Question 1. (25 points) In the Laffer data (*Data/Laffer.csv*, first introduced in the accompanying prediction tutorial), there is one country with a high tax revenue that is an outlier.

In the tutorial, PSIS and WAIC were used to measure the importance of this outlier in the models that were fit.

First, **re-create and run the models provided as part of the prediction tutorial**.

In [43]:
```
df = pd.read_csv("Data/Laffer.csv", delimiter=",")

df = pd.read_csv("Data/Laffer.csv", header=0)
# standardizing tax rate and tax revenue
tax_rate_std = standardize(df.tax_rate)
tax_revenue_std = standardize(df.tax_revenue)

# first-order linear model
with pm.Model() as m_ex4_lin:

    # prior on intercept for expected tax revenue: mean = 0, std = 0.2
    a = pm.Normal("a", 0, 0.2)

    # prior on coefficient for tax rate: mean=0, std=0.5
    b = pm.Normal("b", 0, 0.5)

    # expected tax revenue (using linear model)
    mu = pm.Deterministic("mu", a + b * tax_rate_std)

    # standard deviation for tax revenue
    sigma = pm.Exponential("sigma", 1)

    # data distribution for tax revenue
    R = pm.Normal("tax_revenue", mu, sigma, observed=tax_revenue_std)

    # using Hamiltonian Monte Carlo MCMC
    idata_ex4_lin = pm.sample(idata_kwargs={"log_likelihood": True})
# quadratic model
with pm.Model() as m_ex4_quad:
    # prior on intercept for expected tax revenue: mean = 0, std = 0.2
    a = pm.Normal("a", 0, 0.2)

    # prior on coefficient for tax rate: mean=0, std=0.5
    b = pm.Normal("b", 0, 0.5)

    # prior on coefficient for tax rate^2: mean=0, std=0.5
    b2 = pm.Normal("b2", 0, 0.5)

    # expected tax revenue (using polynomial model)
    mu = pm.Deterministic("mu", a + b * tax_rate_std + b2 * tax_rate_std**2)

    # standard deviation for tax revenue
    sigma = pm.Exponential("sigma", 1)
```

```
    # data distribution for tax revenue
    R = pm.Normal("tax_revenue", mu, sigma, observed=tax_revenue_std)

    # using Hamiltonian Monte Carlo MCMC
    idata_ex4_quad = pm.sample(idata_kwargs={"log_likelihood": True})
```

```
Auto-assigning NUTS sampler…
Initializing NUTS using jitter+adapt_diag…
Sequential sampling (2 chains in 1 job)
NUTS: [a, b, sigma]
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
Sampling 2 chains for 1_000 tune and 1_000 draw iterations (2_000 + 2_000 draws total) took 5 seconds.
We recommend running at least 4 chains for robust computation of convergence diagnostics
Auto-assigning NUTS sampler…
Initializing NUTS using jitter+adapt_diag…
Sequential sampling (2 chains in 1 job)
NUTS: [a, b, b2, sigma]
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
Sampling 2 chains for 1_000 tune and 1_000 draw iterations (2_000 + 2_000 draws total) took 7 seconds.
We recommend running at least 4 chains for robust computation of convergence diagnostics
```

Create a new **cubic** (3rd-degree polynomial) version of the model.

```
In [44]: # cubic model
         with pm.Model() as m_ex4_cubic:
             # prior on intercept for expected tax revenue: mean = 0, std = 0.2
             a = pm.Normal("a", 0, 0.2)

             # prior on coefficient for tax rate: mean=0, std=0.5
             b = pm.Normal("b", 0, 0.5)

             # prior on coefficient for tax rate^2: mean=0, std=0.5
             b2 = pm.Normal("b2", 0, 0.5)

             # prior on coefficient for tax rate^2: mean=0, std=0.5
             b3 = pm.Normal("b3", 0, 0.5)

             # expected tax revenue (using polynomial model)
             mu = pm.Deterministic("mu", a + b * tax_rate_std + b2 * tax_rate_std**2 + b3 * tax_rate_st

             # standard deviation for tax revenue
             sigma = pm.Exponential("sigma", 1)

             # data distribution for tax revenue
             R = pm.Normal("tax_revenue", mu, sigma, observed=tax_revenue_std)

             # using Hamiltonian Monte Carlo MCMC
             idata_ex4_cubic = pm.sample(idata_kwargs={"log_likelihood": True})
```

```
Auto-assigning NUTS sampler…
Initializing NUTS using jitter+adapt_diag…
Sequential sampling (2 chains in 1 job)
NUTS: [a, b, b2, b3, sigma]
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
Sampling 2 chains for 1_000 tune and 1_000 draw iterations (2_000 + 2_000 draws total) took 15 seconds.
We recommend running at least 4 chains for robust computation of convergence diagnostics
```

Create a new (first-order) **linear** version of the model using robust regression with a Student's t distribution for the data distribution. Use $\nu = 2$ for your Student's t distribution.

```
In [45]: # first-order linear model using robust regression with a Student's t distribution for the dat
         with pm.Model() as m_ex4_lin_robust:
             # Prior on intercept for expected tax revenue: mean = 0, std = 0.2
             a = pm.Normal("a", 0, 0.2)

             # Prior on coefficient for tax rate: mean=0, std=0.5
             b = pm.Normal("b", 0, 0.5)

             # Expected tax revenue (using linear model)
             mu = pm.Deterministic("mu", a + b * tax_rate_std)

             # Standard deviation for tax revenue
             sigma = pm.Exponential("sigma", 1)

             # Degrees of freedom
             nu = 2
             R = pm.StudentT("tax_revenue", nu=nu, mu=mu, sigma=sigma, observed=tax_revenue_std)

             # using Hamiltonian Monte Carlo MCMC
             idata_ex4_lin_robust = pm.sample(idata_kwargs={"log_likelihood": True})

         # Set up the PSIS comparison
         model_dict = {
             "m_ex4_lin": idata_ex4_lin,
             "m_ex4_quad": idata_ex4_quad,
             "m_ex4_cubic": idata_ex4_cubic,
             "m_ex4_lin_robust": idata_ex4_lin_robust
         }
         # Execute the PSIS comparison
         compare_df_psis = az.compare(
             compare_dict=model_dict,
             ic="loo",
             scale="deviance"
         )
```

```
Auto-assigning NUTS sampler…
Initializing NUTS using jitter+adapt_diag…
Sequential sampling (2 chains in 1 job)
NUTS: [a, b, sigma]
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
Sampling 2 chains for 1_000 tune and 1_000 draw iterations (2_000 + 2_000 draws total) took 5 seconds.
We recommend running at least 4 chains for robust computation of convergence diagnostics
/opt/conda/envs/fnds/lib/python3.9/site-packages/arviz/stats/stats.py:789: UserWarning: Estimated shape
  warnings.warn(
/opt/conda/envs/fnds/lib/python3.9/site-packages/arviz/stats/stats.py:789: UserWarning: Estimated shape
  warnings.warn(
/opt/conda/envs/fnds/lib/python3.9/site-packages/arviz/stats/stats.py:789: UserWarning: Estimated shape
  warnings.warn(
```

Of the 4 models, which one has the best predictive performance according to PSIS? **Limit your response to no more than 2 sentences.**

The linear version using robust regression has the highest rank (0) out of the 4 models. The value for elpd_loo is ~73.84 which indicates a better predictive performance compared to ~86 for the others, it generalizes data better.

Based on these results, what impact does the outlier point appear to be having on the curved relationship between tax rate and tax revenue? **Limit your response to no more than 5 sentences.**

Note: *You will need to quantify the influence of the outlier data point in your predictive models and comment on this influence in order to receive full credit.*

```
In [46]: print(compare_df_psis)
```

```
                rank   elpd_loo     p_loo  elpd_diff        weight  \
m_ex4_lin_robust   0  73.664599  3.123187   0.000000  7.789287e-01
m_ex4_quad         1  85.623707  5.161043  11.959108  2.210713e-01
m_ex4_lin          2  87.286892  5.070848  13.622293  4.906627e-16
m_ex4_cubic        3  87.353628  5.968676  13.689029  0.000000e+00

                       se        dse  warning     scale
m_ex4_lin_robust  12.261320   0.000000    False  deviance
m_ex4_quad        19.265196  11.671712     True  deviance
m_ex4_lin         18.992424  11.326380     True  deviance
m_ex4_cubic       19.592405  11.809895     True  deviance
```

For the quadratic linear (non-robust), and cubic models, the elpd_loo value is ~86 while the linear (robust) has an elpd_loo value of ~73.84. The linear robust model is influenced less by the outlier value in our data, making it a better fitting model. The others elpd_loo values indicate overfitting at a value of ~86, in addition to their se values being ~18 or 19, which is higher than the se value for linear (robust) se at a value of ~12.28.

Question 2. **(15 points)** Reconsider the urban fox analysis (*Data/foxes.csv*) from the previous homework assignment (HW 3).

On the basis of PSIS and WAIC scores, which combination of variables best predicts body weight (W, `weight`)? **Limit your response to no more than 3 sentences.**

Note: *For this exercise, you will end up creating 7 models.*

```
In [47]: df = pd.read_csv(
             "Data/foxes.csv",
             sep=',',
             header=0
         )
         # prior predictive simulation
         a_std = standardize(df.area)
         g_std = standardize(df.group)
         f_std = standardize(df.avgfood)
         gs_std = standardize(df.groupsize)
         w_std = standardize(df.weight)
         # Store models in a structure
         models = {}
         # Area model
         with pm.Model() as m_AREA:
             a = pm.Normal("a", 0.1, 0.2)
             b_a = pm.Normal("b_a", 0.1, 0.5)
             mu = pm.Deterministic("mu", a + b_a * a_std)
             sigma = pm.Exponential("sigma", 1)
             div = pm.Normal("div", mu, sigma, observed=w_std)
             # Using the quap method for inference (classical approach)
             idata_AREA = pm.sample(idata_kwargs={"log_likelihood": True})
         models["area"] = idata_AREA

         # Average food model
         with pm.Model() as m_GROUP:
             a = pm.Normal("a", 0.1, 0.2)
             b_f = pm.Normal("b_f", 0.1, 0.5)
             mu = pm.Deterministic("mu", a + b_f * f_std)
             sigma = pm.Exponential("sigma", 1)
             div = pm.Normal("div", mu, sigma, observed=w_std)
             idata_AVGFOOD = pm.sample(idata_kwargs={"log_likelihood": True})
         models["AVGFOOD"] = idata_AVGFOOD

         # Group size model
         with pm.Model() as m_GROUPSIZE:
             a = pm.Normal("a", 0.1, 0.2)
             b_gs = pm.Normal("b_gs", 0.1, 0.5)
             mu = pm.Deterministic("mu", a + b_gs * gs_std)
             sigma = pm.Exponential("sigma", 1)
             div = pm.Normal("div", mu, sigma, observed=w_std)
             idata_GROUPSIZE = pm.sample(idata_kwargs={"log_likelihood": True})
         models["GROUPSIZE"] = idata_GROUPSIZE
```

```python
# Groupsize and area model
with pm.Model() as m_GROUPSIZE_AREA:
    a = pm.Normal("a", 0.1, 0.2)
    b_gs = pm.Normal("b_gs", 0.1, 0.5)
    b_a = pm.Normal("b_a", 0.1, 0.5)
    mu = pm.Deterministic("mu", a + b_gs * gs_std + b_a * a_std)
    sigma = pm.Exponential("sigma", 1)
    div = pm.Normal("div", mu, sigma, observed=w_std)
    idata_GROUPSIZE_AREA = pm.sample(idata_kwargs={"log_likelihood": True})
models["GROUPSIZE_AREA"] = idata_GROUPSIZE_AREA

# Average food and groupsize and area model
with pm.Model() as m_AVGFOOD_GROUPSIZE_AREA:
    a = pm.Normal("a", 0.1, 0.2)
    b_a = pm.Normal("b_a", 0.1, 0.5)
    b_f = pm.Normal("b_f", 0.1, 0.5)
    b_gs = pm.Normal("b_gs", 0.1, 0.5)
    mu = pm.Deterministic("mu", a + b_f * f_std + b_gs * gs_std + b_a * a_std)
    sigma = pm.Exponential("sigma", 1)
    div = pm.Normal("div", mu, sigma, observed=w_std)
    idata_AVGFOOD_GROUPSIZE_AREA = pm.sample(idata_kwargs={"log_likelihood": True})
models["AVGFOOD_GROUPSIZE_AREA"] = idata_AVGFOOD_GROUPSIZE_AREA

# Average food and groupsize model
with pm.Model() as m_AVGFOOD_GROUPSIZE:
    a = pm.Normal("a", 0.1, 0.2)
    b_f = pm.Normal("b_f", 0.1, 0.5)
    b_gs = pm.Normal("b_gs", 0.1, 0.5)
    mu = pm.Deterministic("mu", a + b_f * f_std + b_gs * gs_std)
    sigma = pm.Exponential("sigma", 1)
    div = pm.Normal("div", mu, sigma, observed=w_std)
    idata_AVGFOOD_GROUPSIZE = pm.sample(idata_kwargs={"log_likelihood": True})
models["AVGFOOD_GROUPSIZE"] = idata_AVGFOOD_GROUPSIZE

# Average food and area model
with pm.Model() as m_AVGFOOD_AREA:
    a = pm.Normal("a", 0.1, 0.2)
    b_f = pm.Normal("b_f", 0.1, 0.5)
    b_a = pm.Normal("b_a", 0.1, 0.5)
    mu = pm.Deterministic("mu", a + b_f * f_std + b_a * a_std)
    sigma = pm.Exponential("sigma", 1)
    div = pm.Normal("div", mu, sigma, observed=w_std)
    idata_AVGFOOD_AREA = pm.sample(idata_kwargs={"log_likelihood": True})
models["AVGFOOD_AREA"] = idata_AVGFOOD_AREA

# Execute the comparison between 7 models
# PSIS comparison
compare_df_psis_q2 = az.compare(
    compare_dict=models,
    ic="loo",
    scale="deviance"
)
# WAIC comparison
```

```
compare_df_haic_q2 = az.compare(
    compare_dict=models,
    ic="waic",
    scale="deviance"
)
print("PSIS:", compare_df_psis_q2)
print("WAIC:", compare_df_haic_q2)
```

```
Auto-assigning NUTS sampler…
Initializing NUTS using jitter+adapt_diag…
Sequential sampling (2 chains in 1 job)
NUTS: [a, b_a, sigma]
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
Sampling 2 chains for 1_000 tune and 1_000 draw iterations (2_000 + 2_000 draws total) took 4 seconds.
We recommend running at least 4 chains for robust computation of convergence diagnostics
Auto-assigning NUTS sampler…
Initializing NUTS using jitter+adapt_diag…
Sequential sampling (2 chains in 1 job)
NUTS: [a, b_f, sigma]
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
Sampling 2 chains for 1_000 tune and 1_000 draw iterations (2_000 + 2_000 draws total) took 4 seconds.
We recommend running at least 4 chains for robust computation of convergence diagnostics
```

```
Auto-assigning NUTS sampler…
Initializing NUTS using jitter+adapt_diag…
Sequential sampling (2 chains in 1 job)
NUTS: [a, b_gs, sigma]

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Sampling 2 chains for 1_000 tune and 1_000 draw iterations (2_000 + 2_000 draws total) took 4 seconds.
We recommend running at least 4 chains for robust computation of convergence diagnostics
Auto-assigning NUTS sampler…
Initializing NUTS using jitter+adapt_diag…
Sequential sampling (2 chains in 1 job)
NUTS: [a, b_gs, b_a, sigma]

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Sampling 2 chains for 1_000 tune and 1_000 draw iterations (2_000 + 2_000 draws total) took 7 seconds.
We recommend running at least 4 chains for robust computation of convergence diagnostics
Auto-assigning NUTS sampler…
Initializing NUTS using jitter+adapt_diag…
Sequential sampling (2 chains in 1 job)
NUTS: [a, b_a, b_f, b_gs, sigma]

<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
Sampling 2 chains for 1_000 tune and 1_000 draw iterations (2_000 + 2_000 draws total) took 11 seconds.
We recommend running at least 4 chains for robust computation of convergence diagnostics
Auto-assigning NUTS sampler…
Initializing NUTS using jitter+adapt_diag…
Sequential sampling (2 chains in 1 job)
NUTS: [a, b_f, b_gs, sigma]
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
Sampling 2 chains for 1_000 tune and 1_000 draw iterations (2_000 + 2_000 draws total) took 8 seconds.
We recommend running at least 4 chains for robust computation of convergence diagnostics
Auto-assigning NUTS sampler…
Initializing NUTS using jitter+adapt_diag…
Sequential sampling (2 chains in 1 job)
NUTS: [a, b_f, b_a, sigma]
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
Sampling 2 chains for 1_000 tune and 1_000 draw iterations (2_000 + 2_000 draws total) took 7 seconds.
We recommend running at least 4 chains for robust computation of convergence diagnostics
```

```
PSIS:                          rank    elpd_loo     p_loo  elpd_diff        weight  \
AVGFOOD_GROUPSIZE_AREA     0  322.847144  4.439754   0.000000  2.885929e-15
GROUPSIZE_AREA             1  323.534026  3.443261   0.686882  5.030876e-01
AVGFOOD_GROUPSIZE          2  323.692270  3.459335   0.845126  4.969124e-01
GROUPSIZE                  3  330.740111  2.544852   7.892966  0.000000e+00
AVGFOOD                    4  333.415140  2.324560  10.567995  0.000000e+00
area                       5  333.515934  2.481409  10.668790  0.000000e+00
AVGFOOD_AREA               6  334.676162  3.442385  11.829018  2.109424e-15

                               se       dse  warning     scale
AVGFOOD_GROUPSIZE_AREA  15.320499  0.000000    False  deviance
GROUPSIZE_AREA          14.995065  2.761174    False  deviance
AVGFOOD_GROUPSIZE       15.222967  3.308261    False  deviance
GROUPSIZE               14.018231  5.563166    False  deviance
AVGFOOD                 13.265442  6.651596    False  deviance
area                    13.221790  6.703250    False  deviance
AVGFOOD_AREA            13.333225  6.494722    False  deviance
WAIC:                          rank   elpd_waic    p_waic  elpd_diff        weight  \
AVGFOOD_GROUPSIZE_AREA     0  322.799783  4.416073   0.000000  2.704918e-15
GROUPSIZE_AREA             1  323.508161  3.430329   0.708378  5.021318e-01
AVGFOOD_GROUPSIZE          2  323.651538  3.438969   0.851755  4.978682e-01
GROUPSIZE                  3  330.718307  2.533950   7.918524  2.075288e-15
AVGFOOD                    4  333.400106  2.317043  10.600323  0.000000e+00
area                       5  333.497858  2.472371  10.698075  0.000000e+00
AVGFOOD_AREA               6  334.639521  3.424064  11.839737  0.000000e+00

                               se       dse  warning     scale
AVGFOOD_GROUPSIZE_AREA  15.311801  0.000000     True  deviance
GROUPSIZE_AREA          14.991379  2.760849     True  deviance
AVGFOOD_GROUPSIZE       15.214082  3.303694    False  deviance
GROUPSIZE               14.013987  5.558503    False  deviance
AVGFOOD                 13.263224  6.646099    False  deviance
area                    13.219105  6.697645    False  deviance
AVGFOOD_AREA            13.327914  6.490053    False  deviance
```

```
/opt/conda/envs/fnds/lib/python3.9/site-packages/arviz/stats/stats.py:1632: UserWarning: For one or more
See http://arxiv.org/abs/1507.04544 for details
  warnings.warn(
/opt/conda/envs/fnds/lib/python3.9/site-packages/arviz/stats/stats.py:1632: UserWarning: For one or more
See http://arxiv.org/abs/1507.04544 for details
  warnings.warn(
```

The combination of variables Averagefood, Groupsize, and Area ranks highest (0) out of the 7 models. For elpd_loo scores, this combination has the lowest score of ~322 compared to values greater than it for other models, and elpd_waic has the highest value of ~4.27. The elpd_diff value and weight for this combination of variables also yields 0.00 for both, indicating a perfect fit.

Answer the following question based on your analysis above:

How would you interpret the estimates for the coefficients from the best scoring model in terms of their ability to predict weight? **Limit your response to no more than 10 sentences.**

Hint: Answering this question requires an evaluation of the coefficient estimates which can be obtained by passing the inference data object for the best performing model to `arviz.summary()`.

In [48]:
```python
# Write code here
az.summary(idata_AVGFOOD_GROUPSIZE_AREA, kind="stats")
```

Out[48]:

	mean	sd	hdi_5.5%	hdi_94.5%
a	0.019	0.079	-0.106	0.148
b_a	0.276	0.176	-0.006	0.551
b_f	0.295	0.217	-0.041	0.630
b_gs	-0.632	0.186	-0.926	-0.345
sigma	0.958	0.066	0.852	1.057
…	…	…	…	…
mu[111]	-0.086	0.090	-0.239	0.043
mu[112]	-0.086	0.090	-0.239	0.043
mu[113]	-0.311	0.202	-0.642	-0.003
mu[114]	-0.311	0.202	-0.642	-0.003
mu[115]	-0.311	0.202	-0.642	-0.003

[121 rows x 4 columns]

"a" represents the intercept value of the model with a mean of 0.016 ranging from -0.118 to 0.122 in a 89% HPDI, and has little influence on predicting weight. "b_a" is the coefficient for area ranging from -0.026 to 0.522 so it could have little influence or it could have a moderate influence. "b_f" is the coefficient of average food ranging from -0.061 to 0.639 so it could have litte influence to a decent amount of impact on weight. "b_gs" is the coefficient for group size and it ranges from -0.923 to -0.306 suggesting that weight is strongly impacted and decreasing as groupsize increases. Sigma suggests that in a range of 0.867 to 1.063 the model has room for uncertainty but fits the data.

Question 3. (10 points) Using the data in *Data/cherry_blossoms.csv*:

In this problem, you will build predictive models of the relationship between the timing of cherry blossoms (`doy`) and March temperature in the same year (`temp`).

Note: * Only include observations that have recorded values for `doy` and `temp` (i.e., exclude `NaN` values). * The `pandas.DataFrame.dropna` method can be used to exclude `NaN` values.

Construct at least two different models to predict `doy` with `temp`.

In [ ]:
```python
df = pd.read_csv(
    "Data/cherry_blossoms.csv"
)
df_dropped = df.dropna(subset=["doy", "temp"])
t_std = standardize(df_dropped["temp"])
d_std = standardize(df_dropped["doy"])
# first-order linear model
with pm.Model() as cb_linear:
    a = pm.Normal("a", 0, 0.2)
    b = pm.Normal("b", 0, 0.5)
    mu = pm.Deterministic("mu", a + b * t_std)
    sigma = pm.Exponential("sigma", 1)
    R = pm.Normal("doy", mu, sigma, observed=d_std)
    idata_cb_linear = pm.sample(idata_kwargs={"log_likelihood": True})
# quadratic model
with pm.Model() as cb_quadratic:
    a = pm.Normal("a", 0, 0.2)
    b = pm.Normal("b", 0, 0.5)
    b_t = pm.Normal("b_t", 0, 0.5)
    mu = pm.Deterministic("mu", a + b * t_std + b_t * t_std**2)
    sigma = pm.Exponential("sigma", 1)
    R = pm.Normal("doy", mu, sigma, observed=d_std)
    idata_cb_quadratic = pm.sample(idata_kwargs={"log_likelihood": True})
```

```
Auto-assigning NUTS sampler…
Initializing NUTS using jitter+adapt_diag…
Sequential sampling (2 chains in 1 job)
NUTS: [a, b, sigma]
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
Sampling 2 chains for 1_000 tune and 1_000 draw iterations (2_000 + 2_000 draws total) took 4 seconds.
We recommend running at least 4 chains for robust computation of convergence diagnostics
Auto-assigning NUTS sampler…
Initializing NUTS using jitter+adapt_diag…
Sequential sampling (2 chains in 1 job)
NUTS: [a, b, b_t, sigma]
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
In [ ]: # Comparison using PSIS
        cherry_models = {
            "linear" : idata_cb_linear,
            "quadratic" : idata_cb_quadratic
        }
        compare_cherry_models = az.compare(cherry_models, ic="loo", scale="deviance")
        print(compare_cherry_models)
```

State which model is better at making predictions and explain what evidence leads you to this conclusion. **Limit your response to no more than 5 sentences.**

Linear is better at making predictions because it ranked higher (0) than quadratic (1). It has a lower elpd_loo value than quadratic as well, as 2148 < 2150. It is a better predictor than quadratic because the elpd_diff value is 0.00000 whereas for quadratic, it is nonzero at a value of ~2.13.

```
In [ ]: grader.check("q3.1")
```