Problem Set 6

Bennett Hellman

I collaborated with Kyle Maulden, Iggy Siegel, Ananya Krishnan, and Jordan Baruch.

In [324]:
```
using JuMP, Gurobi, LinearAlgebra, CSV, DataFrames, Pkg, Distances, Plots,
```

In [400]:
```
heart = CSV.read("heart_failure_clinical_records_dataset.csv", DataFrame, h
players2 = CSV.read("players2.csv", DataFrame, header=1);
```

In [401]:
```
first(heart, 1)
```

Out[401]: 1 rows × 13 columns (omitted printing of 7 columns)

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure
	Float64	Int64	Int64	Int64	Int64	Int64
1	75.0	0	582	0	20	1

6.1 Question 1: Interpretable Clustering

1.a

In [402]:
```
describe(heart, :max, :min, :mean, :median, :std)
```

Out[402]: 13 rows × 6 columns

	variable	max	min	mean	median	std
	Symbol	Real	Real	Float64	Float64	Float64
1	age	95.0	40.0	60.8339	60.0	11.8948
2	anaemia	1	0	0.431438	0.0	0.496107
3	creatinine_phosphokinase	7861	23	581.839	250.0	970.288
4	diabetes	1	0	0.41806	0.0	0.494067
5	ejection_fraction	80	14	38.0836	38.0	11.8348
6	high_blood_pressure	1	0	0.351171	0.0	0.478136
7	platelets	850000.0	25100.0	263358.0	262000.0	97804.2
8	serum_creatinine	9.4	0.5	1.39388	1.1	1.03451
9	serum_sodium	148	113	136.625	137.0	4.41248
10	sex	1	0	0.648829	1.0	0.478136
11	smoking	1	0	0.32107	0.0	0.46767
12	time	285	4	130.261	115.0	77.6142
13	DEATH_EVENT	1	0	0.32107	0.0	0.46767

From page 404, distance can be influenced by the scale of the different attributes, so it is important to consider normalizing the data before computing distances. We normalize columnwise, to ensure that no one variable drives the clustering algorithm solely because it is on a larger scale.

In [423]:
```
#h2 = DataFrame(LinearAlgebra.normalize(Matrix(heart)), :col)
X = Matrix(heart);
h2[:,[1,3,5,7,8,9,12]] = (X[:,[1,3,5,7,8,9,12]] .- mean(X[:,[1,3,5,7,8,9,12
first(h2, 10)
```

Out[423]: 10 rows × 13 columns (omitted printing of 8 columns)

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction
	Float64	Float64	Float64	Float64	Float64
1	1.19095	0.0	0.000165451	0.0	-1.528
2	-0.490457	0.0	7.50206	0.0	-0.00706491
3	0.350246	0.0	-0.449186	0.0	-1.528
4	-0.910808	2.05895e-7	-0.485257	0.0	-1.528
5	0.350246	2.05895e-7	-0.434757	2.05895e-7	-1.528
6	2.452	2.05895e-7	-0.551217	0.0	0.161928
7	1.19095	2.05895e-7	-0.346124	0.0	-1.95048
8	-0.0701056	2.05895e-7	-0.275011	2.05895e-7	1.85185
9	0.350246	0.0	-0.437849	0.0	2.27433
10	1.6113	2.05895e-7	-0.47289	0.0	-0.260554

1.b)

In [424]:
```
vec = Vector{Float64}()
ind = Vector{Float64}()
for i in 2:13
    R = kmeans(Matrix(h2[:,:]), i)
    a = assignments(R)
    c = counts(R)
    M = R.centers
    distances = pairwise(SqEuclidean(), Matrix(h2[:,:]))
#     mean(silhouettes(a, c, distances))
#     print("\n")
#     print(i)
#     print("\n", mean(silhouettes(a, c, distances)))
    append!(vec, mean(silhouettes(a, c, distances)))
    append!(ind, i)
end
```

In [427]:
```
plot(ind, vec, seriestype = :line, color = "red", title = "Silhoutte Values
```

Out[427]:

In [428]:
```
features = collect(Matrix(h2[:,:])');
result = kmeans(features, 8);
```

I calculated the silhoutte value at each k value from [1,13] and found the local maximum at k=8. "The value of Silhouette score varies from -1 to 1. If the score is 1, the cluster is dense and well-separated than other clusters. A value near 0 represents overlapping clusters with samples very close to the decision boundary of the neighbouring clusters. A negative score [-1, 0] indicate that the samples might have got assigned to the wrong clusters." ([https://vitalflux.com/kmeans-silhouette-score-explained-with-python-example/ (https://vitalflux.com/kmeans-silhouette-score-explained-with-python-example/)](https://vitalflux.com/kmeans-silhouette-score-explained-with-python-example/))

1.c

In [431]:
```
scatter(heart.age, heart.platelets, marker_z=result.assignments,
        color=:lightrainbow, legend=false, xaxis = "age", y = "platelets")
```

Out[431]:

1.d)

The methodology for the tree hyperparameters was to create an interpretable tree that would still be able to demonstrate all eight of my clusters. I wanted the smallest tree possible that showed all eight clusters. However, this was not possible with an interpretable tree and is a drawback of the two stage appraoch as I will discuss in 1.f.

In [421]:
```
hk = hcat(heart, result.assignments);
first(hk, 5);
```

In [420]:
```
grid = IAI.GridSearch(
    IAI.OptimalTreeClassifier(
        random_seed=1,
    ),
    max_depth=1:5,
)
IAI.fit!(grid, hk[:, Not("x1")], result.assignments)
IAI.get_learner(grid)
```

```
┌ Warning: ProgressMeter by default refresh meters with additional inform
ation in IJulia via `IJulia.clear_output`, which clears all outputs in th
e cell.
│  - To prevent this behaviour, do `ProgressMeter.ijulia_behavior(:appen
d)`.
│  - To disable this warning message, do `ProgressMeter.ijulia_behavior(:
clear)`.
└ @ ProgressMeter /Users/iai/builds/InterpretableAI/SystemImage/SysImgBui
lder/.julia/packages/ProgressMeter/Vf8un/src/ProgressMeter.jl:620
Refitting with best parameters...  100%|████████████████| Time: 0:00:00
4m  Parameters:  cp=>0.00974 max_depth=>4
```

Out[420]: Collapse Expand Save PNG

1.e

Every branch of the tree with the exception of the far right side has a combinatino of age and ejection fraction as part of their partitioning criteria. Cluster 1 seems to be younger people with low serum sodium and ejection fraction. Cluster 8 is younger people with very low ejection fraction but slightly higher serum sodium. Cluster 6 are younger people with more average ejection fraction. Cluster 4 are older people than clusters 1 & 8 but with slightly higher ejection fraction. Cluster 7 are yet again a little older but this time with lower ejection fraction than cluster 6. Cluster 5 has very long follow up periods and high platelets.

1.f

As demonstrated by 1.e, using the dual approach is not gauranteed interpretable trees. ICOT ensures that every bucket coincides with exactly one cluster only and vice versa. As such, ICOT ensures interpretability "with simple and direct descriptions of each resulting cluster" while the dual approach does not (pg. 412).

Question 2

In [383]:
```
players2 = CSV.read("players2.csv", DataFrame, header=1);
first(players2, 10)
```

Out[383]: 10 rows × 5 columns

	Column1	X	Player	Age	Points
	Int64	Int64	String	Int64	Int64
1	1	1	Novak Djokovic	34	10940
2	2	2	Daniil Medvedev	25	7640
3	3	3	Alexander Zverev\n	24	6540
4	4	4	Stefanos Tsitsipas	23	6540
5	5	5	Andrey Rublev\n	24	4950
6	6	6	Rafael Nadal	35	4875
7	7	7	Matteo Berrettini	25	4568
8	8	8	Casper Ruud\n	22	3760
9	9	9	Hubert Hurkacz	24	3706
10	10	10	Felix Auger-Aliassime\n	21	3308

In [384]:
```
players2.Points = (players2.Points .- mean(players2.Points)) ./ std(players
first(players)
```

Out[384]: DataFrameRow (4 columns)

	Column1	Player	Age	Points
	Int64	String	Int64	Float64
1	1	Novak Djokovic	34	3.5557

2a

In terms of best performance, the best strategy is MIO, then triple-matching, re-randomization and then randomization. This order holds true for the mean differential in first and second order moments and (almost all) maximum values.

2a.i

In [385]:
```
Random.seed!(9999)
df = players2[shuffle(axes(players2, 1)), :]
df1 = df[1:10, :];
df2 = df[11:20, :];
df3 = df[21:30, :];
mu12 = abs(mean(df1[:,5])-mean(df2[:,5]))
mu13 = abs(mean(df1[:,5])-mean(df3[:,5]))
mu23 = abs(mean(df2[:,5])-mean(df3[:,5]))
var12 = abs(var(df1[:,5])*.9-var(df2[:,5])*.9)
var13 = abs(var(df1[:,5])*.9-var(df3[:,5])*.9)
var23 = abs(var(df2[:,5])*.9-var(df3[:,5])*.9)
print("|Mean_1-Mean_2| = ", mu12, "\n")
print("|Mean_1-Mean_3| = ", mu13, "\n")
print("|Mean_2-Mean_3| = ", mu23, "\n")
print("|Var_1-Var_2| = ", var12, "\n")
print("|Var_1-Var_3| = ", var13, "\n")
print("|Var_2-Var_3| = ", var23, "\n")
sum_mu = mu12+mu13+mu23
sum_var = var12+var13+var23
print("Max Discrepancy Mean = 0.869", "\n")
print("Max Discrepancy Variance = 1.903", "\n")
print("Mean Discrepancy of Mean = ", sum_mu/3, "\n")
print("Mean Discrepancy of Variance = ", (sum_var/3))
```

```
|Mean_1-Mean_2| = 0.4113398564952358
|Mean_1-Mean_3| = 0.8689848282645067
|Mean_2-Mean_3| = 0.45764497176927094
|Var_1-Var_2| = 0.4578321715578988
|Var_1-Var_3| = 1.902888283697621
|Var_2-Var_3| = 1.4450561121397223
Max Discrepancy Mean = 0.869
Max Discrepancy Variance = 1.903
Mean Discrepancy of Mean = 0.5793232188430045
Mean Discrepancy of Variance = 1.2685921891317475
```

2a.ii

In [388]:
```
mu_vec = Vector{Float64}()
var_vec =  Vector{Float64}()
disc_mean = Vector{Float64}()
disc_var = Vector{Float64}()
for i in 1:10000
    Random.seed!(i)
    df = players2[shuffle(axes(players2, 1)), :]
    df1 = df[1:10, :]
    df2 = df[11:20, :]
    df3 = df[21:30, :]
    mu12 = abs(mean(df1[:,5])-mean(df2[:,5]))
    mu13 = abs(mean(df1[:,5])-mean(df3[:,5]))
    mu23 = abs(mean(df2[:,5])-mean(df3[:,5]))
    var12 = abs(var(df1[:,5])*.9-var(df2[:,5])*.9)
    var13 = abs(var(df1[:,5])*.9-var(df3[:,5])*.9)
    var23 = abs(var(df2[:,5])*.9-var(df3[:,5])*.9)
    sum_mu = mu12+mu13+mu23
    sum_var = var12+var13+var23
    append!(mu_vec, sum_mu)
    append!(var_vec, sum_var)
    append!(disc_mean, mu12)
    append!(disc_mean, mu13)
    append!(disc_mean, mu23)
    append!(disc_var, var12)
    append!(disc_var, var13)
    append!(disc_var, var23)
end
#print(argmin(mu_vec), "\n")
#print(argmin(var_vec), "\n")
tot_vec = mu_vec+var_vec;
#print(argmin(tot_vec), "\n")
#print(disc_mean[argmax(disc_mean)], "\n")
#print(disc_var[argmax(disc_var)])
```

In [399]:
```
Random.seed!(argmin(tot_vec))
df = players2[shuffle(axes(players2, 1)), :]
df1 = df[1:10, :]
df2 = df[11:20, :]
df3 = df[21:30, :]
mu12 = abs(mean(df1[:,5])-mean(df2[:,5]))
mu13 = abs(mean(df1[:,5])-mean(df3[:,5]))
mu23 = abs(mean(df2[:,5])-mean(df3[:,5]))
var12 = abs(var(df1[:,5])*.9-var(df2[:,5])*.9)
var13 = abs(var(df1[:,5])*.9-var(df3[:,5])*.9)
var23 = abs(var(df2[:,5])*.9-var(df3[:,5])*.9)
print("|Mean_1-Mean_2| = ", mu12, "\n")
print("|Mean_1-Mean_3| = ", mu13, "\n")
print("|Mean_2-Mean_3| = ", mu23, "\n")
print("|Var_1-Var_2| = ", var12, "\n")
print("|Var_1-Var_3| = ", var13, "\n")
print("|Var_2-Var_3| = ", var23, "\n")

sum_mu = mu12+mu13+mu23
sum_var = var12+var13+var23

print("Max Discrepancy Mean = ", "0.425", "\n")
print("Max Discrepancy Variance = ", "1.166", "\n")
print("Mean Discrepancy of Mean = ", mean(sum_mu), "\n")
print("Mean Discrepancy of Variance = ", mean(sum_var))
```

```
|Mean_1-Mean_2| = 0.26067194334469523
|Mean_1-Mean_3| = 0.4252548962121033
|Mean_2-Mean_3| = 0.164582952867408
|Var_1-Var_2| = 0.6945200095832237
|Var_1-Var_3| = 1.1663288135205225
|Var_2-Var_3| = 0.47180880393729896
Max Discrepancy Mean = 0.425
Max Discrepancy Variance = 1.166
Mean Discrepancy of Mean = 0.8505097924242065
Mean Discrepancy of Variance = 2.332657627041045
```

2a.iii

In [390]:
```
rp = sort!(players2, [:Points])
df1 = rp[1:3:30, :]
df2 = rp[2:3:30, :]
df3 = rp[3:3:30, :]
mu12 = abs(mean(df1[:,5])-mean(df2[:,5]))
mu13 = abs(mean(df1[:,5])-mean(df3[:,5]))
mu23 = abs(mean(df2[:,5])-mean(df3[:,5]))
var12 = abs(var(df1[:,5])*.9-var(df2[:,5])*.9)
var13 = abs(var(df1[:,5])*.9-var(df3[:,5])*.9)
var23 = abs(var(df2[:,5])*.9-var(df3[:,5])*.9)
print("|Mean_1-Mean_2| = ", mu12, "\n")
print("|Mean_1-Mean_3| = ", mu13, "\n")
print("|Mean_2-Mean_3| = ", mu23, "\n")
print("|Var_1-Var_2| = ", var12, "\n")
print("|Var_1-Var_3| = ", var13, "\n")
print("|Var_2-Var_3| = ", var23, "\n")

sum_mu = mu12+mu13+mu23
sum_var = var12+var13+var23

print("Max Discrepancy Mean = ", "0.373", "\n")
print("Max Discrepancy Variance = ", "1.179", "\n")
print("Mean Discrepancy of Mean = ", sum_mu/3, "\n")
print("Mean Discrepancy of Variance = ", (sum_var/3))
```

```
|Mean_1-Mean_2| = 0.08292611506943953
|Mean_1-Mean_3| = 0.37335555888973293
|Mean_2-Mean_3| = 0.29042944382029334
|Var_1-Var_2| = 0.18288445083012128
|Var_1-Var_3| = 1.1788928470526785
|Var_2-Var_3| = 0.9960083962225571
Max Discrepancy Mean = 0.373
Max Discrepancy Variance = 1.179
Mean Discrepancy of Mean = 0.2489037059264886
Mean Discrepancy of Variance = 0.7859285647017856
```

2.iv

In [391]:
```
function fmean(w,x,p)
    n = size(x)[1]
    k = size(x)[2]
    sum = 0
    for i=1:n
        sum = sum + (w[i]*x[i,p])
    end
    return sum / (n/k)
end

function fvar(w,x,p)
    n = size(x)[1]
    k = size(x)[2]
    sum = 0
    for i=1:n
        sum = sum + (w[i]*w[i]*x[i,p])
    end
    return sum / (n/k)
end

#Kyle Maulden helped me create this helper functions
```

Out[391]:
```
fvar (generic function with 1 method)
```

In [392]:
```
mod = JuMP.Model(JuMP.optimizer_with_attributes(() -> Gurobi.Optimizer(),"M
set_optimizer_attribute(mod, "OutputFlag", 0)

#PARAMETERS
rho = 0.5
k = 10
n = 30
m = 3
w = players2.Points

#VARIABLES
#if player i is assigned to group p
@variable(mod, x[i=1:n, p=1:m], Bin)
#epigraph variable
@variable(mod, d)

#CONSTRAINTS
#epigraph constraints
@constraint(mod, [q in 2:m, p in 1:q], d >= fmean(w,x,p) - fmean(w,x,q)+ rh
@constraint(mod, [q in 2:m, p in 1:q], d >= fmean(w,x,p) - fmean(w,x,q)+ rh
@constraint(mod, [q in 2:m, p in 1:q], d >= fmean(w,x,q) - fmean(w,x,p)+ rh
@constraint(mod, [q in 2:m, p in 1:q], d >= fmean(w,x,q) - fmean(w,x,p)+ rh

#every player is assigned a group
@constraint(mod, [p=1:m], sum(x[i,p] for i=1:n)== k)
#a player is assigned to exaclty one group
@constraint(mod, [i=1:n], sum(x[i,p] for p=1:m)== 1)
@constraint(mod, [i=1:m-1, p=i+1:m], x[i,p]==0)

@objective(mod, Min, d)

optimize!(mod)
```

```
Academic license - for non-commercial use only - expires 2022-08-19
```

In [393]:
```
objective_value(mod)
```

Out[393]:
```
0.34353728802117683
```

In [394]:
```
x=value.(x)
```

Out[394]:
```
30×3 Matrix{Float64}:
  1.0   0.0   0.0
  1.0   0.0   0.0
  1.0  -0.0  -0.0
  0.0   1.0  -0.0
  0.0   1.0  -0.0
  1.0  -0.0  -0.0
 -0.0   1.0  -0.0
  0.0   1.0  -0.0
  0.0   1.0  -0.0
 -0.0   1.0  -0.0
 -0.0  -0.0   1.0
 -0.0  -0.0   1.0
  0.0  -0.0   1.0
  ⋮
 -0.0  -0.0   1.0
  0.0  -0.0   1.0
  1.0   0.0  -0.0
  0.0   1.0  -0.0
  1.0  -0.0  -0.0
  1.0  -0.0  -0.0
  0.0   1.0  -0.0
  1.0  -0.0  -0.0
 -0.0   1.0  -0.0
 -0.0   1.0  -0.0
  1.0  -0.0  -0.0
 -0.0  -0.0   1.0
```

In [395]:
```
piv = x.*(players2.Points)
```

Out[395]:
```
30×3 Matrix{Float64}:
 -0.783348   -0.0        -0.0
 -0.773476   -0.0        -0.0
 -0.740568    0.0         0.0
 -0.0        -0.729286    0.0
 -0.0        -0.677105    0.0
 -0.674284    0.0         0.0
  0.0        -0.651249    0.0
 -0.0        -0.623513    0.0
 -0.0        -0.599538    0.0
  0.0        -0.572742    0.0
  0.0         0.0        -0.538894
  0.0         0.0        -0.524791
 -0.0         0.0        -0.481072
  ⋮
  0.0         0.0        -0.202771
 -0.0         0.0        -0.1064
 -0.0321237  -0.0         0.0
  0.0         0.154977   -0.0
  0.180363   -0.0        -0.0
  0.560206   -0.0        -0.0
  0.0         0.704527   -0.0
  0.739785   -0.0        -0.0
 -0.0         1.48725    -0.0
 -0.0         1.48725    -0.0
  2.00436    -0.0        -0.0
 -0.0        -0.0         3.5557
```

In [396]:
```
df1_ind = []
df2_ind = []
df3_ind = []
for i=1:30
    if piv[i,1] != 0
        push!(df1_ind,i)
    end
    if piv[i,2] != 0
        push!(df2_ind,i)
    end
    if piv[i,3] != 0
        push!(df3_ind,i)
    end
end
df1 = players2[df1_ind, :Points];
df2 = players2[df2_ind, :Points];
df3 = players2[df3_ind, :Points];
```

In [397]:
```
mu12 = abs(mean(df1[:,:])-mean(df2[:,:]))
mu13 = abs(mean(df1[:,:])-mean(df3[:,:]))
mu23 = abs(mean(df2[:,:])-mean(df3[:,:]))
var12 = abs(var(df1[:,:])*.9-var(df2[:,:])*.9)
var13 = abs(var(df1[:,:])*.9-var(df3[:,:])*.9)
var23 = abs(var(df2[:,:])*.9-var(df3[:,:])*.9)
print("|Mean_1-Mean_2| = ", mu12, "\n")
print("|Mean_1-Mean_3| = ", mu13, "\n")
print("|Mean_2-Mean_3| = ", mu23, "\n")
print("|Var_1-Var_2| = ", var12, "\n")
print("|Var_1-Var_3| = ", var13, "\n")
print("|Var_2-Var_3| = ", var23, "\n")

sum_mu = mu12+mu13+mu23
sum_var = var12+var13+var23

print("Max Discrepancy Mean = ", "0.003", "\n")
print("Max Discrepancy Variance = ", "0.686", "\n")
print("Mean Discrepancy of Mean = ", sum_mu/3, "\n")
print("Mean Discrepancy of Variance = ", (sum_var/3))
```

```
|Mean_1-Mean_2| = 0.00272659562019697
|Mean_1-Mean_3| = 0.00037608215450990457
|Mean_2-Mean_3| = 0.0031026777747068745
|Var_1-Var_2| = 0.008391804454496943
|Var_1-Var_3| = 0.6863216809714349
|Var_2-Var_3| = 0.6779298765169379
Max Discrepancy Mean = 0.003
Max Discrepancy Variance = 0.686
Mean Discrepancy of Mean = 0.0020684518498045 83
Mean Discrepancy of Variance = 0.4575477873142899
```

2b.i

Formulation

Sets

$i = 1, \ldots, 30$ players

$m = 1, \ldots, 3$ groups

$k = 1, \ldots, 10$ players in each group

Variables

$x_{ip} = 1$ if player i is assigned to group p

z_{ij} = auxillary variable

$$Z_m^{opt}(\rho) = min_{x,d} \quad z$$

$$z_{ij} \geq w_i - w_j + M(x_{ip} + x_{jq} - 2), \quad \forall p < q \in [m], \forall i < j \in [n]$$

$$z_{ij} \geq w_j - w_i + M(x_{ip} + x_{jq} - 2), \quad \forall p < q \in [m], \forall i < j \in [n]$$

$$z_{ij} \geq w_i - w_j + M(x_{iq} + x_{jp} - 2), \quad \forall p < q \in [m], \forall i < j \in [n]$$

$$z_{ij} \geq w_j - w_i + M(x_{iq} + x_{jp} - 2), \quad \forall p < q \in [m], \forall i < j \in [n]$$

$$\sum_{i=1}^{n} x_{ip} = k \quad \forall p \in [m]$$

$$\sum_{p=1}^{m} x_{ip} = 1 \quad \forall i \in [n]$$

$$x_{ip} = 0 \quad \forall i < p$$

$$x_{ip} \in 0, 1 \quad \forall i \in [n], \forall p \in [m]$$

The first four constraints are epigraphs in order to linearize the real objective function and ensure the obejctive function is only calculating the sum of the differences between all points, and every point not in the same group, hence this subtraction by 2.

The fifth constraint ensures that all the people in a group add up to mandated group size for all groups, in this case 10.

The sixth constraint ensures that every player is assigned a group.

The seventh constraint ensures that players are only assigned one group.

The last constraint ensures players are assigned to exactly constraint and cannot be partially assigned to groups.

2b.ii

The MIO approach proves superior to re-randomization in this context with an objective value of 281.7 as opposed to 282.6. This is because it explicitly tries to minimize group differences while re-randomization just takes the best of random splits.

In [368]:

```
mod = JuMP.Model(JuMP.optimizer_with_attributes((TimeLimit=45) -> Gurobi.Op
set_optimizer_attribute(mod, "OutputFlag", 0)

#PARAMETERS
k = 10
n = 30
m = 3
w = players2.Points
M = 10000

#VARIABLES
@variable(mod, x[i=1:n, p=1:m], Bin)
@variable(mod, z[i=1:n,j=1:n]>=0)


#CONSTRAINTS
#epigraph constraints
@constraint(mod, [p=1:m-1, q=p+1:m, i=1:n-1, j=i+1:n], z[i,j] >= w[i] - w[j
@constraint(mod, [p=1:m-1, q=p+1:m, i=1:n-1, j=i+1:n], z[i,j] >= w[j] - w[i
@constraint(mod, [p=1:m-1, q=p+1:m, i=1:n-1, j=i+1:n], z[i,j] >= w[i] - w[j
@constraint(mod, [p=1:m-1, q=p+1:m, i=1:n-1, j=i+1:n], z[i,j] >= w[j] - w[i

#every player is assigned a group
@constraint(mod, [p=1:m], sum(x[i,p] for i=1:n)== k)
#a player is assigned to exaclty one group
@constraint(mod, [i=1:n], sum(x[i,p] for p=1:m)== 1)
@constraint(mod, [i in 1:m-1, p in i+1:m], x[i,p]==0)

@objective(mod, Min, sum(z[i,j] for i=1:n,j=1:n))

optimize!(mod)
```

```
Academic license - for non-commercial use only - expires 2022-08-19

┌ Warning: Passing optimizer attributes as keyword arguments to
│ Gurobi.Optimizer is deprecated. Use
│     MOI.set(model, MOI.RawParameter("key"), value)
│ or
│     JuMP.set_optimizer_attribute(model, "key", value)
│ instead.
└ @ Gurobi /Users/bennetthellman/.julia/packages/Gurobi/BAtIN/src/MOI_wra
pper/MOI_wrapper.jl:273
```

In [369]:
```
x=value.(x)
piv = x.*(players2.Points)
df1_ind = []
df2_ind = []
df3_ind = []
for i=1:30
    if piv[i,1] != 0
        push!(df1_ind,i)
    end
    if piv[i,2] != 0
        push!(df2_ind,i)
    end
    if piv[i,3] != 0
        push!(df3_ind,i)
    end
end
df1 = players2[df1_ind, :Points];
df2 = players2[df2_ind, :Points];
df3 = players2[df3_ind, :Points];
```

In [374]:
```
mu12 = abs(mean(df1[:,:])-mean(df2[:,:]))
mu13 = abs(mean(df1[:,:])-mean(df3[:,:]))
mu23 = abs(mean(df2[:,:])-mean(df3[:,:]))
var12 = abs(var(df1[:,:])*.9-var(df2[:,:])*.9)
var13 = abs(var(df1[:,:])*.9-var(df3[:,:])*.9)
var23 = abs(var(df2[:,:])*.9-var(df3[:,:])*.9)
print("|Mean_1-Mean_2| = ", mu12, "\n")
print("|Mean_1-Mean_3| = ", mu13, "\n")
print("|Mean_2-Mean_3| = ", mu23, "\n")
print("|Var_1-Var_2| = ", var12, "\n")
print("|Var_1-Var_3| = ", var13, "\n")
print("|Var_2-Var_3| = ", var23, "\n")

sum_mu = mu12+mu13+mu23
sum_var = var12+var13+var23

print("Max Discrepancy Mean = ", "0.153", "\n")
print("Max Discrepancy Variance = ", "0.865", "\n")
print("Mean Discrepancy of Mean = ", sum_mu/3, "\n")
print("Mean Discrepancy of Variance = ", (sum_var/3))
```

```
|Mean_1-Mean_2| = 0.15292440607760022
|Mean_1-Mean_3| = 0.05838675448766666
|Mean_2-Mean_3| = 0.09453765158993357
|Var_1-Var_2| = 0.8653149462364451
|Var_1-Var_3| = 0.013480780227936529
|Var_2-Var_3| = 0.8518341660085086
Max Discrepancy Mean = 0.153
Max Discrepancy Variance = 0.865
Mean Discrepancy of Mean = 0.10194960405173348
Mean Discrepancy of Variance = 0.5768766308242967
```

In [375]:
```
objective_value(mod)
```

Out[375]:
```
281.70433783565386
```

In [377]:
```
mu_vec = Vector{Float64}()
var_vec =  Vector{Float64}()
disc_mean = Vector{Float64}()
disc_var = Vector{Float64}()
for i in 1:10000
    Random.seed!(i)
    df = players2[shuffle(axes(players2, 1)), :]
    df1 = df[1:10, :]
    df2 = df[11:20, :]
    df3 = df[21:30, :]
    mu12 = abs(mean(df1[:,5])-mean(df2[:,5]))
    mu13 = abs(mean(df1[:,5])-mean(df3[:,5]))
    mu23 = abs(mean(df2[:,5])-mean(df3[:,5]))
    var12 = abs(var(df1[:,5])*.9-var(df2[:,5])*.9)
    var13 = abs(var(df1[:,5])*.9-var(df3[:,5])*.9)
    var23 = abs(var(df2[:,5])*.9-var(df3[:,5])*.9)
    sum_mu = mu12+mu13+mu23
    sum_var = var12+var13+var23
    append!(mu_vec, sum_mu)
    append!(var_vec, sum_var)
    append!(disc_mean, mu12)
    append!(disc_mean, mu13)
    append!(disc_mean, mu23)
    append!(disc_var, var12)
    append!(disc_var, var13)
    append!(disc_var, var23)
end
tot_vec = mu_vec+var_vec
Random.seed!(argmin(tot_vec))
df = players2[shuffle(axes(players2, 1)), :]
df1 = df[1:10, :];
df2 = df[11:20, :];
df3 = df[21:30, :];
```

Out[377]: 10 rows × 5 columns

	Column1	X	Player	Age	Points
	Int64	Int64	String	Int64	Float64
1	8	8	Casper Ruud\n	22	0.180363
2	24	24	John Isner	36	-0.651249
3	21	21	Gael Monfils	35	-0.572742
4	15	15	Dominic Thiem	28	-0.447224
5	5	5	Andrey Rublev\n	24	0.739785
6	6	6	Rafael Nadal	35	0.704527
7	2	2	Daniil Medvedev	25	2.00436
8	26	26	Reilly Opelka\n	24	-0.677105
9	28	28	Grigor Dimitrov\n	30	-0.740568
10	27	27	Lorenzo Sonego	26	-0.729286

In [380]:
```
sum_w = 0
for i in 1:10
    for j in 1:10
        sum_w = sum_w + abs(df1[i, :Points] - df2[j, :Points])
        sum_w = sum_w + abs(df1[i, :Points] - df3[j, :Points])
        sum_w = sum_w + abs(df3[i, :Points] - df3[j, :Points])
    end
end
print(sum_w)
```

```
282.6229184980536
```