```
import numpy as np
from scipy.special import factorial
import pandas as pd
import matplotlib.pyplot as plt
from math import e
import pymc3 as pm
import theano.tensor as T
import graphviz
import arviz as az


url = 'https://raw.githubusercontent.com/felipeisj/info274-2021/main/billonarios.cs
datos = pd.read_csv(url)


x = datos[['logpibpc', 'logpob', 'gatt']].to_numpy()
y = datos[['nbillonarios']].to_numpy()
```

Definición del modelo

```
with pm.Model() as funcion_poisson:

    # define priors, weakly informative Normal
    b0 = pm.Normal('theta0', mu=0, sd=10, shape=())
    b1 = pm.Normal("logpibpc", mu=0, sd=8, shape=())
    b2 = pm.Normal("logpob", mu=0, sd=15, shape=())
    b3 = pm.Normal("gatt", mu=0, sd=25, shape=())

    # define linear model and exp link function
    theta = (
        b0
        + b1 * datos["logpibpc"]
        + b2 * datos["logpob"]
        + b3 * datos["gatt"]
    )

    #Variable Determinista
    mu = pm.Deterministic("mu", theta)

    ## Verosimilitud de Poisson
    poiss = pm.Poisson("poiss", mu=np.exp(mu), observed=y.T[0])
```

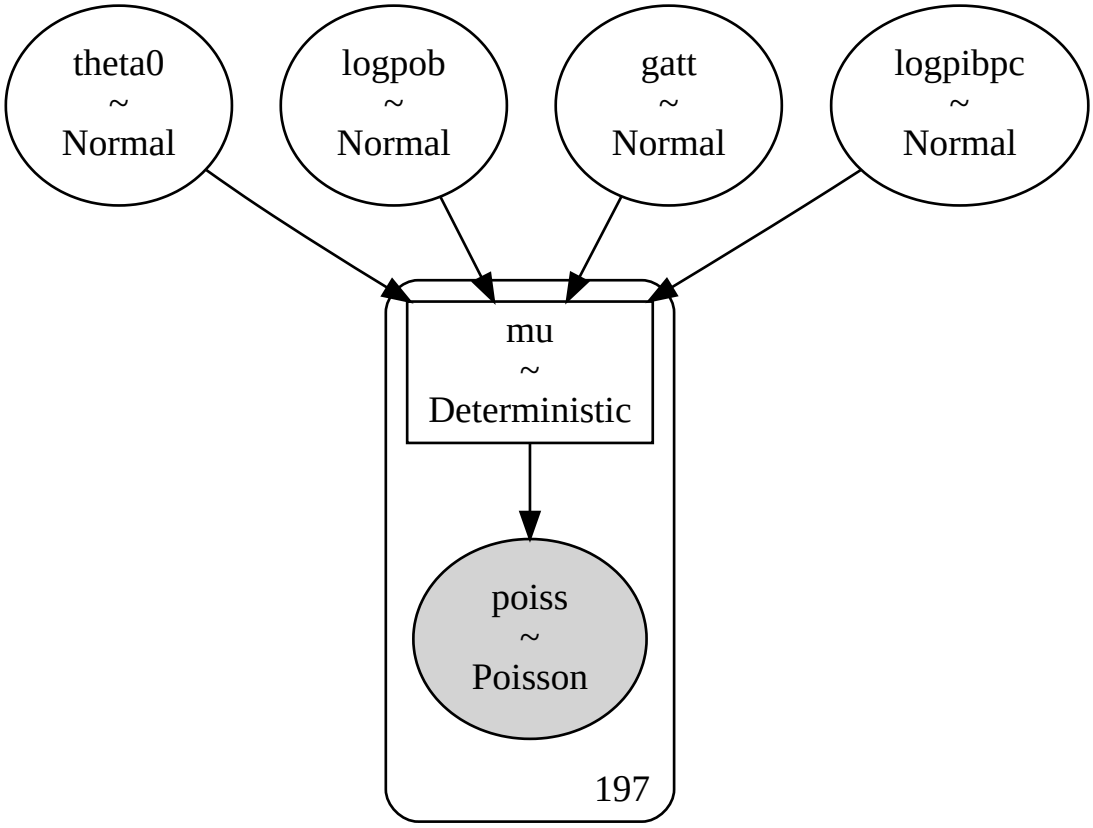
Despliegue de forma del modelo

```
funcion_poisson
```

$$
\begin{aligned}
\text{theta0} &\sim \text{Normal} \\
\text{logpibpc} &\sim \text{Normal} \\
\text{logpob} &\sim \text{Normal} \\
\text{gatt} &\sim \text{Normal} \\
\text{mu} &\sim \text{Deterministic}
\end{aligned}
$$

```
# Printea los Priors
display(funcion_poisson.free_RVs)
# Printea variable determinista
display(funcion_poisson.deterministics)
# Printea variable observada
display(funcion_poisson.observed_RVs)
pm.model_to_graphviz(funcion_poisson)
```

```
[theta0 ~ Normal, logpibpc ~ Normal, logpob ~ Normal, gatt ~ Normal]
[mu ~ Deterministic]
[poiss ~ Poisson]
```




```
with funcion_poisson:
  prior_checks = pm.sample_prior_predictive(samples = 100, var_names = ['theta0',
```

```
(prior_checks['logpibpc'])
```

```
array([ 4.89292699e+00, -7.48802999e+00, -1.01285582e+00, -2.05140822e-01,
       -4.03246856e+00, -2.63514397e+00, -3.82147609e+00,  1.01389473e+01,
        6.29476649e+00,  4.88007460e+00,  4.45859811e+00,  6.41965702e-01,
        1.21151232e+01,  7.28558004e+00,  3.37542481e-01, -9.86791571e-01,
        4.08657862e+00, -1.04298085e+00, -1.51245611e+01, -3.08161976e+00,
        1.22288895e+01, -4.29374917e+00,  8.85405583e+00,  5.35767913e+00,
       -4.32694911e+00, -7.76501255e+00, -7.50164057e+00,  3.98656502e+00,
       -3.14647420e+00,  7.53521971e+00, -7.49221125e+00, -1.12466924e+01,
        1.41722805e+01, -1.18187508e+01,  8.85933206e+00, -1.18279691e+01,
```

```
 1.07584362e+01,  2.26493723e+00, -1.05600583e+01, -4.93239601e+00,
 1.90669438e+01, -2.60016436e+00, -1.06555625e+01,  1.00895532e+01,
 8.41112023e+00,  5.72749657e+00,  1.23087967e+01, -7.79826885e+00,
 6.61688109e+00,  1.52713566e+00, -1.81984583e+01,  3.42466067e+00,
 1.31582253e+00,  1.29726733e+00,  6.22856665e+00,  3.78337454e+00,
-1.81037930e+00,  9.35569829e+00,  3.54929254e+00,  5.50519341e+00,
-5.64385988e+00, -1.28584558e+00,  9.43357029e+00,  4.66267279e+00,
-3.23181125e+00, -1.90537033e+00,  5.53935412e-01, -3.75379181e+00,
 6.14826741e+00, -1.80115075e-01, -8.98527803e+00, -5.38288980e+00,
 3.88029298e+00, -1.97517408e+00,  4.21932450e+00, -1.28469688e+01,
-2.38426396e-01, -4.22396286e+00,  9.35560163e+00, -4.75711204e+00,
-3.88820977e+00, -8.95189618e+00,  4.00163363e+00, -1.24568280e-02,
-2.52429076e+00,  1.13115720e+01,  4.61932815e+00, -3.97178926e+00,
-1.97173199e+01, -8.55092093e+00, -1.29900780e+01,  1.07800831e+01,
 4.46500491e+00,  6.44352919e-01,  4.13399827e+00, -1.69922885e+00,
-1.12636688e+01, -8.08678327e-01, -1.04750470e+01,  2.81184120e+00])
```

Entrenamiento con NUTS

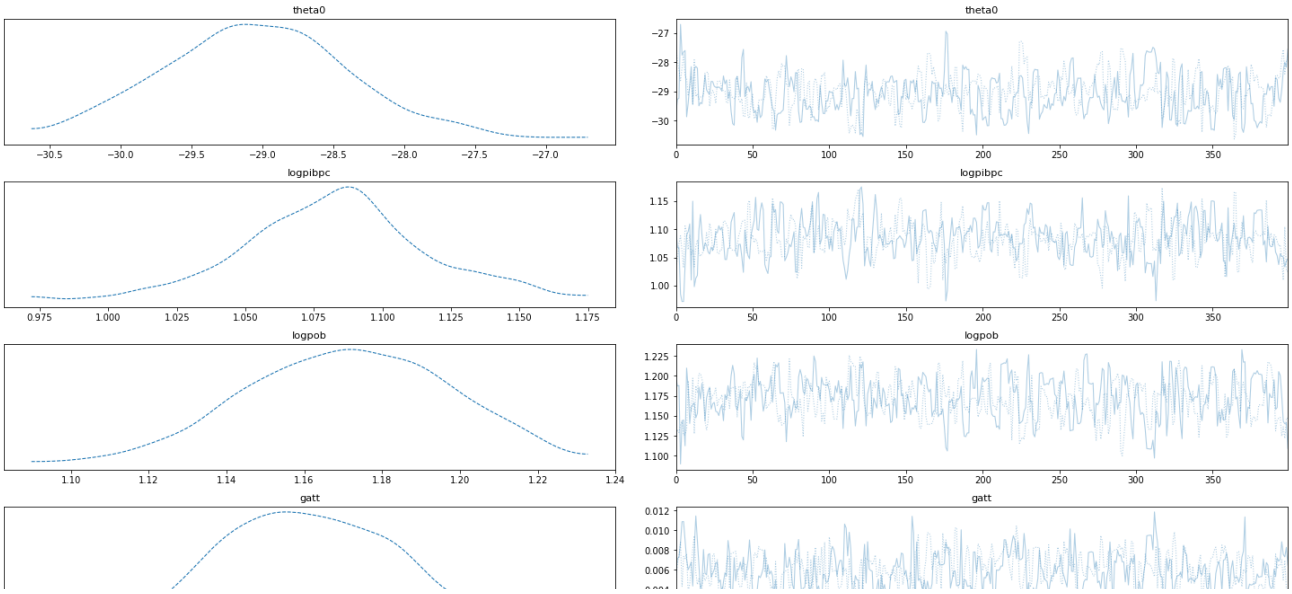
```
with funcion_poisson:
  trace = pm.sample(draws=400, tune=500, chains=2, cores=4, step=pm.NUTS(target_ac
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: FutureWarning

Only 400 samples in chain.
Multiprocess sampling (2 chains in 4 jobs)
NUTS: [gatt, logpob, logpibpc, theta0]
                                        100.00% [1800/1800 01:02<00:00 Sampling 2 chains,
0 divergences]
Sampling 2 chains for 500 tune and 400 draw iterations (1_000 + 800 draws tot
```

Trazas de los parámetros

```
pm.traceplot(trace, figsize=(20, 10), var_names=['theta0', 'logpibpc', 'logpob', '
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: DeprecationWa
  """Entry point for launching an IPython kernel.
/usr/local/lib/python3.7/dist-packages/arviz/data/io_pymc3.py:100: FutureWarn
  FutureWarning,
```



Muestras efectivo:

Respecto a las muestras efectivo, el resultado que lo identifica ess_bulk demuestra ser confiable para medir la eficiencia de muestreo en la mayor parte de la distribución, ya que, sus valores no descienden de 200.

Gelman-Rubin:

El diagnóstico de convergencia visto en el estadístico r_hat demuestra que las estimaciones entre y dentro de la cadena se han mezclado bien. Los resultados de r_hat varían entre 1.00 y 1.02.

Función de autocorrelación:

Los parámetros comienzan regular, sin embargo, tardan muy poco en ajustarse, por lo que tienden a lograr valores cercanos a la convergencia

```
pm.summary(trace, var_names=['theta0', 'logpibpc', 'logpob', 'gatt']).round(3)
```

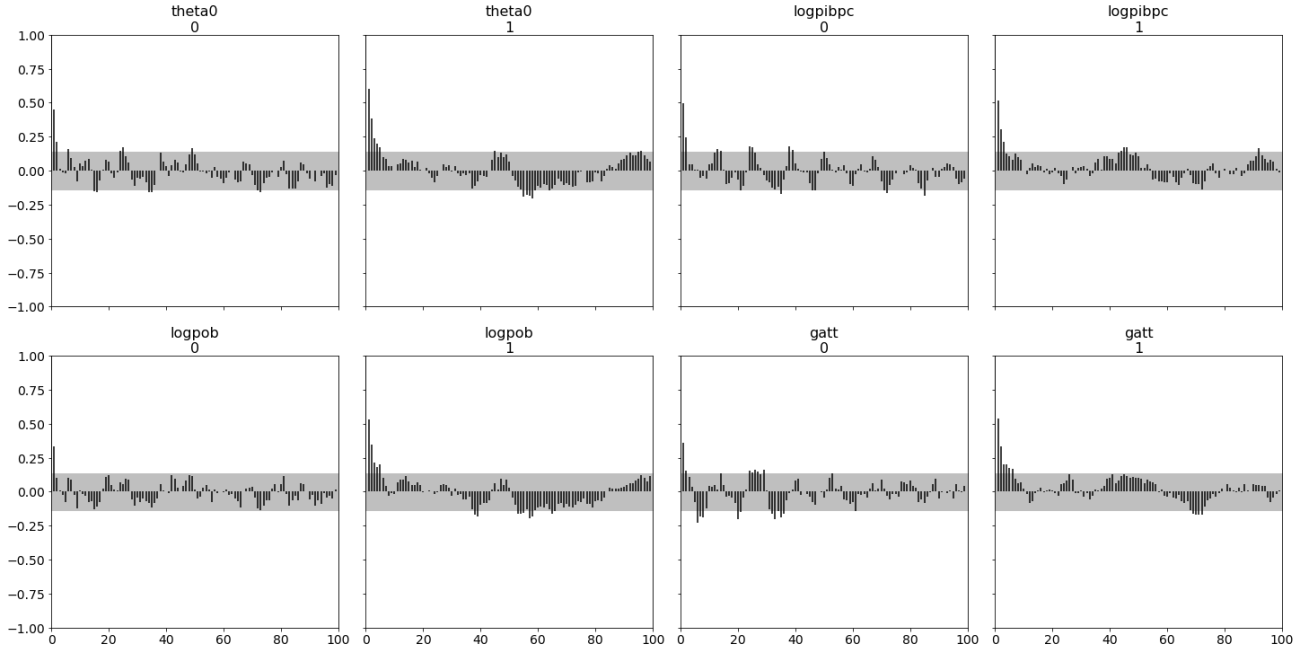
```
/usr/local/lib/python3.7/dist-packages/arviz/data/io_pymc3.py:100: FutureWarn
  FutureWarning,
```

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail
theta0	-29.041	0.655	-30.299	-27.815	0.040	0.029	271.0	245.0
logpibpc	1.083	0.034	1.023	1.154	0.002	0.001	327.0	328.0
logpob	1.171	0.026	1.126	1.219	0.002	0.001	291.0	343.0
gatt	0.006	0.002	0.003	0.010	0.000	0.000	321.0	386.0

```
pm.plots.autocorrplot(trace, figsize=(20, 10), var_names=['theta0', 'logpibpc', 'l
```

```
pm.plots.autocorrplot(trace, figsize=(20, 10), var_names=["theta0", "logpibpc", "t
```

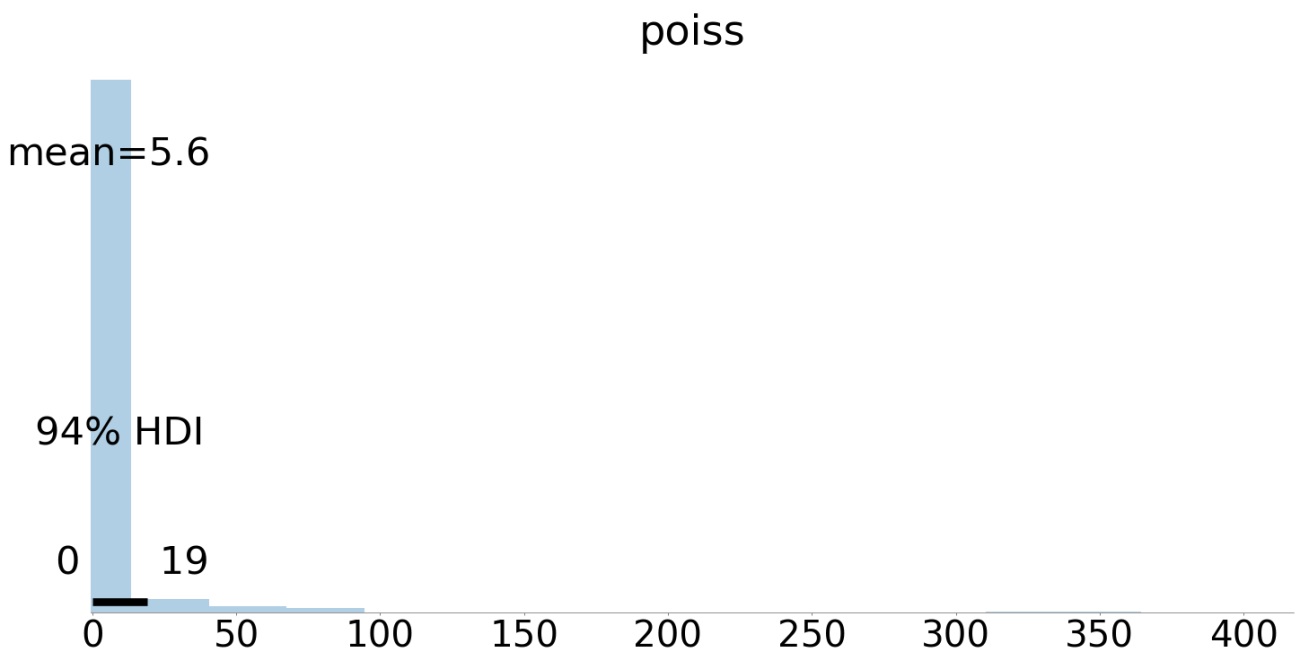
```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: DeprecationWa
  """Entry point for launching an IPython kernel.
/usr/local/lib/python3.7/dist-packages/arviz/data/io_pymc3.py:100: FutureWarn
  FutureWarning,
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f5885ceef50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5885a7f590>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5885a8ed90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5885b74dd0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f58860b3f50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f58860e8d90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5885e23690>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f58861198d0>]],
      dtype=object)
```




```
with funcion_poisson:
    posterior_predictive = pm.sample_posterior_predictive(trace, samples=100, var_
prediccion = posterior_predictive["poiss"][0]
```

```
/usr/local/lib/python3.7/dist-packages/pymc3/sampling.py:1690: UserWarning: s
  "samples parameter is smaller than nchains times ndraws, some draws "
```

```
pm.plot_posterior(posterior_predictive, figsize=(20, 10), var_names=['poiss']);
```



Entrenamiento con Metropolis

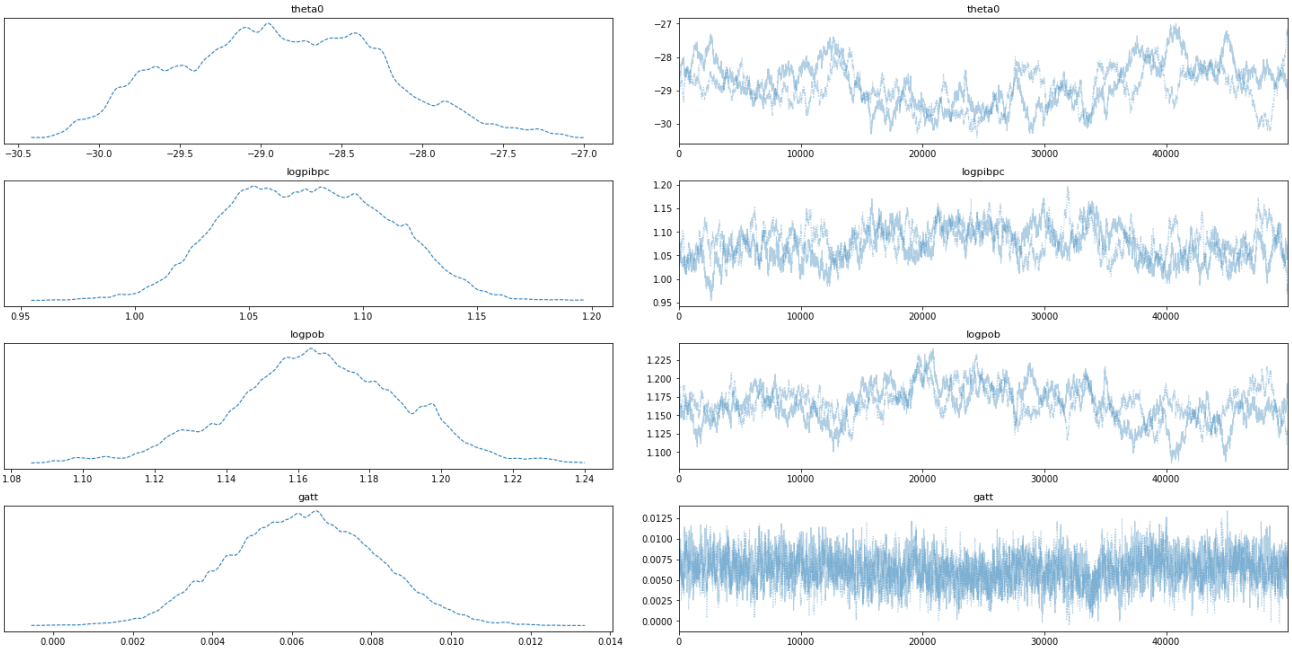
```
with funcion_poisson:
  trace_metropolis = pm.sample(draws=50000, tune=6000, chains=2, cores=4, step=pm.M
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: FutureWarning

Multiprocess sampling (2 chains in 4 jobs)
CompoundStep
>Metropolis: [gatt]
>Metropolis: [logpob]
```

Trazas de los parámetros utilizando Metropolis

```
100.00% [112000/112000 01:38<00:00 Sampling 2
```

```
pm.traceplot(trace_metropolis, figsize=(20, 10), var_names=['theta0', 'logpibpc',

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: DeprecationWa
  """Entry point for launching an IPython kernel.
/usr/local/lib/python3.7/dist-packages/arviz/data/io_pymc3.py:100: FutureWarn
  FutureWarning,
```



Muestras efectivo:

Respecto a las muestras efectivo, el resultado que lo identifica ess_bulk demuestra no ser confiable para medir la eficiencia de muestreo en la mayor parte de la distribución, ya que, sus

valores varían entre 34 y 86 para las variables theta0, logpicpb, logpob, sin embargo, la variable gatt demuestra ser confiable al alcanzar un resultado superior a 400. La hipótesis sostenida es que el hiperparámetro "tune" es muy alto, ya que se desean ignorar valores atípicos registrados al modificar hiperparámetros.

Gelman-Rubin:

El diagnóstico de convergencia visto en el estadístico r_hat demuestra que las estimaciones entre y dentro de la cadena se han mezclado bien. Los resultados de r_hat varían entre 1.00 y 1.03.

Función de autocorrelación:

Se puede decir que las variables son altamente dependientes y a medida que el algoritmo avanza, no se logra la convergencia.

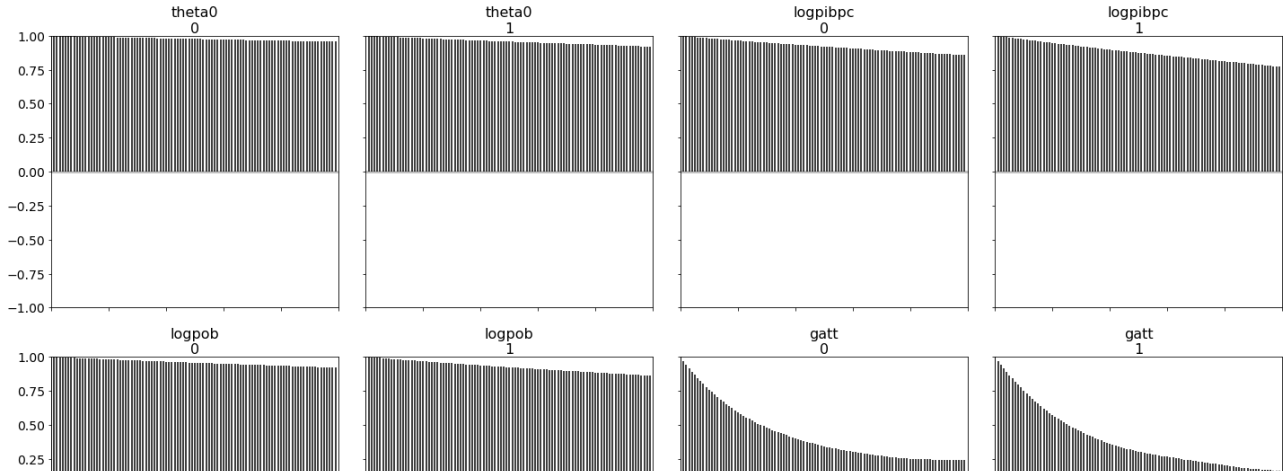
```
pm.summary(trace_metropolis, var_names=['theta0', 'logpibpc', 'logpob', 'gatt']).r
```

```
/usr/local/lib/python3.7/dist-packages/arviz/data/io_pymc3.py:100: FutureWarn
  FutureWarning,
```

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail
theta0	-28.872	0.627	-30.009	-27.743	0.126	0.090	25.0	115.0
logpibpc	1.075	0.035	1.014	1.139	0.006	0.004	35.0	207.0
logpob	1.166	0.024	1.120	1.208	0.004	0.003	30.0	133.0
gatt	0.006	0.002	0.003	0.010	0.000	0.000	252.0	1659.0

```
pm.plots.autocorrplot(trace_metropolis, figsize=(20, 10), var_names=['theta0', 'lo
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: DeprecationWa
  """Entry point for launching an IPython kernel.
/usr/local/lib/python3.7/dist-packages/arviz/data/io_pymc3.py:100: FutureWarn
  FutureWarning,
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f58905e4950>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f58a0c06c90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f58903be0d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f58a0baf110>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f58a0befe50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f5890421e10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f58a0e4add0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f589d949b50>]],
      dtype=object)
```




```
with funcion_poisson:
    posterior_predictive_2 = pm.sample_posterior_predictive(trace_metropolis, samp
prediccion = posterior_predictive_2["poiss"][0]
```

```
/usr/local/lib/python3.7/dist-packages/pymc3/sampling.py:1690: UserWarning: s
  "samples parameter is smaller than nchains times ndraws, some draws "
```

100.00% [100/100 00:00<00:00]

```
pm.plot_posterior(posterior_predictive_2, figsize=(10, 5), var_names=['poiss']);
```

poiss

Medianas y percentiles

```
print("mediana de poiss: ",np.median(posterior_predictive["poiss"][0]))
print("mediana de mu: ", np.median(posterior_predictive["mu"]))
print("mediana de logpibpc: ",np.median(posterior_predictive["logpibpc"]))
print("mediana de logpob: ", np.median(posterior_predictive["logpob"]))
print("mediana de gatt: ", np.median(posterior_predictive["gatt"]))
```

```
mediana de poiss:  0.0
mediana de mu:  -1.8133208830816745
mediana de logpibpc:  1.0769010330609154
mediana de logpob:  1.165881641446438
mediana de gatt:  0.006057868226411718
```

Los Parámetros obtenidos con mediana significativamente distinto de cero son logpibpc y logpob. Esto está dado en base a la gran diferencia que existe entre los ingresos de países ricos y pobres.

- Prediga la cantidad de billonarios usando su modelo y la incertidumbre asociada (posterior predictivo). Muestre graficamente sus resultados
- ¿Cuáles son los 5 países con mayor error en la predicción? ¿Cuáles países tienen un exceso de billonarios? ¿Cúales paises tienen menos billonarios de lo esperado? ¿Qué puede decir sobre Rusia?

```
#Predicción de billonarios por país
for j in range(0, 196):
  print("País :", datos["pais"][j], "predicción de billonarios", prediccion[j])
#prediccion
```

```
País : Suriname predicción de billonarios 0
País : Ecuador predicción de billonarios 2
País : Peru predicción de billonarios 1
País : Brazil predicción de billonarios 36
País : Bolivia predicción de billonarios 0
País : Paraguay predicción de billonarios 0
País : Chile predicción de billonarios 5
País : Argentina predicción de billonarios 4
País : Uruguay predicción de billonarios 0
País : Faeroe Islands predicción de billonarios 0
País : United Kingdom predicción de billonarios 54
País : Ireland predicción de billonarios 3
País : Netherlands predicción de billonarios 8
País : Belgium predicción de billonarios 6
País : Luxembourg predicción de billonarios 0
País : France predicción de billonarios 54
País : Monaco predicción de billonarios 0
País : Liechtenstein predicción de billonarios 0
País : Switzerland predicción de billonarios 6
País : Spain predicción de billonarios 24
País : Andorra predicción de billonarios 0
```

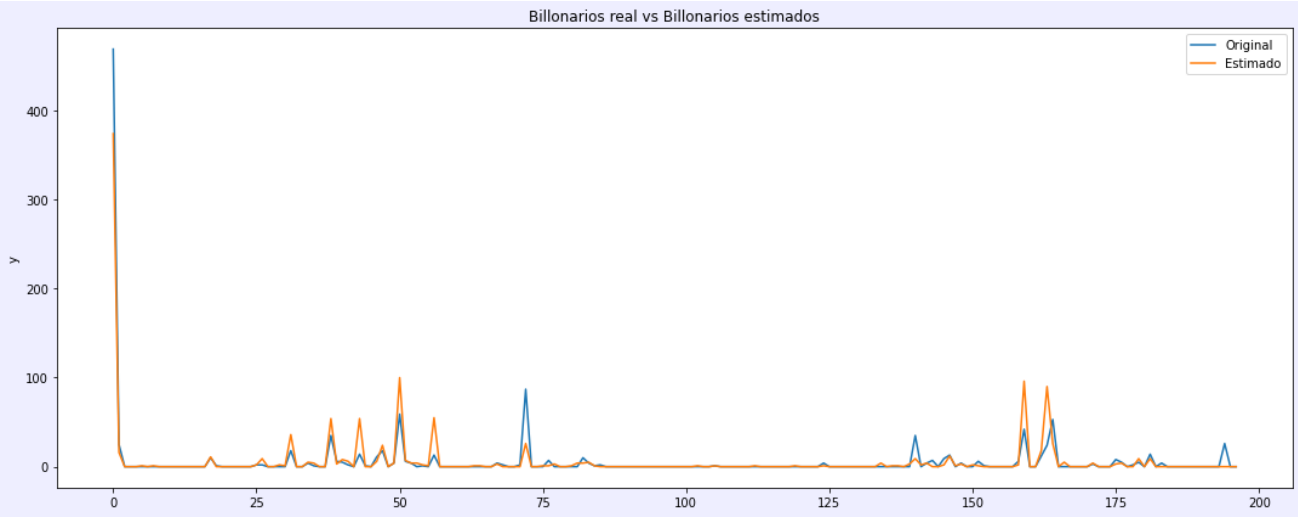
```
País : Andorra predicción de billonarios 0
País : Portugal predicción de billonarios 4
País : Germany predicción de billonarios 100
País : Poland predicción de billonarios 7
País : Austria predicción de billonarios 4
País : Hungary predicción de billonarios 4
País : Czech Republic predicción de billonarios 2
País : Slovak Republic predicción de billonarios 1
País : Italy predicción de billonarios 55
País : San Marino predicción de billonarios 0
País : Malta predicción de billonarios 0
País : Albania predicción de billonarios 0
País : Montenegro predicción de billonarios 0
País : Serbia predicción de billonarios 0
País : Macedonia, FYR predicción de billonarios 0
País : Croatia predicción de billonarios 1
País : Bosnia and Herzegovina predicción de billonarios 1
País : Kosovo predicción de billonarios 0
País : Slovenia predicción de billonarios 0
País : Greece predicción de billonarios 3
País : Cyprus predicción de billonarios 0
País : Bulgaria predicción de billonarios 0
País : Moldova predicción de billonarios 0
País : Romania predicción de billonarios 0
País : Russian Federation predicción de billonarios 26
País : Estonia predicción de billonarios 0
País : Latvia predicción de billonarios 0
País : Lithuania predicción de billonarios 1
País : Ukraine predicción de billonarios 1
País : Belarus predicción de billonarios 3
País : Armenia predicción de billonarios 0
País : Georgia predicción de billonarios 0
País : Azerbaijan predicción de billonarios 1
País : Finland predicción de billonarios 4
País : Sweden predicción de billonarios 4
País : Norway predicción de billonarios 5
País : Denmark predicción de billonarios 1
País : Iceland predicción de billonarios 0
País : Greenland predicción de billonarios 0
País : Cabo Verde predicción de billonarios 0
```

```
#Países con mayor error de predicción
error = abs(prediccion - y.T)
for i in range(197):
  if (error[0][i] > 41):
    print("error de billonarios:", error[0][i],"país: ", datos["pais"][i])
```

```
error de billonarios: 95 país:  United States
error de billonarios: 42 país:  Italy
error de billonarios: 61 país:  Russian Federation
error de billonarios: 54 país:  China
error de billonarios: 66 país:  Japan
```

```
fig, ax = plt.subplots(1, 1, figsize=(15, 6), tight_layout=True, facecolor='#EEF')
ax.plot(y, label = "Original")
ax.plot(prediccion, label = "Estimado")
ax.set_ylabel('y')
ax.set_title("Billonarios real vs Billonarios estimados")
ax.legend()
```

```
<matplotlib.legend.Legend at 0x7f5885e97c10>
```



Países con exceso de billonarios

```
error = (posterior_predictive["poiss"][0] - y.T)


print("Paises con prediccion excesiva: ")
print("-------------------------------------------------------------------------
for i in range(197):
  if (error[0][i] > 0):
    print("error: ", error[0][i], datos["pais"][i], " /// Cantidad real: ", datos[
    #abs(posterior_predictive["poiss"][0] - y.T)/100
    porcentaje_error = (error[0][i]*(100)/(datos["nbillonarios"][i]))
    print("Porcentaje de error: ", porcentaje_error)
```

```
error:  1 Poland  /// Cantidad real:  6  Cantidad predicha:  7
Porcentaje de error:  16.666666666666668
error:  4 Hungary  /// Cantidad real:  0  Cantidad predicha:  4
Porcentaje de error:  inf
error:  1 Czech Republic  /// Cantidad real:  1  Cantidad predicha:  2
Porcentaje de error:  100.0
error:  1 Slovak Republic  /// Cantidad real:  0  Cantidad predicha:  1
Porcentaje de error:  inf
error:  42 Italy  /// Cantidad real:  13  Cantidad predicha:  55
Porcentaje de error:  323.0769230769231
error:  1 Croatia  /// Cantidad real:  0  Cantidad predicha:  1
Porcentaje de error:  inf
```

```
error:  1 Bosnia and Herzegovina  /// Cantidad real:  0  Cantidad predicha:
Porcentaje de error:  inf
error:  1 Lithuania  /// Cantidad real:  0  Cantidad predicha:  1
Porcentaje de error:  inf
error:  3 Belarus  /// Cantidad real:  0  Cantidad predicha:  3
Porcentaje de error:  inf
error:  1 Azerbaijan  /// Cantidad real:  0  Cantidad predicha:  1
Porcentaje de error:  inf
error:  4 Finland  /// Cantidad real:  0  Cantidad predicha:  4
Porcentaje de error:  inf
error:  1 Norway  /// Cantidad real:  4  Cantidad predicha:  5
Porcentaje de error:  25.0
error:  1 Ghana  /// Cantidad real:  0  Cantidad predicha:  1
Porcentaje de error:  inf
error:  1 Kenya  /// Cantidad real:  0  Cantidad predicha:  1
Porcentaje de error:  inf
error:  1 Angola  /// Cantidad real:  0  Cantidad predicha:  1
Porcentaje de error:  inf
error:  4 Algeria  /// Cantidad real:  0  Cantidad predicha:  4
Porcentaje de error:  inf
error:  1 Libya  /// Cantidad real:  0  Cantidad predicha:  1
Porcentaje de error:  inf
error:  1 Sudan  /// Cantidad real:  0  Cantidad predicha:  1
Porcentaje de error:  inf
error:  3 Iran, Islamic Rep.  /// Cantidad real:  0  Cantidad predicha:  3
Porcentaje de error:  inf
error:  2 Iraq  /// Cantidad real:  0  Cantidad predicha:  2
Porcentaje de error:  inf
error:  1 Yemen, Rep.  /// Cantidad real:  0  Cantidad predicha:  1
Porcentaje de error:  inf
error:  2 Qatar  /// Cantidad real:  0  Cantidad predicha:  2
Porcentaje de error:  inf
error:  54 China  /// Cantidad real:  42  Cantidad predicha:  96
Porcentaje de error:  128.57142857142858
error:  6 Korea, Rep.  /// Cantidad real:  12  Cantidad predicha:  18
Porcentaje de error:  50.0
error:  66 Japan  /// Cantidad real:  24  Cantidad predicha:  90
Porcentaje de error:  275.0
error:  5 Pakistan  /// Cantidad real:  0  Cantidad predicha:  5
Porcentaje de error:  inf
error:  1 Thailand  /// Cantidad real:  3  Cantidad predicha:  4
Porcentaje de error:  33.333333333333336
error:  4 Indonesia  /// Cantidad real:  5  Cantidad predicha:  9
Porcentaje de error:  80.0
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: RuntimeWarn
  import sys
```

Países con escasez de billonarios

```
# Paises con prediccion negativa:
print("Paises con prediccion negativa:")
for i in range(197):
  if (error[0][i] < 0):
    print("error: ", error[0][i], datos["pais"][i], " /// Cantidad real: ", datos[
    #abs(posterior_predictive["poiss"][0] - y.T)/100
    porcentaje_error = (error[0][i]*(100)/(datos["nbillonarios"][i]))
    print("Porcentaje de error: ", porcentaje_error)
```

```
print("Porcentaje de error: ", porcentaje_error)
```

```
Paises con prediccion negativa:
error:  -95 United States  /// Cantidad real:  469  Cantidad predicha:  374
Porcentaje de error:  -20.255863539445627
error:  -9 Canada  /// Cantidad real:  25  Cantidad predicha:  16
Porcentaje de error:  -36.0
error:  -1 Belize  /// Cantidad real:  1  Cantidad predicha:  0
Porcentaje de error:  -100.0
error:  -3 Ireland  /// Cantidad real:  6  Cantidad predicha:  3
Porcentaje de error:  -50.0
error:  -1 Monaco  /// Cantidad real:  1  Cantidad predicha:  0
Porcentaje de error:  -100.0
error:  -5 Switzerland  /// Cantidad real:  11  Cantidad predicha:  6
Porcentaje de error:  -45.45454545454545
error:  -1 Greece  /// Cantidad real:  4  Cantidad predicha:  3
Porcentaje de error:  -25.0
error:  -2 Cyprus  /// Cantidad real:  2  Cantidad predicha:  0
Porcentaje de error:  -100.0
error:  -2 Romania  /// Cantidad real:  2  Cantidad predicha:  0
Porcentaje de error:  -100.0
error:  -61 Russian Federation  /// Cantidad real:  87  Cantidad predicha:
Porcentaje de error:  -70.11494252873563
error:  -6 Ukraine  /// Cantidad real:  7  Cantidad predicha:  1
Porcentaje de error:  -85.71428571428571
error:  -6 Sweden  /// Cantidad real:  10  Cantidad predicha:  4
Porcentaje de error:  -60.0
error:  -2 Iceland  /// Cantidad real:  2  Cantidad predicha:  0
Porcentaje de error:  -100.0
error:  -3 South Africa  /// Cantidad real:  4  Cantidad predicha:  1
Porcentaje de error:  -75.0
error:  -26 Turkey  /// Cantidad real:  35  Cantidad predicha:  9
Porcentaje de error:  -74.28571428571429
error:  -7 Lebanon  /// Cantidad real:  7  Cantidad predicha:  0
Porcentaje de error:  -100.0
error:  -7 Israel  /// Cantidad real:  9  Cantidad predicha:  2
Porcentaje de error:  -77.77777777777777
error:  -1 Saudi Arabia  /// Cantidad real:  13  Cantidad predicha:  12
Porcentaje de error:  -7.6923076923076925
error:  -1 Kuwait  /// Cantidad real:  4  Cantidad predicha:  3
Porcentaje de error:  -25.0
error:  -5 United Arab Emirates  /// Cantidad real:  6  Cantidad predicha:
Porcentaje de error:  -83.33333333333333
error:  -1 Oman  /// Cantidad real:  1  Cantidad predicha:  0
Porcentaje de error:  -100.0
error:  -4 Kazakhstan  /// Cantidad real:  6  Cantidad predicha:  2
Porcentaje de error:  -66.66666666666667
error:  -27 India  /// Cantidad real:  53  Cantidad predicha:  26
Porcentaje de error:  -50.943396226415096
error:  -5 Malaysia  /// Cantidad real:  8  Cantidad predicha:  3
Porcentaje de error:  -62.5
error:  -1 Singapore  /// Cantidad real:  5  Cantidad predicha:  4
Porcentaje de error:  -20.0
error:  -2 Philippines  /// Cantidad real:  2  Cantidad predicha:  0
Porcentaje de error:  -100.0
error:  -5 Australia  /// Cantidad real:  14  Cantidad predicha:  9
Porcentaje de error:  -35.714285714285715
error:  -4 New Zealand  /// Cantidad real:  4  Cantidad predicha:  0
Porcentaje de error:  -100.0
```

Rusia es el país mas afectado, al poseer una cantidad de billonarios igual a 87, sin embargo, se predijo solo 26, lo que se traduce en una diferencia de ~300% de disminución de billonarios. Esta información nos permite concluir que existen otros parámetros además de pib, pob y gatt que pueden influir en la cantidad de billonarios de un país.

✓ 0 s completado a las 22:29