Bayesian_Rugby

May 23, 2015

Rugby Analytics

- Peadar Coyle
- Will give this at PyData Berlin 2015
- Data Science Meetup Luxembourg May 2015

1 Contents: Probabilistic Programming applied to Rugby

- I'll discuss what probabilistic programming is, why should you care and how to use PyMC from Python to implement these methods.
- I'll be applying these methods to studying the problem of 'rugby sports analytics' particularly how to model the winning team in the recent Six Nations in Rugby.
- I will discuss the framework and how I was able to quickly and easily produce an innovative and powerful model as a non-expert.

1.1 Who am I?

- I'm a Data Analytics Professional based in Luxembourg
- I currently work for Vodafone
- My intellectual background is in Physics and Mathematics
- I've made open source contributions to Pandas and Probabilistic Programming and Bayesian Methods for Hackers.
- I've helped companies solve analytics challenges in Supply Chain Management, Air Traffic Analysis and with Customer Analytics
- All opinions are my own!

2 All Sports Commentary!

* Attribution: Xkcd

3 How can statistics help with sports?

- Well fundamentally a Rugby game is a predictible event.
- How do we generate a model to predict the outcome of a tournament?
- How do we quantify our uncertainty in our model?

4 What influenced me on this?

Attribution: Quantopian blog

5 What's wrong with statistics

- Models should not be built for mathematical convenience (e.g. normality assumption), but to most accurately model the data.
- Pre-specified models, like frequentist statistics, make many assumptions that are all to easily violated.

6 "The purpose of computation is insight, not numbers." – Richard Hamming

7 What is Bayesian Statistics?

- At the core: formula to update our beliefs after having observed data (Bayes formula)
- Implies that we have a prior belief about the world.
- Updated beliefs after observing data is called posterior.
- Beliefs are represented using random variables.

7.0.1 So what problem could I apply Bayesian models to?

- Rugby Analysis! Attribution: The-office-bar.eu

8 Bayesian Rugby

I came across the following blog post on http://danielweitzenfeld.github.io/passtheroc/blog/2014/10/28/bayes-premier-league/

- Based on the work of Baio and Blangiardo
- In this talk, I'm going to reproduce the first model described in the paper using pymc.
- Since I am a rugby fan I decide to apply the results of the paper Bayesian Football to the Six Nations.

8.1 What they did?

Deriving our new measuring model and verifying that it works took some effort! But it was all worth it, because now we have:

Automatic weight estimations for each Zalando article, which saves workers time A reliable way to know the accuracy of our estimations And most importantly: our warehouse workers can now focus on getting your fashion to you as quickly as possible. That's isn't just saving money–that's priceless.

9 So why Bayesians?

- Probabilistic Programming is a new paradigm.
- Attributions: My friend Thomas Wiecki influenced a lot of my thinking on this.
- I'm going to compare Blackbox Machine Learning with scikit-learn
- Source: Olivier Grisel's talk on ML

10 Limitations of Machine learning

- A big limitation of Machine Learning is that most of the models are black boxes.

Source: Olivier Grisel's talk on ML

11 Probabilistic Programming - what's the big deal?

- We are able to use data and our prior beliefs to generate a model.
- Generating a model is extremely powerful
- We can tell a story, which appeals to our understanding of stories.

In [22]:
```
import os
import math
import warnings
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import pymc # I know folks are switching to "as pm" but I'm just not there yet
%matplotlib inline
import seaborn as sns
from IPython.core.pylabtools import figsize
import seaborn as sns
figsize(12, 8)
```

12 Six Nations Rugby

- Rugby is a physical sport popular worldwide.
- Six Nations consists of Italy, Ireland, Scotland, England, France and Wales
- Game consists of scoring tries (similar to touch downs) or kicking the goal.
- Average player is something like 100kg and 1.82m tall.
- Paul O'Connell the Irish captain is Height: 6' 6" (1.98 m) Weight: 243 lbs (110 kg)

13 They compete for this!

- Significant this year because the World Cup occurs in 2015.
- Photo: Hostelrome

13.1 Motivation

- Someone in the Sports Analytics community made the point (I'm paraphrasing here) that your estimate of one team's strength depends on your estimate of all the others. The conclusions you draw from team X beating team Y depends how strong team Y is, which in turn depends on the conclusions you draw from team Y's other games, which in turn depends on how strong Y's opponents were, etc.
- Ireland are a stronger team than Italy for example - but by how much?
- Source for Results 2014 are Wikipedia.
- I handcrafted these results

In [23]:
```
DATA_DIR = os.path.join(os.getcwd(), 'data/')
```

In [24]:
```
#The results_2014 is a handcrafted results table from Wikipedia
data_file = DATA_DIR + 'results_2014.csv'
df = pd.read_csv(data_file, sep=',')
df.tail()
```

Out[24]:

	home_team	away_team	home_score	away_score
10	Scotland	France	17	19
11	England	Wales	29	18
12	Italy	England	11	52

```
13     Wales   Scotland          51           3
14    France    Ireland          20          22
```

```
In [25]: teams = df.home_team.unique()
         teams = pd.DataFrame(teams, columns=['team'])
         teams['i'] = teams.index
         teams.head()
```

```
Out[25]:        team  i
         0     Wales  0
         1    France  1
         2   Ireland  2
         3  Scotland  3
         4     Italy  4
```

- Now we need to do some merging and munging

```
In [26]: df = pd.merge(df, teams, left_on='home_team', right_on='team', how='left')
         df = df.rename(columns = {'i': 'i_home'}).drop('team', 1)
         df = pd.merge(df, teams, left_on='away_team', right_on='team', how='left')
         df = df.rename(columns = {'i': 'i_away'}).drop('team', 1)
         df.head()
```

```
Out[26]:   home_team away_team  home_score  away_score  i_home  i_away
         0     Wales     Italy          23          15       0       4
         1    France   England          26          24       1       5
         2   Ireland  Scotland          28           6       2       3
         3   Ireland     Wales          26           3       2       0
         4  Scotland   England           0          20       3       5
```

```
In [27]: observed_home_goals = df.home_score.values
         observed_away_goals = df.away_score.values
         home_team = df.i_home.values
         away_team = df.i_away.values
         num_teams = len(df.i_home.drop_duplicates())
         num_games = len(home_team)
```

Now we need to prepare the model for PyMC.

```
In [28]: g = df.groupby('i_away')
         att_starting_points = np.log(g.away_score.mean())
         g = df.groupby('i_home')
         def_starting_points = -np.log(g.away_score.mean())
```

14 What do we want to infer?

- We want to infer the latent paremeters (every team's strength) that are generating the data we observe (the scorelines).
- Moreover, we know that the scorelines are a noisy measurement of team strength, so ideally, we want a model that makes it easy to quantify our uncertainty about the underlying strengths.

15 While my MCMC gently samples

- Often we don't know what the Bayesian Model is explicitly, so we have to 'estimate' the Bayesian Model'

- If we can't solve something, approximate it.
- Markov-Chain Monte Carlo (MCMC) instead draws samples from the posterior.
- Fortunately, this algorithm can be applied to almost any model.
- Hattip: @twiecki

16 What do we want?

- We want to quantify our uncertainty
- We want to also use this to generate a model
- We want the answers as distributions *not* point estimates

16.1 What assumptions do we know for our 'generative story'?

- We know that the Six Nations in Rugby only has 6 teams.
- We have data from last year!
- We also know that in sports scoring is modelled as a Poisson distribution
- Attribution: *Wikipedia*

17 The model.

The league is made up by a total of T= 6 teams, playing each other once in a season. We indicate the number of points scored by the home and the away team in the g-th game of the season (15 games) as y_{g1} and y_{g2} respectively.

The vector of observed counts $\curvearrowright = (y_{g1}, y_{g2})$ is modelled as independent Poisson: $y_{gi}|\theta_{gj}\tilde{}\ Poisson(\theta_{gj})$ where the theta parameters represent the scoring intensity in the g-th game for the team playing at home (j=1) and away (j=2), respectively.

We model these parameters according to a formulation that has been used widely in the statistical literature, assuming a log-linear random effect model:

$$log\theta_{g1} = home + att_{h(g)} + def_{a(g)}$$

$$log\theta_{g2} = att_{a(g)} + def_{h(g)}$$

the parameter home represents the advantage for the team hosting the game and we assume that this effect is constant for all the teams and throughout the season.

- Key assumption home effect is an advantage in Sports
- We know these things empirically from our 'domain specific' knowledge
- Bayesian Models allow you to incorporate *beliefs* or *knowledge* into your model!

In addition, the scoring intensity is determined jointly by the attack and defense ability of the two teams involved, represented by the parameters att and def, respectively. In line with the Bayesian approach, we have to specify some suitable prior distributions for all the random parameters in our model. The variable *home* is modelled as a fixed effect, assuming a standard flat prior distribution. We use the notation of describing the Normal distribution in terms of mean and the precision. $home\tilde{}Normal(0, 0.0001)$

Conversely, for each t = 1, . . . , T, the team-specific effects are modelled as exchangeable from a common distribution: $att_t\tilde{}Normal(\mu_{att}, \tau_{att})$ and $def_t\tilde{}Normal(\mu_{def}, \tau_{def})$

Note that they're breaking out team strength into attacking and defending strength. A negative defense parameter will sap the mojo from the opposing team's attacking parameter.

I made two tweaks to the model. It didn't make sense to me to have a μ_{att} when we're enforcing the sum-to-zero constraint by subtracting the mean anyway. So I eliminated μ_{att} and μ_{def}

Also because of the sum-to-zero constraint, it seemed to me that we needed an intercept term in the log-linear model, capturing the average goals scored per game by the away team. This we model with the following hyperprior.

$$intercept\tilde{}Normal(0, 0.001)$$

In [29]:
```
#hyperpriors
home = pymc.Normal('home', 0, .0001, value=0)
tau_att = pymc.Gamma('tau_att', .1, .1, value=10)
tau_def = pymc.Gamma('tau_def', .1, .1, value=10)
intercept = pymc.Normal('intercept', 0, .0001, value=0)
#team-specific parameters
atts_star = pymc.Normal("atts_star",
                        mu=0,
                        tau=tau_att,
                        size=num_teams,
                        value=att_starting_points.values)
defs_star = pymc.Normal("defs_star",
                        mu=0,
                        tau=tau_def,
                        size=num_teams,
                        value=def_starting_points.values)
# trick to code the sum to zero constraint
@pymc.deterministic
def atts(atts_star=atts_star):
    atts = atts_star.copy()
    atts = atts - np.mean(atts_star)
    return atts

@pymc.deterministic
def defs(defs_star=defs_star):
    defs = defs_star.copy()
    defs = defs - np.mean(defs_star)
    return defs

@pymc.deterministic
def home_theta(home_team=home_team,
               away_team=away_team,
               home=home,
               atts=atts,
               defs=defs,
               intercept=intercept):
    return np.exp(intercept +
                  home +
                  atts[home_team] +
                  defs[away_team])

@pymc.deterministic
def away_theta(home_team=home_team,
               away_team=away_team,
               home=home,
               atts=atts,
               defs=defs,
               intercept=intercept):
    return np.exp(intercept +
                  atts[away_team] +
                  defs[home_team])
```

18 Let us run the model!

- We specify the priors as Gamma distributions

```
In [30]: home_points = pymc.Poisson('home_points',
                                    mu=home_theta,
                                    value=observed_home_goals,
                                    observed=True)
         away_points = pymc.Poisson('away_points',
                                    mu=away_theta,
                                    value=observed_away_goals,
                                    observed=True)

         mcmc = pymc.MCMC([home, intercept, tau_att, tau_def,
                           home_theta, away_theta,
                           atts_star, defs_star, atts, defs,
                           home_points, away_points])
         map_ = pymc.MAP( mcmc )
         map_.fit()

         mcmc.sample(200000, 40000, 20)
```

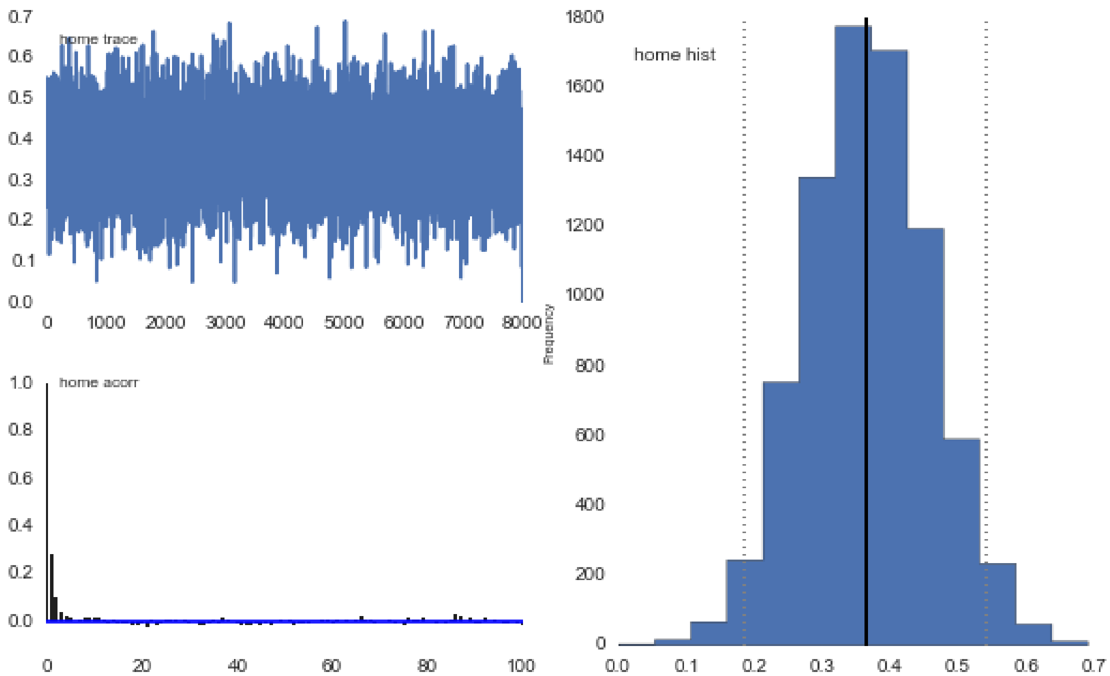
```
[-----------------100%-----------------] 200000 of 200000 complete in 69.6 sec
```

19 Diagnostics

Let's see if/how the model converged. The home parameter looks good, and indicates that home field advantage amounts to goals per game at the intercept. We can see that it converges just like the model for the Premier League in the other tutorial. I wonder and this is left as a question if all field sports have models of this form that converge.

```
In [31]: pymc.Matplot.plot(home)
```

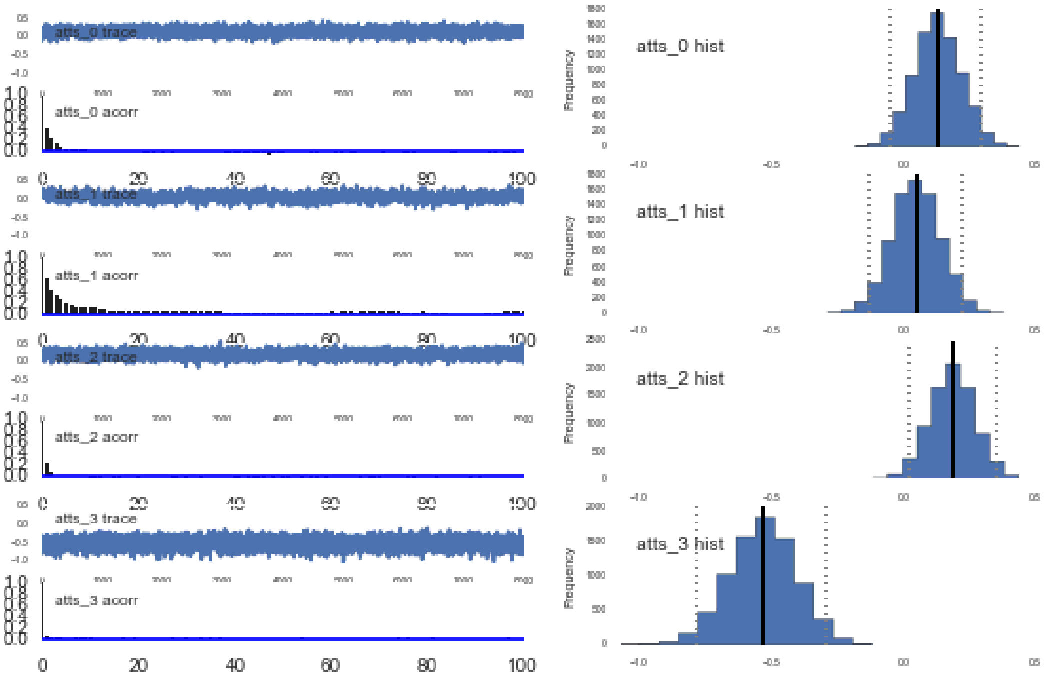
```
Plotting home
```

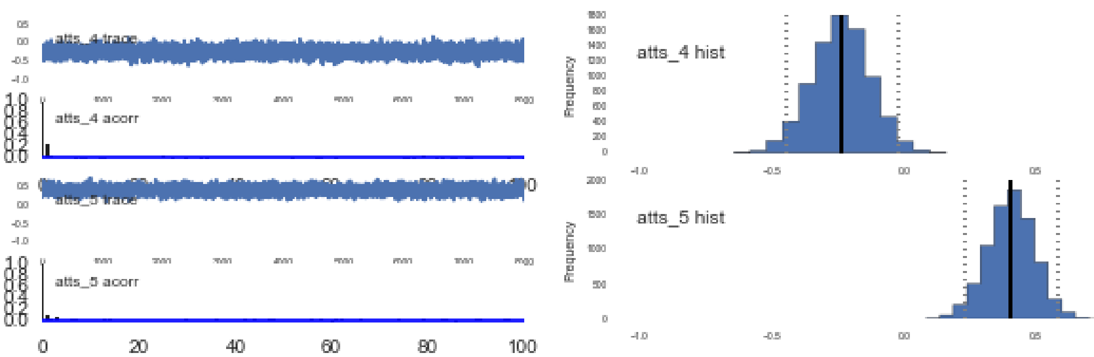


- We can see in this analysis that home advantage gives about 0.55 points advantage.
- We also see not too much auto-correlation, so this looks quite good plot wise.
- We see here how *probabilistic programming* allows us to quantify our uncertainty about certain parameters.

In [32]:
```
pymc.Matplot.plot(atts)
# We can plot all of the parameters, just to see.
```

```
Plotting atts_0
Plotting atts_1
Plotting atts_2
Plotting atts_3
Plotting atts_4
Plotting atts_5
```





20 Simulating a season

We would like to now simulate a season. Just to see what happens.

```
In [33]: def simulate_season():
             """
             Simulate a season once, using one random draw from the mcmc chain.
             """
             num_samples = atts.trace().shape[0]
             draw = np.random.randint(0, num_samples)
             atts_draw = pd.DataFrame({'att': atts.trace()[draw, :],})
             defs_draw = pd.DataFrame({'def': defs.trace()[draw, :],})
```

```
    home_draw = home.trace()[draw]
    intercept_draw = intercept.trace()[draw]
    season = df.copy()
    season = pd.merge(season, atts_draw, left_on='i_home', right_index=True)
    season = pd.merge(season, defs_draw, left_on='i_home', right_index=True)
    season = season.rename(columns = {'att': 'att_home', 'def': 'def_home'})
    season = pd.merge(season, atts_draw, left_on='i_away', right_index=True)
    season = pd.merge(season, defs_draw, left_on='i_away', right_index=True)
    season = season.rename(columns = {'att': 'att_away', 'def': 'def_away'})
    season['home'] = home_draw
    season['intercept'] = intercept_draw
    season['home_theta'] = season.apply(lambda x: math.exp(x['intercept'] +
                                                           x['home'] +
                                                           x['att_home'] +
                                                           x['def_away']), axis=1)
    season['away_theta'] = season.apply(lambda x: math.exp(x['intercept'] +
                                                           x['att_away'] +
                                                           x['def_home']), axis=1)
    season['home_goals'] = season.apply(lambda x: np.random.poisson(x['home_theta']), axis=1)
    season['away_goals'] = season.apply(lambda x: np.random.poisson(x['away_theta']), axis=1)
    season['home_outcome'] = season.apply(lambda x: 'win' if x['home_goals'] > x['away_goals']
                                                    'loss' if x['home_goals'] < x['away_goals'
    season['away_outcome'] = season.apply(lambda x: 'win' if x['home_goals'] < x['away_goals']
                                                    'loss' if x['home_goals'] > x['away_goals'
    season = season.join(pd.get_dummies(season.home_outcome, prefix='home'))
    season = season.join(pd.get_dummies(season.away_outcome, prefix='away'))
    return season


def create_season_table(season):
    """
    Using a season dataframe output by simulate_season(), create a summary dataframe with wins

    """
    g = season.groupby('i_home')
    home = pd.DataFrame({'home_goals': g.home_goals.sum(),
                         'home_goals_against': g.away_goals.sum(),
                         'home_wins': g.home_win.sum(),
                         'home_losses': g.home_loss.sum()
                         })
    g = season.groupby('i_away')
    away = pd.DataFrame({'away_goals': g.away_goals.sum(),
                         'away_goals_against': g.home_goals.sum(),
                         'away_wins': g.away_win.sum(),
                         'away_losses': g.away_loss.sum()
                         })
    df = home.join(away)
    df['wins'] = df.home_wins + df.away_wins
    df['losses'] = df.home_losses + df.away_losses
    df['points'] = df.wins * 2
    df['gf'] = df.home_goals + df.away_goals
    df['ga'] = df.home_goals_against + df.away_goals_against
    df['gd'] = df.gf - df.ga
    df = pd.merge(teams, df, left_on='i', right_index=True)
```

```
    df = df.sort_index(by='points', ascending=False)
    df = df.reset_index()
    df['position'] = df.index + 1
    df['champion'] = (df.position == 1).astype(int)
    df['relegated'] = (df.position > 5).astype(int)
    return df

def simulate_seasons(n=100):
    dfs = []
    for i in range(n):
        s = simulate_season()
        t = create_season_table(s)
        t['iteration'] = i
        dfs.append(t)
    return pd.concat(dfs, ignore_index=True)
```

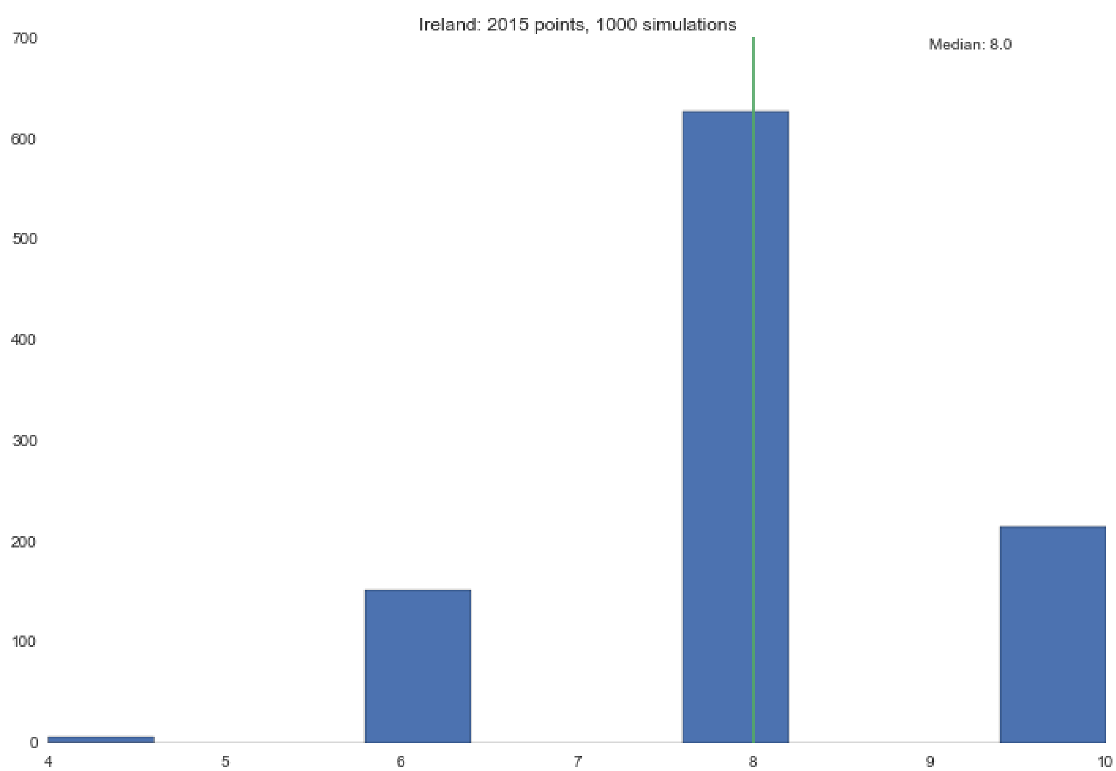
21 Simulation

- We are going to simulate 1000 seasons

```
In [34]: simuls = simulate_seasons(1000)
```

```
In [35]: def fig1():
             ax = simuls.points[simuls.team == 'Ireland'].hist()
             median = simuls.points[simuls.team == 'Ireland'].median()
             ax.set_title('Ireland: 2015 points, 1000 simulations')
             ax.plot([median, median], ax.get_ylim())
             plt.annotate('Median: %s' % median, xy=(median + 1, ax.get_ylim()[1]-10))
```

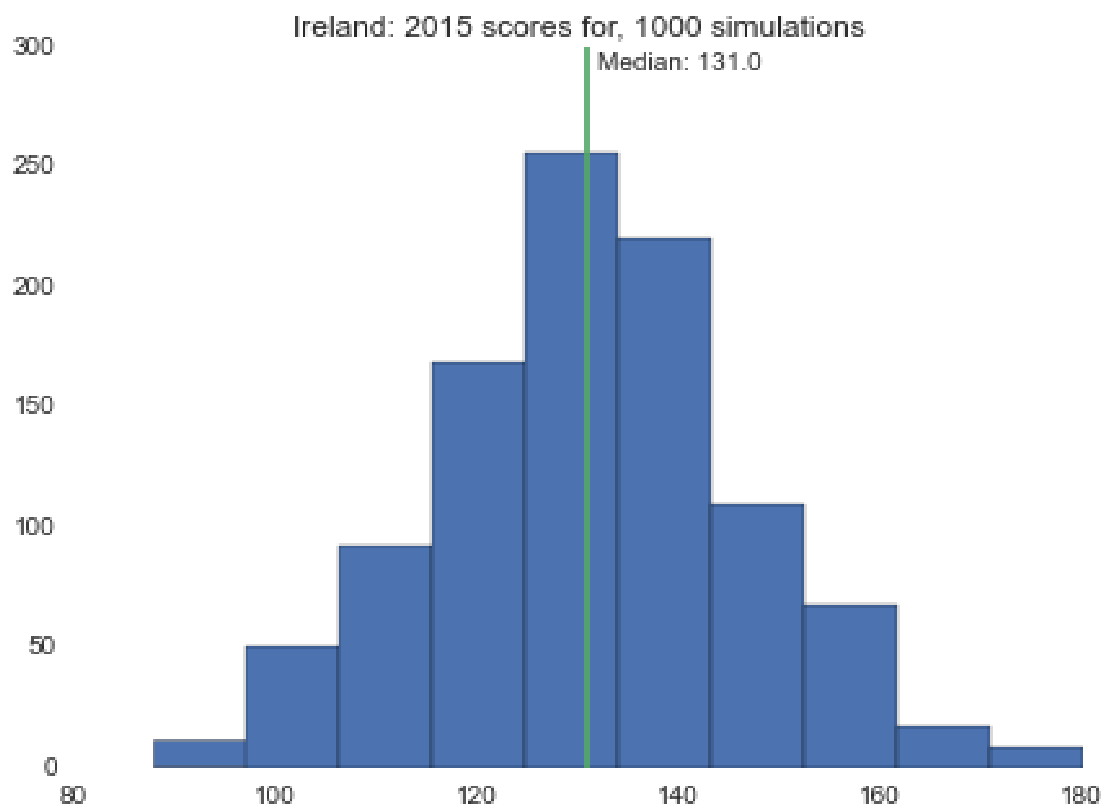
```
In [36]: fig1()
```



- So what have we learned so far, we've got 1000 simulations of Ireland and their median points in the table is 8.
- In Rugby you get 2 points per win, and there are 5 games per year. So this model predicted that Ireland would win most of the time 4 games.

```
In [37]: def fig2():
             ax = simuls.gf[simuls.team == 'Ireland'].hist(figsize=(7,5))
             median = simuls.gf[simuls.team == 'Ireland'].median()
             ax.set_title('Ireland: 2015 scores for, 1000 simulations')
             ax.plot([median, median], ax.get_ylim())
             plt.annotate('Median: %s' % median, xy=(median + 1, ax.get_ylim()[1]-10))
```

```
In [38]: fig2()
```



22 What happened in reality?

- Well Ireland actually scored 119 points, so the model over predicted this!
- We call this 'shrinkage' in the literature.
- All models are wrong, but some are useful

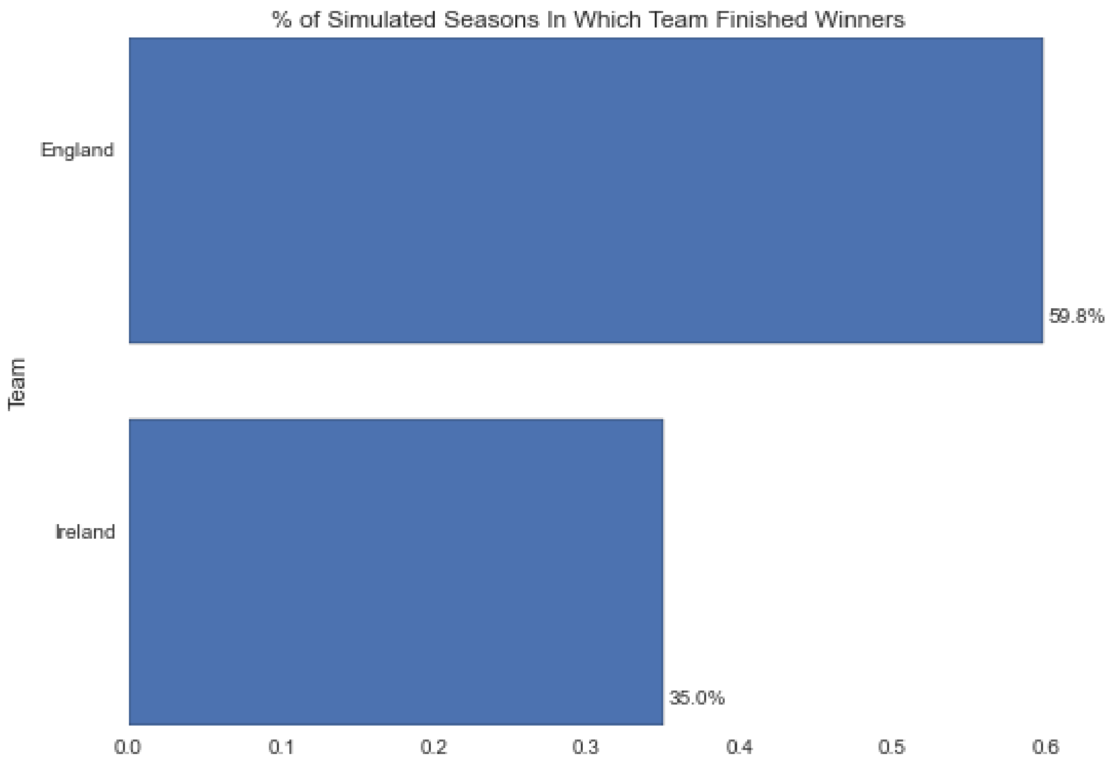
23 What are the predictions of the model?

- So let us look at the winning team on average.
- We do a simulation and we'll assign probability of 'winning' to the team
- We used the MCMC to do this.

```
In [39]: g = simuls.groupby('team')
         df_champs = pd.DataFrame({'percent_champs': g.champion.mean()})
         df_champs = df_champs.sort_index(by='percent_champs')
         df_champs = df_champs[df_champs.percent_champs > .05]
         df_champs = df_champs.reset_index()
```

```
In [40]: fig, ax = plt.subplots(figsize=(8,6))
         ax.barh(df_champs.index.values, df_champs.percent_champs.values)

         for i, row in df_champs.iterrows():
             label = "{0:.1f}%".format(100 * row['percent_champs'])
             ax.annotate(label, xy=(row['percent_champs'], i), xytext = (3, 10), textcoords = 'offset po
         ax.set_ylabel('Team')
         ax.set_title('% of Simulated Seasons In Which Team Finished Winners')
         _= ax.set_yticks(df_champs.index + .5)
         _= ax.set_yticklabels(df_champs['team'].values)
```



Unfortunately it seems that in most of the Universes England come top of the Six Nations. And as an Irish man this is firm proof that I put Mathematical rigour before patriotism :) This is a reasonable result, and I hope it proved a nice example of Bayesian models in Rugby Analytics.

24 What actually happened

We need to investigate like 'scientists' what actually happened.

```
In [41]: pd.read_csv('data/results_2015.csv', sep='\t', header=0,
                     names =['Rank','Team','Games','Wins','Draws',
                             'Losses','Points_For','Points_Against','Points'])
```

```
Out[41]:    Rank      Team  Games  Wins  Draws  Losses  Points_For  Points_Against  \
         0     1  Ireland       5     4      0       1         119              56
         1     2  England       5     4      0       1         157             100
         2     3  Wales         5     4      0       1         146              93
         3     4  France        5     2      0       3         103             101
         4     5  Italy         5     1      0       4          62             182
         5     6  Scotland      5     0      0       5          73             128

            Points
         0       8
         1       8
         2       8
         3       4
         4       2
         5       0
```

25 Conclusion

- 'All models are wrong, some are useful'
- Model correctly predicts that Ireland would win 4 games
- Model incorrectly predicted that England would come out on top
- In reality it was very close
- Recommendation: Don't use this model to bet on the Six Nations next years

- Thomas Wiecki Blog on all things Bayesian

- Twitter: [@springcoil](https://twitter.com/springcoil)

- Probilistic Programming for Hackers – IPython Notebook book on Bayesian stats using PyMC2

- Doing Bayesian Data Analysis – Great book by Kruschke.

- Get PyMC3 alpha

- Zalando Example

```
In [42]: from IPython.display import Image
         Image(filename='the-most-interesting-man-in-the-world-meme-generator-i-don-t-give-many-talks-b
```

```
Out[42]:
```

TODO: Include some pedagogical points about Bayesian models. How do you pick your prior