A Hierarchical model for Rugby prediction

In this example, we're going to reproduce the first model described in Baio and Blangiardo [2010] using PyMC. Then show how to sample from the posterior predictive to simulate championship outcomes from the scored goals which are the modeled quantities.

We apply the results of the paper to the Six Nations Championship, which is a competition between Italy, Ireland, Scotland, England, France and Wales.

Motivation

Your estimate of the strength of a team depends on your estimates of the other strengths

Ireland are a stronger team than Italy for example - but by how much?

Source for Results 2014 are Wikipedia. I've added the subsequent years, 2015, 2016, 2017. Manually pulled from Wikipedia.

- We want to infer a latent parameter - that is the 'strength' of a team based only on their **scoring intensity**, and all we have are their scores and results, we can't accurately measure the 'strength' of a team.
- Probabilistic Programming is a brilliant paradigm for modeling these **latent** parameters
- Aim is to build a model for the upcoming Six Nations in 2018.

Attention

This notebook uses libraries that are not PyMC dependencies and therefore need to be installed specifically to run this notebook. Open the dropdown below for extra guidance.

Skip to main content

Extra dependencies install instructions

In order to run this notebook (either locally or on binder) you won't only need a working PyMC installation with all optional dependencies, but also to install some extra dependencies. For advise on installing PyMC itself, please refer to Installation

You can install these dependencies with your preferred package manager, we provide as an example the pip and conda commands below.

> $ pip install seaborn numba

Note that if you want (or need) to install the packages from inside the notebook instead of the command line, you can install the packages by running a variation of the pip command:

> import sys
>
> !{sys.executable} -m pip install seaborn numba

You should not run `!pip install` as it might install the package in a different environment and not be available from the Jupyter notebook even if installed.

Another alternative is using conda instead:

> $ conda install seaborn numba

when installing scientific python packages with conda, we recommend using conda forge

```
!date

import arviz as az
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import pymc as pm
import pytensor.tensor as pt
import seaborn as sns
```

Skip to main content

```
%matplotlib inline
```

```
sáb 02 abr 2022 03:24:55 EEST
```

```
az.style.use("arviz-darkgrid")
plt.rcParams["figure.constrained_layout.use"] = False
```

This is a Rugby prediction exercise. So we'll input some data. We've taken this from Wikipedia and BBC sports.

```
try:
    df_all = pd.read_csv("../data/rugby.csv", index_col=0)
except:
    df_all = pd.read_csv(pm.get_data("rugby.csv"), index_col=0)
```

What do we want to infer?

- We want to infer the latent parameters (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

- 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.

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

Skip to main content

Visualization/EDA

We should do some some exploratory data analysis of this dataset.

The plots should be fairly self-explantory, we'll look at things like difference between teams in terms of their scores.

```
df_all.describe()
```

	home_score	away_score	year
count	60.000000	60.000000	60.000000
mean	23.500000	19.983333	2015.500000
std	14.019962	12.911028	1.127469
min	0.000000	0.000000	2014.000000
25%	16.000000	10.000000	2014.750000
50%	20.500000	18.000000	2015.500000
75%	27.250000	23.250000	2016.250000
max	67.000000	63.000000	2017.000000

```
# Let's look at the tail end of this dataframe
df_all.tail()
```

	home_team	away_team	home_score	away_score	year
55	Italy	France	18	40	2017
56	England	Scotland	61	21	2017
57	Scotland	Italy	29	0	2017
58	France	Wales	20	18	2017
59	Ireland	England	13	9	2017

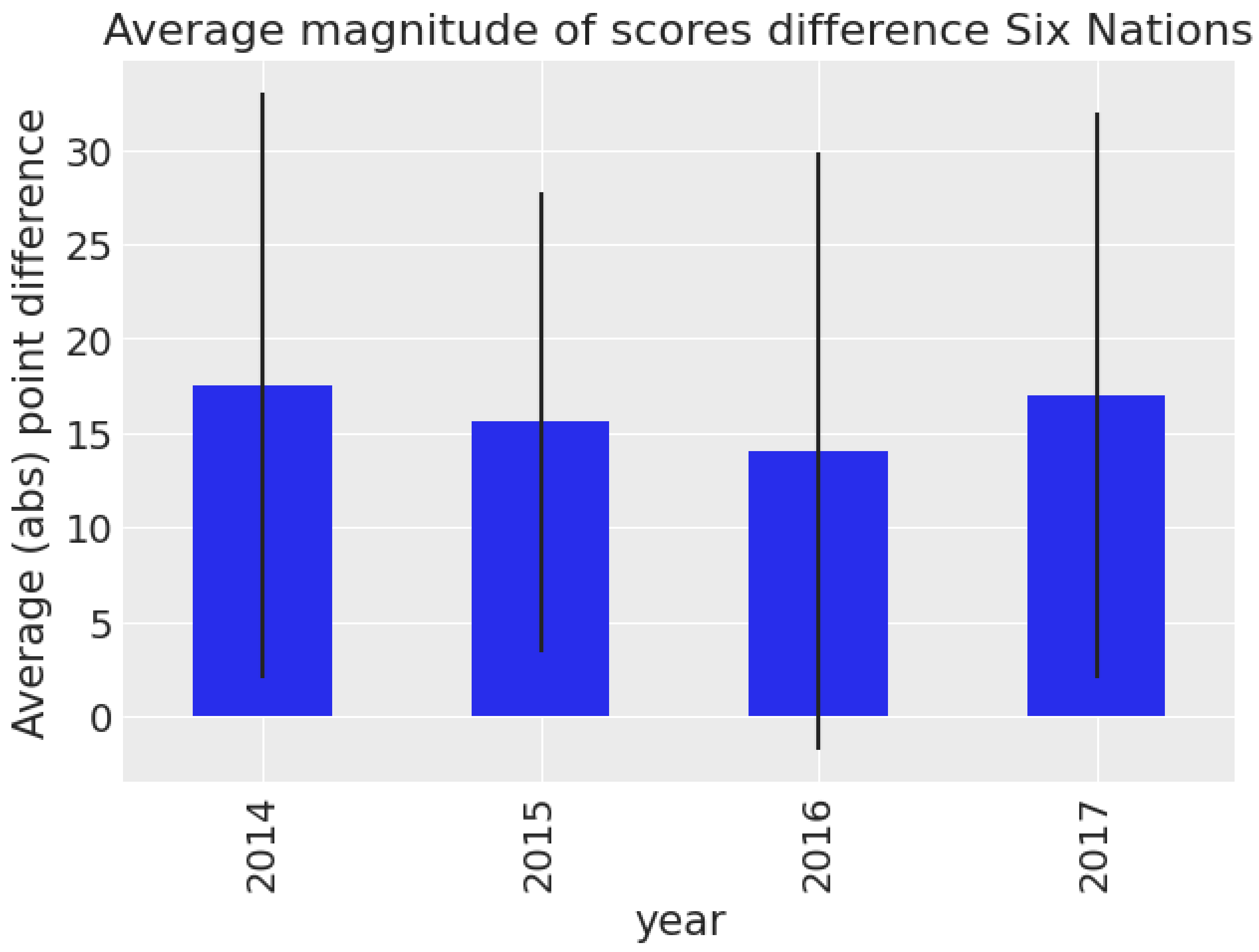
There are a few things here that we don't need. We don't need the year for our model. But that is

Skip to main content

Firstly let us look at differences in scores by year.

```
df_all["difference"] = np.abs(df_all["home_score"] - df_all["away_score"])
```

```
(
    df_all.groupby("year")["difference"]
    .mean()
    .plot(
        kind="bar",
        title="Average magnitude of scores difference Six Nations",
        yerr=df_all.groupby("year")["difference"].std(),
    )
    .set_ylabel("Average (abs) point difference")
);
```



We can see that the standard error is large. So we can't say anything about the differences. Let's look country by country.

```
df_all["difference_non_abs"] = df_all["home_score"] - df_all["away_score"]
```

Skip to main content

Let us first loook at a Pivot table with a sum of this, broken down by year.

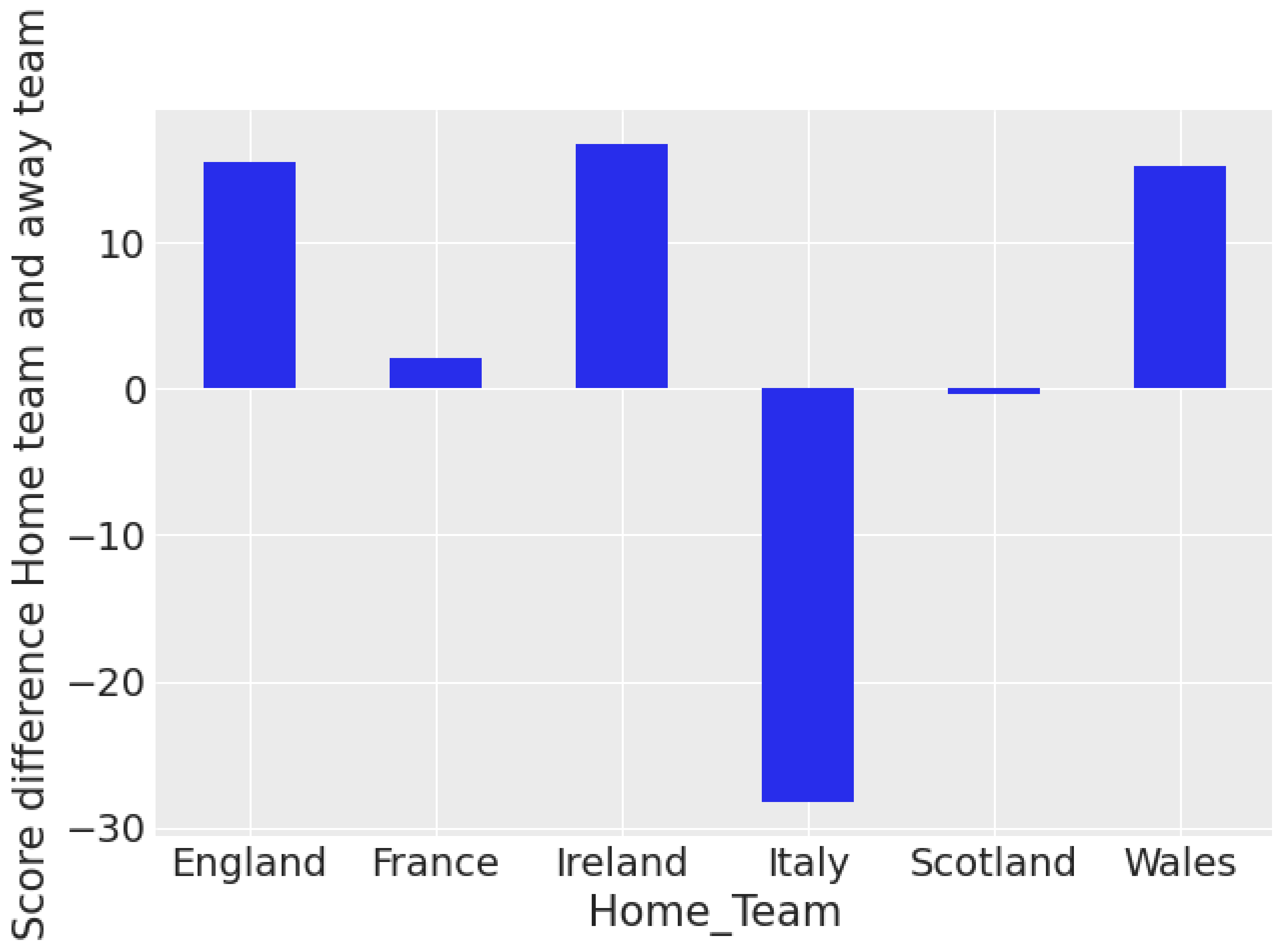
```
df_all.pivot_table("difference_non_abs", "home_team", "year")
```

year	2014	2015	2016	2017
home_team				
England	7.000000	20.666667	7.500000	21.333333
France	6.666667	0.000000	-2.333333	4.000000
Ireland	28.000000	8.500000	17.666667	7.000000
Italy	-21.000000	-31.000000	-23.500000	-33.666667
Scotland	-11.000000	-12.000000	2.500000	16.666667
Wales	25.666667	1.000000	22.000000	4.000000

Now let's first plot this by home team without year.

```
(
    df_all.pivot_table("difference_non_abs", "home_team")
    .rename_axis("Home_Team")
    .plot(kind="bar", rot=0, legend=False)
    .set_ylabel("Score difference Home team and away team")
);
```

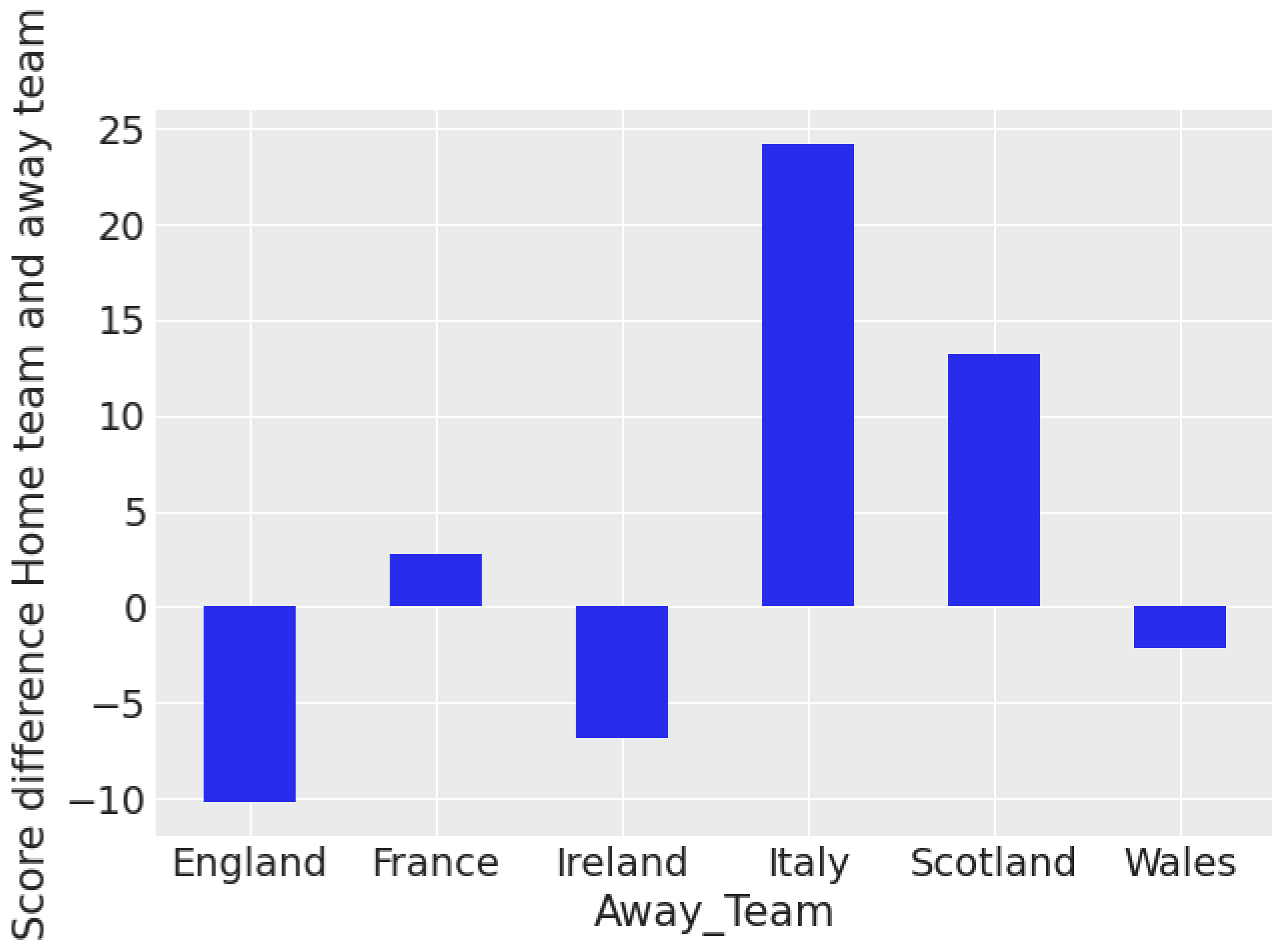
Skip to main content



You can see that Italy and Scotland have negative scores on average. You can also see that England, Ireland and Wales have been the strongest teams lately at home.

```
(
    df_all.pivot_table("difference_non_abs", "away_team")
    .rename_axis("Away_Team")
    .plot(kind="bar", rot=0, legend=False)
    .set_ylabel("Score difference Home team and away team")
);
```

Skip to main content



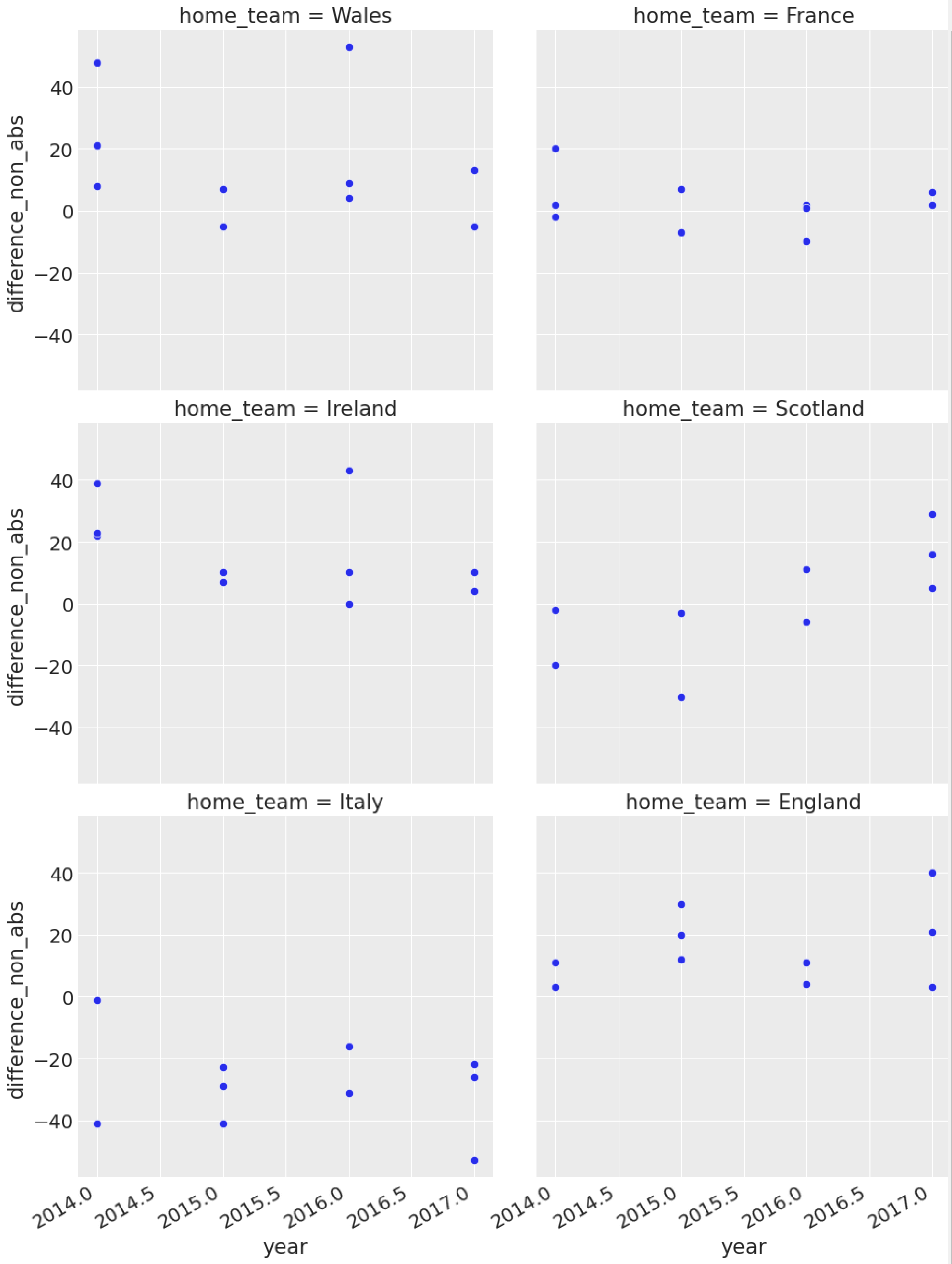
This indicates that Italy, Scotland and France all have poor away from home form. England suffers the least when playing away from home. This aggregate view doesn't take into account the strength of the teams.

Let us look a bit more at a timeseries plot of the average of the score difference over the year.

We see some changes in team behaviour, and we also see that Italy is a poor team.

```
g = sns.FacetGrid(df_all, col="home_team", col_wrap=2, height=5)
g.map(sns.scatterplot, "year", "difference_non_abs")
g.fig.autofmt_xdate()
```

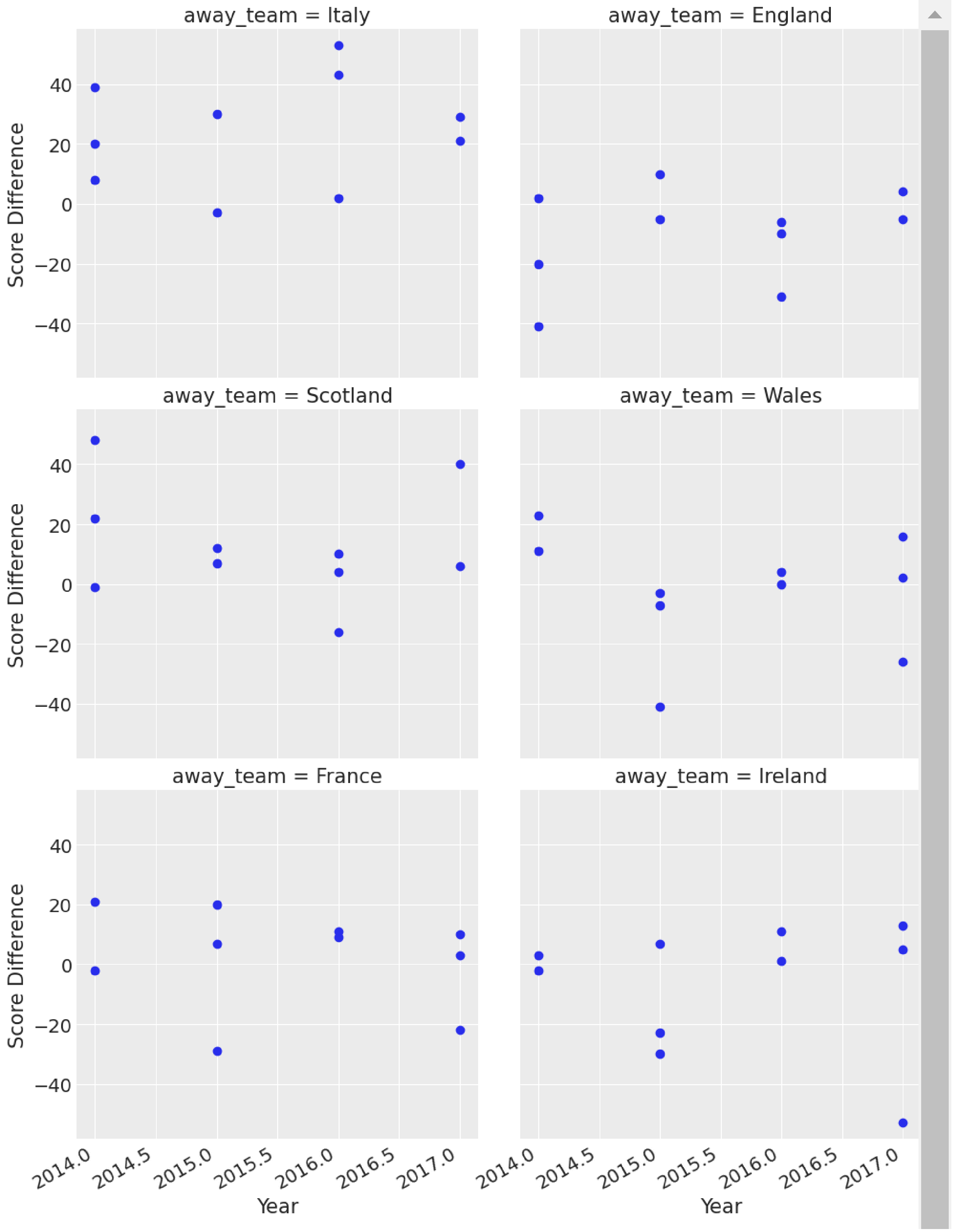
Skip to main content



Skip to main content

```
g = sns.FacetGrid(df_all, col="away_team", col_wrap=2, height=5)
g = g.map(plt.scatter, "year", "difference_non_abs").set_axis_labels("Year", "Score Diffe
g.fig.autofmt_xdate()
```

Skip to main content



Skip to main content

You can see some interesting things here like Wales were good away from home in 2015. In that year they won three games away from home and won by 40 points or so away from home to Italy.

So now we've got a feel for the data, we can proceed on with describing the model.

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

- We know that the Six Nations in Rugby only has 6 teams - they each play each other once
- We have data from the last few years
- We also know that in sports scoring is modelled as a Poisson distribution
- We consider home advantage to be a strong effect in sports

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 $\mathbb{y} = (y_{g1}, y_{g2})$ 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
- 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

Skip to main content

- Conversely, for each t = 1, …, T, the team-specific effects are modelled as exchangeable from a common distribution:
- $att_t \sim Normal(\mu_{att}, \tau_{att})$ and $def_t \sim Normal(\mu_{def}, \tau_{def})$
- We did some munging above and adjustments of the data to make it **tidier** for our model.
- The log function to away scores and home scores is a standard trick in the sports analytics literature

Building of the model

We now build the model in PyMC, specifying the global parameters, the team-specific parameters and the likelihood function

```
plt.rcParams["figure.constrained_layout.use"] = True
home_idx, teams = pd.factorize(df_all["home_team"], sort=True)
away_idx, _ = pd.factorize(df_all["away_team"], sort=True)
coords = {"team": teams}
```

```
with pm.Model(coords=coords) as model:
    # constant data
    home_team = pm.ConstantData("home_team", home_idx, dims="match")
    away_team = pm.ConstantData("away_team", away_idx, dims="match")

    # global model parameters
    home = pm.Normal("home", mu=0, sigma=1)
    sd_att = pm.HalfNormal("sd_att", sigma=2)
    sd_def = pm.HalfNormal("sd_def", sigma=2)
    intercept = pm.Normal("intercept", mu=3, sigma=1)

    # team-specific model parameters
    atts_star = pm.Normal("atts_star", mu=0, sigma=sd_att, dims="team")
    defs_star = pm.Normal("defs_star", mu=0, sigma=sd_def, dims="team")

    atts = pm.Deterministic("atts", atts_star - pt.mean(atts_star), dims="team")
    defs = pm.Deterministic("defs", defs_star - pt.mean(defs_star), dims="team")
    home_theta = pt.exp(intercept + home + atts[home_idx] + defs[away_idx])
    away_theta = pt.exp(intercept + atts[away_idx] + defs[home_idx])

    # likelihood of observed data
    home_points = pm.Poisson(
        "home_points",
        mu=home_theta,
        observed=df_all["home_score"],
        dims=("match"),
```

Skip to main content

```
            "away_points",
            mu=away_theta,
            observed=df_all["away_score"],
            dims=("match"),
        )
        trace = pm.sample(1000, tune=1500, cores=4)
```

```
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
/home/oriol/miniconda3/envs/arviz/lib/python3.9/site-packages/pymc/pytensorf.py:1005: Use
  pytensor_function = pytensor.function(
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [home, sd_att, sd_def, intercept, atts_star, defs_star]
```

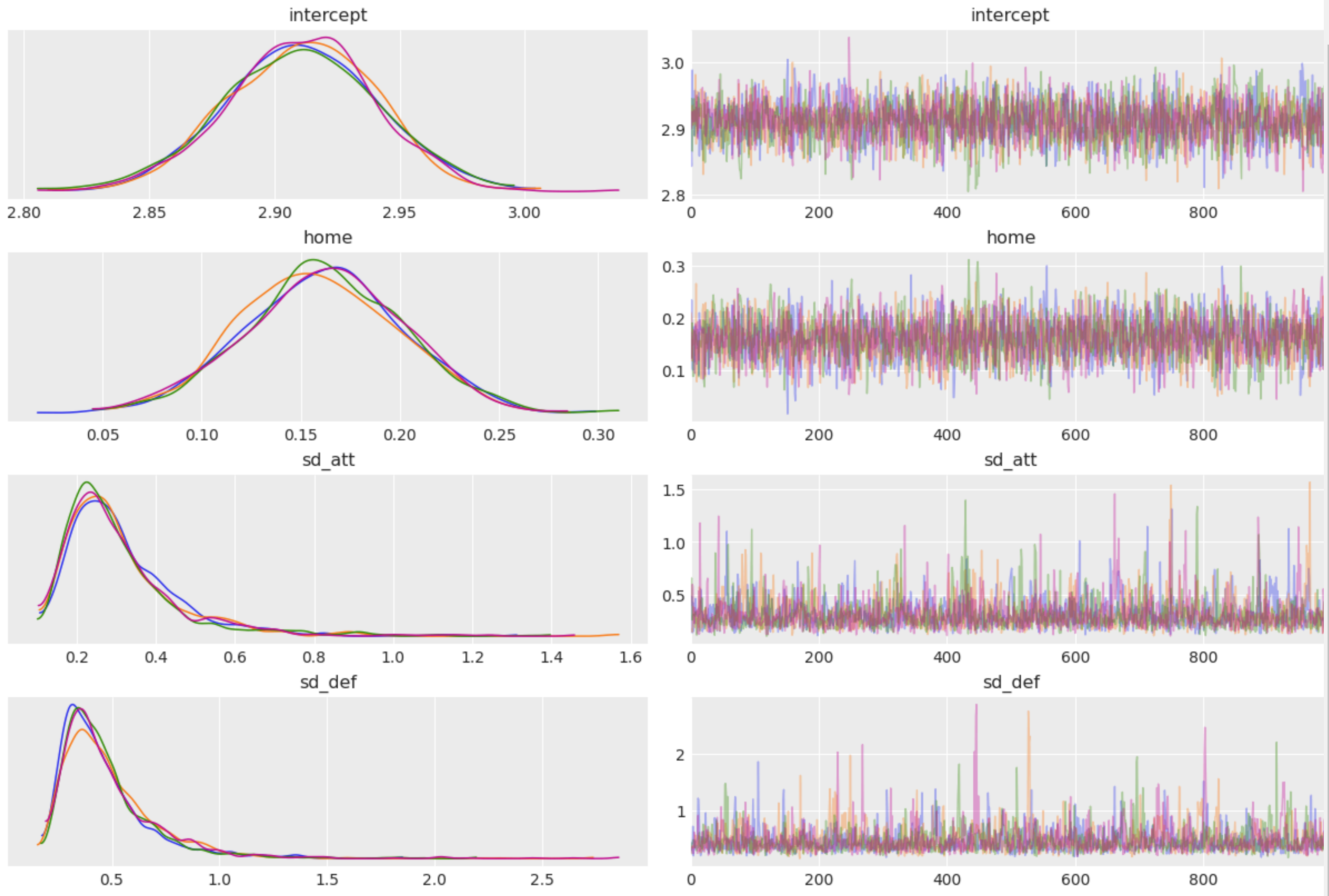
100.00% [10000/10000 00:24<00:00 Sampling 4 chains, (
divergences]

```
Sampling 4 chains for 1_500 tune and 1_000 draw iterations (6_000 + 4_000 draws total) to
```

- We specified the model and the likelihood function
- All this runs on an PyTensor graph under the hood

```
az.plot_trace(trace, var_names=["intercept", "home", "sd_att", "sd_def"], compact=False);
```

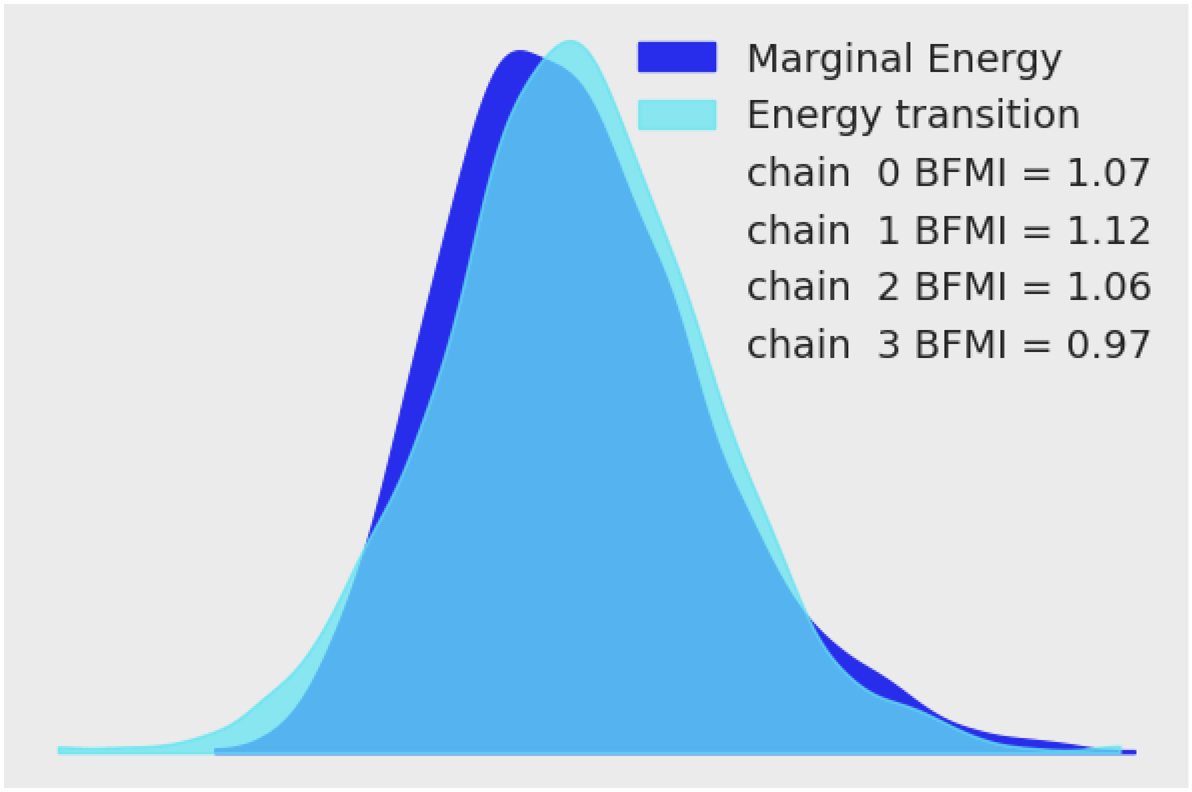
Skip to main content



Let us apply good *statistical workflow* practices and look at the various evaluation metrics to see if our NUTS sampler converged.

```
az.plot_energy(trace, figsize=(6, 4));
```

Skip to main content



```
az.summary(trace, kind="diagnostics")
```

	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
home	0.001	0.001	2694.0	2250.0	1.0
intercept	0.001	0.000	2743.0	2455.0	1.0
atts_star[England]	0.004	0.003	1243.0	1121.0	1.0
atts_star[France]	0.004	0.003	1247.0	1111.0	1.0
atts_star[Ireland]	0.004	0.003	1208.0	1160.0	1.0
atts_star[Italy]	0.004	0.003	1352.0	1375.0	1.0
atts_star[Scotland]	0.004	0.003	1318.0	1178.0	1.0
atts_star[Wales]	0.004	0.003	1252.0	1122.0	1.0
defs_star[England]	0.007	0.005	1110.0	899.0	1.0
defs_star[France]	0.007	0.006	1091.0	876.0	1.0
defs_star[Ireland]	0.007	0.005	1113.0	881.0	1.0

Skip to main content

	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
defs_star[Scotland]	0.007	0.006	1081.0	854.0	1.0
defs_star[Wales]	0.007	0.006	1098.0	889.0	1.0
sd_att	0.004	0.003	1767.0	1709.0	1.0
sd_def	0.006	0.004	1762.0	1662.0	1.0
atts[England]	0.001	0.000	5489.0	3204.0	1.0
atts[France]	0.001	0.000	5441.0	3286.0	1.0
atts[Ireland]	0.001	0.000	5257.0	3260.0	1.0
atts[Italy]	0.001	0.001	5157.0	2957.0	1.0
atts[Scotland]	0.001	0.001	4926.0	3003.0	1.0
atts[Wales]	0.001	0.000	4537.0	3306.0	1.0
defs[England]	0.001	0.001	4779.0	3293.0	1.0
defs[France]	0.001	0.001	4183.0	2917.0	1.0
defs[Ireland]	0.001	0.001	4766.0	3227.0	1.0
defs[Italy]	0.001	0.000	4419.0	3561.0	1.0
defs[Scotland]	0.001	0.000	4068.0	3290.0	1.0
defs[Wales]	0.001	0.001	4177.0	3164.0	1.0

Our model has converged well and $\hat{R}$ looks good.

Let us look at some of the stats, just to verify that our model has returned the correct attributes. We can see that some teams are stronger than others. This is what we would expect with attack

```
trace_hdi = az.hdi(trace)
trace_hdi["atts"]
```

xarray.DataArray 'atts' (**team**: 6, **hdi**: 2)

```
array([[ 0.18302232,  0.33451681],
       [-0.16672247,  0.00056686],
```

Skip to main content

```
        [-0.20865349, -0.03104828],
        [ 0.09821099,  0.25031702]])
```

▼ Coordinates:

team (team) <U8 'England' 'France' ... 'Wales'

hdi (hdi) <U6 'lower' 'higher'

► Attributes: (0)

```
trace.posterior["atts"].median(("chain", "draw"))
```

xarray.DataArray 'atts' (**team**: 6)

```
array([ 0.25676819, -0.08392998,  0.10809197, -0.33498374, -0.11633869,
        0.17182582])
```

▼ Coordinates:

team (team) <U8 'England' 'France' ... 'Wales'

► Attributes: (0)

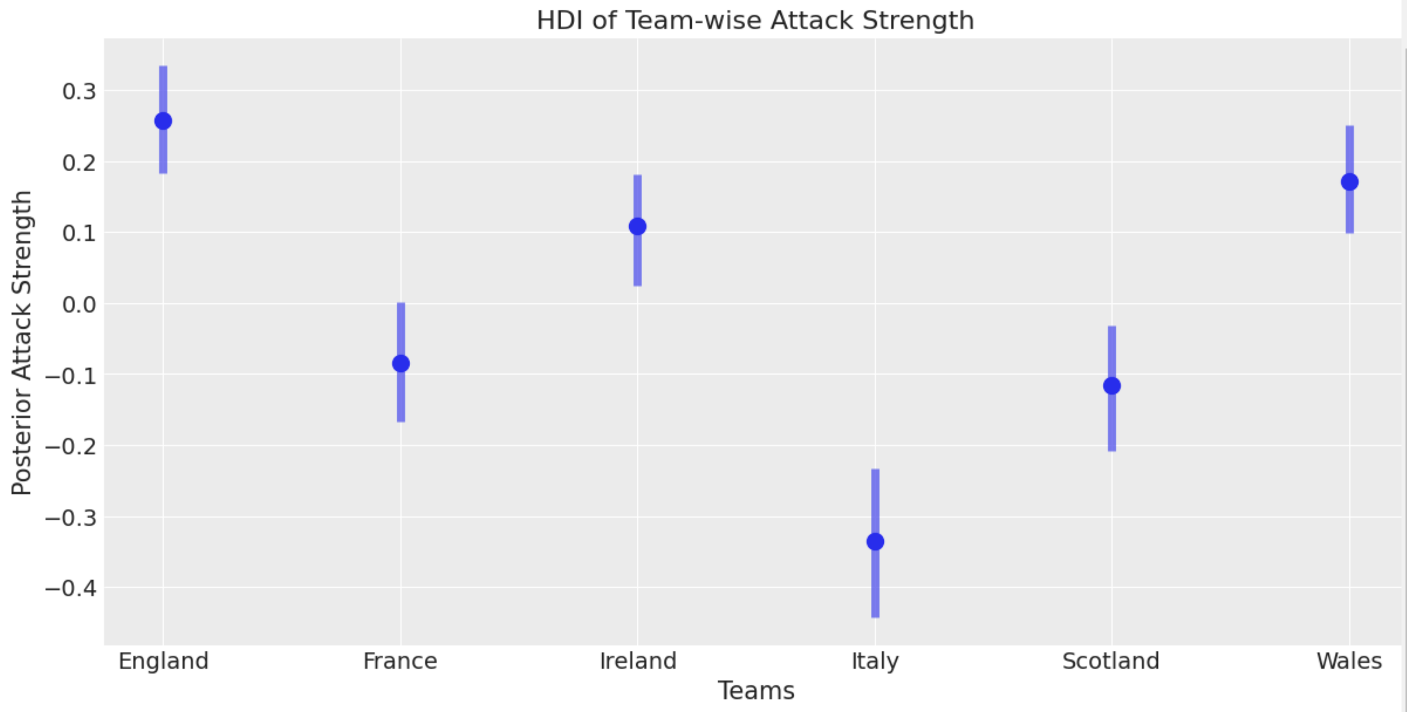
Results

From the above we can start to understand the different distributions of attacking strength and defensive strength. These are probabilistic estimates and help us better understand the uncertainty in sports analytics

```
_, ax = plt.subplots(figsize=(12, 6))

ax.scatter(teams, trace.posterior["atts"].median(dim=("chain", "draw")), color="C0", alpha
ax.vlines(
    teams,
    trace_hdi["atts"].sel({"hdi": "lower"}),
    trace_hdi["atts"].sel({"hdi": "higher"}),
    alpha=0.6,
    lw=5,
    color="C0",
)
ax.set_xlabel("Teams")
ax.set_ylabel("Posterior Attack Strength")
ax.set_title("HDI of Team-wise Attack Strength");
```

Skip to main content



This is one of the powerful things about Bayesian modelling, we can have *uncertainty quantification* of some of our estimates. We've got a Bayesian credible interval for the attack strength of different countries.

We can see an overlap between Ireland, Wales and England which is what you'd expect since these teams have won in recent years.

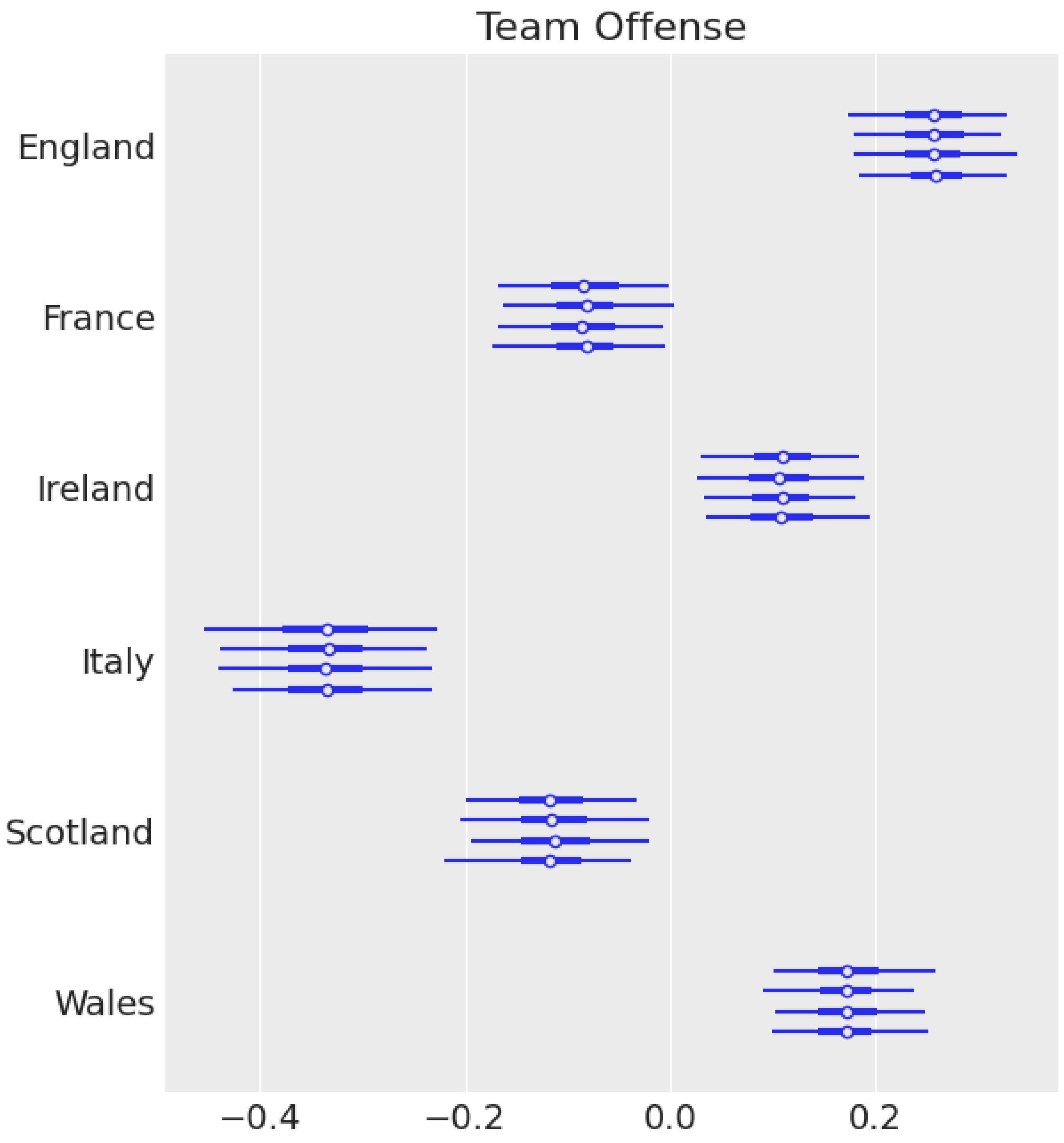
Italy is well behind everyone else - which is what we'd expect and there's an overlap between Scotland and France which seems about right.

There are probably some effects we'd like to add in here, like weighting more recent results more strongly. However that'd be a much more complicated model.

```
# subclass arviz labeller to omit the variable name
class TeamLabeller(az.labels.BaseLabeller):
    def make_label_flat(self, var_name, sel, isel):
        sel_str = self.sel_to_str(sel, isel)
        return sel_str
```

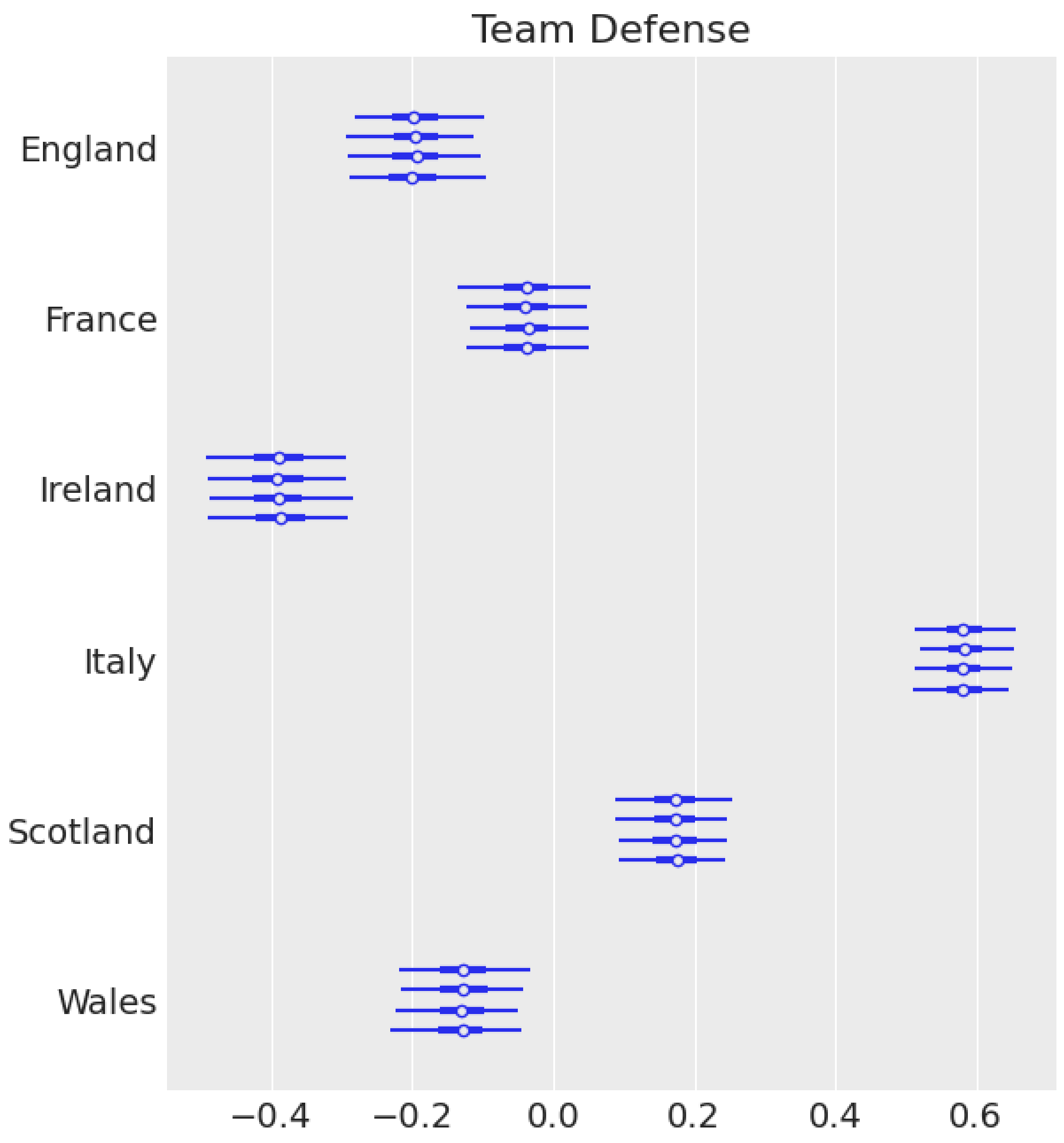
```
ax = az.plot_forest(trace, var_names=["atts"], labeller=TeamLabeller())
ax[0].set_title("Team Offense");
```

Skip to main content



```
ax = az.plot_forest(trace, var_names=["defs"], labeller=TeamLabeller())
ax[0].set_title("Team Defense");
```

Skip to main content



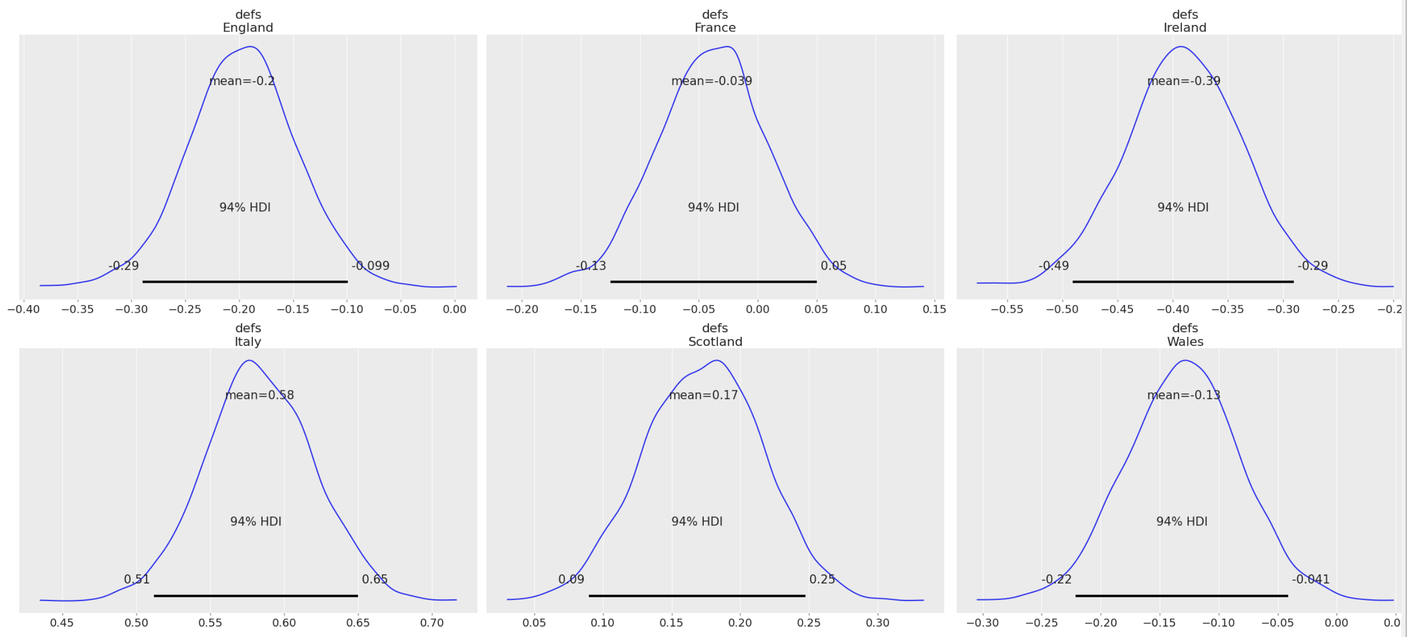
Good teams like Ireland and England have a strong negative effect defense. Which is what we expect. We expect our strong teams to have strong positive effects in attack and strong negative effects in defense.

This approach that we're using of looking at parameters and examining them is part of a good statistical workflow. We also think that perhaps our priors could be better specified. However this is beyond the scope of this article. We recommend for a good discussion of 'statistical workflow' you visit Robust Statistical Workflow with RStan

Let's do some other plots. So we can see our range for our defensive effect. I'll print the teams below too just for reference

Skip to main content

```
az.plot_posterior(trace, var_names=["defs"]);
```



We can see that Ireland's mean is -0.39 which means we expect Ireland to have a strong defense. Which is what we'd expect, Ireland generally even in games it loses doesn't lose by say 50 points. And we can see that the 94% HDI is between -0.491, and -0.28

In comparison with Italy, we see a strong positive effect 0.58 mean and a HDI of 0.51 and 0.65. This means that we'd expect Italy to concede a lot of points, compared to what it scores. Given that Italy often loses by 30 - 60 points, this seems correct.

We see here also that this informs what other priors we could bring into this. We could bring some sort of world ranking as a prior.

As of December 2017 the rugby rankings indicate that England is 2nd in the world, Ireland 3rd, Scotland 5th, Wales 7th, France 9th and Italy 14th. We could bring that into a model and it can explain some of the fact that Italy is apart from a lot of the other teams.

Now let's simulate who wins over a total of 4000 simulations, one per sample in the posterior.

```
with model:
    pm.sample_posterior_predictive(trace, extend_inferencedata=True)
pp = trace.posterior_predictive
const = trace.constant_data
team_da = trace.posterior.team
```

Skip to main content

```
/home/oriol/miniconda3/envs/arviz/lib/python3.9/site-packages/pymc/pytensorf.py:1005: Use
  pytensor_function = pytensor.function(
```

100.00% [4000/4000 00:00<00:00]

The posterior predictive samples contain the goals scored by each team in each match. We modelec and therefore simulated according to scoring and devensive powers using goals as observed variable.

Our goal now is to see who wins the competition, so we can estimate the probability each team has of winning the whole competition. From that we need to convert the scored goals to points:

```
# fmt: off
pp["home_win"] = (
    (pp["home_points"] > pp["away_points"]) * 3     # home team wins and gets 3 points
    + (pp["home_points"] == pp["away_points"]) * 2  # tie -> home team gets 2 points
)
pp["away_win"] = (
    (pp["home_points"] < pp["away_points"]) * 3
    + (pp["home_points"] == pp["away_points"]) * 2
)
# fmt: on
```

Then add the points each team has collected throughout all matches:

```
groupby_sum_home = pp.home_win.groupby(team_da[const.home_team]).sum()
groupby_sum_away = pp.away_win.groupby(team_da[const.away_team]).sum()

pp["teamscores"] = groupby_sum_home + groupby_sum_away
```

And eventually generate the ranks of all teams for each of the 4000 simulations. As our data is stored in xarray objects inside the InferenceData class, we will use xarray-einstats:

```
from xarray_einstats.stats import rankdata

pp["rank"] = rankdata(-pp["teamscores"], dims="team", method="min")
pp[["rank"]].sel(team="England")
```

xarray.Dataset

Skip to main content

```
▼ Coordinates:
    chain             (chain)          int64  0 1 2 3
    draw              (draw)           int64  0 1 2 3 4 5 ... 995 996 997 998 999
    team              ()               <U7    'England'
▼ Data variables:
    rank              (chain, draw)    int64  2 1 2 2 2 1 1 1 ... 1 2 1 3 1 2 2 1
▼ Attributes:
    created_at :            2022-04-02T00:25:59.622442
    arviz_version :         0.12.0
    inference_libra...      pymc
    inference_libra...      4.0.0b6
```

As you can see, we now have a collection of 4000 integers between 1 and 6 for each team, 1 meaning they win the competition. We can use a histogram with bin edges at half integers to count and normalize how many times each team finishes in each position:

```python
from xarray_einstats.numba import histogram

bin_edges = np.arange(7) + 0.5
data_sim = (
    histogram(pp["rank"], dims=("chain", "draw"), bins=bin_edges, density=True)
    .rename({"bin": "rank"})
    .assign_coords(rank=np.arange(6) + 1)
)
```

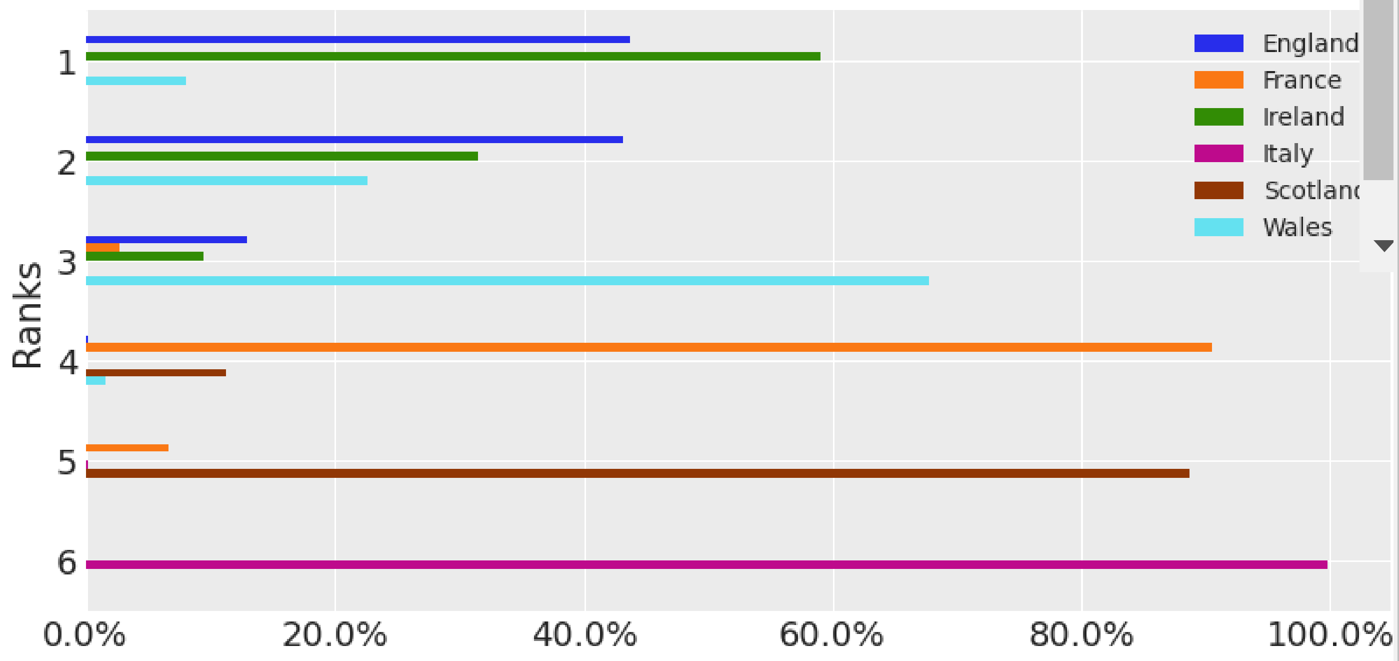
Now that we have reduced the data to a 2 dimensional array, we will convert it to a pandas DataFrame which is now a more adequate choice to work with our data:

```python
idx_dim, col_dim = data_sim.dims
sim_table = pd.DataFrame(data_sim, index=data_sim[idx_dim], columns=data_sim[col_dim])
```

```python
fig, ax = plt.subplots(figsize=(8, 4))
ax = sim_table.T.plot(kind="barh", ax=ax)
ax.xaxis.set_major_formatter(StrMethodFormatter("{x:.1%}"))
ax.set_xlabel("Rank-wise Probability of results for all six teams")
ax.set_yticklabels(np.arange(1, 7))
```

Skip to main content

```
ax.invert_yaxis()
```



Peadar Coyle , Meenal Jhajharia , Oriol Abril-Pla . "A Hierarchical model for Rugby prediction". In: *PyMC Examples*. Ed. by PyMC Team. DOI: 10.5281/zenodo.5654871

We see according to this model that Ireland finishes with the most points about 60% of the time,

© Copyright 2022, PyMC Community.

Created using Sphinx 7.3.7.

Built with the PyData Sphinx Theme 0.14.4.