```
#In this project, we wanted to find Stephen Curry's 3 best shooting positions using his 2017 season data. We first plot the coordinate's of each of his shots then use kmeans clustering to localize regions where he takes shots. Using this, we were able to find his best shooting positions. We then made a Gaussian Mixture Model with the clustering data.

#In the second portion, we analyze Stephen Curry's Playoff Data with his seasonal data to see whether or not he performs better or worse during the Playoffs.
```

```
begin
    import Pkg
    Pkg.add("CSV")
    Pkg.add("DataFrames")
    Pkg.add("Plots")
    Pkg.add("Clustering")
    Pkg.add("Statistics")
    Pkg.add("LinearAlgebra")
    Pkg.add("GaussianMixtures")
    Pkg.add("Distributions")
    Pkg.add("FillArrays")
    Pkg.add("HypothesisTests")
end
```

```
    Updating registry at `~/.julia/registries/General`
   Resolving package versions...
  No Changes to `~/.julia/environments/v1.6/Project.toml`
  No Changes to `~/.julia/environments/v1.6/Manifest.toml`
Precompiling project...
[32m  ✓ [39mPluto
  1 dependency successfully precompiled in 13 seconds (150 already precomp
iled, 4 skipped during auto due to previous errors)
   Resolving package versions...
  No Changes to `~/.julia/environments/v1.6/Project.toml`
  No Changes to `~/.julia/environments/v1.6/Manifest.toml`
Precompiling project...
[32m  ✓ [39mPluto
  1 dependency successfully precompiled in 12 seconds (150 already precomp
iled, 4 skipped during auto due to previous errors)
   Resolving package versions...
  No Changes to `~/.julia/environments/v1.6/Project.toml`
  No Changes to `~/.julia/environments/v1.6/Manifest.toml`
Precompiling project...
[32m  ✓ [39mPluto
  1 dependency successfully precompiled in 12 seconds (150 already precomp
iled, 4 skipped during auto due to previous errors)
   Resolving package versions...
  No Changes to `~/.julia/environments/v1.6/Project.toml`
  No Changes to `~/.julia/environments/v1.6/Manifest.toml`
Precompiling project...
[32m  ✓ [39mPluto
```

```
  1 dependency successfully precompiled in 12 seconds (150 already precomp
iled, 4 skipped during auto due to previous errors)
   Resolving package versions...
  No Changes to `~/.julia/environments/v1.6/Project.toml`
  No Changes to `~/.julia/environments/v1.6/Manifest.toml`
Precompiling project...
[32m  ✓ [39mPluto
  1 dependency successfully precompiled in 12 seconds (150 already precomp
iled, 4 skipped during auto due to previous errors)
   Resolving package versions...
  No Changes to `~/.julia/environments/v1.6/Project.toml`
  No Changes to `~/.julia/environments/v1.6/Manifest.toml`
Precompiling project...
[32m  ✓ [39mPluto
  1 dependency successfully precompiled in 12 seconds (150 already precomp
iled, 4 skipped during auto due to previous errors)
   Resolving package versions...
  No Changes to `~/.julia/environments/v1.6/Project.toml`
  No Changes to `~/.julia/environments/v1.6/Manifest.toml`
Precompiling project...
[32m  ✓ [39mPluto
  1 dependency successfully precompiled in 12 seconds (150 already precomp
iled, 4 skipped during auto due to previous errors)
   Resolving package versions...
  No Changes to `~/.julia/environments/v1.6/Project.toml`
  No Changes to `~/.julia/environments/v1.6/Manifest.toml`
Precompiling project...
[32m  ✓ [39mPluto
  1 dependency successfully precompiled in 13 seconds (150 already precomp
iled, 4 skipped during auto due to previous errors)
   Resolving package versions...
  No Changes to `~/.julia/environments/v1.6/Project.toml`
  No Changes to `~/.julia/environments/v1.6/Manifest.toml`
Precompiling project...
[32m  ✓ [39mPluto
  1 dependency successfully precompiled in 12 seconds (150 already precomp
iled, 4 skipped during auto due to previous errors)
   Resolving package versions...
   Installed HypothesisTests ─ v0.10.10
   Installed Roots ─────────── v2.0.1
    Updating `~/.julia/environments/v1.6/Project.toml`
  [09f84164] + HypothesisTests v0.10.10
    Updating `~/.julia/environments/v1.6/Manifest.toml`
  [861a8166] + Combinatorics v1.0.2
  [38540f10] + CommonSolve v0.2.0
  [187b0558] + ConstructionBase v1.3.0
  [09f84164] + HypothesisTests v0.10.10
  [f2b01f46] + Roots v2.0.1
  [efcf1570] + Setfield v0.8.2
Precompiling project...
[32m  ✓ [39m[90mRoots[39m
[32m  ✓ [39mHypothesisTests
[32m  ✓ [39mPluto
  3 dependencies successfully precompiled in 12 seconds (154 already preco
mpiled, 4 skipped during auto due to previous errors)
```

```julia
begin
    using CSV
    using DataFrames
    using Plots
    using Clustering
    using Statistics
    using LinearAlgebra
    using GaussianMixtures
    import Distributions as di
    using Random
    using Distributions
    using FillArrays
    using HypothesisTests
end
```

```julia
#### stephen Curry's regular season data analysis
```

	name	team_name	game_date	season	espn_player_id
1	"Stephen Curry"	"Golden State Warriors"	2017-12-04	2017	3975
2	"Stephen Curry"	"Golden State Warriors"	2018-01-04	2017	3975
3	"Stephen Curry"	"Golden State Warriors"	2017-12-03	2017	3975
4	"Stephen Curry"	"Golden State Warriors"	2018-03-02	2017	3975
5	"Stephen Curry"	"Golden State Warriors"	2017-11-08	2017	3975
6	"Stephen Curry"	"Golden State Warriors"	2018-01-13	2017	3975
7	"Stephen Curry"	"Golden State Warriors"	2018-01-27	2017	3975
8	"Stephen Curry"	"Golden State Warriors"	2018-02-12	2017	3975
9	"Stephen Curry"	"Golden State Warriors"	2017-10-23	2017	3975
10	"Stephen Curry"	"Golden State Warriors"	2017-11-29	2017	3975
	more				
761	"Stephen Curry"	"Golden State Warriors"	2018-01-23	2017	3975

```julia
begin
    csv_reader = CSV.File("nba_savant201939.csv")
    df_reader = DataFrame(csv_reader)
end
```

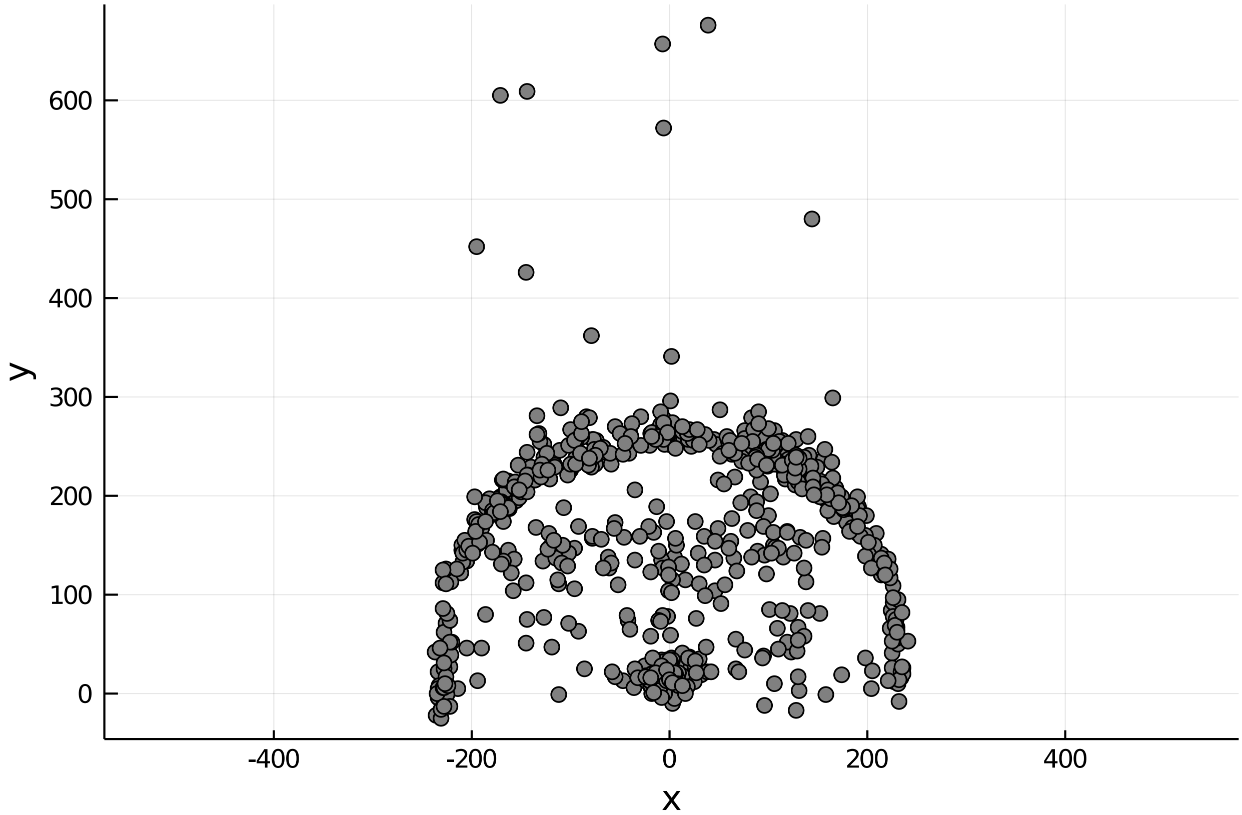
```
["name", "team_name", "game_date", "season", "espn_player_id", "team_id", "espn_game
```

```
names(df_reader)
```

df =

	name	shot_made_flag	x	y	opponent	shot_
1	"Stephen Curry"	1	7	11	"New Orleans Pelicans"	1
2	"Stephen Curry"	1	-7	21	"Houston Rockets"	2
3	"Stephen Curry"	0	-16	10	"Miami Heat"	1
4	"Stephen Curry"	1	-8	7	"Atlanta Hawks"	1
5	"Stephen Curry"	0	-5	8	"Minnesota Timberwolves"	0
6	"Stephen Curry"	1	-8	20	"Toronto Raptors"	2
7	"Stephen Curry"	1	5	11	"Boston Celtics"	1
8	"Stephen Curry"	1	6	0	"Phoenix Suns"	0
9	"Stephen Curry"	1	32	19	"Dallas Mavericks"	3
10	"Stephen Curry"	1	-1	11	"Los Angeles Lakers"	1
more						
761	"Stephen Curry"	0	-9	73	"New York Knicks"	7

```
df = df_reader[:,["name","shot_made_flag","x","y","opponent","shot_distance"]]
```

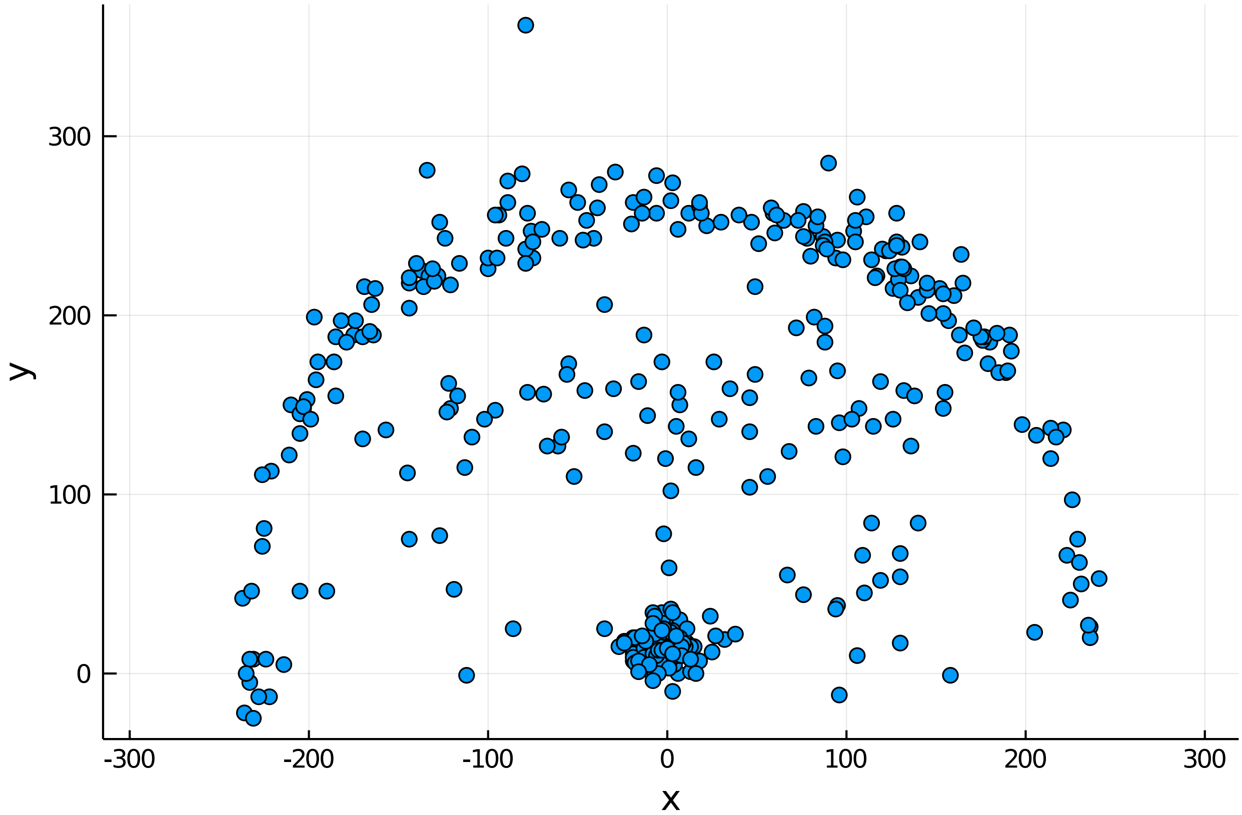



```
#plot all Curry's shot (include both 1&0)
begin
    scatter(df.x,df.y,xlabel="x",ylabel="y",color=:gray,
    label=false,aspect_ratio=:equal)
end
```

```
#Create a dataframe of Curry's all succussful shotings, named new_df
begin
    new_df = DataFrame()
    for i in 1:size(df.shot_made_flag,1)
        if df.shot_made_flag[i] !=0
            push!(new_df, df[i,:])
        end
    end
end
```

	name	shot_made_flag	x	y	opponent	shot_di
1	"Stephen Curry"	1	7	11	"New Orleans Pelicans"	1
2	"Stephen Curry"	1	-7	21	"Houston Rockets"	2
3	"Stephen Curry"	1	-8	7	"Atlanta Hawks"	1
4	"Stephen Curry"	1	-8	20	"Toronto Raptors"	2
5	"Stephen Curry"	1	5	11	"Boston Celtics"	1
6	"Stephen Curry"	1	6	0	"Phoenix Suns"	0
7	"Stephen Curry"	1	32	19	"Dallas Mavericks"	3
8	"Stephen Curry"	1	-1	11	"Los Angeles Lakers"	1
9	"Stephen Curry"	1	-19	20	"Houston Rockets"	2
10	"Stephen Curry"	1	-7	6	"Miami Heat"	0
	more					
377	"Stephen Curry"	1	-228	-13	"Memphis Grizzlies"	22

```
new_df
```




```julia
# plot the xy position of new_df
begin
    scatter(new_df.x,new_df.y,xlabel="x",ylabel="y",label=false,aspect_ratio=:equal)
end
```

```
KmeansResult{Matrix{Float64}, Float64, Int64}(
   centers = 2×8 Matrix{Float64}:
             -206.864    94.0    -0.648148  -51.0714   -6.9375  168.667  157.848
               32.0455  236.345  16.5185    256.595   143.0      45.0    172.304
   assignments =  [3, 3, 3, 3, 3, 3, 3, 3, 3,   more ,1]

   costs =  [88.9482, 60.4297, 144.652, 66.1704, 62.3556, 317.059, 1072.06, 30.5778
   counts =  [22, 55, 108, 42, 32, 24, 46, 48]
   wcounts =  [22, 55, 108, 42, 32, 24, 46, 48]
   totalcost = 671943.8197523897
   iterations = 22
   converged = true
)
```

```julia
# k mean argorithm find few centers, set 8 clusters
begin
    position_matrix = zeros(2,length(new_df.x))
    for i in 1:size(new_df.x,1)
        position_matrix[1,i] = new_df.x[i]
        position_matrix[2,i] = new_df.y[i]
    end
    n_class = 8
    Rx = kmeans(position_matrix,n_class)
end
```

```
2×8 Matrix{Float64}:
 -206.864    94.0    -0.648148  -51.0714   -6.9375  168.667  157.848  -158.146
   32.0455  236.345  16.5185    256.595   143.0      45.0    172.304   178.896
```

```julia
Rx.centers
```

```
second_largest (generic function with 1 method)
```

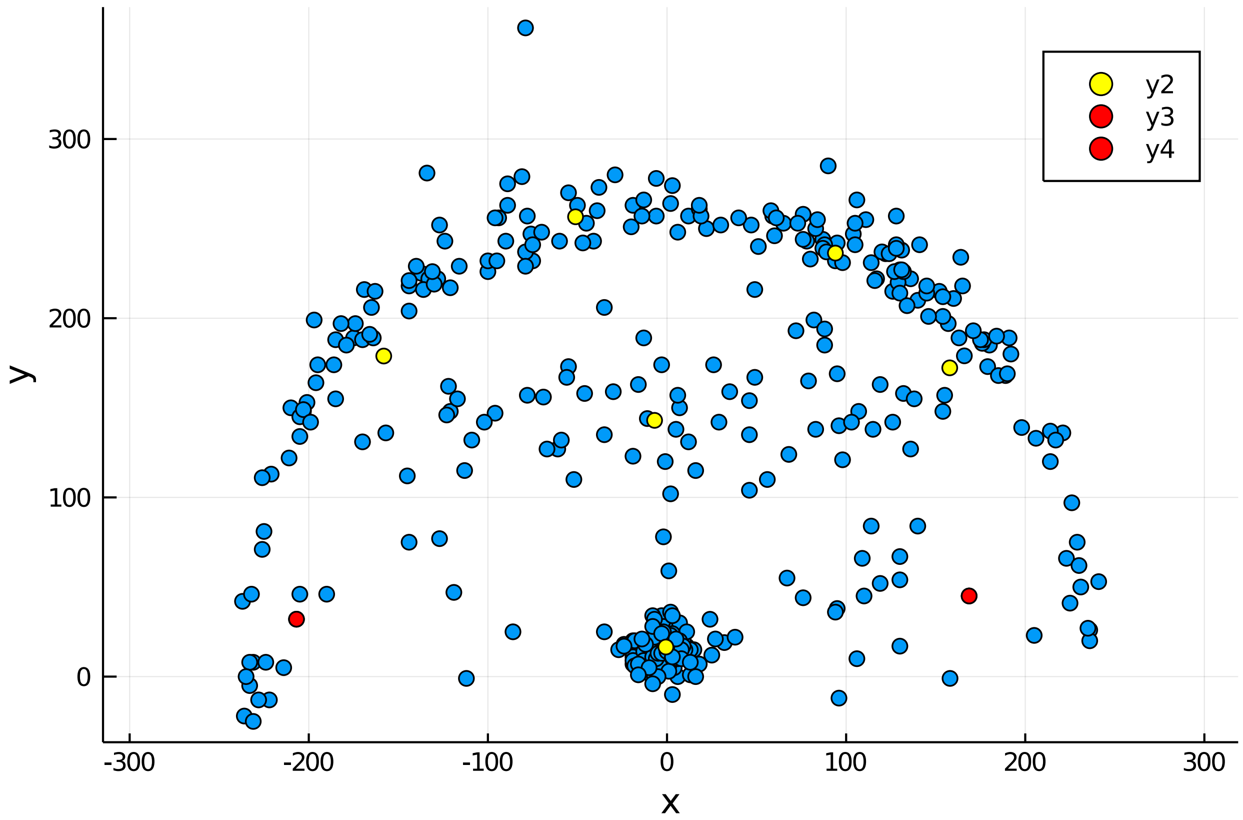
```julia
#find 2 clusters which contains the minimum two # of sucessful shotings
begin
    function second_largest(numbers)
        m1, m2 = 10000,10000
        for x in numbers
            if x <= m1
                m1, m2 = x, m1
            elseif x < m2
                m2 = x
            end
        end
        return m1,m2
    end
end
```

```
mins =  (22, 24)
```

```julia
#2 minimum # of counts

mins = second_largest(Rx.counts)
```

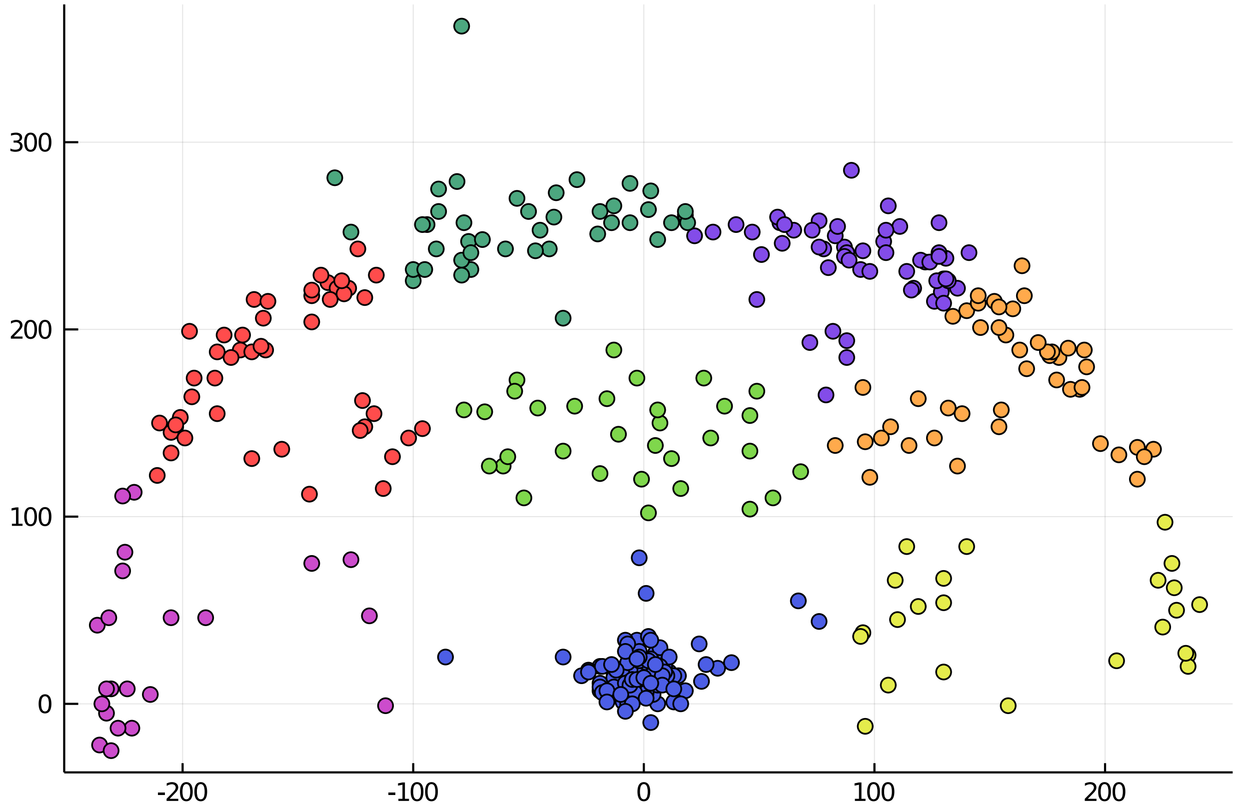
6

```
begin
    #get the index of the two cluster
    idx_1 = findfirst(Rx.counts .== mins[1])
    idx_2 = findfirst(Rx.counts .== mins[2])
end
```




```
# plot the position and kmean centers (red are the two cluster with minimum # of successfull shots)
begin
    scatter(new_df.x,new_df.y,xlabel="x",ylabel="y",label=false,aspect_ratio=:equal)
    scatter!(Rx.centers[1,:], Rx.centers[2,:],color=:yellow) #plot k mean centers
    scatter!([Rx.centers[1,idx_1]], [Rx.centers[2,idx_1]],color=:red)
    scatter!([Rx.centers[1,idx_2]], [Rx.centers[2,idx_2]],color=:red)
end
```

```
# pick out the dots corresponding each centers
```




```
#use different color to show the different cluster in scatter plot
scatter(new_df.x, new_df.y, marker_z=Rx.assignments, color=:lightrainbow,
legend=false)
```

RxAssSet = [1, 5, 7, 6, 4, 2, 8, 3]

```
RxAssSet = [x for x in Set(Rx.assignments)]
```

create_empty (generic function with 1 method)

```
begin
function create_empty(n_class)
    pindex = []
    for i in 1:n_class
        new_index = []
        push!(pindex,new_index)
    end
    return pindex
end
end
```

get_pindex (generic function with 1 method)

```
#got index of points in different classes
begin
    function get_pindex(pindex)
    for j in 1:n_class
        for i in 1:size(Rx.assignments,1)
            if Rx.assignments[i] == RxAssSet[j]
                push!(pindex[j],i)
            end
        end
    end
        return pindex
    end
end
```

```
#pick out points corresponding to each cluster
```

cluster_separate (generic function with 1 method)

```
begin
    function cluster_separate(pindex,cluster,data,RxAssSet)
    for i in 1:size(pindex,1)
        for j in pindex[i]
            # push!(new_p,j)
            push!(cluster[RxAssSet[i]], data[j])

        end
        #push!(cluster[i],new_p)
    end
        return cluster
    end
end
```

check_cluster (generic function with 1 method)

```
begin
    function check_cluster(cluster,total_num)
        count = 0
        for i in 1:size(cluster,1)
            count = length(cluster[i]) + count
        end
        if count == total_num

            print("cluster correct and match!")
            return count
        else
            print("incorrect!!! ")
            return 0
        end
    end
end
```

```
[[-226, -190, -237, -112, -231, -214, -205, -236, -221,   more  ,-228], [76, 78, 130,
```

```
begin
    pindex_1 = create_empty(n_class)
    pindex_1 = get_pindex(pindex_1)
    cluster_x = create_empty(n_class)
    cluster_x_new=cluster_separate(pindex_1,cluster_x,new_df.x,RxAssSet)
end
```

```
[[71, 46, 42, -1, 8, 5, 46, -22, 113,   more  ,-13], [258, 243, 227, 257, 250, 215, 24
```

```
begin
    pindex_2 = create_empty(n_class)
    pindex_2 = get_pindex(pindex_2)
    cluster_y = create_empty(n_class)
    cluster_y_new=cluster_separate(pindex_2,cluster_y,new_df.y,RxAssSet)
end
```

377

```
check_cluster(cluster_x_new,length(new_df.x))
```

cluster correct and match! ?

377

```
check_cluster(cluster_y_new,length(new_df.y))
```

cluster correct and match! ?

```
# calculate covariance matrix and plot gaussian covariance elipse of each
cluster
```

cov_mat (generic function with 1 method)

```
begin
    function cov_mat(x,y)
        new_matrix = zeros(2,2)
        new_matrix[1,1] = cov(x, x)
        new_matrix[2,2] = cov(y, y)
        new_matrix[1,2] = cov(x, y)
        new_matrix[2,1] = cov(y, x)
        return new_matrix
    end

end
```

```
P = 2×2 Matrix{Float64}:
     310.623    48.1336
      48.1336  157.205
```

```julia
P= cov_mat(cluster_x_new[3], cluster_y_new[3])
```

```
SVD{Float64, Float64, Matrix{Float64}}
U factor:
2×2 Matrix{Float64}:
 -0.961002  -0.27654
 -0.27654    0.961002
singular values:
2-element Vector{Float64}:
 324.4737575970886
 143.35421055798224
Vt factor:
2×2 Matrix{Float64}:
 -0.961002  -0.27654
 -0.27654    0.961002
```

```julia
U,s,_= svd(P)
```

```
covariance_ellipse (generic function with 1 method)
```

```julia
begin
    function covariance_ellipse(P)
        U,s,_= svd(P)
        width = sqrt(s[1])
        height = sqrt(s[2])
        if height > width
            print("width must be greater than height")
        end
        return width, height
    end
end
```

```julia
#w, h = covariance_ellipse(P)
```

```
plot_ellipse (generic function with 1 method)
```

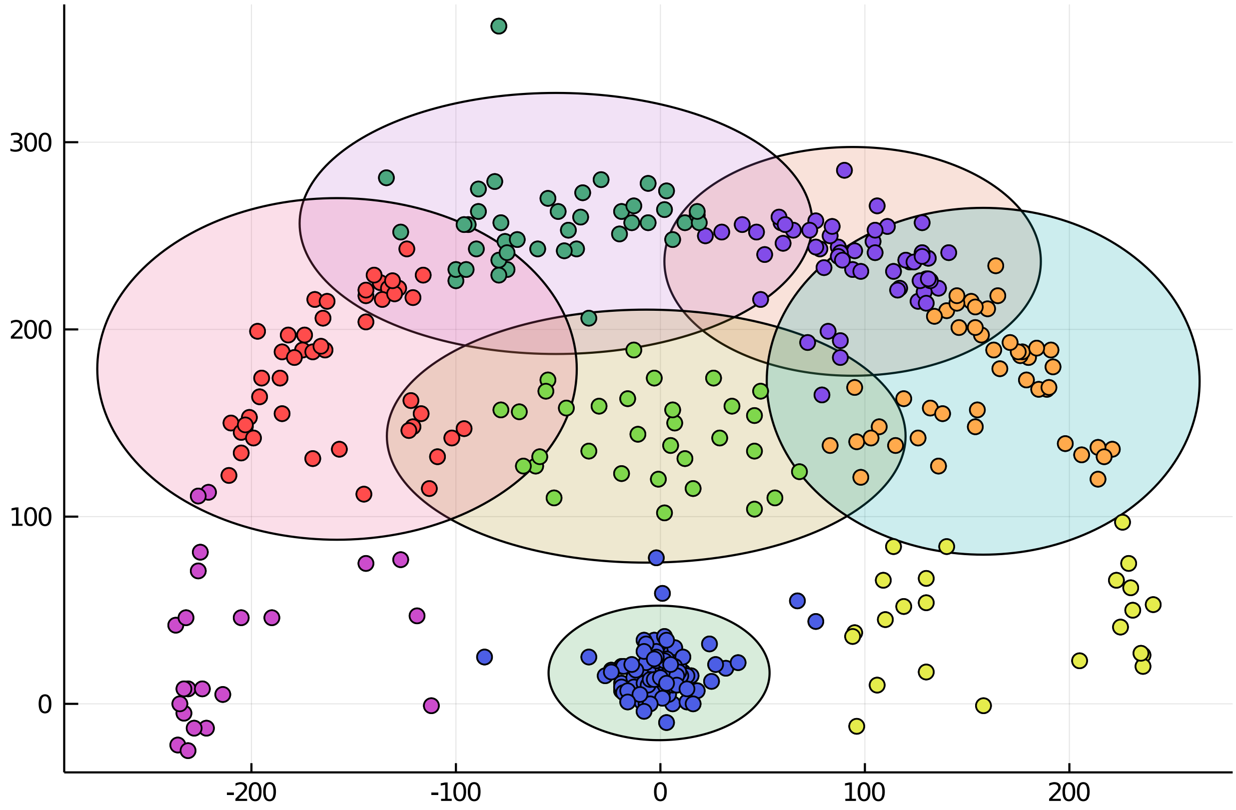
```julia
begin
    function plot_ellipse(posx, posy, w, h)
        rng = range(0, 2π, length = 221)
        ellipse(posx,posy, w, h) = Shape(w*sin.(rng).+posx, h*cos.(rng).+posy)
        elps = ellipse(posx,posy, w, h)
        plot!(elps, fillalpha = 0.2)
    end
end
```

plot_cluster_ellipse (generic function with 1 method)

```
begin
    function plot_cluster_ellipse(cluster_x_new, cluster_y_new, centers, index)
        P= cov_mat(cluster_x_new[index], cluster_y_new[index])
        w, h = covariance_ellipse(P)
        posx, posy = centers[:,index][1], centers[:,index][2]
        plot_ellipse(posx, posy, w*3, h*3)
    end
end
```

get_w_h (generic function with 1 method)

```
begin
    function get_w_h(cluster_x_new, cluster_y_new, centers, index)
        P= cov_mat(cluster_x_new[index], cluster_y_new[index])
        w, h = covariance_ellipse(P)
        return w, h
    end
end
```




```
# plot the ellipse of each cluster
begin
    for i in 1:n_class
        if i != idx_1 && i != idx_2
            plot_cluster_ellipse(cluster_x_new, cluster_y_new, Rx.centers, i)
        end
    end
    scatter!(new_df.x, new_df.y, marker_z=Rx.assignments, color=:lightrainbow,
    legend=false)
end
```

```
#calculate widths and hights of each ellipse
begin
    w_h = []
    for i in 1:n_class
            push!(w_h, get_w_h(cluster_x_new, cluster_y_new, Rx.centers, i))
    end
end
```

```julia
#calculate the number of points in the original dataframe located in each cluster to further get the weight of successful shot in each cluster
begin
    counts = []
    for j in 1:n_class
        count = 0
            for i in 1:size(df.x,1)
                if (df.x[i]-Rx.centers[1,j])^2+(df.y[i]-Rx.centers[2,j])^2 <=
                (w_h[j][1]*3)^2+(w_h[j][2]*3)^2
                    count = count+1
                end
            end

        push!(counts, count)
    end
end

```

```julia
begin
    #calculate weight of each ellipse (sucessful shot/total shots in each cluster)
    weights = []
    for i in 1:size(Rx.counts,1)
        push!(weights, Rx.counts[i]/counts[i])
    end
end
```

```julia
#find the index of 3 centers with highest weights
begin
    first = findfirst(weights .== maximum(weights))
    second = 0
    for i in 1:size(weights,1)
        if i != first
            second = findfirst(weights .== maximum(weights[i]))
        end
    end
    third = 0
    for i in 1:size(weights,1)
        if i != first && i != second
            third = findfirst(weights .== maximum(weights[i]))
        end
    end
end
```

center_1 = [-0.648148, 16.5185]

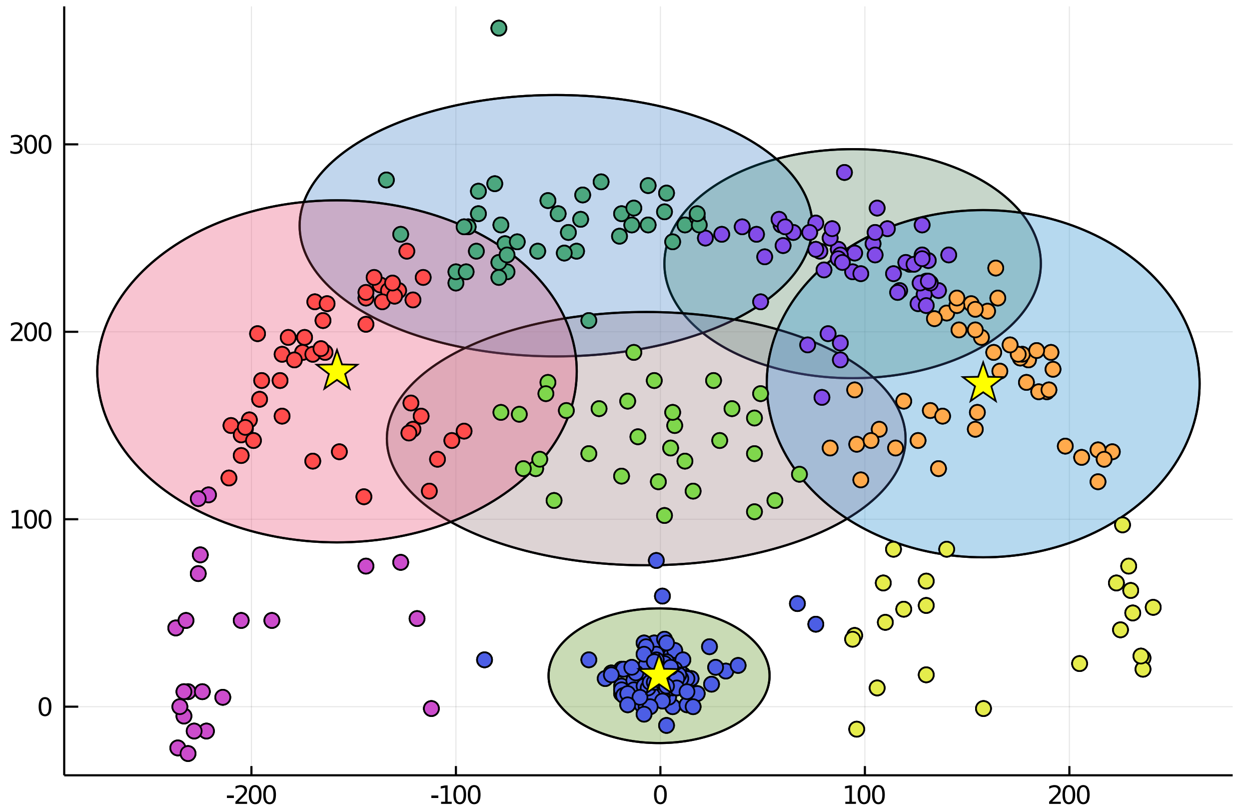
```julia
# Using the kmeans clustering data and highest sucessful shooting rate data, we plotted the 3 best points Stephen Curry has the best chance of making in a shot.
center_1 = Rx.centers[:,first]
```

center_2 = [-158.146, 178.896]

```
center_2 = Rx.centers[:,second]
```

center_3 = [157.848, 172.304]

```
center_3 = Rx.centers[:,third]
```




```
#Curry's 3 best shooting positions are denoted by the yellow star.
Unsurprisingly, one of his best shooting positions is near the basketball rim.
This is most likely because it is simply easier to score the closer you are to
the rim.

begin
    for i in 1:n_class
        if i != idx_1 && i != idx_2
            plot_cluster_ellipse(cluster_x_new, cluster_y_new, Rx.centers, i)
        end
    end
    scatter!(new_df.x, new_df.y, marker_z=Rx.assignments, color=:lightrainbow,
    legend=false)
    scatter!([center_1[1]], [center_1[2]], color = "yellow", label = "",
    markershape=:star5, markersize = 10)
    scatter!([center_2[1]], [center_2[2]], color = "yellow", label = "",
    markershape=:star5, markersize = 10)
    scatter!([center_3[1]], [center_3[2]], color = "yellow", label = "",
    markershape=:star5, markersize = 10)
end
```

```
### the gaussian mixture model construction
```

```
begin
    P_gmm = []
    for i in 1:n_class
        if i != idx_1 && i != idx_2
            push!(P_gmm, cov_mat(cluster_x_new[i], cluster_y_new[i]))
        end
    end
end
```

```
#P_gmm
```

cal_per_v1 (generic function with 1 method)

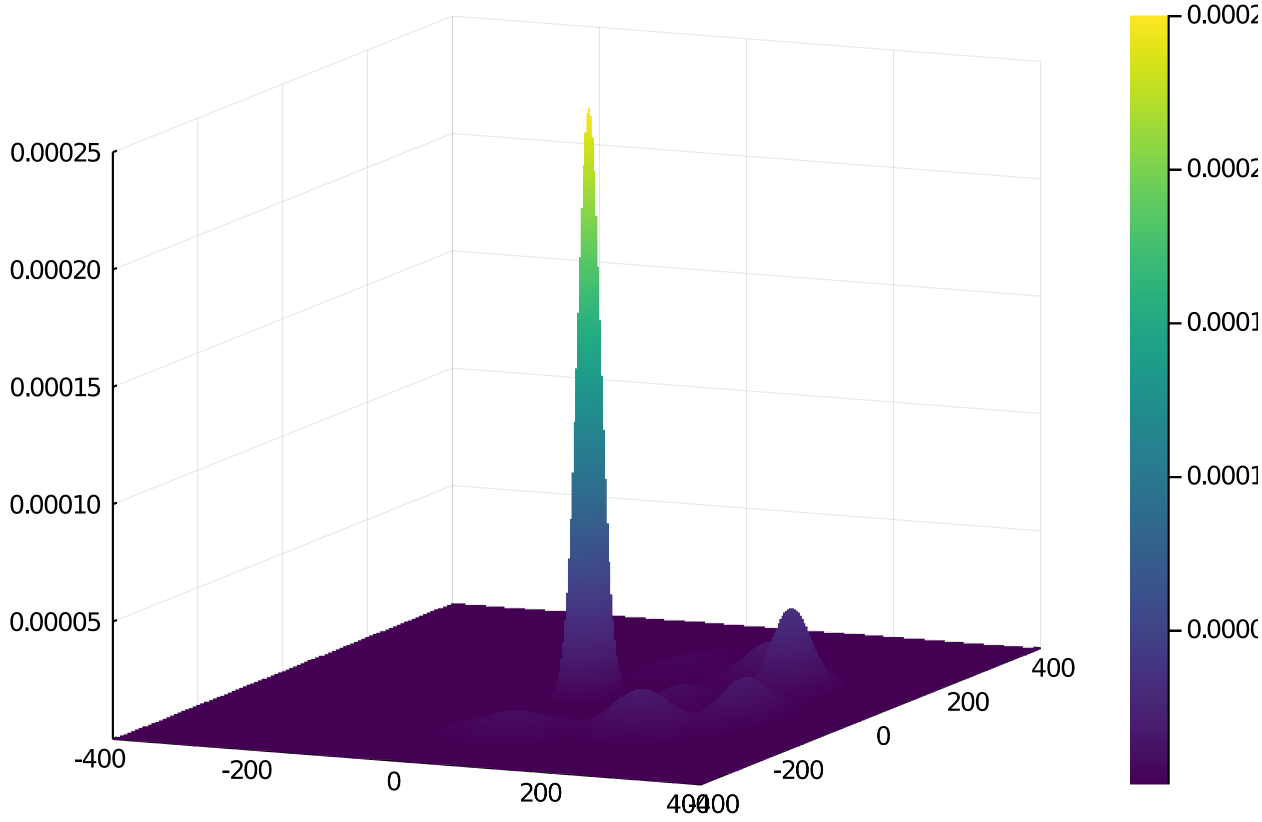
```
# calculate percentage
begin
    function cal_per_v1(counts,total)
        percentages = []
        for i in counts
            push!(percentages,i/total)
        end
        return percentages
    end
end
```

```
begin
    Centers_gmm = []
    for i in 1:n_class
        if i != idx_1 && i != idx_2
            push!(Centers_gmm, [Rx.centers[1,i], Rx.centers[2,i]])
        end
    end
end
```

```
begin
    # weights normalization (the sum of 8 weights is not 1, because there are some overlaps, so need to do normalization before apply to MixtureModel)
    weights_norm = []
    for i in 1:size(weights,1)
        weights_new = 0
        weights_new = weights[i]/sum(weights)
        push!(weights_norm, weights_new)
    end

end
```

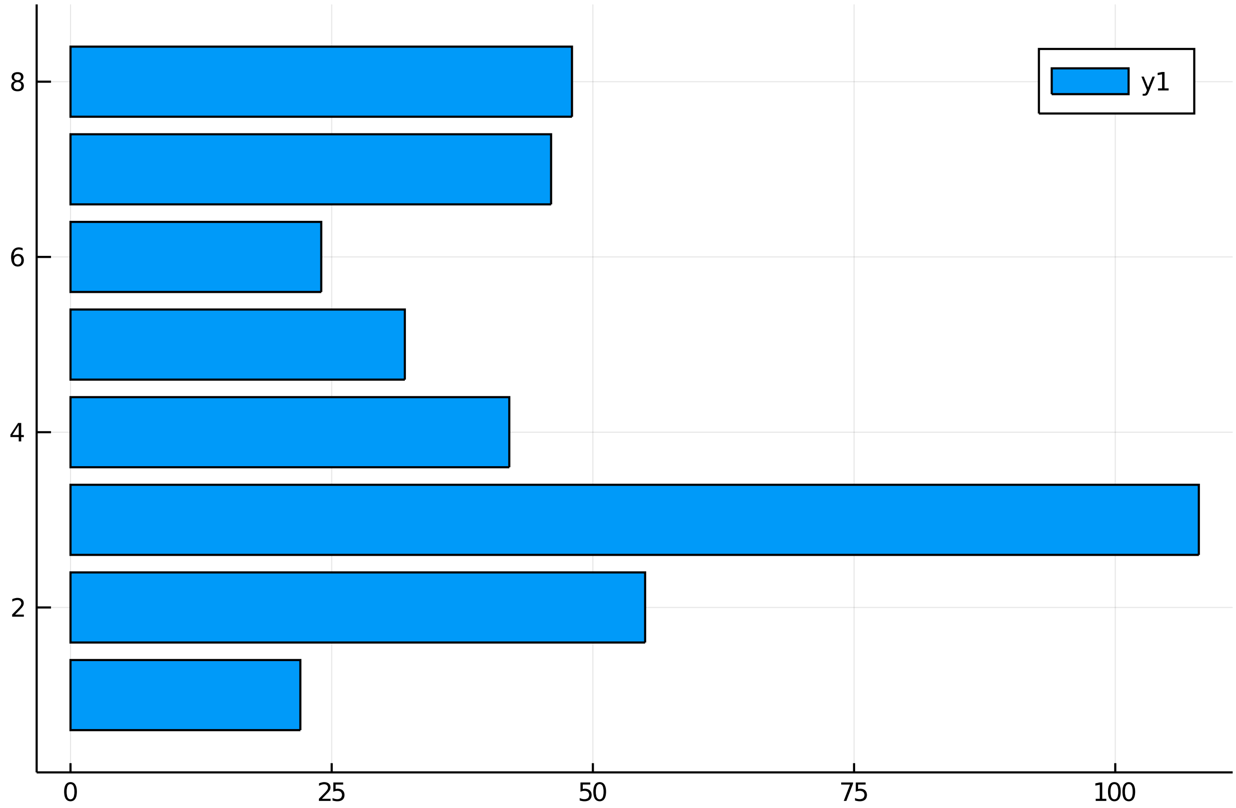
```
GMM = MixtureModel{Distributions.DiagNormal}(K = 8)
      components[1] (prior = 0.0880): DiagNormal(
      dim: 2
      μ: [-206.86363636363637, 32.04545454545455]
      Σ: [1682.4090909090905 0.0; 0.0 1793.5692640692637]
      )

      components[2] (prior = 0.1412): DiagNormal(
      dim: 2
      μ: [94.0, 236.34545454545454]
      Σ: [904.925925925926 0.0; 0.0 451.15622895622886]
      )

      components[3] (prior = 0.3475): DiagNormal(
      dim: 2
      μ: [-0.6481481481481481, 16.51851851851852]
      Σ: [310.6227068189684 0.0; 0.0 157.20526133610235]
      )

      components[4] (prior = 0.1027): DiagNormal(
      dim: 2
      μ: [-51.07142857142857, 256.5952380952381]
      Σ: [1739.5313588850177 0.0; 0.0 546.7346109175378]
      )

      components[5] (prior = 0.0458): DiagNormal(
      dim: 2
      μ: [-6.9375, 143.0]
      Σ: [1767.2217741935483 0.0; 0.0 527.2258064516129]
      )

      components[6] (prior = 0.0354): DiagNormal(
      dim: 2
```

```
# This is a gaussian mixture model we generated with our clusters
GMM = MixtureModel([di.MvNormal(Rx.centers[:,1],[std(cluster_x_new[1]),
std(cluster_y_new[1])]), di.MvNormal(Rx.centers[:,2],[std(cluster_x_new[2]),
std(cluster_y_new[2])]), di.MvNormal(Rx.centers[:,3],[std(cluster_x_new[3]),
std(cluster_y_new[3])]), di.MvNormal(Rx.centers[:,4],[std(cluster_x_new[4]),
std(cluster_y_new[4])]), di.MvNormal(Rx.centers[:,5],[std(cluster_x_new[5]),
std(cluster_y_new[5])]), di.MvNormal(Rx.centers[:,6],[std(cluster_x_new[6]),
std(cluster_y_new[6])]), di.MvNormal(Rx.centers[:,7],[std(cluster_x_new[7]),
std(cluster_y_new[7])]), di.MvNormal(Rx.centers[:,8],[std(cluster_x_new[8]),
std(cluster_y_new[8])])], [weights_norm[1], weights_norm[2], weights_norm[3],
weights_norm[4], weights_norm[5], weights_norm[6], weights_norm[7],
weights_norm[8]])
```




```
#Visualization of the GMM
begin
        Z = [pdf(GMM,[i,j]) for i in -400:400, j in -400:400]
        plot(-400:400,-400:400,Z,st=:surface, color=:viridis)
end
```

```
#future work: k means clustering has some drawback, some points might be
calculated multiple times, so for the more accurate calculation/prediction, we
can also use neuron network
```




```
bar(collect(keys(Rx.counts)), collect(values(Rx.counts)),
orientation=:horizontal, yticks= :all)
```

```
# In this next segment, we will attempt to see if Stephen Curry's performs
better or worse during the Playoffs as compared to during the regular season. A
quick Google search will show that Stephen Curry has a 47.3% shot percentage
during the regular season and a 38.5% shot percentage during the Playoffs.
These results alone would indicate that Stephen Curry performs worse in the
Playoffs. We wanted to investigate the validity of this by comparing Stephen
Curry's 2017 seasonal shot data with his total Playoff data. To do this, we
compared how Stephen Curry performed against the opponents he faced in the
playoffs with how he performed when he faced those same opponents during the
2017 season.
```

```
["New Orleans Pelicans", "Houston Rockets", "Atlanta Hawks", "Toronto Raptors", "Bos
```

```
new_df.opponent
```

```
op_sets =
  Set(["Phoenix Suns", "Philadelphia 76ers", "Portland Trail Blazers", "Miami Heat", '
```

```
op_sets = Set(new_df.opponent)
```

```
op_list_array =
  ["Phoenix Suns", "Philadelphia 76ers", "Portland Trail Blazers", "Miami Heat", "Utal
```

```
op_list_array = [a for a in op_sets]
```

```
"Phoenix Suns"
```

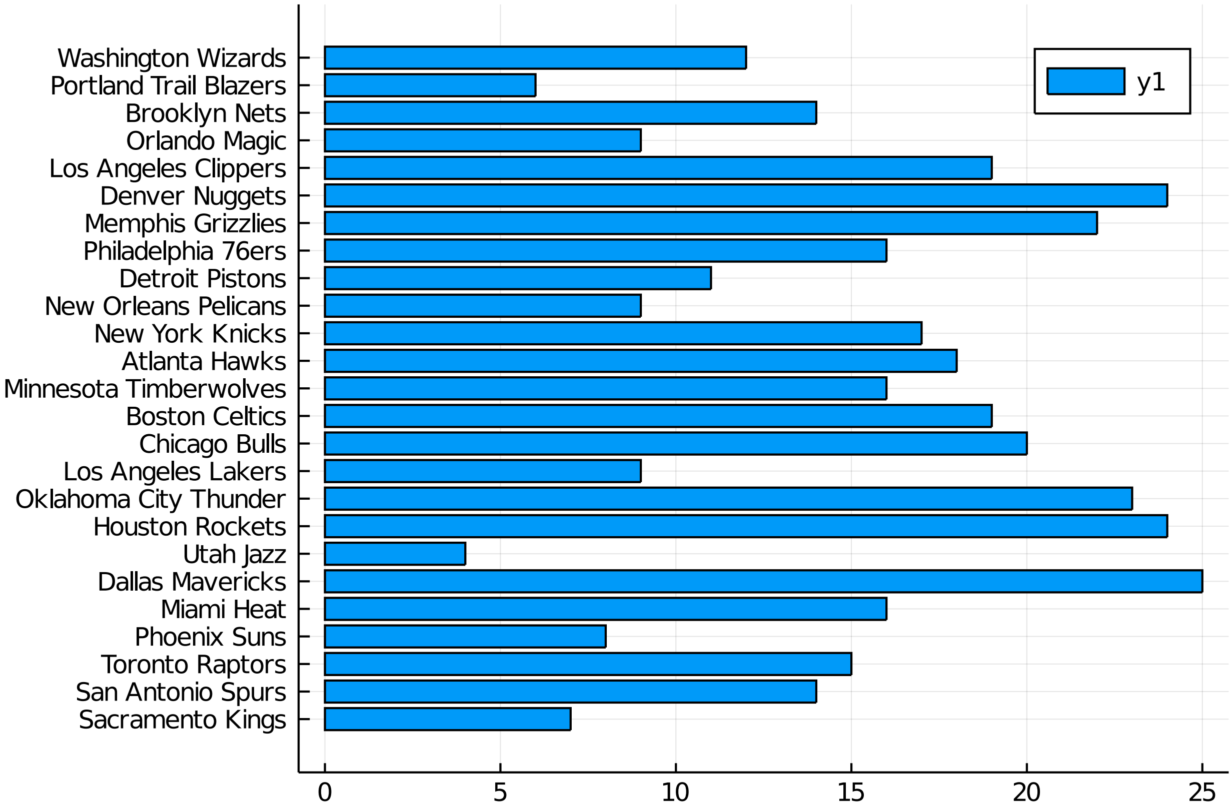
```julia
op_list_array[1]
```

```julia
#Dictionary with all of Stephen Curry's shots taken against all teams
begin
    op_dict_Total = Dict{String, Int}()
    #op_dict["PS"] = 0
    #op_dict["Phil"] = 0
    for j in 1:size(op_list_array,1)
        op_dict_Total[op_list_array[j]] = 0
        for i in 1:size(df.opponent,1)
            if df.opponent[i] == op_list_array[j]
                op_dict_Total[op_list_array[j]] += 1
            end
        end
    end
end
```

```julia
#This code is to make a dictionary with a record of all shots that stephen
curry has successfully made against each team. We will use this later to
calculate his shot probability against each team.
begin
    op_dict = Dict{String, Int}()
    #op_dict["PS"] = 0
    #op_dict["Phil"] = 0
    for j in 1:size(op_list_array,1)
        op_dict[op_list_array[j]] = 0
        for i in 1:size(new_df.opponent,1)
            if new_df.opponent[i] == op_list_array[j]
                op_dict[op_list_array[j]] += 1
            end
        end

    end
end
```

```
Dict("Sacramento Kings" ⇒ 7, "San Antonio Spurs" ⇒ 14, "Toronto Raptors" ⇒ 15, "
```

```julia
op_dict
```




```
bar(collect(keys(op_dict)), collect(values(op_dict)), orientation=:horizontal,
yticks= :all)
```

delete_teams (generic function with 1 method)

```
#This function will delete all teams that Stephen Curry did not face in the
playoffs. The purpose of this is so that we can compare the teams that he faced
in both the season and playoffs to draw a better conclusion as to whether he
performs better or worse in the Playoffs.
function delete_teams(X)
delete!(X,"Miami Heat");delete!(X,"Toronto Raptors");delete!(X,"Washington
Wizards");delete!(X,"Brooklyn Nets");delete!(X,"Orlando Magic");delete!
(X,"Phoenix Suns"); delete!(X,"Atlanta Hawks"); delete!(X,"Utah Jazz"); delete!
(X,"Detroit Pistons"); delete!(X,"Philadelphia 76ers"); delete!(X,"New York
Knicks"); delete!(X,"Chicago Bulls"); delete!(X,"Minnesota Timberwolves");
delete!(X,"Boston Celtics"); delete!(X,"Dallas Mavericks"); delete!
(X,"Sacramento Kings"); delete!(X,"Los Angeles Lakers")
end
```

Dict("San Antonio Spurs" ⇒ 26, "Houston Rockets" ⇒ 58, "Oklahoma City Thunder" ⇒

```
begin
    delete_teams(op_dict)
    delete_teams(op_dict_Total)
end
```

```
#This code takes the values in the dictionary and converts it into an [Any] array. In order to get the percent shot made against each team, the shots successfully made are divided by the total shots taken then multiplied by 100. Note that the values are being sorted to match the corresponding Playoff team for a paired t-test that will later be conducted.
begin
    Probability_Season = []
    for i in 1:size(collect(values(sort(op_dict))),1)
        push!(Probability_Season, collect(values(sort(op_dict)))[i] ./ collect(values(sort(op_dict_Total)))[i] .* 100)
    end
end
```

[51.0638, 41.3793, 67.8571, 64.7059, 47.3684, 45.098, 35.2941, 53.8462]

```
Probability_Season
```

	name	team_name	game_date	season_Playoff	espn_pl
1	"Stephen Curry"	"Golden State Warriors"	"6/5/2016"	2015	3975
2	"Stephen Curry"	"Golden State Warriors"	"5/24/2016"	2015	3975
3	"Stephen Curry"	"Golden State Warriors"	"5/30/2016"	2015	3975
4	"Stephen Curry"	"Golden State Warriors"	"6/13/2016"	2015	3975
5	"Stephen Curry"	"Golden State Warriors"	"5/6/2018"	2017	3975
6	"Stephen Curry"	"Golden State Warriors"	"6/10/2016"	2015	3975
7	"Stephen Curry"	"Golden State Warriors"	"5/30/2016"	2015	3975
8	"Stephen Curry"	"Golden State Warriors"	"5/4/2018"	2017	3975
9	"Stephen Curry"	"Golden State Warriors"	"5/9/2015"	2014	3975
10	"Stephen Curry"	"Golden State Warriors"	"5/27/2015"	2014	3975
	more				
939	"Stephen Curry"	"Golden State Warriors"	"5/13/2015"	2014	3975

```
# We will then follow the same sequence of steps for the Playoff data. Below is the CSV file for Stephen Curry's Playoff statistics.
begin
    csv_reader_Playoff = CSV.File("nba_savant (2) .csv")
    df_reader_Playoff = DataFrame(csv_reader_Playoff)
end
```

df_Playoff =

	name	shot_made_flag_Playoff	x_Playoff	y_Playoff	opponent_Play
1	"Stephen Curry"	1	-56	52	"Cleveland Cavali
2	"Stephen Curry"	1	84	56	"Oklahoma City Th
3	"Stephen Curry"	1	-37	61	"Oklahoma City Th
4	"Stephen Curry"	1	2	62	"Cleveland Cavali
5	"Stephen Curry"	1	29	202	"New Orleans Peli
6	"Stephen Curry"	0	-78	21	"Cleveland Cavali
7	"Stephen Curry"	0	-16	47	"Oklahoma City Th
8	"Stephen Curry"	0	-9	116	"New Orleans Peli
9	"Stephen Curry"	1	31	9	"Memphis Grizzlie
10	"Stephen Curry"	1	-4	44	"Houston Rockets"
more					
939	"Stephen Curry"	1	102	230	"Memphis Grizzlie

```
df_Playoff = df_reader_Playoff[:,
["name","shot_made_flag_Playoff","x_Playoff","y_Playoff","opponent_Playoff","sho
t_distance_Playoff"]]
```

1

```
df_Playoff.shot_made_flag_Playoff[1]
```

```
begin
    new_df_Playoff = DataFrame()
    for i in 1:size(df_Playoff.shot_made_flag_Playoff,1)
        if df_Playoff.shot_made_flag_Playoff[i] !=0
            push!(new_df_Playoff, df_Playoff[i,:])
        end
    end
end
```

["Cleveland Cavaliers", "Oklahoma City Thunder", "Oklahoma City Thunder", "Clevelan

```
new_df_Playoff.opponent_Playoff
```

op_sets_Playoff =
Set(["Portland Trail Blazers", "Oklahoma City Thunder", "Houston Rockets", "San Ant

```
op_sets_Playoff = Set(new_df_Playoff.opponent_Playoff)
```

```
op_list_array_Playoff =
  ["Portland Trail Blazers", "Oklahoma City Thunder", "Houston Rockets", "San Antonio
```

```julia
op_list_array_Playoff = [a for a in op_sets_Playoff]
```

```
Dict("Los Angeles Clippers" ⇒ 88, "Houston Rockets" ⇒ 96, "Oklahoma City Thunder"
```

```julia
#Total shots taken against each team in the Playoffs. Note, we needed to delete
the Cleveland Cavaliers from this because Stephen Curry did not face against
the Cleveland Cavaliers during the 2017 season.
begin
    op_dict_TotalPlayoff = Dict{String, Int}()
    #op_dict["PS"] = 0
    #op_dict["Phil"] = 0
    for j in 1:size(op_list_array_Playoff,1)
        op_dict_TotalPlayoff[op_list_array_Playoff[j]] = 0
        for i in 1:size(df_Playoff.opponent_Playoff,1)
            if df_Playoff.opponent_Playoff[i] == op_list_array_Playoff[j]
                op_dict_TotalPlayoff[op_list_array_Playoff[j]] += 1
            end
        end
    end
    delete!(op_dict_TotalPlayoff, "Cleveland Cavaliers")
end
```

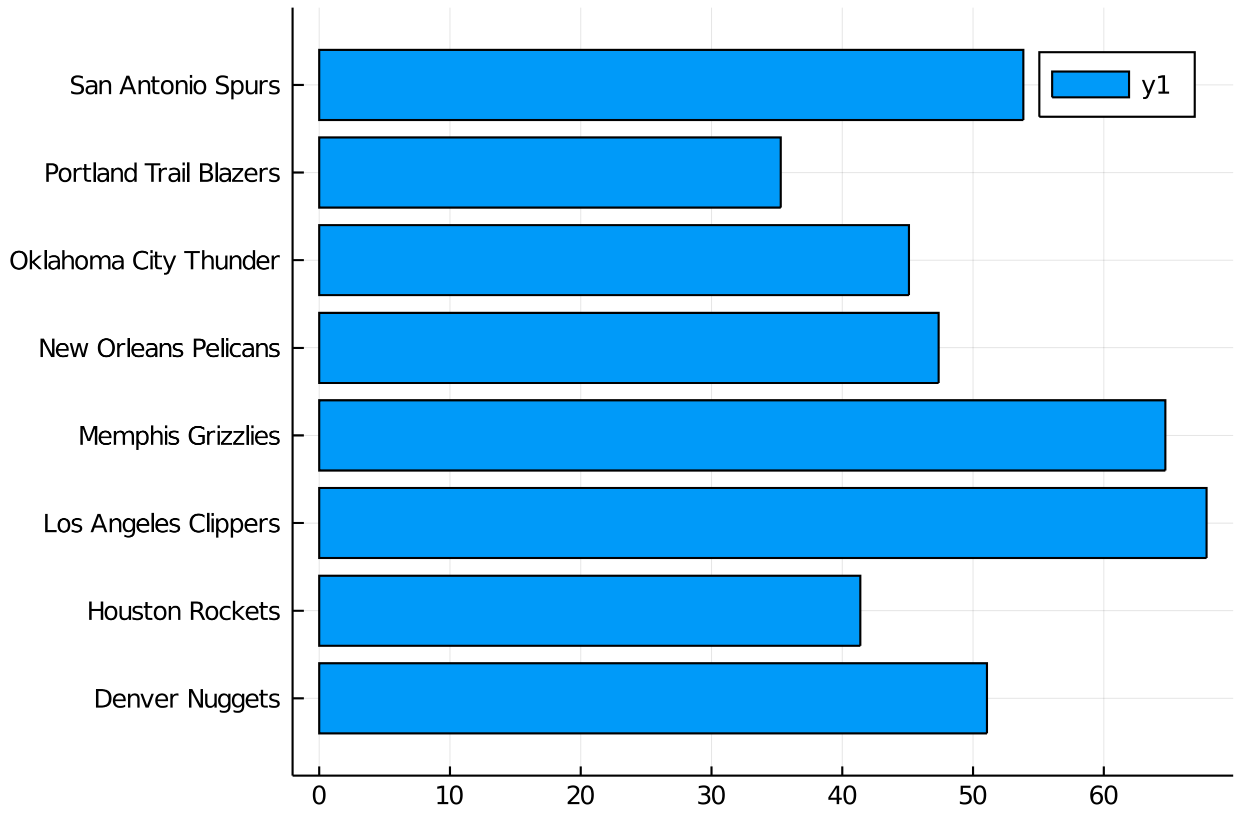
```
Dict("Los Angeles Clippers" ⇒ 33, "Houston Rockets" ⇒ 46, "Oklahoma City Thunder"
```

```julia
#Dictionary with Successful shots made against each team
begin
    op_dict_Playoff = Dict{String, Int}()
    #op_dict["PS"] = 0
    #op_dict["Phil"] = 0
    for j in 1:size(op_list_array_Playoff,1)
        op_dict_Playoff[op_list_array_Playoff[j]] = 0
        for i in 1:size(new_df_Playoff.opponent_Playoff,1)
            if new_df_Playoff.opponent_Playoff[i] == op_list_array_Playoff[j]
                op_dict_Playoff[op_list_array_Playoff[j]] += 1
            end
        end
    end
    delete!(op_dict_Playoff, "Cleveland Cavaliers")
end
```

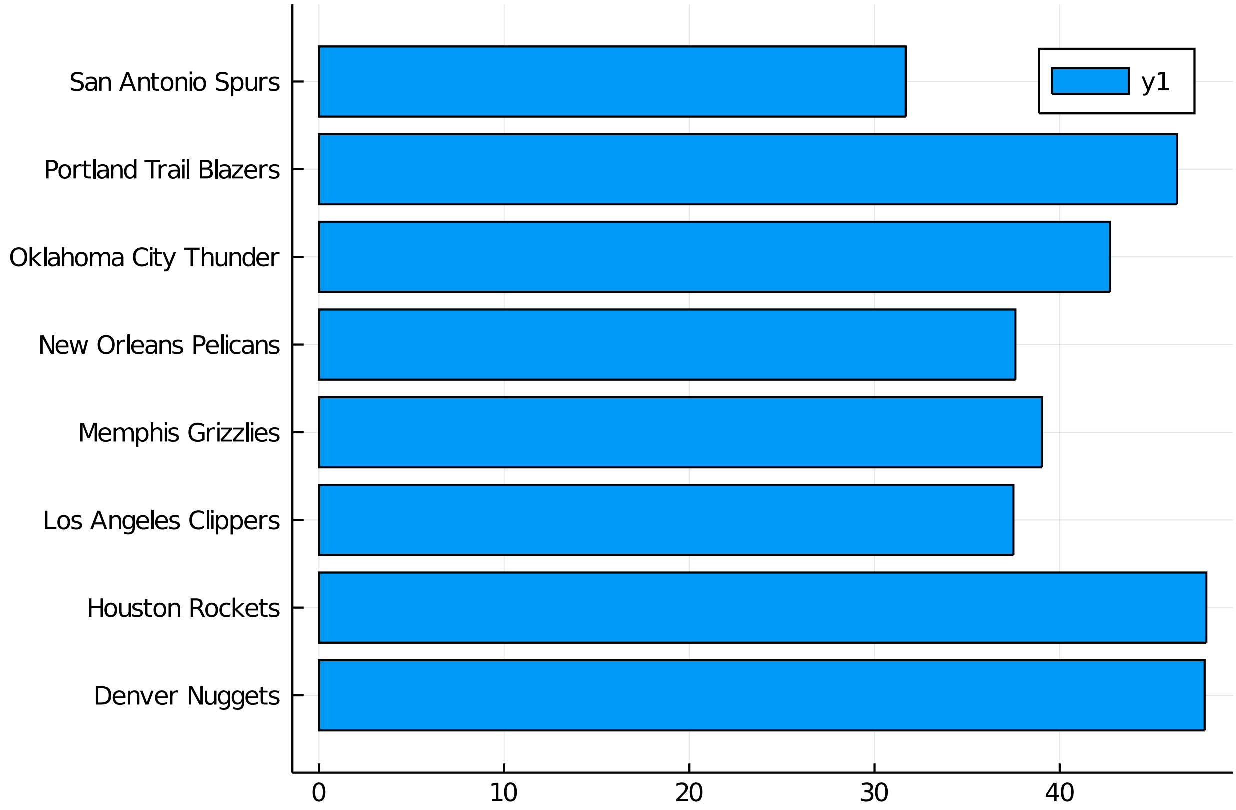
```julia
#Stephen Curry's shot made percentage against each team in the Playoffs
begin
    Probability_Playoff = []
    for i in 1:size(collect(values(sort(op_dict_Playoff))),1)
        push!(Probability_Playoff, collect(values(sort(op_dict_Playoff)))[i] ./
        collect(values(sort(op_dict_TotalPlayoff)))[i] .* 100)
    end
end
```

[47.8261, 47.9167, 37.5, 39.0476, 37.6068, 42.7184, 46.3415, 31.6832]

```
Probability_Playoff
```




```
#Bar graph depicting Stephen Curry's Shot made Percentage against each team
during the 2017 regular season
bar(collect(keys(sort(op_dict))), Probability_Season, orientation=:horizontal,
yticks= :all,)
```




```
#Bar graph depicting Stephen Curry's Shot made Percentage against each team
during the Playoffs
bar(collect(keys(sort(op_dict_Playoff))), Probability_Playoff,
orientation=:horizontal, yticks= :all,)
```

VecP_S = [51.0638, 41.3793, 67.8571, 64.7059, 47.3684, 45.098, 35.2941, 53.8462]

```
VecP_S = Vector{Float64}(vec(Probability_Season))
```

VecP_P = [47.8261, 47.9167, 37.5, 39.0476, 37.6068, 42.7184, 46.3415, 31.6832]

```
VecP_P = Vector{Float64}(vec(Probability_Playoff))
```

```
One sample t-test
-----------------
Population details:
    parameter of interest:   Mean
    value under h_0:         0
    point estimate:          9.49658
    95% confidence interval: (-3.245, 22.24)

Test summary:
    outcome with 95% confidence: fail to reject h_0
    two-sided p-value:           0.1214

Details:
    number of observations:   8
    t-statistic:              1.7624740731587087
    degrees of freedom:       7
    empirical standard error: 5.388207523846388
```

```
#We will now conduct a paired T-test to see if there is a significant difference between the probability of making a shot during the season vs probability of making a shot against the same team in the playoff.
OneSampleTTest(vec(VecP_S), vec(VecP_P))
```

```
#From the T-test, the results arrived at p=0.1214 which indicated that there is no significant difference between the two data sets (p>0.05). This indicates that Stephen Curry performs no worse during the playoffs compared to his season. The discrepancy in his total shot made percentage during the regular season (47.3%) as compared to his playoff shot percentage (38.5%) could be attributed to the fact that the playoff teams are harder opponents to score against. During the regular season, weaker teams could inflate Stephen Curry's field goal percentage. When meeting playoff teams during the regular season, Stephen Curry appears to perform similarly.
```