Estimating the shooting efficiency of top NBA point guard

Xiang Gao, Jerome Finn, Nilesh Malpekar

2017/12/03

Purpose

Background

As we all know,a point guard controls the court by passing the ball to the player who is wide open. He is a decision maker to deliver assists or finish the attack by himself. The question arises, who has the highest shooting percentage among the top NBA point guards. Is this related to the professional years of experience?

We are using logistic regression to assess the probablity of shooting, also use a binomial model to estimate field goals made by the NBA players. In order to build a hierachical model, each player is treated as a individual group by introducing a random effect called player effect. What's more, recent three years data will be used to check the continuous improvement.

Data

Original data

Original data is retrieved from Kaggle competition site NBA Dataset. Using our homework datasets as a guide, Xiang got our data set to manageable level for our questions. We concentrate on players and years with players representing groups similar to how rats where used as groups in our previous lectures.

The zip file contains two separate CSV files:

- Seasons_Stats.csv - season specific data since 1950
- Players.csv - player specific data

```
season_data <- read.table("Seasons_Stats.csv", header=TRUE, sep = ",", quote = '"')
```

For this project, we are focusing on specific fields within the `season_data` dataset which are described beow:

Datasource	Field_Name	Description
Seasons_Stats	Year	NBA year
Seasons_Stats	Player	Player name
Seasons_Stats	Pos	Player position
Seasons_Stats	FG	Field Goals
Seasons_Stats	FGA	Field Goals Attempted

The dataset contains duplicate rows for multiple players for the same year. As part of the data preparation, we have removed duplicate rows based on **Year** and **Player**.

```
season_data <- season_data[with(season_data, order(Year, Player, -FG)), ]
season_data <- distinct(season_data, Year, Player, .keep_all = TRUE)
```

Feature creation

For this project, we need to extract following two features from the original dataset

- **Experience** as number of years of NBA experience.
- **FG%** as $FieldGoals/FieldGoalsAttempted$

```
season_data <- season_data %>% group_by(Player) %>% mutate(EXP = 1:n())
season_data[, "FG%"] <- season_data$FG / season_data$FGA
```

Preparing modeling data

For our project, we decided to use the data for the last 3 NBA seasons, i.e. 2017, 2016 and 2015.

The data so far is in long format, i.e. for each player there is one row per year. However, we need daa in wide format so that there is a single row per player and year specific attributes should be columns in the dataset. As an example for **FG** (Field Goals) columns should be as follows:

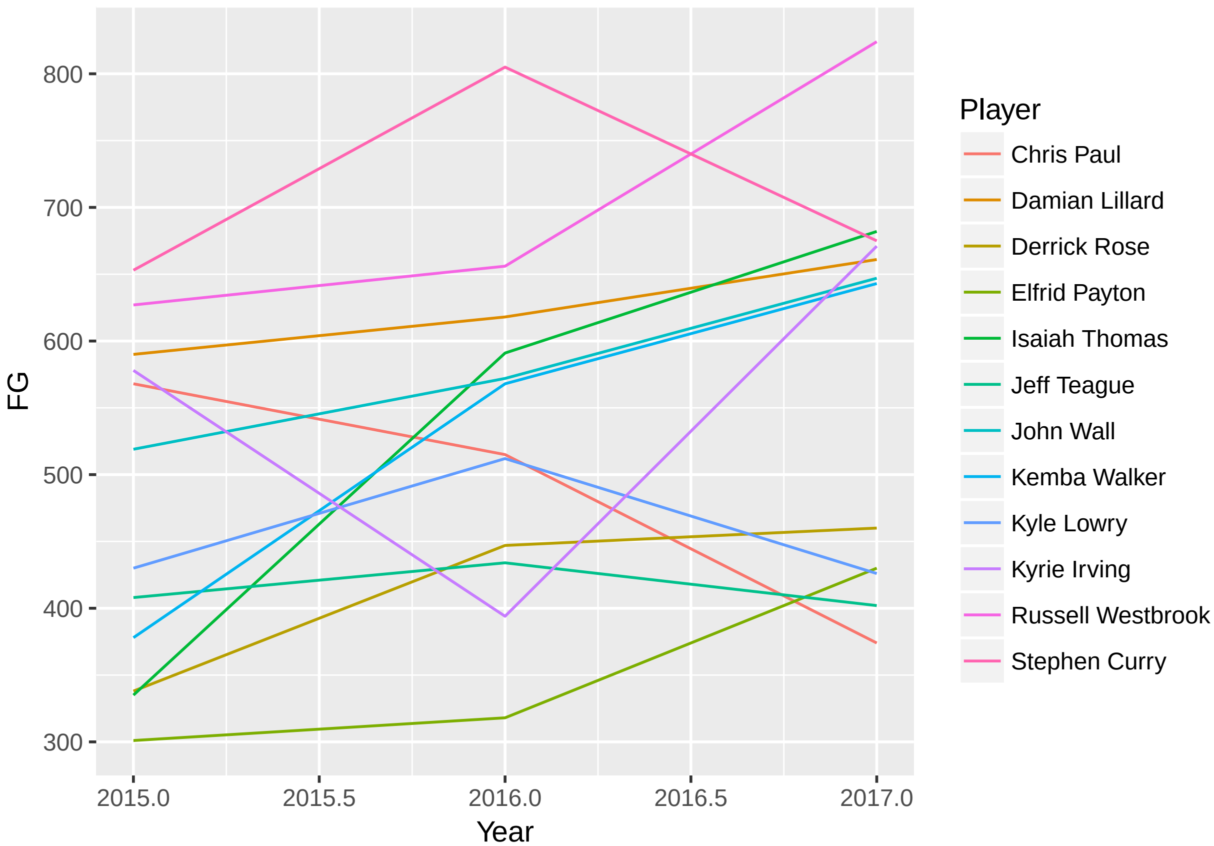
- latest year: FG_0
- 1st prior year: FG_PRIOR_1
- Nth prior year: FG_PRIOR_N

```
# get modeling data
model_data <- get_model_data_wide(season_data, INTERESTED_YEARS,
                INTERESTED_POSITIONS, 300)
model_data_row_count <- nrow(model_data)
display_column_names <- c("Player", get_column_names("FG", YEAR_COUNT),
                get_column_names("FGA", YEAR_COUNT))
head(model_data[, display_column_names])
```

```
##            Player FG_0 FG_PRIOR_1 FG_PRIOR_2 FGA_0 FGA_PRIOR_1 FGA_PRIOR_2
## 1     Chris Paul  374        515        568   785        1114        1170
## 2 Damian Lillard  661        618        590  1488        1474        1360
## 3   Derrick Rose  460        447        338   977        1048         835
## 4  Elfrid Payton  430        318        301   912         730         708
## 5  Isaiah Thomas  682        591        335  1473        1382         797
## 6    Jeff Teague  402        434        408   909         988         887
```

Modeling data visualization

The chart below shows field goals for each player per year. For most player there is growth in field goals from previous year to next year.



Model

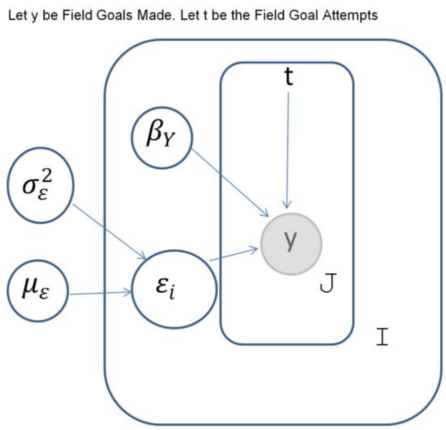
Notation

$$y_{ij} = \text{Shooting rate of player i at year j } x_1 = 2017,\ x_2 = 2016,\ x_3 = 2015$$

$$y_i \mid \beta,\ X_i \quad \text{indep. Bin}(n_i,\ p_i)\ \text{logit}(p_i) = X_i\beta + \epsilon_i,$$

$$\epsilon_i \quad \text{iid N}(0,\ \sigma_\epsilon^2)$$

DAG Model



Code

JAGS model. I will use scaled-t1 on coefficients in beta. and a flat uniform distribution for sigma of player effect

```
data {
  dimY <- dim(FGM)
}
model {
  for (i in 1:dimY[1]) { ## row per player; total 8 players
    for (j in 1:dimY[2]) { ## column per year; total 3 years i.e. 2017, 2016, 2015
      FGM[i,j] ~ dbin(prob[i,j], FGA[i,j])
      logit(prob[i,j]) <- beta.Year[i]*Yr.Exper[i,j]+Player.Effect[i]
      FGMrep[i,j] ~ dbin(prob[i,j],FGA[i,j])
    }
    beta.Year[i] ~ dt(0,0.16,1)
    Player.Effect[i] ~ dnorm(mu, 1/sigmaPE^2)
  }
  mu ~ dt(0,0.01,1)
  sigmaPE ~ dunif(0,100)
}
```

Computation

Prepare data binding

Subset out the FGM(field goal made),FGA(field goal attempt),Yr.Exper(Years of professional experience)

```
d1 <- list(FGM = model_data[, get_column_names("FG", YEAR_COUNT)],
           FGA = model_data[,get_column_names("FGA", YEAR_COUNT)],
```

```
          Yr.Exper = model_data[,get_column_names("EXP", YEAR_COUNT)])

inits1 <- list(list(beta.Year=rep(-1, model_data_row_count), mu=10, sigmaPE=0.001,
                    .RNG.name = "base::Marsaglia-Multicarry", .RNG.seed = 123),
               list(beta.Year=rep(-1, model_data_row_count), mu=-10, sigmaPE=99,
                    .RNG.name = "base::Marsaglia-Multicarry", .RNG.seed = 234),
               list(beta.Year=rep(-1, model_data_row_count), mu=10, sigmaPE=99,
                    .RNG.name = "base::Marsaglia-Multicarry", .RNG.seed = 345),
               list(beta.Year=rep(-1, model_data_row_count), mu=-10, sigmaPE=0.01,
                    .RNG.name = "base::Marsaglia-Multicarry", .RNG.seed = 456))
```

Build model

```
m1 <- jags.model('model-logistic.bug', d1, inits = inits1, n.chains = 4, n.adapt = 1000)
```

```
## Compiling data graph
##    Resolving undeclared variables
##    Allocating nodes
##    Initializing
##    Reading data back into data table
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 36
##    Unobserved stochastic nodes: 62
##    Total graph size: 288
##
## Initializing model
```

Burn-ins and check for convergence

We tried various values to speed convergence. None lead to very fast convergence but the above values, after much trial and error, were finally acceptable. Still it took a burn-in of over 1 million iterations to get convergence.

```
update(m1, 1024000)
```

Posterior samples and Gelman Statistic

```
x1 <- coda.samples(m1,c('beta.Year','Player.Effect'), n.iter = 70000)
```

```
g.d <- gelman.diag(x1, autoburnin = F)
```

Gelman-Rubin statistic value is 1.0077374.

For details on individual parameters and Gelman plots, please refer to the appendix.

Effective samples sizes are adequate.

```
e.s <- effectiveSize(x1)
all(e.s > 400)
```

```
## [1] TRUE
```

For details on individual sample sizes, please refer to the appendix.

Retrieve replicate dataset and probabilities

```
x2 <- coda.samples(m1, c('beta.Year','Player.Effect','prob','FGMrep'),
                   n.iter = 70000, thin=40)
```

Model Assessment

Coda summary

```
s.x2 <- summary(x2)
```

Beta statistics

	Mean	SD	Naive SE	Time-series SE
Player.Effect[1]	-0.2535138	0.3025547	0.0036162	0.0102671
Player.Effect[2]	-0.3759121	0.1392040	0.0016638	0.0023955
Player.Effect[3]	-0.7880552	0.2914805	0.0034839	0.0089302
Player.Effect[4]	-0.4203693	0.1045817	0.0012500	0.0013642
Player.Effect[5]	-0.6054766	0.1960335	0.0023430	0.0042727
Player.Effect[6]	-0.1965309	0.2557812	0.0030572	0.0068957
Player.Effect[7]	-0.3682344	0.1958464	0.0023408	0.0044500
Player.Effect[8]	-0.7344583	0.2063455	0.0024663	0.0059110
Player.Effect[9]	-0.7281658	0.3304472	0.0039496	0.0113746
Player.Effect[10]	-0.2737947	0.1743541	0.0020839	0.0035304
Player.Effect[11]	-0.3038040	0.2188216	0.0026154	0.0058350
Player.Effect[12]	-0.0652149	0.2486316	0.0029717	0.0078824
beta.Year[1]	0.0138154	0.0279565	0.0003341	0.0009190
beta.Year[2]	0.0256190	0.0341016	0.0004076	0.0005824
beta.Year[3]	0.0755970	0.0412197	0.0004927	0.0012368
beta.Year[4]	0.0977752	0.0470167	0.0005620	0.0006651
beta.Year[5]	0.0708450	0.0375604	0.0004489	0.0008247

	Mean	SD	Naive SE	Time-series SE
beta.Year[6]	-0.0027595	0.0363294	0.0004342	0.0009739
beta.Year[7]	0.0209617	0.0321592	0.0003844	0.0007293
beta.Year[8]	0.0829499	0.0398182	0.0004759	0.0011314
beta.Year[9]	0.0463668	0.0331953	0.0003968	0.0011394
beta.Year[10]	0.0258893	0.0341098	0.0004077	0.0006856
beta.Year[11]	0.0044956	0.0269657	0.0003223	0.0007218
beta.Year[12]	0.0011180	0.0352344	0.0004211	0.0011111

Beta Quantiles

	2.5%	25%	50%	75%	97.5%
Player.Effect[1]	-0.7939002	-0.4510983	-0.2834109	-0.0732006	0.4216359
Player.Effect[2]	-0.6506571	-0.4687184	-0.3749317	-0.2836372	-0.0982935
Player.Effect[3]	-1.3994506	-0.9792988	-0.7650130	-0.5693854	-0.3105092
Player.Effect[4]	-0.6266516	-0.4895738	-0.4211129	-0.3519107	-0.2155998
Player.Effect[5]	-1.0091113	-0.7351999	-0.5936589	-0.4673938	-0.2592812
Player.Effect[6]	-0.6484707	-0.3766422	-0.2173076	-0.0327701	0.3525163
Player.Effect[7]	-0.7402195	-0.5001791	-0.3766062	-0.2435686	0.0379135
Player.Effect[8]	-1.1546913	-0.8704581	-0.7292564	-0.5890170	-0.3616296
Player.Effect[9]	-1.4647884	-0.9336924	-0.6853600	-0.4840256	-0.1905313
Player.Effect[10]	-0.6021396	-0.3924766	-0.2794194	-0.1605832	0.0772087
Player.Effect[11]	-0.7287097	-0.4446624	-0.3189520	-0.1591141	0.1543301
Player.Effect[12]	-0.4983440	-0.2463964	-0.0717557	0.0995105	0.4492143
beta.Year[1]	-0.0488601	-0.0029749	0.0165657	0.0322507	0.0643880
beta.Year[2]	-0.0411567	0.0031245	0.0254155	0.0485526	0.0924443
beta.Year[3]	0.0072156	0.0447896	0.0726601	0.1029926	0.1613756
beta.Year[4]	0.0037381	0.0665928	0.0981593	0.1285646	0.1904097
beta.Year[5]	0.0037487	0.0439081	0.0691230	0.0956596	0.1476653
beta.Year[6]	-0.0801957	-0.0258506	-0.0001444	0.0227185	0.0606714
beta.Year[7]	-0.0459187	0.0006116	0.0217723	0.0424610	0.0818939
beta.Year[8]	0.0091428	0.0549798	0.0821329	0.1090257	0.1639401
beta.Year[9]	-0.0082050	0.0222705	0.0421992	0.0670346	0.1199708
beta.Year[10]	-0.0432642	0.0033008	0.0270595	0.0493144	0.0896647
beta.Year[11]	-0.0513154	-0.0132249	0.0064790	0.0221301	0.0562648
beta.Year[12]	-0.0708188	-0.0220978	0.0025195	0.0265261	0.0627473

For details on the coda summary, please refer to the appendix.

Check overdispersion, chi-square discrepancy

No over dispersion problem as 0.2894286.

Marginal posterior p-value

Here we are checking the marginal posterior predictive p-value of (FGMrep[i] > y[i]). Effectively, comparing replicated data to our model's actual data. The closer we get to 1 or 0, the more the model is off.

Player	pValue.2017	pValue.2016	pValue.2015
Chris Paul	0.5361429	0.7684286	0.2205714
Damian Lillard	0.3728571	0.8154286	0.3244286
Derrick Rose	0.2265714	0.7015714	0.7278571
Elfrid Payton	0.4547143	0.6694286	0.4341429
Isaiah Thomas	0.3341429	0.7401429	0.5031429
Jeff Teague	0.5771429	0.6611429	0.2821429
John Wall	0.3701429	0.8488571	0.2860000
Kemba Walker	0.4494286	0.3687143	0.7775714
Kyle Lowry	0.1962857	0.6658571	0.7172857
Kyrie Irving	0.4631429	0.8075714	0.2820000
Russell Westbrook	0.7668571	0.0890000	0.6601429
Stephen Curry	0.8551429	0.1078571	0.4728571

We can see that the model looks good for Elfrid Payton, but for the other players we are not too precise. The evidence is not strong that years of experience effect a higher rate of field goals made per field goals attempted.

Results

Density of Various Player through the Years

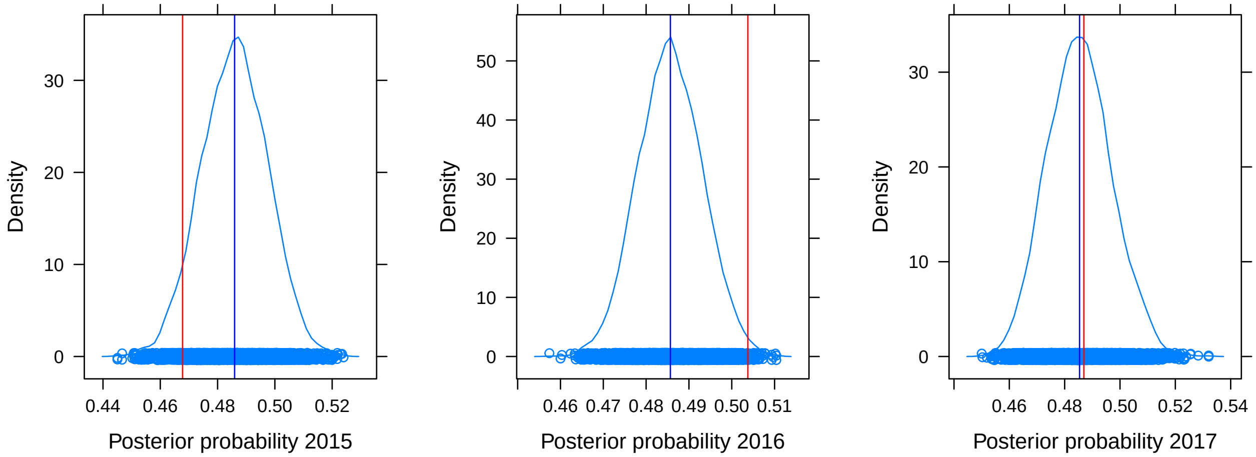
If we look as Stephen Curry's density plots we see no improvement in his performance over the 3 years examined. This was the case with most of our players

```
player_row_id <- which(model_data$Player == "Stephen Curry")
# posterior prob
posterior <- get_player_posterior_probs(df, model_data, player_row_id)
df_posterior_observed <- get_player_posterior_vs_observed(model_data, player_row_id, posterior)
kable(df_posterior_observed)
```

	posterior	observed
YEAR_0	0.4859435	0.4677755
YEAR_PRIOR_1	0.4856599	0.5037547
YEAR_PRIOR_2	0.4853849	0.4869500

The posterior density does not show Stephen's improvement of making a field goal, let's also check Russell Westbrook.

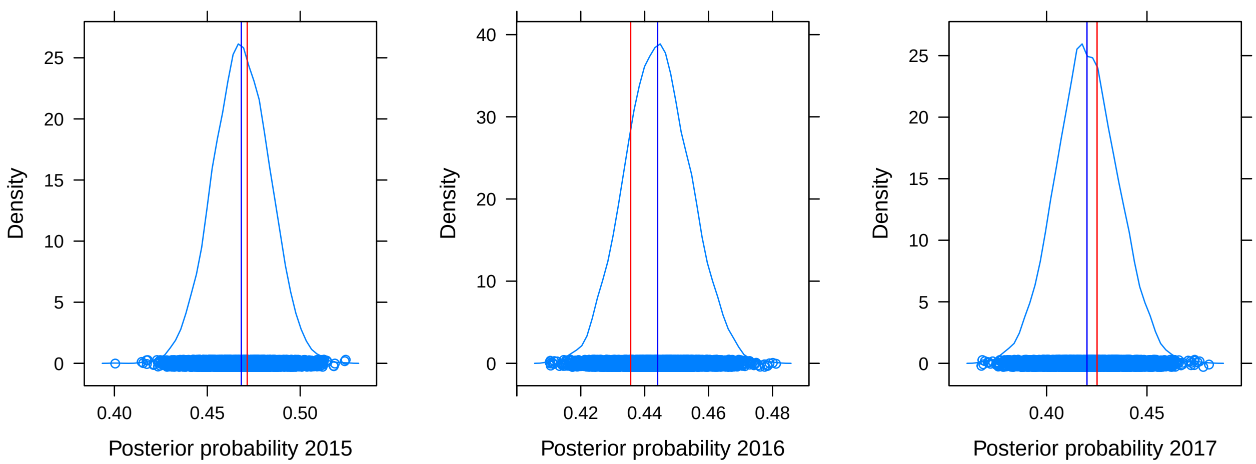
```
plot_player_posterior_probs(model_data, player_row_id, posterior)
```



Check Elfrid Payton successfully makes an attempted field goal for the past three years.

	posterior	observed
YEAR_0	0.4683113	0.4714912
YEAR_PRIOR_1	0.4440544	0.4356164
YEAR_PRIOR_2	0.4201235	0.4251412

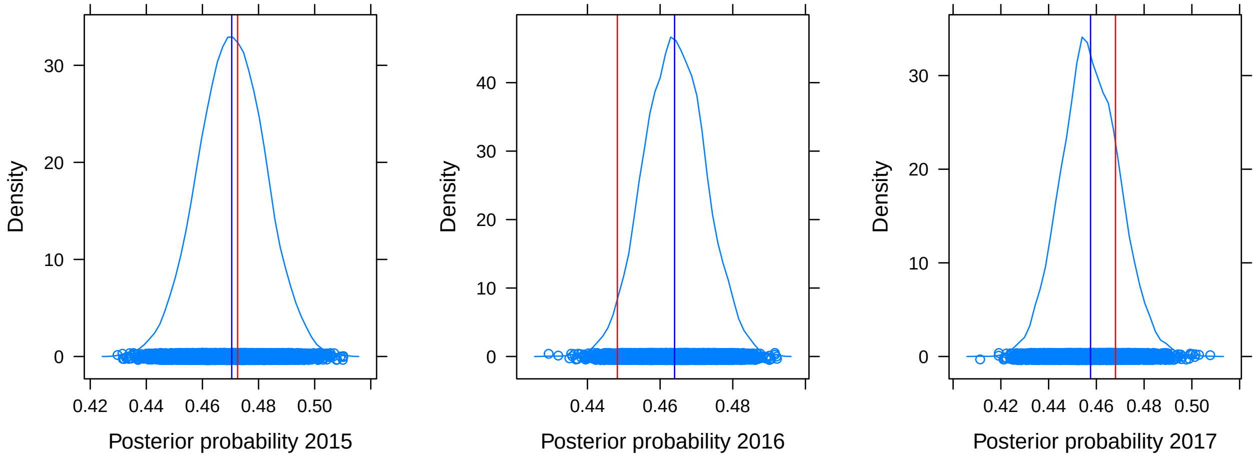
```
plot_player_posterior_probs(model_data, player_row_id, posterior)
```



Check Kyrie Irving successfully makes an attempted field goal for the past three years.

	posterior	observed
YEAR_0	0.4704366	0.4725352
YEAR_PRIOR_1	0.4639856	0.4482366
YEAR_PRIOR_2	0.4575674	0.4680162

```r
plot_player_posterior_probs(model_data, player_row_id, posterior)
```



Posterior Odds

Here we show the posterior odds of each player improving from one year to the next. We take our poster sample for each player, and take the mean of comparing one year's vector being greater than the previous. As we can see the odds are not extreme that a player may improve from one year to the next, nor do we have a clear pattern. The model does not support the proposition that players improve from one year to another.

Player	2016-2017	2015-2016
Chris Paul	0.7222857	0.2777143
Damian Lillard	0.7787143	0.2212857
Derrick Rose	0.9864286	0.0135714
Elfrid Payton	0.9798571	0.0201429
Isaiah Thomas	0.9801429	0.0198571
Jeff Teague	0.4988571	0.5011429
John Wall	0.7538571	0.2461429
Kemba Walker	0.9892857	0.0107143
Kyle Lowry	0.9492857	0.0507143
Kyrie Irving	0.7782857	0.2217143
Russell Westbrook	0.5915714	0.4084286
Stephen Curry	0.5268571	0.4731429

Conclusion

This project shows that the shooting accuracy doesn't have a statistically significant relationship with years of professional experience. The player's personal effect is still the major impact. This is related to the individual player to make a better decision not to pass but taking over and make an attempted basket.

Contributions

Xiang deserves a bulk of the credit as the idea was his and did the data gathering and model design, as well a first pass at much of the analysis. We all contributed to the final analysis, although Nilesh did much to improve the R coding. Jerry also contributed to the analysis and lead on much of the early work with regards to putting together the proposal and video presenation with the team's input.

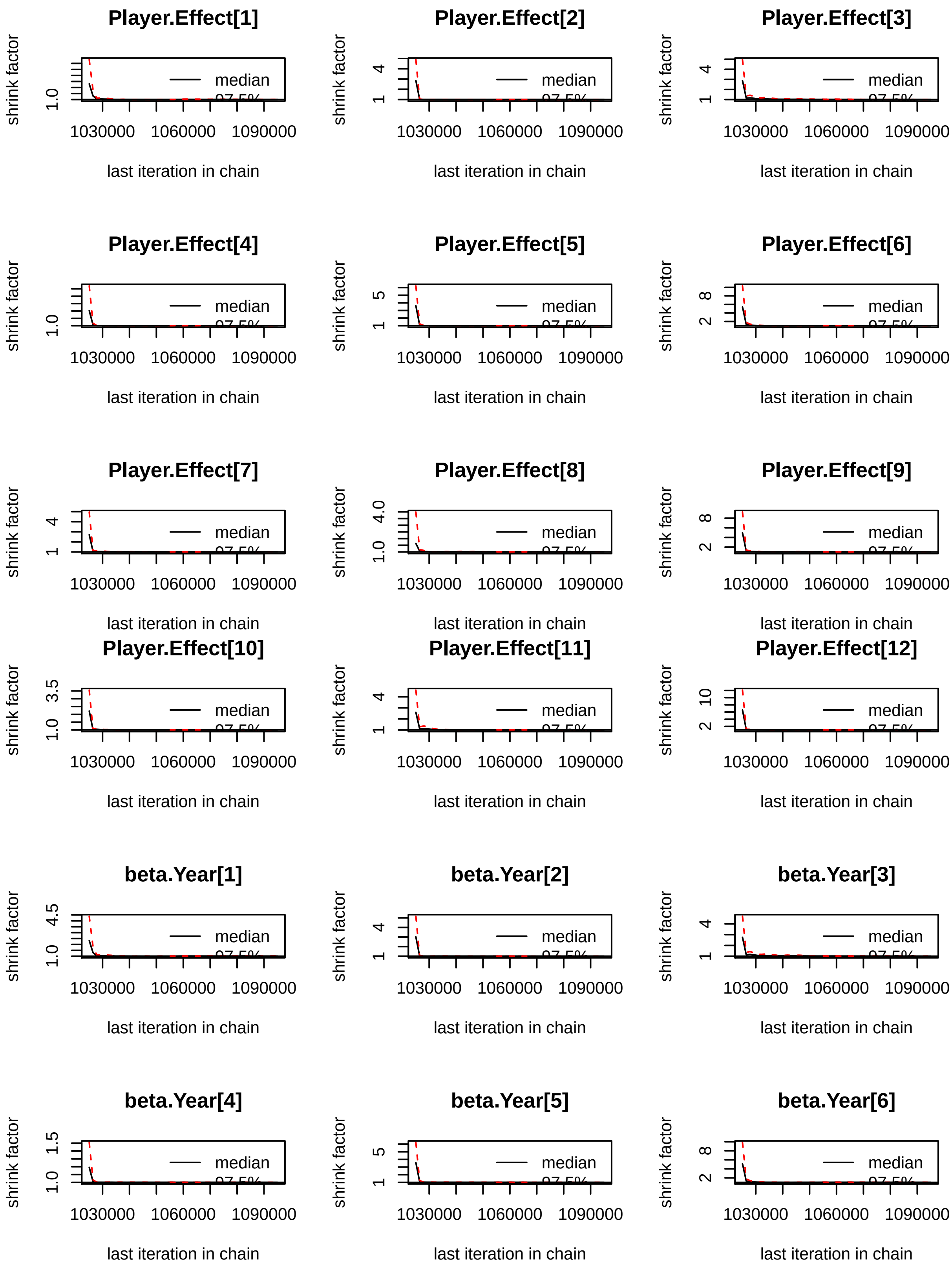
References

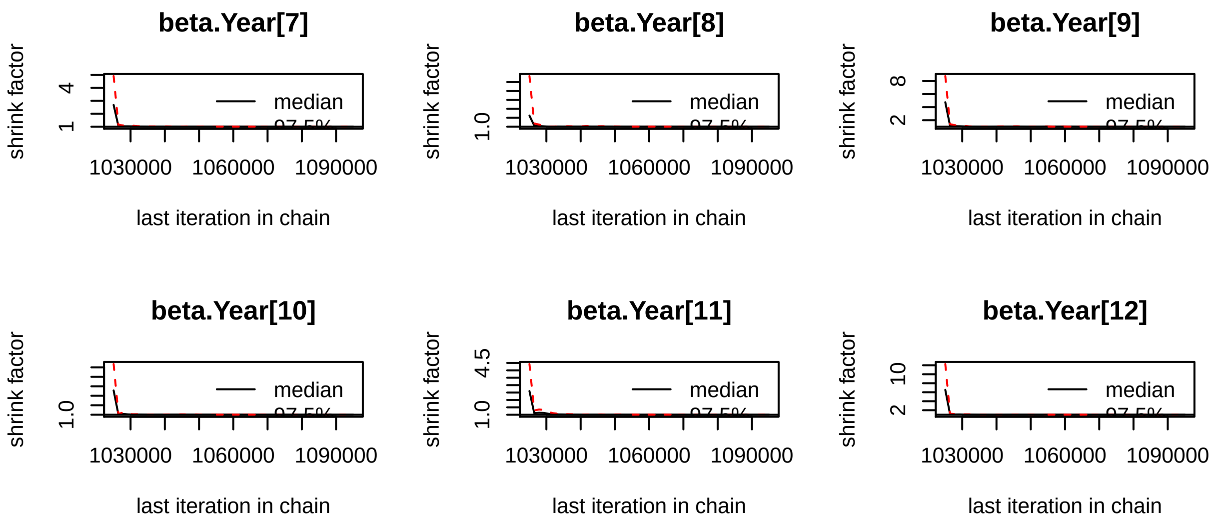
- NBA players stats since 1950
- NBA Dataset
- Basketball reference glossary

Appendix

Gelman-Rubin statistic details

```
##                     Point est. Upper C.I.
## Player.Effect[1]      1.006857   1.019355
## Player.Effect[2]      1.000415   1.001087
## Player.Effect[3]      1.003305   1.009762
## Player.Effect[4]      1.000042   1.000086
## Player.Effect[5]      1.000505   1.001069
## Player.Effect[6]      1.002985   1.007620
## Player.Effect[7]      1.001810   1.004738
## Player.Effect[8]      1.000666   1.001401
## Player.Effect[9]      1.002136   1.003347
## Player.Effect[10]     1.000159   1.000362
## Player.Effect[11]     1.002447   1.006951
## Player.Effect[12]     1.001361   1.003628
## beta.Year[1]          1.006823   1.019268
## beta.Year[2]          1.000406   1.001064
## beta.Year[3]          1.003301   1.009750
## beta.Year[4]          1.000035   1.000072
## beta.Year[5]          1.000500   1.001065
## beta.Year[6]          1.002954   1.007538
## beta.Year[7]          1.001826   1.004826
## beta.Year[8]          1.000626   1.001333
## beta.Year[9]          1.002096   1.003284
## beta.Year[10]         1.000165   1.000375
## beta.Year[11]         1.002455   1.006976
## beta.Year[12]         1.001352   1.003574
```


Player.Effect[1]
Player.Effect[2]
Player.Effect[3]
Player.Effect[4]
Player.Effect[5]
Player.Effect[6]
Player.Effect[7]
Player.Effect[8]
Player.Effect[9]
Player.Effect[10]
Player.Effect[11]
Player.Effect[12]
beta.Year[1]
beta.Year[2]
beta.Year[3]
beta.Year[4]
beta.Year[5]
beta.Year[6]
shrink factor
median
1030000
1060000
1090000
last iteration in chain



Effective sample size details

```
##  Player.Effect[1]  Player.Effect[2]  Player.Effect[3]  Player.Effect[4]
##         1174.4830         4283.2318         1314.7603        13356.9430
##  Player.Effect[5]  Player.Effect[6]  Player.Effect[7]  Player.Effect[8]
##         2434.5999         1609.3560         2235.2799         2084.1916
##  Player.Effect[9] Player.Effect[10] Player.Effect[11] Player.Effect[12]
##          994.6579         3106.0883         1506.9399         1211.5094
##      beta.Year[1]      beta.Year[2]      beta.Year[3]      beta.Year[4]
##         1172.1036         4240.7869         1307.8506        13693.7725
##      beta.Year[5]      beta.Year[6]      beta.Year[7]      beta.Year[8]
##         2451.1078         1625.0236         2262.1143         2105.0415
##      beta.Year[9]     beta.Year[10]     beta.Year[11]     beta.Year[12]
##         1000.2167         3167.7392         1513.0312         1224.3861
```

Coda summary details

```
##
## Iterations = 1095040:1165000
## Thinning interval = 40
## Number of chains = 4
## Sample size per chain = 1750
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                          Mean        SD  Naive SE Time-series SE
## FGMrep[1,1]       375.122286 17.001590 2.032e-01      2.721e-01
## FGMrep[2,1]       652.298000 25.656134 3.066e-01      3.319e-01
## FGMrep[3,1]       444.063857 20.376175 2.435e-01      3.442e-01
## FGMrep[4,1]       427.216571 20.356768 2.433e-01      2.462e-01
```

```
## FGMrep[5,1]       670.793857 25.282034 3.022e-01      3.423e-01
## FGMrep[6,1]       405.016286 18.979276 2.268e-01      2.760e-01
## FGMrep[7,1]       638.608857 24.431601 2.920e-01      3.296e-01
## FGMrep[8,1]       639.083571 25.609596 3.061e-01      4.286e-01
## FGMrep[9,1]       409.369143 18.878312 2.256e-01      3.223e-01
## FGMrep[10,1]      667.823000 25.138564 3.005e-01      3.384e-01
## FGMrep[11,1]      843.426571 28.249026 3.376e-01      4.401e-01
## FGMrep[12,1]      701.245143 25.201645 3.012e-01      4.397e-01
## FGMrep[1,2]       528.840714 19.728415 2.358e-01      2.394e-01
## FGMrep[2,2]       637.256571 21.932840 2.621e-01      2.598e-01
## FGMrep[3,2]       456.535286 18.891972 2.258e-01      2.351e-01
## FGMrep[4,2]       324.212429 15.321031 1.831e-01      1.838e-01
## FGMrep[5,2]       604.741714 22.048638 2.635e-01      2.634e-01
## FGMrep[6,2]       440.912143 18.107066 2.164e-01      2.165e-01
## FGMrep[7,2]       593.325857 21.140539 2.527e-01      2.497e-01
## FGMrep[8,2]       560.152000 21.144879 2.527e-01      2.527e-01
## FGMrep[9,2]       520.388857 20.199956 2.414e-01      2.479e-01
## FGMrep[10,2]      407.926857 16.406078 1.961e-01      2.015e-01
## FGMrep[11,2]      626.224286 21.580651 2.579e-01      2.550e-01
## FGMrep[12,2]      776.065143 23.035547 2.753e-01      2.827e-01
## FGMrep[1,3]       551.310429 21.059175 2.517e-01      3.191e-01
## FGMrep[2,3]       578.615571 23.654495 2.827e-01      3.097e-01
## FGMrep[3,3]       348.538429 18.147629 2.169e-01      3.300e-01
## FGMrep[4,3]       297.606571 17.335748 2.072e-01      2.072e-01
## FGMrep[5,3]       334.753429 17.247409 2.061e-01      2.853e-01
## FGMrep[6,3]       396.547714 18.674799 2.232e-01      3.190e-01
## FGMrep[7,3]       506.713429 21.203735 2.534e-01      3.130e-01
## FGMrep[8,3]       392.822000 20.046489 2.396e-01      3.619e-01
## FGMrep[9,3]       441.110286 19.879559 2.376e-01      3.557e-01
## FGMrep[10,3]      564.999000 22.670008 2.710e-01      3.145e-01
## FGMrep[11,3]      636.089429 23.854299 2.851e-01      3.641e-01
## FGMrep[12,3]      650.967714 23.878247 2.854e-01      4.944e-01
## Player.Effect[1]   -0.253514  0.302555 3.616e-03      1.027e-02
## Player.Effect[2]   -0.375912  0.139204 1.664e-03      2.396e-03
## Player.Effect[3]   -0.788055  0.291481 3.484e-03      8.930e-03
## Player.Effect[4]   -0.420369  0.104582 1.250e-03      1.364e-03
## Player.Effect[5]   -0.605477  0.196034 2.343e-03      4.273e-03
## Player.Effect[6]   -0.196531  0.255781 3.057e-03      6.896e-03
## Player.Effect[7]   -0.368234  0.195846 2.341e-03      4.450e-03
## Player.Effect[8]   -0.734458  0.206345 2.466e-03      5.911e-03
## Player.Effect[9]   -0.728166  0.330447 3.950e-03      1.137e-02
## Player.Effect[10]  -0.273795  0.174354 2.084e-03      3.530e-03
## Player.Effect[11]  -0.303804  0.218822 2.615e-03      5.835e-03
## Player.Effect[12]  -0.065215  0.248632 2.972e-03      7.882e-03
## beta.Year[1]        0.013815  0.027957 3.341e-04      9.190e-04
## beta.Year[2]        0.025619  0.034102 4.076e-04      5.824e-04
## beta.Year[3]        0.075597  0.041220 4.927e-04      1.237e-03
## beta.Year[4]        0.097775  0.047017 5.620e-04      6.651e-04
```

```
## beta.Year[5]         0.070845  0.037560 4.489e-04      8.247e-04
## beta.Year[6]        -0.002760  0.036329 4.342e-04      9.739e-04
## beta.Year[7]         0.020962  0.032159 3.844e-04      7.293e-04
## beta.Year[8]         0.082950  0.039818 4.759e-04      1.131e-03
## beta.Year[9]         0.046367  0.033195 3.968e-04      1.139e-03
## beta.Year[10]        0.025889  0.034110 4.077e-04      6.856e-04
## beta.Year[11]        0.004496  0.026966 3.223e-04      7.218e-04
## beta.Year[12]        0.001118  0.035234 4.211e-04      1.111e-03
## prob[1,1]            0.478096  0.012418 1.484e-04      2.909e-04
## prob[2,1]            0.438394  0.011418 1.365e-04      1.622e-04
## prob[3,1]            0.454342  0.013727 1.641e-04      3.066e-04
## prob[4,1]            0.468311  0.015189 1.815e-04      1.884e-04
## prob[5,1]            0.455044  0.011343 1.356e-04      1.879e-04
## prob[6,1]            0.445603  0.013173 1.574e-04      2.388e-04
## prob[7,1]            0.444877  0.011073 1.323e-04      1.783e-04
## prob[8,1]            0.441120  0.012015 1.436e-04      2.340e-04
## prob[9,1]            0.445717  0.012642 1.511e-04      3.072e-04
## prob[10,1]           0.470437  0.011861 1.418e-04      1.868e-04
## prob[11,1]           0.434566  0.009479 1.133e-04      1.809e-04
## prob[12,1]           0.485943  0.011517 1.377e-04      2.795e-04
## prob[1,2]            0.474644  0.009004 1.076e-04      1.138e-04
## prob[2,2]            0.432080  0.007547 9.020e-05      9.043e-05
## prob[3,2]            0.435662  0.009235 1.104e-04      1.103e-04
## prob[4,2]            0.444054  0.010206 1.220e-04      1.234e-04
## prob[5,2]            0.437532  0.008145 9.735e-05      1.007e-04
## prob[6,2]            0.446266  0.009347 1.117e-04      1.117e-04
## prob[7,2]            0.439694  0.007786 9.306e-05      9.003e-05
## prob[8,2]            0.420769  0.008265 9.878e-05      1.034e-04
## prob[9,2]            0.434279  0.008820 1.054e-04      1.022e-04
## prob[10,2]           0.463986  0.008396 1.003e-04      1.014e-04
## prob[11,2]           0.433451  0.007062 8.441e-05      8.763e-05
## prob[12,2]           0.485660  0.007488 8.950e-05      9.402e-05
## prob[1,3]            0.471204  0.010247 1.225e-04      2.114e-04
## prob[2,3]            0.425826  0.011107 1.328e-04      1.749e-04
## prob[3,3]            0.417217  0.013625 1.629e-04      3.556e-04
## prob[4,3]            0.420124  0.015611 1.866e-04      2.009e-04
## prob[5,3]            0.420222  0.013143 1.571e-04      2.501e-04
## prob[6,3]            0.446963  0.012733 1.522e-04      2.921e-04
## prob[7,3]            0.434555  0.011133 1.331e-04      2.131e-04
## prob[8,3]            0.400748  0.013338 1.594e-04      3.143e-04
## prob[9,3]            0.422945  0.011335 1.355e-04      2.676e-04
## prob[10,3]           0.457567  0.011997 1.434e-04      2.101e-04
## prob[11,3]           0.432361  0.009875 1.180e-04      2.080e-04
## prob[12,3]           0.485385  0.011585 1.385e-04      3.282e-04
##
## 2. Quantiles for each variable:
##
##                          2.5%        25%        50%       75%     97.5%
```

```
## FGMrep[1,1]        342.000000  3.630e+02  3.750e+02 387.00000 408.00000
## FGMrep[2,1]        601.000000  6.350e+02  6.530e+02 669.00000 702.00000
## FGMrep[3,1]        405.000000  4.300e+02  4.440e+02 458.00000 484.00000
## FGMrep[4,1]        387.000000  4.130e+02  4.270e+02 441.00000 467.00000
## FGMrep[5,1]        622.000000  6.540e+02  6.700e+02 688.00000 720.00000
## FGMrep[6,1]        367.000000  3.930e+02  4.050e+02 418.00000 442.00000
## FGMrep[7,1]        591.000000  6.220e+02  6.390e+02 655.00000 687.00000
## FGMrep[8,1]        589.000000  6.220e+02  6.390e+02 656.00000 689.00000
## FGMrep[9,1]        372.000000  3.970e+02  4.090e+02 422.00000 447.00000
## FGMrep[10,1]       618.000000  6.510e+02  6.680e+02 685.00000 718.00000
## FGMrep[11,1]       787.000000  8.250e+02  8.430e+02 863.00000 899.00000
## FGMrep[12,1]       653.000000  6.840e+02  7.010e+02 718.00000 751.00000
## FGMrep[1,2]        491.000000  5.160e+02  5.290e+02 542.00000 568.00000
## FGMrep[2,2]        594.000000  6.220e+02  6.370e+02 652.00000 681.00000
## FGMrep[3,2]        420.000000  4.440e+02  4.570e+02 469.00000 494.00000
## FGMrep[4,2]        294.000000  3.140e+02  3.240e+02 334.00000 354.00000
## FGMrep[5,2]        561.000000  5.900e+02  6.050e+02 620.00000 649.00000
## FGMrep[6,2]        405.000000  4.290e+02  4.410e+02 453.00000 477.00000
## FGMrep[7,2]        552.000000  5.790e+02  5.935e+02 608.00000 634.00000
## FGMrep[8,2]        519.000000  5.460e+02  5.600e+02 575.00000 601.00000
## FGMrep[9,2]        482.000000  5.070e+02  5.200e+02 534.00000 560.00000
## FGMrep[10,2]       376.000000  3.970e+02  4.080e+02 419.00000 441.00000
## FGMrep[11,2]       584.000000  6.120e+02  6.260e+02 640.00000 669.00000
## FGMrep[12,2]       731.000000  7.600e+02  7.760e+02 792.00000 821.00000
## FGMrep[1,3]        510.000000  5.370e+02  5.510e+02 566.00000 593.00000
## FGMrep[2,3]        532.000000  5.630e+02  5.780e+02 594.00000 625.00000
## FGMrep[3,3]        313.000000  3.360e+02  3.480e+02 361.00000 383.00000
## FGMrep[4,3]        264.000000  2.860e+02  2.980e+02 309.00000 332.00000
## FGMrep[5,3]        302.000000  3.230e+02  3.350e+02 346.00000 369.00000
## FGMrep[6,3]        361.000000  3.840e+02  3.960e+02 409.00000 433.00000
## FGMrep[7,3]        465.000000  4.930e+02  5.070e+02 521.00000 548.00000
## FGMrep[8,3]        354.000000  3.790e+02  3.930e+02 406.00000 432.02500
## FGMrep[9,3]        402.000000  4.280e+02  4.410e+02 454.00000 480.00000
## FGMrep[10,3]       520.000000  5.500e+02  5.650e+02 580.00000 610.00000
## FGMrep[11,3]       589.000000  6.200e+02  6.360e+02 652.00000 683.00000
## FGMrep[12,3]       605.000000  6.350e+02  6.510e+02 667.00000 698.00000
## Player.Effect[1]    -0.793900 -4.511e-01 -2.834e-01  -0.07320   0.42164
## Player.Effect[2]    -0.650657 -4.687e-01 -3.749e-01  -0.28364  -0.09829
## Player.Effect[3]    -1.399451 -9.793e-01 -7.650e-01  -0.56939  -0.31051
## Player.Effect[4]    -0.626652 -4.896e-01 -4.211e-01  -0.35191  -0.21560
## Player.Effect[5]    -1.009111 -7.352e-01 -5.937e-01  -0.46739  -0.25928
## Player.Effect[6]    -0.648471 -3.766e-01 -2.173e-01  -0.03277   0.35252
## Player.Effect[7]    -0.740219 -5.002e-01 -3.766e-01  -0.24357   0.03791
## Player.Effect[8]    -1.154691 -8.705e-01 -7.293e-01  -0.58902  -0.36163
## Player.Effect[9]    -1.464788 -9.337e-01 -6.854e-01  -0.48403  -0.19053
## Player.Effect[10]   -0.602140 -3.925e-01 -2.794e-01  -0.16058   0.07721
## Player.Effect[11]   -0.728710 -4.447e-01 -3.190e-01  -0.15911   0.15433
## Player.Effect[12]   -0.498344 -2.464e-01 -7.176e-02   0.09951   0.44921
```

```
## beta.Year[1]       -0.048860 -2.975e-03  1.657e-02   0.03225   0.06439
## beta.Year[2]       -0.041157  3.124e-03  2.542e-02   0.04855   0.09244
## beta.Year[3]        0.007216  4.479e-02  7.266e-02   0.10299   0.16138
## beta.Year[4]        0.003738  6.659e-02  9.816e-02   0.12856   0.19041
## beta.Year[5]        0.003749  4.391e-02  6.912e-02   0.09566   0.14767
## beta.Year[6]       -0.080196 -2.585e-02 -1.444e-04   0.02272   0.06067
## beta.Year[7]       -0.045919  6.116e-04  2.177e-02   0.04246   0.08189
## beta.Year[8]        0.009143  5.498e-02  8.213e-02   0.10903   0.16394
## beta.Year[9]       -0.008205  2.227e-02  4.220e-02   0.06703   0.11997
## beta.Year[10]      -0.043264  3.301e-03  2.706e-02   0.04931   0.08966
## beta.Year[11]      -0.051315 -1.322e-02  6.479e-03   0.02213   0.05626
## beta.Year[12]      -0.070819 -2.210e-02  2.520e-03   0.02653   0.06275
## prob[1,1]           0.453051  4.699e-01  4.784e-01   0.48658   0.50180
## prob[2,1]           0.415844  4.308e-01  4.383e-01   0.44611   0.46083
## prob[3,1]           0.428253  4.450e-01  4.539e-01   0.46345   0.48261
## prob[4,1]           0.438522  4.580e-01  4.683e-01   0.47863   0.49797
## prob[5,1]           0.433167  4.473e-01  4.550e-01   0.46262   0.47749
## prob[6,1]           0.419167  4.369e-01  4.458e-01   0.45435   0.47109
## prob[7,1]           0.423036  4.377e-01  4.449e-01   0.45241   0.46691
## prob[8,1]           0.418007  4.328e-01  4.409e-01   0.44927   0.46487
## prob[9,1]           0.421693  4.371e-01  4.454e-01   0.45405   0.47143
## prob[10,1]          0.447139  4.624e-01  4.705e-01   0.47857   0.49379
## prob[11,1]          0.415802  4.282e-01  4.346e-01   0.44090   0.45314
## prob[12,1]          0.463242  4.781e-01  4.861e-01   0.49391   0.50809
## prob[1,2]           0.457155  4.684e-01  4.747e-01   0.48084   0.49211
## prob[2,2]           0.417535  4.268e-01  4.321e-01   0.43735   0.44665
## prob[3,2]           0.417655  4.294e-01  4.357e-01   0.44197   0.45383
## prob[4,2]           0.424603  4.371e-01  4.440e-01   0.45083   0.46408
## prob[5,2]           0.421691  4.321e-01  4.376e-01   0.44294   0.45366
## prob[6,2]           0.427972  4.399e-01  4.461e-01   0.45255   0.46443
## prob[7,2]           0.424407  4.344e-01  4.398e-01   0.44507   0.45476
## prob[8,2]           0.404845  4.151e-01  4.207e-01   0.42636   0.43721
## prob[9,2]           0.417268  4.283e-01  4.344e-01   0.44025   0.45163
## prob[10,2]          0.447891  4.582e-01  4.639e-01   0.46972   0.48035
## prob[11,2]          0.419516  4.287e-01  4.333e-01   0.43817   0.44723
## prob[12,2]          0.471220  4.806e-01  4.856e-01   0.49072   0.50044
## prob[1,3]           0.451708  4.641e-01  4.711e-01   0.47806   0.49140
## prob[2,3]           0.403892  4.184e-01  4.256e-01   0.43325   0.44803
## prob[3,3]           0.389255  4.082e-01  4.178e-01   0.42688   0.44185
## prob[4,3]           0.389536  4.096e-01  4.197e-01   0.43066   0.45102
## prob[5,3]           0.393754  4.115e-01  4.208e-01   0.42940   0.44458
## prob[6,3]           0.423225  4.381e-01  4.464e-01   0.45537   0.47309
## prob[7,3]           0.413057  4.271e-01  4.345e-01   0.44193   0.45723
## prob[8,3]           0.373973  3.917e-01  4.009e-01   0.41019   0.42587
## prob[9,3]           0.399501  4.156e-01  4.234e-01   0.43072   0.44416
## prob[10,3]          0.434689  4.495e-01  4.571e-01   0.46566   0.48161
## prob[11,3]          0.413446  4.257e-01  4.321e-01   0.43879   0.45212
## prob[12,3]          0.463804  4.773e-01  4.851e-01   0.49308   0.50885
```

R code

Data preperation code

```r
# returns column year suffix as "_0", "_PRIOR_1", "_PRIOR_2", ..., "_PRIOR_{N-1}"
get_column_year_suffix <- function(num_years) {
  year_suffix <- c("_0", paste("_PRIOR_", 1:(num_years - 1), sep = ""))
  return (year_suffix)
}

# returns column names as "{COL_NAME}_0", "{COL_NAME}_PRIOR_1", "{COL_NAME}_PRIOR_2", ...
get_column_names <- function(col_name, num_years) {
  suffix <- get_column_year_suffix(num_years)
  column_names <- paste(col_name, suffix, sep = "")
  return (column_names)
}

# returns modeling data for given year, positions and age
get_model_data_wide <- function(season_data, years, positions, minFG) {
  num_years <- length(years)
  sorted_years <- sort(years, decreasing = TRUE)
  yearly_suffix <- get_column_year_suffix(num_years)
  suppressWarnings(rm(wide_data))

  last_n_seasons <- subset(season_data, Year %in% years,
      select = c(COMMON_COLUMNS, c("Year"), YEARLY_COLUMNS))

  if (! missing(positions)) {
    last_n_seasons <- subset(last_n_seasons, Pos %in% positions,
      select = c(COMMON_COLUMNS, c("Year"), YEARLY_COLUMNS))
  }

  if (! missing(minFG)) {
    last_n_seasons <- subset(last_n_seasons, FG >= minFG,
      select = c(COMMON_COLUMNS, c("Year"), YEARLY_COLUMNS))
  }

  for (year_idx in 1:num_years) {
    year <- sorted_years[year_idx]
    yearly_data <- subset(last_n_seasons, Year == year, select = -c(Year))
    colnames(yearly_data) <- c(COMMON_COLUMNS, paste(YEARLY_COLUMNS,
        yearly_suffix[year_idx], sep = ""))

    if (exists("wide_data")) {
      wide_data <- merge(wide_data, yearly_data, by = COMMON_COLUMNS)
    } else {
      wide_data <- yearly_data
```

```
    }
  }

  wide_data$Pos <- factor(wide_data$Pos)

  return(wide_data)
}

# converts long format to wide format
wide_data_to_long_data <- function(wide_data, years) {
  num_years <- length(years)
  sorted_years <- sort(years, decreasing = TRUE)
  yearly_suffix <- get_column_year_suffix(num_years)

  suppressWarnings(rm(long_data))

  for (year_idx in 1:num_years) {
    year <- sorted_years[year_idx]
    yearly_data <- subset(wide_data, select = c(COMMON_COLUMNS,
        paste(YEARLY_COLUMNS, yearly_suffix[year_idx], sep = "")))

    colnames(yearly_data) <- c(COMMON_COLUMNS, YEARLY_COLUMNS)
    yearly_data$Year = year

    if (exists("long_data")) {
      long_data <- rbind(long_data, yearly_data)
    } else {
      long_data <- yearly_data
    }
  }

  return(long_data)
}
```

Model data visualization

```
model_data_long <- wide_data_to_long_data(model_data, INTERESTED_YEARS)
ggplot(data = model_data_long, aes(x=Year, y=FG)) + geom_line(aes(colour=Player))
```

Check overdispersion, chi-square discrepancy

```
df <- as.matrix(x2)

get_posterior_columns <- function(col_name, i, j) {
```

```
  suppressWarnings(rm(columns.v))

  for (j1 in 1:j) {
    temp <- paste( col_name, "[", 1:i, ",", j1, "]", sep = "")

    if (exists("columns.v")) {
      columns.v <- c(columns.v, temp)
    } else {
      columns.v <- temp
    }
  }
  return(columns.v)
}
probs <-  df[, get_posterior_columns("prob", model_data_row_count, YEAR_COUNT)]
FGMrep <-  df[, get_posterior_columns("FGMrep", model_data_row_count, YEAR_COUNT)]
FGM.v <- unlist(d1$FGM)
FGA.v <- unlist(d1$FGA)

Tchi <- matrix(NA, nrow(FGMrep), model_data_row_count * YEAR_COUNT)
Tchirep <- matrix(NA, nrow(FGMrep), model_data_row_count * YEAR_COUNT)
for (s in 1:nrow(FGMrep)){
  Tchi[s,] <- sum((FGM.v - FGA.v * probs[s,])^2 /
                    (FGA.v * probs[s,] * (1-probs[s,])))
  Tchirep[s,] <- sum((FGMrep[s,] - FGA.v * probs[s,])^2 /
                       (FGA.v * probs[s,] * (1-probs[s,])))
}
```

Marginal posterior p-value

```
FGM.p2017 <- numeric(model_data_row_count)
FGM.p2016 <- numeric(model_data_row_count)
FGM.p2015 <- numeric(model_data_row_count)

for (s in 1:model_data_row_count) {
  col_index <- which(colnames(FGMrep) == paste("FGMrep[", s, ",1]", sep = ""))
  FGM.p2017[s] <- mean(FGMrep[,  col_index] >= model_data[s, "FG_0"])

  col_index <- which(colnames(FGMrep) == paste("FGMrep[", s, ",2]", sep = ""))
  FGM.p2016[s] <- mean(FGMrep[,  col_index] >= model_data[s, "FG_PRIOR_1"])

  col_index <- which(colnames(FGMrep) == paste("FGMrep[", s, ",3]", sep = ""))
  FGM.p2015[s] <- mean(FGMrep[,  col_index] >= model_data[s, "FG_PRIOR_2"])
}
posterior_p_df <- data.frame( Player = model_data$Player,
    pValue.2017 = FGM.p2017,
    pValue.2016 = FGM.p2016,
```

```r
    pValue.2015 = FGM.p2015
    )
kable(posterior_p_df)
```

Posterior related utility functions

```r
ilogit <-  function(x) 1/(1+exp(-x))

get_player_posterior_probs <- function(df, data, player_row_id) {
  beta.Year <- df[,paste('beta.Year[', player_row_id, ']', sep = '')]
  player.Effect <- df[,paste('Player.Effect[', player_row_id, ']', sep = '')]

  ## HARD_CODED to 3
  posterior_0 <- numeric(nrow(df))
  posterior_PRIOR_1 <- numeric(nrow(df))
  posterior_PRIOR_2 <- numeric(nrow(df))

  ## HARD_CODED to Experience
  col_names <- get_column_names("EXP", YEAR_COUNT)

  for (s in 1:nrow(df)) {
    posterior_0[s] <- ilogit(beta.Year[s] * data[player_row_id,
        col_names[1]] + player.Effect[s])
    posterior_PRIOR_1[s] <- ilogit(beta.Year[s] * data[player_row_id,
        col_names[2]] + player.Effect[s])
    posterior_PRIOR_2[s] <- ilogit(beta.Year[s] * data[player_row_id,
        col_names[3]] + player.Effect[s])
  }
  posterior <- cbind(posterior_0, posterior_PRIOR_1, posterior_PRIOR_2)

  return(posterior)
}

get_player_posterior_vs_observed <- function(data, player_row_id, posterior) {
  posterior_means <- apply(posterior, 2, mean)

  df_compare <- data.frame(posterior = as.vector(posterior_means),
      observed = as.vector(as.matrix(
      data[player_row_id, get_column_names("FG%", YEAR_COUNT)])))
  rownames(df_compare) <- get_column_names("YEAR", YEAR_COUNT)

  return (df_compare)
}

plot_player_posterior_probs <- function(data, player_row_id, posterior) {
  fg_column_names <- get_column_names("FG%", YEAR_COUNT)
```

```r
  plot1 <- densityplot(posterior[,"posterior_0"],
            panel = function(x, ...) {
             panel.densityplot(x, ...)
              panel.abline(v = mean(x), col.line = "blue")
              panel.abline(v = data[player_row_id, fg_column_names[1]],
                           col.line = "red")
            },
            xlab = paste("Posterior probability", INTERESTED_YEARS[1])
            )
  plot2 <- densityplot(posterior[,"posterior_PRIOR_1"],
            panel = function(x, ...) {
             panel.densityplot(x, ...)
              panel.abline(v = mean(x), col.line = "blue")
              panel.abline(v = data[player_row_id, fg_column_names[2]],
                           col.line = "red")
            },
            xlab = paste("Posterior probability", INTERESTED_YEARS[2])
            )
  plot3 <- densityplot(posterior[,"posterior_PRIOR_2"],
            panel = function(x, ...) {
             panel.densityplot(x, ...)
              panel.abline(v = mean(x), col.line = "blue")
              panel.abline(v = data[player_row_id, fg_column_names[3]],
                           col.line = "red")
            },
            xlab = paste("Posterior probability", INTERESTED_YEARS[3])
            )

  grid.arrange(plot1, plot2, plot3, ncol = 3, nrow = 1)
}
```

Posterior odds

```r
postodds <- data.frame(matrix(NA, model_data_row_count, ncol=2))
for (i in 1:model_data_row_count) {
  post_prob <- get_player_posterior_probs(df, model_data, i)
  postodds[i,] <- c(mean(post_prob[,1] > post_prob[,2]),
                    mean(post_prob[,2] > post_prob[,1]))
}
postodds <- cbind(model_data$Player, postodds)
names(postodds) <- c("Player", "2016-2017", "2015-2016")
kable(postodds)
```