In [1]:
```julia
using Plots
using Random
using Statistics
using LinearAlgebra
using Distances
```

Q1 (Chelsea'srestaurant)Chelsea is a Michelin chef and he has a restaurant located at the city of Taipei. To run her Michelin restaurant, she needs 100 magic ingredients. For simplicity, her names this 100 ingredients from 1 to 100. Each ingredient corresponds to several properties, including price, quality (freshness) level, and nutrition level. And there are 100 customers in this market, with their location recorded in latitude and longitude on a 2-dimensional grid. The latitude and longitude of the customers are in the following array respectively.

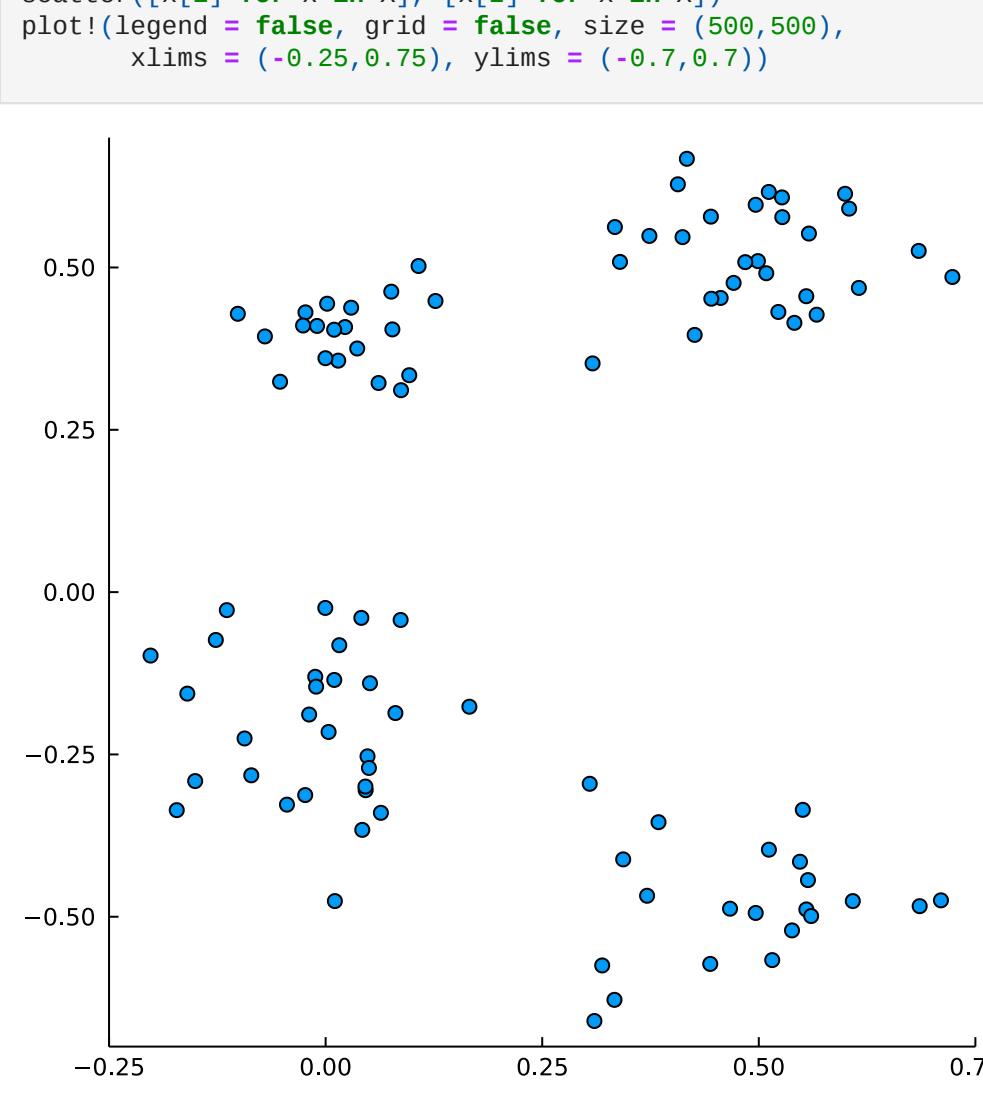
Chelsea's restaurant locates at (latitude, longitude) = (0.2, 0). (red dot/trace 4)

In [2]:
```julia
function kmeans(x, k; maxiters = 100, tol = 1e-5)
        N = length(x) #number point of cluster
        n = length(x[1]) # the dimention
        distances = zeros(N)  # used to store the distance of each
        reps = [zeros(n) for j=1:k]  # used to store representatives.
        # 'assignment' is an array of N integers between 1 and k.
        assignment = [ rand(1:k) for i in 1:N ] # The initial assignment is chosen randomly.
        Jprevious = Inf  # used in stopping condition
        for iter = 1:maxiters
        # Cluster j representative is average of points in cluster j.
                for j = 1:k
        group = [i for i=1:N if assignment[i] == j]
        reps[j] = sum(x[group]) / length(group);
    end;
    # For each x[i], find distance to the nearest representative
        # and its group index.
        for i=1:N
        (distances[i], assignment[i]) =
        findmin([norm(x[i] - reps[j]) for j = 1:k])
        end;
        # Compute clustering objective.
        J = norm(distances)^2 / N
        # Show progress and terminate if J stopped decreasing.
        println("Iteration ", iter, ": Jclust = ", J, ".")
        if iter>1&&abs(J-Jprevious)<tol*J
        return assignment, reps
        end
        Jprevious = J
        end
end
```

Out[2]: kmeans (generic function with 1 method)

In [3]:
```julia
Random.seed!(80)
X = vcat( [ [0, -0.2] + 0.1*randn(2) for i = 1:30 ],
                [ [0.5, 0.5] + 0.1*randn(2) for i = 1:30 ],
                [ [0.5, -0.5] + 0.1*randn(2) for i = 1:20 ],
                [ [0, 0.4] + 0.06* randn(2) for i = 1:20])
scatter([x[1] for x in X], [x[2] for x in X])
plot!(legend = false, grid = false, size = (500,500),
        xlims = (-0.25,0.75), ylims = (-0.7,0.7))
```

Out[3]:

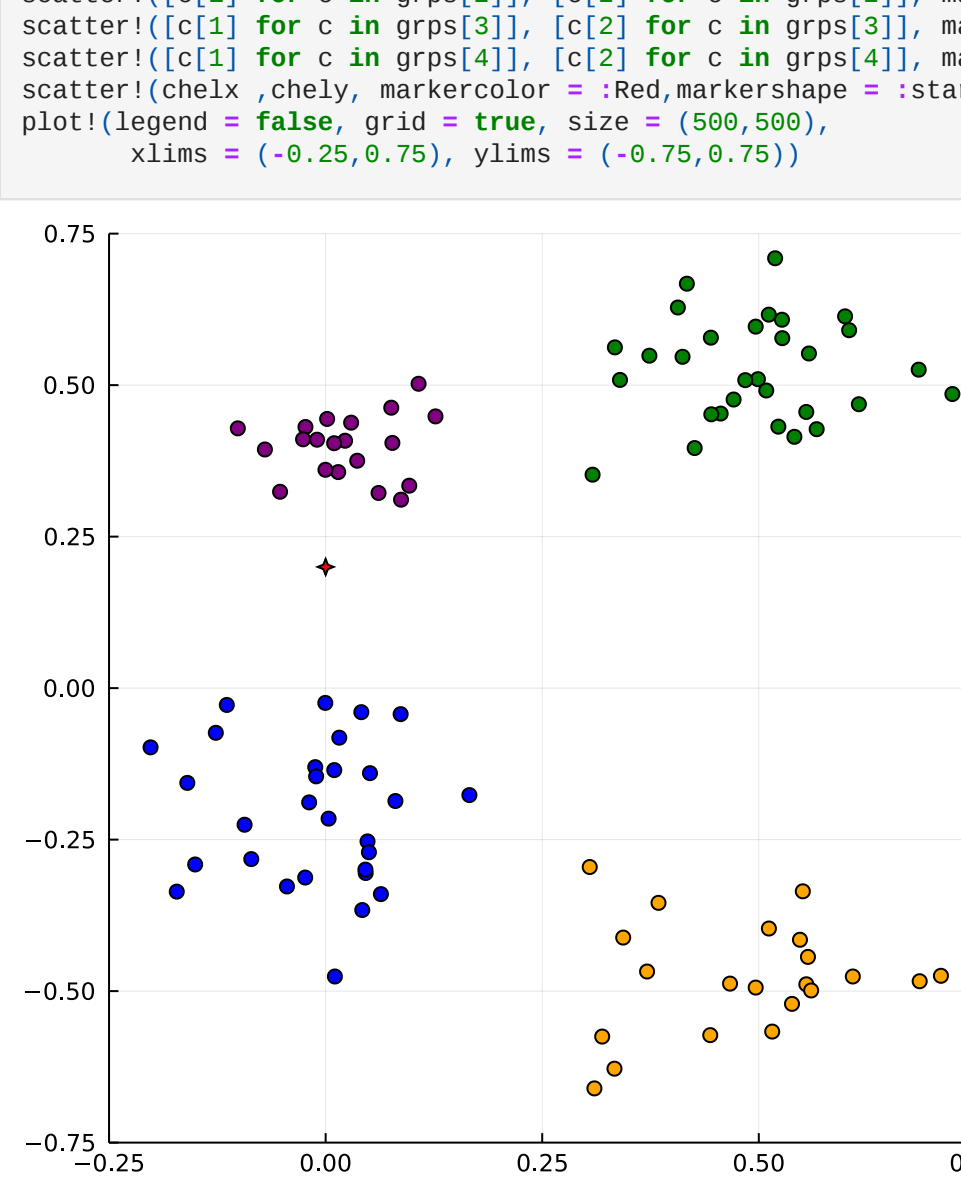


In [5]:
```julia
using LinearAlgebra
assignment, reps = kmeans(X, 4)
k = 4
N = length(X)
grps  = [[X[i] for i=1:N if assignment[i] == j] for j=1:k]
c1 = [ [0, 0.2]]
chelx = [x[1] for x in c1];
chely = [x[2] for x in c1];
```

```
Iteration 1: Jclust = 0.17042001437840845.
Iteration 2: Jclust = 0.03093274593217816.
Iteration 3: Jclust = 0.017346367918245634.
Iteration 4: Jclust = 0.01680809465434592.
Iteration 5: Jclust = 0.016747054195745868.
Iteration 6: Jclust = 0.016747054195745868.
```

In [6]:
```julia
scatter([c[1] for c in grps[1]], [c[2] for c in grps[1]], markercolor = :green,label ="cust_clus_1")
scatter!([c[1] for c in grps[2]], [c[2] for c in grps[2]], markercolor = :blue,label ="cust_clus_2")
scatter!([c[1] for c in grps[3]], [c[2] for c in grps[3]], markercolor = :orange,label ="cust_clus_3")
scatter!([c[1] for c in grps[4]], [c[2] for c in grps[4]], markercolor = :Purple,label ="cust_clus_4")
scatter!(chelx ,chely, markercolor = :Red,markershape = :star4,label ="Chelsea")
plot!(legend = false, grid = true, size = (500,500),
        xlims = (-0.25,0.75), ylims = (-0.75,0.75))
```

Out[6]:



Q2 Following question1 ,help Chelsea with the following problems.

since there is only one reps to match which is[0, 0.2] so the assignment should be 1 for the lenght of X

In [8]:
```julia
x = X
assignment = [1 for x in X] #how many of l2-norm each point to chelsea
reps = [[0, 0.2] ] #chelsea location
```

Out[8]:
```
1-element Vector{Vector{Float64}}:
 [0.0, 0.2]
```

In [9]:
```julia
#l1
Jclustl1(x,reps,assignment) =
    [cityblock(x[i],reps[assignment[i]]) for i=1:length(x)]
# %% codecell
#l2
Jclustl2(x,reps,assignment) =
    [euclidean(x[i],reps[assignment[i]]) for i=1:length(x)]
```

Out[9]: Jclustl2 (generic function with 1 method)

In [10]:
```julia
#distance l1 l2
dist_chel_l2 = Jclustl2(x,reps,assignment)
dist_chel_l1 = Jclustl1(x,reps,assignment)
```

Out[10]:
```
100-element Vector{Float64}:
 0.6037405415742244
 0.2975248195514337
 0.8000729267562832
 0.567981247309809
 0.34239099525898076
 0.4998323302857391
 0.35655561873714
 0.6084974419928058
 0.22476606590264157
 0.5012768021310703
 0.4668277251532178
 0.6865545740856729
 0.5160176029526422
 ⋮
 0.23666564134387297
 0.37522160536323756
 0.4095471420438517
 0.2639149419491269
 0.1605047590063114
 0.23067100587382036
 0.1766116819755589
 0.3385438830965444
 0.21397081078662103
 0.21171139940736902
 0.2675122059191885
 0.1831578966108126
```

In [11]:
```julia
twoa = mean(dist_chel_l2)
```

Out[11]: 0.5182495858172949

closest distance euclidean (L2) distance

In [12]:
```julia
twob = smallest_distance_l2 = findmin(dist_chel_l2)
```

Out[12]: (0.13471216806328742, 95)

In [13]:
```julia
smallest_distance_l2_argmin = argmin(dist_chel_l2)
```

Out[13]: 95

In [14]:
```julia
check_twob = euclidean([0, 0.2],X[smallest_distance_l2_argmin])
```

Out[14]: 0.13471216806328742

what is the farthest customer to Chelsea's restaurant in terms of 1-norm?

In [15]:
```julia
twoc = findmax(dist_chel_l1)
```

Out[15]: (1.3849899594784718, 77)

2d Are the 5 closest customers to Chelsea's restaurant calculated using1-norm and 2-norm the same?

In [16]:
```julia
function small_value(x,r)
        #x = dist_chel
        for i=1:r
                b = findmin(x)
                a = deleteat!(x,argmin(x))

                println(b)
        end
end
```

Out[16]: small_value (generic function with 1 method)

5 closest customers

In [17]:
```julia
l1_5_smallest = small_value(dist_chel_l1,5) #l1 5 smallest
```

```
(0.1605047590063114, 93)
(0.1710378447967606, 87)
(0.1766116819755589, 93)
(0.1831578966108126, 97)
(0.19802513956694254, 83)
```

In [18]:
```julia
l2_5_smallest = small_value(dist_chel_l2,5) #l2 5 smallest
```

```
(0.13471216806328742, 95)
(0.1364951197145926, 99)
(0.14104298008700597, 83)
(0.15718976804781376, 86)
(0.1602991540513241, 91)
```

The answer is not all is the same

Plot the predicted price on the y-axis and the true price on the x-axis. Which data point is the farthest (in terms of 2-norm) to its prediction? price! = βquantity× $quality_i$ + $\beta nutrition_i$ + v The parameters are βquantity'(= 0.5, $\beta_{nutrition}$ = 7.1, and v = 0.1.)

In [19]:
```julia
Random.seed!(8)
y = randn(100)
Random.seed!(8)
quality = y * 0.5 + rand(100) * 0.1
Random.seed!(10)
nutrition = y * 0.1 + rand(100) * 0.2
price = quality * 1 + nutrition * 0.1 + rand(100) * 2
```

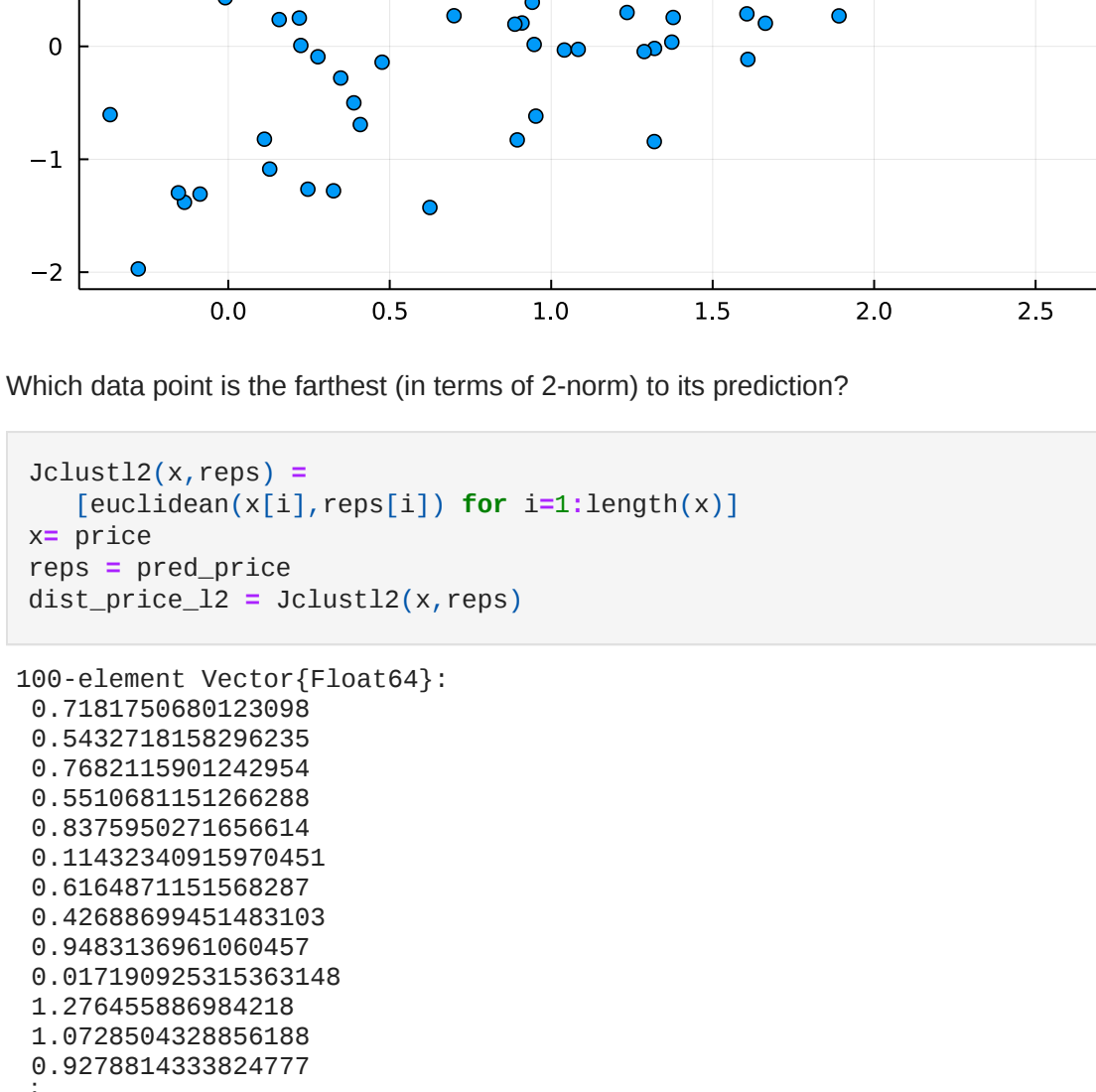
Out[19]:
```
100-element Vector{Float64}:
 -0.000851516838157651
  1.6235845696603337
  2.5846052605028182
  0.9414274099496751
  0.8461935074630715
  2.187696877681609
  0.47637236397694386
  1.408735754714744
 -0.21237690962130745
  0.8003412498506579
  0.40604368813399955
  1.0406850459297647
  1.769537404691874
  ⋮
  0.026392846092910094
 -0.2788050730559315
  1.585093458851335
  2.8320382058495976
  0.7505145088279714
  1.3200552724372794
  0.4790338793258766
  1.2879753074206888
  0.9524932549190859
  0.8720287887882157
  1.471984940256487
  0.29868172574755997
```

In [20]:
```julia
bq = 0.5
bn = 7.1
v = 0.1
pred_price = (bq * quality) + (bn * nutrition) .+ v
```

Out[20]:
```
100-element Vector{Float64}:
  0.7173235511741521
  1.0803127538307102
  1.8163936703785228
  0.3903592948230463
  1.683788534628733
  2.0733734685219045
 -0.14011481117988492
  1.8356227492295751
  0.7359367864847383
  0.8175321751660211
  1.6824995751182175
 -0.032165386955854136
  0.846559713167097
  ⋮
  0.7777161645214463
 -1.9713857711282077
  1.5767146273409622
  2.5791730763634653
  0.7259812619780989
 -0.01819861318150806
  0.9312579681140158
 -0.04587937449036636
 -0.6178279675056734
  1.0770174305811908
  1.1723378469696333
  0.6153200637138181
```

In [21]:
```julia
scatter(price,pred_price,label ="price")
```

Out[21]:



Which data point is the farthest (in terms of 2-norm) to its prediction?

In [22]:
```julia
Jclustl2(x,reps) =
    [euclidean(x[i],reps[i]) for i=1:length(x)]
x= price
reps = pred_price
dist_price_l2 = Jclustl2(x,reps)
```

Out[22]:
```
100-element Vector{Float64}:
 0.7181750680123098
 0.5432718158296235
 0.7682115901242954
 0.5510681151266288
 0.8375950271656614
 0.11432340915970451
 0.6164871151568287
 0.42688699451483103
 0.9483136961060457
 0.017190925315363148
 1.276455888694218
 1.0728504328856188
 0.9278814338824777
 ⋮
 0.7513233184285362
 1.6925806980722762
 0.008378831510372775
 0.2528651294861324
 0.02453324684987257
 1.3382538856187876
 0.45222408887883924
 1.3338546819111254
 1.5703212224247594
 0.20498864179297516
 0.2996470932868538
 0.3166383379662581
```

In [23]:
```julia
farthest_distance = findmax(dist_price_l2)
```

Out[23]: (2.1624181222985337, 81)

In [24]:
```julia
farthest_distance_argmax = argmax(dist_price_l2)
```

Out[24]: 81

In [25]:
```julia
farthest_distance_l2_check = euclidean(price[farthest_distance_argmax],pred_price[farthest_distance_argmax])
```

Out[25]: 2.1624181222985337

q4 What is best location to open these three(3) restaurants for Chelsea's franchisee let's assume the first location is closed

In [26]:
```julia
Random.seed!(80)
X = vcat( [ [0, -0.2] + 0.1*randn(2) for i = 1:30 ],
                [ [0.5, 0.5] + 0.1*randn(2) for i = 1:30 ],
                [ [0.5, -0.5] + 0.1*randn(2) for i = 1:20 ],
                [ [0, 0.4] + 0.06* randn(2) for i = 1:20])
assignment, reps = kmeans(X, 3)
```

```
Iteration 1: Jclust = 0.1849968268499434.
Iteration 2: Jclust = 0.05574755188184208.
Iteration 3: Jclust = 0.04630689368265297.
Iteration 4: Jclust = 0.045491126206254284.
Iteration 5: Jclust = 0.045491126206254284.
```

Out[26]:
```
([1, 1, 2, 1, 1, 1, 1, 1, 1, 1  …  3, 3, 3, 3, 3, 3, 3, 3, 3, 3], [[-0.0156260695213798, -0.20507992562840205], [0.481622334944703, -0.4784536466599418], [0.30679090523
45134, 0.4744286672265481]])
```

In [27]:
```julia
k = 3
N = length(X)
grps1  = [[X[i] for i=1:N if assignment[i] == j] for j=1:k]
```

Out[27]:
```
3-element Vector{Vector{Vector{Float64}}}:
 [[0.06380351046867376, -0.33993703110555074], [0.015760604781896736, -0.08176421476953691], [-0.08582615487430792, -0.2821550924355011], [-0.011989920601035408, -0.130
40107465794532], [-0.20208602369583528, -0.0977463065899038], [-0.010965765979349477, -0.1455898527577905], [0.04228924611100514, -0.36620819588180054], [-0.00038477057
331151154, -0.024375888453130046], [0.048348440714758155, -0.25292836141631214], [0.08054535613113474, -0.18628236902208306]  …  [0.049969006395814475, -0.2708285458810
901], [0.009985649206610337, -0.1352644951917898], [-0.0190256898577342, -0.1885626621096125], [-0.04464726456523704, -0.3273304459502331], [-0.17185690942930454, -0.33
581097294815637], [0.041219325121722716, -0.03968204653782806], [0.04593016859766092, -0.2994721536101378], [-0.1267639515114295, -0.07370883697299038], [-0.09356469981
176908, -0.22542085777058457], [-0.0235911054282832, -0.31259519501129174]]
 [[0.30485827042442804, -0.2952146563318551], [0.31922783128942356, -0.575014150067835], [0.3842468361526979, -0.3543628854940997], [0.46681045794388276, -0.48761636953
153537], [0.34347985032169237, -0.41165567472928744], [0.5383288129712571, -0.521077035047463], [0.5548713398781417, -0.4887286197531036], [0.5604472107423055, -0.49891
472489006916], [0.5475287185686607, -0.4152579502756996], [0.6085217341841759, -0.47588127050206336]  …  [0.49643553086138376, -0.4941665480989609], [0.3102454463287344
6, -0.660547102091307], [0.6857073842711754, -0.48368906114384563], [0.5567396821015518, -0.4435526505004759], [0.5154823348161125, -0.5667789294882837], [0.44396070210
086525, -0.5726536610991075], [0.7102636601626897, -0.4747262993157822], [0.37103419381428626, -0.46766702319841774], [0.550811794188971, -0.3353122762485178], [0.51166
39671509474, -0.3967087017358324]]
 [[0.3736871791410399, 0.548591877974777], [0.518855871804593, 0.7093494399914868], [0.33977634615270713, 0.508589612183677], [0.5225657474970661, 0.4314800062739155],
[0.5086064042292293, 0.49114826978722265], [0.4989866232424618, 0.5098343588607916], [0.4259659590993544, 0.3961681444428998], [0.6155226268829705, 0.4685010675520822],
[0.6845933026657671, 0.5255248532326086], [0.33389053101828425, 0.5623020002319503]  …  [0.1072876892077482, 0.5022594528361035], [-0.0701043888554336, 0.39381055309369
33], [-0.0002057369822694941, 0.3602990220240419], [0.022422689751965343, 0.408248316121855], [-0.05258802247101238, 0.32402365950454654], [0.07576978042935423, 0.46277
410266719016], [0.00981404581334714, 0.4041567649732739], [0.036457517560725045, 0.375253881846644], [0.029443650566829373, 0.43806855535235917], [0.06110346504055548,
0.32205443157025715]]
```

What is best location to open these three restaurants for Chelsea's franchisee, i.e., where are the three clustering centres?

In [28]:
```julia
center1x = [mean([c[1] for c in grps1[1]])]
center1y = [mean([c[2] for c in grps1[1]])]
center2x = [mean([c[1] for c in grps1[2]])]
center2y = [mean([c[2] for c in grps1[2]])]
center3x = [mean([c[1] for c in grps1[3]])]
center3y = [mean([c[2] for c in grps1[3]])]
```

Out[28]:
```
1-element Vector{Float64}:
 0.47442866722654814
```

In [29]:
```julia
center1 = [center1x,center1y]
```

Out[29]:
```
2-element Vector{Vector{Float64}}:
 [-0.015626069521379796]
 [-0.2050799256284021]
```

In [30]:
```julia
center2 = [center2x,center2y]
```

Out[30]:
```
2-element Vector{Vector{Float64}}:
 [0.481622334944703]
 [-0.4784536466599417]
```

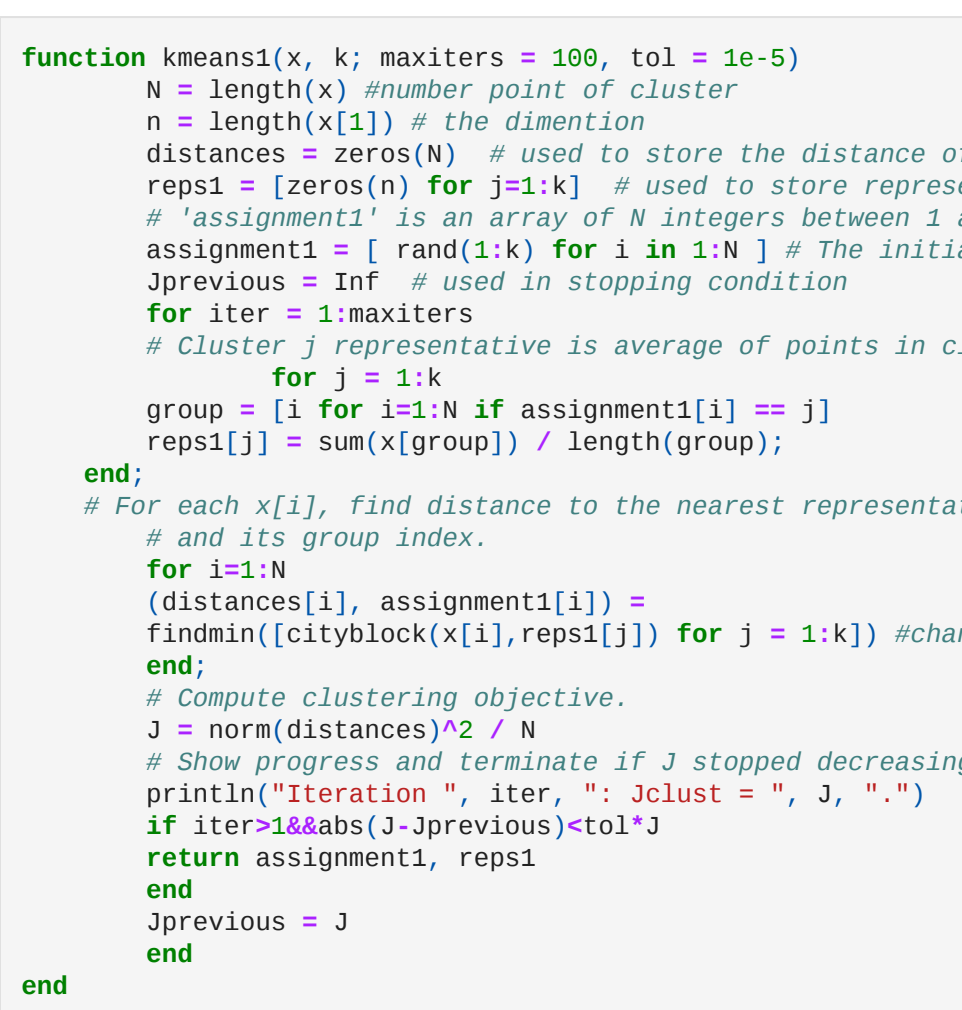
In [31]:
```julia
center3 = [center3x,center3y]
```

Out[31]:
```
2-element Vector{Vector{Float64}}:
 [0.3067909052345134]
 [0.47442866722654814]
```

Mark these customers group with different colors and mark these new restaurants on the map.

In [32]:
```julia
scatter([c[1] for c in grps[1]], [c[2] for c in grps1[1]], markercolor = :green,label ="cust_clus_1")
scatter!([c[1] for c in grps[2]], [c[2] for c in grps1[2]], markercolor = :blue,label ="cust_clus_2")
scatter!([c[1] for c in grps[3]], [c[2] for c in grps1[3]], markercolor = :orange,label ="cust_clus_3")
scatter!(center1x,center1y, markercolor = :pink,markershape = :star4, label ="center1")
scatter!(center2x,center2y, markercolor = :yellow,markershape = :star4, label ="center2")
scatter!(center3x,center3y, markercolor = :black,markershape = :star4, label ="center3")
plot!(legend = false, grid = true, size = (500,500),
        xlims = (-0.25,0.75), ylims = (-0.75,0.75))
```

Out[32]:



Use 1-norm to measure distance in the *k*-means algorithm. And plot the customers on the map. Are they the same as (b)?

In [33]:
```julia
function kmeans1(x, k; maxiters = 100, tol = 1e-5)
        N = length(x) #number point of cluster
        n = length(x[1]) # the dimention
        distances = zeros(N)  # used to store the distance of each
        reps1 = [zeros(n) for j=1:k]  # used to store representatives.
        assignment1 = [ rand(1:k) for i in 1:N ] # The initial assignment1 is chosen randomly.
        Jprevious = Inf  # used in stopping condition
        for iter = 1:maxiters
        # Cluster j representative is average of points in cluster j.
                for j = 1:k
        group = [i for i=1:N if assignment1[i] == j]
        reps1[j] = sum(x[group]) / length(group);
    end;
    # For each x[i], find distance to the nearest representative
        # and its group index.
        for i=1:N
        (distances[i], assignment1[i]) =
        findmin([cityblock(x[i],reps1[j]) for j = 1:k]) #change to l1
        end;
        # Compute clustering objective.
        J = norm(distances)^2 / N
        # Show progress and terminate if J stopped decreasing.
        println("Iteration ", iter, ": Jclust = ", J, ".")
        if iter>1&&abs(J-Jprevious)<tol*J
        return assignment1, reps1
        end
        Jprevious = J
        end
end
```

Out[33]: kmeans1 (generic function with 1 method)

In [38]:
```julia
assignment1, reps1 = kmeans1(X, 3)
k = 3
N = length(X)
#idk why sometimes it error and sometime it dont
```

```
Iteration 1: Jclust = 0.33839061037078016.
MethodError: no method matching zero(::Type{Vector{Float64}})
Closest candidates are:
  zero(::Union{Type{P}, P}) where P<:Dates.Period at /Users/sabae/src/julia/usr/share/julia/stdlib/v1.7/Dates/src/periods.jl:53
  zero(::AbstractIrrational) at irrationals.jl:150
  zero(::T) where T<:Dates.TimeType at /Users/sabae/src/julia/usr/share/julia/stdlib/v1.7/Dates/src/types.jl:450
  ...

Stacktrace:
  [1] reduce_empty(#unused#::typeof(+), #unused#::Type{Vector{Float64}})
    @ Base ./reduce.jl:313
  [2] reduce_empty(#unused#::typeof(Base.add_sum), #unused#::Type{Vector{Float64}})
    @ Base ./reduce.jl:322
  [3] mapreduce_empty(#unused#::typeof(identity), op::Function, T::Type)
    @ Base ./reduce.jl:345
  [4] reduce_empty(op::Base.MappingRF{typeof(identity), typeof(Base.add_sum)}, #unused#::Type{Vector{Float64}})
    @ Base ./reduce.jl:331
  [5] reduce_empty_iter
    @ ./reduce.jl:357 [inlined]
  [6] mapreduce_empty_iter(f::Function, op::Function, itr::Vector{Vector{Float64}}, ItrEltype::Base.HasEltype)
    @ Base ./reduce.jl:353
  [7] _mapreduce(f::typeof(identity), op::typeof(Base.add_sum), #unused#::IndexLinear, A::Vector{Vector{Float64}})
    @ Base ./reduce.jl:402
  [8] _mapreduce_dim
    @ ./reducedim.jl:330 [inlined]
  [9] #mapreduce#725
    @ ./reducedim.jl:322 [inlined]
 [10] mapreduce
    @ ./reducedim.jl:322 [inlined]
 [11] #_sum#735
    @ ./reducedim.jl:894 [inlined]
 [12] _sum
    @ ./reducedim.jl:894 [inlined]
 [13] #_sum#734
    @ ./reducedim.jl:893 [inlined]
 [14] _sum
    @ ./reducedim.jl:893 [inlined]
 [15] #sum#732
    @ ./reducedim.jl:889 [inlined]
 [16] sum
    @ ./reducedim.jl:889 [inlined]
 [17] kmeans1(x::Vector{Vector{Float64}}, k::Int64; maxiters::Int64, tol::Float64)
    @ Main ./In[33]:13
 [18] kmeans1(x::Vector{Vector{Float64}}, k::Int64)
    @ Main ./In[33]:2
 [19] top-level scope
    @ In[38]:1
 [20] eval
    @ ./boot.jl:373 [inlined]
 [21] include_string(mapexpr::typeof(REPL.softscope), mod::Module, code::String, filename::String)
    @ Base ./loading.jl:1196
```

2D Do you think 3 is a good clustering number? Why?

No, because there is 4 cluster there so it wouldn't be equal to three of them

(Ida the mathematician) Write a function Gram_Schmidt that takes the input of an array containing vectors. The function returns the orthonormal set of vectors when the input array contains vectors that are all linearly independent; otherwise, return "Vectors are linearly dependent.".

In [39]:
```julia
function gram_schmidt(a; tol = 1e-10)
        q=[]
        for i = 1:length(a)
        qtilde = a[i]
                for j = 1:i-1
                qtilde -= (q[j]'*a[i]) * q[j]
                end
    if norm(qtilde) < tol
        println("Vectors are linearly dependent.")
        return q
        end
    push!(q, qtilde/norm(qtilde))
        end;
        return q
end
```

Out[39]: gram_schmidt (generic function with 1 method)

In [40]:
```julia
a = [ [-1, 1, -1, 1], [-1, 3, -1, 3], [1, 3, 5, 7] ]
q = gram_schmidt(a)
```

Out[40]:
```
3-element Vector{Any}:
 [-0.5, 0.5, -0.5, 0.5]
 [0.5, 0.5, 0.5, 0.5]
 [-0.5, -0.5, 0.5, 0.5]
```

In [41]:
```julia
# replace a3 with a linear combination of a1 and a2 it become linearly dependent
b = [ a[1], a[2], 1.3*a[1] + 0.5*a[2] ]
q = gram_schmidt(b)
```

```
Vectors are linearly dependent.
```

Out[41]:
```
2-element Vector{Any}:
 [-0.5, 0.5, -0.5, 0.5]
 [0.5, 0.5, 0.5, 0.5]
```

In [42]:
```julia
three_two_vectors = [ [1,1], [1,2], [-1,1] ]
q = gram_schmidt(three_two_vectors)
```

```
Vectors are linearly dependent.
```

Out[42]:
```
2-element Vector{Any}:
 [0.7071067811865475, 0.7071067811865475]
 [-0.7071067811865471, 0.7071067811865478]
```