In [1]:
```
%matplotlib inline
```

In [2]:
```
import feather as ft
import pandas as pd
import numpy as np
import pymc3 as pm
import theano.tensor as T
```

```
/home/paulc/anaconda3/envs/py36/lib/python3.6/site-packages/h5py/__init__.py:36: FutureWarning: Conversion of the second argument of issubdtype from `float` to `np.floating` is deprecated. In future, it will be treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
```

First we have to get the bangladesh dataset. Feather (https://github.com/wesm/feather) is perfect for this. Thanks to Hadley Wickham and Wes McKinney for producing this invaluable set of tools.

The code to export from R should look something like this, assuming you've installed rethinking and feather packages and their dependencies:

```
library(rethinking)
library (feather)

data(bangladesh)
path <- "."
write_feather("bangladesh.feather", path)
```

That's all the R we'll need for now.

Next step is to read in the data

In [3]:
```
# Read the bangledesh data from home directory
bangladesh = ft.read_dataframe("bangladesh.feather")
bangladesh.head()
```

Out[3]:

	woman	district	use.contraception	living.children	age.centered	urban
0	1	1	0	4	18.4400	1
1	2	1	0	1	-5.5599	1
2	3	1	0	3	1.4400	1
3	4	1	0	4	8.4400	1
4	5	1	0	1	-13.5590	1

From here on we're following Richard McElreath's blog.

We set the random seed so we can reproduce the results as required and we sample 100 records from the bangladesh set.

In [4]:
```
seed = np.random.RandomState(seed=1)

b = bangladesh.sample(100, random_state=seed)
b.head()
```

Out[4]:

	woman	district	use.contraception	living.children	age.centered	urban
661	662	18	0	3	-6.5599	0
1047	1048	31	0	4	16.4400	1
643	644	18	0	3	1.4400	0
1712	1713	52	1	4	10.4400	0
1507	1508	46	1	4	7.4400	0

We only need contraception use tied to district. The data dataframe will hold this information.

In [5]:
```
dat = pd.DataFrame()
dat['y'] = b['use.contraception']
dat['district_id'] = pd.Categorical(b['district']).codes
dat.head()
```

Out[5]:

	y	district_id
661	0	15
1047	0	24
643	0	15
1712	1	39
1507	1	36

Our first model is the **centered parameterization**:

$$\begin{aligned} y_i &\sim \text{Bernoulli}(p_i) \\ \text{logit}(p_i) &= \alpha_{unit[i]} \\ \alpha_j &\sim \text{Normal}(a, \tau) \\ a &\sim \text{Normal}(0, 10) \\ \tau &\sim \text{Half-Cauchy}(0, 1) \end{aligned}$$

In [6]:
```
with pm.Model() as c_model:
    tau = pm.HalfCauchy('tau',1)
    a = pm.Normal('a', 0, 10)
    alpha = pm.Normal('alpha', a, tau, shape=len(b['district'].unique()))
    p = pm.math.invlogit(alpha[dat['district_id'].values])
    y = pm.Bernoulli('y', p, observed=dat['y'])

#     for RV in c_model.basic_RVs:
#         print(RV.name, RV.logp(c_model.test_point))
    trace_c = pm.sample(1000, tune=1000)
```

```
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
/home/paulc/anaconda3/envs/py36/lib/python3.6/site-packages/pymc3/model.py:
384: FutureWarning: Conversion of the second argument of issubdtype from `f
loat` to `np.floating` is deprecated. In future, it will be treated as `np.
float64 == np.dtype(float).type`.
  if not np.issubdtype(var.dtype, float):
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [alpha, a, tau_log__]
 57%|█████     | 1135/2000 [00:05<00:03, 219.45it/s]INFO (theano.gof.compil
elock): Waiting for existing lock by process '16081' (I am process '16082')
INFO (theano.gof.compilelock): To manually release the lock, delete /home/p
aulc/.theano/compiledir_Linux-4.13--generic-x86_64-with-debian-stretch-sid-
x86_64-3.6.3-64/lock_dir
 59%|█████     | 1172/2000 [00:05<00:03, 222.26it/s]INFO (theano.gof.compil
elock): Waiting for existing lock by process '16081' (I am process '16083')
INFO (theano.gof.compilelock): To manually release the lock, delete /home/p
aulc/.theano/compiledir_Linux-4.13--generic-x86_64-with-debian-stretch-sid-
x86_64-3.6.3-64/lock_dir
100%|██████████| 2000/2000 [00:08<00:00, 238.60it/s]
INFO (theano.gof.compilelock): Waiting for existing lock by process '16082'
(I am process '16083')
INFO (theano.gof.compilelock): To manually release the lock, delete /home/p
aulc/.theano/compiledir_Linux-4.13--generic-x86_64-with-debian-stretch-sid-
x86_64-3.6.3-64/lock_dir
There were 17 divergences after tuning. Increase `target_accept` or reparam
eterize.
The acceptance probability does not match the target. It is 0.6809466137372
715, but should be close to 0.8. Try to increase the number of tuning steps
.
There were 2 divergences after tuning. Increase `target_accept` or reparame
terize.
There were 60 divergences after tuning. Increase `target_accept` or reparam
eterize.
There were 6 divergences after tuning. Increase `target_accept` or reparame
terize.
The estimated number of effective samples is smaller than 200 for some para
meters.
```

As with the R Stan version, there's a whole bunch of divergences in this log. This suggests that the results may not be reliable.

Our second model is the **non-centered parameterization**:

$$
\begin{aligned}
y_i &\sim \text{Bernoulli}(p_i) \\
\text{logit}(p_i) &= a + \tau z_{unit[i]} \\
z_j &\sim \text{Normal}(0, 1) \\
a &\sim \text{Normal}(0, 10) \\
\tau &\sim \text{Half-Cauchy}(0, 1)
\end{aligned}
$$

In [7]:
```
with pm.Model() as nc_model:
    # pymc3 HalfCauchy only uses the second (scale) parameter
    tau = pm.HalfCauchy('tau', beta=1)
    a = pm.Normal('a', 0, 10)
    z = pm.Normal('z', 0, 1, shape=len(b['district'].unique()))
    p = pm.math.invlogit(a + tau * z[dat['district_id'].values])
    y = pm.Bernoulli('y', p, observed=dat['y'])

#     for RV in c_model.basic_RVs:
#         print(RV.name, RV.logp(c_model.test_point))
    trace_nc = pm.sample(1000, tune=1000)
```

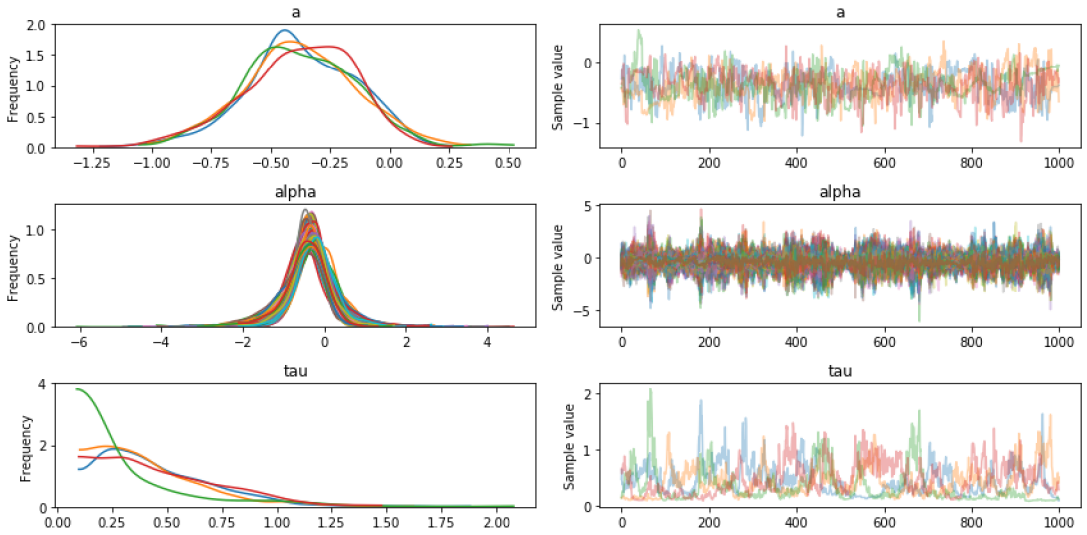
```
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
/home/paulc/anaconda3/envs/py36/lib/python3.6/site-packages/pymc3/model.py:
384: FutureWarning: Conversion of the second argument of issubdtype from `f
loat` to `np.floating` is deprecated. In future, it will be treated as `np.
float64 == np.dtype(float).type`.
  if not np.issubdtype(var.dtype, float):
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [z, a, tau_log__]
100%|██████████| 2000/2000 [00:06<00:00, 302.29it/s]
There were 3 divergences after tuning. Increase `target_accept` or reparame
terize.
The acceptance probability does not match the target. It is 0.7201418506887
862, but should be close to 0.8. Try to increase the number of tuning steps
.
There were 3 divergences after tuning. Increase `target_accept` or reparame
terize.
The acceptance probability does not match the target. It is 0.6577402127331
399, but should be close to 0.8. Try to increase the number of tuning steps
.
```

This log looks much better. That suggests already that we might see more reliable results from this approach.

Let's take a look at the traceplots:

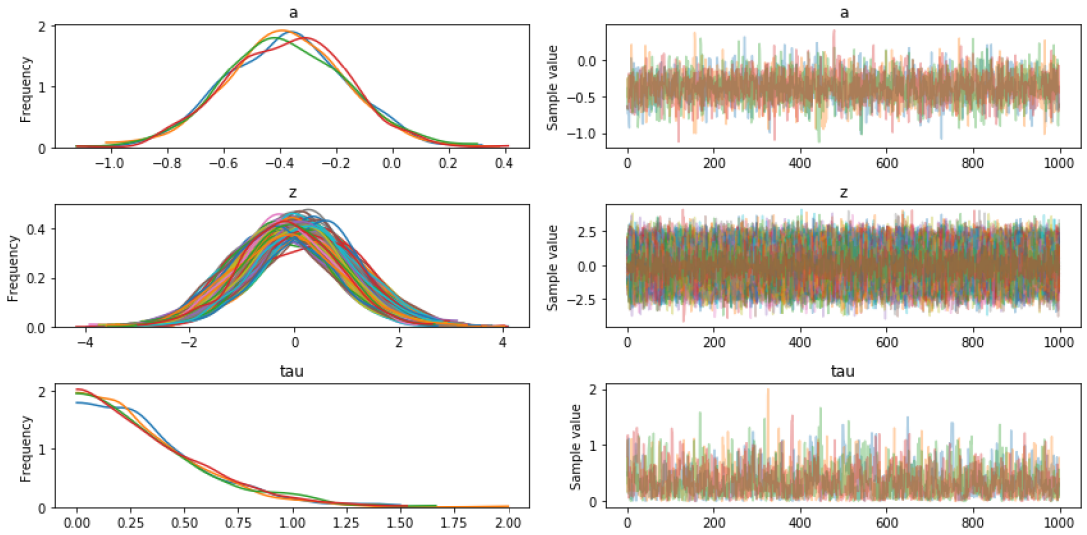
In [8]:
```
_ = pm.plots.traceplot(trace_c)
d = pm.diagnostics.effective_n(trace_c)
print(f"n_eff('a') = {d['a']}\nn_eff('tau') = {d['tau']}")
```

```
n_eff('a') = 101.0
n_eff('tau') = 53.0
```



In [9]:
```
_ = pm.plots.traceplot(trace_nc)
d = pm.diagnostics.effective_n(trace_nc)
print(f"n_eff('a') = {d['a']}\nn_eff('tau') = {d['tau']}")
```

```
n_eff('a') = 3319.0
n_eff('tau') = 1321.0
```



These reflect the results from the blog.

The centered model has very low effective samples, the distributions look quite spread out.

The non-centered model has decent effective samples and the traceplots look clean with relatively tightly clustered distributions.