MECH&AE 298 Mini-Project 2 Report

William Hsieh

February 16, 2025

1 Introduction

This mini-project infers the result of championship games between two top teams. It does so by analyzing regular season games, and then simulating 500 two-leg finals. A Bayesian statistical model is used in order to calculate the probabilities of each team winning, losing, or drawing a match. A multi-level model is used, indicating a hierarchical structure of parameters where teams "share" statistics.

2 Our Model

2.1 Multi-Level Model

First, the hyperparameters are set: the mean attack and defense are sampled from normal distributions, the variance of attack and defense are sampled from an exponential distribution, and the home advantage is sampled from a normal distribution. These statistics are used to determine the attack and defense for each of the 20 teams.

Compared to other methods of pooling, this multi-level model uses partial pooling. A model that uses no pooling would sample each team's attack and defense power independently, instead of using the statistics sampled from our hyperparameters. On the other hand, a model that uses complete pooling would end up with teams that share the same attack and defense power, instead of having slightly different attack and defense powers like we do here.

2.2 Description

The mean attacking power, mean defending power, variance of attacking power, variance of defending power, and home advantage are defined as follows:

$$\mu_{\text{att}} \sim \mathcal{N}(0, 0.1)$$
$$\mu_{\text{def}} \sim \mathcal{N}(0, 0.1)$$
$$\sigma_{\text{att}} \sim \exp(1)$$
$$\sigma_{\text{def}} \sim \exp(1)$$
$$\text{home} \sim \mathcal{N}(0, 1)$$

These are then used to create a distribution to sample from, resulting in each team's attack and defense power:

$$\text{att[team\_i]} \sim \mathcal{N}(\mu_{\text{att}}, \sigma_{\text{att}})$$
$$\text{def[team\_i]} \sim \mathcal{N}(\mu_{\text{def}}, \sigma_{\text{def}})$$

Finally, in a head to head, each team's attack and defense are used to calculate the win probabilities for each team:

$$\text{win home} = \text{home advantage} + \text{attack[home]} + \text{defense[away]} - \text{offset}$$
$$\text{win away} = \text{attack[away]} + \text{defense[home]} - \text{offset}$$

Note that offsets (calculated via $\mu_{\text{att}} + \mu_{\text{def}}$) are added to center the distribution. Finally, the scores can be calculated using the win probabilities:

$$\text{score} \sim \text{Poisson}\big(\exp(\text{win})\big)$$

2.3 Championship Games

After 3000 samples of matchups between the 20 teams, the best two teams are selected. Now, we can sample 500 "finals" between the two teams. Team A and team B take turns being the home team; the win probabilities and scores are calculated using the same methods as above.

However, there is one key difference when calculating the scores. Instead of updating the parameters for each team after each game, the parameters remain unchanged. The scores are instead sampled from the Poisson distribution 500 times for each team, to simulate 500 matches without changes in parameters:

$$\text{score} = \text{rand}\Big(\text{Poisson}\big(\exp(\text{win})\big), 500\Big)$$

The result is a list of scores, one for each team for the 150,000 (3000 × 500) championship games. (The 3000 comes from the previous step with the 20 teams.)

3 Results

Plotting the scores of team A (Manchester City) and team B (Manchester United) gives the following:

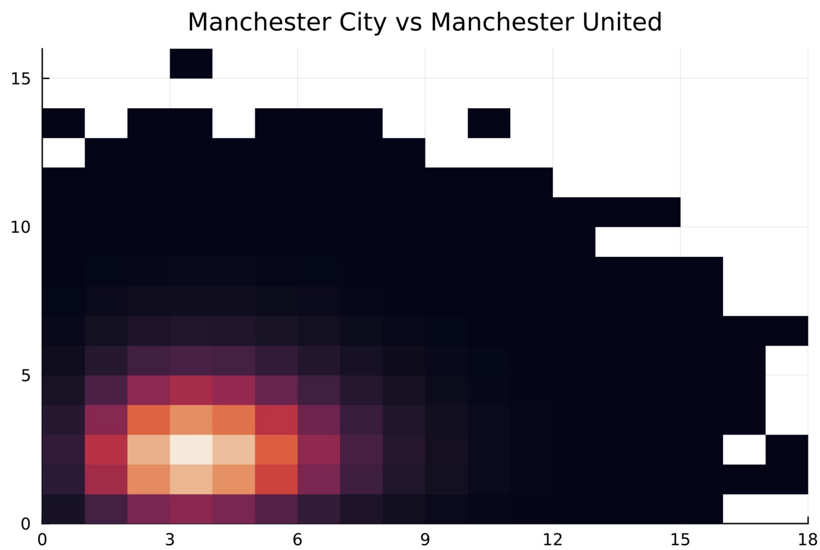


Figure 1: Heatmap of Scores

Each cell represents the number of points each team scored in a game. For example, the whitest spot is the most common, while the darker and non-populated spots show a low frequency. The one brightest spot represents a game where team A scored 3 points and team B scored 2 points, resulting in a win for team A.

By counting the number of occurrences where team A scored more than team B, we can figure out the win ratio for team A, and vice versa for team B. If they scored the same amount of points (represented by the diagonal starting at $(0, 0$, it results in a draw.

- Win probability for team A: 61.504%
- Win probability for team B: 24.090%
- Draw probability: 14.405%

As a sanity check, all of these probabilities add up to 100%.

4 Discussion and Conclusion

Based on the heatmap, we can see that most of the games had around 2-4 points, but in some rare cases could go up to 17 points. The games were also relatively close, with most of the probability mass around the slope $m = 1$. However, this small difference was enough to correspond to a win probability of 61.504% for team A and a much lower win probability of 24.090% for team B.

Overall, this mini-project analyzed games between 20 teams, picked the best 2 teams out from there, and simulated many championship games between these two teams. Bayesian statistics is used to calculate the win probabilities and scores for each game, and a multi-level model was used to convert raw team statistics into a quantifiable score.

5 Code

The Jupyter notebook used is appended to this report.

In [15]:

```julia
using JSON
using DataFrames
using StatsPlots
using Turing
using LinearAlgebra
using Random
```

Multi-level model using football match simulation as an example

In [16]:

```julia
## First, import the data and do some data wrangling

england_league = JSON.parsefile("../data/matches_England.json")

matches_df = DataFrame(home = [], away = [], score_home = [], score_aw
```

0×4 DataFrame

Row	home	away	score_home	score_away
	Any	Any	Any	Any

In [17]:

```julia
# example entry for each game in england_league:  "label" => "Burnley
matches = []
for match in england_league
    push!(matches, split(match["label"], ",")) # "Burnley - AFC Bourne
end

for match in matches
    home, away = split(match[1], " - ")  # "Burnley" # "AFC Bournemout
    score_home, score_away = split(match[2], " - ") # "1" # "2"
    push!(matches_df,[home, away, parse(Int,score_home), parse(Int,sco
end

matches_df

teams = unique(collect(matches_df[:,1]))
```

```
20-element Vector{Any}:
 "Burnley"
 "Crystal Palace"
 "Huddersfield Town"
 "Liverpool"
 "Manchester United"
 "Newcastle United"
 "Southampton"
 "Swansea City"
 "Tottenham Hotspur"
 "West Ham United"
 "Manchester City"
 "Leicester City"
 "Chelsea"
 "Arsenal"
 "Everton"
 "AFC Bournemouth"
 "Watford"
 "West Bromwich Albion"
 "Stoke City"
 "Brighton & Hove Albion"
```

In [ ]:

```
## Now, our model

@model function football_matches(home_teams, away_teams, score_home, s

    # Hyper priors
    μatt ~ Normal(0, 0.1)
    μdef ~ Normal(0, 0.1)
    σatt ~ Exponential(1)
    σdef ~ Exponential(1)
    home ~ Normal(0, 1)

    # Team-specific effects

    att = zeros(length(teams))
    def = zeros(length(teams))

    for i in 1:length(teams)
        att[i] ~ Normal(μatt, σatt)
        def[i] ~ Normal(μdef, σdef)
    end

    #att ~ filldist(Normal(μatt, σatt), length(teams))  # more compact
    #def ~ filldist(Normal(μdef, σdef), length(teams))

    offset = mean(att) + mean(def)

    # the number of matches
    n_matches = length(home_teams)

    # scoring rates θ
    θ_home = Vector{Real}(undef, n_matches)     # or just θ_home = zer
```

```julia
    θ_away = Vector{Real}(undef, n_matches)     # or just θ_away = zer

    # Modeling score-rate and scores for each match
    for i in 1:n_matches
        # scoring rate
        home_team_idx = findfirst(isequal(home_teams[i]), teams)
        away_team_idx = findfirst(isequal(away_teams[i]), teams)

        θ_home[i] = home + att[home_team_idx] + def[away_team_idx] - o
        θ_away[i] = att[away_team_idx] + def[home_team_idx] - offset

        # scores
        score_home[i] ~ Poisson(exp(θ_home[i]))  # To ensure positive
        score_away[i] ~ Poisson(exp(θ_away[i]))
    end
end
```

```
football_matches (generic function with 2 methods)
```

In [ ]:

```julia
model = football_matches(matches_df[:,1], matches_df[:,2], matches_df[

posterior = sample(model, NUTS(), 3000)
```

```
Sampling   0%|                                                  |  ETA: N/A
┌ Info: Found initial step size
│   ϵ = 0.00625
└ @ Turing.Inference /Users/lime/.julia/packages/Turing/vX5F9/src/mcmc/
hmc.jl:213
Sampling   0%|▏                                                 |  ETA: 0:06:16
Sampling   1%|▎                                                 |  ETA: 0:05:25
Sampling   2%|▍                                                 |  ETA: 0:04:34
Sampling   2%|▋                                                 |  ETA: 0:04:21
Sampling   2%|▊                                                 |  ETA: 0:04:09
Sampling   3%|▉                                                 |  ETA: 0:04:10
Sampling   4%|█                                                 |  ETA: 0:04:14
Sampling   4%|█▏                                                |  ETA: 0:04:11
Sampling   4%|█▎                                                |  ETA: 0:04:04
Sampling   5%|█▌                                                |  ETA: 0:03:54
Sampling   6%|█▋                                                |  ETA: 0:03:48
Sampling   6%|█▊                                                |  ETA: 0:03:43
Sampling   6%|█▉                                                |  ETA: 0:03:44
Sampling   7%|██                                                |  ETA: 0:03:40
Sampling   8%|██▏                                               |  ETA: 0:03:35
Sampling   8%|██▍                                               |  ETA: 0:03:30
Sampling   8%|██▌                                               |  ETA: 0:03:24
Sampling   9%|██▋                                               |  ETA: 0:03:25
Sampling  10%|██▊                                               |  ETA: 0:03:21
Sampling  10%|██▉                                               |  ETA: 0:03:17
Sampling  10%|███                                               |  ETA: 0:03:13
Sampling  11%|███▎                                              |  ETA: 0:03:10
Sampling  12%|███▍                                              |  ETA: 0:03:08
Sampling  12%|███▌                                              |  ETA: 0:03:06
Sampling  12%|███▋                                              |  ETA: 0:03:07
```

```
Sampling  91%|                                                  |  ETA: 0:00:12
Sampling  92%|                                                  |  ETA: 0:00:12
Sampling  92%|                                                  |  ETA: 0:00:11
Sampling  92%|                                                  |  ETA: 0:00:10
Sampling  93%|                                                  |  ETA: 0:00:10
Sampling  94%|                                                  |  ETA: 0:00:09
Sampling  94%|                                                  |  ETA: 0:00:08
Sampling  94%|                                                  |  ETA: 0:00:08
Sampling  95%|                                                  |  ETA: 0:00:07
Sampling  96%|                                                  |  ETA: 0:00:06
Sampling  96%|                                                  |  ETA: 0:00:05
Sampling  96%|                                                  |  ETA: 0:00:05
Sampling  97%|                                                  |  ETA: 0:00:04
Sampling  98%|                                                  |  ETA: 0:00:03
Sampling  98%|                                                  |  ETA: 0:00:03
Sampling  98%|                                                  |  ETA: 0:00:02
Sampling  99%|                                                  |  ETA: 0:00:01
Sampling 100%|                                                  |  ETA: 0:00:01
Sampling 100%|                                                  | Time: 0:02:16
Sampling 100%|                                                  | Time: 0:02:16
Chains MCMC chain (3000×57×1 Array{Float64, 3}):

Iterations        = 1001:1:4000
Number of chains  = 1
Samples per chain = 3000
Wall duration     = 136.29 seconds
Compute duration  = 136.29 seconds
parameters        = μatt, μdef, σatt, σdef, home, att[1], def[1], att[
2], def[2], att[3], def[3], att[4], def[4], att[5], def[5], att[6], de
f[6], att[7], def[7], att[8], def[8], att[9], def[9], att[10], def[10],
att[11], def[11], att[12], def[12], att[13], def[13], att[14], def[14],
att[15], def[15], att[16], def[16], att[17], def[17], att[18], def[18],
att[19], def[19], att[20], def[20]
internals         = lp, n_steps, is_accept, acceptance_rate, log_densit
y, hamiltonian_energy, hamiltonian_energy_error, max_hamiltonian_energy
_error, tree_depth, numerical_error, step_size, nom_step_size

Summary Statistics
  parameters      mean       std      mcse    ess_bulk    ess_tail
rhat   ⋯
      Symbol   Float64   Float64   Float64     Float64     Float64   Fl
oat64  ⋯


        μatt   -0.0106    0.1006    0.0059    286.0900    721.8933
1.0155   ⋯
        μdef   -0.0008    0.0994    0.0059    288.4936    464.7421
1.0031   ⋯
        σatt    0.3849    0.0779    0.0035    621.2962    339.6839
1.0023   ⋯
        σdef    0.2154    0.0612    0.0034    327.5896    117.0460
1.0028   ⋯
        home    0.3380    0.0425    0.0009   2208.4531   2263.3756
```

```
1.0004   ⋯
      att[1]   -0.2672    0.2014    0.0100    409.4098    955.1314
1.0069   ⋯
      def[1]   -0.1680    0.1656    0.0079    421.9315    441.1022
1.0004   ⋯
      att[2]   -0.0720    0.1883    0.0100    355.6188    657.8336
1.0078   ⋯
      def[2]    0.0594    0.1556    0.0078    400.4448   1113.5101
1.0011   ⋯
      att[3]   -0.4594    0.2072    0.0103    402.8109   1040.9419
1.0095   ⋯
      def[3]    0.0862    0.1563    0.0078    406.0630    531.2944
1.0042   ⋯
      att[4]    0.5008    0.1743    0.0101    295.1097    771.1988
1.0159   ⋯
      def[4]   -0.1551    0.1678    0.0081    425.6426    571.6784
0.9998   ⋯
      att[5]    0.2910    0.1777    0.0102    306.1388    709.1792
1.0109   ⋯
      def[5]   -0.3300    0.1843    0.0084    445.9616    167.5677
0.9999   ⋯
      att[6]   -0.1918    0.1969    0.0104    360.3907    940.1000
1.0115   ⋯
      def[6]   -0.0497    0.1541    0.0069    500.0782    991.6641
1.0016   ⋯
      ⋮           ⋮         ⋮         ⋮          ⋮           ⋮          ⋮
⋱
                                                   1 column and 28 row
s omitted

Quantiles
  parameters      2.5%     25.0%     50.0%     75.0%     97.5%
      Symbol   Float64   Float64   Float64   Float64   Float64

        μatt   -0.2005   -0.0769   -0.0106    0.0569    0.1855
        μdef   -0.2022   -0.0655    0.0009    0.0668    0.1866
        σatt    0.2633    0.3281    0.3729    0.4302    0.5561
        σdef    0.0804    0.1757    0.2123    0.2506    0.3494
        home    0.2516    0.3101    0.3398    0.3673    0.4177
      att[1]   -0.6797   -0.3978   -0.2636   -0.1333    0.1242
      def[1]   -0.4877   -0.2831   -0.1685   -0.0515    0.1470
      att[2]   -0.4397   -0.1989   -0.0713    0.0565    0.2913
      def[2]   -0.2495   -0.0467    0.0640    0.1688    0.3637
      att[3]   -0.8748   -0.5914   -0.4588   -0.3222   -0.0598
      def[3]   -0.2396   -0.0133    0.0914    0.1876    0.3825
      att[4]    0.1529    0.3881    0.5069    0.6142    0.8376
      def[4]   -0.4933   -0.2669   -0.1504   -0.0392    0.1603
      att[5]   -0.0623    0.1790    0.2905    0.4048    0.6297
      def[5]   -0.6938   -0.4537   -0.3274   -0.2027    0.0213
      att[6]   -0.5740   -0.3206   -0.1891   -0.0596    0.1925
      def[6]   -0.3658   -0.1516   -0.0457    0.0545    0.2445
      ⋮           ⋮         ⋮         ⋮         ⋮         ⋮
```

28 rows omitted

In [20]:
```
posterior_df=DataFrame(posterior)
```

3000×59 DataFrame *2975 rows omitted*

Row	iteration	chain	μatt	μdef	σatt	σdef	home
	Int64	Int64	Float64	Float64	Float64	Float64	Float6
1	1001	1	-0.0340967	0.0488872	0.371805	0.341548	0.41
2	1002	1	0.120224	0.0379764	0.442336	0.104175	0.318
3	1003	1	0.120224	0.0379764	0.442336	0.104175	0.318
4	1004	1	0.14776	0.0602034	0.431715	0.100922	0.329
5	1005	1	0.14776	0.0602034	0.431715	0.100922	0.329
6	1006	1	0.14776	0.0602034	0.431715	0.100922	0.329
7	1007	1	0.132189	0.0570596	0.422664	0.106005	0.358
8	1008	1	0.140849	0.0218837	0.340951	0.149594	0.302
9	1009	1	0.000211701	0.00791739	0.55847	0.0972845	0.407
10	1010	1	-0.0436966	-0.0107217	0.500536	0.108407	0.4
11	1011	1	0.198782	0.0145961	0.331272	0.195929	0.260
12	1012	1	-0.00990157	0.133114	0.629202	0.263415	0.343
13	1013	1	0.143073	0.0929881	0.423556	0.258046	0.315
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2989	3989	1	-0.152038	0.000856577	0.322627	0.227176	0.2
2990	3990	1	0.0130078	0.0471447	0.487366	0.20001	0.379
2991	3991	1	-0.120245	-0.0915178	0.35016	0.21365	0.365
2992	3992	1	-0.0803213	-0.0308569	0.334981	0.158142	0.40
2993	3993	1	-0.0574519	-0.0464465	0.347634	0.225209	0.308
2994	3994	1	0.10894	-0.106624	0.42737	0.23224	0.381
2995	3995	1	0.00348895	-0.0755697	0.39041	0.162871	0.317
2996	3996	1	0.0239133	-0.128041	0.357514	0.194984	0.395
2997	3997	1	0.180213	-0.0651568	0.314539	0.16265	0.37
2998	3998	1	0.0579803	0.0572436	0.451031	0.199167	0.369
2999	3999	1	0.176402	0.0165819	0.444763	0.227071	0.329
3000	4000	1	-0.0406021	-0.0143352	0.464827	0.241888	0.33

In [21]:
```
DataFrames.transform!(posterior_df, AsTable(Between("att[1]","att[20]"
```

```
DataFrames.transform!(posterior_df, AsTable(Between("def[1]","def[20]"
DataFrames.transform!(posterior_df, AsTable([:att_mean,:def_mean]) =>
```

3000×62 DataFrame *2975 rows omitted*

Row	iteration	chain	μatt	μdef	σatt	σdef	home
	Int64	Int64	Float64	Float64	Float64	Float64	Float6
1	1001	1	-0.0340967	0.0488872	0.371805	0.341548	0.41
2	1002	1	0.120224	0.0379764	0.442336	0.104175	0.318
3	1003	1	0.120224	0.0379764	0.442336	0.104175	0.318
4	1004	1	0.14776	0.0602034	0.431715	0.100922	0.329
5	1005	1	0.14776	0.0602034	0.431715	0.100922	0.329
6	1006	1	0.14776	0.0602034	0.431715	0.100922	0.329
7	1007	1	0.132189	0.0570596	0.422664	0.106005	0.358
8	1008	1	0.140849	0.0218837	0.340951	0.149594	0.302
9	1009	1	0.000211701	0.00791739	0.55847	0.0972845	0.407
10	1010	1	-0.0436966	-0.0107217	0.500536	0.108407	0.4
11	1011	1	0.198782	0.0145961	0.331272	0.195929	0.260
12	1012	1	-0.00990157	0.133114	0.629202	0.263415	0.34
13	1013	1	0.143073	0.0929881	0.423556	0.258046	0.31
⋮	⋮	⋮	⋮	⋮	⋮	⋮	
2989	3989	1	-0.152038	0.000856577	0.322627	0.227176	0.2
2990	3990	1	0.0130078	0.0471447	0.487366	0.20001	0.379
2991	3991	1	-0.120245	-0.0915178	0.35016	0.21365	0.365
2992	3992	1	-0.0803213	-0.0308569	0.334981	0.158142	0.40
2993	3993	1	-0.0574519	-0.0464465	0.347634	0.225209	0.308
2994	3994	1	0.10894	-0.106624	0.42737	0.23224	0.381
2995	3995	1	0.00348895	-0.0755697	0.39041	0.162871	0.317
2996	3996	1	0.0239133	-0.128041	0.357514	0.194984	0.395
2997	3997	1	0.180213	-0.0651568	0.314539	0.16265	0.37
2998	3998	1	0.0579803	0.0572436	0.451031	0.199167	0.369
2999	3999	1	0.176402	0.0165819	0.444763	0.227071	0.329
3000	4000	1	-0.0406021	-0.0143352	0.464827	0.241888	0.33

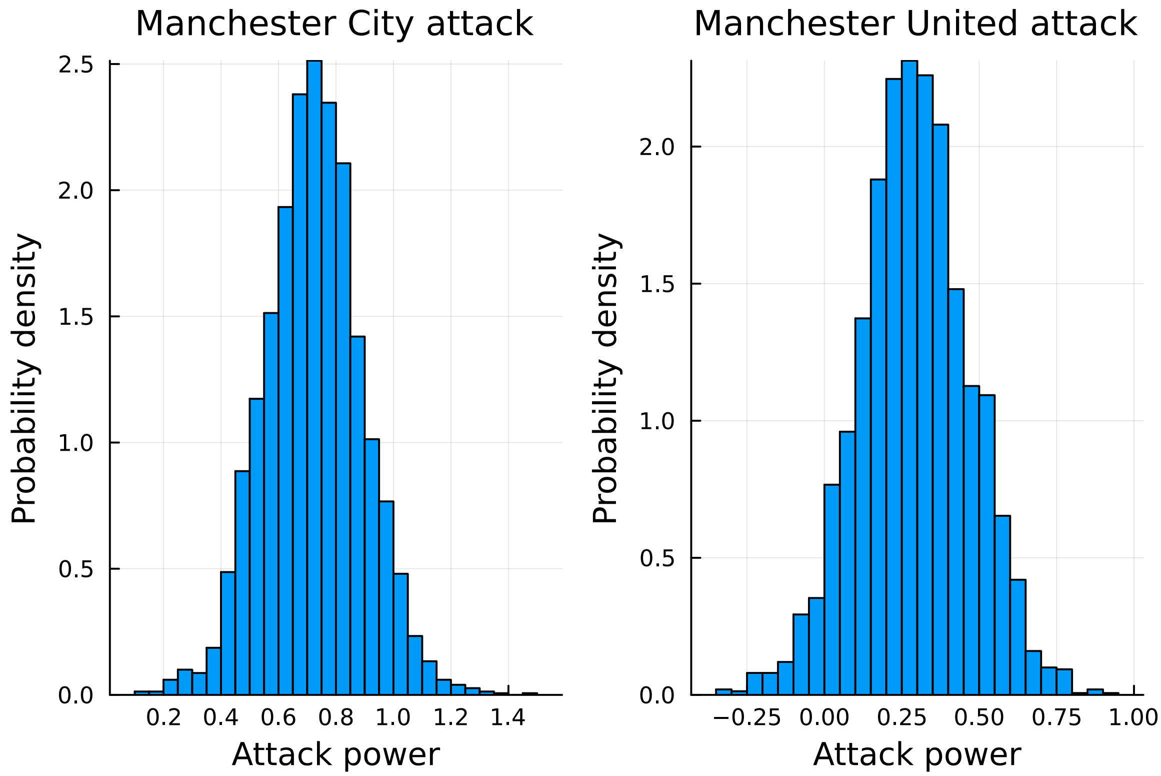
In [22]:
```
# For this example, we are interested in a pair of teams (no need to u
```

```julia
teamA = "Manchester City"
teamB = "Manchester United"

teamA_id = findfirst(isequal(teamA), teams)
teamB_id = findfirst(isequal(teamB), teams)

teamA_att_post = posterior_df[:,"att[$teamA_id]"]
teamA_def_post = posterior_df[:,"def[$teamA_id]"]

teamB_att_post = posterior_df[:,"att[$teamB_id]"]
teamB_def_post = posterior_df[:,"def[$teamB_id]"]
```

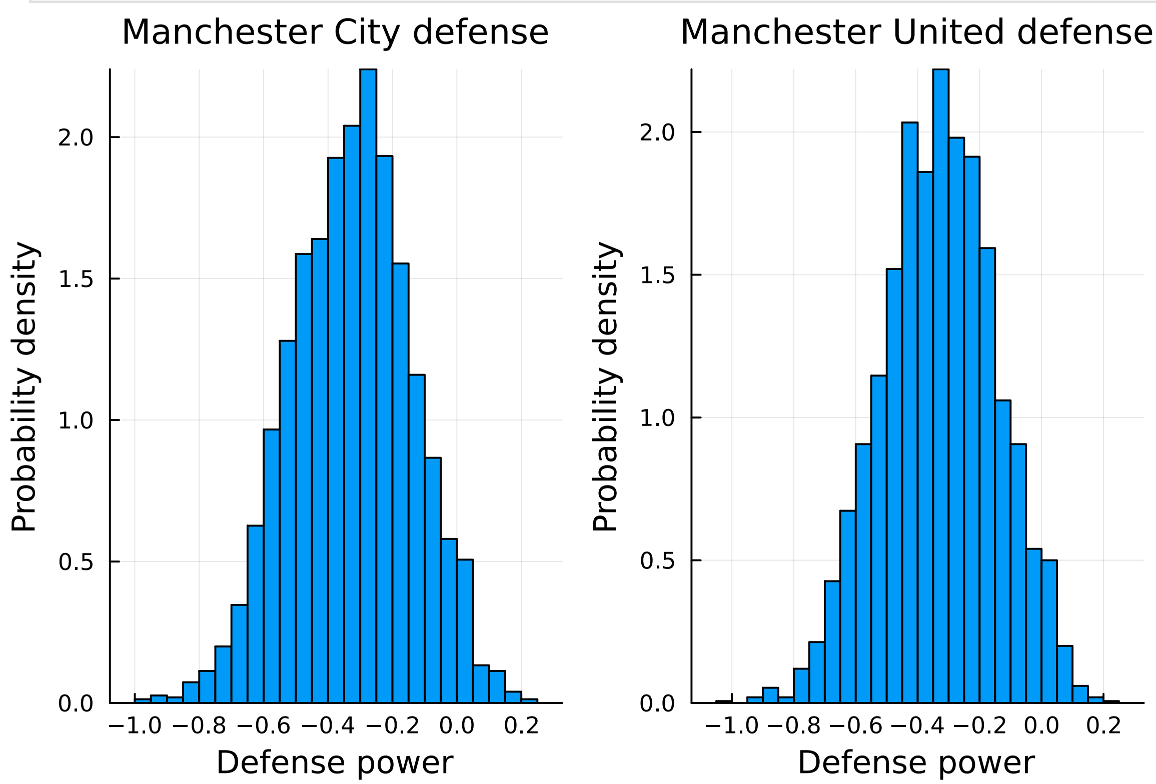
```
3000-element Vector{Float64}:
 -0.31659686077845584
  0.010909344846748792
  0.010909344846748792
 -0.005602923863187977
 -0.005602923863187977
 -0.005602923863187977
 -0.12470192624633054
 -0.23946056821044293
 -0.17673201401029642
 -0.12582298768177858
  ⋮
 -0.28907656726617503
 -0.600319104560645
 -0.25490945685899014
 -0.4968341292721567
 -0.2274802264310944
 -0.7068032570364412
 -0.04770233420003743
 -0.03970682618439173
 -0.5052118038597926
```

In [23]:

```julia
ha1 = histogram(teamA_att_post, title=teamA*" attack", titlefontsize =
ha2 = histogram(teamB_att_post, title=teamB*" attack", titlefontsize =
plot(ha1, ha2, layout=(1,2));
xlabel!("Attack power");
ylabel!("Probability density")
```



In [24]:
```
hd1 = histogram(teamA_def_post, title=teamA*" defense", titlefontsize
hd2 = histogram(teamB_def_post, title=teamB*" defense", titlefontsize
plot(hd1, hd2, layout=(1,2));
xlabel!("Defense power");
ylabel!("Probability density")
```

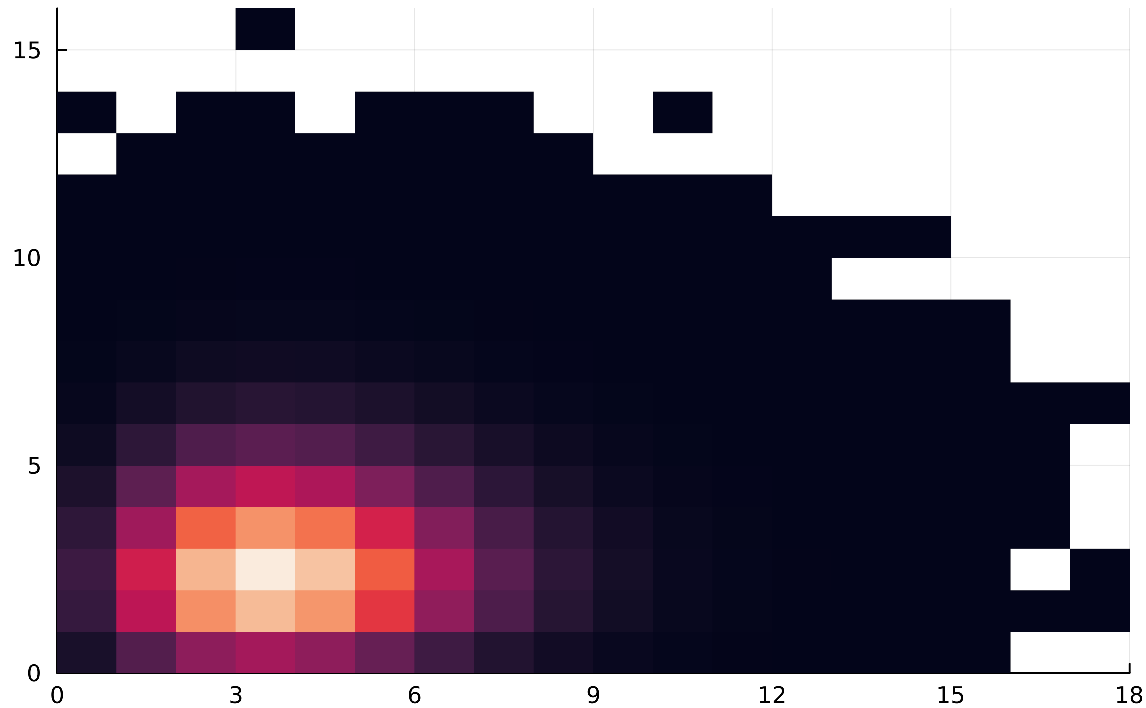


Mini Project

Consult the lecture notes.

In [ ]:
```
Random.seed!(205579184)
# hint: let's simulate 500 hypothetical finals (then you will have a t

# first leg: teamA is the home team and teamB is the away team

θ_home = posterior_df[:,:home] + posterior_df[:,"att[$teamA_id]"] + po
θ_away = posterior_df[:,"att[$teamB_id]"] + posterior_df[:,"def[$teamA

teamA_score = rand.(Poisson.(exp.(θ_home)),500)
teamB_score = rand.(Poisson.(exp.(θ_away)),500)

# second leg: teamA is the away team and teamB is the home team

θ_home = posterior_df[:,:home] + posterior_df[:,"att[$teamB_id]"] + po
θ_away = posterior_df[:,"att[$teamA_id]"] + posterior_df[:,"def[$teamB

teamA_score += rand.(Poisson.(exp.(θ_away)),500)  # add the first-leg
teamB_score += rand.(Poisson.(exp.(θ_home)),500)

# transform into long column vectors
teamA_score = vcat(teamA_score...)
teamB_score = vcat(teamB_score...)

display(histogram2d(teamA_score, teamB_score, title=teamA*" vs "*teamB
# https://docs.juliaplots.org/dev/generated/colorschemes/

# Winning probabilities
winning_prob_A = sum(teamA_score .> teamB_score) / length(teamA_score)
println("Winning probability of "*teamA*" against "*teamB*" is "*strin

winning_prob_B = sum(teamA_score .< teamB_score) / length(teamA_score)
println("Winning probability of "*teamB*" against "*teamA*" is "*strin

draw_prob = sum(teamA_score .== teamB_score) / length(teamA_score)
println("Draw probability between "*teamA*" and "*teamB*" is "*string(

println("Sum of probabilities (sanity check): "*string((winning_prob_A
```

Manchester City vs Manchester United




```
Winning probability of Manchester City against Manchester United is 61.
504%
Winning probability of Manchester United against Manchester City is 24.
09%
Draw probability between Manchester City and Manchester United is 14.40
5%
Sum of probabilities (sanity check): 100.0%
```