Class Activity 6: Quiz [10 minutes]

1. What's the difference between these three data models. Can they be used for Bayesian inference? How?

$$\mathbf{y}_{n\times 1} \sim \mathcal{MVN}(\mathbf{X}_{n\times p}\boldsymbol{\beta}_{p\times 1}, \boldsymbol{\Sigma}_{n\times n} = \sigma^2 I_{n\times n})$$
$$p(\mathbf{y}|\mathbf{X},\boldsymbol{\beta},\Sigma) = (2\pi\sigma^2)^{-n/2}\exp\left(-\frac{1}{2\sigma^2}(\mathbf{y}-\mathbf{X}\boldsymbol{\beta})^\top(\mathbf{y}-\mathbf{X}\boldsymbol{\beta})\right)$$

$$\mathbf{y}_i \sim \mathcal{MVN}(\boldsymbol{\mu}_{p\times 1}, \boldsymbol{\Sigma}_{p\times p} = \mathbf{DRD})$$
$$p(\mathbf{y}|\boldsymbol{\mu},\Sigma) = (2\pi)^{-n/2}\det(\boldsymbol{\Sigma})^{-1/2}\exp\left(-\frac{1}{2}(\mathbf{y}-\boldsymbol{\mu})^\top\boldsymbol{\Sigma}^{-1}(\mathbf{y}-\boldsymbol{\mu})\right)$$

$$y_i \sim \text{Bernoulli}\Big(\Pr(y_i=1|\mathbf{x}_i) = \underbrace{\Phi}_{\text{CDF of }\mathcal{N}(0,1)}(\mathbf{x}_i^\top\boldsymbol{\beta}_{p\times 1})\Big) \quad \text{Hint: or use } \Pr(y_i=1|\mathbf{x}_i) = \frac{1}{1+e^{-\mathbf{x}_i^\top\boldsymbol{\beta}}}$$
$$p(y|\mathbf{x},\boldsymbol{\beta}) = \Phi(\mathbf{x}^\top\boldsymbol{\beta}_{p\times 1})^y(1-\Phi(\mathbf{x}^\top\boldsymbol{\beta}_{p\times 1}))^{1-y}$$

2. How could the middle model be used to create ***multivariate linear regression*** predicting a ***vector*** outcome $\mathbf{y}$ (as opposed to just ***linear regression*** predicting a ***univariate*** y_i, which is actually what the first model does...)?

3. Show that $\Phi(\mathbf{x}^\top\boldsymbol{\beta}_{p\times 1}) = \Pr(z \geq 0)$ for $z \sim \mathcal{N}(\mathbf{x}^\top\boldsymbol{\beta}_{p\times 1}, 1)$
   Hint: show that $\int_{-\infty}^{x^\top\boldsymbol{\beta}_{p\times 1}} e^{-\frac{1}{2}z^2}dz = \int_0^\infty e^{-\frac{1}{2}(z-x^\top\boldsymbol{\beta}_{p\times 1})^2}dz$

Class Activity 6: Solutions + Another Question [10 minutes]

1. The first is ***linear regression model*** where the features $\mathbf{X}$ are used to predict outcomes $\mathbf{y}$; whereas,
   the second is simply a ***multivariate normal distribution***; whereas,
   the third is a ***generalized linear model*** predicting a binary outcome (often called a ***classification model***) parameterizing the chance of "success" using either a ***probit*** $\Phi(\mathbf{x}^\top\boldsymbol{\beta}_{p\times 1})$ or ***logit*** $\frac{1}{1+e^{-\mathbf{x}^\top\boldsymbol{\beta}}}$ ***link function**.

For Bayesian inference on β, σ^2 or $\boldsymbol{\mu}$ or $\boldsymbol{\Sigma}$ or β, just put priors on these parameters...

2. Each element $\mu_{ik} = \mathbf{x}_i^T \boldsymbol{\beta}_{(k)}$ of $\boldsymbol{\mu}_i$ becomes a ***linear model*** and then the ***covariance matrix*** $\boldsymbol{\Sigma}$ captures the ***residual covariance dependence structure*** observed over the ***residual multivariate outcomes*** $(\mathbf{y}_i - \boldsymbol{\mu}_i)$

$$\mu_{ik} = \mathbf{x}_i^T \boldsymbol{\beta}_{(k)} \quad \text{and} \quad p(\mathbf{y}|\boldsymbol{\mu}, \Sigma) = (2\pi)^{-n/2} \det(\boldsymbol{\Sigma})^{-1/2} \exp\left(-\frac{1}{2}(\mathbf{y}_i - \boldsymbol{\mu}_i)^\top \boldsymbol{\Sigma}^{-1} (\mathbf{y}_i - \boldsymbol{\mu}_i)\right)$$

3. $\Phi(\mathbf{x}_i^\top \boldsymbol{\beta}_{p\times 1}) = \Pr(z_i \geq 0)$ for $z_i \sim \mathcal{N}(\mathbf{x}_i^\top \boldsymbol{\beta}_{p\times 1}, 1)$ since
$\int_{-\infty}^{x_i^\top \boldsymbol{\beta}_{p\times 1}} e^{-\frac{1}{2}z_i^2} dz = \int_{-\infty}^{0} e^{-\frac{1}{2}(z_i + x_i^\top \boldsymbol{\beta}_{p\times 1})^2} dz = \int_{-\infty}^{0} e^{-\frac{1}{2}(-z_i - x_i^\top \boldsymbol{\beta}_{p\times 1})^2} dz = \int_{0}^{\infty} e^{-\frac{1}{2}(z_i - x_i^\top \boldsymbol{\beta}_{p\times 1})^2} dz$

4. Specify a ***multivariate generalized linear regression*** model whose ***marginal distributions*** are each univariate ***bernoulli distributions*** but which also ***parameterizes*** a ***joint dependency*** structure between ***marginal outcomes*** Hint: $\Pr(y_{ij} = 1|\mathbf{x}_i) = \Pr(z_{ij} \geq 0)$ and specify a joint distribution for unobserved $\boldsymbol{z}_i$ that depends on $\mathbf{x}_i$

Class Activity 6: Solutions [5 minutes]

4. Specify a ***multivariate generalized linear regression*** model whose ***marginal distributions*** are each univariate ***bernoulli distributions*** but which also ***parameterizes*** a ***joint dependency*** structure between ***marginal outcomes*** Hint: $\Pr(y_{ij} = 1|\mathbf{x}_i) = \Pr(z_{ij} \geq 0)$ and specify a joint distribution for unobserved $\boldsymbol{z}_i$ that depends on $\mathbf{x}_i$

Generalized Multivariate Linear Model (GMLM)

- $\Pr(y_{ij} = 1|\mathbf{x}_i) = \Pr(z_{ij} \geq 0)$
- $\mathbf{z}_i \sim \mathcal{MVN}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma} = \mathbf{DRD} = \mathbf{R})$

$$p(\mathbf{z}_i|\boldsymbol{\mu}_i, \boldsymbol{\Sigma} = \mathbf{DRD} = \mathbf{R}) = (2\pi)^{-n/2} \det(\mathbf{R})^{-1/2} \exp\left(-\frac{1}{2}(\mathbf{z}_i - \boldsymbol{\mu}_i)^\top \mathbf{R}^{-1} (\mathbf{z}_i - \boldsymbol{\mu}_i)\right)$$

- with $\mu_{ij} = \mathbf{x}_i^T \boldsymbol{\beta}_{(j)}$

and where $\mathbf{R}$ is used in the same spirit as $\Pr(y_{ij}=1|\mathbf{x}_i) = \Phi(\mathbf{x}_i^\top \boldsymbol{\beta}_{p\times 1}) = \Pr(z_i \geq 0)$ for $z_i \sim \mathcal{N}(\mathbf{x}_i^\top \boldsymbol{\beta}_{p\times 1}, 1)$ but additionally captures ***residual correlation dependence structure*** observed over the ***residual multivariate outcomes***

The transformation on the ***marginal distribition*** to the ***probability parameter*** $\Pr(y_{ij}=1|\mathbf{x}_i)$ of a ***Bernoulli distribution*** could be carried out with the ***logit*** $\frac{1}{1+e^{-z_{ij}}}$ ***link function*** instead of the ***probit*** $\Phi(\mu_{ij} = \mathbf{x}_i^T \boldsymbol{\beta}_{(j)})$ ***link function*** as above

- The transformation of the ***marginal distributions*** of a ***multivariate normal random variable*** into other ***distributional forms*** (such as ***Bernoulli distributions*** as done here) is the core principle behind ***copulas***

Homework 6: Part I

1. Go get data from kaggle.com and do a ***(Univariate) Bayesian Logistic Regression*** analysis

2. Adjust the code below to specify that the outcomes have a Bernoulli distribution and use a ***logit*** or ***probit link function*** (or $\Pr(z \leq 0)$ for latent z) to correctly paramterize the predicted values of the observed outcomes

```python
import pymc as pm; import numpy as np
n,p=100,10; X,y=np.zeros((n,p)),np.ones((n,1))
# Replace this made up data with your data set from kaggle...
with pm.Model() as MLR:
    betas = pm.MvNormal('betas', mu=np.zeros((p,1)), cov=np.eye(p), shape=(p,1))
    sigma = pm.TruncatedNormal('sigma', mu=1, sigma=1, lower=0) # half normal
    y = pm.Normal('y', mu=pm.math.dot(X, betas), sigma=sigma, observed=y)

with MLR:
    idata = pm.sample()
```

3. Choose ***prior*** that are sensible for your specification

4. [Optional] Assess the performance of the MCMC and any issues or warnings in the standard manner

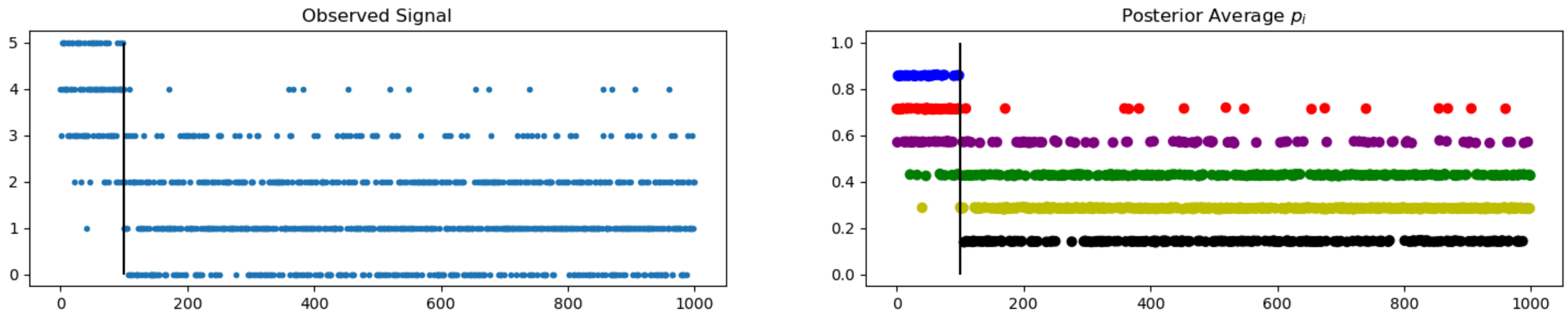
5. [Optional] Go get data from kaggle.com and do a ***Multivariate Bayesian Logistic Regression*** analysis

Bayesian Multiplicity Adjustment: Significance Dichotomy [7 minutes]

Hypothesis testing involves a mixture of ***effect sizes***, some large and some negligable

This can coarsely be viewed as the dichotomy of ***significant*** versus ***insignificant tests***

In [ ]:
```
import numpy as np; from scipy import stats; import matplotlib.pyplot as plt;
fig,ax = plt.subplots(1,2,figsize=(18,3))
# first 100 are "interesting" (high p) distributions;
# but the (posterior p) "evidence" depends only on observed outcome
np.random.seed(1);
y_obs = np.r_[stats.binom(p=0.75,n=5).rvs(100),stats.binom(p=0.25,n=5).rvs(900)];
ax[0].plot(y_obs,'.'); ax[0].set_title('Observed Signal');
ax[0].vlines(100,ymin=0,ymax=5,color='k');
ax[1].scatter(np.arange(1000), idata.posterior['p'].values.reshape((-1,1000)).mean(axis=0),c=[['k','y','g','purple'
ax[1].set_title('Posterior Average $p_i$');
ax[1].vlines(100,ymin=0,ymax=1,color='k');
```



In [ ]:
```
import pymc as pm
with pm.Model() as no_multiplicity_correction:
    p = pm.Beta('p', alpha=1, beta=1, shape=1000)
    y = pm.Binomial('y', n=5, p=p, observed=y_obs)
```

```
with no_multiplicity_correction:
    idata = pm.sample()
```

```
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [p]
```

100.00% [8000/8000 00:03<00:00 Sampling 4 chains, 0 divergences]

```
Sampling 4 chains for 1_000 tune and 1_000 draw iterations (4_000 + 4_000 draws total) took 3 seconds.
```

Bayesian Multiplicity Adjustment: Hierarchical Modeling [10 minutes]

$$\cancel{p_i} \sim \cancel{\text{beta}(\alpha,\beta)} \qquad p \sim \text{beta}(\alpha,\beta) \quad x_i \sim \text{Bernoulli}(p) \qquad p_0 \sim \text{beta}(\alpha_0,\beta_0)$$

$$\cancel{y_i} \sim \cancel{\text{binomial}(p,n)} \qquad y_i \sim \text{binomial}(\underbrace{p_0 + x_i p_1}_{\text{restrict to }[0,1]}, n) \qquad p_1 \sim \text{beta}(\alpha_1,\beta_1)$$

In [ ]:
```
with pm.Model() as multiplicity_correction:
    p = pm.Beta('p', alpha=1, beta=1)
    x = pm.Binomial('x', n=1, p=p, shape=1000)
    p0 = pm.Beta('p0', alpha=1, beta=1)
    p1 = pm.Beta('p1', alpha=1, beta=1)
    no_negatives = pm.math.switch(pm.math.lt(p0+x*p1,0), 0, p0+x*p1)
    in_unit_interval = pm.math.switch(pm.math.gt(no_negatives,1), 1, p0+x*p1)
    p01 = pm.Deterministic('p01', in_unit_interval)
    y = pm.Binomial('y', n=5, p=p01, observed=y_obs) # p=0.25+x*0.5

with multiplicity_correction:
    idata2 = pm.sample()
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>NUTS: [p, p0, p1]
>Metropolis: [x]
```

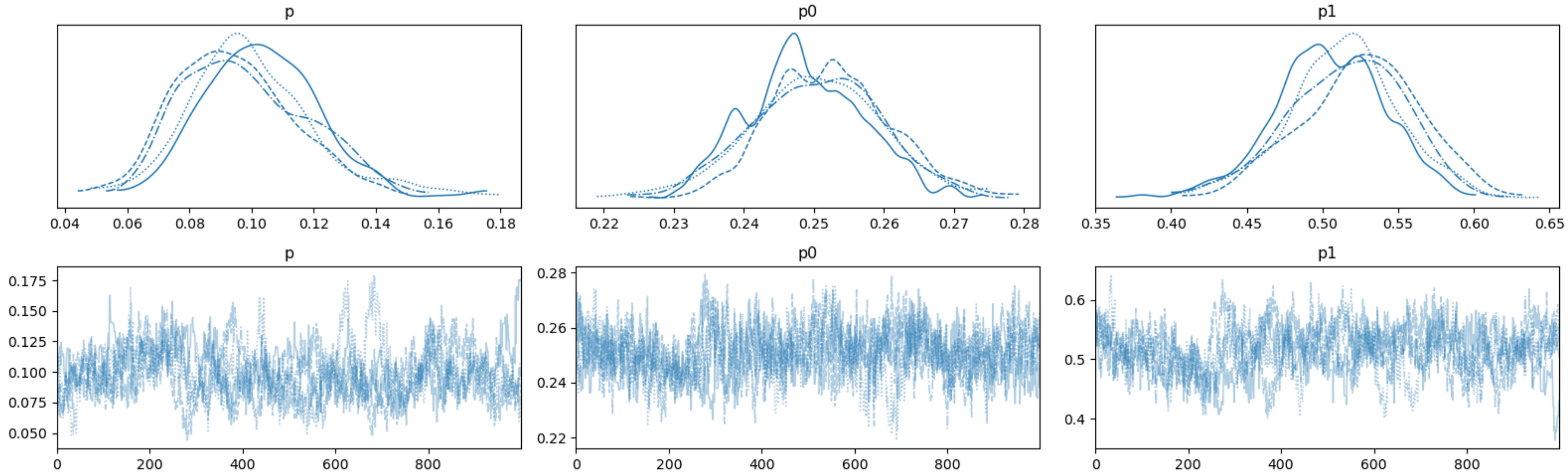
100.00% [8000/8000 02:34<00:00 Sampling 4 chains, 0 divergences]

```
Sampling 4 chains for 1_000 tune and 1_000 draw iterations (4_000 + 4_000 draws total) took 155 seconds.
/Users/scottschwartz/miniconda3/envs/PyMC/lib/python3.11/site-packages/arviz/stats/diagnostics.py:592: RuntimeWarnin
g: invalid value encountered in scalar divide
  (between_chain_variance / within_chain_variance + num_samples - 1) / (num_samples)
The rhat statistic is larger than 1.01 for some parameters. This indicates problems during sampling. See https://arx
iv.org/abs/1903.08008 for details
The effective sample size per chain is smaller than 100 for some parameters.  A higher number is needed for reliable
rhat and ess computation. See https://arxiv.org/abs/1903.08008 for details
```

Bayesian Multiplicity Adjustment: Hierarchical Modeling [3 minutes]

$$\begin{array}{llll} \cancel{p_i} \sim \cancel{\text{beta}(\alpha,\beta)} & p \sim \text{beta}(\alpha,\beta) \quad x_i \sim \text{Bernoulli}(p) & & p_0 \sim \text{beta}(\alpha_0,\beta_0) \\ \cancel{y_i} \sim \cancel{\text{binomial}(p,n)} & y_i \sim \text{binomial}(\underbrace{p_0 + x_i p_1}_{\text{restrict to } [0,1]}, n) & & p_1 \sim \text{beta}(\alpha_1,\beta_1) \end{array}$$

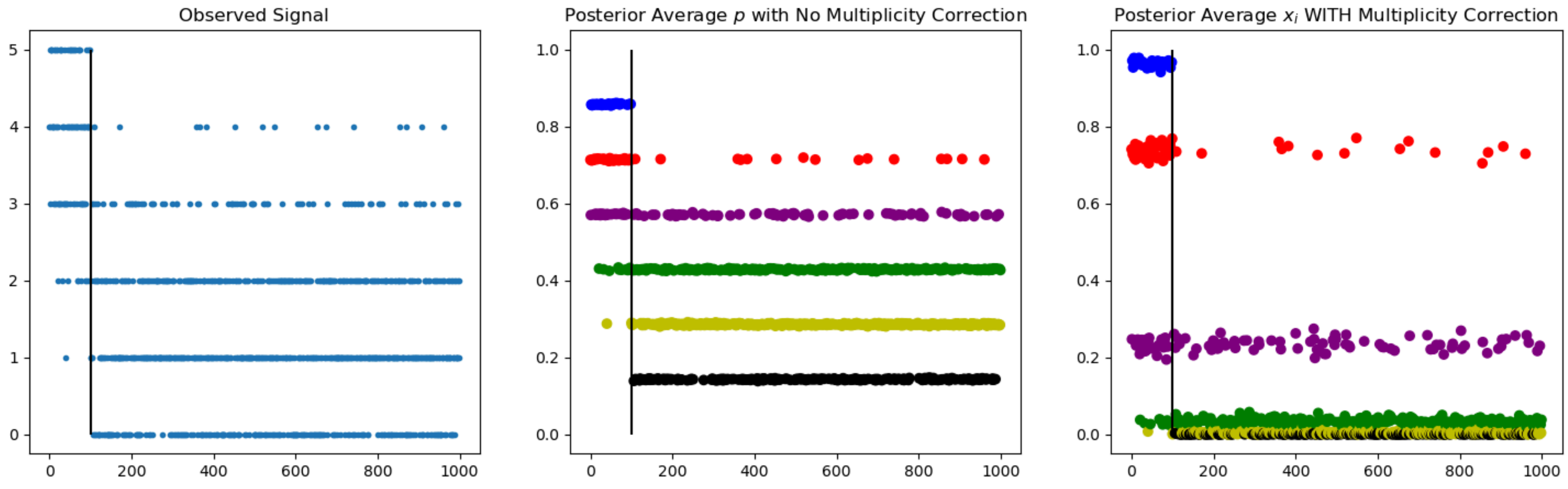
In [ ]:
```
fig,ax = plt.subplots(2,3,figsize=(16,5))
import arviz as az; az.plot_trace(idata2, var_names=["p","p0","p1"], axes=ax.T);
plt.tight_layout();
```



Bayesian Multiplicity Adjustment: Hierarchical Modeling and the Power of Information Sharing [5 minutes]

Learning the proportion of examples that are "interesting" actually means learning
WHICH examples are interesting by ***imputing*** weather or not each of the examples is interesting one or not

In [ ]:
```
# first 100 are "interesting" (high p) distributions; but the (posterior p) "evidence" depends only on observed out
fig,ax = plt.subplots(1,3,figsize=(18,5)); np.random.seed(1);
y_obs = np.r_[stats.binom(p=0.75,n=5).rvs(100),stats.binom(p=0.25,n=5).rvs(900)];
ax[0].plot(y_obs,'.'); ax[0].set_title('Observed Signal');
ax[0].vlines(100,ymin=0,ymax=5,color='k');
ax[1].scatter(np.arange(1000), idata.posterior['p'].values.reshape((-1,1000)).mean(axis=0),
              c=[['k','y','g','purple','r','b'][c] for c in y_obs]);
ax[1].set_title('Posterior Average $p$ with No Multiplicity Correction');
ax[1].vlines(100,ymin=0,ymax=1,color='k');
ax[2].scatter(np.arange(1000), idata2.posterior['x'].values.reshape((-1,1000)).mean(axis=0),
              c=[['k','y','g','purple','r','b'][c] for c in y_obs]);
ax[2].set_title('Posterior Average $x_i$ WITH Multiplicity Correction');
ax[2].vlines(100,ymin=0,ymax=1,color='k');
```



Bayesian Variable Selection: "Spike and Slab" [12 minutes]

$$
\begin{array}{lll}
p \sim \mathrm{beta}(\alpha,\beta) & & \\
s_i \sim \mathrm{Bernoulli}(p) & b_i \sim \mathrm{Normal}(\mu_0,\sigma_0) & \beta_i = b_i \times s_i \\
y_i \sim \mathrm{Normal}(x_i^T\beta,\sigma) & \sigma \sim \mathrm{HalfNormal}(\sigma_0) &
\end{array}
$$

In [ ]:
```
m,q = 20,10; betas = np.zeros((m,1)); betas[0:q,0] = np.linspace(0,q-1,q); np.random.seed(2)
n = 100; X = stats.binom(n=1,p=0.5).rvs(size=(n,m)); y_obs=X.dot(betas).flatten() + stats.norm().rvs(size=n)

with pm.Model() as spikeNslab:
    p = pm.Beta('p', alpha=1, beta=1)
    spike = pm.Binomial('spike', n=1, p=p, shape=m)
    slab = pm.Normal('slab', mu=0, sigma=10, shape=m)
    beta = pm.Deterministic('beta', spike*slab) # elementwise multiplication
    sigma = pm.HalfNormal('sigma', sigma=2)
    y = pm.Normal('y', mu=pm.math.dot(X, beta), sigma=sigma, observed=y_obs)
```

```
with spikeNslab:
    idata3 = pm.sample()
```

```
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>NUTS: [p, slab, sigma]
>Metropolis: [spike]
```

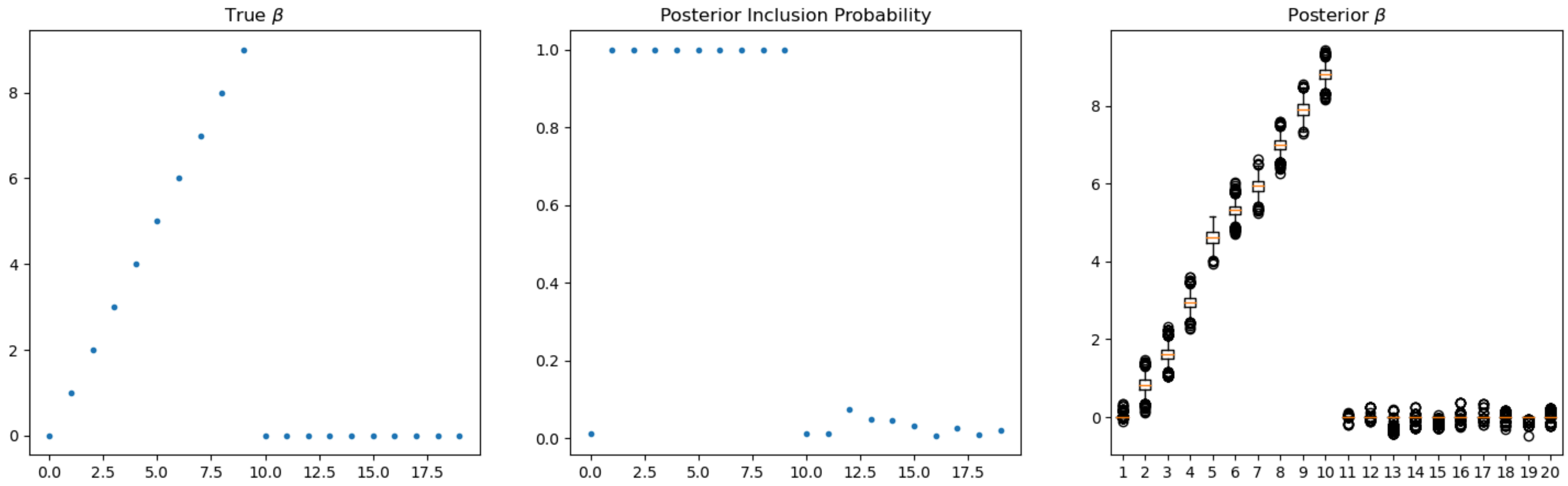
100.00% [8000/8000 00:07<00:00 Sampling 4 chains, 1,033 divergences]

```
Sampling 4 chains for 1_000 tune and 1_000 draw iterations (4_000 + 4_000 draws total) took 8 seconds.
/Users/scottschwartz/miniconda3/envs/PyMC/lib/python3.11/site-packages/arviz/stats/diagnostics.py:592: RuntimeWarnin
g: invalid value encountered in scalar divide
  (between_chain_variance / within_chain_variance + num_samples - 1) / (num_samples)
The rhat statistic is larger than 1.01 for some parameters. This indicates problems during sampling. See https://arx
iv.org/abs/1903.08008 for details
The effective sample size per chain is smaller than 100 for some parameters.  A higher number is needed for reliable
rhat and ess computation. See https://arxiv.org/abs/1903.08008 for details
There were 1033 divergences after tuning. Increase `target_accept` or reparameterize.
```

Bayesian Variable Selection: "Spike and Slab" [4 minutes]

$$\begin{align*} p &\sim \text{beta}(\alpha, \beta) & & \\ s_i &\sim \text{Bernoulli}(p) & b_i &\sim \text{Normal}(\mu_0, \sigma_0) & \beta_i &= b_i \times s_i \\ y_i &\sim \text{Normal}(x_i^T\beta, \sigma) & \sigma &\sim \text{HalfNormal}(\sigma_0) & & \end{align*}$$

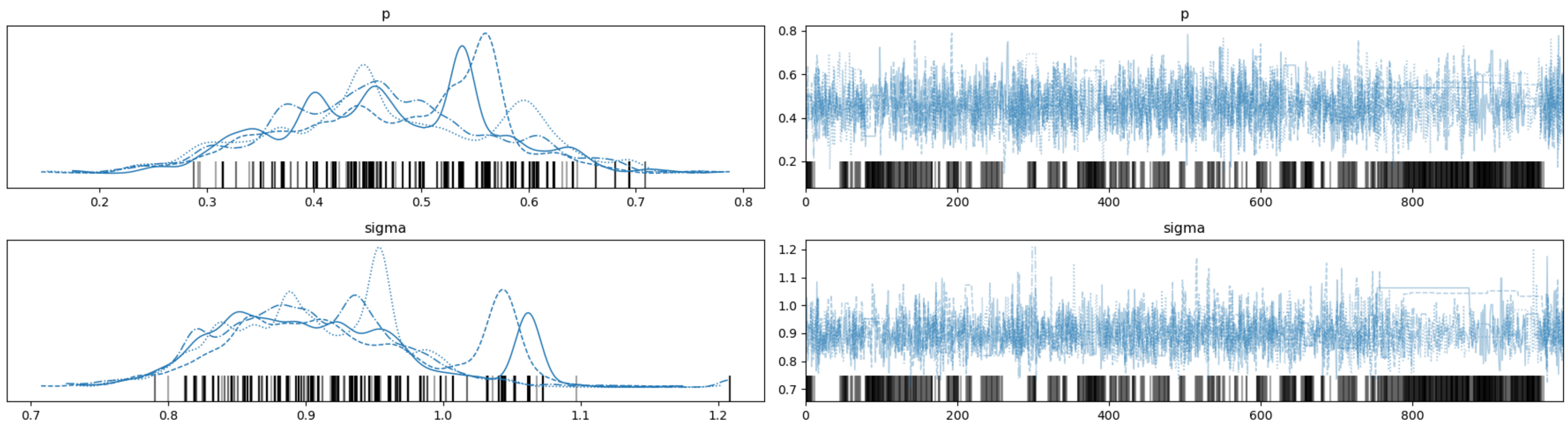
In [ ]:
```
fig,ax = plt.subplots(1,3,figsize=(18,5)); ax[0].plot(betas,'.'); ax[0].set_title('True $\\beta$'); ax[1].plot(idat
```



Bayesian Variable Selection: "Spike and Slab" [4 minutes]

$$
\begin{aligned}
p &\sim \text{beta}(\alpha, \beta) & & & \\
s_i &\sim \text{Bernoulli}(p) & b_i &\sim \text{Normal}(\mu_0, \sigma_0) & \beta_i &= b_i \times s_i \\
y_i &\sim \text{Normal}(x_i^T \beta, \sigma) & \sigma &\sim \text{HalfNormal}(\sigma_0) & &
\end{aligned}
$$

In [ ]:
```
import arviz as az; az.plot_trace(idata3, var_names=["p","sigma"], figsize=(18,5)); plt.tight_layout();
```



Bayesian Shrinkage Estimation: "Lasso Regression" [10 minutes]

Ridge Regression L_2***-penalty*** is just using ***norml priors*** for the ***regression coefficients***

$$\beta_i \sim \text{Normal}(b_i, s_i) \qquad f(\beta_i|b_i, s_i) = \frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{1}{2}\left(\frac{\beta_i - b_i}{s_i}\right)^2} \qquad \underbrace{\overbrace{|\beta_i - b_i|^2}^{L_2}}_{\text{Squared Penalization}}$$

$$y_i \sim \text{Normal}(x_i^T\beta, \sigma) \qquad \sigma \sim \text{HalfNormal}(\sigma_0)$$

Lasso Regression L_1***-penalty*** is just replacing the ***norml priors*** with ***Laplace ("Double Exponential") prior distributions***

$$\beta_i \sim \text{Laplace}(b_i, s_i) \qquad f(\beta_i|b_i, s_i) = \frac{1}{2b}\exp\left(-\frac{|\beta_i - b_i|}{s_i}\right) \qquad \underbrace{\overbrace{|\beta_i - b_i|}^{L_1}}_{\text{Absolute Penalization}}$$

$$y_i \sim \text{Normal}(x_i^T\beta, \sigma) \qquad \sigma \sim \text{HalfNormal}(\sigma_0)$$

Homework 6: Part II
Regularized Loss Functions

Machine Learning fits models by optimizing penalized ***loss functions***

Two classic regularizations are "ridge" and "lasso" regression, which respectively use L_2 and L_1 penalty functions

- Lasso:

$$\sum_{i=1}^{n} \frac{1}{2}(y_i - x_i^T\beta_{p\times 1})^2 + \lambda \sum_{j=1}^{n} \beta_j^2 = \frac{1}{2}(y - X\beta)^T(y - X\beta) + \lambda \sum_{j=1}^{n} \beta_j^2 = \frac{1}{2}||y - X\beta||_2^2 + \lambda||\beta||_2^2$$

- Ridge:

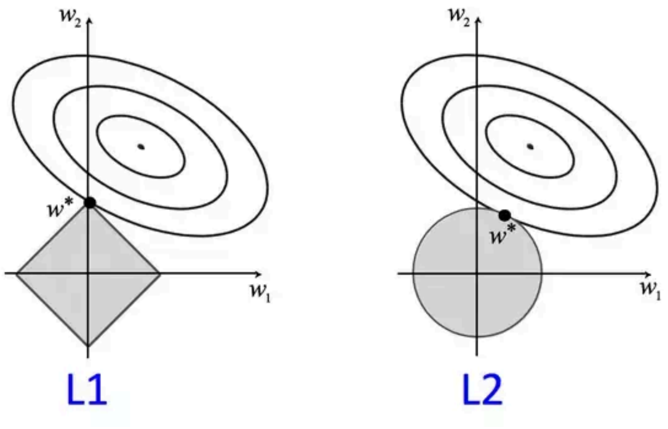
$$\sum_{i=1}^{n} \frac{1}{2}(y - x_i^T\beta_{p\times 1})^2 + \lambda \sum_{j=1}^{n} |\beta_j| = \frac{1}{2}(y - X\beta)^T(y - X\beta) + \lambda \sum_{j=1}^{n} |\beta_j| = \frac{1}{2}||y - X\beta||_2^2 + \lambda||\beta||_1$$

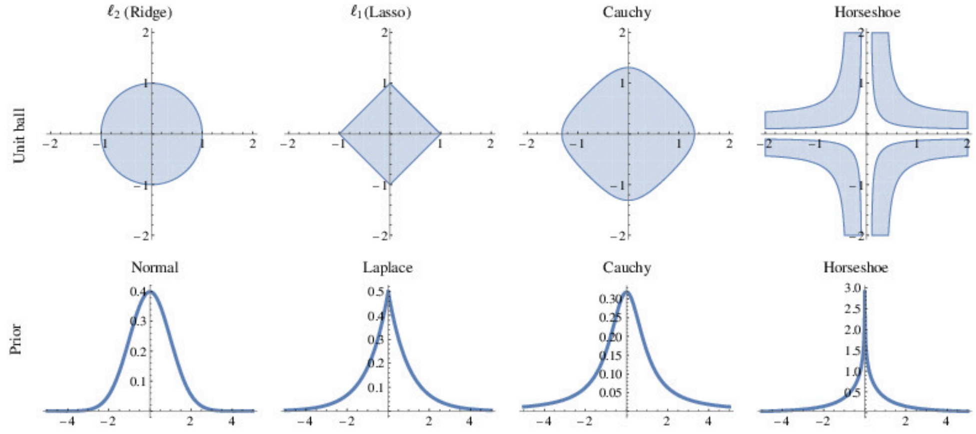
Show that for $\sigma = 1$ and ***hyperparameters*** $b_i = 0$ (ignoring normalizing proportionality constants) the log posterior distributions for β using either ***normal*** or ***Laplace*** prior distributions have analagous forms to the above expressions

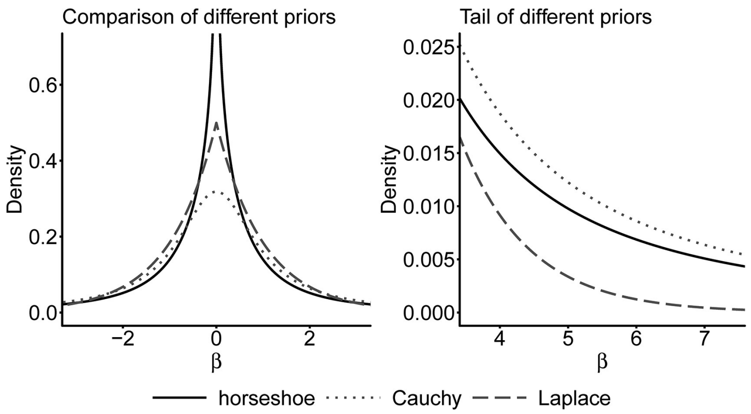
Now write down and understand the following: "Bayesians do not optimize posterior distributions, they sample from them; but, the posterior distributions are nonetheless 'regularizations' of the likelihood through the prior."

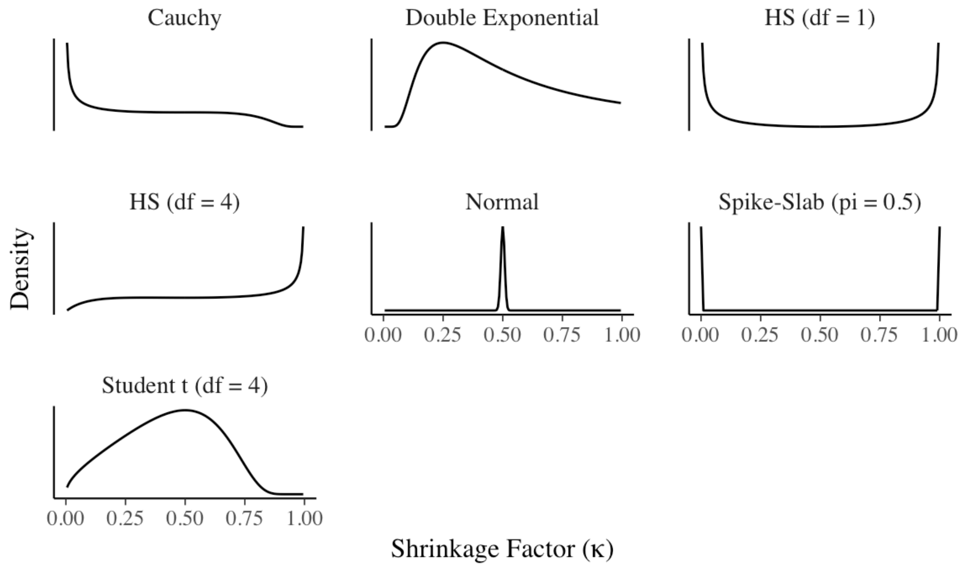
Bayesian Shrinkage Estimation: Geometrically [15 minutes]

Machine Learning thought they invented "regularization"... *Priors are Regularization!*
But Bayes been regularizing since 1763: Bayesian analysis is the original regularization methodology









Bayesian Shrinkage Estimation: "Lasso Regression" [2 minutes]

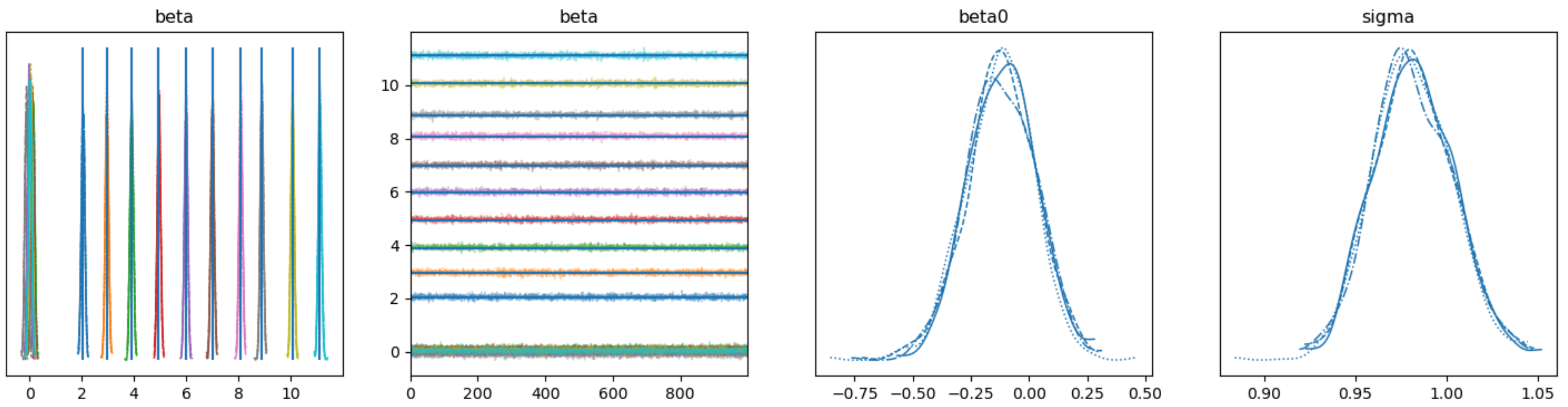
In [ ]:
```
m,q = 20,10; betas = np.zeros((m,1)); betas[0:q,0] = np.linspace(2,q+1,q); np.random.seed(2)
n = 1000; X = stats.binom(n=1,p=0.5).rvs(size=(n,m)); y_obs=X.dot(betas).flatten() + stats.norm().rvs(size=n)
with pm.Model() as lasso:
    beta = pm.Laplace('beta', mu=0, b=1, shape=m); beta0 = pm.Normal('beta0', mu=0, sigma=10); sigma = pm.HalfNorma
    y = pm.Normal('y', mu=beta0+pm.math.dot(X, beta), sigma=sigma, observed=y_obs)
    idata4 = pm.sample()
```

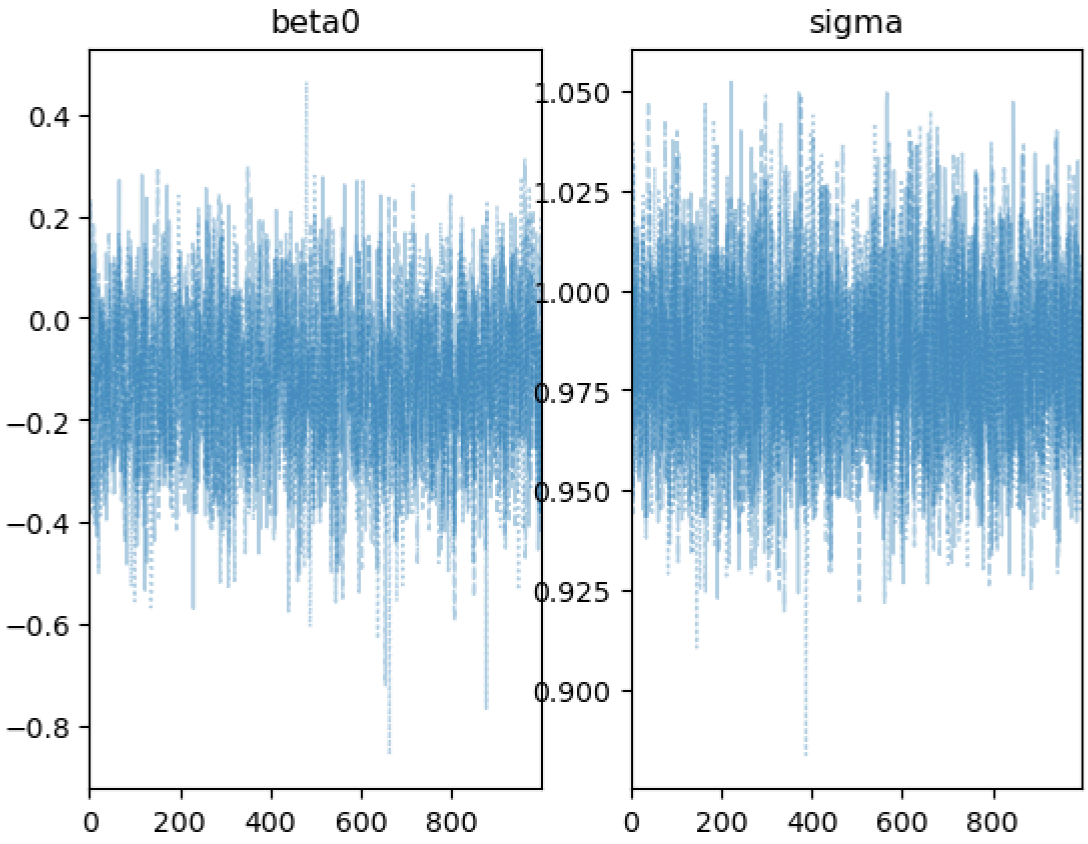
```
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [beta, beta0, sigma]
```

100.00% [8000/8000 00:13<00:00 Sampling 4 chains, 0 divergences]

```
Sampling 4 chains for 1_000 tune and 1_000 draw iterations (4_000 + 4_000 draws total) took 13 seconds.
```

In [ ]:
```
fig,ax = plt.subplots(1,4,figsize=(18,4)); fig2,ax2 = plt.subplots(1,2); az.plot_trace(idata4, axes=np.r_[ax[:3],ax
```





Bayesian Shrinkage Estimation: "Lasso Regression" [3 minutes]

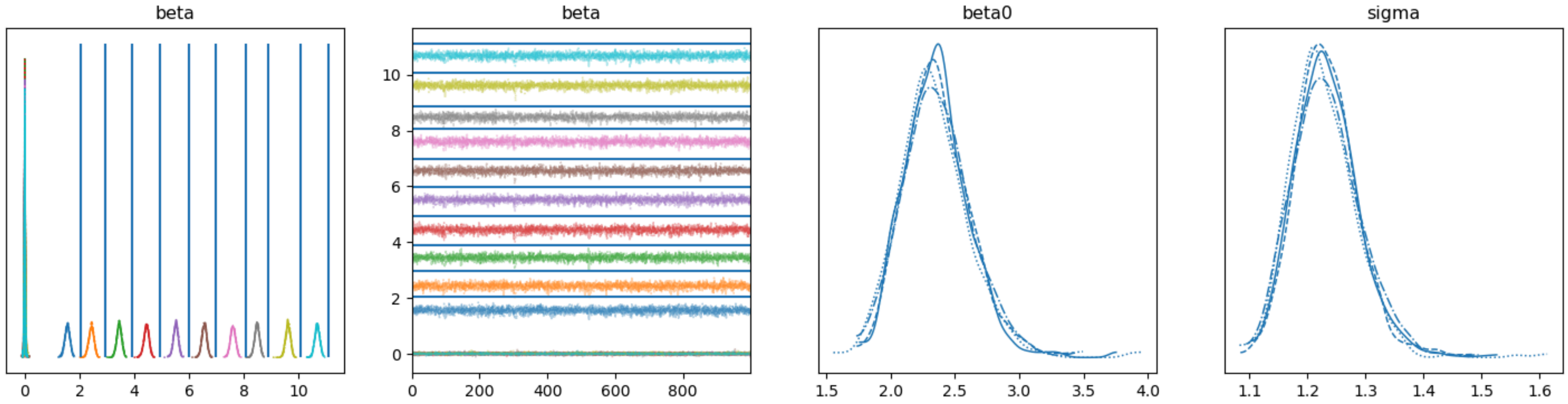
In [ ]:
```
with pm.Model() as lasso:
    beta = pm.Laplace('beta', mu=0, b=1/75, shape=m, initval=betas[:,0])
    beta0 = pm.Normal('beta0', mu=0, sigma=10); sigma = pm.HalfNormal('sigma', sigma=100)
    y = pm.Normal('y', mu=beta0+pm.math.dot(X, beta), sigma=sigma, observed=y_obs)
with lasso:
    idata4 = pm.sample()
```

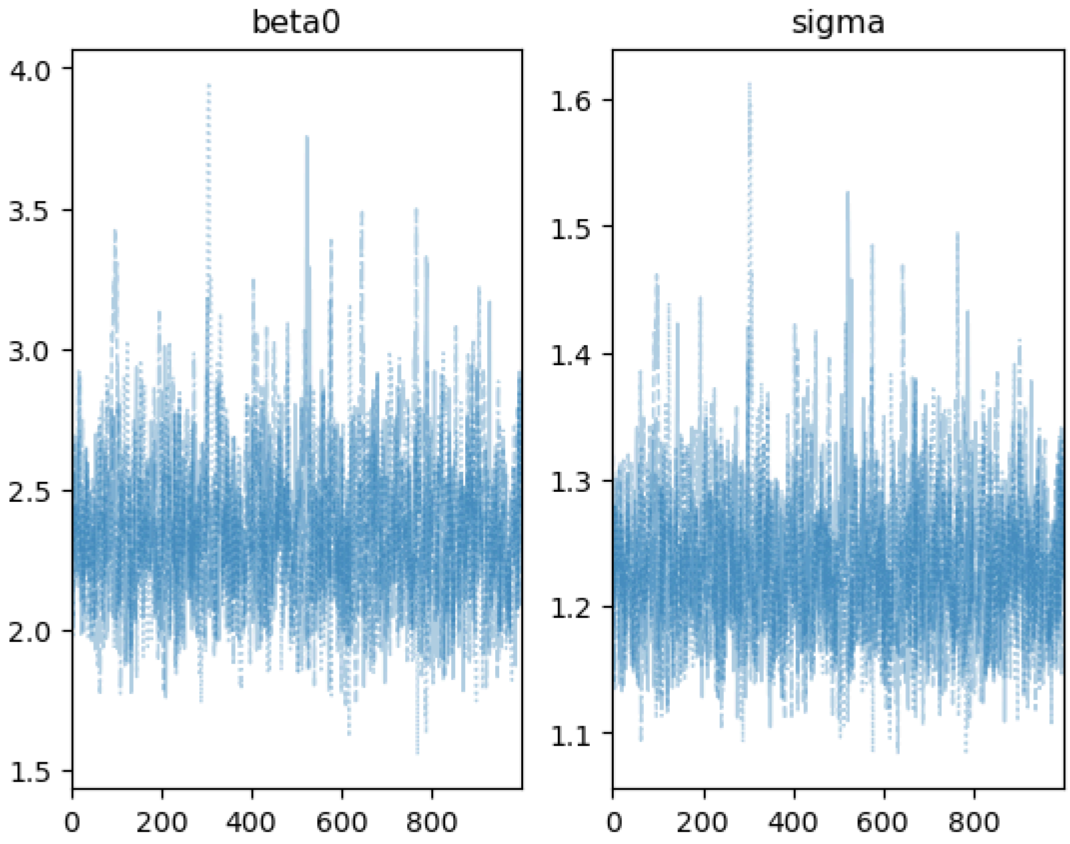
```
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [beta, beta0, sigma]
```

100.00% [8000/8000 00:26<00:00 Sampling 4 chains, 0 divergences]

```
Sampling 4 chains for 1_000 tune and 1_000 draw iterations (4_000 + 4_000 draws total) took 26 seconds.
```

In [ ]:
```
fig,ax = plt.subplots(1,4,figsize=(18,4)); fig2,ax2 = plt.subplots(1,2); az.plot_trace(idata4, axes=np.r_[ax[:3],ax
```





Bayesian Shrinkage Estimation: "The Horseshoe" [15 minutes]

The PyMC overview and many other resources provide ***Horseshoe prior*** [1] [2] implementations

| Half-Cauchy $\mathrm{HC}_+(\xi)$ | Horseshoe Prior HSP | Shrinkage κ | Change of Variables| |:-:|:-:|:-:|:-:| |

$$f(x \mid \xi) = \frac{2 \cdot 1_{[x>=0]}(x)}{\pi\xi\left[1 + \left(\frac{x}{\xi}\right)^2\right]}$$

|

$$
\begin{aligned}
w_i|\tau &\sim N(0, \sigma^2 = \lambda_i^2\tau^2)\\
\lambda_i &\sim HC_+(1)\\
\tau &\sim HC_+(\tau_0)
\end{aligned}
$$

|

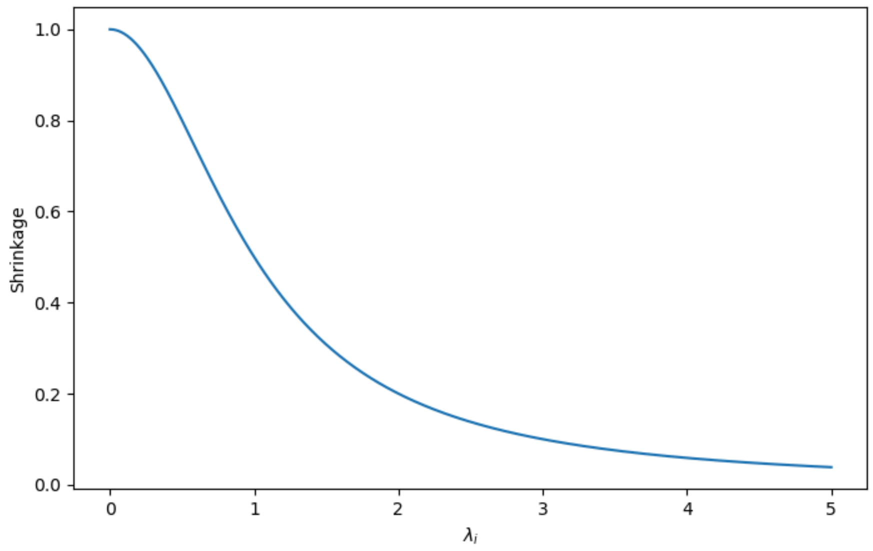
$$
\begin{aligned}
\kappa_{\lambda_i} &= 1/(1+\lambda_i^2)\\
\lambda_i &= \sqrt{1/\kappa_{\lambda_i} - 1}\\
J_{\kappa_{\lambda_i}} &= \frac{1}{2}(\kappa_{\lambda_i}^{-1} - 1)^{-\frac{1}{2}} \times \kappa_{\lambda_i}^{-2}
\end{aligned}
$$

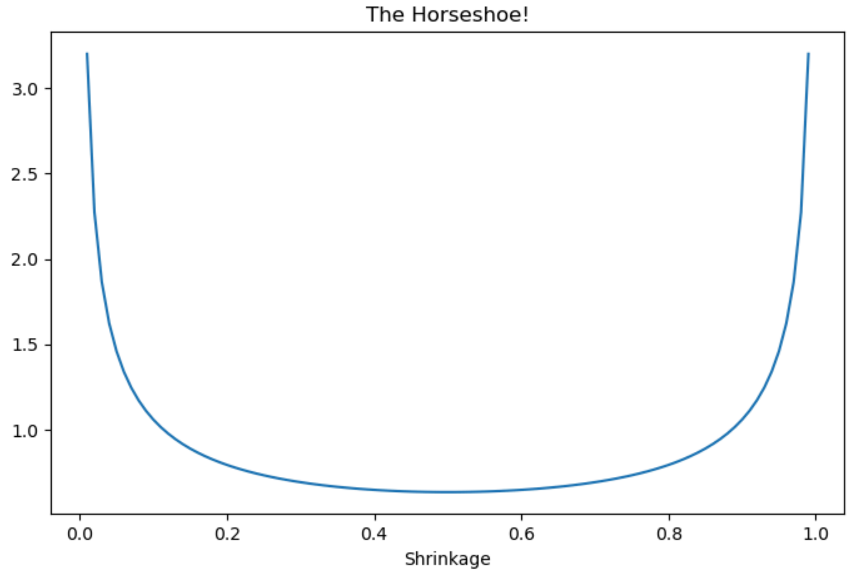
|

$$
\begin{aligned}
f(\kappa_{\lambda_i}) = f&\left(\lambda_i = \sqrt{1/\kappa_{\lambda_i} - 1}\right)\\
&\times \underbrace{\frac{1}{2}(\kappa_{\lambda_i}^{-1} - 1)^{-\frac{1}{2}} \times \kappa_{\lambda_i}^{-2}}_{J_{\kappa_{\lambda_i}}}
\end{aligned}
$$

|

In [ ]:
```
fig,ax = plt.subplots(1,2,figsize=(18,5)); support = np.linspace(0,5,1000); # shrnk = trans(spprt) = 1/(1+sprt**2)
shrinkage = 1/(1+support**2); ax[0].plot(support, shrinkage); ax[0].set_ylabel("Shrinkage"); ax[0].set_xlabel("$\\l
# change of variables: spprt = (1/shrnk-1)**0.5; E.g., 1/(1+.5**2), (1/.8-1)**0.5; jacobian: .5(1/shrnk-1)**(-.5)*s
shrinkage = np.linspace(0.01,.99,99); ax[1].plot(shrinkage, stats.halfcauchy(scale=1).pdf((1/shrinkage-1)**0.5) * .
```

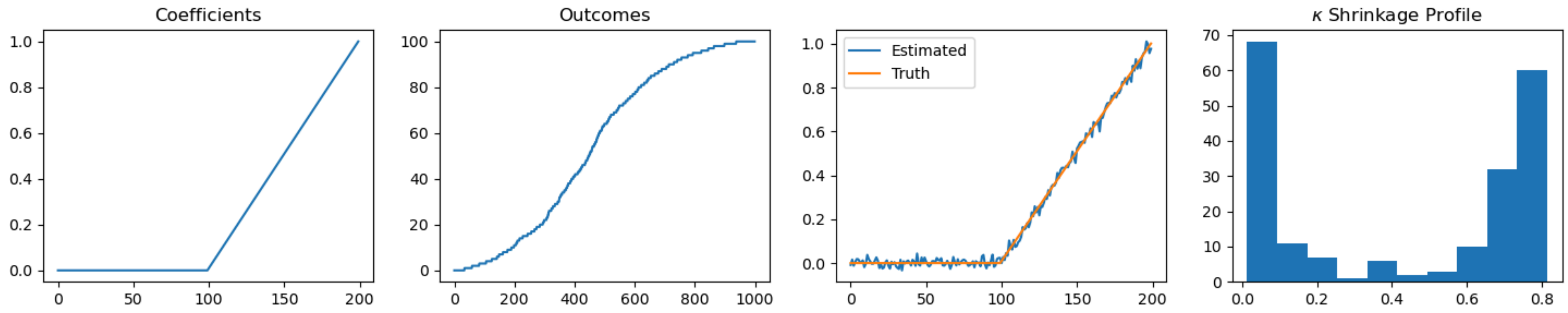




Bayesian Shrinkage Estimation: "The Horseshoe" [15 minutes]

Trying it for ***binomial regression*** coefficients... a linear model and nonlinear transformation for non normal outcomes

```
In [ ]: from scipy.special import expit as invlogit; fig,ax = plt.subplots(1,4,figsize=(18,3)); K,Q=100,100; w=np.arange(1,
        # nonzero and noise coefficients "weights" w (beta is reserved for the Half-Cauchy parameter below)
        P=1000; X=stats.bernoulli(p=0.5).rvs(size=(K+Q)*P).reshape(P,(K+Q)); N=100; THETA=-25; x=stats.binom(p=invlogit(THE
        ax[2].plot(posterior.posterior['weights'].values.reshape((-1,4000,200))[0,:,:].mean(axis=0), label='Estimated'); ax
        ax[3].hist(1/(1+posterior.posterior['lambdas'].values.reshape((-1,4000,200))[0,:,:].mean(axis=0)**2)); ax[3].set_ti
```



In [ ]:
```
from pymc.math import invlogit as tt_invlogit
with pm.Model() as horseshoe:
    tau_0 = 1; tau = pm.HalfCauchy('tau', beta=tau_0, shape=1)
    lambdas = pm.HalfCauchy('lambdas', beta=1, shape=(Q+K)); theta = pm.Normal('theta', mu=0, sigma=50)
    weights = pm.Normal('weights', mu=0, sigma=tau*lambdas, shape=(Q+K))
    successes = pm.Binomial('successes', p=tt_invlogit(X@weights+THETA), n=[N]*P, observed=x)
    posterior = pm.sample()
```

```
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [tau, lambdas, theta, weights]
```

100.00% [8000/8000 02:38<00:00 Sampling 4 chains, 831 divergences]

```
Sampling 4 chains for 1_000 tune and 1_000 draw iterations (4_000 + 4_000 draws total) took 159 seconds.
The rhat statistic is larger than 1.01 for some parameters. This indicates problems during sampling. See https://arx
iv.org/abs/1903.08008 for details
The effective sample size per chain is smaller than 100 for some parameters.  A higher number is needed for reliable
rhat and ess computation. See https://arxiv.org/abs/1903.08008 for details
There were 831 divergences after tuning. Increase `target_accept` or reparameterize.
```

Bayesian Shrinkage Estimation: "Regularized Horseshoe" [10 minutes]

$$\kappa_j = \frac{1}{1+\lambda_j^2} \quad \text{for } \tau = \tau_0 = n = 1 \text{ and } \sigma^2 = Var(x_j) = s_{x_j}^2 \quad \text{generalizes too} \quad \frac{1}{1+n\sigma^{-2}\tau^2 s_{x_j}^2 \lambda_j^2}$$

The *smaller* $\frac{n}{\sigma^2}\tau^2$ (with $s_{x_j}^2 = 1$) is the *greater* the number of the D parameters experiencing shrinkage [1] [2] [3]

- ***Hyperparameter*** τ_0 drives the ***effective number of parameters*** $E\left[\sum_{k=1}^D 1-\kappa_j\right] = \frac{\frac{\tau_0}{\sigma}\sqrt{n}}{1+\frac{\tau_0}{\sigma}\sqrt{n}} D = p_0$
  - $\tau_0 = \frac{p_0}{D-p_0}\frac{\sigma}{\sqrt{n}}$ can be interpreted based on the number of "effectively non-zero" parameters p_0

In [ ]:
```
import random; w_prime = w.copy(); w_prime[random.sample(range(len(w)), 150)] = 0 # zero out some coefficients
THETA_ = -6; x_ = stats.binom(p=invlogit(THETA_+X@w_prime), n=[N]*P).rvs(); #plt.figure(figsize=(18,3)); plt.plot(s
with pm.Model() as mod_reg_1p0:
    tau_0 = 1; tau = tau_0 #tau = pm.HalfCauchy('tau', beta=tau_0, shape=1)
    lambdas = pm.HalfCauchy('lambdas', beta=1, shape=(Q+K)); theta = pm.Normal('theta', mu=0, sigma=20)
    weights = pm.Normal('weights', mu=0, sigma=tau*lambdas, shape=(Q+K))
    successes = pm.Binomial('successes', p=tt_invlogit(X@weights+theta), n=[N]*P, observed=x_)
    posterior_reg_1p0 = pm.sample()
with pm.Model() as mod_reg_0p03:
    tau_0 = 0.03; tau = tau_0 #tau = pm.HalfCauchy('tau', beta=tau_0, shape=1)
    lambdas = pm.HalfCauchy('lambdas', beta=1, shape=(Q+K)); theta = pm.Normal('theta', mu=0, sigma=20)
    weights = pm.Normal('weights', mu=0, sigma=tau*lambdas, shape=(Q+K))
    successes = pm.Binomial('successes', p=tt_invlogit(X@weights+theta), n=[N]*P, observed=x_)
    posterior_reg_0p03 = pm.sample()
```

```
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [lambdas, theta, weights]
```

100.00% [8000/8000 03:32<00:00 Sampling 4 chains, 1,158 divergences]

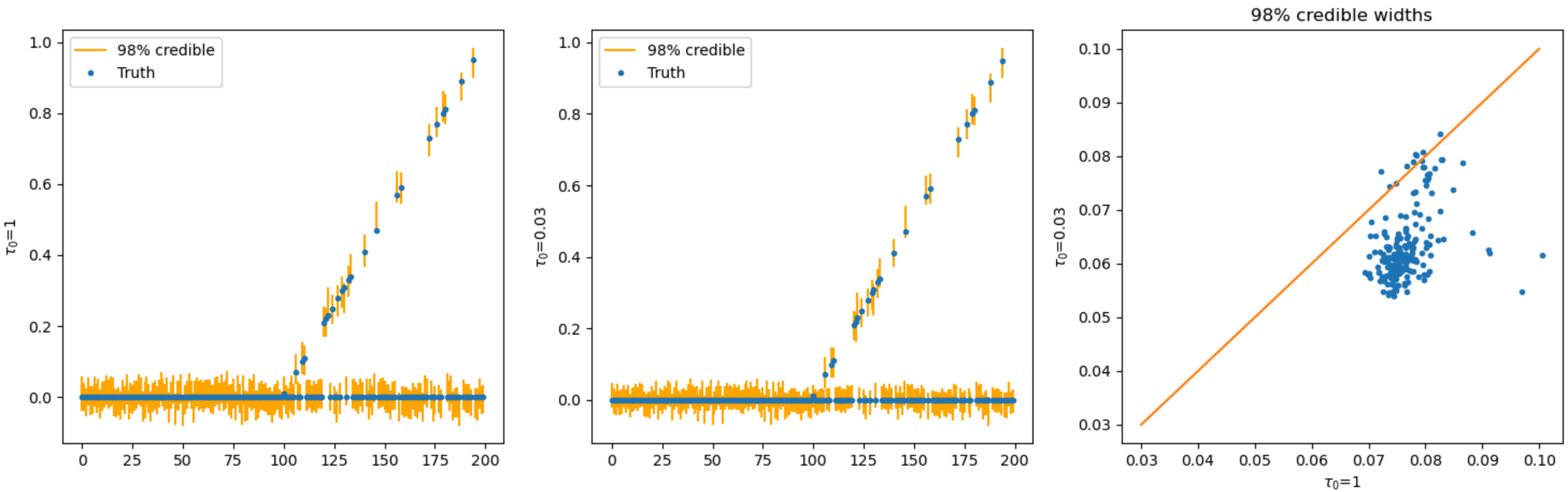
```
Sampling 4 chains for 1_000 tune and 1_000 draw iterations (4_000 + 4_000 draws total) took 213 seconds.
The rhat statistic is larger than 1.01 for some parameters. This indicates problems during sampling. See https://arx
iv.org/abs/1903.08008 for details
The effective sample size per chain is smaller than 100 for some parameters.  A higher number is needed for reliable
rhat and ess computation. See https://arxiv.org/abs/1903.08008 for details
There were 1158 divergences after tuning. Increase `target_accept` or reparameterize.
```

Bayesian Shrinkage Estimation: "Regularized Horseshoe" [5 minutes]

In [ ]:
```
# Approximating the calculations of the previous page which are technically for linear model regression
sigma2hat = \
(x_ - N*invlogit(posterior_reg_1p0.posterior['theta'].values.reshape((-1,4000,1))[0].mean(axis=0)+
                 X@posterior_reg_1p0.posterior['weights'].values.reshape((-1,4000,200))[0].mean(axis=0))).var()
tau0 = ((w_prime!=0).sum()/(Q+K-(w_prime!=0).sum()))*(sigma2hat/(X.var(axis=0).mean()*1000))**0.5; tau0
#tau0*((X.var(axis=0).mean()*1000)/sigma2hat)**0.5/(1+tau0*((X.var(axis=0).mean()*1000)/sigma2hat)**0.5)*200
```

Out[ ]: 0.027886390597189

In [ ]:
```
fix,ax = plt.subplots(1,3,figsize=(18,5)); par_1p0 = np.array(2*[np.arange(len(posterior_reg_1p0.posterior['weights
```



Homework 6: Part III
Robust regression: scale mixtures (of normals)

$$\int \frac{w\lambda_i}{\sqrt{2\pi}} e^{-\frac{1}{2}\left(w^{-2}\lambda_i^{-2}(y_i-\mu)^2\right)} \frac{\frac{\nu}{2}^{\frac{\nu}{2}}}{\Gamma(\frac{\nu}{2})} \lambda_i^{\nu-\frac{1}{2}} e^{-\frac{\nu}{2}\lambda_i^{-2}} d\lambda_i = \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\Gamma(\frac{\nu}{2})\sqrt{\pi\nu w^2}} \left(1 + \frac{1}{\nu}\left(\frac{y_i-\mu}{w}\right)^2\right)^{-\frac{\nu+1}{2}}$$

$$\begin{array}{rcl} y_i|\lambda_i & \sim & \mathcal{N}(X\beta, \sigma^2 = w^2\lambda_i^2) \\ \lambda_i^{-2} & \sim & \text{Gamma}(\alpha = \nu/2, \beta = \nu/2) \end{array} \quad \Longrightarrow \quad y_i \sim t_\nu(\mu, w^2)$$

1. Return to your kaggle.com regression data set; or, find another data set; and use the above specification to perform a robust regression analysis in `PyMC`
2. Use the posterior distributions of the λ_i's to identify "outlier" (and potentially "influential") data points
3. [Optional] Assess the performance of the MCMC and any issues or warnings in the standard manner
4. [Optional] Perform ***Multiple Linear Regression*** diagnostics... residual plots, etc.