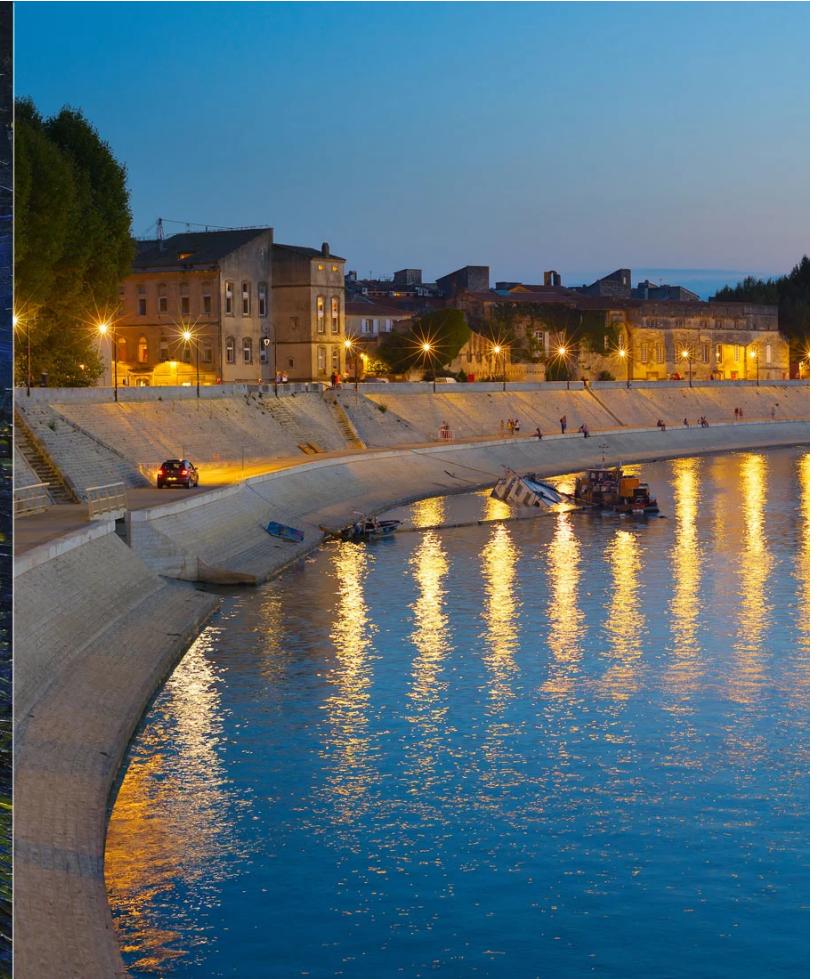
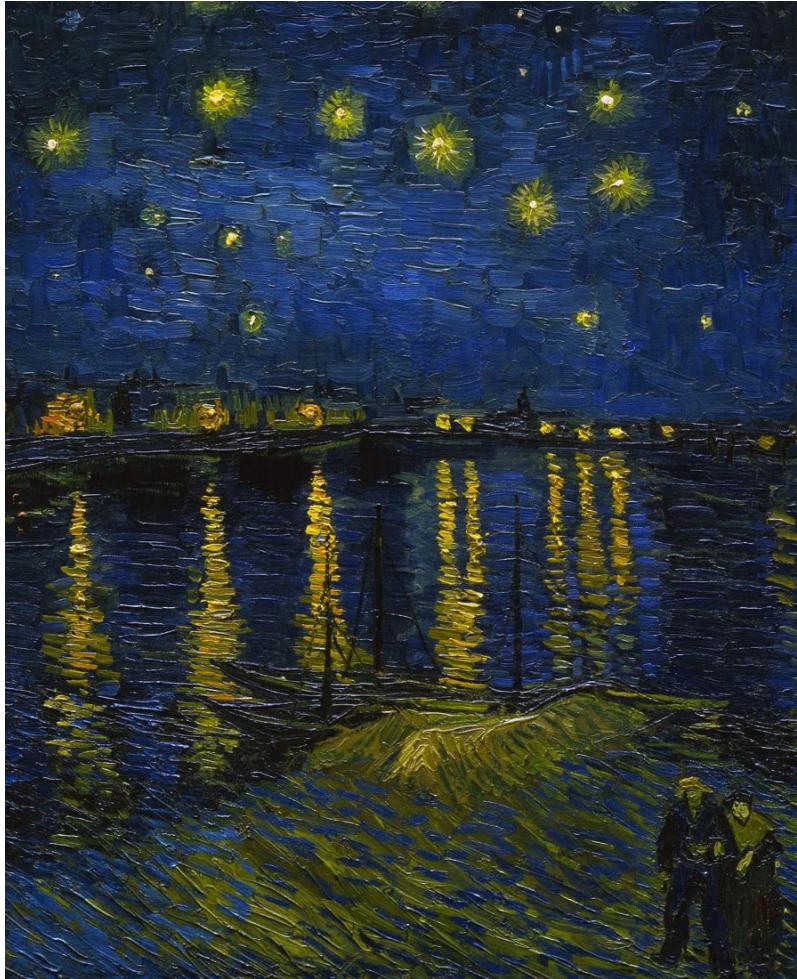


COMPARACIÓN DE MODELOS

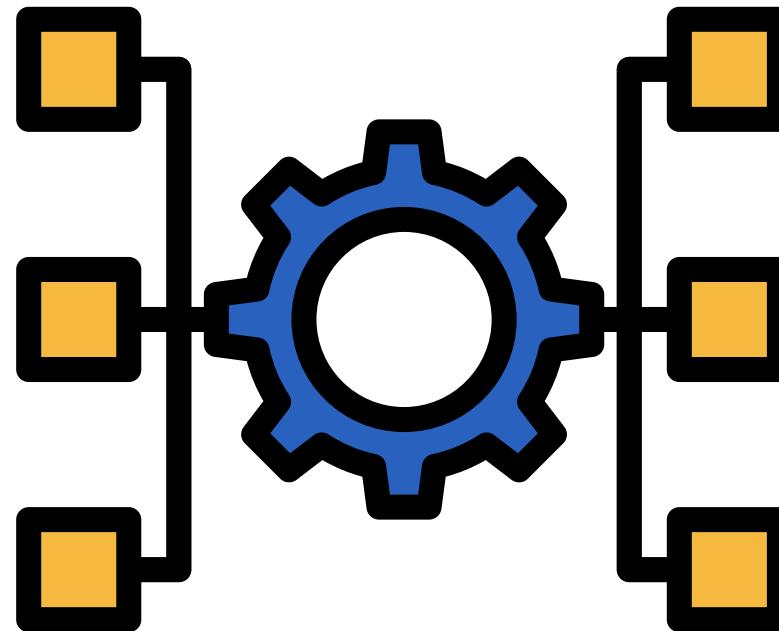
Santiago Alonso-Díaz, PhD

Universidad Javeriana

¿Cuál modelo representa mejor la realidad?



MODELOS JERÁRQUICOS



Surang at flaticon.com

Principales fuentes: Gelman, et al (2013); Kruschke (2014).

Hay muchas estructuras jerarquicas:

- Probabilidad de supervivencia COVID
 - Depende de país.
 - Depende del hospital.
- Actividad M1 en decisiones económicas (motor cortex)
 - Depende de SMA (supplementary motor area)
 - Depende de IPS (intraparietal sulcus; acum. evidencia)
 - Depende de vmPFC (ventromedial prefrontal cortex; valor subjetivo)

PRIMER EJEMPLO: RIESGO DE TUMOR EN RATAS

Descripción: determinar θ , la probabilidad de tumor en ratas de laboratorio (F344) que no reciben medicamento experimental.

Datos: 1 experimento que se acaba de hacer. 70 grupos de ratas testeadas antes.

Este ejemplo se va a modelar de forma jerárquica y con priors basados en data anterior.

```
In [2]: #Data  
rats = pd.DataFrame(  
    np.loadtxt("data/5_CB/rats.asc", skiprows=1),  
    columns = ['y', 'n'])  
rats.head() #y: with tumor; n: total rats
```

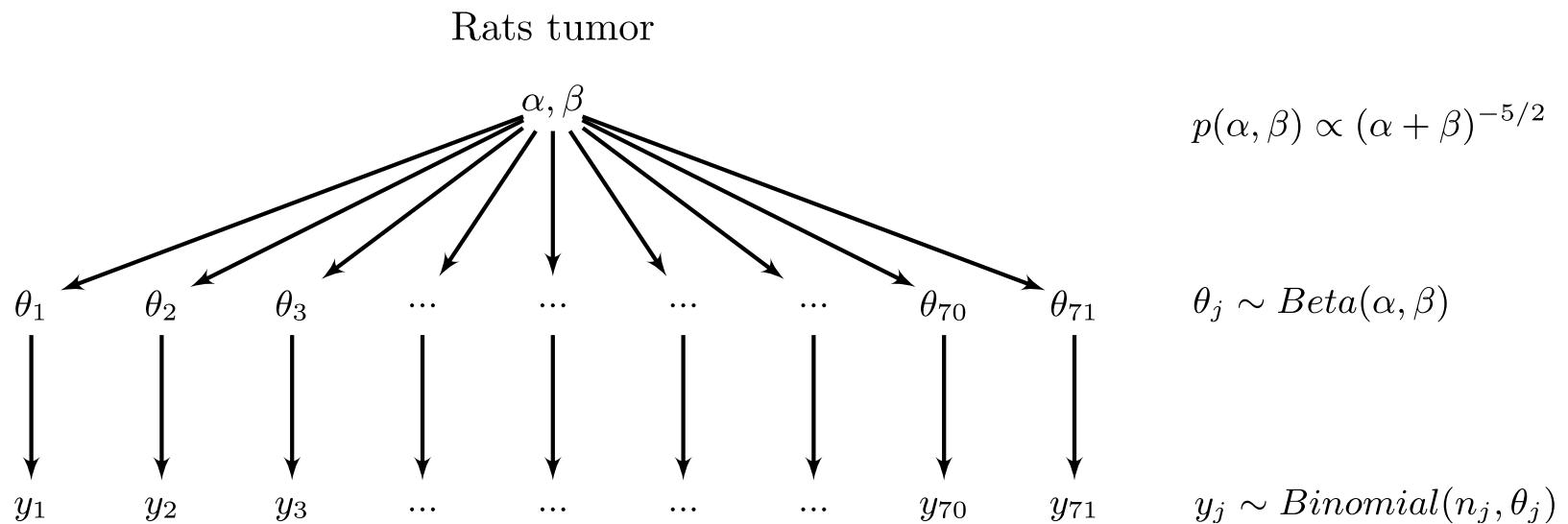
Out[2]:

| | y | n |
|---|-----|------|
| 0 | 0.0 | 20.0 |
| 1 | 0.0 | 20.0 |
| 2 | 0.0 | 20.0 |
| 3 | 0.0 | 20.0 |
| 4 | 0.0 | 20.0 |

71 experimentos con j grupos de ratas.

θ_j depende de hiperparámetros α, β :

Hiperparámetros reflejan condiciones del grupo, experimento, o día.



Las distribuciones para y_j & θ_j :

- Beta esta entre [0,1], perfecto para proporciones.
- Binomial, es discreta para conteos.

La distribución $p(\alpha, \beta)$:

- Uniforme entre

$$\left[\frac{\alpha}{\alpha + \beta}, (\alpha + \beta)^{-1/2} \right]$$

- El 1er extremo es el promedio de θ_j
- El 2do extremo depende del denominador del promedio de θ_j : $\alpha + \beta$
- $(\alpha + \beta)^{-5/2}$, es luego de volver a las variables originales (ver Gelman, et al, 2013, Hierarchical Models).

```
In [3]: # p(alpha,beta) en log (el algoritmo trabaja con logs)
def logp_ab(a,b):
    return tt.log(tt.pow(a+b, -5/2))
with pm.Model() as model:
    a = pm.Uniform('alpha', 0.05, 100)
    b = pm.Uniform('beta', 0.05, 100)
    pm.Potential('p(a, b)', logp_ab(a,b))
    #Pueden remover el Potential (es simila a increase_log de stan)
    #(pero ver Gelman, 2013, Ch 5, ed 3)
    #El algoritmo no falla si limitamos alpha y beta.

    theta = pm.Beta('theta', alpha=a, beta=b,
                    shape=rats.shape[0])

    y = pm.Binomial('y', p=theta, n=rats.loc[:, 'n'],
                    observed=rats.loc[:, 'y'])

    # Sampleo
    trace = pm.sample(5000, target_accept=.95)
    #the step size in PyMC3 is tuned automatically during warm up,
    #but we can coerce smaller step sizes by tweaking the configuration
    #of PyMC3's adaptation routine. In particular, we can increase the
    #target_accept parameter from its default value of 0.8 closer
    #to its maximum value of 1
    ##posterior_predictive = pm.sample_posterior_predictive(trace)
```

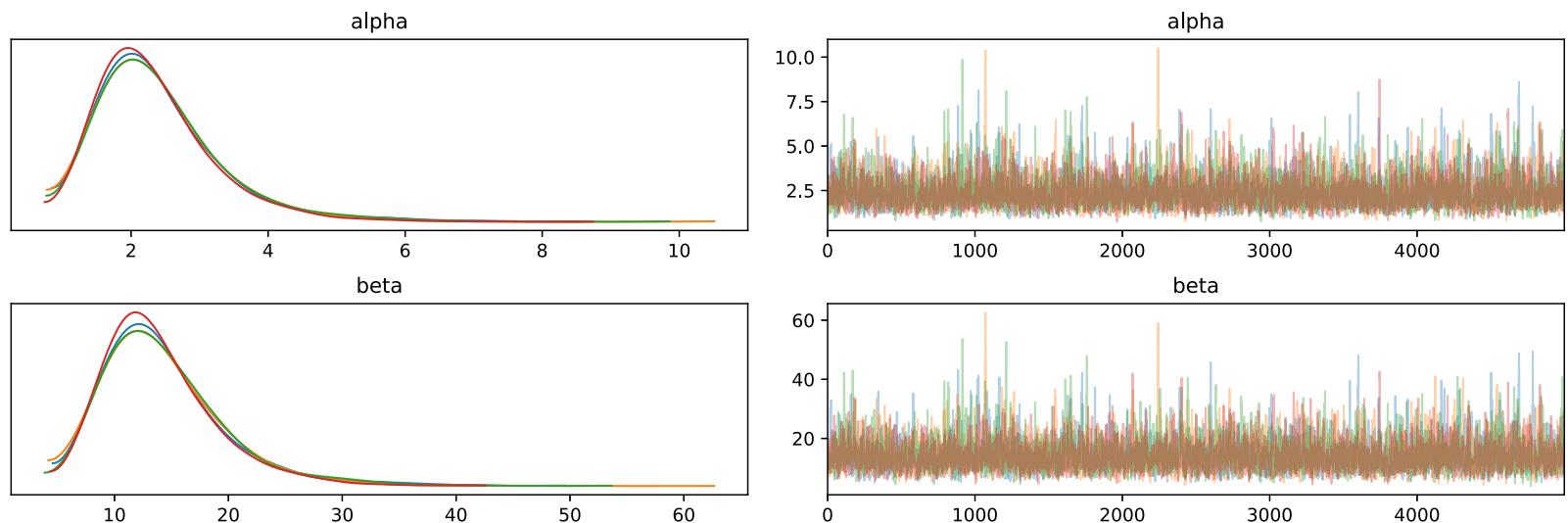
Auto-assigning NUTS sampler...
 Initializing NUTS using jitter+adapt_diag...
 Multiprocess sampling (4 chains in 4 jobs)
 NUTS: [theta, beta, alpha]

chains, 0 divergences]

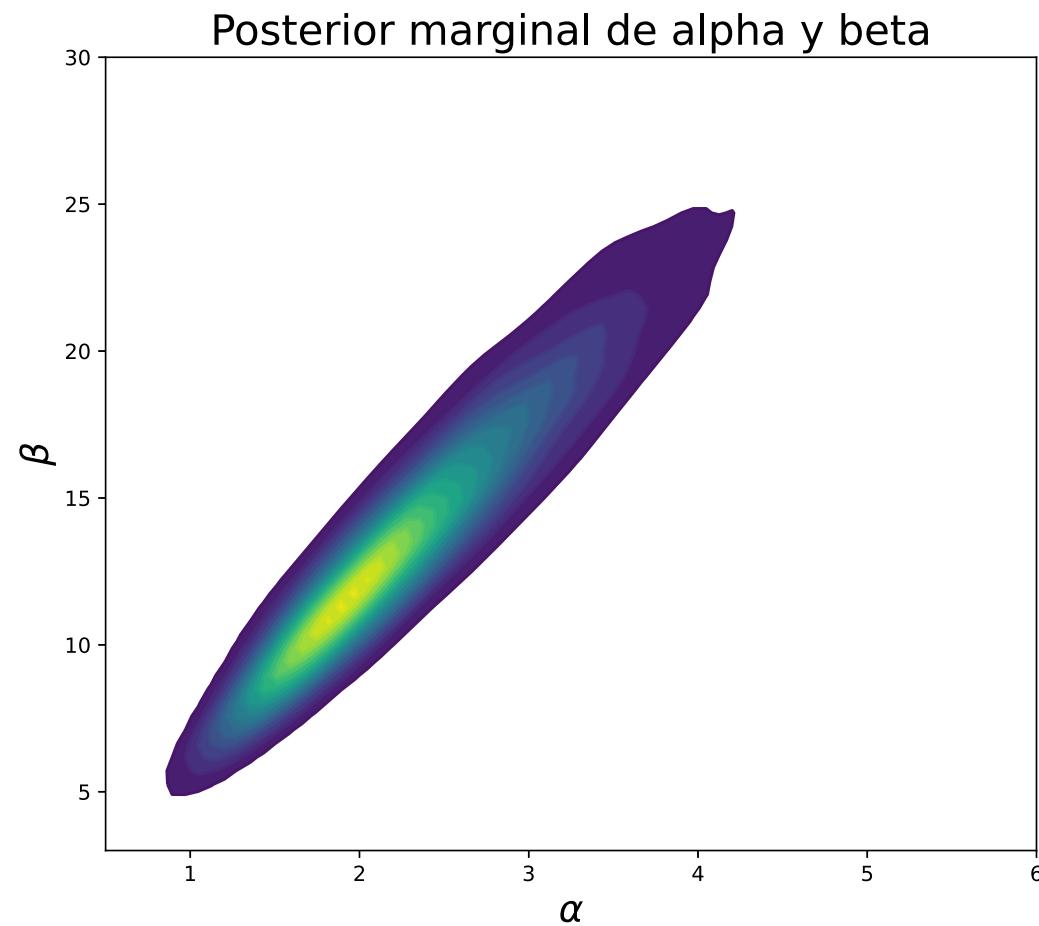
Sampling 4 chains for 1_000 tune and 5_000 draw iterations (4_000 + 20_000 draws total) took 44 seconds.

The number of effective samples is smaller than 25% for some parameters.

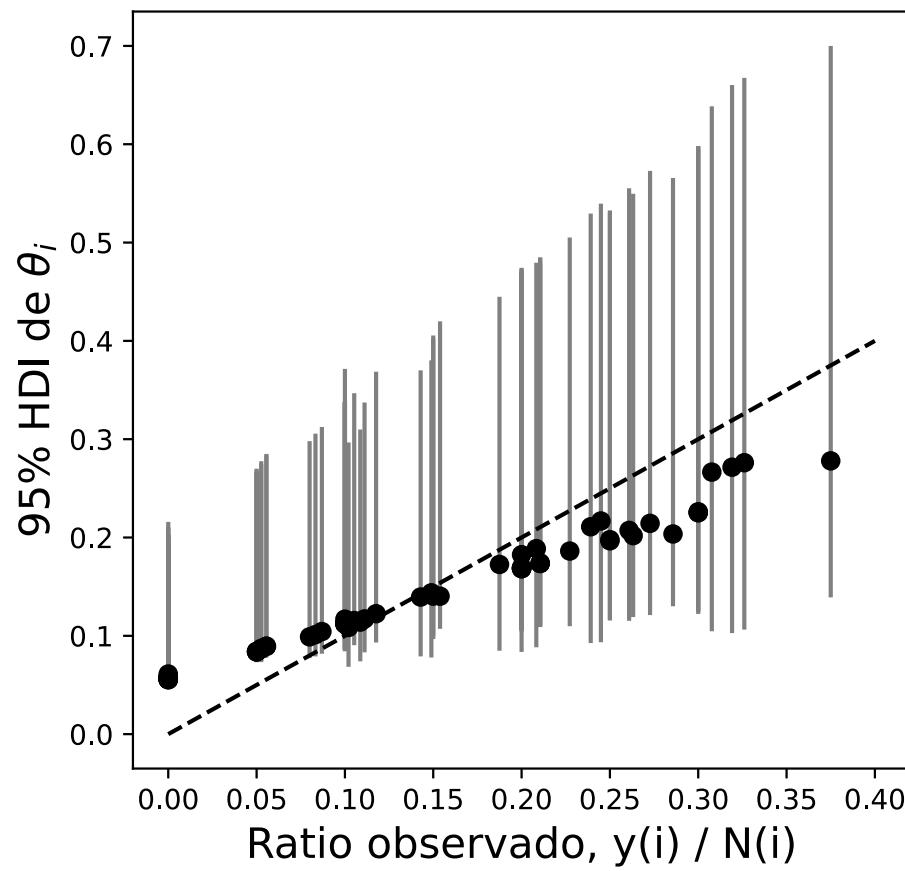
Combina bien



Posterior de los hiperparametros

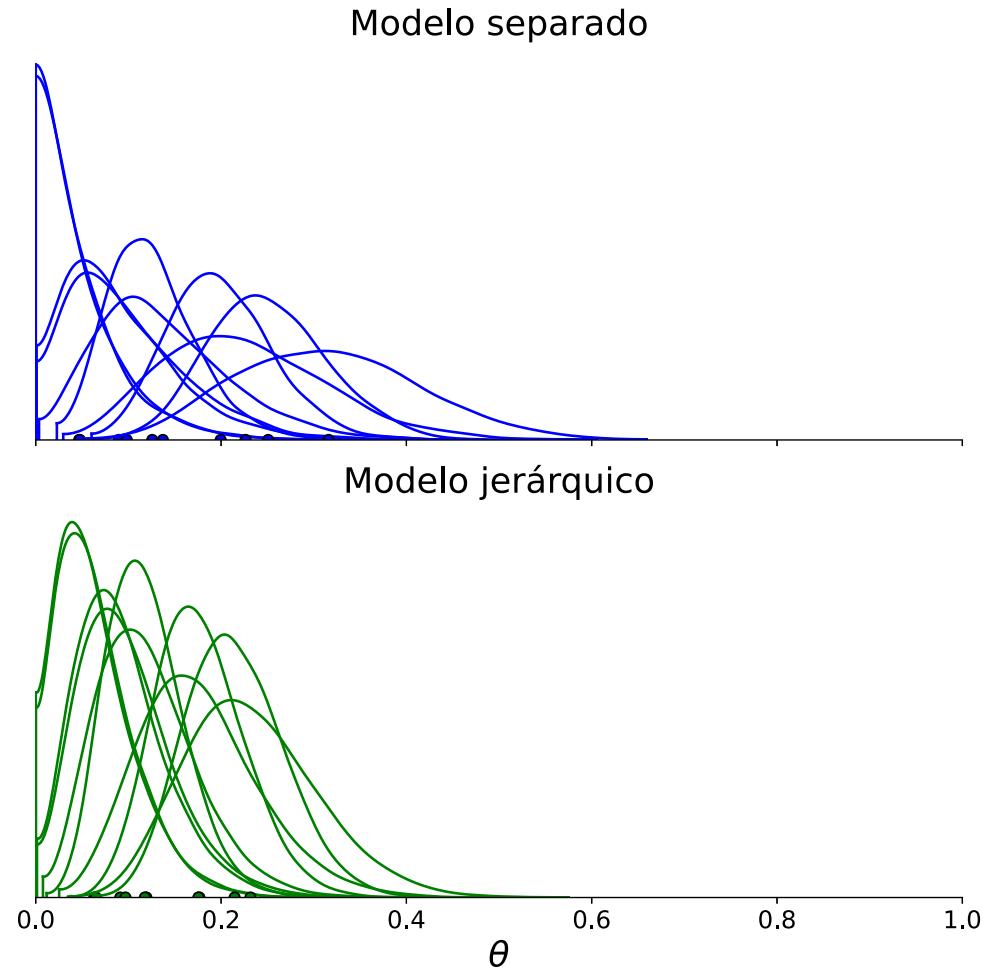


El modelo predice bien las proporciones (linea punteada modelo separado)



```
In [7]: # Comparación del modelo separado (cada experimento aparte)
# y el jerárquico.
x = np.linspace(0, 1, 250)
# binomial likelihood, beta (1,1) prior ,
# resulting posterior is beta(y+1,n-y+1)
# Por claridad, solo cada 7a distribución
theta_sep = st.beta.rvs(a = rats.iloc[0:-1:7,0] + 1,
                        b = rats.iloc[0:-1:7,1] - rats.iloc[0:-1:7,0] +
1,
                        size = [5000,rats.iloc[0:-1:7,0].shape[0]])
```

θ_j son más angostos que los del modelo separado. Modelo jeráquico tiene menos incertidumbre agrupandolos via α y β . Pero ¿cuál escoger? Veamos otro ejemplo primero.



SEGUNDO EJEMPLO: OCHO COLEGIOS

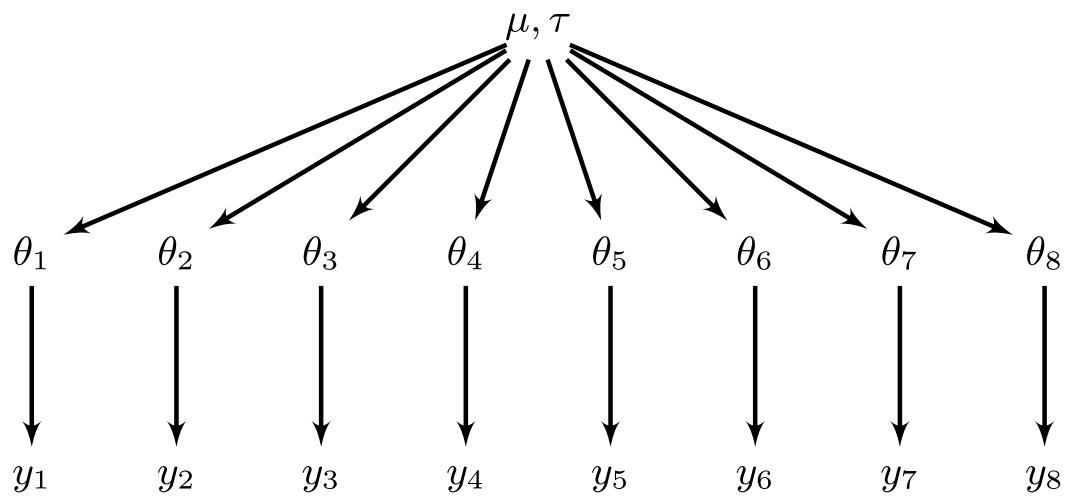
Problema: ¿programas de entrenamiento mejoran puntajes en la prueba de ingreso a la universidad?

Detalles: 8 colegios, cada uno con su propio programa de entrenamiento.

Data: aumento en puntaje (grupo control de cada colegio no hizo el entrenamiento)

Modelo jerárquico

8 schools



$$\begin{aligned}\mu &\sim Uniform(0, K_1) \\ \tau &\sim Uniform(0, K_2)\end{aligned}$$

$$\theta_j \sim Normal(\mu, \tau)$$

$$y_j \sim Normal(\theta_j, \sigma_{j \text{ data}}^2)$$

```
In [9]: # SAT-example data (SAT: Scholastic Aptitude Test)
# y is the estimated treatment effect
# s is the standard error of effect estimate
y = np.array([28, 8, -3, 7, -1, 1, 18, 12])
s = np.array([15, 10, 16, 11, 9, 11, 10, 18])
schools = pd.DataFrame({'Efecto':y, 'SE':s})
schools
```

Out[9]:

| | Efecto | SE |
|---|--------|----|
| 0 | 28 | 15 |
| 1 | 8 | 10 |
| 2 | -3 | 16 |
| 3 | 7 | 11 |
| 4 | -1 | 9 |
| 5 | 1 | 11 |
| 6 | 18 | 10 |
| 7 | 12 | 18 |

```
In [10]: with pm.Model() as model:
    mu = pm.Uniform('mu', 0, 30)
    tau = pm.Uniform('tau', 0, 30)

    theta = pm.Normal('theta', mu=mu, sigma=tau,
                      shape=schools.shape[0])

    y = pm.Normal('y', mu = theta, sigma = schools.loc[:, 'SE'],
                  observed = schools.loc[:, 'Efecto'])

    # Sampleo
    trace = pm.sample(5000, tune = 1000, target_accept = 0.99)
    posterior_predictive = pm.sample_posterior_predictive(trace)
```

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

 100.00% [24000/24000 00:52<00:00 Sampling 4
 chains, 54 divergences]

Sampling 4 chains for 1_000 tune and 5_000 draw iterations (4_000 + 20_000 draws total) took 53 seconds.

There were 15 divergences after tuning. Increase `target_accept` or reparameterize.

There were 11 divergences after tuning. Increase `target_accept` or reparameterize.

There were 14 divergences after tuning. Increase `target_accept` or reparameterize.

The acceptance probability does not match the target. It is 0.9688440056834188, but should be close to 0.99. Try to increase the number of tuning steps.

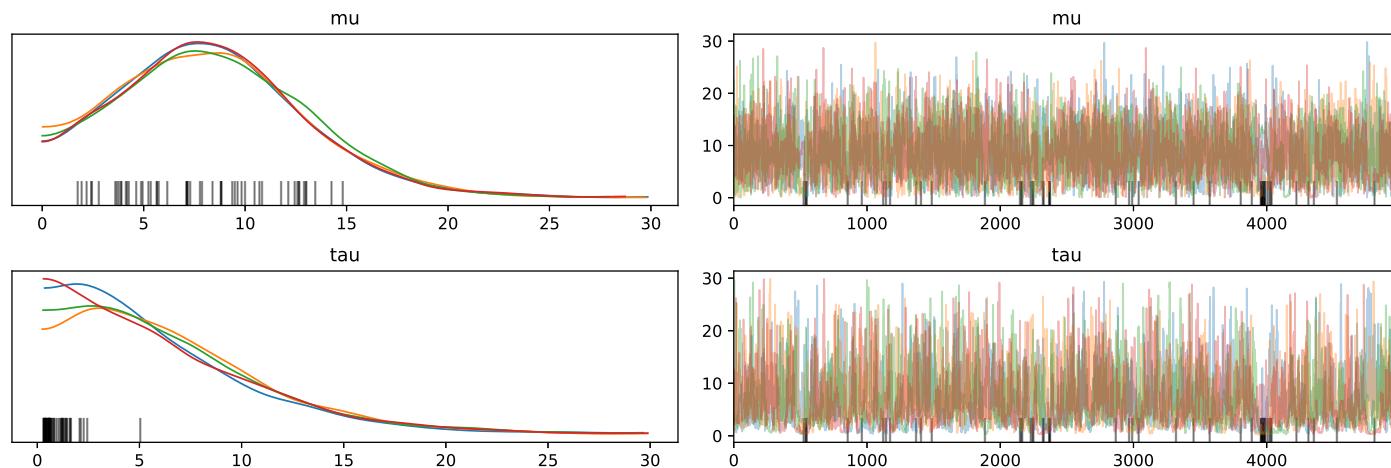
There were 14 divergences after tuning. Increase `target_accept` or reparameterize.

The acceptance probability does not match the target. It is 0.962770824741211

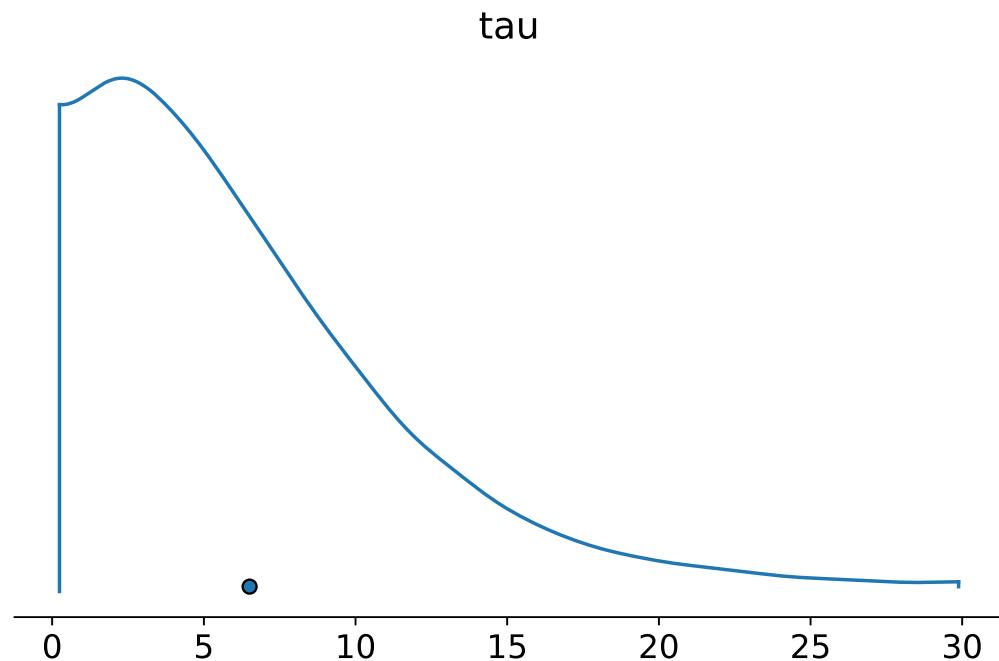
7, but should be close to 0.99. Try to increase the number of tuning steps.
The number of effective samples is smaller than 10% for some parameters.

100.00% [20000/20000 00:09<00:00]

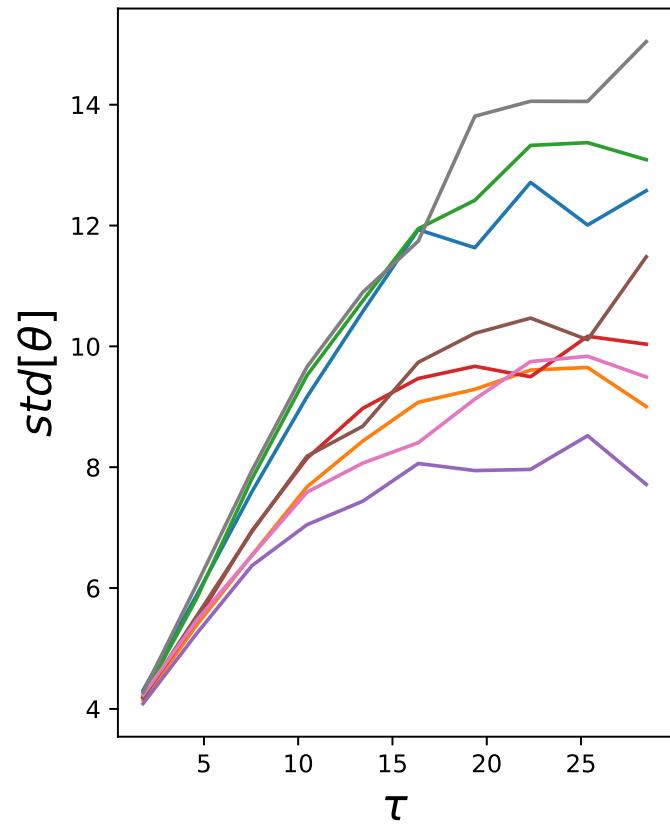
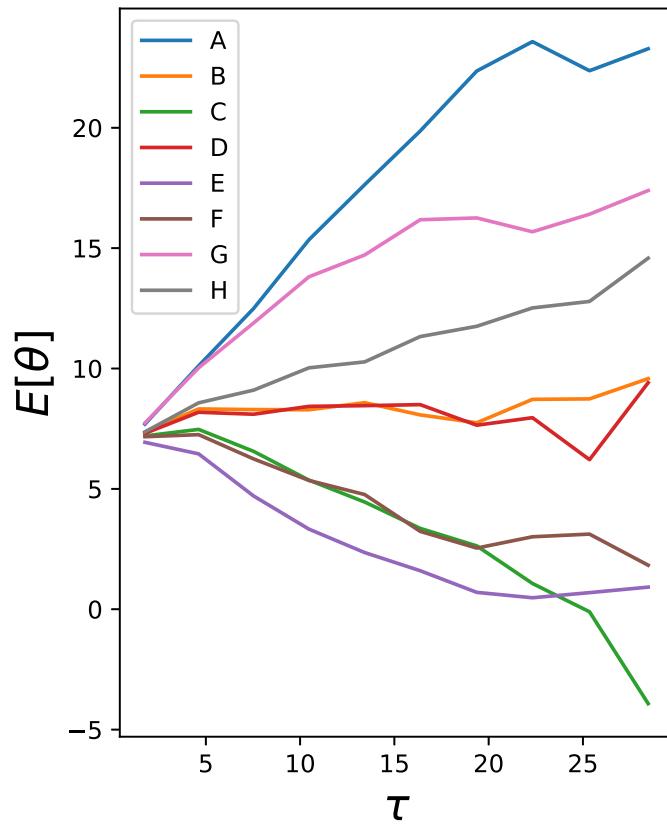
Combina bien



El proceso latente de donde salen los efectos del promedio
entrenamiento (μ) no es muy ruidoso



Un efecto tan fuerte como el de la escuela A (28 puntos de incremento) parece improbable, aún con hiperparametro τ alto

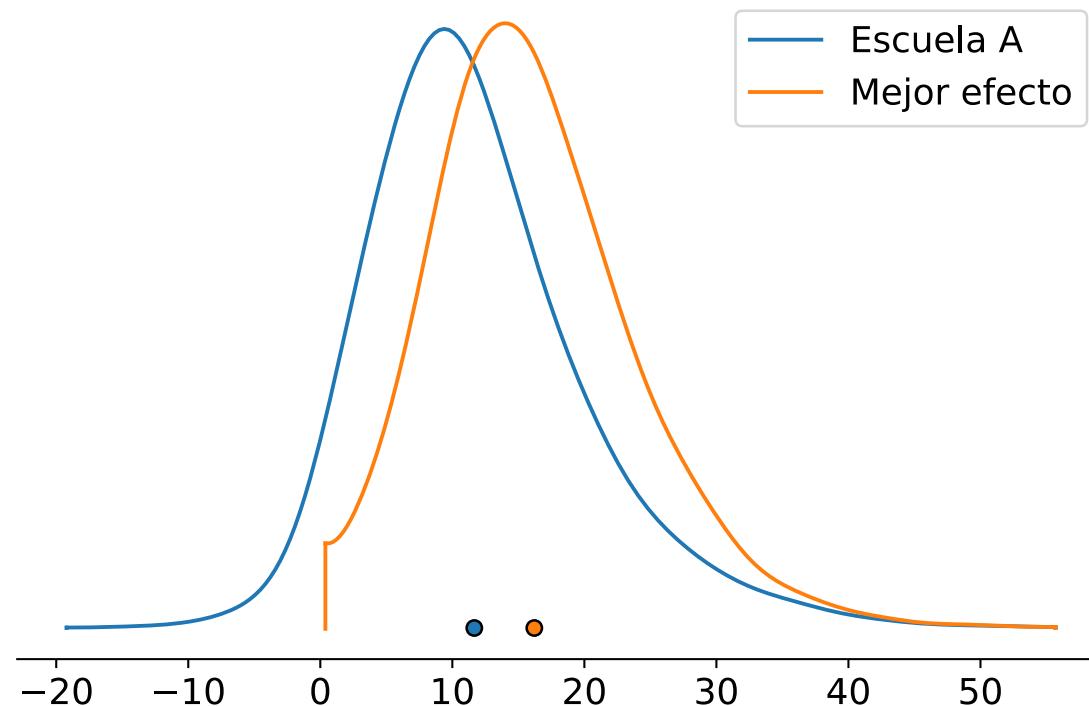


```
In [14]: az.summary(data)
```

Out[14]:

| | mean | sd | hdi_3% | hdi_97% | mcse_mean | mcse_sd |
|----------|--------|-------|--------|---------|-----------|---------|
| theta[0] | 11.662 | 8.216 | -2.097 | 27.994 | 0.149 | 0.105 |
| theta[1] | 8.031 | 6.174 | -3.531 | 20.165 | 0.079 | 0.059 |
| theta[2] | 6.444 | 7.478 | -7.796 | 21.110 | 0.087 | 0.068 |
| theta[3] | 7.945 | 6.436 | -4.240 | 20.383 | 0.081 | 0.060 |
| theta[4] | 5.293 | 6.074 | -6.982 | 16.342 | 0.089 | 0.063 |
| theta[5] | 6.377 | 6.566 | -6.495 | 18.873 | 0.083 | 0.060 |
| theta[6] | 10.840 | 6.673 | -0.279 | 24.326 | 0.130 | 0.092 |
| theta[7] | 8.805 | 7.620 | -5.063 | 24.165 | 0.099 | 0.077 |
| mu | 8.417 | 4.552 | 0.001 | 15.884 | 0.084 | 0.059 |
| tau | 6.508 | 4.948 | 0.241 | 15.460 | 0.150 | 0.106 |

El programa de la escuela A es bueno, pero parece que no hay evidencia que sea siempre el mejor



In [16]:

```
# Probabilidad que el programa de la escuela A  
# sea mejor que el del peor en los datos  
conteo = np.sum(trace['theta'][:,0]>trace['theta'][:,2])  
prob = conteo/trace['theta'].shape[0]  
print("Probabilidad que el programa de la escuela A sea mejor que d  
e la escuela C: ", np.round(prob,2))
```

Probabilidad que el programa de la escuela A sea mejor que de la escuela C:
0.68

EJERCICIOS (FUENTE: GELMAN, ET AL, 3ERA ED.)

1. Utilicé las muestras del ejemplo de ocho colegios:

- Para cada escuela, calcule la probabilidad que su programa es el mejor de todos.
- Para cada par de escuelas j, k calcule la probabilidad que el programa j es mejor que el programa k .

In []: *#Escriba su código acá*

1. Repita 1 con τ fijo y muy grande (esto es equivalente a una estimación separada por colegio). Discuta las diferencias con los resultados del punto 1.

In []: *#Escriba su código acá*

1. Utilice la data de las ratas pero ahora trabaje en la escala log-odds. Para ello, tome muestras de $\text{logit}(\theta_j) \sim \text{Normal}(\mu, \tau)$, donde τ es la desviación estandar. Tip: calcule el logit de μ y τ . Interprete los resultados en esa escala.

In []: #Escriba su código acá

1. En los alrededores del campus de una universidad se hizo un conteo de tráfico de bicicletas y otros vehículos. Sesenta bloques se seleccionaron y se observaron por una hora. Los bloques se escogieron acorde a su nivel de actividad (bien ocupado, ocupación normal, y residencial) y si tenían bicirutitas o no. La data la puede encontrar en transit.csv. Centrese en las 10 primeras filas. Haga lo siguiente:

- Un modelo jerárquico como el del tumor de las ratas. Ahora n es el total de vehículos (bicis+otros) y la variable de interés es la proporción de bicis. Es decir, infiera θ_j , la proporción latente de bicicletas de cada bloque, con la data disponible (solo las 10 primeras filas).
- Compare la posterior del parámetro θ_j con las proporciones que hay en la data en cada bloque (# de bicicletas / # vehículos incluyendo bicicletas). Comente cómo difieren (o no) la proporciones de la data y la inferencia de la posterior del parámetro θ_j
- Calcule el intervalo posterior al 95% de la proporción de tráfico de bicicletas en todos los bloques. Agrupe todos los θ_j , queremos un intervalo general.
- Corra el modelo ahora con los siguientes priors y likelihood:
 - Likelihood para el conteo de bicicletas es una $Poisson(\theta_j)$. Ahora θ_j es el número promedio de bicis por hora.

- Prior $\text{Gamma}(\alpha, \beta)$ para θ_j . Asigne uniformes para los hiperparámetros α y β
- Haga un gráfica con los hiperparámetros (eje x: α , eje y: β)

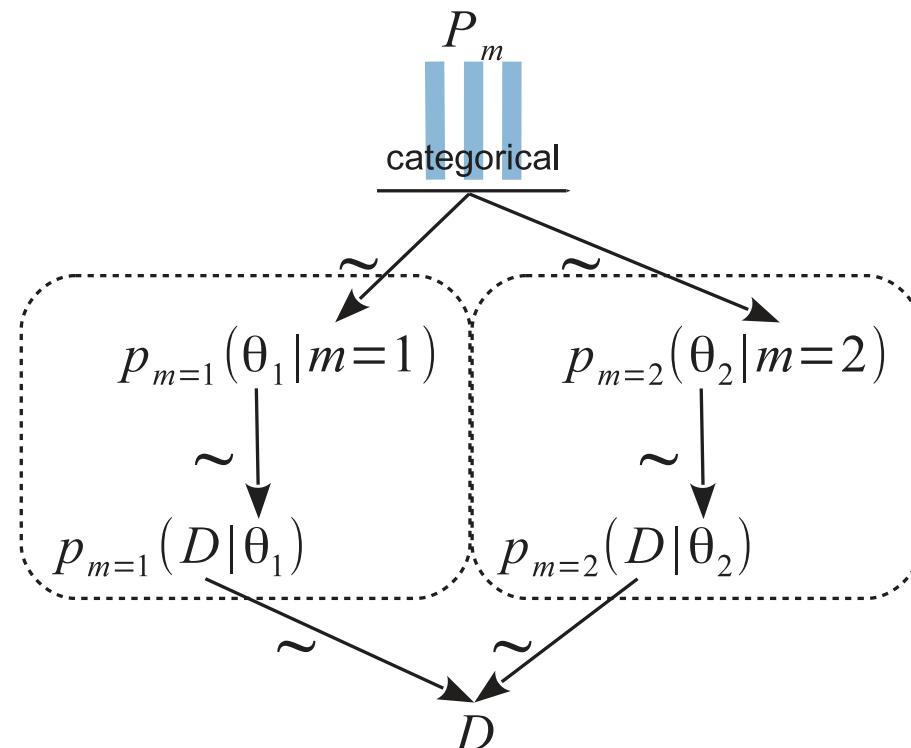
In []: *#Escriba su código acá*

COMPARAR MODELOS

En la anterior sección comparamos datos vs. posterior (ver ejemplo ratas).

Vamos a ver más técnicas/aproximaciones al problema de escoger el mejor modelo.

Una aproximación es pensar el problema como uno jerárquico. Dos modelos (m) con diferentes priors y likelihoods. La distribución categórica al tope de la jerárquía tiene hiperparámetros. La inferencia los determina.



Fuente: Kruschke (2014)

Una empresa que fabrica monedas tiene dos sitios de producción.

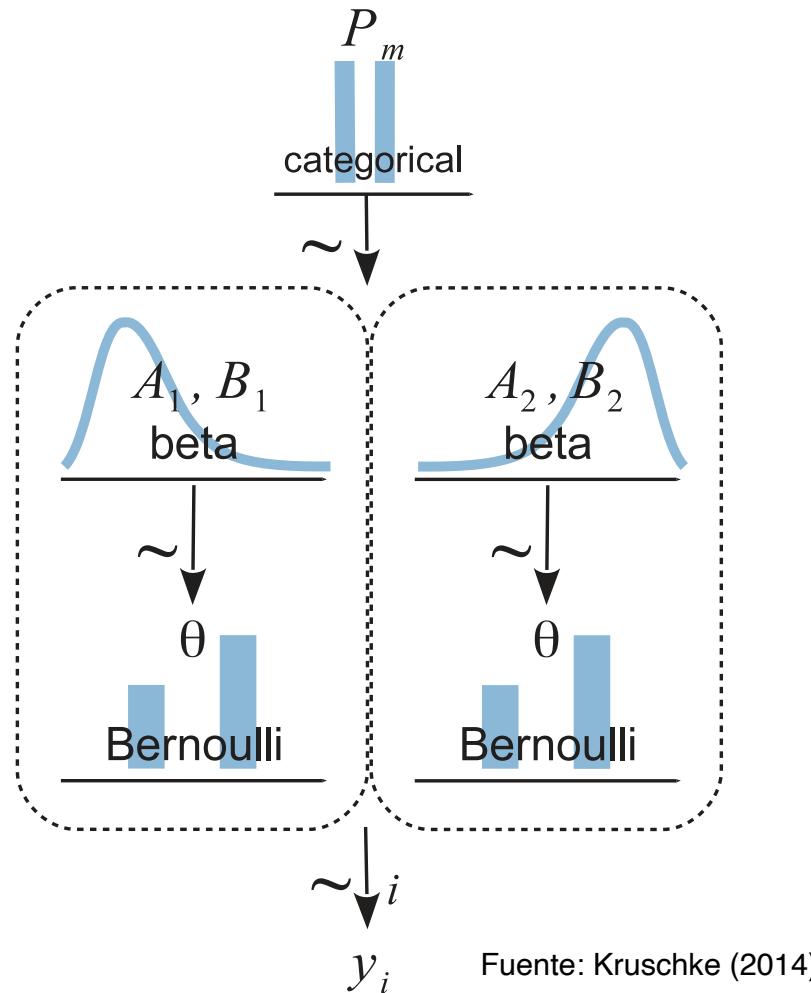
Sitio 1: monedas sesgadas a cara e.g. beta(9,4)

Sitio 2: monedas sesgadas a sello e.g. beta(4,9)



El problema en representación jerárquica.

Buscamos las prob. posteriores de $m=1, m=2$ dado los lanzamientos de la moneda y_i



```
In [3]: #Data  
N=9  
z=6  
y_o = [ 0 ]*(N-z) + [ 1 ]*z  
y_o
```

```
Out[3]: [0, 0, 0, 1, 1, 1, 1, 1]
```

```
In [4]: # Models to compare
m1_info = np.array([3,9])
m2_info = np.array([9,3])
# Inference
with pm.Model() as model:
    #Prior for m (model)
    mi = pm.DiscreteUniform('model_index', lower=0, upper=1)

    #Samples from the models to test
    thetaM1 = pm.Beta('thetaM1', m1_info[0], m1_info[1] )
    thetaM2 = pm.Beta('thetaM2', m2_info[0], m2_info[1] )

    #Likelihood
    theta = pm.Deterministic('theta', (1-mi)*thetaM1 + (mi)*thetaM2)
    y = pm.Bernoulli('y', theta, observed = y_o)

    #Muestras
    start = pm.find_MAP()
    step = pm.Metropolis()
    trace = pm.sample(10000, step = step, start = start)
    burned_trace = trace[5000:] #burn in ( para este ejemplo, grande
    por conveniencia: graficas con poco MB).
```

100.00% [7/7 00:00<00:00 logp = -7.6129, ||grad|| =
3.2882]

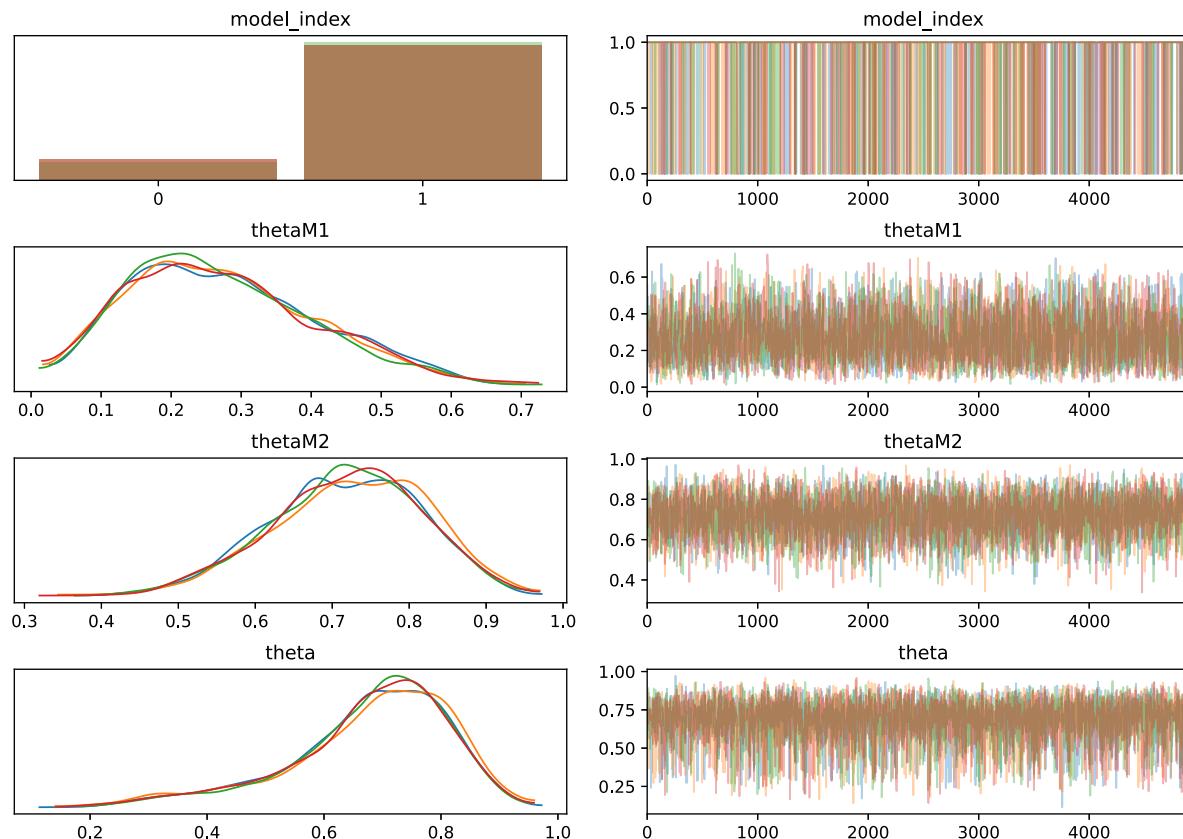
Multiprocess sampling (4 chains in 4 jobs)
CompoundStep
>Metropolis: [thetaM2]
>Metropolis: [thetaM1]
>Metropolis: [model_index]

100.00% [44000/44000 00:09<00:00 Sampling 4

chains, 0 divergences]

Sampling 4 chains for 1_000 tune and 10_000 draw iterations (4_000 + 40_000 draws total) took 10 seconds.

The number of effective samples is smaller than 25% for some parameters.

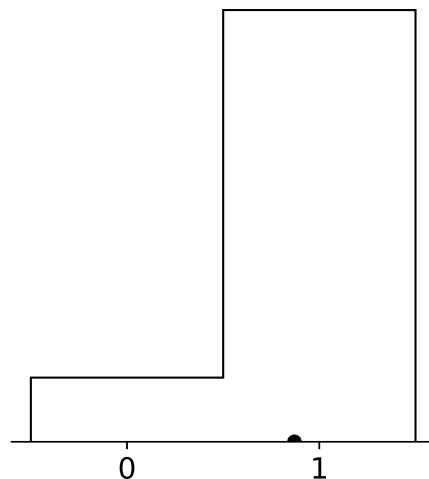


```
In [7]: print("p(m=2|D): ", np.round(burned_trace[ 'model_index' ].mean(),4))  
print("p(theta|D): ", np.round(burned_trace[ 'theta' ].mean(),4))  
print("Prop. data: ", np.round(z/N,4)) #Data
```

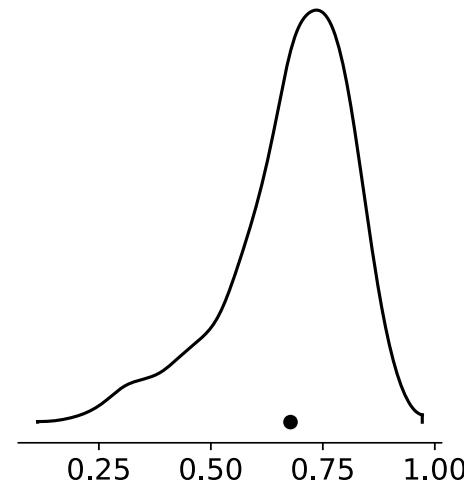
p(m=2|D): 0.8706
p(theta|D): 0.678
Prop. data: 0.6667

El posterior del indice sugiere que el modelo 2 es mejor para la data.

Indice Modelos



Prop. inferida: θ



Asumimos priors diferentes (θ_{M1}, θ_{M2}), y el mismo likelihood.

Es perfectamente valido que los modelos también tengan diferentes likelihoods.

Vemos como incluirlo en nuestro modelo.

```
In [8]: # No vamos a correr el modelo, pues no tiene sentido.  
# Es solo para ver cómo se puede considerar diferentes  
# likelihoods  
with model:  
    #pm.math.eq(A, B) checks if A==B.  
    theta_cond = pm.math.switch(pm.math.eq(mi, 0),  
                                theta, #1er likelihood  
                                2*theta #2do likelihood  
                                )  
    y_cond = pm.Bernoulli('y_cond', theta_cond, observed = y_o)
```

BAYES FACTOR

Una métrica propuesta para cuantificar la ventaja de un modelo sobre otro es con Bayes factor. La definición es

$$BF = \frac{p(Data|Modelo_1)}{p(Data|Modelo_2)}$$

Otra formulación equivalente es (no la vamos a demostrar):

$$\begin{aligned} BF &= \frac{Post_{odds}}{Prior_{odds}} \\ &= \frac{\frac{p(Modelo_1|Data)}{p(Modelo_2|Data)}}{\frac{p(Modelo_1)}{p(Modelo_2)}} \end{aligned}$$

```
In [9]: # Bayes factor para el ejemplo de la moneda
p_M2 = burned_trace['model_index'].mean() # i.e. la tasa de muestras
de M2
Post_odds = (p_M2)/(1-p_M2)
Prior_odds = 0.5/0.5
BF = Post_odds / Prior_odds # Bayes Factor a favor de M2
BF
```

Out[9]: 6.727975270479137

Esta vez fue fácil de calcular pero el Bayes Factor requiere computación intensiva pues el marginal likelihood son integrales multiples sobre muchos parámetros.

Veamos otra forma de cuantificar ventajas de modelos.

INFORMATION CRITERION (IC)

Basado en [PyMC website](#)
(https://docs.pymc.io/notebooks/model_comparison.html).

Los information criterion (IC) son medidas para comparar modelos. Hay varios, por ejemplo, BIC, AIC, WAIC. Todos se basan en una función del likelihood llamada deviance ([Gelman, et al
\(\[http://www.stat.columbia.edu/~gelman/research/published/waic_ur\]\(http://www.stat.columbia.edu/~gelman/research/published/waic_ur\)](http://www.stat.columbia.edu/~gelman/research/published/waic_ur)

IMPORTANTE: Modelos con IC bajos explican mejor la data y.

Veamos WAIC. Esta es la formula:

$$WAIC = -2lppd + 2p_{waic}$$

$lppd$ es log pointwise predictive density (i.e. likelihood). En la práctica usamos las S muestras del sampleador θ^s . n es número de datos:

$$\text{computed } lppd = \sum_{i=1}^n \log \left(\frac{1}{S} \sum_{s=1}^S p(y_i | \theta^s) \right)$$

p_{waic} es parámetros efectivos, un termino para penalizar modelos complejos:

$$p_{waic} = \sum_{i=1}^n V_s^S (\log(p(y_i | \theta^s)))$$

V es varianza

Volvamos a la data de las 8 escuelas.

Objetivo: inferir los efectos de entrenamiento en puntajes de ingreso a la universidad.

Modelos:

- Pooled: UN solo efecto compartido para las 8
- Hierarchical: Efectos diferentes para cada escuela

Data: 8 efectos/ganancias en puntajes y desviación estándar (σ)

```
In [2]: J = 8  
y = np.array([28, 8, -3, 7, -1, 1, 18, 12])  
sigma = np.array([15, 10, 16, 11, 9, 11, 10, 18])
```

```
In [3]: with pm.Model() as pooled:  
    mu = pm.Normal('mu', 0, sd=1e6)  
    obs = pm.Normal('obs', mu, sd=sigma, observed=y)  
    trace_p = pm.sample(2000)  
  
with pm.Model() as hierarchical:  
    eta = pm.Normal('eta', 0, 1, shape=J)  
    mu = pm.Normal('mu', 0, sd=1e6)  
    tau = pm.HalfCauchy('tau', 5)  
    theta = pm.Deterministic('theta', mu + tau*eta)  
    obs = pm.Normal('obs', theta, sd=sigma, observed=y)  
    trace_h = pm.sample(2000, target_accept=0.99)
```

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

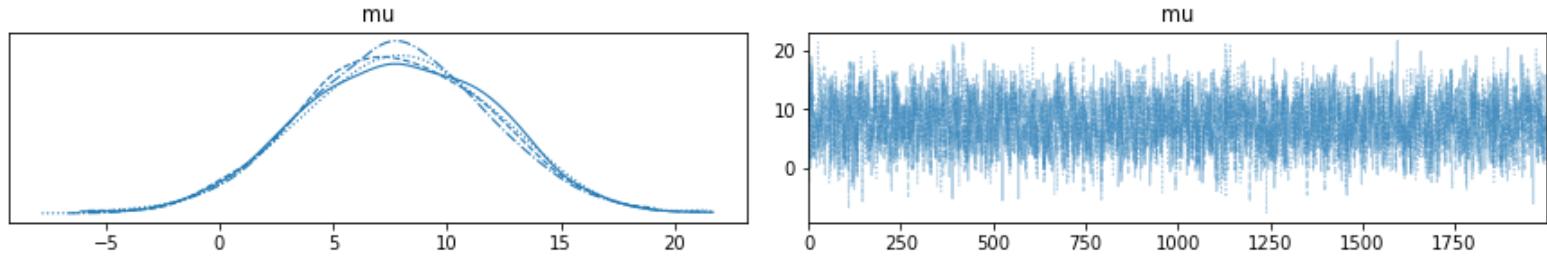
[0%| | 0.00/12000 00:02<00:00 Sampling 4
chains, 0 divergences]

Sampling 4 chains for 1_000 tune and 2_000 draw iterations (4_000 + 8_000 draw
s total) took 3 seconds.
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [tau, mu, eta]

[0%| | 0.00/12000 00:08<00:00 Sampling 4
chains, 0 divergences]

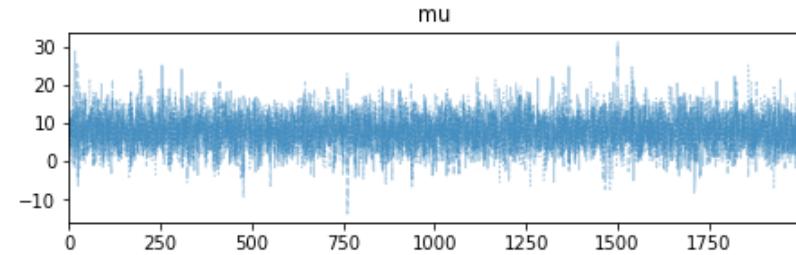
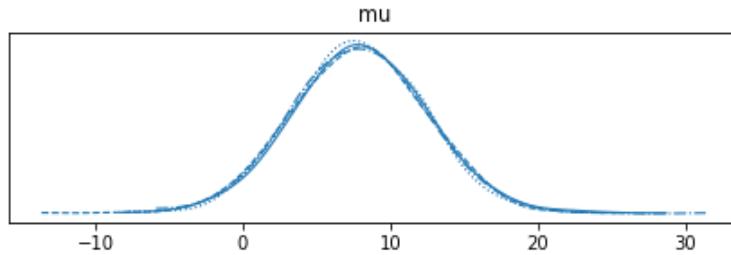
Sampling 4 chains for 1_000 tune and 2_000 draw iterations (4_000 + 8_000 draw
s total) took 9 seconds.

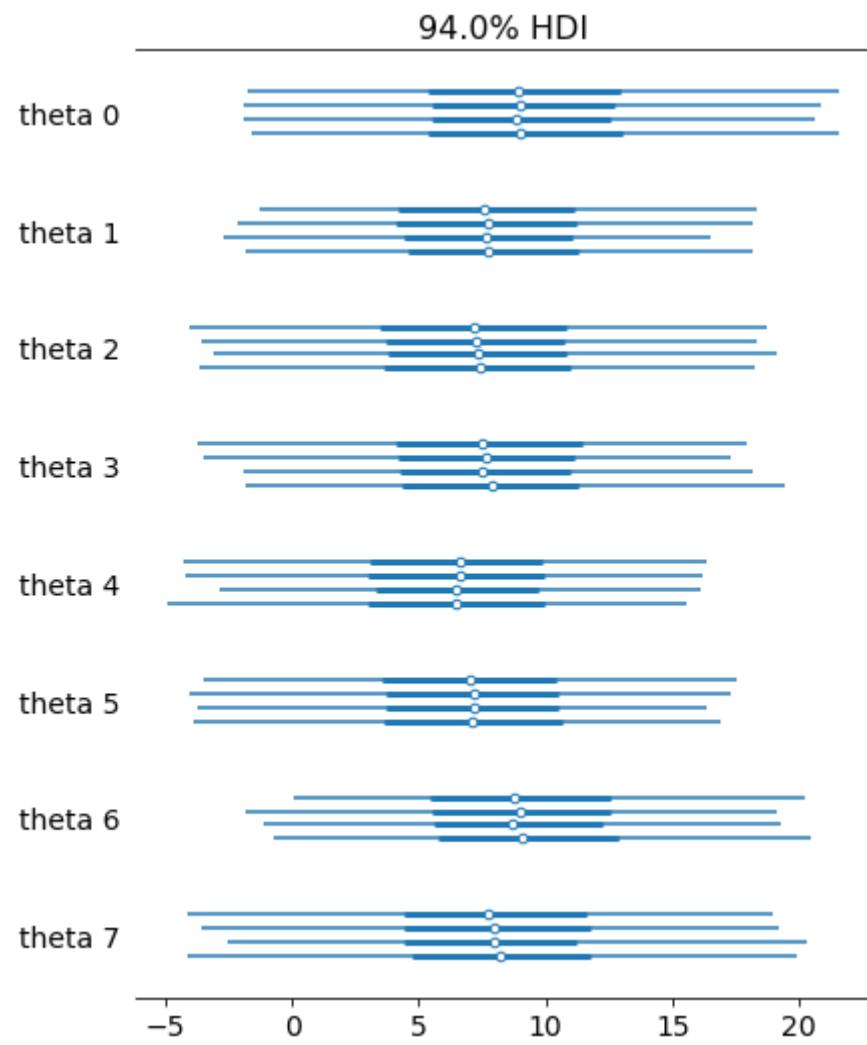
```
In [4]: with pooled:  
    pm.traceplot(trace_p, var_names=[ 'mu' ]);
```



```
In [5]: with hierarchical:
```

```
    pm.traceplot(trace_h, var_names=[ 'mu' ]);  
    pm.forestplot(trace_h, var_names=[ 'theta' ]);
```





Podemos calcular WAIC con PyMC

Para comparar WAIC de diferentes modelos. También loo.

```
In [7]: with pooled:  
    df_comp_WAIC = pm.compare({'hierarchical': trace_h,  
                               'pooled': trace_p}, ic = 'waic')  
    df_comp_loo = pm.compare({'hierarchical': trace_h,  
                               'pooled': trace_p}, ic = 'loo')
```

```
/opt/anaconda3/lib/python3.7/site-packages/arviz/stats/stats.py:151: UserWarning:  
The scale is now log by default. Use 'scale' argument or 'stats.ic_scale' rcParam if you rely on a specific value.  
A higher log-score (or a lower deviance) indicates a model with better predictive accuracy.  
"\nThe scale is now log by default. Use 'scale' argument or "  
/opt/anaconda3/lib/python3.7/site-packages/arviz/stats/stats.py:683: UserWarning: Estimated shape parameter of Pareto distribution is greater than 0.7 for one or more samples. You should consider using a more robust model, this is because importance sampling is less likely to work well if the marginal posterior and LOO posterior are very different. This is more likely to happen with a non-robust model and highly influential observations.  
"Estimated shape parameter of Pareto distribution is greater than 0.7 for "
```

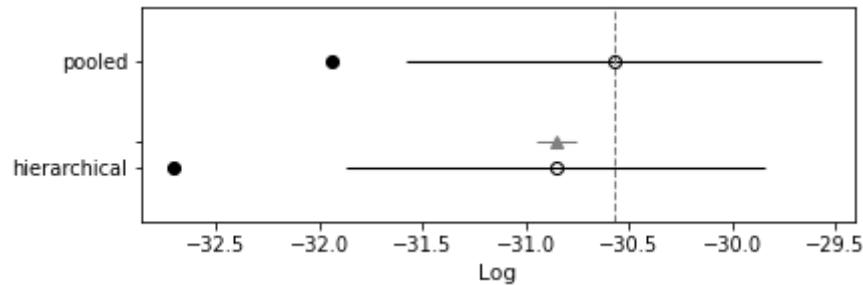
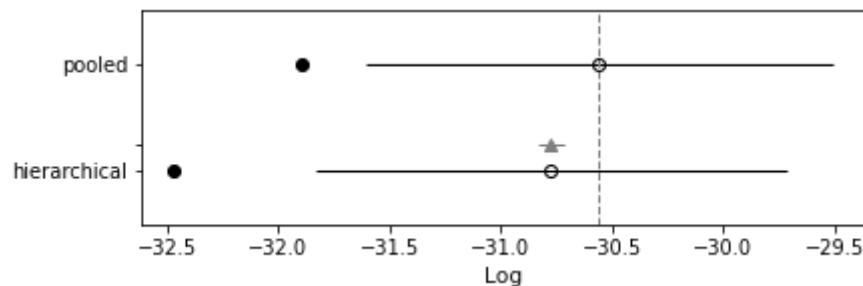
```
In [8]: display(df_comp_WAIC) # waic
display(df_comp_loo) # leave-one-out cross validation
#rank: order of models (worst to best); waic/loo: self-explained
#p_waic: eff. number of parameters; d_waic/d_loo: difference in
...
#se: of waic/loo; dse: of d_waic/d_loo; warning: should be false
```

| | rank | waic | p_waic | d_waic | weight | |
|---------------------|------|----------|----------|----------|----------|-----|
| pooled | 0 | -30.5587 | 0.668387 | 0 | 0.554159 | 1.0 |
| hierarchical | 1 | -30.7746 | 0.845831 | 0.215917 | 0.445841 | 1.0 |

| | rank | loo | p_loo | d_loo | weight | |
|---------------------|------|----------|----------|---------|----------|-----|
| pooled | 0 | -30.5726 | 0.682366 | 0 | 0.570608 | 1.0 |
| hierarchical | 1 | -30.8531 | 0.924314 | 0.28042 | 0.429392 | 1.0 |

```
In [9]: with pooled:
```

```
    pm.compareplot(df_comp_WAIC);  
    pm.compareplot(df_comp_loo);
```

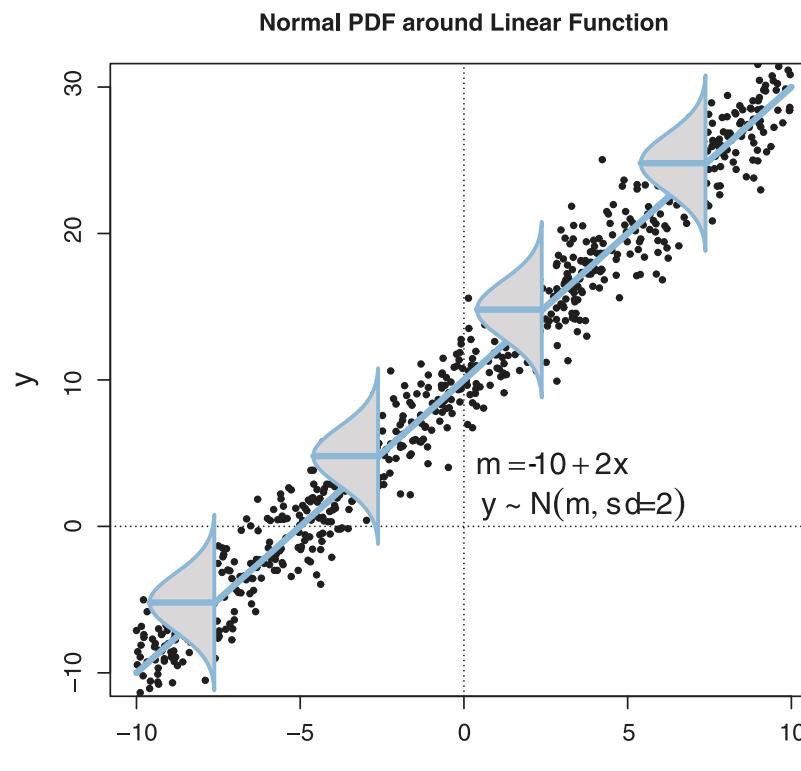


El modelo jerárquico no dominó al pooled. Ambos hacen un buen trabajo.

REGRESIONES

Basado en Gelman et al y en [Prasad Ostwal](#)
[\(https://ostwalprasad.github.io/machine-learning/Bayesian-Linear-Regression-using-PyMC3.html\)](https://ostwalprasad.github.io/machine-learning/Bayesian-Linear-Regression-using-PyMC3.html)

En una regresión tradicional inferimos parámetros de una normal que cambia de promedio en función del valor de variables independientes X:



Fuente: Kruschke (2015)

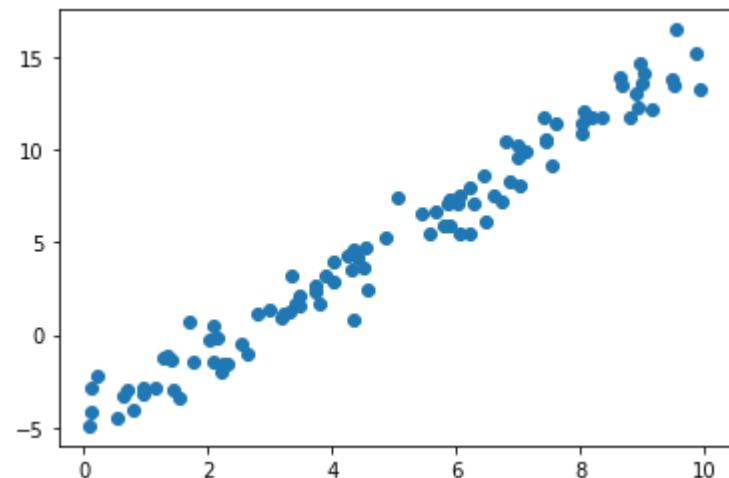
$$y|\beta, \sigma, X \sim Normal(X\beta, \sigma I)$$

I es la identidad; asumimos homocedasticidad.

La aproximación bayesiana permite inferir β y σ con incertidumbre

In [8]:

```
n = 100
x = np.random.uniform(0,10,n)
y = 2 * x - 5 + np.random.normal(0,1,n)
plt.scatter(x, y);
```

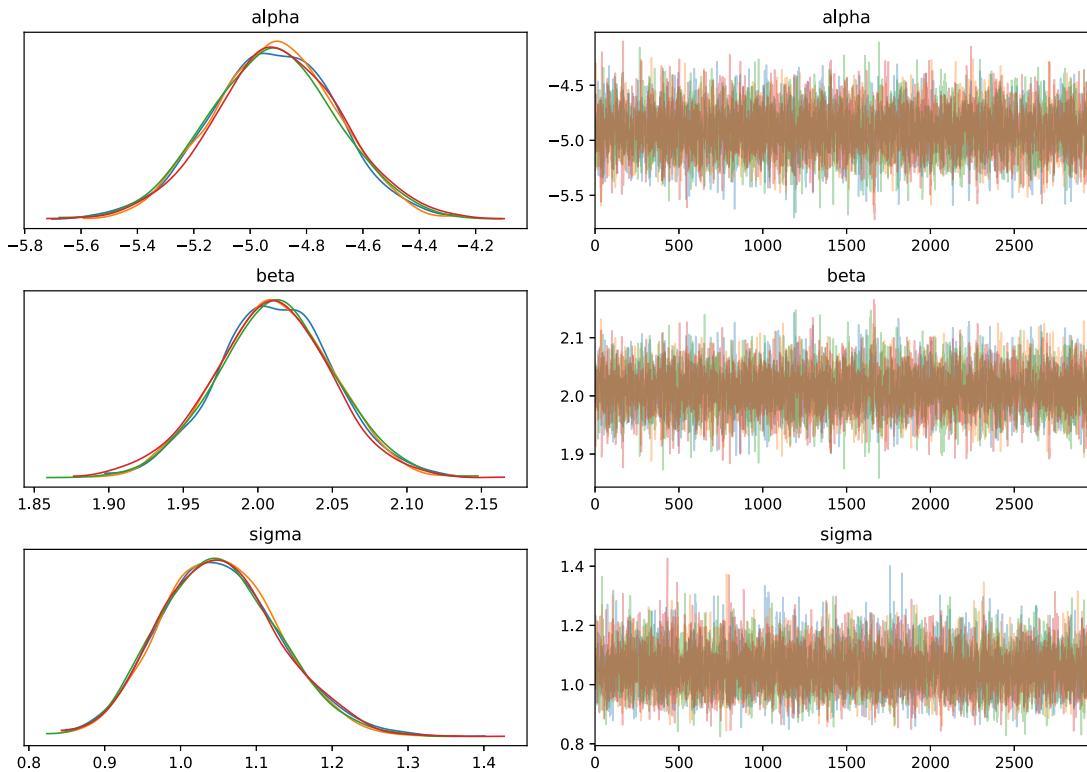


```
In [9]: with pm.Model() as basic_model:  
  
    #Priors  
    alpha = pm.Normal('alpha', mu=0, sd=10)  
    beta = pm.Normal('beta', mu=0, sd=10)  
    sigma = pm.HalfNormal('sigma', sd=3)  
  
    # Deterministics  
    #mu = pm.Deterministic("mu", alpha + beta*x) #to save in trace the deterministic value  
    mu = alpha + beta*x  
  
    # Likelihood  
    Ylikelihood = pm.Normal('Ylikelihood', mu=mu, sd=sigma, observed=y)  
  
    # Samples  
    trace = pm.sample(draws=3000)
```

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

100.00% [16000/16000 00:04<00:00 Sampling 4 chains, 0 divergences]

Sampling 4 chains for 1_000 tune and 3_000 draw iterations (4_000 + 12_000 draws total) took 6 seconds.
The acceptance probability does not match the target. It is 0.8849638452892347, but should be close to 0.8. Try to increase the number of tuning steps.

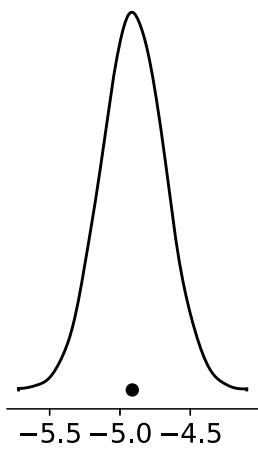


```
In [11]: with basic_model:  
    print(pm.summary(trace).round(2))  
    #pm.plots.forestplot(trace)
```

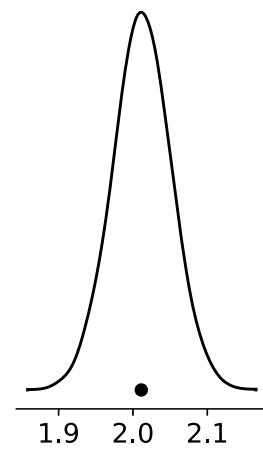
| | mean | sd | hdi_3% | hdi_97% | mcse_mean | mcse_sd | ess_mean | ess_sd | \ |
|-------|-------|------|--------|---------|-----------|---------|----------|--------|---|
| alpha | -4.91 | 0.22 | -5.31 | -4.49 | 0.0 | 0.0 | 5388.0 | 5364.0 | |
| beta | 2.01 | 0.04 | 1.94 | 2.09 | 0.0 | 0.0 | 5381.0 | 5373.0 | |
| sigma | 1.05 | 0.08 | 0.92 | 1.21 | 0.0 | 0.0 | 6651.0 | 6611.0 | |

| | ess_bulk | ess_tail | r_hat |
|-------|----------|----------|-------|
| alpha | 5390.0 | 5660.0 | 1.0 |
| beta | 5394.0 | 5758.0 | 1.0 |
| sigma | 6665.0 | 6080.0 | 1.0 |

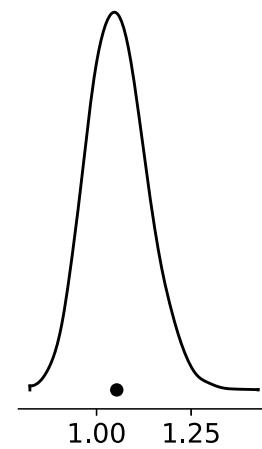
Intercept: α



Slope: β



Noise: σ



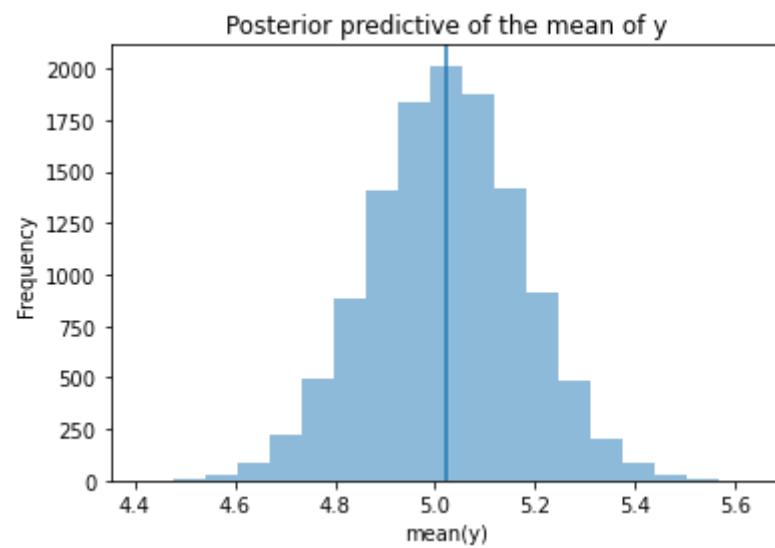
Podemos revisar el modelo de varias formas. Empecemos con posterior predictive checks (i.e. simular nueva data). Usemos el promedio de y en la data para comparar.

```
In [13]: with basic_model:  
    ypred = pm.sampling.sample_posterior_predictive(trace=trace)  
    y_sample_posterior_predictive = np.asarray(ypred[ 'Ylikelihood' ])
```

100.00% [12000/12000 00:11<00:00]

```
In [14]:
```

```
_ , ax = plt.subplots()  
ax.hist([n.mean() for n in y_sample_posterior_predictive], bins=19,  
alpha=0.5)  
ax.axvline(y.mean()) #actual data  
ax.set(title='Posterior predictive of the mean of y', xlabel='mean  
(y)', ylabel='Frequency');
```



El posterior predictive check nos dice que el modelo reproduce una estadística importante (el promedio de y).

Ahora veamos si un modelo nulo, con solo intercepto, nos sirve. Usaremos WAIC.

```
In [15]: with pm.Model() as null_model:  
    #Priors  
    alpha = pm.Normal('alpha', mu=0, sd=10)  
    sigma = pm.HalfNormal('sigma', sd=4)  
  
    # Deterministics  
    #mu = pm.Deterministic("mu", alpha + beta*x) #to save in trace the deterministic value  
    mu = alpha  
  
    # Likelihood  
    Ylikelihood = pm.Normal('Ylikelihood', mu=mu, sd=sigma, observed=y)  
  
    # Samples  
    trace_null = pm.sample(draws=3000)
```

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

100.00% [16000/16000 00:03<00:00 Sampling 4 chains, 0 divergences]

```
Sampling 4 chains for 1_000 tune and 3_000 draw iterations (4_000 + 12_000 draws total) took 3 seconds.
```

```
In [16]: with null_model:  
    print(pm.summary(trace_null).round(2))
```

| | mean | sd | hdi_3% | hdi_97% | mcse_mean | mcse_sd | ess_mean | ess_sd | \ |
|-------|------|------|--------|---------|-----------|---------|----------|---------|---|
| alpha | 5.00 | 0.58 | 3.89 | 6.07 | 0.01 | 0.0 | 10880.0 | 10785.0 | |
| sigma | 5.79 | 0.41 | 5.05 | 6.59 | 0.00 | 0.0 | 10879.0 | 10697.0 | |

| | ess_bulk | ess_tail | r_hat |
|-------|----------|----------|-------|
| alpha | 10890.0 | 8179.0 | 1.0 |
| sigma | 11001.0 | 8565.0 | 1.0 |

```
In [17]: with null_model:
    df_comp_WAIC = pm.compare({'Base': trace,
                               'Null': trace_null}, ic = 'waic')
display(df_comp_WAIC) # waic
```

/opt/anaconda3/lib/python3.7/site-packages/arviz/stats/stats.py:151: UserWarning:

The scale is now log by default. Use 'scale' argument or 'stats.ic_scale' rcParam if you rely on a specific value.

A higher log-score (or a lower deviance) indicates a model with better predictive accuracy.

"\nThe scale is now log by default. Use 'scale' argument or "
 /opt/anaconda3/lib/python3.7/site-packages/arviz/stats/stats.py:1415: UserWarning: For one or more samples the posterior variance of the log predictive densities exceeds 0.4. This could be indication of WAIC starting to fail.

See <http://arxiv.org/abs/1507.04544> for details

"For one or more samples the posterior variance of the log predictive "

| | rank | waic | p_waic | d_waic | weight | | se |
|-------------|-------------|-------------|---------------|---------------|---------------|---------|-----------|
| Null | 0 | -318.375 | 1.41601 | 0 | NaN | 2355.38 | 0 |
| Base | 1 | -18478.5 | 14613.2 | 18160.1 | 0 | 4.67814 | 231 |

CONCLUSIÓN

Hay modelos que explican mejor la data.

Podemos compararlos cualitativa (e.g. posterior predictive checks) o cuantitativamente (e.g. waic)

No hay modelo perfecto