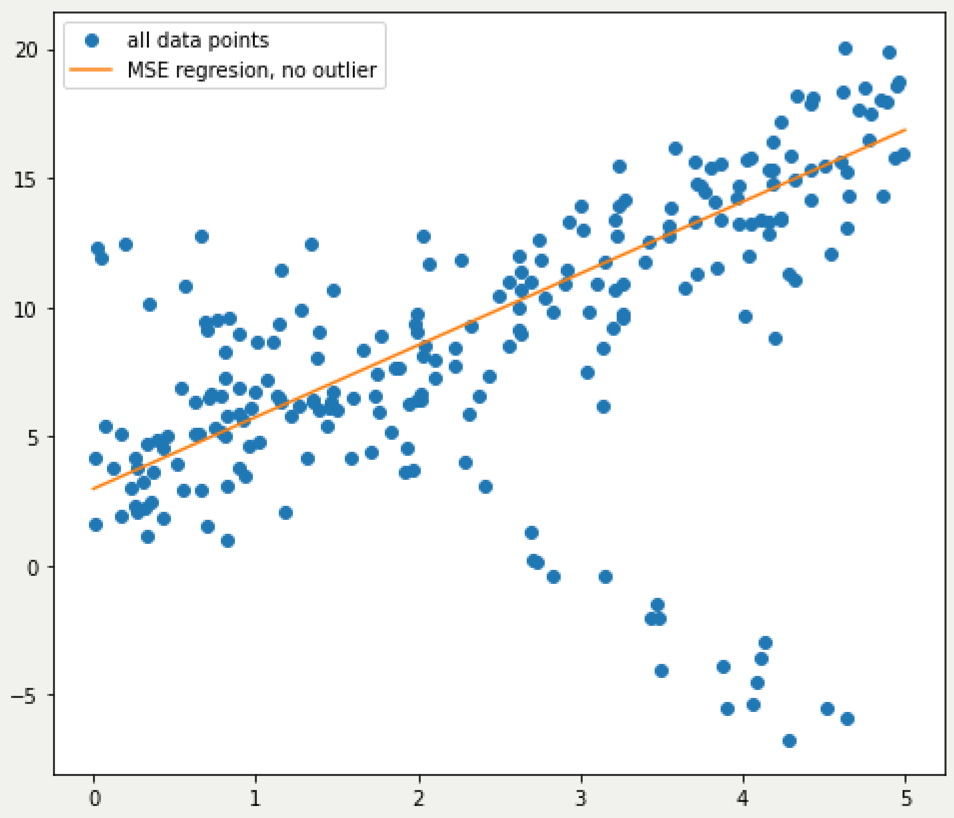
```
[2.77656518 2.96212176]
```



Linear Regression with Mean Squared Error

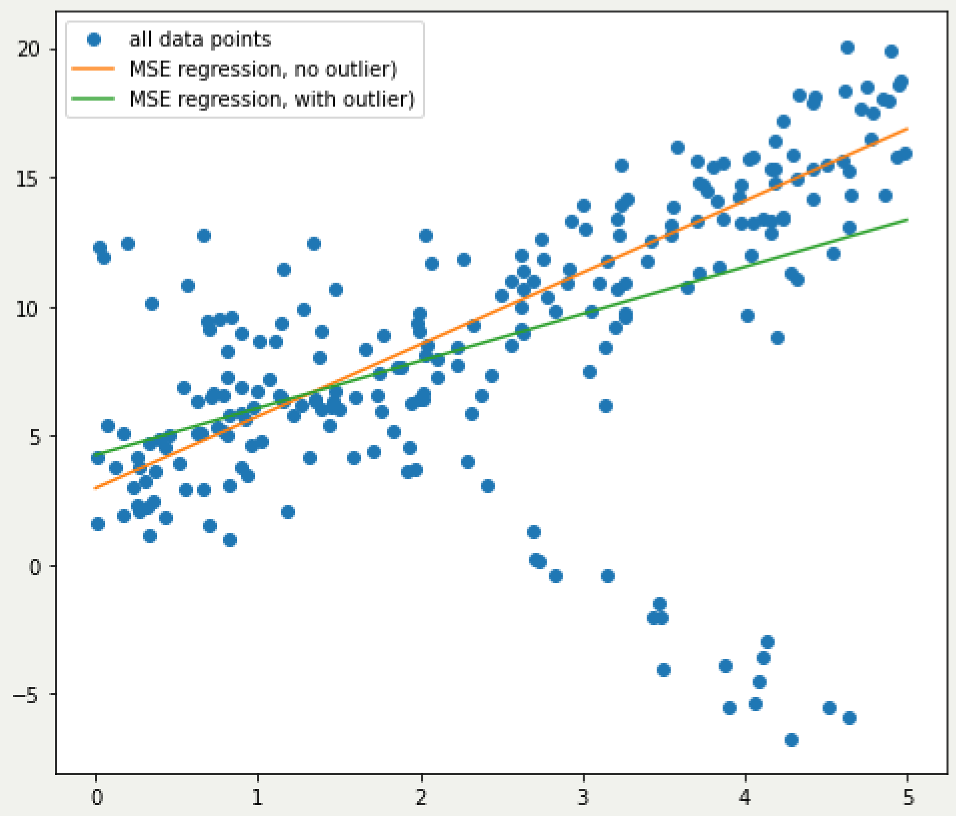
In [5]:
```
# apply a linear transform
def F(x,c):
    return x * c[0] + c[1]

def error(c):
    return F(x,c) - y

sol_mse = least_squares(error,[0.0,0.0]).x
print(sol_mse)
plt.figure(figsize=(8,7))
plt.plot(x_c,y_c,"o",
         RANGE,F(RANGE,sol_mse),"-");
plt.legend(["all data points",
            "MSE regresion, no outlier"]);
```

```
[2.77656518 2.96212176]
```



In [6]:
```
def error(c):
    return F(x_c,c) - y_c

sol_mse_2 = least_squares(error,[0.0,0.0]).x
print(sol_mse_2)
plt.figure(figsize=(8,7))
plt.plot(x_c,y_c,"o",
         RANGE,F(RANGE,sol_mse),"-",
         RANGE,F(RANGE,sol_mse_2),"-")
plt.legend(["all data points",
            "MSE regression, no outlier)",
            "MSE regression, with outlier)"]);
```

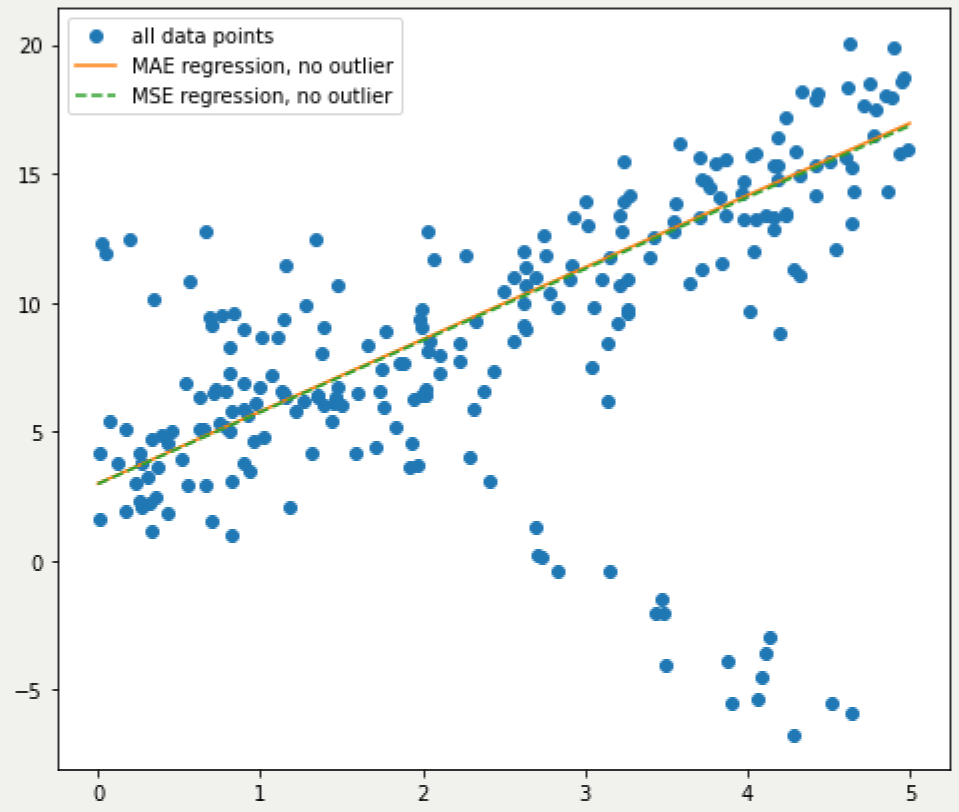
```
[1.81789917 4.24155588]
```



Linear Regression with Mean Absolute Error

In [7]:
```
def error(c):
    return np.sqrt(np.abs(F(x,c) - y))

sol_mae = least_squares(error,[0.0,0.0]).x
print(sol_mae)
plt.figure(figsize=(8,7))
plt.plot(x_c,y_c,"o",
         RANGE,F(RANGE,sol_mae),"-",
         RANGE,F(RANGE,sol_mse),"--");
plt.legend(["all data points",
            "MAE regression, no outlier",
            "MSE regression, no outlier"]);
```

```
[2.79244327 2.9804278 ]
```

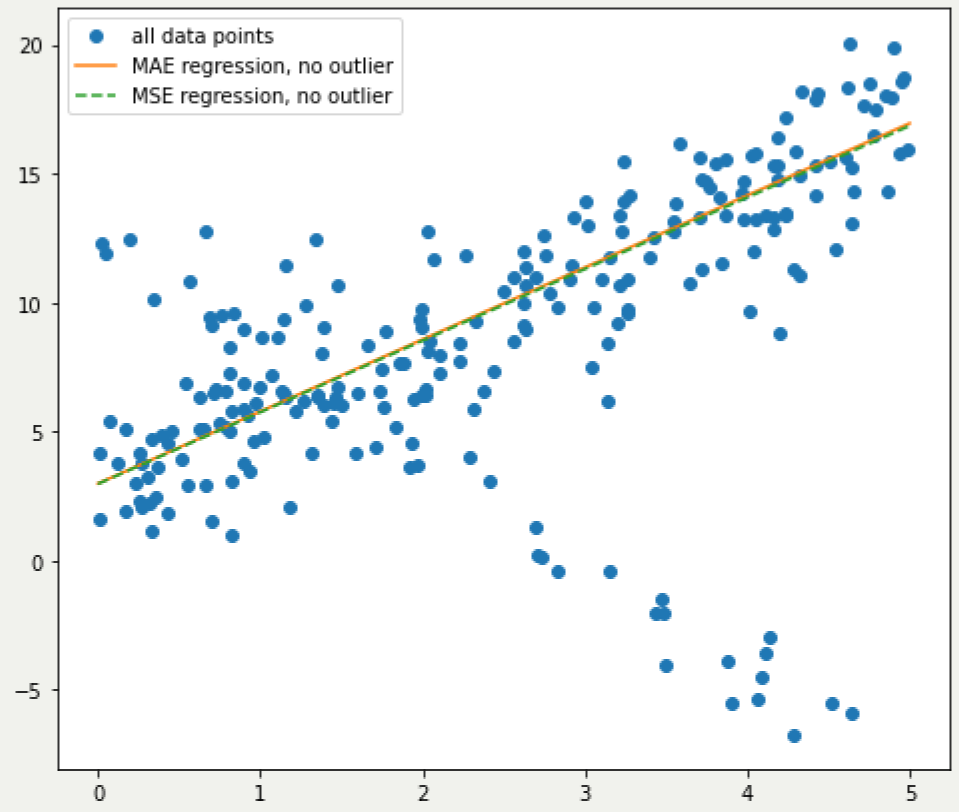


Linear Regression with Mean Absolute Error

In [7]:

```
def error(c):
    return np.sqrt(np.abs(F(x,c) - y))

sol_mae = least_squares(error,[0.0,0.0]).x
print(sol_mae)
plt.figure(figsize=(8,7))
plt.plot(x_c,y_c,"o",
         RANGE,F(RANGE,sol_mae),"-",
         RANGE,F(RANGE,sol_mse),"--");
plt.legend(["all data points",
            "MAE regression, no outlier",
            "MSE regression, no outlier"]);
```

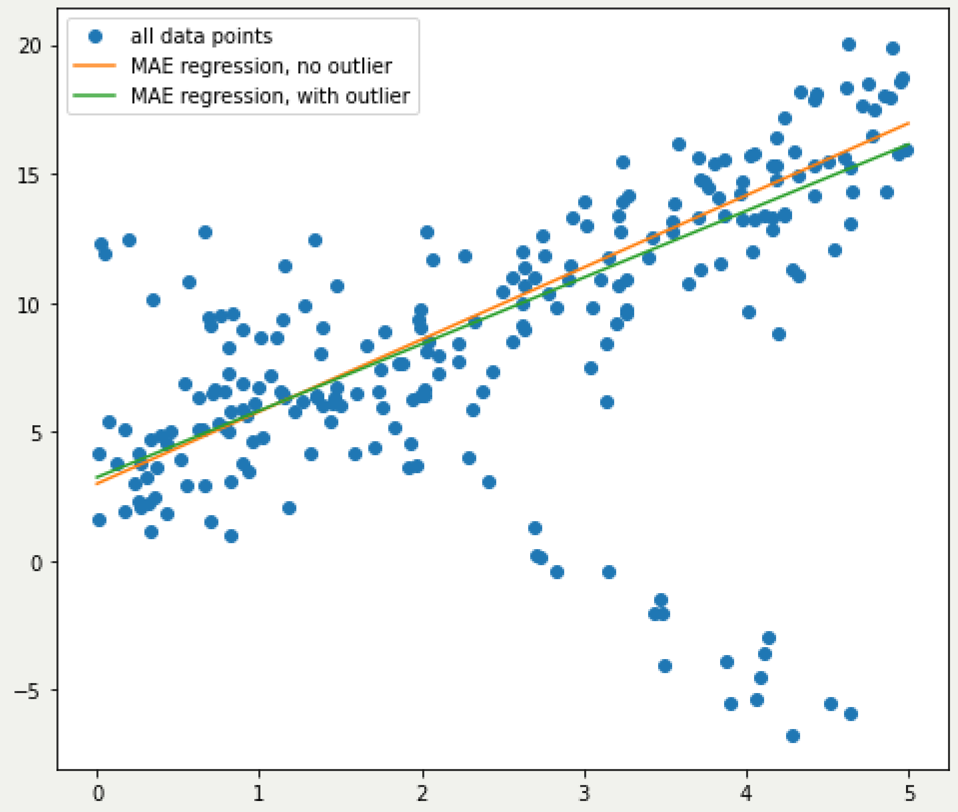
```
[2.79244327 2.9804278 ]
```



In [8]:

```
def error(c):
    return np.sqrt(np.abs(F(x_c,c) - y_c))

sol_mae_2 = least_squares(error,[0.0,0.0]).x
print(sol_mae_2)
plt.figure(figsize=(8,7))
plt.plot(x_c,y_c,"o",
         RANGE,F(RANGE,sol_mae),"-",
         RANGE,F(RANGE,sol_mae_2),"-");
plt.legend(["all data points",
            "MAE regression, no outlier",
            "MAE regression, with outlier"]);
```

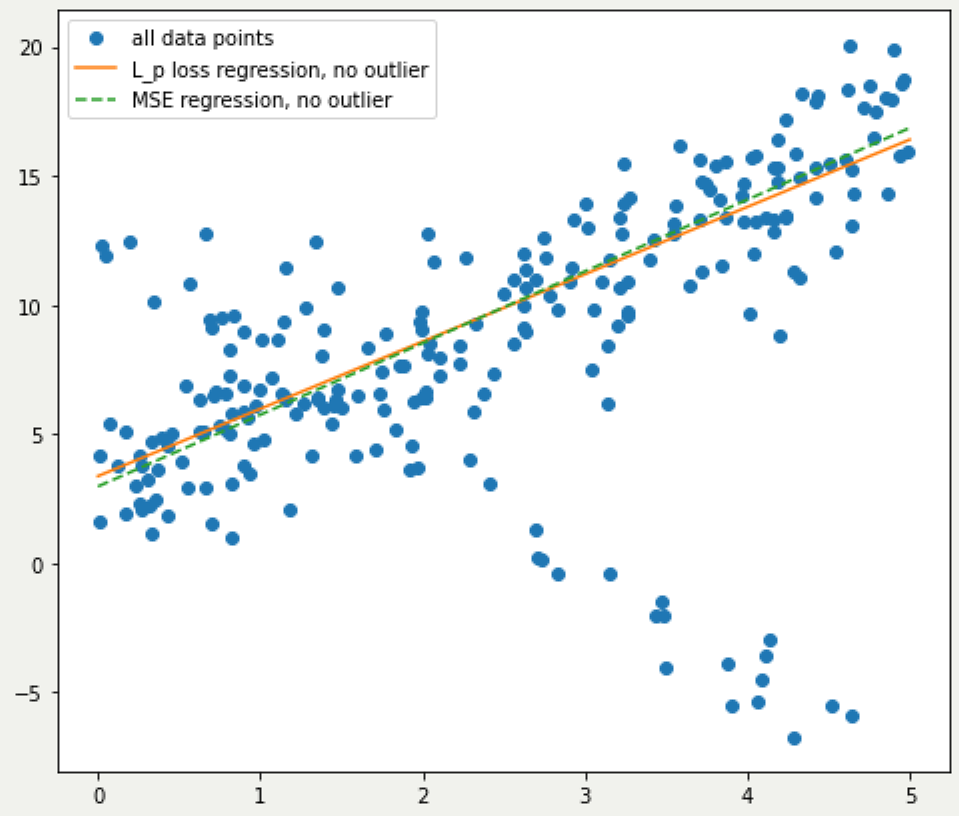
```
[2.58027493 3.22709177]
```



Linear Regression: higher powers of absolute error (L_p loss)

In [9]:
```
p = 4
def error(c):
    return np.abs(F(x,c) - y)**(p/2.0)

sol_high = least_squares(error,[0.0,0.0]).x
print(sol_high)
plt.figure(figsize=(8,7))
plt.plot(x_c,y_c,"o", RANGE,F(RANGE,sol_high),"-",
         RANGE,F(RANGE,sol_mse),"--")
plt.legend(["all data points",
            "L_p loss regression, no outlier",
            "MSE regression, no outlier"]);
```

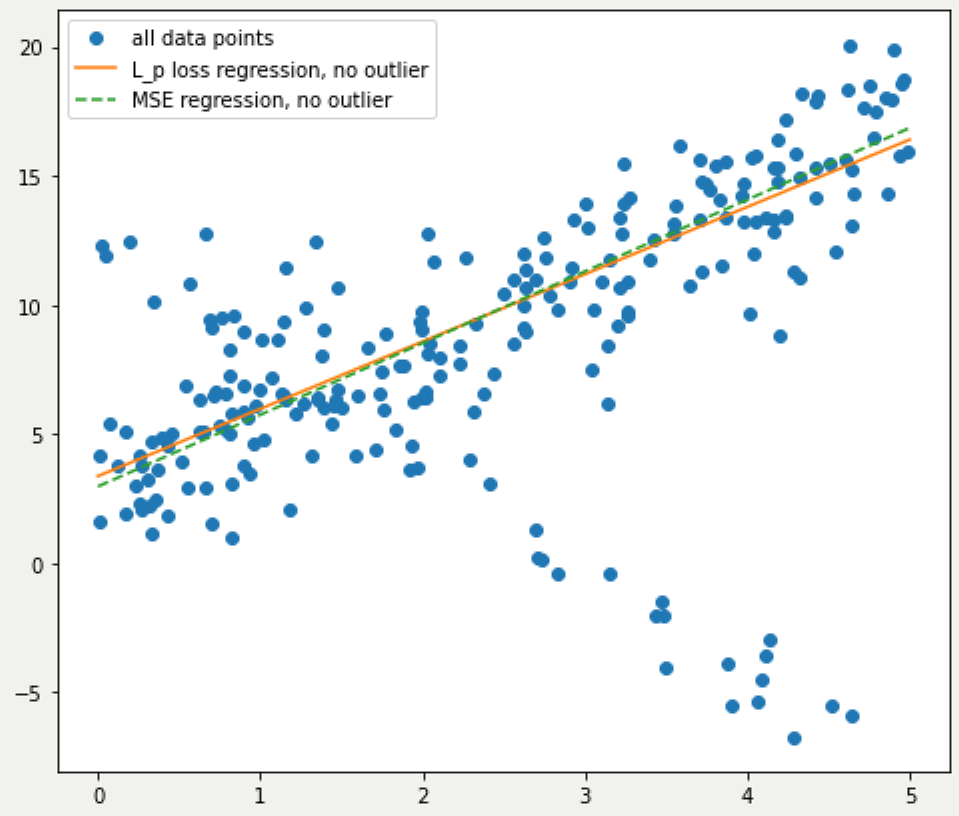
```
[2.60627088 3.36281537]
```



Linear Regression: higher powers of absolute error (L_p loss)

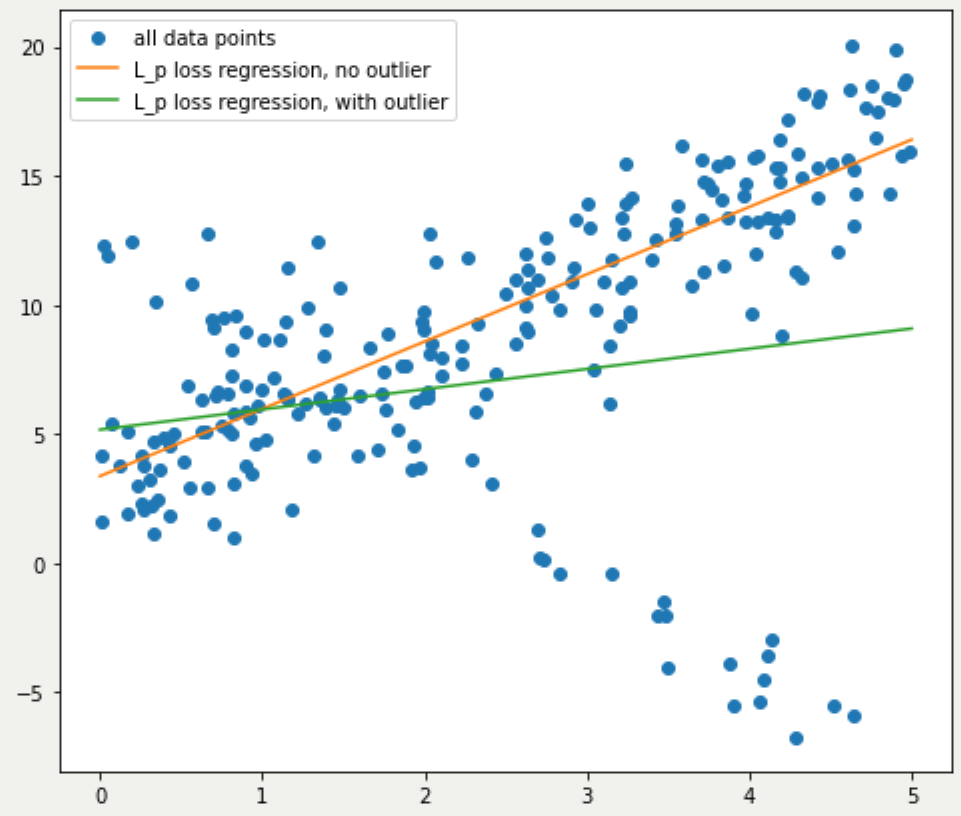
In [9]:
```
p = 4
def error(c):
    return np.abs(F(x,c) - y)**(p/2.0)

sol_high = least_squares(error,[0.0,0.0]).x
print(sol_high)
plt.figure(figsize=(8,7))
plt.plot(x_c,y_c,"o", RANGE,F(RANGE,sol_high),"-",
         RANGE,F(RANGE,sol_mse),"--")
plt.legend(["all data points",
            "L_p loss regression, no outlier",
            "MSE regression, no outlier"]);
```

[2.60627088 3.36281537]



In [10]:
```
def error(c):
    return np.abs(F(x_c,c) - y_c)**(p/2.0)

sol_high_2 = least_squares(error,[0.0,0.0]).x
print(sol_high_2)
plt.figure(figsize=(8,7))
plt.plot(x_c,y_c,"o",
         RANGE,F(RANGE,sol_high),"-",
         RANGE,F(RANGE,sol_high_2),"-")
plt.legend(["all data points",
            "L_p loss regression, no outlier",
            "L_p loss regression, with outlier"]);
```

[0.78329347 5.16268326]



In the limit...

- This converges to minimizing the maximum difference between $(f(x_i) = c_0 x_i + c_1$ and $y_i)$
- This is called: L_∞ loss
- **Digression:** how to find the best solution under L_∞ loss?
- Minimize: $\max_i \{|c_0 x_i + c_1 - y_i|\}$
    - c_0 and c_1 are variables
    - x_i's and y_i's are constants

In the limit...

- This converges to minimizing the maximum difference between $(f(x_i) = c_0 x_i + c_1 \text{ and } y_i)$
- This is called: L_∞ loss
- **Digression:** how to find the best solution under L_∞ loss?
- Minimize: $\max_i \{|c_0 x_i + c_1 - y_i|\}$
  - c_0 and c_1 are variables
  - x_i's and y_i's are constants
- **Another way to express it:** minimize z such that

$$|c_0 x_i + c_1 - y_i| \leq z \qquad \text{for all } i$$

In the limit...

- **Another way to express it:** minimize z such that

$$|c_0 x_i + c_1 - y_i| \leq z \qquad \text{for all } i$$

- **Linear programming formulation:** minimize z such that

$$c_0 x_i + c_1 - y_i \leq z \qquad \text{for all } i$$

and

$$-(c_0 x_i + c_1 - y_i) \leq z \qquad \text{for all } i$$

1D insight into the outlier sensitivity

- **Input:** set of points in $\mathbb{R}$
- What point minimizes MSE, MAE, ... as a representative of these points?

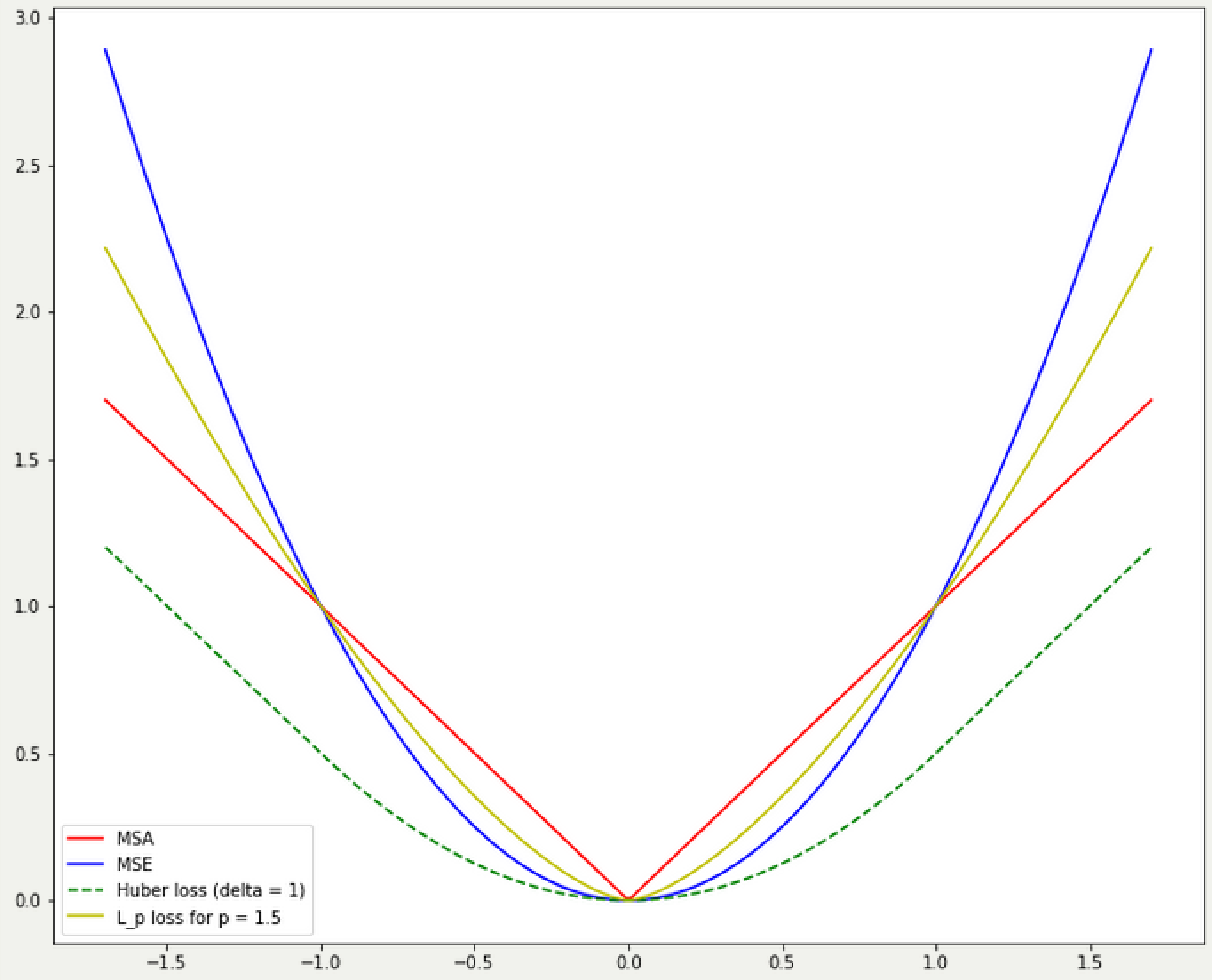
- MSE (L_2): mean
- MAE (L_1): median
- L_∞: the mean of the maximum and minimum

Something in between MSE and MAE?

- L_p loss for $p \in (1, 2)$?
- Huber loss:
  - quadratic for small distances
  - linear for large distances

$$L_\delta(f(x), y) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{if } |y - f(x)| \leq \delta \\ \delta|y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise} \end{cases}$$

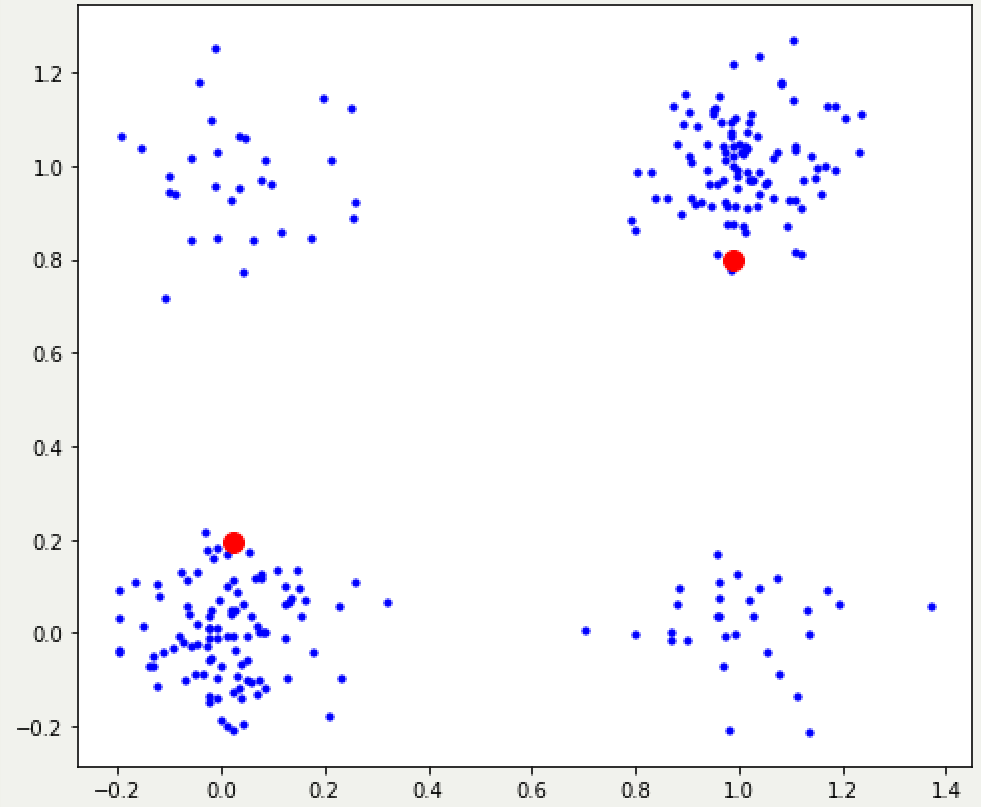
In [11]:
```
def huber(z, delta=1.0):
    return z**2/2 if abs(z) < delta \
    else delta*abs(z) - delta**2/2

xs = np.linspace(-1.7,1.7,num=250)
plt.figure(figsize=(12,10))
plt.plot(xs,np.abs(xs),"r-",xs,xs**2,"b-",
         xs,[huber(x) for x in xs],"g--",
         xs,np.abs(xs)**1.5,"y")
plt.legend(["MSA","MSE","Huber loss (delta = 1)",
           "L_p loss for p = 1.5"]);
```



Different context: Clustering

- k–means: uses MSE as the cost of a cluster (= sum of squared distances to the cluster center)

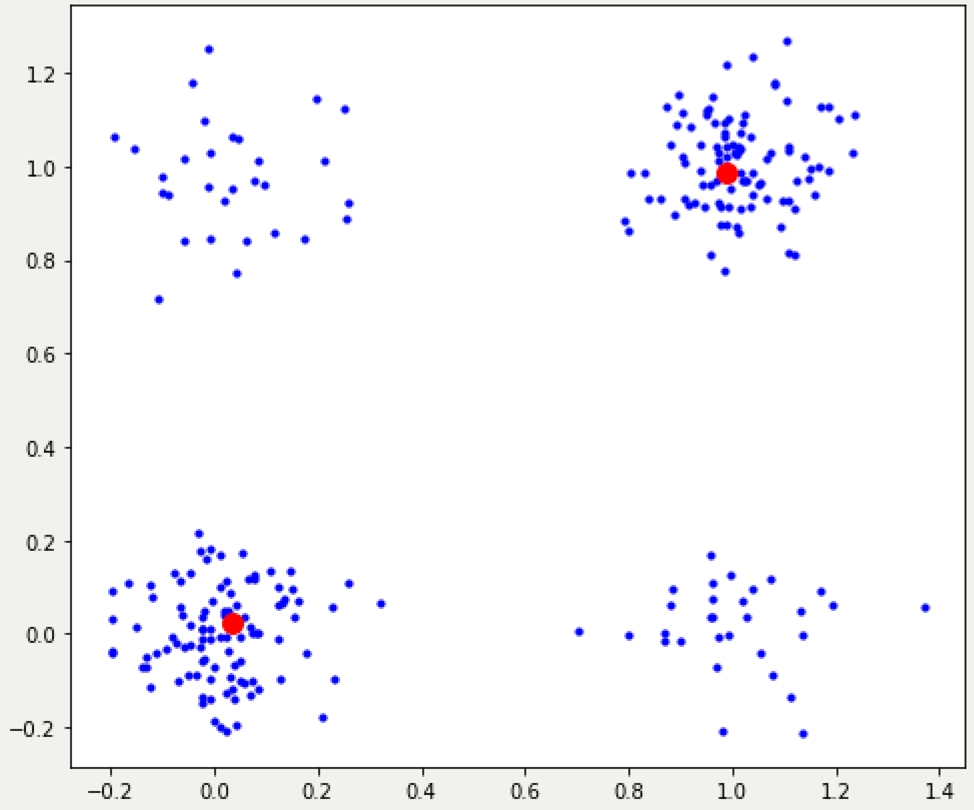
```
In [12]: import clustering_examples
         clustering_examples.kmeans()
```



Different context: Clustering

- k–median: uses MAE as the cost of a cluster (= sum of distances to the center cluster)

In [13]:
```
clustering_examples.kmedian()
```



Different context: Clustering

- k–center: uses maximum distance of any point to the closest cluster center (L_∞ loss like behavior)

In [14]:
```
clustering_examples.kcenter()
```