Inferencia Bayesiana Entrega 2: Matias Moran LU 806/19

Ejercicio 1

Supongamos que uno tiene una variable aleatoria Y que puede modelar con una distribucion geométrica. Es decir que $P(Y=y|\theta)=\theta(1—\theta)^{y-1}$ para $y\in\{1,2,3,..\}$. Se utiliza un prior $\textcolor{red}{\backslash\mathrm{Beta}}(a,b)$ para θ.

A) Qué situacion se representa con una variable aleatoria geométrica?

La variable aleatoria geometrica se usa para representar una distribucion similar a la binomial pero donde queremos saber "Cuantos experimentos de bernoulli con probabilidad p de exito tengo que hacer hasta obtener un exito".

por ejemplo cuantas monedas tengo que tirar hasta sacar cara, cuantos dados tengo que tirar hasta que salga un 3, etc

B) Derivar la distribucion posterior para θ suponiendo que se observé $Y=y$. Identificar la distribucion encontrada y sus parametros.

$$P(\theta|Y=y)=\frac{P(Y=y|\theta)\cdot L(\theta)}{P(Y=y)}$$

$$P(\theta|Y=y)=\frac{f(\theta)\cdot f(\theta|Y=y)}{P(Y=y)}$$

$$P(\theta|Y=y)=\frac{f(\theta)\cdot(1-\theta)^{y-1}\cdot\theta}{P(Y=y)}$$

$$P(\theta|Y=y)\propto f(\theta)\cdot(1-\theta)^{y-1}\cdot\theta$$

Vemos que la forma $(1-\theta)^{y-1}\cdot\theta$ se asemeja mucho a la pdf de la Beta, asi que vamos a probar un prior $\textcolor{red}{\backslash\mathrm{Beta}}(a,b)$

$$P(\theta|Y=y)\propto(1-\theta)^{b-1}\cdot\theta^{a-1}\cdot(1-\theta)^{y-1}\cdot\theta$$

$$P(\theta|Y=y)\propto(1-\theta)^{b+y-2}\cdot\theta^{a}$$

$$P(\theta|Y=y)\propto B(a+1,b+y-1)$$

> ***NOTA:*** La razon porque la geometrica es tambien una distribucion conjugada de la Beta es porque es un "caso particular" de la distribucion binomial para k=1 (exitos igual a 1) y donde en vez de multiplicarlo por el numero de posibles permutaciones

> $\binom{n}{1}$ simplemente se omite esta constante porque el exito siempre esta en la ultima posicion y como esto es una constante se va cuando normalizamos la posterior ya que sigue siendo proporcional

C) El modelo Beta es un prior conjugado de la Geométrica?

Si, vimos que si tenemos un prior beta y nuestra distribucion es geometrica nuestro post tambien es Beta, asi que es un prior conjugado

Ejercicio 2

Identificar una pregunta que se pueda responder con un modelo de regresion lineal con $\beta_0. \beta_1, \sigma$ como parametros a estimar. Por ejemplo puede ser, altura del hijo en funcion de altura de la madre, velocidad maxima de un auto en funcion de caballos de fuerza, etc. Usatus propios datos o simulalos.

$$Y_i \sim \mathcal{N}(\mu(x_i), \sigma^2)$$

$$\mu = \beta_0 + \beta_1 \cdot x_i$$

> ***DATASETS:*** El dataset que vamos a utilizar es la informacion de alquiler de bicicletas en Washinton DC en funcion de la temperatura, este es un ejemplo del capitulo 9 del libro de Bayes rules.

Setup

In [274…

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import logging
import concurrent.futures as theads_futures
import pymc as pm

from scipy.stats import beta, norm, expon
from matplotlib.lines import Line2D
from IPython.display import Markdown

np.random.seed(42)
logging.basicConfig(level=logging.INFO)
```

In [275…

```
#https://www.kaggle.com/datasets/marklvl/bike-sharing-dataset?resource=download
df_bikes = pd.read_csv('../../sandbox/bikes/day.csv')

# temp: Normalized temperature in Celsius. The values are derived via (t-t_min)/(t_max-t_min), t_min=-8, t_max=+39 (onl
#unormalize temperatures
```

```
df_bikes.temp = (39 - (-8)) * df_bikes.temp + (-8)
df_bikes
```

Out[275…

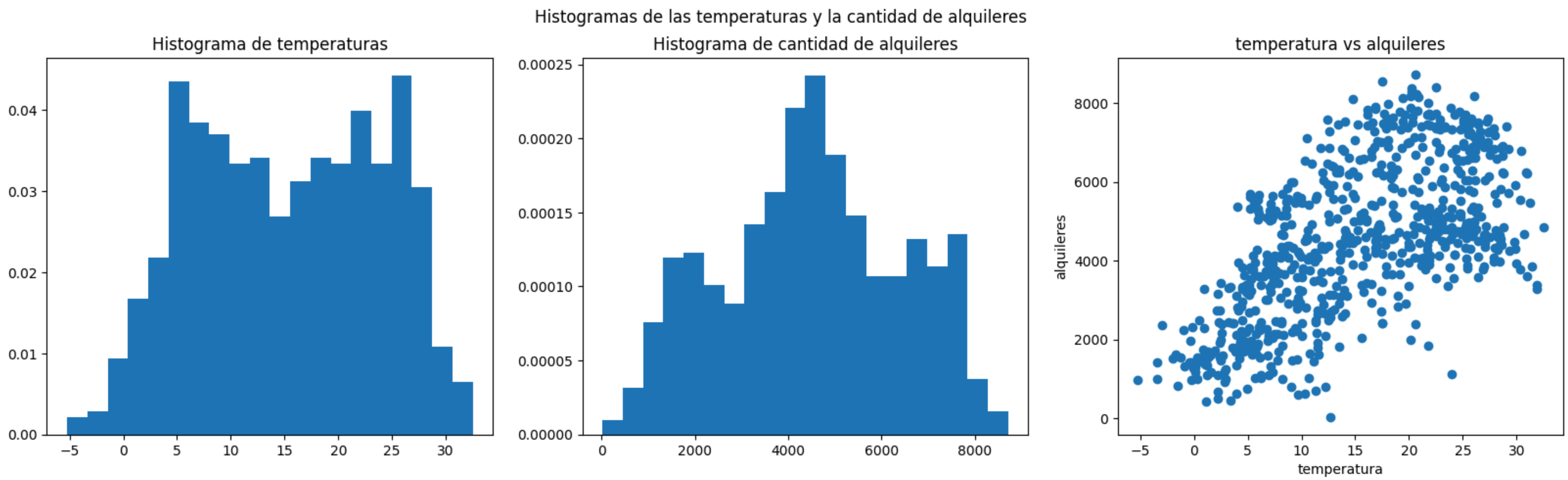
	instant	dteday	season	yr	mnth	holiday	weekday	workingday	weathersit	temp	atemp	hum	windspeed	casual
0	1	2011-01-01	1	0	1	0	6	0	2	8.175849	0.363625	0.805833	0.160446	331
1	2	2011-01-02	1	0	1	0	0	0	2	9.083466	0.353739	0.696087	0.248539	131
2	3	2011-01-03	1	0	1	0	1	1	1	1.229108	0.189405	0.437273	0.248309	120
3	4	2011-01-04	1	0	1	0	2	1	1	1.400000	0.212122	0.590435	0.160296	108
4	5	2011-01-05	1	0	1	0	3	1	1	2.666979	0.229270	0.436957	0.186900	82
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
726	727	2012-12-27	1	1	12	0	4	1	2	3.945849	0.226642	0.652917	0.350133	247
727	728	2012-12-28	1	1	12	0	5	1	2	3.906651	0.255046	0.590000	0.155471	644
728	729	2012-12-29	1	1	12	0	6	0	2	3.906651	0.242400	0.752917	0.124383	159
729	730	2012-12-30	1	1	12	0	0	0	1	4.024151	0.231700	0.483333	0.350754	364
730	731	2012-12-31	1	1	12	0	1	1	2	2.144151	0.223487	0.577500	0.154846	439

731 rows × 16 columns

In [276…

```
bins_n=20

fig, ax = plt.subplots(1, 3, figsize=(20, 5))
fig.suptitle('Histogramas de las temperaturas y la cantidad de alquileres')

ax[0].plot()
ax[0].set_title('Histograma de temperaturas')
ax[0].hist(df_bikes.temp, bins=bins_n, density=True)
```

```python
ax[1].plot()
ax[1].set_title('Histograma de cantidad de alquileres')
ax[1].hist(df_bikes.cnt, bins=bins_n, density=True)

ax[2].plot()
ax[2].set_title('temperatura vs alquileres')
ax[2].scatter(df_bikes.temp, df_bikes.cnt)
ax[2].set_xlabel('temperatura')
ax[2].set_ylabel('alquileres')

plt.show()
```



A) Proponer priors para $\beta_0. \beta_1, \sigma$

Para el caso de las bicicletas vamos a basarnos en el libro de bayes rules y vamos a hacer la regresion de temperatura vs cantidad de bicicletas alquiladas diarias poniendo priors normales a los parametros de la relacion lineal y un parametro exponencial para sigma para limitar el rango a que solo sea de valores positivos

$$\text{data:} \quad Y_i|\beta_0, \beta_1, \sigma \sim \mathcal{N}(\mu_i, \sigma^2) \quad \text{with} \quad \mu(x_i) = \beta_0 + \beta_1 \cdot X_i$$

$$\text{priors:} \quad \beta_0 \sim \mathcal{N}(m_0, s_0^2)$$

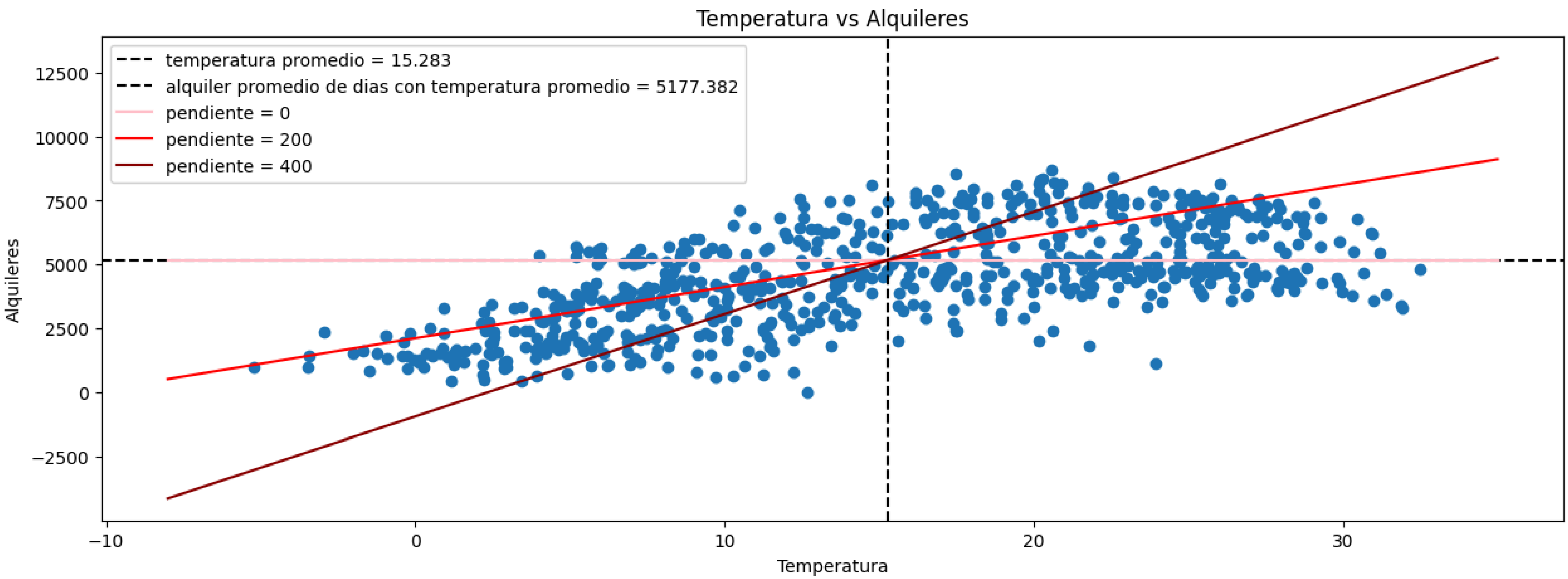
$$\beta_1 \sim \mathcal{N}(m_1, s_1^2)$$

$$\sigma \sim Exp(l)$$

In [277…

```
bins_n=20

fig, ax = plt.subplots(1, 1, figsize=(15, 5))

ax.set_title('Temperatura vs Alquileres')
ax.scatter(df_bikes.temp, df_bikes.cnt)
ax.set_xlabel('Temperatura')
ax.set_ylabel('Alquileres')

# Calculamos la temperatura media y la media de la cantidad de bicicletas alquiladas en ese dia para los datos que tien
mean_temperature = round(df_bikes.temp.mean(), 3)
average_temperature_samples = df_bikes[np.isclose(df_bikes.temp, np.full_like(df_bikes.temp, mean_temperature), atol= r
mean_rides = round(average_temperature_samples.cnt.mean(), 3)

ax.axvline(x=mean_temperature, color='black', linestyle='--', label=f'temperatura promedio = {mean_temperature}')
ax.axhline(y=mean_rides, color='black', linestyle='--', label=f'alquiler promedio de dias con temperatura promedio = {m


x = np.linspace(-8, 35, 400)
y = 0/5 * (x - mean_temperature) + mean_rides
plt.plot(x, y, label='pendiente = 0', color='pink')
y = 1000/5 * (x - mean_temperature) + mean_rides
plt.plot(x, y, label='pendiente = 200', color='red')
y = 2000/5 * (x - mean_temperature) + mean_rides
plt.plot(x, y, label='pendiente = 400', color='darkred')


ax.legend()
plt.show()


Markdown(fr"""
Para el caso de las bicicletas, podemos ver:

- en un dia promedio de {mean_temperature} de temperatura, El alquier de bicicletas es de {mean_rides} con una desviaci
esto nos dice que $\beta_{{0}} \sim \mathcal{{N}}({mean_rides}, 3000^{{2}})$ donde estamos centrando los datos en en el

- por cada aumento de 5 en el valor de la temperatura la cantidad de bicicletas alquiladas sube en 200 aproximadamente

- en un dia cualquiera cantidad de alquileres tiene una rango que puede ser de $\pm$ 3000 alquileres como sabemos que 3
esto nos dice que podemos modelar nuestro $\sigma$ para que su expected value sea 1000.<br>
$E(\sigma) = \frac{{1}}{{l}} = 1000 \implies l = {round(1/1000, 6)}$


Finalmente tenemos los priors:
```

```
$$
\text{{data:}} \quad Y_{{i}}|\beta_{{0}}, \beta_{{1}} ,\sigma \sim \mathcal{{N}}(\mu_{{i}}, \sigma^{{2}}) \quad  \text{
$$

$$
\text{{priors:}} \quad \beta_{{0}} \sim \mathcal{{N}}({mean_rides}, 3000^{{2}})
$$
$$
\quad \quad \quad \quad \beta_{{1}} \sim \mathcal{{N}}(200, 200^{{2}})
$$
$$
\quad \quad \quad \sigma \sim Exp({round(1/1000, 6)})
$$

""")
```



Out[277… Para el caso de las bicicletas, podemos ver:

- en un dia promedio de 15.283 de temperatura, El alquier de bicicletas es de 5177.382 con una desviacion de hasta 1817 y 8555 esto nos dice que $\beta_0 \sim \mathcal{N}(5177.382, 3000^2)$ donde estamos centrando los datos en en el origen como lo indican las lineas punteadas negras
- por cada aumento de 5 en el valor de la temperatura la cantidad de bicicletas alquiladas sube en 200 aproximadamente y esto puede variar entre a lo sumo 0 y 400 como lo representan las rectas marcadas en rojo, esto nos dice que $\beta_1 \sim \mathcal{N}(200, 200^2)$ es una distribucion razonable
- en un dia cualquiera cantidad de alquileres tiene una rango que puede ser de $\pm$ 3000 alquileres como sabemos que 3 desviaciones estandard deberia contener a casi todos los datos deducimos que la desviacion estandar es de $\frac{3000}{/}3 = 1000$. esto nos dice que podemos modelar nuestro σ para que su expected value sea 1000. $E(\sigma) = \frac{1}{l} = 1000 \implies l = 0.001$

Finalmente tenemos los priors:

$$\text{data:} \quad Y_i|\beta_0, \beta_1, \sigma \sim \mathcal{N}(\mu_i, \sigma^2) \quad \text{with} \quad \mu(x_i) = \beta_0 + \beta_1 \cdot X_i$$

$$\text{priors:} \quad \beta_0 \sim \mathcal{N}(5177.382, 3000^2)$$

$$\beta_1 \sim \mathcal{N}(200, 200^2)$$

$$\sigma \sim Exp(0.001)$$

In [278…

```
beta_0_mean = mean_rides
beta_0_std = 3000

beta_1_mean = 200
beta_1_std = 200

sigma_scale = 1000
```

B) Escribir la likelihood de los datos de forma analitica en funcion de los parametros.

La Likehood de los parametros en funcion de los datos $\overrightarrow{y} = (y_1, y_2, \ldots, y_n)$ se puede definir como $L(\beta_0, \beta_1, \sigma|\overrightarrow{y}) = f(\overrightarrow{y}|\beta_0, \beta_1, \sigma) = \prod_{i=1}^{n} f(y_i|\beta_0, \beta_1, \sigma)$

C) Implementar el algoritmo de Metrépolis (MCMC) y generar 5.000 samples del posterior sin descontar el tramo inicial de la cadena (burn = 0).

In [279…
```
def metropolis_hasting(log_posterior, proposal_function, initial_candidate, num_samples):
    result_chain = np.zeros(shape=(num_samples, len(initial_candidate)))
    result_chain[0] = initial_candidate

    for i in range(1, num_samples):

        next_candidate = proposal_function(result_chain[i-1])
        log_acceptance_ratio = log_posterior(next_candidate) - log_posterior(result_chain[i-1])

        if log_acceptance_ratio >= 0:
            result_chain[i] = next_candidate
            continue

        if np.log(np.random.rand()) < log_acceptance_ratio:
            result_chain[i] = next_candidate
        else:
            result_chain[i] = result_chain[i-1]

    return result_chain

# hacemos que el siguiente candidato sea normalmente distribuido con media = current_candidate y desviacion = std ya qu
# en la mayoria de los casos (3 desviaciones estadard).
# para beta_0, beta_1 y sigma elegimos std de dos ordenes de magnitud mas chicos
def bikes_normal_proposal(current_candidate):
    new_beta_0 = np.random.normal(current_candidate[0], 50)
    new_beta_1 = np.random.normal(current_candidate[1], 5)
    new_sigma = np.random.normal(current_candidate[2], 50)
    return [new_beta_0, new_beta_1, new_sigma]

def bikes_log_posterior_unnormalize(candidate):
    log_prior = np.log(norm.pdf(candidate[0], beta_0_mean, beta_0_std)) + np.log(norm.pdf(candidate[1], beta_1_mean, be

    temps = df_bikes['temp'].values
    cnts = df_bikes['cnt'].values

    predicted = candidate[0] + candidate[1] * temps

    log_likehoods = np.log(norm.pdf(cnts, predicted, candidate[2]))

    log_likehood = np.sum(log_likehoods)

    return log_prior + log_likehood
```

In [280…
```
bike_chain_1 = metropolis_hasting(bikes_log_posterior_unnormalize, bikes_normal_proposal, [beta_0_mean, beta_1_mean, si
bike_chain_1
```

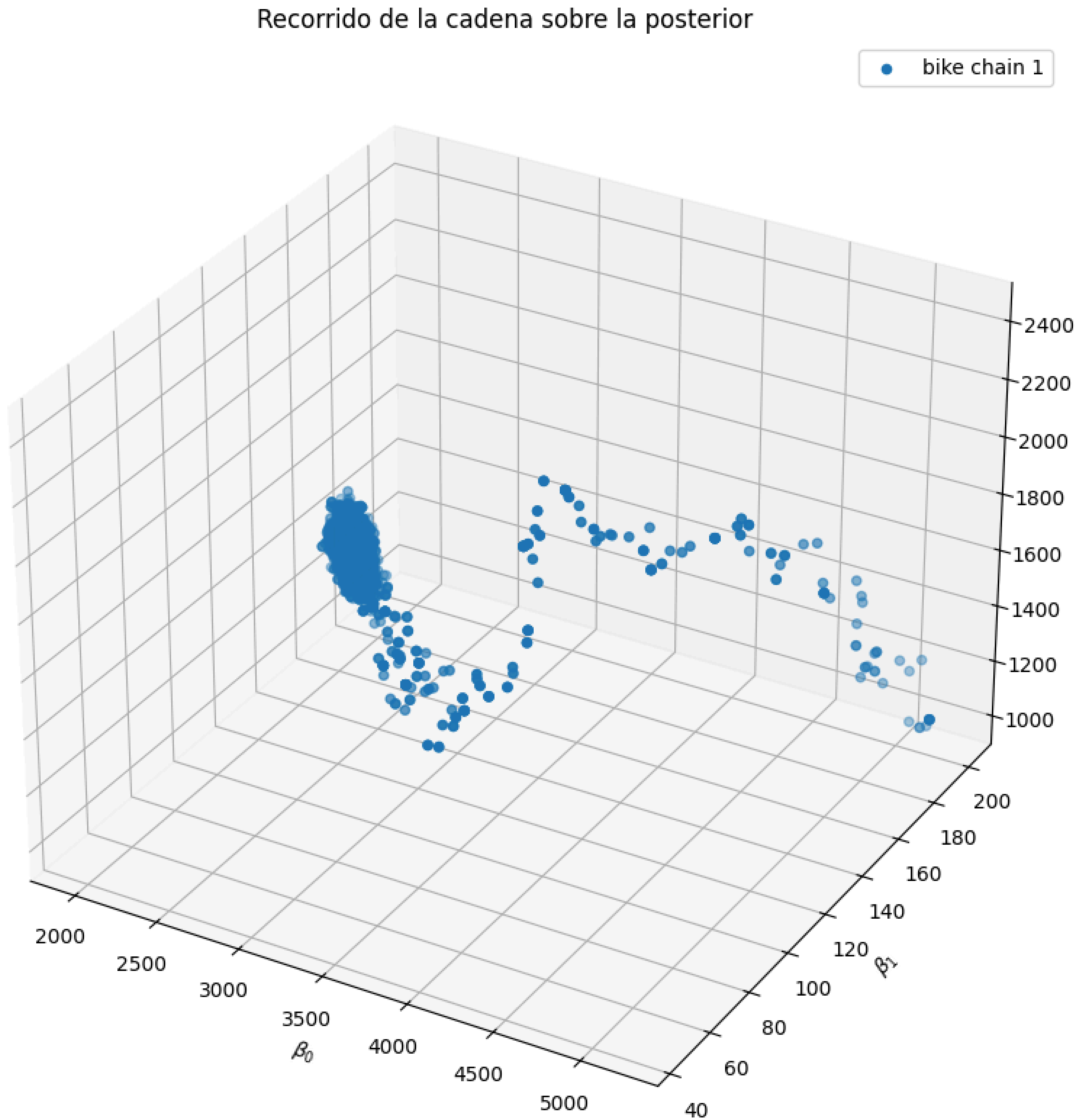
Out[280…

```
array([[5177.382     ,  200.        , 1000.        ],
       [5202.21770765,  199.30867849, 1032.38442691],
       [5202.21770765,  199.30867849, 1032.38442691],
       ...,
       [2498.8362679 ,  139.69833086, 1519.53298261],
       [2520.51890111,  131.06431485, 1560.61377022],
       [2403.11548126,  131.03570065, 1550.37073851]])
```

D) Graficar la cadena resultante (en 3d para todo el posterior o independientemente para cada parametro en 3 graficos distintos). Parece haber convergido la cadena?

In [281…

```python
fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(111, projection='3d')

ax.scatter(bike_chain_1[:, 0], bike_chain_1[:, 1], bike_chain_1[:, 2], label='bike chain 1')

ax.set_xlabel(r'$\beta_{0}$')
ax.set_ylabel(r'$\beta_{1}$')
ax.set_zlabel(r'$\sigma$')
ax.set_title('Recorrido de la cadena sobre la posterior')

ax.legend()
plt.show()


Markdown(fr"""

#### La cadena parece haber convergido y estabilizado

""")
```



Recorrido de la cadena sobre la posterior
bike chain 1
2000
2500
3000
3500
4000
4500
5000
β_0
40
60
80
100
120
140
160
180
200
β_1
1000
1200
1400
1600
1800
2000
2200
2400
σ

Out[281… La cadena parece haber convergido y estabilizado

E) Repetir el inciso c) y d) para 3 cadenas paralelas. Grafiquelas superpuestas con colores distintos. Cuantos samples tarda en llegar a estado de equilibrio (a ojo)?

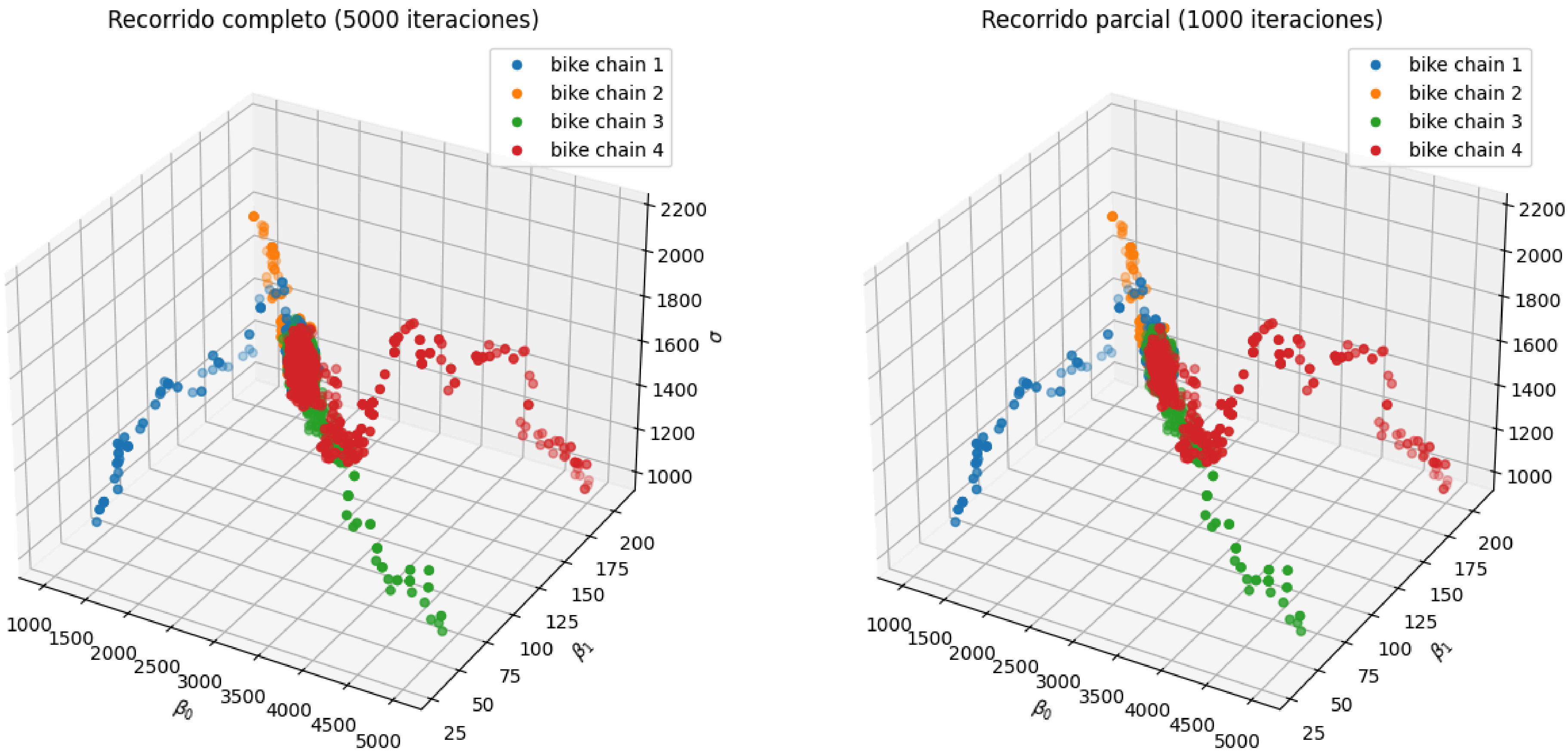
In [282…

```
arguments = [
    (bikes_log_posterior_unnormalize, bikes_normal_proposal, [1000, 60, 1000], 5000),
    (bikes_log_posterior_unnormalize, bikes_normal_proposal, [1000, 200, 1800], 5000),
    (bikes_log_posterior_unnormalize, bikes_normal_proposal, [5000, 60, 1000], 5000),
    (bikes_log_posterior_unnormalize, bikes_normal_proposal, [5000, 200, 1000], 5000),
]

with theads_futures.ThreadPoolExecutor(max_workers=4) as executor:
    futures = [executor.submit(metropolis_hasting, *args) for args in arguments]
    theads_futures.wait(futures)
    bike_chains = [future.result() for future in futures]
```

In [283…

```
fig = plt.figure(figsize=(15,15))

ax0 = fig.add_subplot(121, projection='3d')

ax0.scatter(bike_chains[0][:, 0], bike_chains[0][:, 1], bike_chains[0][:, 2], label='bike chain 1')
ax0.scatter(bike_chains[1][:, 0], bike_chains[1][:, 1], bike_chains[1][:, 2], label='bike chain 2')
ax0.scatter(bike_chains[2][:, 0], bike_chains[2][:, 1], bike_chains[2][:, 2], label='bike chain 3')
ax0.scatter(bike_chains[3][:, 0], bike_chains[3][:, 1], bike_chains[3][:, 2], label='bike chain 4')

ax0.set_xlabel(r'$\beta_{0}$')
ax0.set_ylabel(r'$\beta_{1}$')
ax0.set_zlabel(r'$\sigma$')
ax0.set_title('Recorrido completo (5000 iteraciones)')

ax0.legend()

ax1 = fig.add_subplot(122, projection='3d')

mc_convergence_iterations = 1000

ax1.scatter(bike_chains[0][:mc_convergence_iterations, 0], bike_chains[0][:mc_convergence_iterations, 1], bike_chains[0
ax1.scatter(bike_chains[1][:mc_convergence_iterations, 0], bike_chains[1][:mc_convergence_iterations, 1], bike_chains[1
ax1.scatter(bike_chains[2][:mc_convergence_iterations, 0], bike_chains[2][:mc_convergence_iterations, 1], bike_chains[2
ax1.scatter(bike_chains[3][:mc_convergence_iterations, 0], bike_chains[3][:mc_convergence_iterations, 1], bike_chains[3

ax1.set_xlabel(r'$\beta_{0}$')
```

```
ax1.set_ylabel(r'$\beta_{1}$')
ax1.set_zlabel(r'$\sigma$')
ax1.set_title(f'Recorrido parcial ({mc_convergence_iterations} iteraciones)')

ax1.legend()

plt.show()


Markdown(fr"""

#### Vemos que las cadenas se mezclan y se estabilizan a partir de las {mc_convergence_iterations} iteraciones aproxima

""")
```



Out[283… Vemos que las cadenas se mezclan y se estabilizan a partir de las 1000 iteraciones aproximadamente

F) Elija al azar 100 samples del posterior y grafique las 100 rectas correspondientes superpuestas a los datos.

In [284…

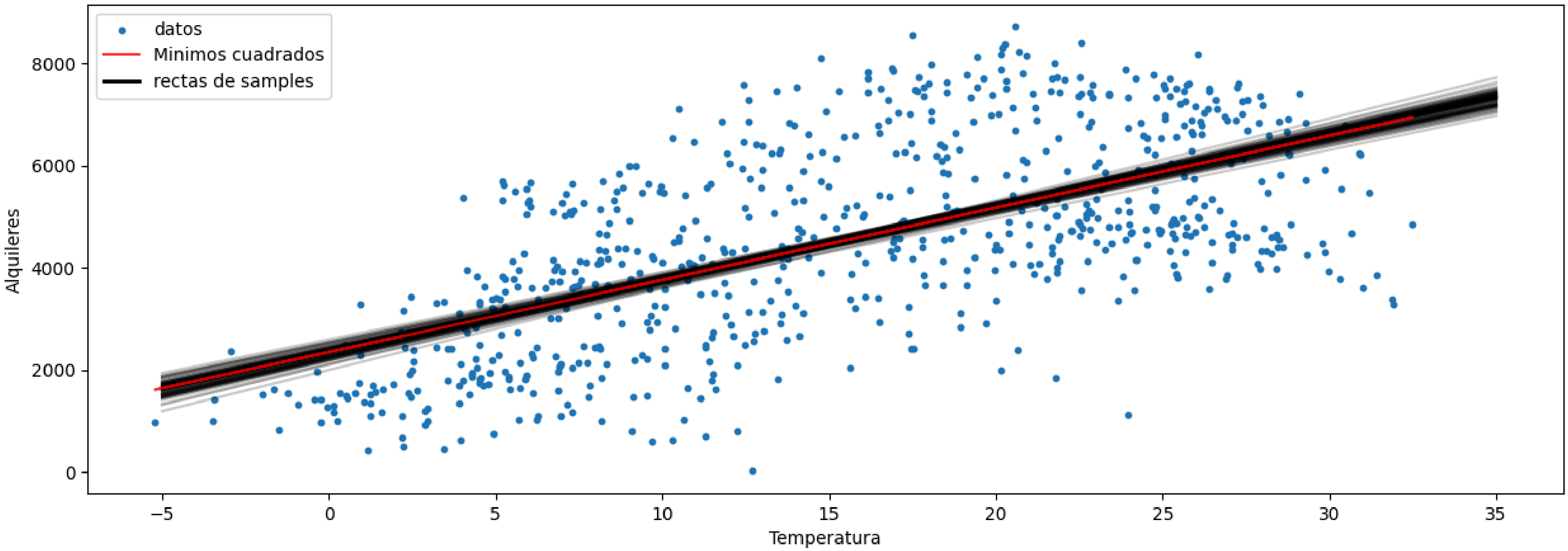
```python
# nos quedamos con los 4000 samples de cada cadena evitando tomar los 1000 primeros de burn de cada una
bike_final_chain = np.concatenate([bike_chains[0][mc_convergence_iterations:], bike_chains[1][mc_convergence_iterations
bike_final_chain.shape
```

Out[284…

```
(16000, 3)
```

In [285…

```python
bikes_100_posteriors = bike_final_chain[np.random.choice(bike_final_chain.shape[0], 100, replace=False)]

plt.figure(figsize=(15, 5))

plt.scatter(df_bikes.temp, df_bikes.cnt, s=10, label = 'datos')

x = np.linspace(-5, 35, 100)

for i in range(len(bikes_100_posteriors)):
    plt.plot(x, bikes_100_posteriors[i, 0] + bikes_100_posteriors[i, 1] * x, color = 'black', alpha=0.2)

regression_line = np.polyval(np.polyfit(df_bikes.temp, df_bikes.cnt, 1), df_bikes.temp)
plt.plot(df_bikes.temp, regression_line, color='red', alpha= 0.8, label = 'Minimos cuadrados')


handles, labels = plt.gca().get_legend_handles_labels()
legend_item = Line2D([0], [0], label='rectas de samples', color='black', lw=2)
handles.append(legend_item)
plt.legend(handles=handles, labels=labels + ['rectas de samples'])

plt.xlabel('Temperatura')
plt.ylabel('Alquileres')
plt.show()



Markdown(fr"""

#### Vemos que las rectas de la posterior tienden a centrarse a la recta de minimos cuadrados y que algunas de las rect

""")
```



Out[285… Vemos que las rectas de la posterior tienden a centrarse a la recta de minimos cuadrados y que algunas de las rectas factibles tienen una pendiente mas baja y un intercepto mas alta

G) Genere una distribucion posterior predictive para Y con algun X fijo. Utilicela para responder alguna pregunta relevante a sus datos.

Cual es la probabilidad de que un dia de 20 grados tenga mas de 6000 alquileres de bicicletas?

In [286…
```
# obtenemos 5000 samples de la posterior
bikes_5000_posteriors = bike_final_chain[np.random.choice(bike_final_chain.shape[0], 5000, replace=False)]

days_with_more_than_6000_rides = 0

for k in range(len(bikes_5000_posteriors)):
    posterior_sample = bikes_5000_posteriors[k]
    temperature_predicted = np.random.normal(posterior_sample[0] + posterior_sample[1] * 20, posterior_sample[2], 1)
    if temperature_predicted > 6000:
        days_with_more_than_6000_rides += 1


Markdown(fr"""

### La probabilidad de que el tiempo de alquiler de bicicletas sea mayor a 6000 es de {days_with_more_than_6000_rides/l
```

```
""")
```

Out[286…

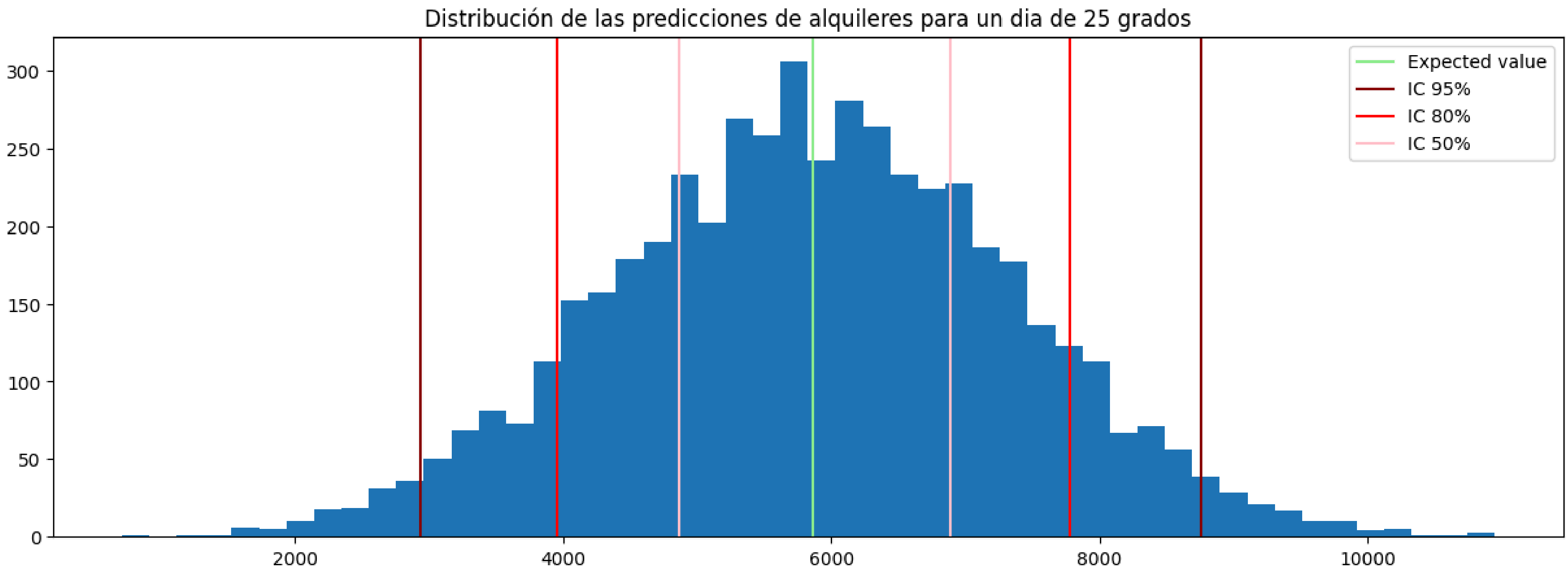
La probabilidad de que el tiempo de alquiler de bicicletas sea mayor a 6000 es de 0.2954.

Cuantos alquileres de bicis voy a tener en un dia de 25 grados?

In [287…

```
MH_temperatures_predicted = np.array([])

for k in range(len(bikes_5000_posteriors)):
    posterior_sample = bikes_5000_posteriors[k]
    temperature_predicted = np.random.normal(posterior_sample[0] + posterior_sample[1] * 25, posterior_sample[2], 1)
    MH_temperatures_predicted = np.append(MH_temperatures_predicted, temperature_predicted)

plt.figure(figsize=(15, 5))

plt.title('Distribución de las predicciones de alquileres para un dia de 25 grados')

plt.hist(MH_temperatures_predicted, bins=50)

percentil_50 = np.percentile(MH_temperatures_predicted, 50)
percentil_2_5 = np.percentile(MH_temperatures_predicted, 2.5)
percentil_10 = np.percentile(MH_temperatures_predicted, 10)
percentil_25 = np.percentile(MH_temperatures_predicted, 25)
percentil_75 = np.percentile(MH_temperatures_predicted, 75)
percentil_90 = np.percentile(MH_temperatures_predicted, 90)
percentil_97_5 = np.percentile(MH_temperatures_predicted, 97.5)

plt.axvline(percentil_50, color = 'lightgreen', label = 'Expected value')

plt.axvline(percentil_2_5, color = 'darkred')
plt.axvline(percentil_97_5, color = 'darkred', label = 'IC 95%')

plt.axvline(percentil_10, color = 'red')
plt.axvline(percentil_90, color = 'red', label = 'IC 80%')

plt.axvline(percentil_25, color = 'pink')
plt.axvline(percentil_75, color = 'pink', label = 'IC 50%')

plt.legend()
plt.show()

Markdown(fr"""

#### El expected value de un dia de 25 grados es de {percentil_50}
#### El intervalo de credibilidad del 50% es de ({percentil_25}, {percentil_75})
```

```
#### El intervalo de credibilidad del 80% es de ({percentil_10}, {percentil_90})
#### El intervalo de credibilidad del 95% es de ({percentil_2_5}, {percentil_97_5})

""")
```



Out[287… El expected value de un dia de 25 grados es de 5858.2750198375625

El intervalo de credibilidad del 50% es de (4861.546296993775, 6886.136028625924)

El intervalo de credibilidad del 80% es de (3957.5946997613637, 7785.51289569356)

El intervalo de credibilidad del 95% es de (2942.1424495970846, 8757.984813566463)

Ejercicio 3

Para los mismos datos, mismo modelo y mismos priors del ejercicio 2, haga la regresion lineal utilizando brms o un paquete similar.

A) Grafique, utilizando el paquete, las cadenas marginales. Parecen haber convergido?

In [288…

```python
with pm.Model() as model:  # model specifications in PyMC are wrapped in a with-statement
    # Define priors
    beta_0 = pm.Normal("beta_0", beta_0_mean, sigma=beta_0_std)
    beta_1 = pm.Normal("beta_1", beta_1_mean, sigma=beta_1_std)
    sigma = pm.Exponential("sigma", scale=sigma_scale)

    # Define likelihood
    likelihood = pm.Normal("y", mu=beta_0 + beta_1 * df_bikes.temp.values, sigma=sigma, observed=df_bikes.cnt.values)

    # Inference!
    # draw 5000 posterior samples using NUTS sampling
    idata = pm.sample(5000, chains = 4)
```

```
INFO:pymc.sampling.mcmc:Auto-assigning NUTS sampler...
INFO:pymc.sampling.mcmc:Initializing NUTS using jitter+adapt_diag...
INFO:pymc.sampling.mcmc:Multiprocess sampling (4 chains in 2 jobs)
INFO:pymc.sampling.mcmc:NUTS: [beta_0, beta_1, sigma]
/home/matias/Database/files/Vault/files/Repos/UBA/EstimacionBayesiana/.venv/lib/python3.12/site-packages/rich/
py:231: UserWarning: install "ipywidgets" for Jupyter support
  warnings.warn('install "ipywidgets" for Jupyter support')
```

```
INFO:pymc.sampling.mcmc:Sampling 4 chains for 1_000 tune and 5_000 draw iterations (4_000 + 20_000 draws total) took 23
seconds.
```

In [289…

```python
idata.posterior
```

Out[289... xarray.Dataset

▸ Dimensions: (**chain**: 4, **draw**: 5000)

▾ Coordinates:

chain	(chain)	int64	0 1 2 3
draw	(draw)	int64	0 1 2 3 4 ... 4996 4997 4998 4999

▾ Data variables:

beta_0	(chain, draw)	float64	2.44e+03 2.509e+03 ... 2.496e+03
beta_1	(chain, draw)	float64	137.9 139.1 143.9 ... 143.1 130.2
sigma	(chain, draw)	float64	1.482e+03 1.477e+03 ... 1.46e+03

▸ Indexes: (2)

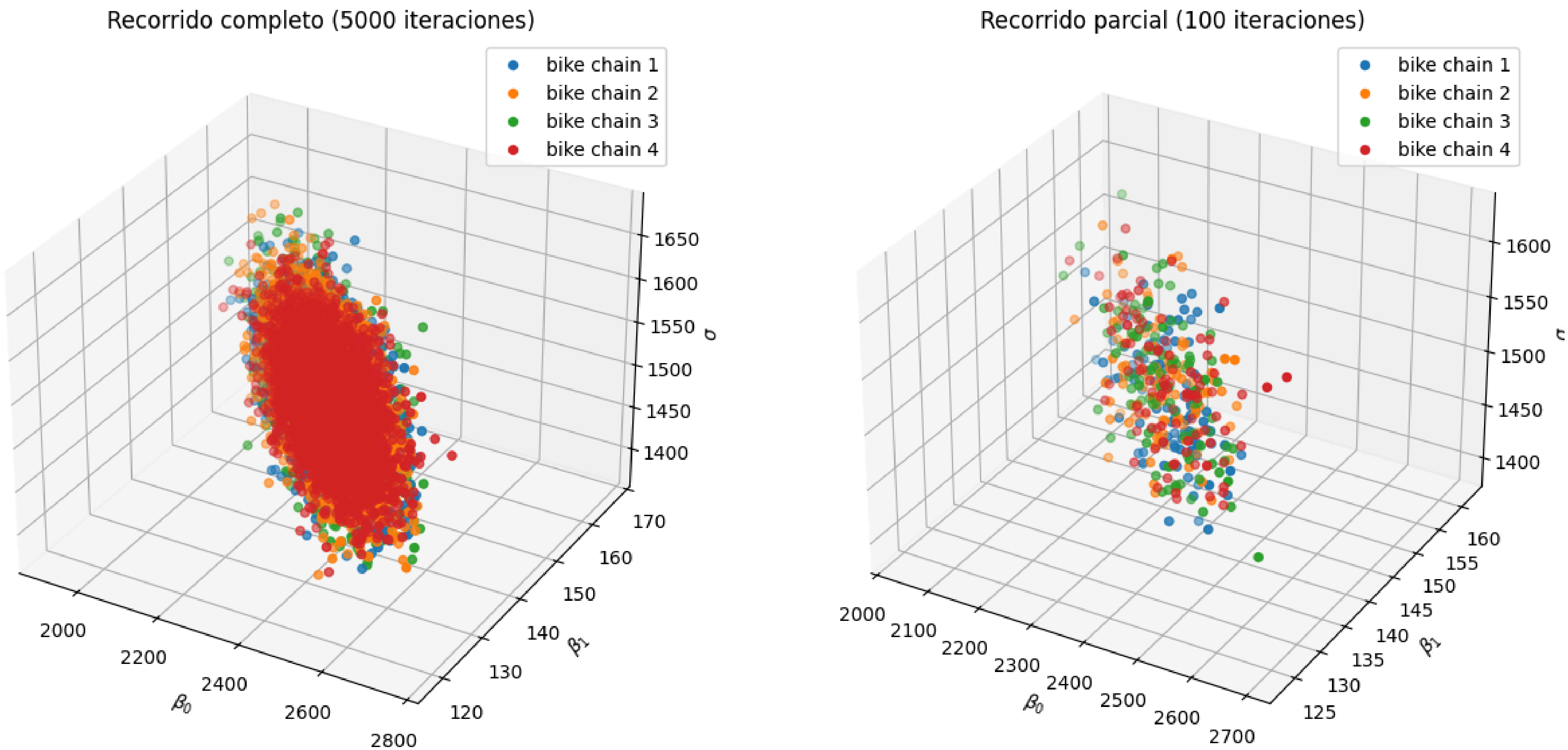
▾ Attributes:

created_at :	2024-10-03T00:07:46.173678+00:00
arviz_version :	0.20.0
inference_librar...	pymc
inference_librar...	5.16.2
sampling_time :	22.822019815444946
tuning_steps :	1000

In [290...

```
fig = plt.figure(figsize=(15,15))

ax0 = fig.add_subplot(121, projection='3d')

ax0.scatter(idata.posterior["beta_0"][0], idata.posterior["beta_1"][0], idata.posterior["sigma"][0], label='bike chain
ax0.scatter(idata.posterior["beta_0"][1], idata.posterior["beta_1"][1], idata.posterior["sigma"][1], label='bike chain
ax0.scatter(idata.posterior["beta_0"][2], idata.posterior["beta_1"][2], idata.posterior["sigma"][2], label='bike chain
ax0.scatter(idata.posterior["beta_0"][3], idata.posterior["beta_1"][3], idata.posterior["sigma"][3], label='bike chain

ax0.set_xlabel(r'$\beta_{0}$')
ax0.set_ylabel(r'$\beta_{1}$')
ax0.set_zlabel(r'$\sigma$')
ax0.set_title('Recorrido completo (5000 iteraciones)')
```

```
ax0.legend()

ax1 = fig.add_subplot(122, projection='3d')

pymc_convergence_iterations = 100

ax1.scatter(idata.posterior["beta_0"][0][:pymc_convergence_iterations], idata.posterior["beta_1"][0][:pymc_convergence_
ax1.scatter(idata.posterior["beta_0"][1][:pymc_convergence_iterations], idata.posterior["beta_1"][1][:pymc_convergence_
ax1.scatter(idata.posterior["beta_0"][2][:pymc_convergence_iterations], idata.posterior["beta_1"][2][:pymc_convergence_
ax1.scatter(idata.posterior["beta_0"][3][:pymc_convergence_iterations], idata.posterior["beta_1"][3][:pymc_convergence_

ax1.set_xlabel(r'$\beta_{0}$')
ax1.set_ylabel(r'$\beta_{1}$')
ax1.set_zlabel(r'$\sigma$')
ax1.set_title(f'Recorrido parcial ({pymc_convergence_iterations} iteraciones)')

ax1.legend()

plt.show()


Markdown(fr"""

#### Vemos que las cadenas no arrancan desde una posicion inical y van convergiendo sino que comienzan cerca de la zona

""")
```

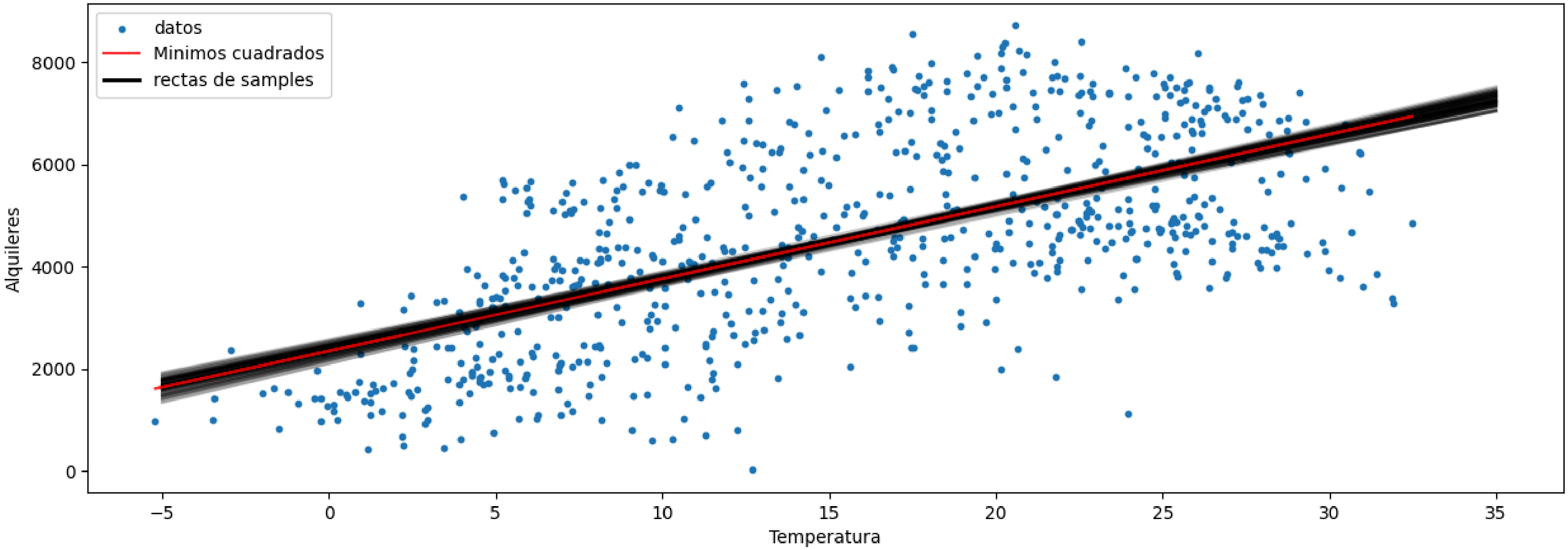


Out[290… Vemos que las cadenas no arrancan desde una posicion inical y van convergiendo sino que comienzan cerca de la zona de convergencia

In [291…

```
plt.figure(figsize=(15, 5))

plt.scatter(df_bikes.temp, df_bikes.cnt, s=10, label = 'datos')

x = np.linspace(-5, 35, 100)

for i in range(100):
    plt.plot(x, idata.posterior["beta_0"][0][i].item() + idata.posterior["beta_1"][0][i].item() * x, color = 'black', a

regression_line = np.polyval(np.polyfit(df_bikes.temp, df_bikes.cnt, 1), df_bikes.temp)
plt.plot(df_bikes.temp, regression_line, color='red', alpha= 0.8, label = 'Minimos cuadrados')


handles, labels = plt.gca().get_legend_handles_labels()
legend_item = Line2D([0], [0], label='rectas de samples', color='black', lw=2)
handles.append(legend_item)
plt.legend(handles=handles, labels=labels + ['rectas de samples'])
```

```
plt.xlabel('Temperatura')
plt.ylabel('Alquileres')
plt.show()



Markdown(fr"""

#### Vemos que las rectas de la posterior tienden a centrarse a la recta de minimos cuadrados y que algunas de las rect

""")
```



Out[291…

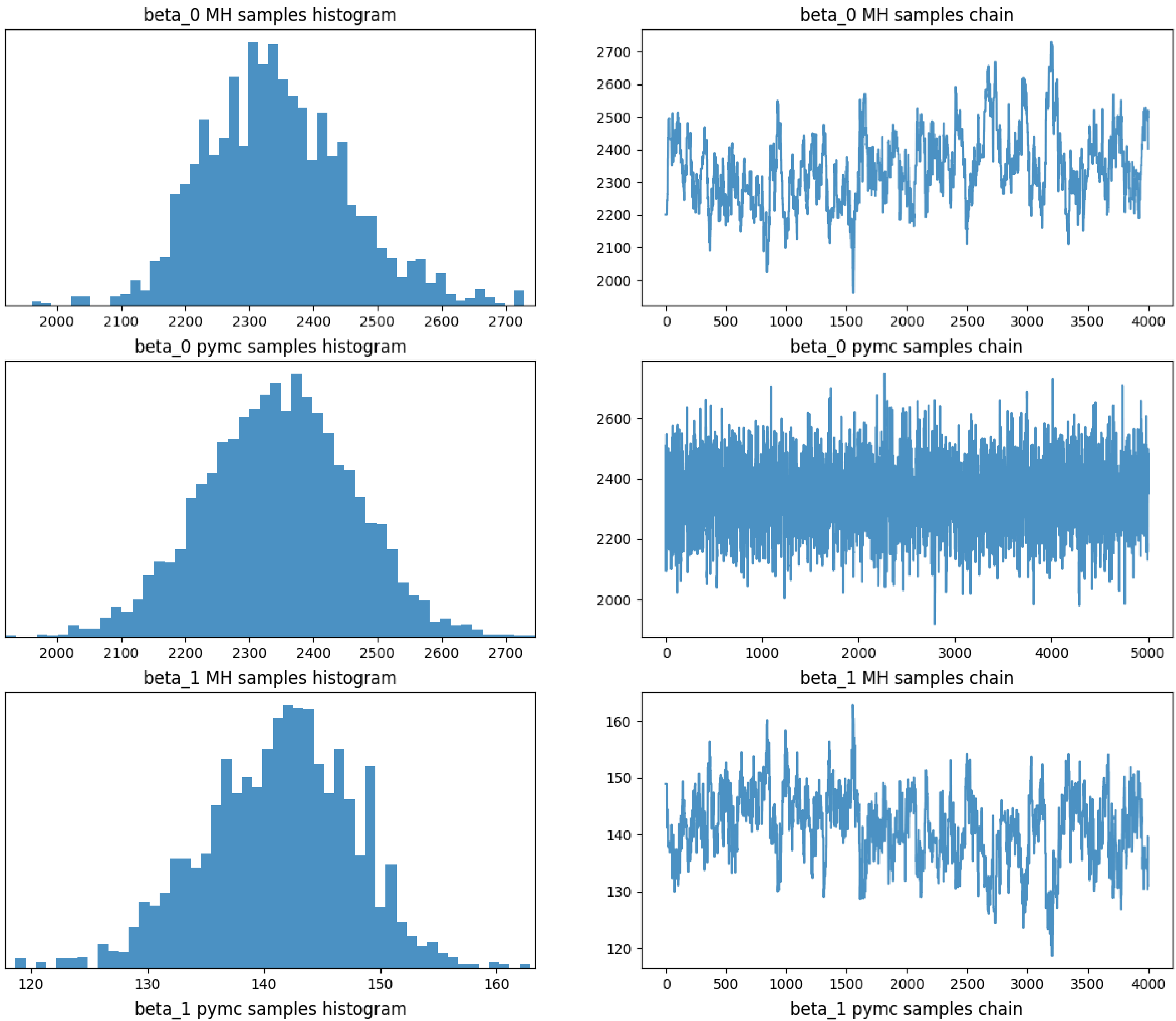
Vemos que las rectas de la posterior tienden a centrarse a la recta de minimos cuadrados y que algunas de las rectas factibles tienen una pendiente mas baja y un intercepto mas alta o una pendiente mas alta e intercepcion mas alta

B) Coinciden los posteriors marginales (los de cada parametro por separado) con los calculados en el ejercicio 2?

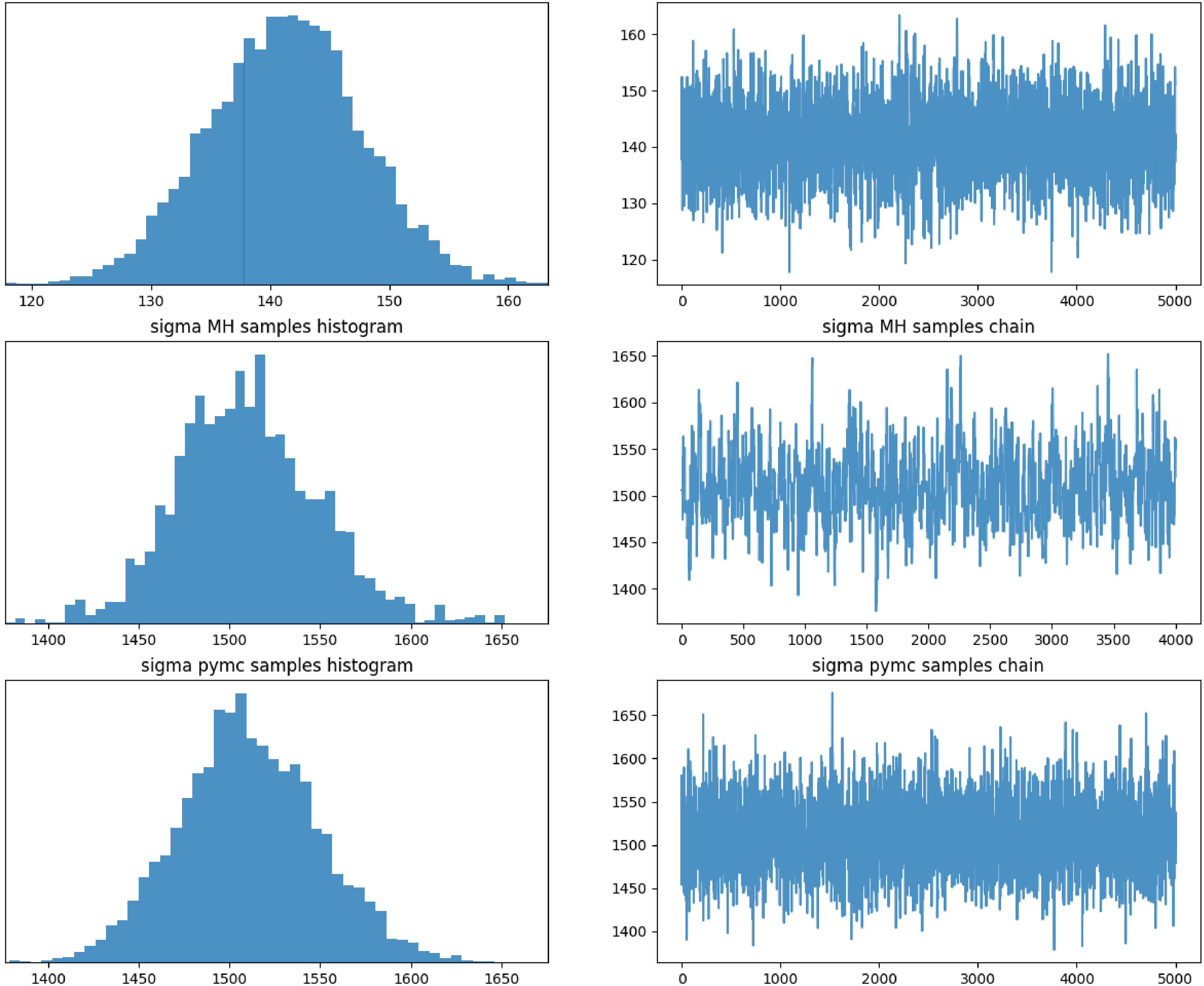
In [293…

```
_, ax = plt.subplots(6, 2, figsize=(15, 25))

for index, param in enumerate(['beta_0', 'beta_1', 'sigma']):

    index = index * 2
```

```
    min_value = min(np.min(bike_chain_1[mc_convergence_iterations:][:, index//2]), np.min(idata.posterior[param][0]))
    max_value = max(np.max(bike_chain_1[mc_convergence_iterations:][:, index//2]), np.max(idata.posterior[param][0]))
    ax[index, 0].set_xlim(min_value, max_value)
    ax[index + 1, 0].set_xlim(min_value, max_value)

    ax[index, 0].set_title(f"{param} MH samples histogram")
    ax[index, 0].set_yticks([])
    ax[index, 0].hist(bike_chain_1[mc_convergence_iterations:][:, index//2], alpha=0.8, density=True, bins=50)

    ax[index + 1, 0].set_title(f"{param} pymc samples histogram")
    ax[index + 1, 0].set_yticks([])
    ax[index + 1, 0].hist(idata.posterior[param][0], alpha=0.8, density=True, bins=50)

    ax[index, 1].set_title(f"{param} MH samples chain")
    ax[index, 1].plot(np.arange(len(bike_chain_1[mc_convergence_iterations:][:, index//2])), bike_chain_1[mc_convergenc

    ax[index + 1, 1].set_title(f"{param} pymc samples chain")
    ax[index + 1, 1].plot(np.arange(len(idata.posterior[param][0])), idata.posterior[param][0], alpha=0.8, linestyle='-

plt.show()

Markdown(fr"""

#### Vemos que las distribuciones entre metropilis hasting y pymc se parecen mucho, tanto el histograma como la cadenas

""")
```



beta_0 MH samples histogram
beta_0 MH samples chain
beta_0 pymc samples histogram
beta_0 pymc samples chain
beta_1 MH samples histogram
beta_1 MH samples chain
beta_1 pymc samples histogram
beta_1 pymc samples chain


120 130 140 150 160
sigma MH samples histogram
160 150 140 130 120
0 1000 2000 3000 4000 5000
sigma MH samples chain
1400 1450 1500 1550 1600 1650
sigma pymc samples histogram
1650 1600 1550 1500 1450 1400
0 500 1000 1500 2000 2500 3000 3500 4000
sigma pymc samples chain
1400 1450 1500 1550 1600 1650
1650 1600 1550 1500 1450 1400
0 1000 2000 3000 4000 5000

Out[293…

Vemos que las distribuciones entre metropilis hasting y pymc se parecen mucho, tanto el histograma como la cadenas de samples, lo que podemos notar es que las cadenas de MH parecen tener un mayor indice de correlacion que las de pymc y vemos que se pueden llegar a afectar las predicciones ya que los histogramas no son tan 'suaves'

C) Compute la respuesta al item G) del Ejercicio 2. Coinciden estas respuestas?

Cual es la probabilidad de que un dia de 20 grados tenga mas de 6000 alquileres de bicicletas?

In [294…

```
# obtenemos 5000 samples de la posterior
bikes_5000_posteriors = bike_final_chain[np.random.choice(bike_final_chain.shape[0], 5000, replace=False)]

MH_days_with_more_than_6000_rides = 0
pymc_days_with_more_than_6000_rides = 0

for k in range(len(bikes_5000_posteriors)):
    posterior_sample = bikes_5000_posteriors[k]
    temperature_predicted = np.random.normal(posterior_sample[0] + posterior_sample[1] * 20, posterior_sample[2], 1)
    if temperature_predicted > 6000:
        MH_days_with_more_than_6000_rides += 1

    temperature_predicted = np.random.normal(idata.posterior["beta_0"][0][k].item() + idata.posterior["beta_1"][0][k].i
    if temperature_predicted > 6000:
        pymc_days_with_more_than_6000_rides += 1


Markdown(fr"""

### MH: La probabilidad de que el tiempo de alquiler de bicicletas sea mayor a 6000 es de {MH_days_with_more_than_6000_
### pymc: La probabilidad de que el tiempo de alquiler de bicicletas sea mayor a 6000 es de {pymc_days_with_more_than_6

#### Vemos que hay una diferencia insignificante entre los valores predichos por MH y pymc, podriamos usar cualquiera d

""")
```

Out[294…

MH: La probabilidad de que el tiempo de alquiler de bicicletas sea mayor a 6000 es de 0.292.

pymc: La probabilidad de que el tiempo de alquiler de bicicletas sea mayor a 6000 es de 0.2984.

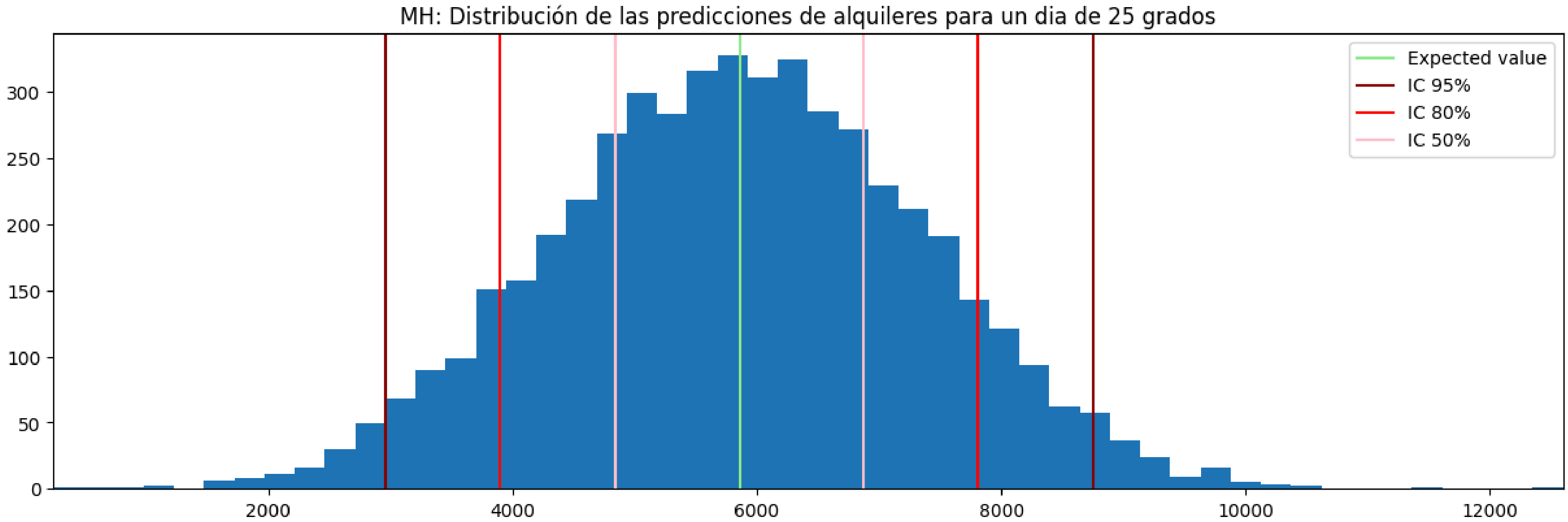
Vemos que hay una diferencia insignificante entre los valores predichos por MH y pymc, podriamos usar cualquiera de los 2 para hacer predicciones aunque como vimos en el punto anterior pymc parece hacer un mejor trabajo en generar cadenas independientes

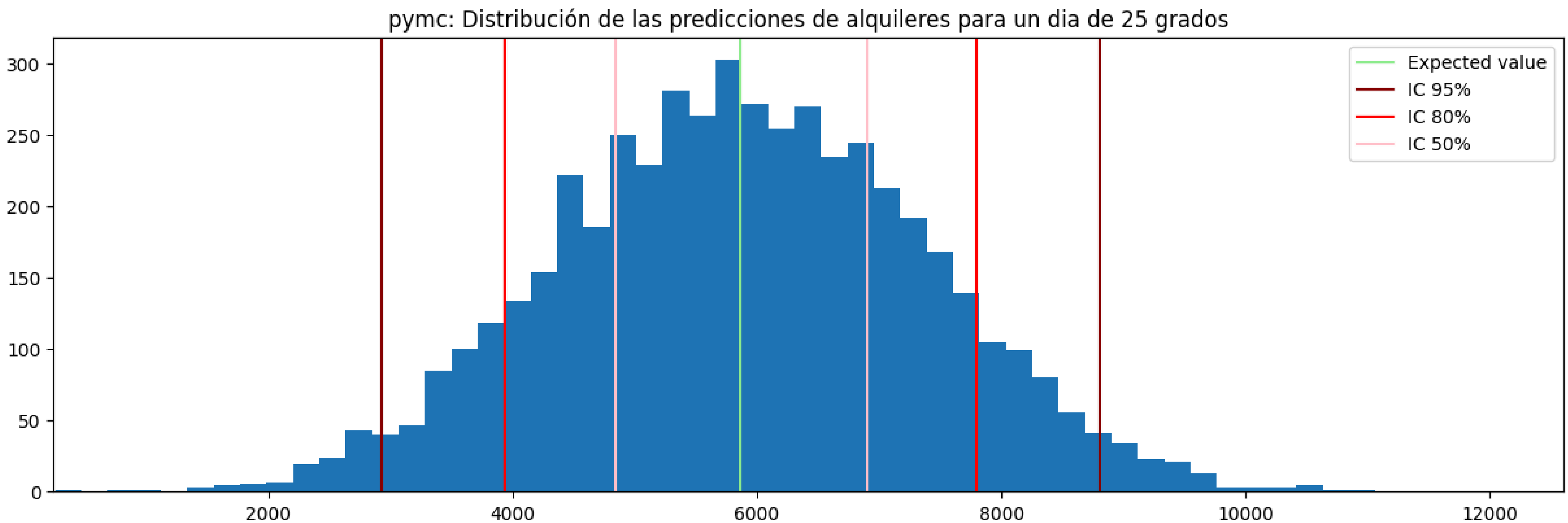
Cuantos alquileres de bicis voy a tener en un dia de 25 grados?

In [295…

```
MH_temperatures_predicted = np.array([])
pymc_temperatures_predicted = np.array([])


for k in range(len(bikes_5000_posteriors)):
    posterior_sample = bikes_5000_posteriors[k]
    temperature_predicted = np.random.normal(posterior_sample[0] + posterior_sample[1] * 25, posterior_sample[2], 1)
    MH_temperatures_predicted = np.append(MH_temperatures_predicted, temperature_predicted)

    temperature_predicted = np.random.normal(idata.posterior["beta_0"][0][k].item() + idata.posterior["beta_1"][0][k].i
    pymc_temperatures_predicted = np.append(pymc_temperatures_predicted, temperature_predicted)


min_value = min(np.min(MH_temperatures_predicted), np.min(pymc_temperatures_predicted))
max_value = max(np.max(MH_temperatures_predicted), np.max(pymc_temperatures_predicted))


_, ax = plt.subplots(2, 1, figsize=(15, 10))

for i, (temperatures_predicted, name) in enumerate([(MH_temperatures_predicted, "MH"), (pymc_temperatures_predicted, "p
    ax[i].set_title(f'{name}: Distribución de las predicciones de alquileres para un dia de 25 grados')
    ax[i].set_xlim(min_value, max_value)

    ax[i].hist(temperatures_predicted, bins=50)

    percentil_50 = np.percentile(temperatures_predicted, 50)
    percentil_2_5 = np.percentile(temperatures_predicted, 2.5)
    percentil_10 = np.percentile(temperatures_predicted, 10)
    percentil_25 = np.percentile(temperatures_predicted, 25)
    percentil_75 = np.percentile(temperatures_predicted, 75)
    percentil_90 = np.percentile(temperatures_predicted, 90)
    percentil_97_5 = np.percentile(temperatures_predicted, 97.5)
```

```
    ax[i].axvline(percentil_50, color = 'lightgreen', label = 'Expected value')

    ax[i].axvline(percentil_2_5, color = 'darkred')
    ax[i].axvline(percentil_97_5, color = 'darkred', label = 'IC 95%')

    ax[i].axvline(percentil_10, color = 'red')
    ax[i].axvline(percentil_90, color = 'red', label = 'IC 80%')

    ax[i].axvline(percentil_25, color = 'pink')
    ax[i].axvline(percentil_75, color = 'pink', label = 'IC 50%')

    ax[i].legend()

plt.show()


Markdown(fr"""

#### Vemos que hay una diferencia insignificante entre los valores predichos por MH y pymc, podriamos usar cualquiera c

""")
```





Out[295…

Vemos que hay una diferencia insignificante entre los valores predichos por MH y pymc, podriamos usar cualquiera de los 2 para hacer predicciones aunque como vimos en el punto anterior pymc parece hacer un mejor trabajo en generar cadenas independientes