Probabilistic Programming

Marius Popescu

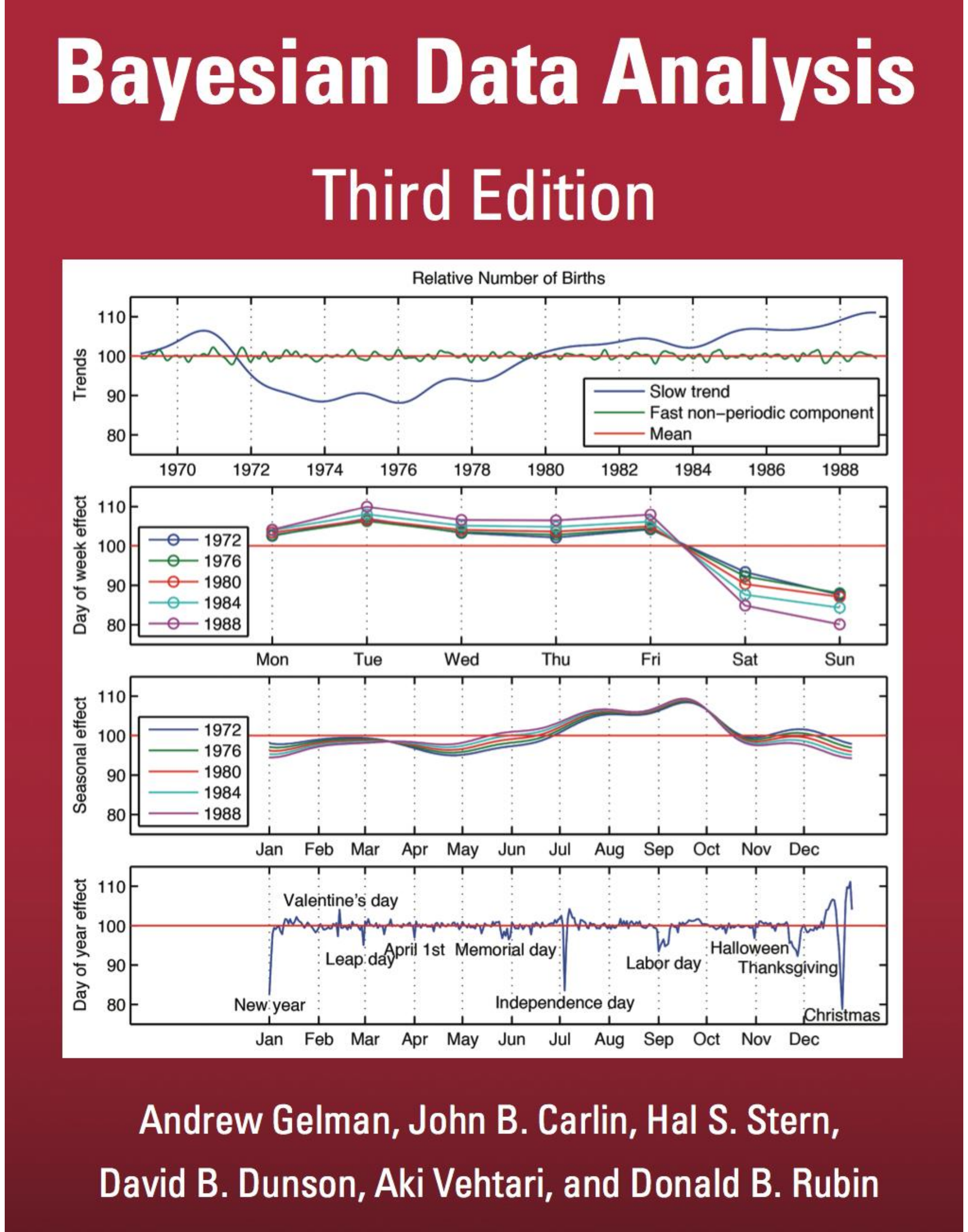
popescunmarius@gmail.com

2019 - 2020

Hierarchical Models

From PyMC2 to PyMC4 via TFP

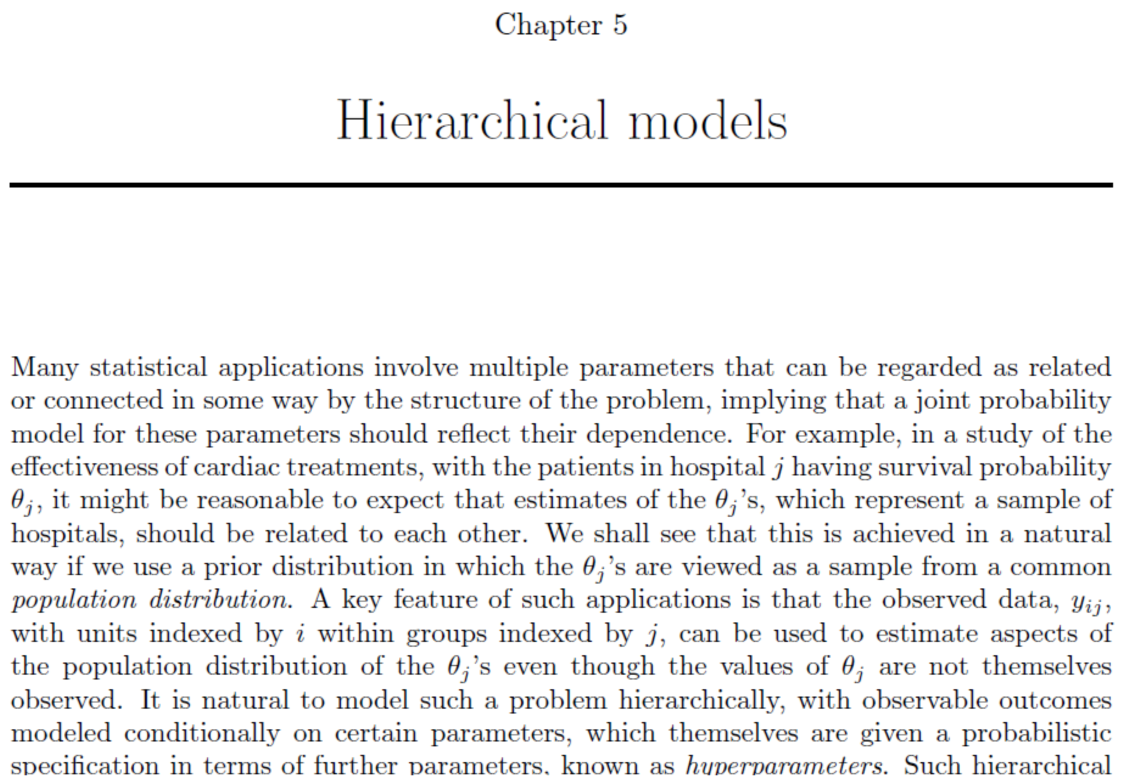
Hierarchical Models



Bayesian Data Analysis

Third Edition

Andrew Gelman, John B. Carlin, Hal S. Stern, David B. Dunson, Aki Vehtari, and Donald B. Rubin



Chapter 5

Hierarchical models

Many statistical applications involve multiple parameters that can be regarded as related or connected in some way by the structure of the problem, implying that a joint probability model for these parameters should reflect their dependence. For example, in a study of the effectiveness of cardiac treatments, with the patients in hospital j having survival probability θ_j, it might be reasonable to expect that estimates of the θ_j's, which represent a sample of hospitals, should be related to each other. We shall see that this is achieved in a natural way if we use a prior distribution in which the θ_j's are viewed as a sample from a common *population distribution*. A key feature of such applications is that the observed data, y_{ij}, with units indexed by i within groups indexed by j, can be used to estimate aspects of the population distribution of the θ_j's even though the values of θ_j are not themselves observed. It is natural to model such a problem hierarchically, with observable outcomes modeled conditionally on certain parameters, which themselves are given a probabilistic specification in terms of further parameters, known as *hyperparameters*. Such hierarchical

Hierarchical Models

Many statistical applications involve multiple parameters that can be regarded as related or connected in some way by the structure of the problem, implying that a joint probability model for these parameters should reflect their dependence.

For example, in a study of the effectiveness of cardiac treatments, with the patients in hospital j having survival probability θ_j, it might be reasonable to expect that estimates of the θ_j's, which represent a sample of hospitals, should be related to each other. We shall see that this is achieved in a natural way if we use a prior distribution in which the θ_j's are viewed as a sample from a *common population distribution*. A key feature of such applications is that the observed data, y_{ij}, with units indexed by i within groups indexed by j, can be used to estimate aspects of the population distribution of the θ_j's even though the values of θ_j are not themselves observed.

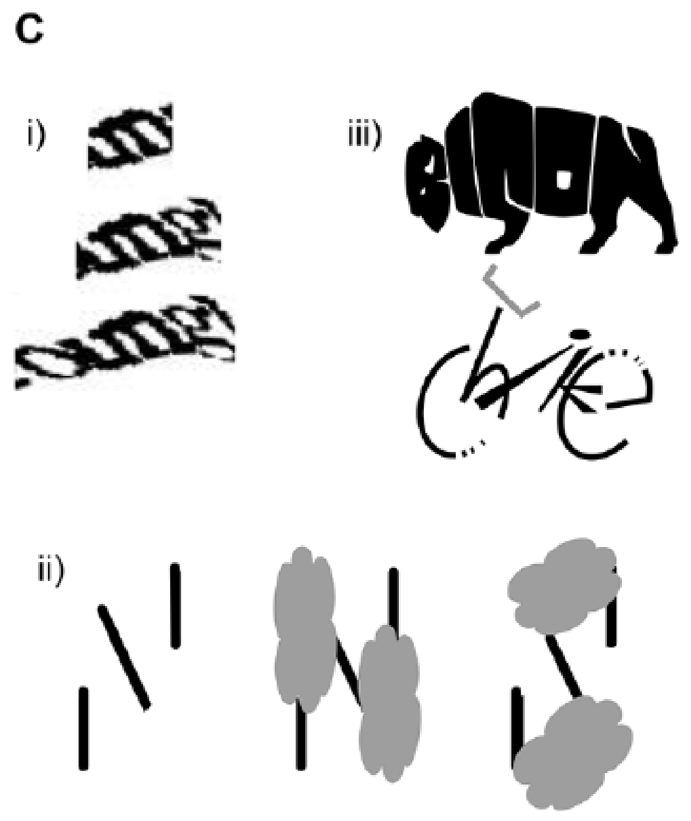
Hierarchical Models

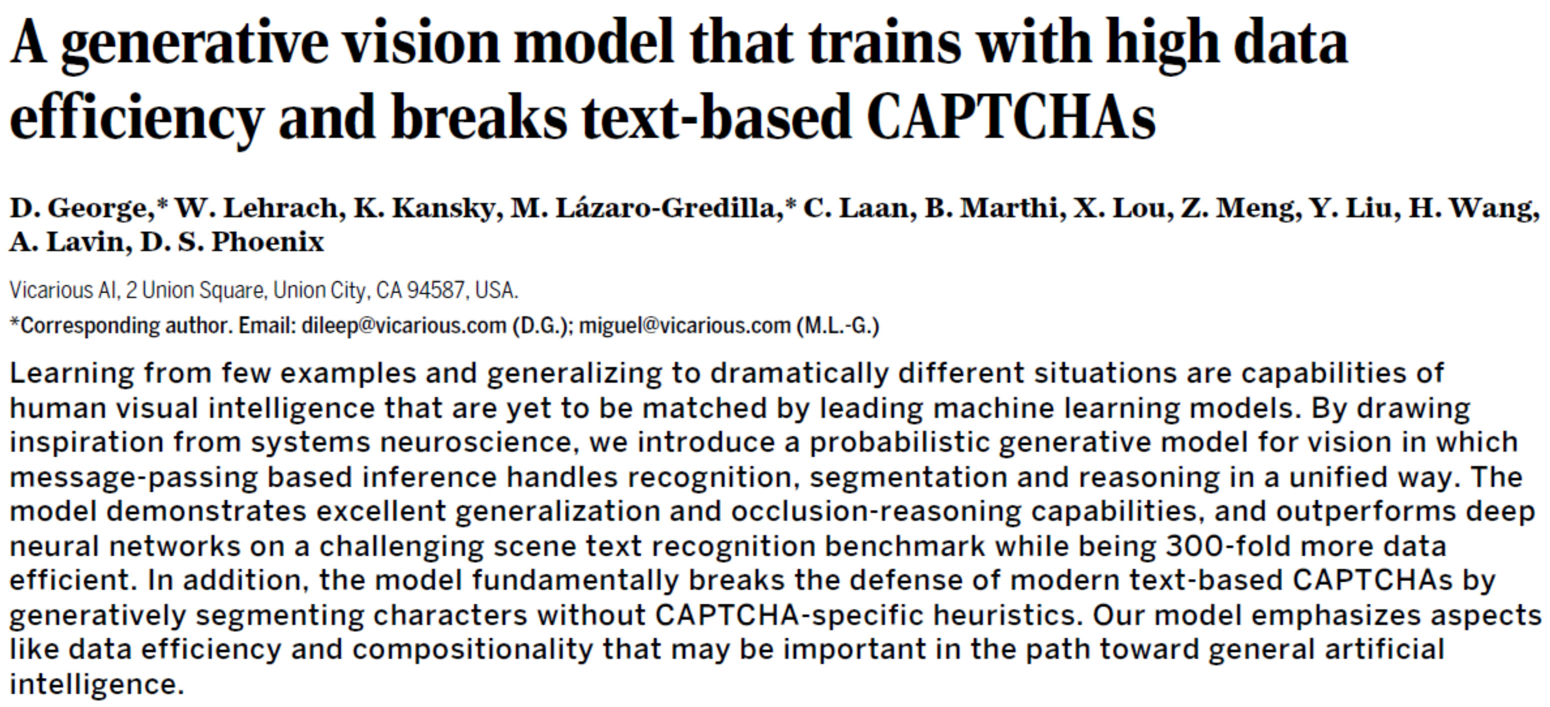
“It is natural to model such a problem hierarchically, with observable outcomes modeled conditionally on certain parameters, which themselves are given a probabilistic specification in terms of further parameters, known as *hyperparameters*. Such hierarchical thinking helps in understanding multiparameter problems and also plays an important role in developing computational strategies.“

Why Hierarchical Models Matter?

A Cognitive Perspective

Common sense and context affect letter form perception: (i) m vs u and n. (ii) the same line segments are interpreted as N or S depending on occluder positions (iii) perception of the shapes aids the recognition .





A generative vision model that trains with high data efficiency and breaks text-based CAPTCHAs

D. George,* W. Lehrach, K. Kansky, M. Lázaro-Gredilla,* C. Laan, B. Marthi, X. Lou, Z. Meng, Y. Liu, H. Wang, A. Lavin, D. S. Phoenix

Vicarious AI, 2 Union Square, Union City, CA 94587, USA.

*Corresponding author. Email: dileep@vicarious.com (D.G.); miguel@vicarious.com (M.L.-G.)

Learning from few examples and generalizing to dramatically different situations are capabilities of human visual intelligence that are yet to be matched by leading machine learning models. By drawing inspiration from systems neuroscience, we introduce a probabilistic generative model for vision in which message-passing based inference handles recognition, segmentation and reasoning in a unified way. The model demonstrates excellent generalization and occlusion-reasoning capabilities, and outperforms deep neural networks on a challenging scene text recognition benchmark while being 300-fold more data efficient. In addition, the model fundamentally breaks the defense of modern text-based CAPTCHAs by generatively segmenting characters without CAPTCHA-specific heuristics. Our model emphasizes aspects like data efficiency and compositionality that may be important in the path toward general artificial intelligence.

www.sciencemag.org/cgi/content/full/science.aag2612/DC1

Supplementary Materials for

A generative vision model that trains with high data efficiency and breaks text-based CAPTCHAs

D. George,* W. Lehrach, K. Kansky, M. Lázaro-Gredilla,* C. Laan, B. Marthi, X. Lou, Z. Meng, Y. Liu, H. Wang, A. Lavin, D. S. Phoenix

*Corresponding author. Email: dileep@vicarious.com (D.G.); miguel@vicarious.com (M.L.-G.)

Contents

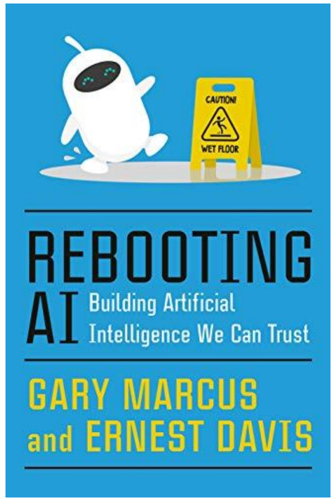
Perception as Inference

The properties of the world that we care about - which drive behavior - are not directly provided by sensory input. There are no sensors that measure surface shape, motion of objects, material properties, or object identity. Rather, these properties are entangled among multiple sensor values and must be *disentangled* to be made explicit.

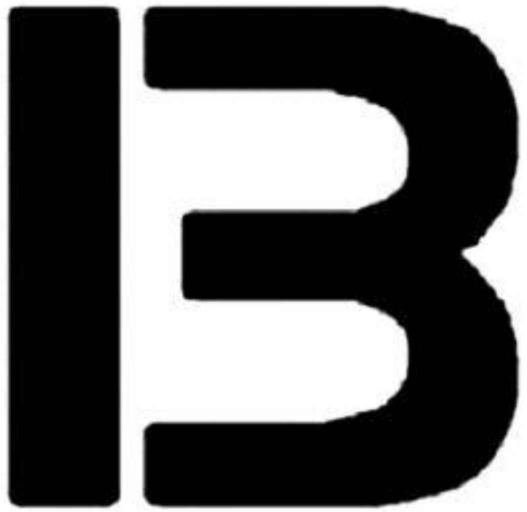
Importantly, the nature of these disentangling problems is that they are often ill-posed, meaning that there is not enough information provided by the sensory data to uniquely recover the properties of interest. In other words, the various aspects of a scene that are needed to drive behavior can not simply be deduced from sensory measurements. Rather, they must be inferred by combining sensory data together with prior knowledge.

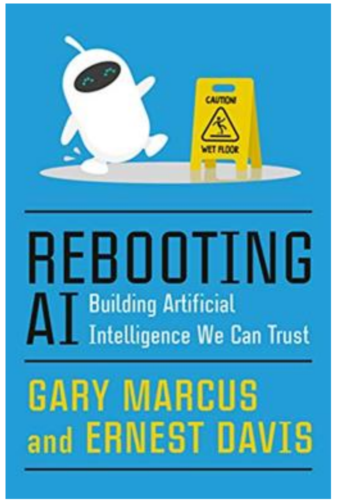
A Robust Understanding of the World Requires Both Top-Down and Bottom-Up Information

“ Cognitive psychologists often distinguish between two kinds of knowledge: bottom-up information, which is information that comes directly from our senses, and top-down knowledge, which is our prior knowledge about the world (for example, that letters and numbers form distinct categories, that words and numbers are composed from elements drawn from those categories, and so forth). An ambiguous image looks one way in one context and another in a different context because we try to integrate the light falling on our retina with a coherent picture of the world. ”

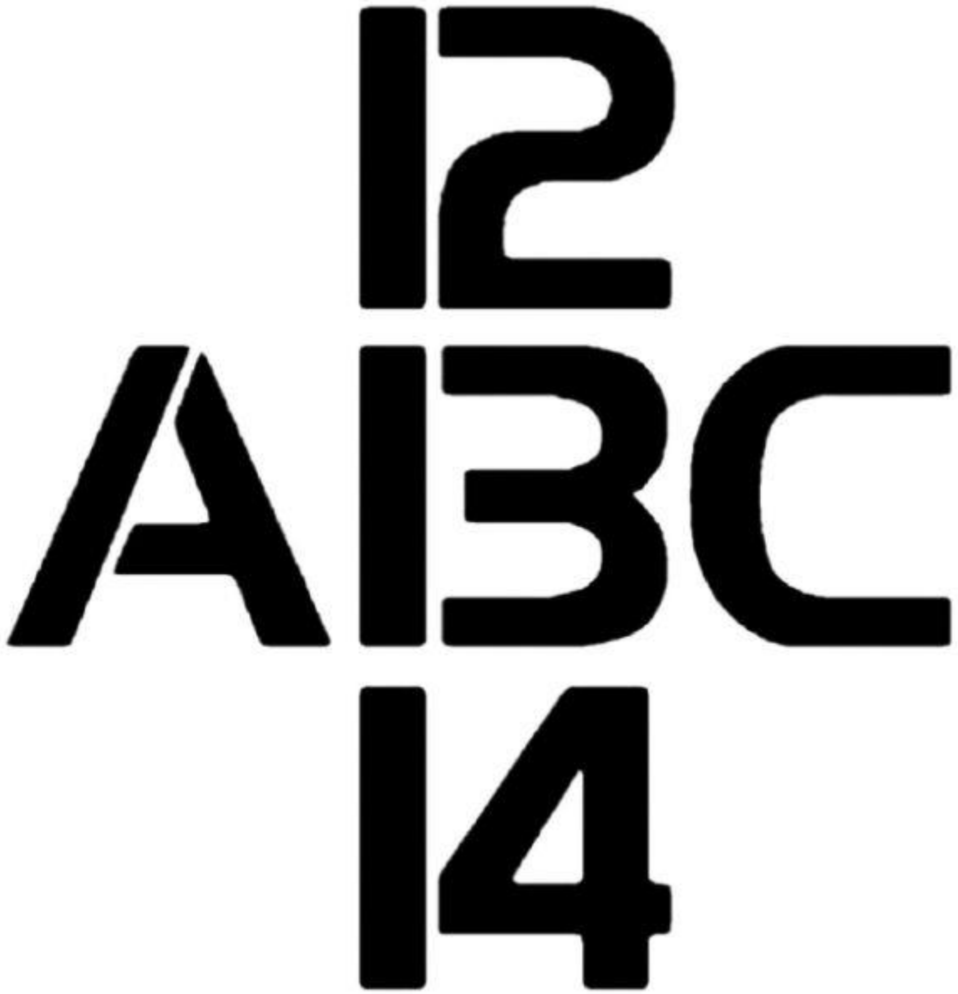

REBOOTING AI
Building Artificial Intelligence We Can Trust
GARY MARCUS and ERNEST DAVIS

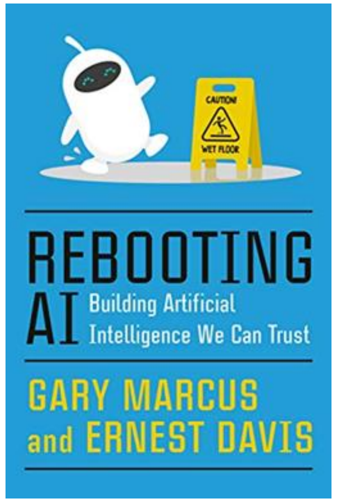
Letter B or Numeral 13?



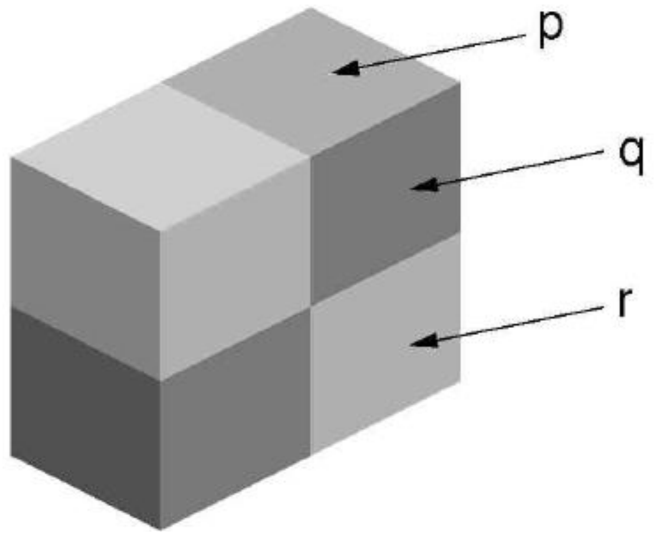


Letter B or Numeral 13?
Interpretation depend on Context




REBOOTING AI
Building Artificial Intelligence We Can Trust
GARY MARCUS and ERNEST DAVIS

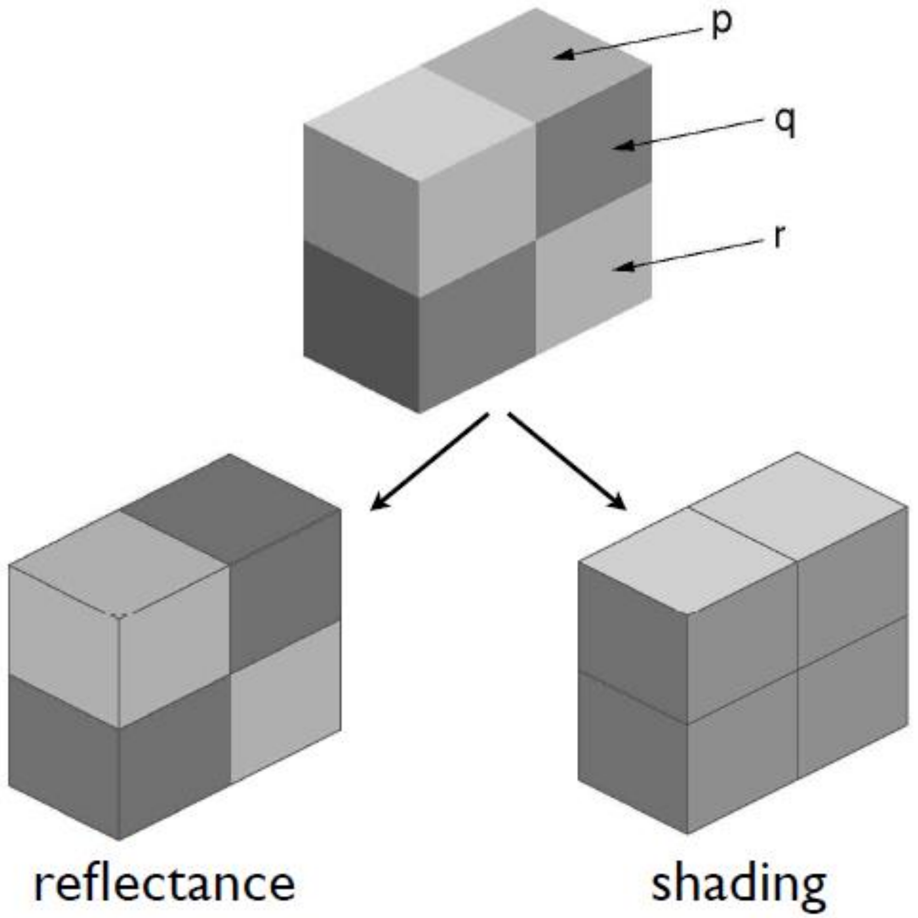
Perception as Inference



Computing a representation of the 2D edges in this image is easy, but understanding what they mean is far more difficult. Note that there are three different types of edges: 1) those due to a change in reflectance (the boundary between q and r), 2) those due to a change in 3D object shape (the boundary between p and q), and 3) those due to the boundary between the object and background.

Adelson, E. H. (2000). Lightness Perception and Lightness Illusions. In M. Gazzaniga (Ed.), The New Cognitive Neurosciences, 2nd Ed. (pp. 339-351): MIT Press.

What do these edges mean?



The Problem of Disentangling

In order to interpret this image, one must understand how illumination, 3D shape, and reflectance interact, and how an object combines with its background in projecting to a 2D image (i.e., occlusion). If an edge is ascribed to be due to a reflectance change, it can not also be due to a shape change (an edge could be due to both, but then the contribution of each of these causes would need to be reduced so that when combined they still match what is in the image). Thus, the computation of reflectance depends on the computation of shape, and vice-versa. And both of these require prior knowledge of what shapes and reflectance changes are likely in order to arrive at a plausible interpretation consistent with the data.

Perception as Inference

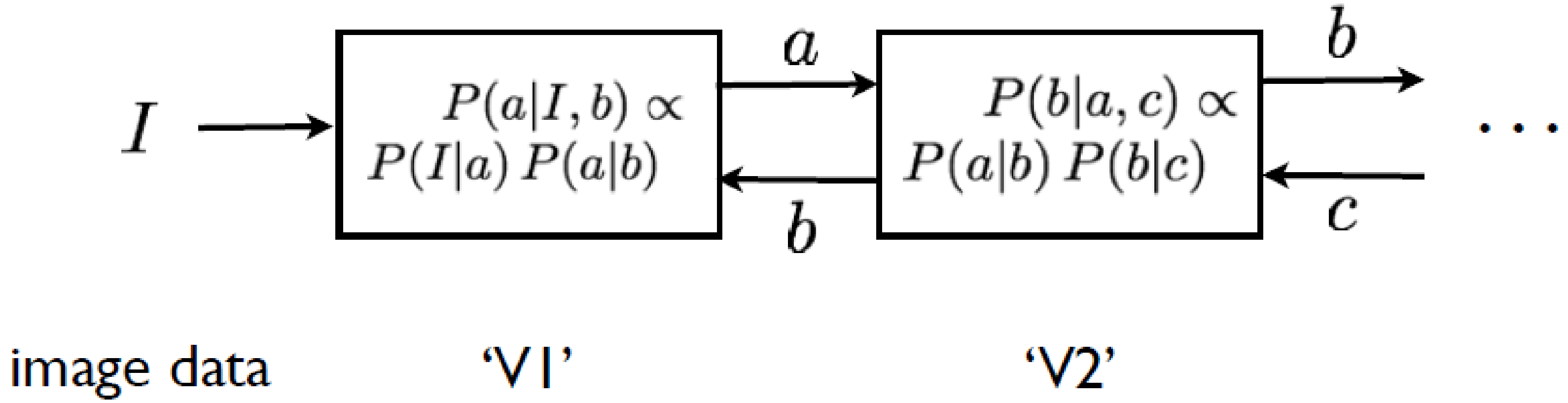
In perception, we are interested in estimating properties of the external environment from sensory data. For example, in vision we are given a set of photoreceptor activations or pixel intensities, I, and we wish to infer properties such as shape, s, and reflectance, r. We can formulate this problem as follows:

$$P(s,r|I) \propto P(I|s,r)P(s)P(r)$$

Here the likelihood term $P(I|s,r)$ expresses the rendering model - i.e., how images are generated as a function of shape and reflectance. This is a well-posed computation that is routinely solved by computer graphics algorithms. However the problem of going the other direction - from the image to compute shape and reflectance - is highly ill-posed because there are multiple ways to set these parameters that would result in the same image. This degeneracy is resolved by the priors over shape and reflectance, $\mathrm{P}(s)$ and $\mathrm{P}(r)$, which favor certain settings of s and r over others.

These priors can be obtained by measuring statistics of shape and reflectance on a large database of objects.

Hierarchical Bayesian Inference in Visual Cortex


I
$P(a|I,b) \propto P(I|a)\,P(a|b)$
a
b
$P(b|a,c) \propto P(a|b)\,P(b|c)$
b
c
$\cdots$
image data
'V1'
'V2'

The variables represented at each level are inferred from a combination of bottom-up and top-down inputs. Bottom-up inputs enter into the likelihood, while top-down inputs enter into the prior. The two are combined to form the unnormalized posterior, which guides the inference of variables at each level.

Lee, T. S., & Mumford, D. (2003). Hierarchical Bayesian inference in the visual cortex. Journal of the Optical Society of America, A, 20(7), 1434-1448.

The Eight Schools Problem

It has become a classic problem that illustrates the usefulness of hierarchical modeling for sharing information between exchangeable groups.

The Eight Schools Problem

A study was performed for the Educational Testing Service to analyze the effects of special coaching programs for SAT-V (Scholastic Aptitude Test-Verbal) in each of eight high schools. The outcome variable in each study was the score on a special administration of the SAT-V, a standardized multiple choice test administered by the Educational Testing Service and used to help colleges make admissions decisions; the scores can vary between 200 and 800, with mean about 500 and standard deviation about 100. The SAT examinations are designed to be resistant to short-term efforts directed specifically toward improving performance on the test; instead they are designed to reflect knowledge acquired and abilities developed over many years of education. Nevertheless, each of the eight schools in this study considered its short-term coaching program to be very successful at increasing SAT scores. Also, there was no prior reason to believe that any of the eight programs was more effective than any other or that some were more similar in effect to each other than to any other.

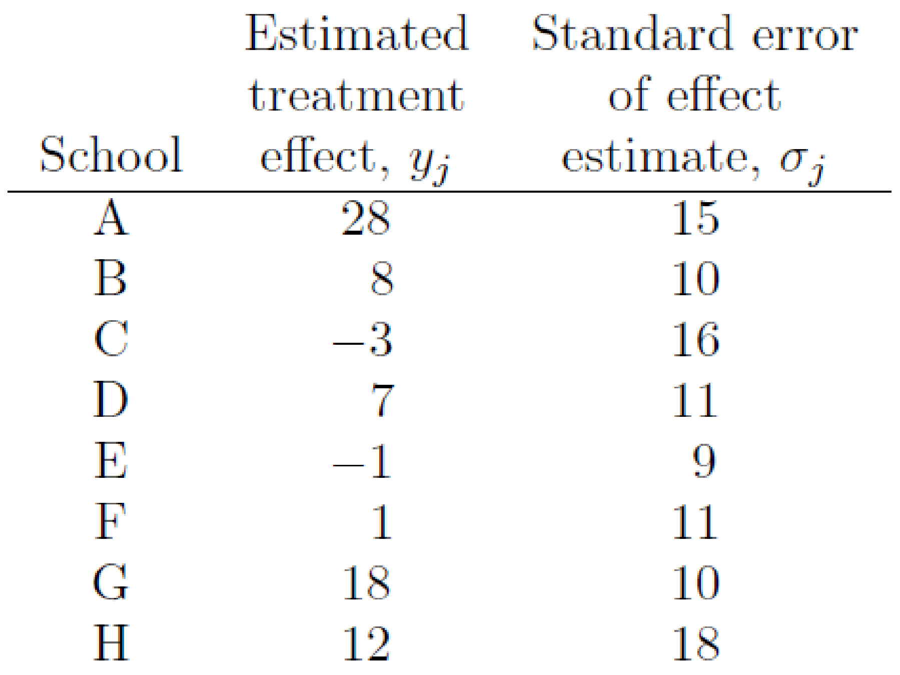
The Eight Schools Problem

“

All students in the experiments had already taken the PSAT (Preliminary SAT), and allowance was made for differences in the PSAT-M (Mathematics) and PSAT-V test scores between coached and uncoached students. In particular, in each school the estimated coaching effect and its standard error were obtained by an analysis of covariance adjustment (that is, a linear regression was performed of SAT-V on treatment group, using PSAT-M and PSAT-V as control variables) appropriate for a completely randomized experiment. A separate regression was estimated for each school. The sample sizes in all of the eight experiments were relatively large, over thirty students in each school. Incidentally, an increase of eight points on the SAT-V corresponds to about one more test item correct.

“

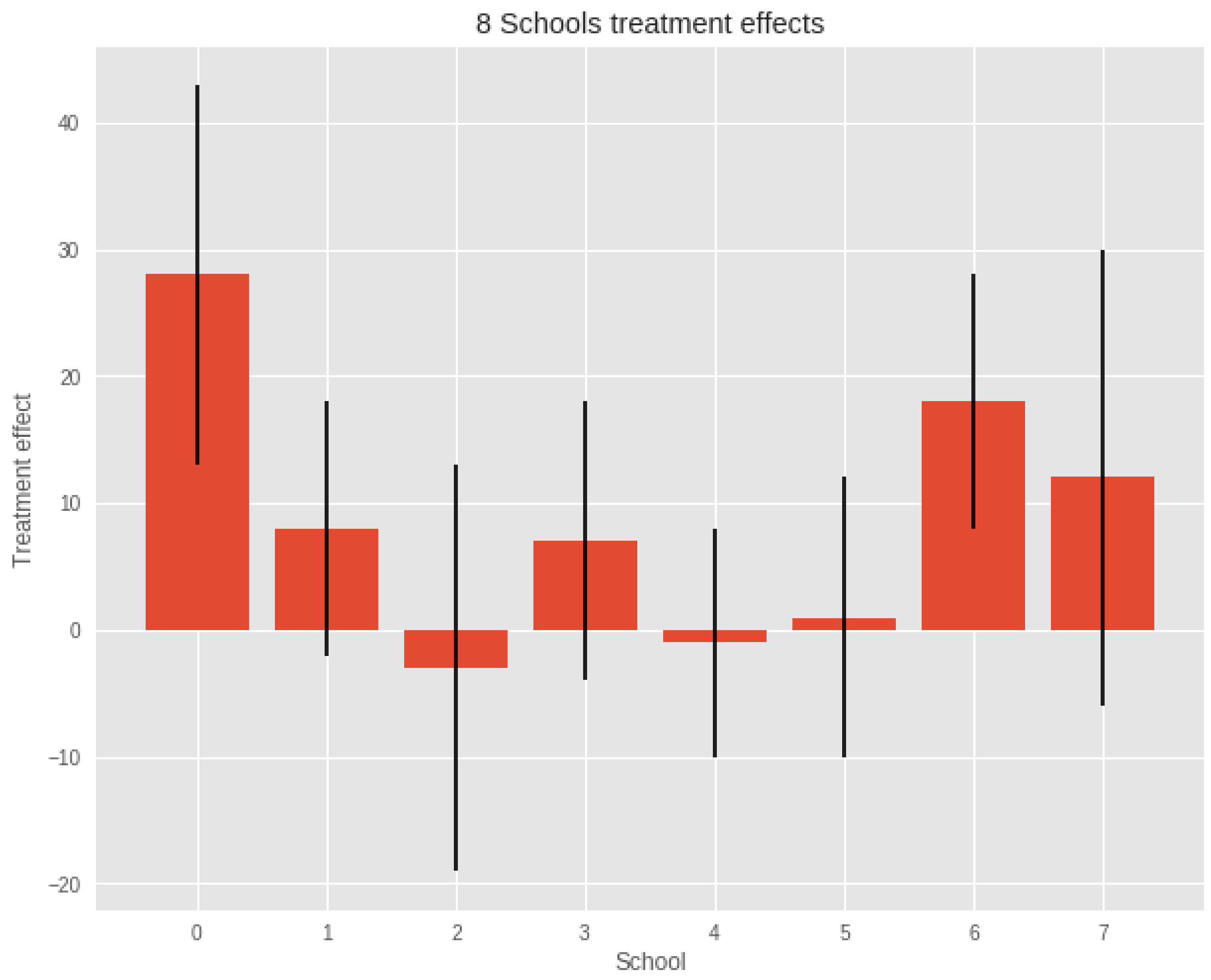
The Eight Schools Treatment Effects



School	Estimated treatment effect, y_j	Standard error of effect estimate, σ_j
A	28	15
B	8	10
C	-3	16
D	7	11
E	-1	9
F	1	11
G	18	10
H	12	18

Table 5.2 *Observed effects of special preparation on SAT-V scores in eight randomized experiments. Estimates are based on separate analyses for the eight experiments.*

The Eight Schools Treatment Effects



Inferences Based on Nonhierarchical Models and their Problems

Two simpler nonhierarchical methods:

- Separate estimates
- A pooled estimate

Separate Estimates

A cursory examination may at first suggest that some coaching programs have moderate effects (in the range 18–28 points), most have small effects (0–12 points), and two have small negative effects; however, when we take note of the standard errors of these estimated effects, we see that it is difficult statistically to distinguish between any of the experiments. For example, treating each experiment separately and applying the simple normal analysis in each yields 95% posterior intervals that all overlap substantially.

A Pooled Estimate

The general overlap in the posterior intervals based on independent analyses suggests that all experiments might be estimating the same quantity. Under the hypothesis that all experiments have the same effect and produce independent estimates of this common effect, we could treat the data in Table 5.2 as eight normally distributed observations with known variances. With a noninformative prior distribution, the posterior mean for the common coaching effect is estimated to be 7.7 points with standard error equal to 4.1 which would lead to the 95% posterior interval [−0.5, 15.9], or approximately [8 ± 8].

Inferences Based on Nonhierarchical Models and their Problems

Consider the effect in school A. The effect in school A is estimated as 28.4 with a standard error of 14.9 under the separate analysis, versus a pooled estimate of 7.7 with a standard error of 4.1 under the common-effect model. The separate analyses of the eight schools imply the following posterior statement: ‘the probability is 1/2 that the true effect in A is more than 28.4,’ a doubtful statement, considering the results for the other seven schools. On the other hand, the pooled model implies the following statement: ‘the probability is 1/2 that the true effect in A is less than 7.7,’ which, despite the non-significant χ2 test, seems an inaccurate summary of our knowledge. The pooled model also implies the statement: ‘the probability is 1/2 that the true effect in A is less than the true effect in C,’ which also is difficult to justify given the data.

A Hierarchical Normal Model

We consider the problem of estimating a parameter θ using data from a small experiment and a prior distribution constructed from similar previous (or historical) experiments. Mathematically, we will consider the current and historical experiments to be a random sample from a common population.

Consider a set of experiments $j = 1 \dots J$, in which experiment j has data (vector) y_j and parameter (vector) θ_j, with likelihood $p(y_j|\theta_j)$. (Throughout this chapter we use the word 'experiment' for convenience, but the methods can apply equally well to nonexperimental data.) Some of the parameters in different experiments may overlap; for example, each data vector y_j may be a sample of observations from a normal distribution with mean μ_j and common variance σ^2, in whichcase $\theta_j = (\mu_j, \sigma^2)$. In order to create a joint probability model for all the parameters θ, we use the crucial idea of *exchangeability*.

Exchangeability

If no information — other than the data y — Is available to distinguish any of the θ_j's from any of the others, and no ordering or grouping of the parameters can be made, one must assume symmetry among the parameters in their prior distribution. This symmetry is represented probabilistically by exchangeability; the parameters $(\theta_1, \dots, \theta_J)$ are exchangeable in their joint distribution if $p(\theta_1, \dots, \theta_J)$ is invariant to permutations of the indexes $(1, \dots, J)$.

The simplest form of an exchangeable distribution has each of the parameters θ_j as an independent sample from a prior (or population) distribution governed by some unknown parameter vector ϕ; thus,

$$p(\theta|\phi) = \prod_{j=1}^{J} p(\theta_j|\phi)$$

In general, ϕ is unknown, so our distribution for θ must average over our uncertainty in ϕ:

$$p(\theta) = \int \left(\prod_{j=1}^{J} p(\theta_j|\phi) \right) p(\phi) d\phi$$

The Bayesian Treatment of the Hierarchical Model

The key ‘hierarchical’ part of these models is that ϕ is not known and thus has its own prior distribution. we must assign a prior distribution to ϕ. If little is known about ϕ, we can assign a diffuse prior distribution, but we must be careful when using an improper prior density to check that the resulting posterior distribution is proper, and we should assess whether our conclusions are sensitive to this simplifying assumption. In most real problems, one should have enough substantive knowledge about the parameters in ϕ at least to constrain the hyperparameters into a finite region, if not to assign a substantive hyperprior distribution. As in nonhierarchical models, it is often practical to start with a simple, relatively noninformative, prior distribution on ϕ and seek to add more prior information if there remains too much variation in the posterior distribution.

The Eight Schools Hierarchical Model

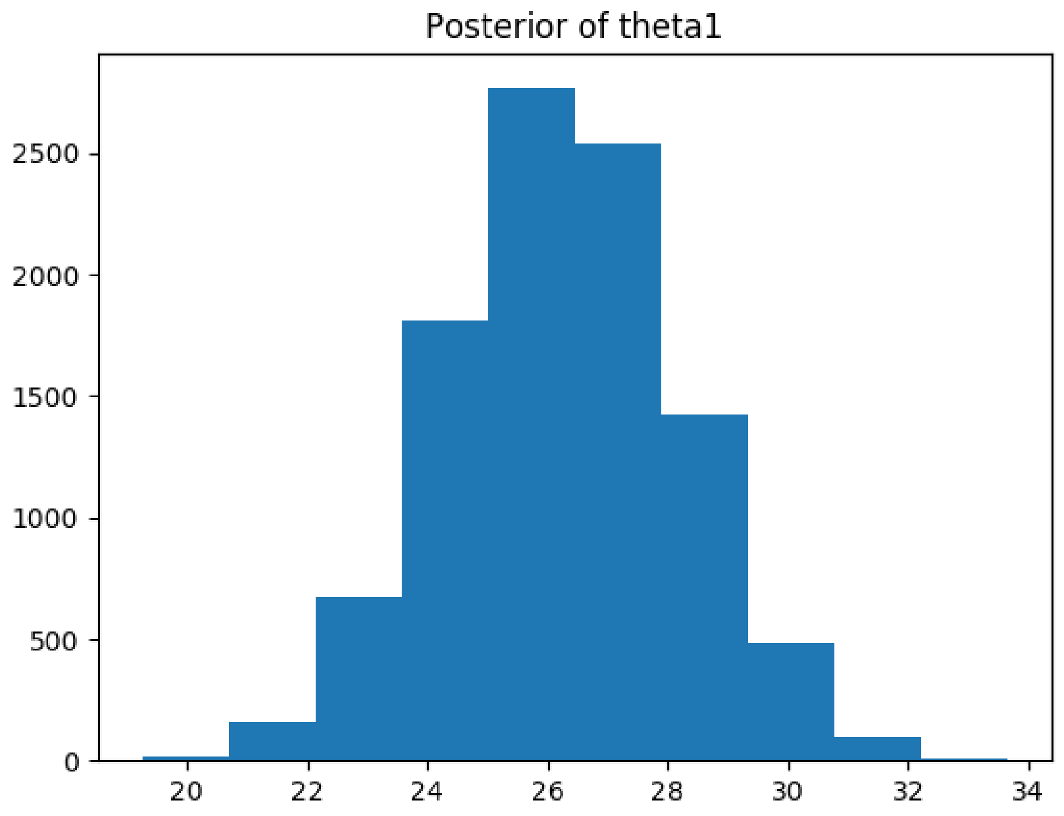
$$\begin{aligned}
\mu &\sim N(0, 10) \\
\log\tau &\sim N(5, 1) \\
\theta_j &\sim N(\mu, \tau), \qquad j = 1, \dots, 8 \\
y_j &\sim N(\theta_j, \sigma_j), \qquad j = 1, \dots, 8
\end{aligned}$$

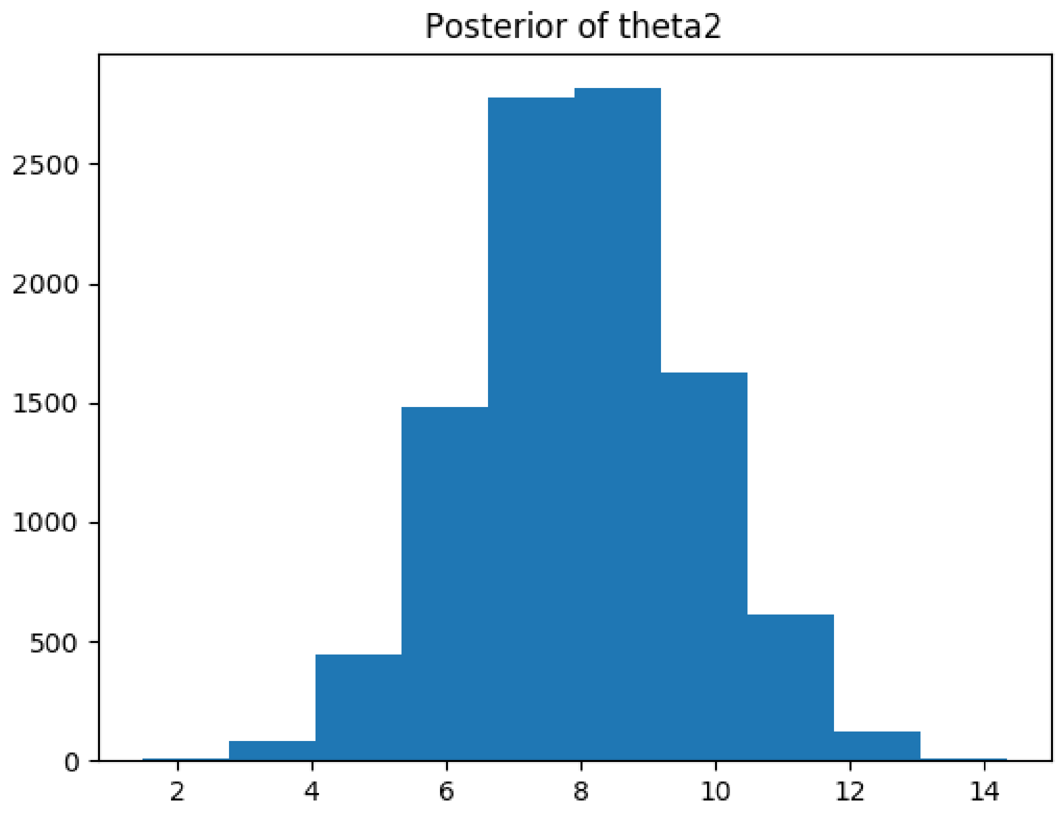
where μ represents the prior average treatment effect and τ controls how much variance there is between schools. The y_j and σ_j are observed. As $\tau \to \infty$, the model approaches the no-pooling model, i.e., each of the school treatment effect estimates are allowed to be more independent. As $\tau \to 0$, the model approaches the complete-pooling model, i.e., all of the school treatment effects are closer to the group average μ. To restrict the standard deviation to be positive, we draw τ from a lognormal distribution (which is equivalent to drawing $\log\tau$ from a normal distribution).

8 Schools in PyMC2

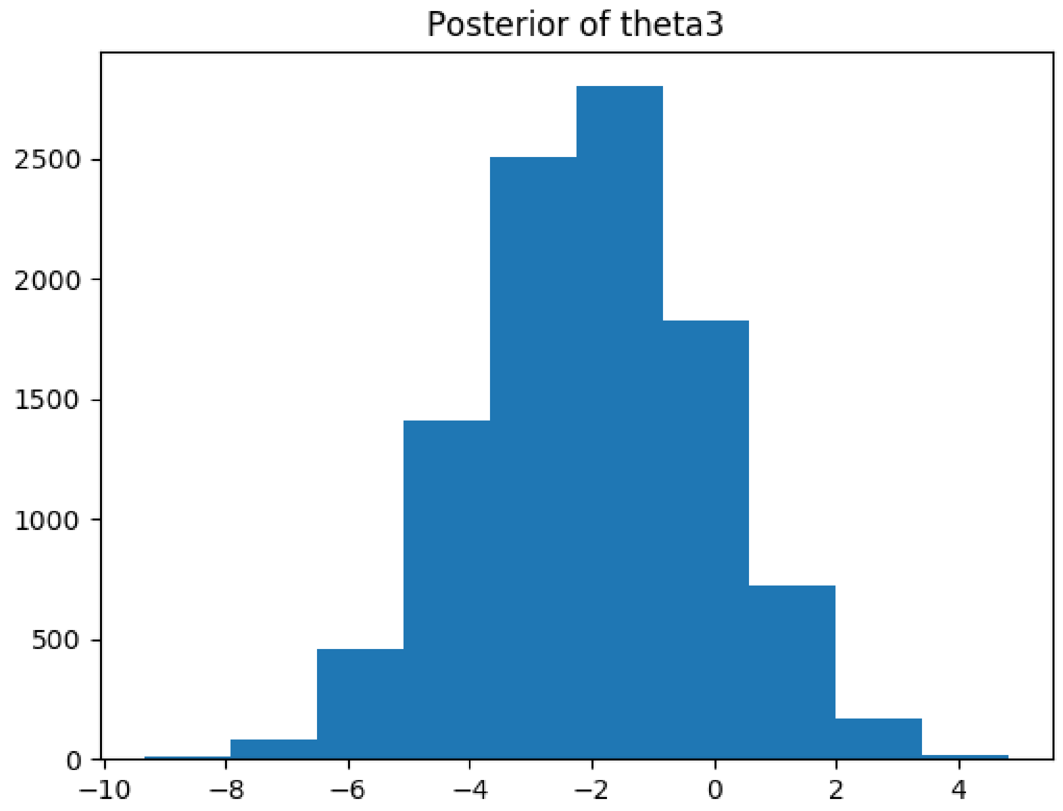
```
num_schools = 8  # number of schools
treatment_effects = np.array([28, 8, -3, 7, -1, 1, 18, 12], dtype=np.float32)  # treatment effects
treatment_stddevs = np.array([15, 10, 16, 11, 9, 11, 10, 18], dtype=np.float32)  # treatment SE

treatment_prec = 1. / np.sqrt(treatment_stddevs)

mu = pm.Normal('mu', 0., 0.01)
log_tau = pm.Normal('log_tau', 5., 1.)

@pm.deterministic
def tau(log_tau = log_tau):
    return 1. / np.sqrt(np.exp(log_tau))

theta1 = pm.Normal('theta1', mu, tau)
theta2 = pm.Normal('theta2', mu, tau)
theta3 = pm.Normal('theta3', mu, tau)
theta4 = pm.Normal('theta4', mu, tau)
theta5 = pm.Normal('theta5', mu, tau)
theta6 = pm.Normal('theta6', mu, tau)
theta7 = pm.Normal('theta7', mu, tau)
theta8 = pm.Normal('theta8', mu, tau)

y = pm.Normal('y', [theta1, theta2, theta3, theta4, theta5, theta6, theta7, theta8], treatment_prec,
              value = treatment_effects, observed = True)

model = pm.Model([mu, log_tau, tau, theta1, theta2, theta3, theta4, theta5, theta6, theta7, theta8, y])
map_ = pm.MAP(model)
map_.fit()
mcmc = pm.MCMC(model)
mcmc.sample(iter=200000, burn=100000, thin=10)
```

The Results

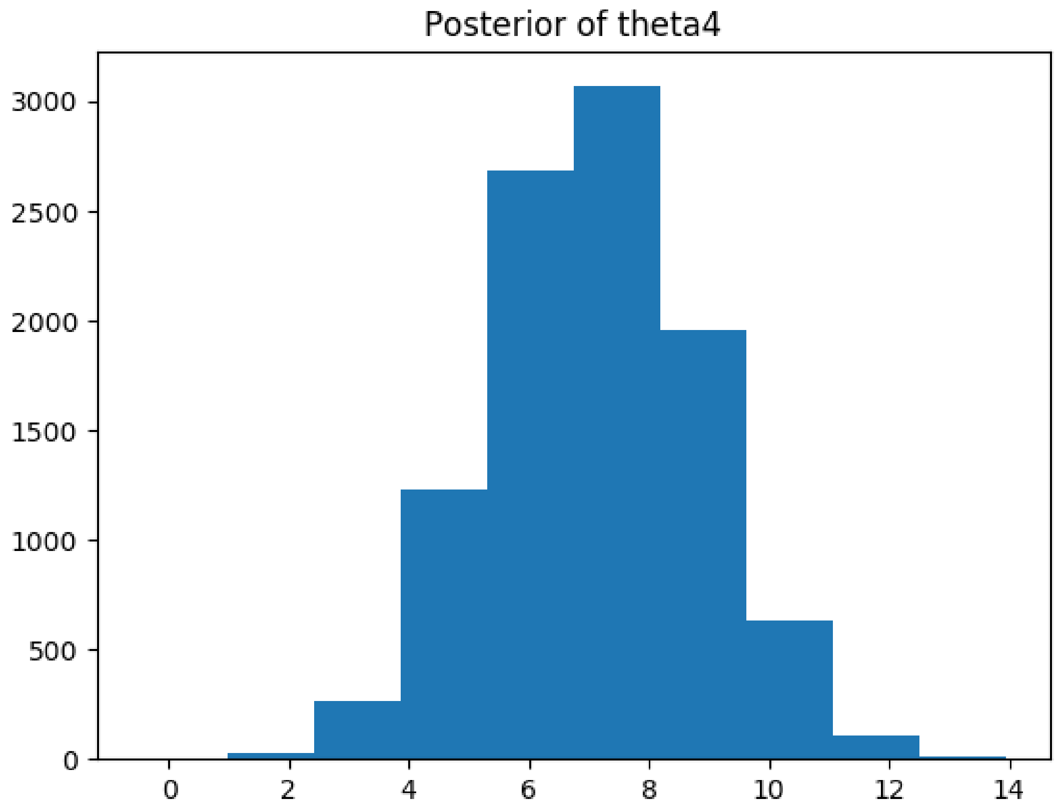

Posterior of theta1
0
500
1000
1500
2000
2500
20
22
24
26
28
30
32
34


Posterior of theta2
0
500
1000
1500
2000
2500
2
4
6
8
10
12
14

The Results

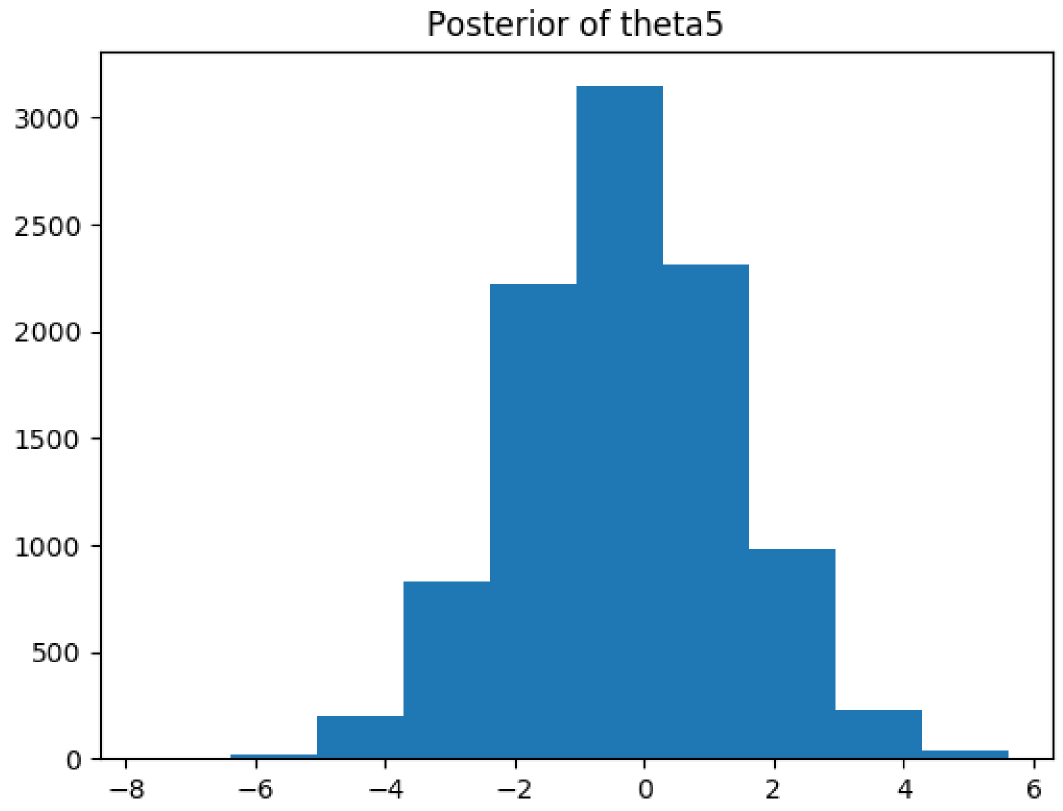


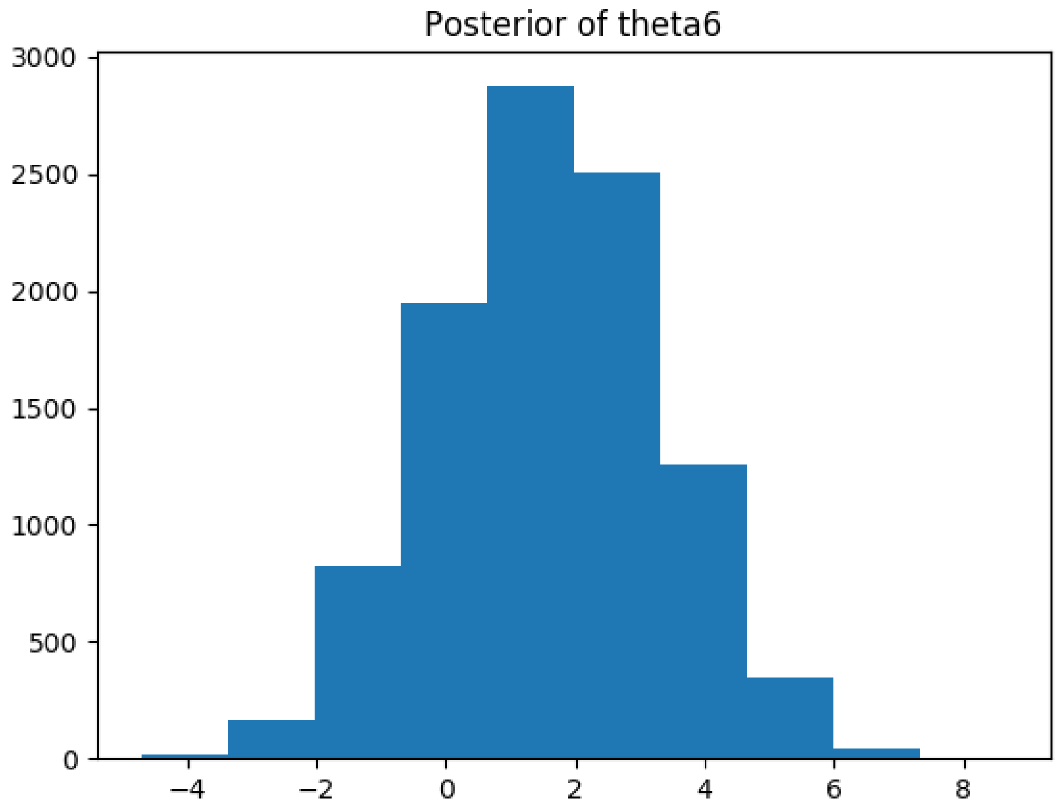
Posterior of theta3
2500
2000
1500
1000
500
0
−10
−8
−6
−4
−2
0
2
4



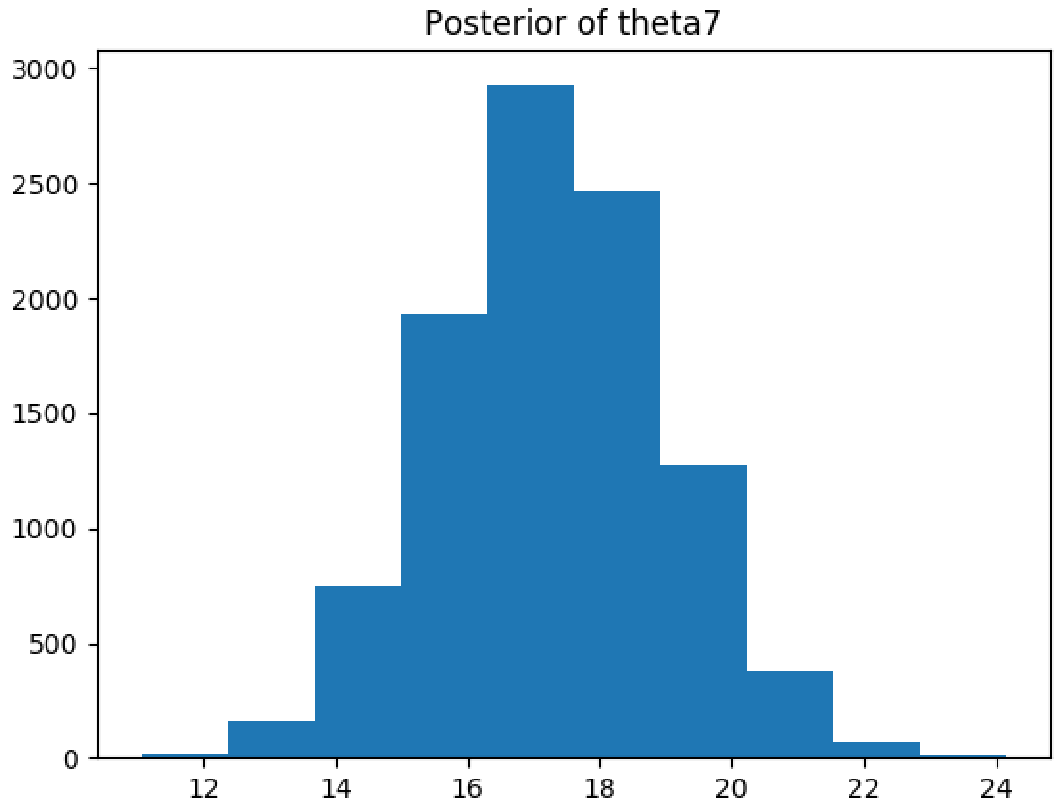
Posterior of theta4
3000
2500
2000
1500
1000
500
0
0
2
4
6
8
10
12
14

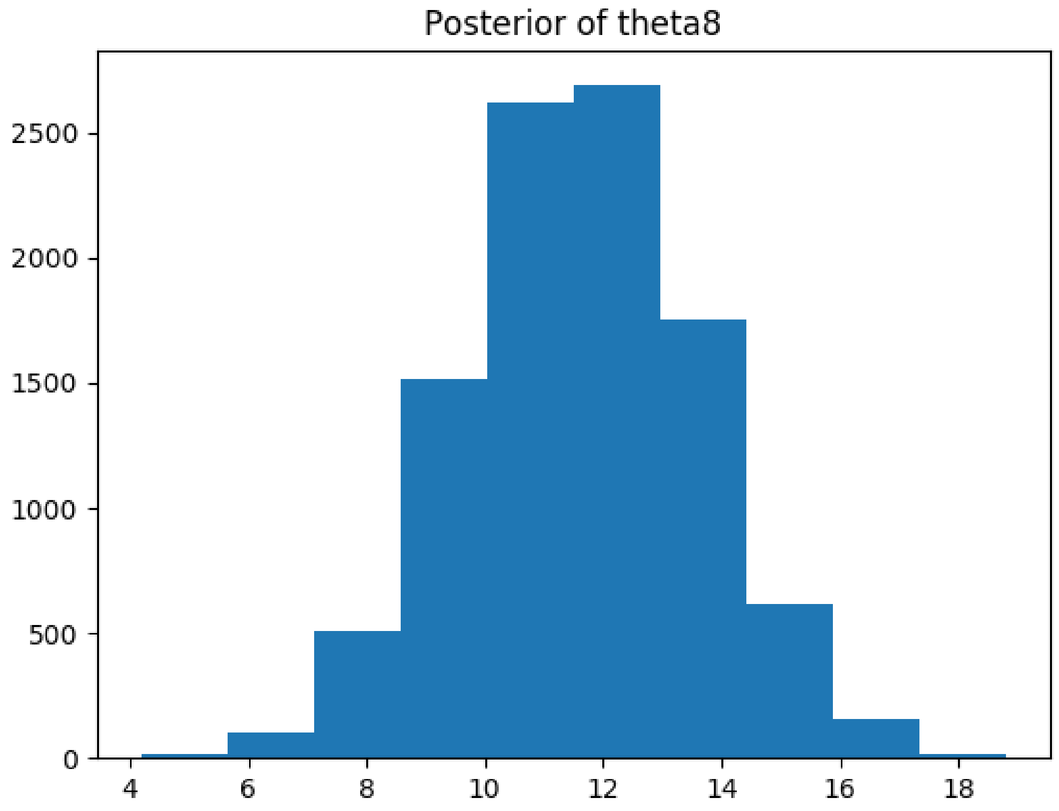
The Results


Posterior of theta5
3000
2500
2000
1500
1000
500
0
−8
−6
−4
−2
0
2
4
6

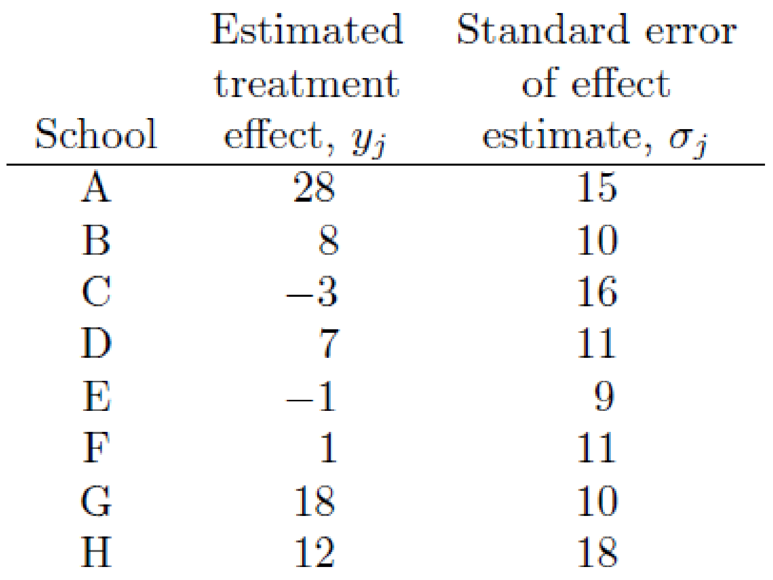

Posterior of theta6
3000
2500
2000
1500
1000
500
0
−4
−2
0
2
4
6
8

The Results


Posterior of theta7
3000
2500
2000
1500
1000
500
0
12
14
16
18
20
22
24


Posterior of theta8
2500
2000
1500
1000
500
0
4
6
8
10
12
14
16
18

The Results

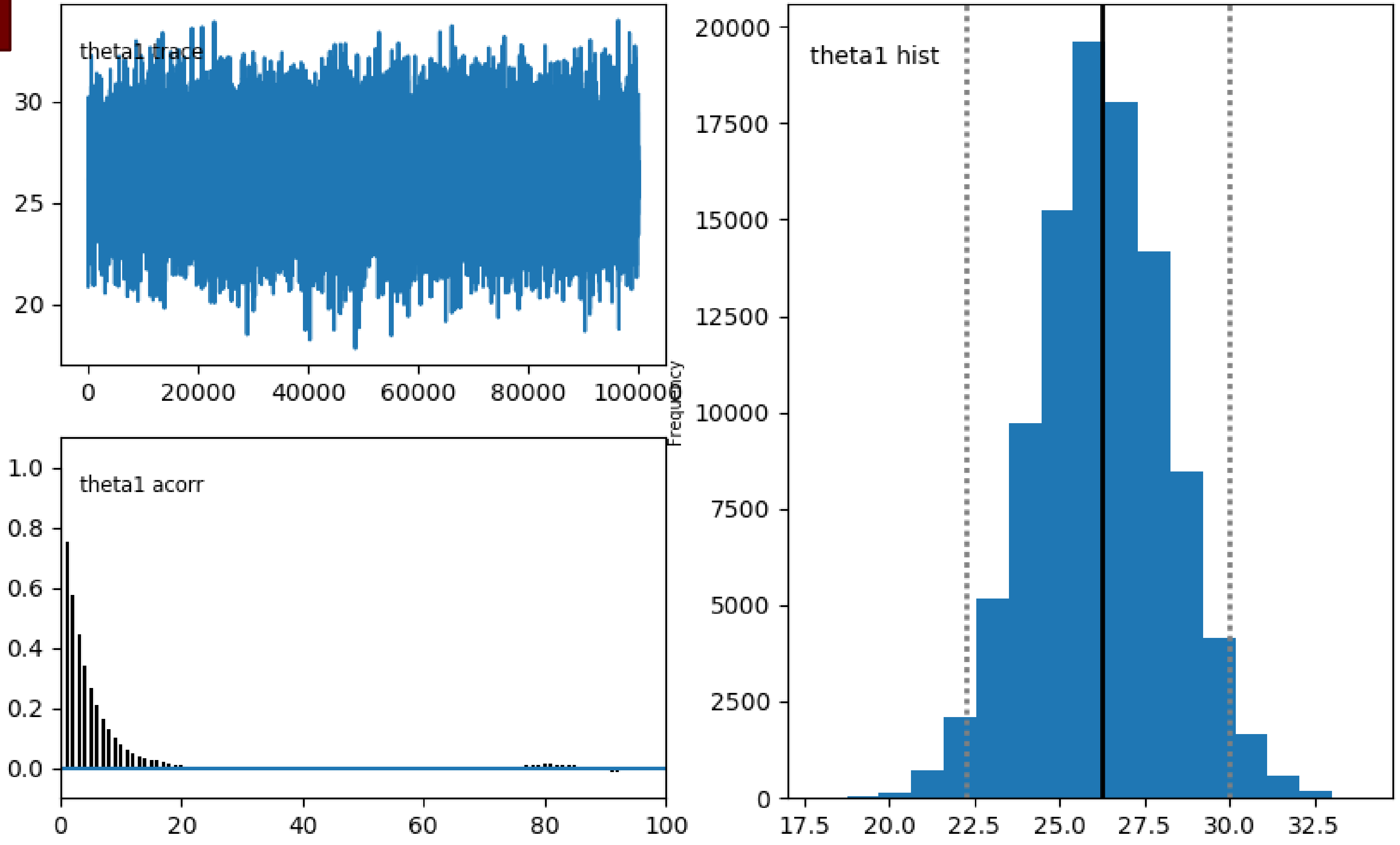


School	Estimated treatment effect, y_j	Standard error of effect estimate, σ_j
A	28	15
B	8	10
C	-3	16
D	7	11
E	-1	9
F	1	11
G	18	10
H	12	18

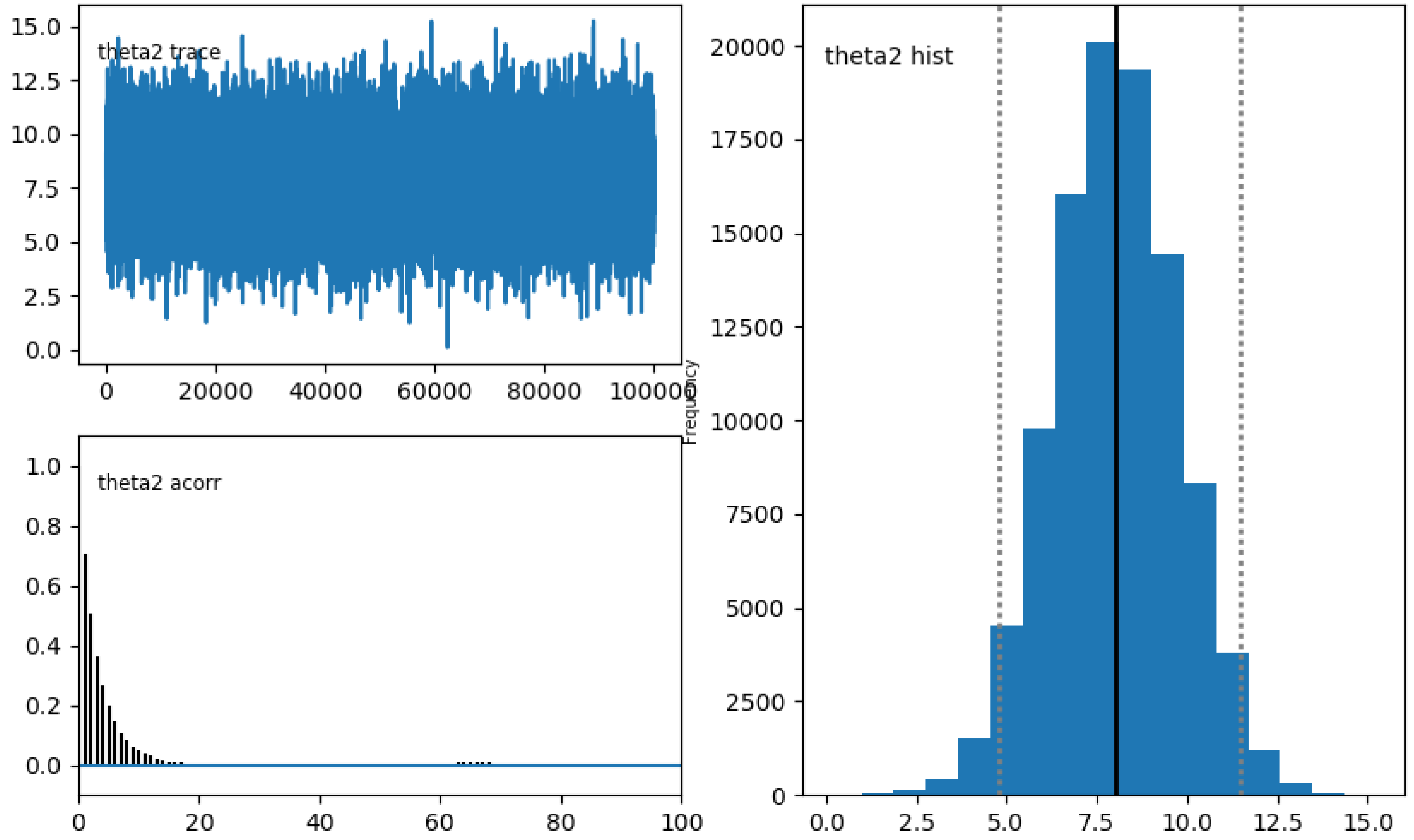
```
print(mcmc.trace('theta1')[:].mean())
print(mcmc.trace('theta2')[:].mean())
print(mcmc.trace('theta3')[:].mean())
print(mcmc.trace('theta4')[:].mean())
print(mcmc.trace('theta5')[:].mean())
print(mcmc.trace('theta6')[:].mean())
print(mcmc.trace('theta7')[:].mean())
print(mcmc.trace('theta8')[:].mean())
```

```
26.2473940786
8.0174740141
-1.99846748906
7.10777255222
-0.312653714224
1.57251181192
17.2781573738
11.6265107593
```

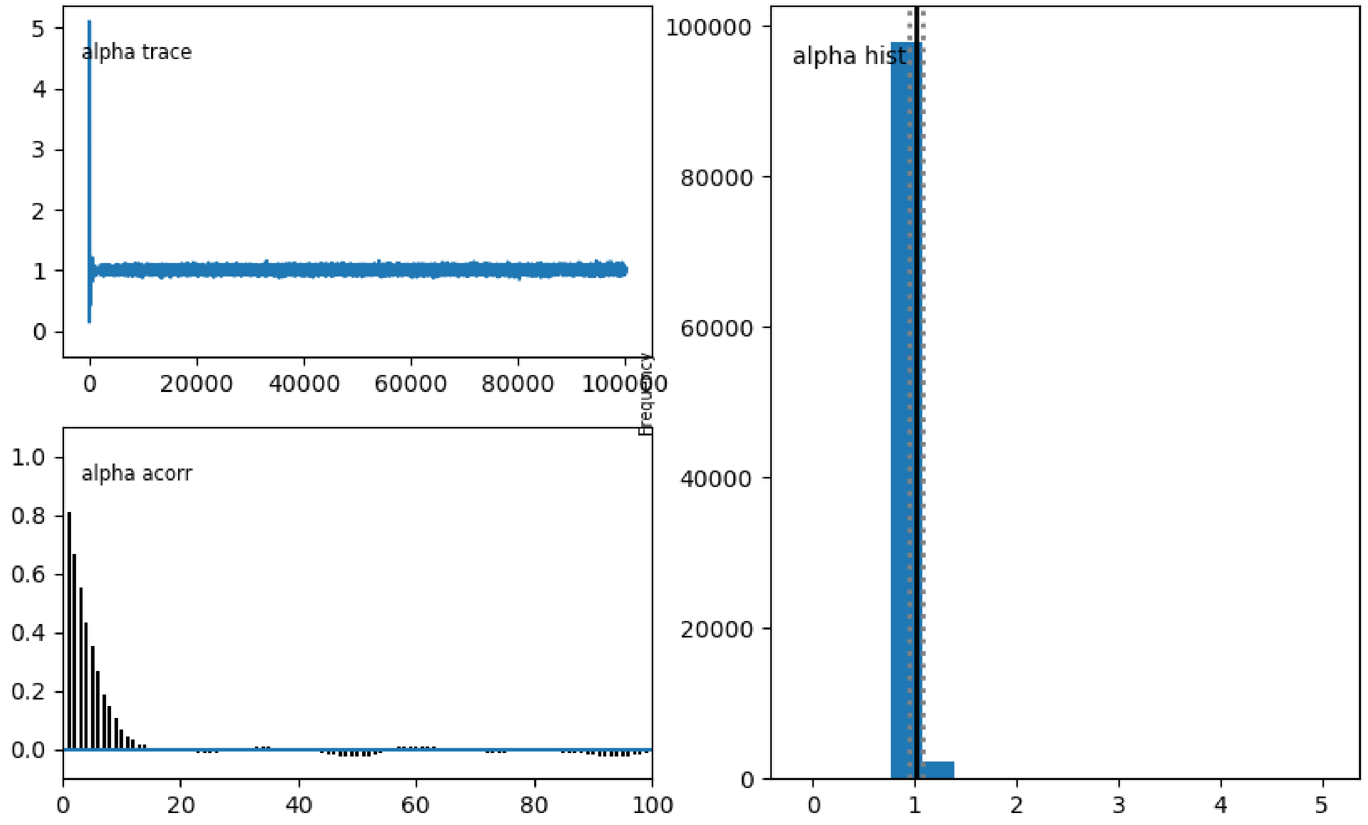
Convergence Diagnostics



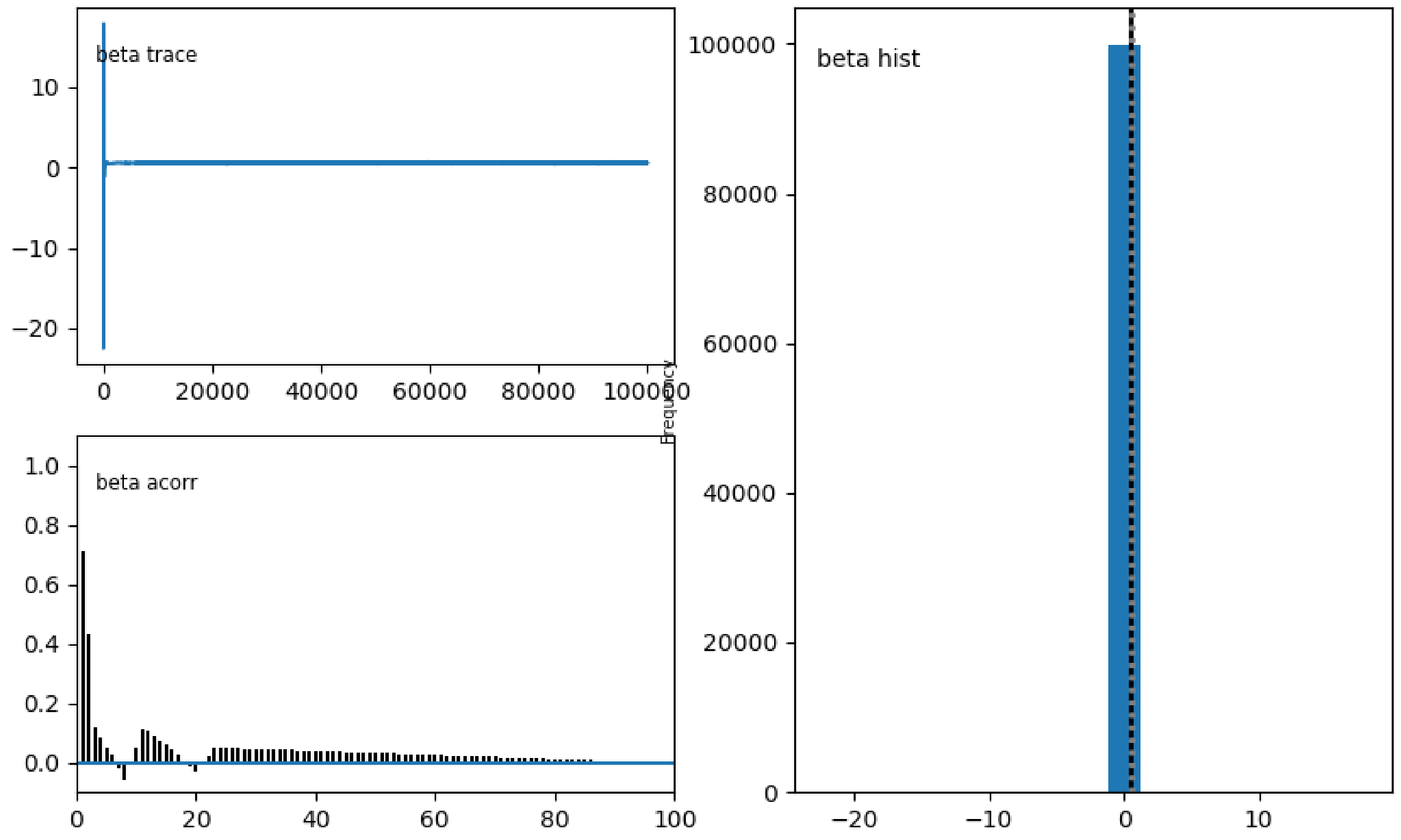
Convergence Diagnostics



Compare to the convergence in Bayesian simple regression



Compare to the convergence in Bayesian simple regression



Hamiltonian Monte Carlo

HMC differs from the Metropolis–Hastings algorithm by reducing the correlation between successive sampled states by using a Hamiltonian evolution between states and additionally by targeting states with a higher acceptance criteria than the observed probability distribution. This causes it to converge more quickly to the absolute probability distribution. It was devised by Simon Duane, A.D. Kennedy, Brian Pendleton and Duncan Roweth in 1987.

There is no free lunch, and Hamiltonian MC has its price: HMC uses not only energy, but also it's gradient. Hence possible applications are limited to the case when gradient exists and can be computed in reasonable time.

The major differences compared to Metropolis-Hastings are:

- distances between successive generated points are typically large, so we need less iterations to get representative sampling
- 'price' of a single iteration is higher, but HMC is still significantly more efficient
- Hamiltonian MC in most cases accepts new states
- still, HMC has problems with sampling from distributions with isolated local minimums

An alternative implementation of the same model

Inferring the hierarchical hyperparameters, μ and τ, together with the group-level parameters, $\theta_1, \dots, \theta_8$, allows the model to pool data across the groups and reduce their posterior variance. Unfortunately, this pooling also squeezes the posterior distribution into a particularly challenging geometry that obstructs geometric ergodicity and hence biases MCMC estimation.

In this case study we'll first examine the direct centered parameterization of the Eight Schools model and see how divergences identify this bias before there are any other indications of problems. We'll then use these divergences to study the source of the bias and motivate the necessary fix, a reimplementation of the model with a non-centered parameterization.

Betancourt, Michael, and Mark Girolami. 2015. "Hamiltonian Monte Carlo for Hierarchical Models." In Current Trends in Bayesian Methodology with Applications, edited by Umesh Singh Dipak K. Dey and A. Loganathan. Chapman & Hall/CRC Press.

A Non-Centered Eight Schools Implementation

$$\mu \sim N(0, 10)$$
$$\log\tau \sim N(5, 1)$$
$$\theta_j{}' \sim N(0\ 1), \qquad j = 1, \dots, 8$$
$$\theta_j = \mu + \tau\theta_j{}', \qquad j = 1, \dots, 8$$
$$y_j \sim N(\theta_j, \sigma_j), \qquad j = 1, \dots, 8$$

The Model in TFP

```
import tensorflow as tf
import tensorflow_probability as tfp
from tensorflow_probability import edward2 as ed
```

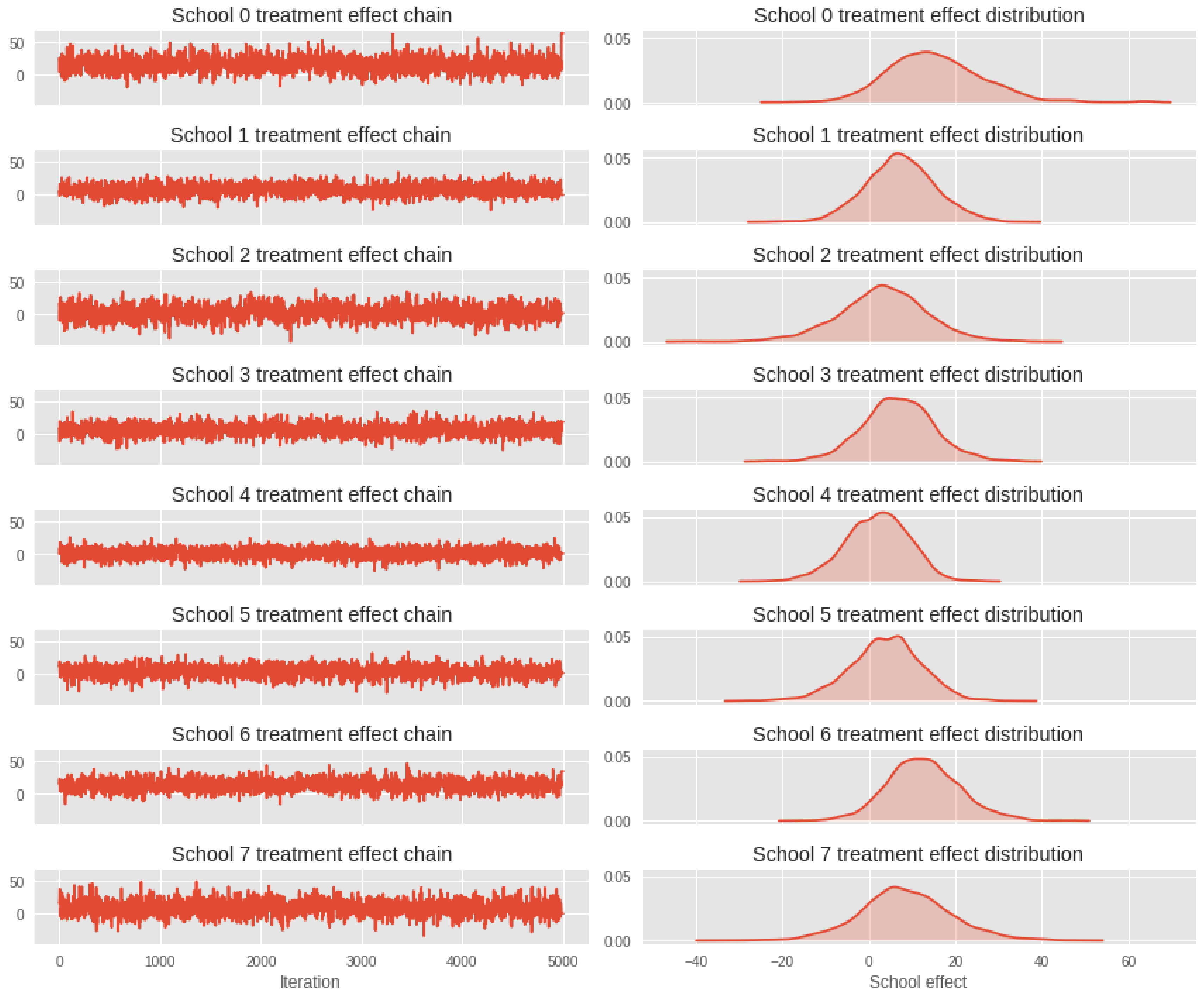
```
def schools_model(num_schools, treatment_stddevs):
  avg_effect = ed.Normal(loc=0., scale=10., name="avg_effect")  # `mu` above
  avg_stddev = ed.Normal(
      loc=5., scale=1., name="avg_stddev")  # `log(tau)` above
  school_effects_standard = ed.Normal(
      loc=tf.zeros(num_schools),
      scale=tf.ones(num_schools),
      name="school_effects_standard")  # `theta_prime` above
  school_effects = avg_effect + tf.exp(
      avg_stddev) * school_effects_standard  # `theta` above
  treatment_effects = ed.Normal(
      loc=school_effects, scale=treatment_stddevs,
      name="treatment_effects")  # `y` above
  return treatment_effects

log_joint = ed.make_log_joint_fn(schools_model)

def target_log_prob_fn(avg_effect, avg_stddev, school_effects_standard):
  return log_joint(
      num_schools=num_schools,
      treatment_stddevs=treatment_stddevs,
      avg_effect=avg_effect,
      avg_stddev=avg_stddev,
      school_effects_standard=school_effects_standard,
      treatment_effects=treatment_effects)
```

HMC in TFP

```
num_results = 5000
num_burnin_steps = 3000
```

```
states, kernel_results = tfp.mcmc.sample_chain(
    num_results=num_results,
    num_burnin_steps=num_burnin_steps,
    current_state=[
        tf.zeros([], name='init_avg_effect'),
        tf.zeros([], name='init_avg_stddev'),
        tf.ones([num_schools], name='init_school_effects_standard'),
    ],
    kernel=tfp.mcmc.HamiltonianMonteCarlo(
        target_log_prob_fn=target_log_prob_fn,
        step_size=0.4,
        num_leapfrog_steps=3))

avg_effect, avg_stddev, school_effects_standard = states

with tf.Session() as sess:
  [
      avg_effect_,
      avg_stddev_,
      school_effects_standard_,
      is_accepted_,
  ] = sess.run([
      avg_effect,
      avg_stddev,
      school_effects_standard,
      kernel_results.is_accepted,
  ])
```

The Results



```
num_accepted = np.sum(is_accepted_)
print('Acceptance rate:’)
print(num_accepted / num_results)
```

```
Acceptance rate:
0.6186
```

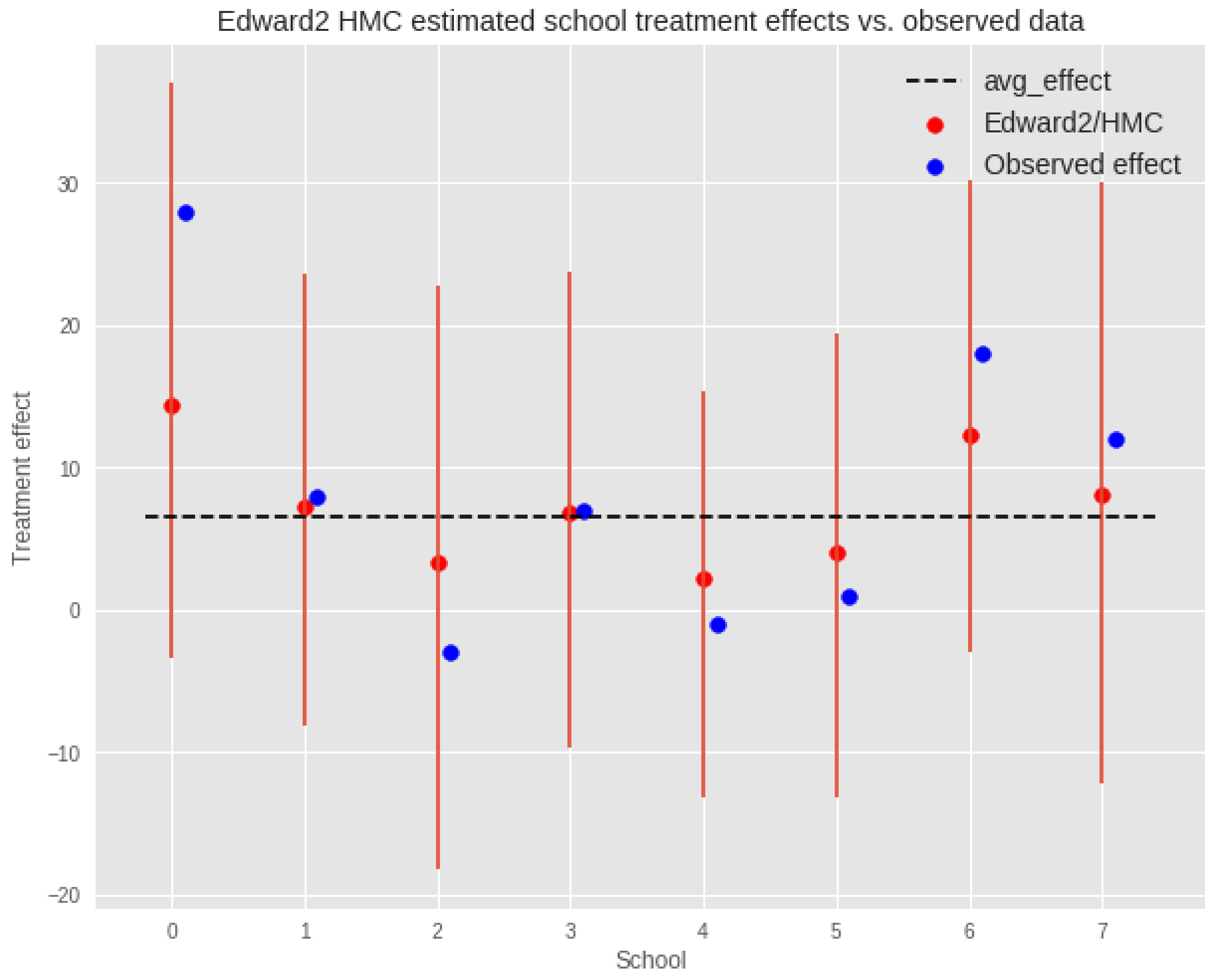
The Results

```
school_effects_samples = (
    avg_effect_[:, np.newaxis] +
    np.exp(avg_stddev_)[:, np.newaxis] * school_effects_standard_)

print("E[avg_effect] = {}".format(avg_effect_.mean()))
print("E[avg_stddev] = {}".format(avg_stddev_.mean()))
print("E[school_effects_standard] =")
print(school_effects_standard_[:, ].mean(0))
print("E[school_effects] =")
print(school_effects_samples[:, ].mean(0))
print("Inferred posterior mean: {0:.2f}".format(
    np.mean(school_effects_samples[:,])))
print("Inferred posterior mean se: {0:.2f}".format(
    np.std(school_effects_samples[:,])))
```

The Results

```
E[avg_effect] = 6.494342803955078

E[avg_stddev] = 2.436934471130371

E[school_effects_standard] =

[ 0.66298497  0.06529777 -0.26186424  0.01561266 -0.385506   -0.2165112 0.47715753  0.1520656 ]

E[school_effects] =

[15.3593855  7.3573384  3.1578748  6.775903   1.9274986  3.822677 12.657511   8.500975 ]

Inferred posterior mean: 7.44

Inferred posterior mean se: 10.01
```

We can observe the shrinkage toward the group `avg_effect`



Model & Inference Criticism

- Assessing how model predictions match the true data
- Assessing how data generated from the model matches the true data

8 Schools Model Criticism

To get the posterior predictive distribution, i.e., a model of new data y^* given the observed data y:

$$p(y^*|y) \propto \int_{\theta} p(y^*|\theta)p(\theta|y)d\theta$$

we "intercept" the values of the random variables in the model to set them to the mean of the posterior distribution and sample from that model to generate new data y^*.

Manipulating Model Computation: Interceptors

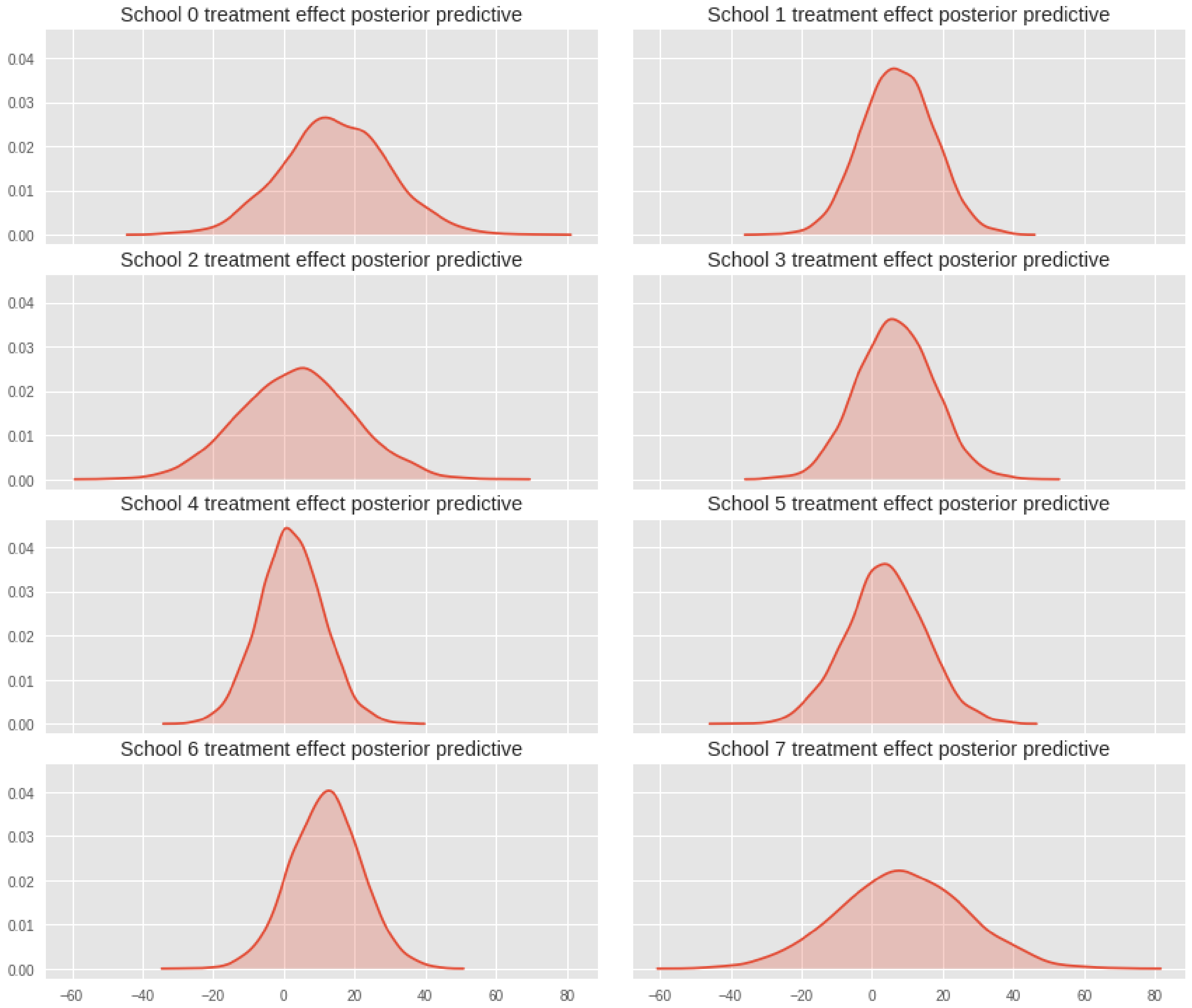
An interceptor is a function that acts on another function `f` and its arguments `*args, **kwargs`. It performs various computations before returning an output (typically `f(*args, **kwargs)`: the result of applying the function itself). The `ed.interception` context manager pushes interceptors onto a stack, and any interceptable function is intercepted by the stack. All random variable constructors are interceptable.

In particular, we make predictions with a model learned posterior means rather than with its priors.

Posterior Predictive

```
def interceptor(rv_constructor, *rv_args, **rv_kwargs):
  """Replaces prior on effects with empirical posterior mean from MCMC."""
  name = rv_kwargs.pop("name")
  if name == "avg_effect":
    rv_kwargs["value"] = np.mean(avg_effect_, 0)
  elif name == "avg_stddev":
    rv_kwargs["value"] = np.mean(avg_stddev_, 0)
  elif name == "school_effects_standard":
    rv_kwargs["value"] = np.mean(school_effects_standard_, 0)
  return rv_constructor(*rv_args, **rv_kwargs)

with ed.interception(interceptor):
  posterior = schools_model(
      num_schools=num_schools, treatment_stddevs=treatment_stddevs)

with tf.Session() as sess:
  posterior_predictive = sess.run(
      posterior.distribution.sample(sample_shape=(5000)))
```

The Results


School 0 treatment effect posterior predictive
School 1 treatment effect posterior predictive
School 2 treatment effect posterior predictive
School 3 treatment effect posterior predictive
School 4 treatment effect posterior predictive
School 5 treatment effect posterior predictive
School 6 treatment effect posterior predictive
School 7 treatment effect posterior predictive
0.00
0.01
0.02
0.03
0.04
−60
−40
−20
0
20
40
60
80

PyMC4

A Keras for TFP?

8 Schools in PyMC4 (January 2019)

```
import numpy as np

import pymc4 as pm4

import tensorflow as tf # For Random Variable operation

from tensorflow_probability import edward2 as ed # For defining random variables


J = 8 # No. of schools

y = np.array([28.,  8., -3.,  7., -1.,  1., 18., 12.])

sigma = np.array([15., 10., 16., 11.,  9., 11., 10., 18.])
```

8 Schools in PyMC4 (January 2019)

```
pymc4_non_centered_eight = pm4.Model(num_schools=J, y=y, sigma=sigma)

@pymc4_non_centered_eight.define
def process(cfg):
    mu = ed.Normal(loc=0., scale=5., name="mu")
    log_tau = ed.Normal(
        loc=5., scale=1., name="log_tau")
    theta_prime = ed.Normal(
        loc=tf.zeros(cfg.num_schools),
        scale=tf.ones(cfg.num_schools),
        name="theta_prime")
    theta = mu + tf.exp(
        log_tau) * theta_prime
    y = ed.Normal(
        loc=theta,
        scale=np.float32(cfg.sigma),
        name="y")
    return y
pymc4_non_centered_eight.observe(y=y)
```

8 Schools in PyMC4 (January 2019)

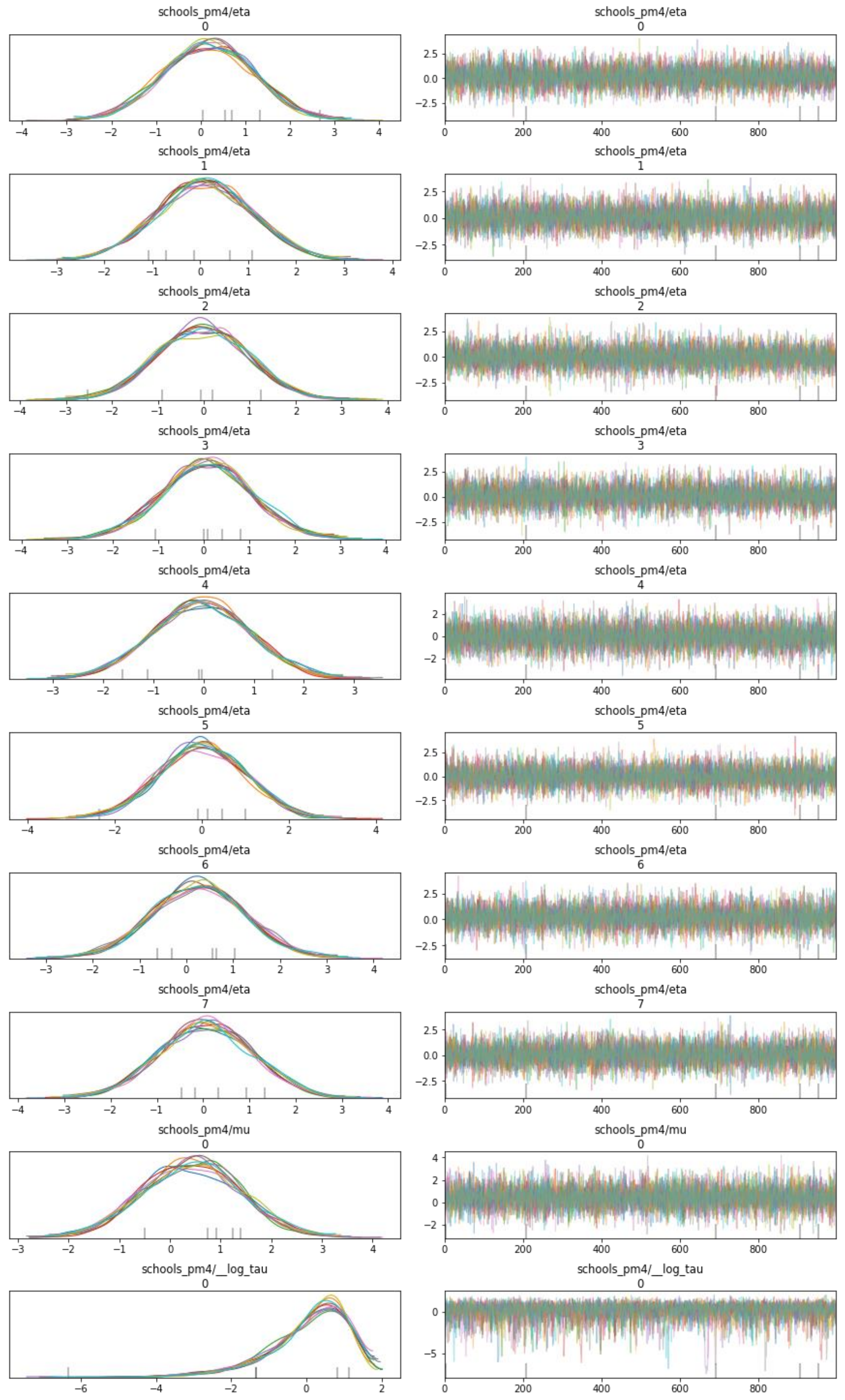
```
pymc4_trace = pm4.sample(pymc4_non_centered_eight)

pymc4_theta = pymc4_trace['mu'][:, np.newaxis] + np.exp(pymc4_trace['log_tau'])[:, np.newaxis] * pymc4_trace['theta_prime']

print(np.mean(pymc4_theta, axis = 0))


[14.93055    7.3458967  3.091199   6.523052   2.3198733  3.6610856  12.801121   7.5998383]
```

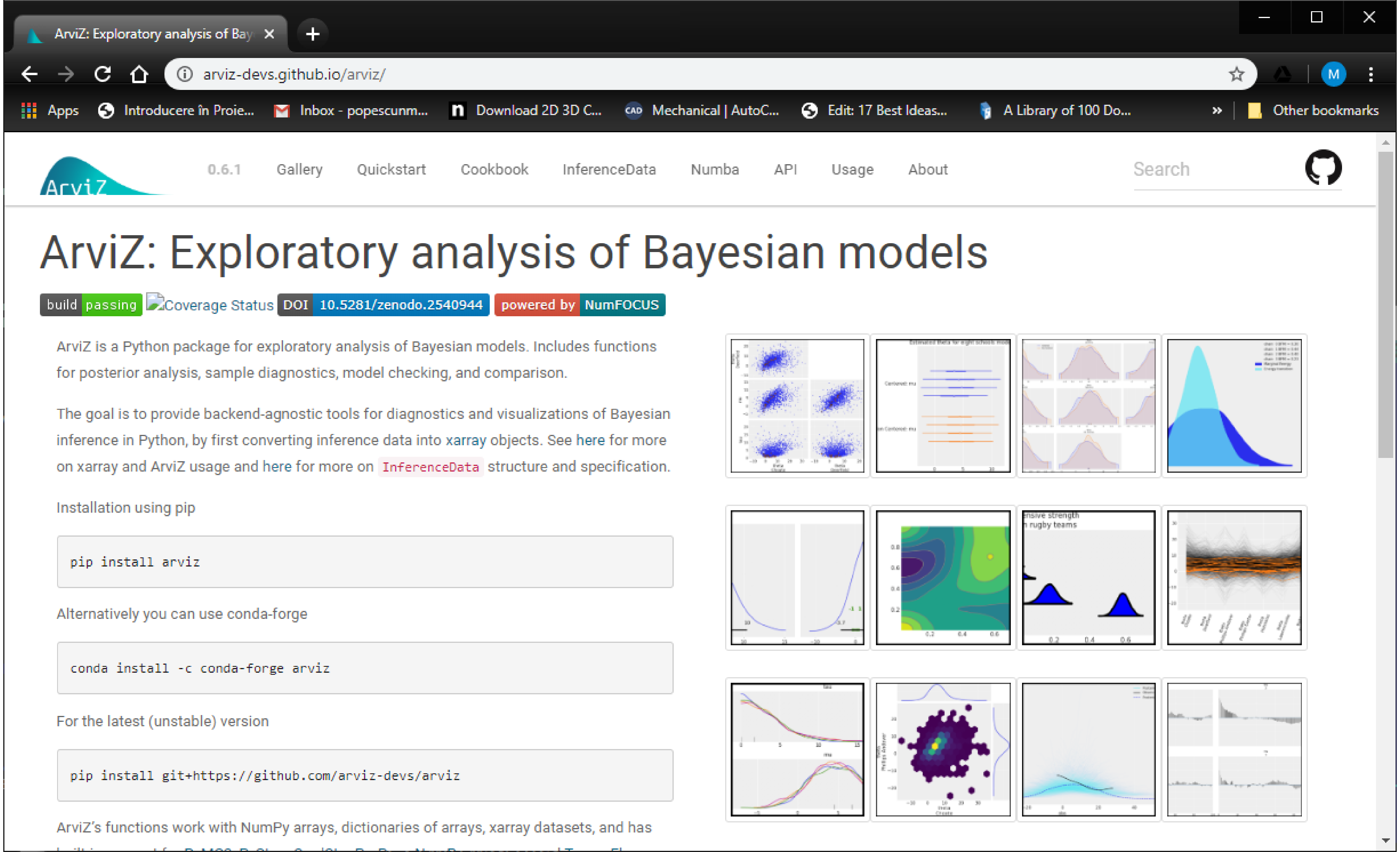
8 Schools in PyMC4 (January 2020)

```
import numpy as np

import pymc4 as pm

Import arviz as az


J = 8 # No. of schools

y = np.array([28.,  8., -3.,  7., -1.,  1., 18., 12.])

sigma = np.array([15., 10., 16., 11.,  9., 11., 10., 18.])
```

8 Schools in PyMC4 (January 2020)

```
@pm.model
def schools_pm4():
    eta = yield pm.Normal("eta", 0, 1, plate=J)
    mu = yield pm.Normal("mu", 0, 1, plate=1)
    tau = yield pm.HalfNormal('tau', 2., plate=1)

    theta = mu + tau * eta

    obs = yield pm.Normal('obs', theta, scale=sigma, observed=y)

    return obs

tf_trace, sample_stats  = pm.sample(schools_pm4())

az_trace = pm.trace_to_arviz(tf_trace, sample_stats)

az.plot_trace(az_trace);
```


schools_pm4/eta 0
schools_pm4/eta 0
schools_pm4/eta 1
schools_pm4/eta 1
schools_pm4/eta 2
schools_pm4/eta 2
schools_pm4/eta 3
schools_pm4/eta 3
schools_pm4/eta 4
schools_pm4/eta 4
schools_pm4/eta 5
schools_pm4/eta 5
schools_pm4/eta 6
schools_pm4/eta 6
schools_pm4/eta 7
schools_pm4/eta 7
schools_pm4/mu 0
schools_pm4/mu 0
schools_pm4/__log_tau 0
schools_pm4/__log_tau 0



ArviZ: Exploratory analysis of Bayesian models

build passing Coverage Status DOI 10.5281/zenodo.2540944 powered by NumFOCUS

ArviZ is a Python package for exploratory analysis of Bayesian models. Includes functions for posterior analysis, sample diagnostics, model checking, and comparison.

The goal is to provide backend-agnostic tools for diagnostics and visualizations of Bayesian inference in Python, by first converting inference data into xarray objects. See here for more on xarray and ArviZ usage and here for more on `InferenceData` structure and specification.

Installation using pip

```
pip install arviz
```

Alternatively you can use conda-forge

```
conda install -c conda-forge arviz
```

For the latest (unstable) version

```
pip install git+https://github.com/arviz-devs/arviz
```

ArviZ's functions work with NumPy arrays, dictionaries of arrays, xarray datasets, and has