PyMC is a probabilistic programming library for Python that allows users to build Bayesian models with a simple Python API and fit them using Markov chain Monte Carlo (MCMC) methods.

- **Modern** - Includes state-of-the-art inference algorithms, including MCMC (NUTS) and variational inference (ADVI).
- **User friendly** - Write your models using friendly Python syntax. Learn Bayesian modeling from the many example notebooks.
- **Fast** - Uses Aesara as its computational backend to compile to C and JAX, run your models on the GPU, and benefit from complex graph-optimizations.
- **Batteries included** - Includes probability distributions, Gaussian processes, ABC, SMC and much more. It integrates nicely with ArviZ for visualizations and diagnostics, as well as Bambi for high-level mixed-effect models.
- **Community focused** - Ask questions on discourse, join MeetUp events, follow us on Twitter, and start contributing.

```python
import pymc as pm
```
ArviZ is a Python package for exploratory analysis of Bayesian models. Includes functions for posterior analysis, data storage, sample diagnostics, model checking, and comparison.

- **Interoperability** - Integrates with all major probabilistic programming libraries: PyMC, CmdStanPy, PyStan, Pyro, NumPyro, and emcee.

- **Large Suite of Visualizations** - Provides over 25 plotting functions for all parts of Bayesian workflow: visualizing distributions, diagnostics, and model checking. See the gallery for examples.

- **State of the Art Diagnostics** - Latest published diagnostics and statistics are implemented, tested and distributed with ArviZ.

- **Flexible Model Comparison** - Includes functions for comparing models with information criteria, and cross validation (both approximate and brute force).

- **Built for Collaboration** - Designed for flexible cross-language serialization using netCDF or Zarr formats. ArviZ also has a Julia version that uses the same data schema.

- **Labeled Data** - Builds on top of xarray to work with labeled dimensions and coordinates.

```python
import arviz as az
```
Some history

PyMC3
The dependable PPL you've been using for years

PyMC
Same great API, will release at 4.0

Theano
The original backend

Aesara
So much graph greatness

Numba

JAX

2016 2022
Model basics

All models are derived from the `Model()` class, unlike what we have seen previously PyMC makes heavy use of Python’s context manager using the `with` statement to add model components to a model.

```python
with pm.Model() as norm:
    x = pm.Normal("x", mu=0, sigma=1)
```

```python
x = pm.Normal("x", mu=0, sigma=1)
```

Error: `TypeError: No model on context stack, which is needed to instantiate distributions. Add variable inside with block.`

Note that `with` blocks do not have their own scope - so variables defined inside are added to the parent scope (be careful about overwriting other variables).

```python
x
```

```python
type(x)
```

```
<class 'pytensor.tensor.var.TensorVariable'>
```
Random Variables

`pm.Normal()` is an example of a PyMC distribution, which are used to construct models, these are implemented using the `TensorVariable` class which is used for all of the builtin distributions (and can be used to create custom distributions). Generally you will not be interacting with these objects directly, but with that said some useful methods and attributes:

```python
1 type(norm.x)
<class 'pytensor.tensor.var.TensorVariable'>
```

```python
1 norm.x.owner.op
<pytensor.tensor.random.basic.NormalRV object at 0x17280e980>
```

```python
1 pm.draw(norm.x)
array(0.16579)
```
Standalone RVs

If you really want to construct a `TensorVariable` outside of a model this can be done via the `dist` method for each distribution.

```python
1  z = pm.Normal.dist(mu=1, sigma=2, shape=[2,3])
2  z

normal_rv{0, (0, 0), floatX, False}.out

1  pm.draw(z)

array([[ 1.83488,  4.45201, -1.60857],
       [-0.29248,  1.78870,  1.81323]])
```
Modifying models

Because of this construction it is possible to add additional components to an existing (named) model via subsequent `with` statements (only the first needs `pm.Model()`)

```python
1   with norm:
2     y = pm.Normal("y", mu=x, sigma=1, shape=3)
```

```python
1   norm.basic_RVs

[x, y]
```
Variable heirarchy

Note that we defined $y|x \sim (x, 1)$, so what is happening when we use `pm.draw(norm.y)`?

```python
pm.draw(norm.y)
```

```python
array([-1.49714, -1.75064, -0.81631])
```

```python
obs = pm.draw(norm.y, draws=1000); obs
```

```python
array([[ 2.2078 ,  2.29494,  1.4284 ],
       [-0.93994, -0.55348,  0.15047],
       [ 0.54024,  1.27989,  0.67047],
       ...
       [ 1.6475 ,  0.83945,  0.71665],
       [-0.18381, -0.30989, -0.36771],
       [-0.52248, -0.31888, -0.42917]])
```

```python
np.mean(obs)
```

```
0.06400085943586256
```

```python
np.var(obs)
```

```
1.9553522392586231
```

```python
np.std(obs)
```

```
1.3983391002395031
```

Each time we ask for a draw from $y$, PyMC is first drawing from $x$ for us.
Beta-Binomial model

We will now build a basic model where we know what the solution should look like and compare the results.

```python
with pm.Model() as beta_binom:
    p = pm.Beta("p", alpha=10, beta=10)
    x = pm.Binomial("x", n=20, p=p, observed=5)
```

In order to sample from the posterior we add a call to `sample()` within the model context.

```python
with beta_binom:
    trace = pm.sample(random_seed=1234, progressbar=False)
```

Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [p]
Sampling 4 chains for 1_000 tune and 1_000 draw iterations (4_000 + 4_000 draws total) took 0 seconds.
pm.sample() results

```python
print(trace)
```

Inference data with groups:
  > posterior
  > sample_stats
  > observed_data

```python
print(type(trace))
```

<class 'arviz.data.inference_data.InferenceData'>
Xarray - N-D labeled arrays and datasets in Python

Xarray (formerly xray) is an open source project and Python package that makes working with labelled multi-dimensional arrays simple, efficient, and fun!

Xarray introduces labels in the form of dimensions, coordinates and attributes on top of raw NumPy-like arrays, which allows for a more intuitive, more concise, and less error-prone developer experience. The package includes a large and growing library of domain-agnostic functions for advanced analytics and visualization with these data structures.

Xarray is inspired by and borrows heavily from pandas, the popular data analysis package focused on labelled tabular data. It is particularly tailored to working with netCDF files, which were the source of xarray’s data model, and integrates tightly with dask for parallel computing.

See here for more details on xarray
Digging into `trace`

```python
print(trace.posterior)
```

```python
<xarray.Dataset>
Dimensions:  (chain: 4, draw: 1000)
Coordinates:
  * chain    (chain) int64 0 1 2 3
  * draw     (draw) int64 0 1 2 3 4 5 6 7 8 ... 992 993 994 995 996 997 998 999
Data variables:
  p        (chain, draw) float64 0.5068 0.4518 0.3853 ... 0.166 0.3242 0.3242
Attributes:
  created_at:                 2023-03-24T16:28:00.773567
  arviz_version:              0.15.1
  inference_library:          pymc
  inference_library_version:  5.1.2
  sampling_time:              0.32303428649902344
  tuning_steps:               1000
```

```python
print(trace.posterior["p"].shape)
```

```
(4, 1000)
```

```python
print(trace.sel(chain=0).posterior["p"].shape)
```

```
(1000,)
```

```python
print(trace.sel(draw=slice(500, None, 10)).posterior["p"].shape)
```

```
(4, 50)
```
As DataFrame

Posterior values, or subsets, can be converted to DataFrames via the `to_dataframe()` method.

```python
trace.posterior.to_dataframe()
```

<table>
<thead>
<tr>
<th>chain</th>
<th>draw</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0.506797</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0.451803</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0.385329</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0.507061</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0.374281</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>995</td>
<td>0</td>
<td>0.284547</td>
</tr>
<tr>
<td>996</td>
<td>0</td>
<td>0.202096</td>
</tr>
<tr>
<td>997</td>
<td>0</td>
<td>0.165955</td>
</tr>
<tr>
<td>998</td>
<td>0</td>
<td>0.324233</td>
</tr>
<tr>
<td>999</td>
<td>0</td>
<td>0.324233</td>
</tr>
</tbody>
</table>

[4000 rows x 1 columns]

```python
trace.posterior["p"][0,:].to_dataframe()
```

<table>
<thead>
<tr>
<th>chain</th>
<th>draw</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0.506797</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0.451803</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0.385329</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0.507061</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0.374281</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>995</td>
<td>0</td>
<td>0.356664</td>
</tr>
<tr>
<td>996</td>
<td>0</td>
<td>0.356664</td>
</tr>
<tr>
<td>997</td>
<td>0</td>
<td>0.397380</td>
</tr>
<tr>
<td>998</td>
<td>0</td>
<td>0.401865</td>
</tr>
<tr>
<td>999</td>
<td>0</td>
<td>0.401865</td>
</tr>
</tbody>
</table>

[1000 rows x 2 columns]
Traceplot

```python
1  ax = az.plot_trace(trace)
2  plt.show()
```
Posterior plot

```python
1 ax = az.plot_posterior(trace, ref_val=[15/40])
2 plt.show()
```
PyMC vs Theoretical

```python
p = np.linspace(0, 1, 100)
post_beta = scipy.stats.beta.pdf(p, 15, 25)
ax = az.plot_posterior(trace, hdi_prob="hide", point_estimate=None)
plt.plot(p, post_beta, "-k", alpha=0.5, label="Theoretical")
plt.legend(['PyMC NUTS', 'Theoretical'])
plt.show()
```
Autocorrelation plots

```python
1 ax = az.plot_autocorr(trace, grid=(2, 2), max_lag=50)
2 plt.show()
```
Forest plots

```python
1  ax = az.plot_forest(trace)
2  plt.show()
```
Other useful diagnostics

Standard MCMC diagnostic statistics are available via `summary()` from ArviZ

```python
az.summary(trace)
```

```
mean     sd  hdi_3%  hdi_97%  mcse_mean  mcse_sd  ess_bulk  ess_tail  r_hat
p  0.372  0.074   0.232    0.511      0.002    0.001    1615.0    2222.0    1.0
```

individual methods are available for each statistics,

```python
print(az.ess(trace, method="bulk"))
```

```python
< xarray.Dataset 
Dimensions: ()
Data variables:
    p   float64 1.615e+03
```

```python
print(az.rhat(trace))
```

```python
< xarray.Dataset 
Dimensions: ()
Data variables:
    p   float64 1.002
```

```python
print(az.ess(trace, method="tail"))
```

```python
< xarray.Dataset 
Dimensions: ()
Data variables:
    p   float64 2.222e+03
```

```python
print(az.mcse(trace))
```

```python
< xarray.Dataset 
Dimensions: ()
Data variables:
    p   float64 0.001844
```

Sta 663 - Spring 2023
Demo 1 - Linear regression

Given the below data, we want to fit a linear regression model to the following synthetic data,

```python
np.random.seed(1234)
n = 11
m = 6
b = 2
x = np.linspace(0, 1, n)
y = m*x + b + np.random.randn(n)
```
Model

```python
with pm.Model() as lm:
    m = pm.Normal('m', mu=0, sigma=50)
    b = pm.Normal('b', mu=0, sigma=50)
    sigma = pm.HalfNormal('sigma', sigma=5)

    pm.Normal('y', mu=m*x + b, sigma=sigma, observed=y)

trace = pm.sample(progressbar=False, random_seed=1234)
```

Auto-assigning NUTS sampler...

Initializing NUTS using jitter+adapt_diag...

Multiprocess sampling (4 chains in 4 jobs)

NUTS: [m, b, sigma]

Sampling 4 chains for 1_000 tune and 1_000 draw iterations (4_000 + 4_000 draws t
### Posterior summary

```python
az.summary(trace)
```

<table>
<thead>
<tr>
<th></th>
<th>mean</th>
<th>sd</th>
<th>hdi_3%</th>
<th>hdi_97%</th>
<th>mcse_mean</th>
<th>mcse_sd</th>
<th>ess_bulk</th>
<th>ess_tail</th>
<th>r_hat</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>5.573</td>
<td>1.344</td>
<td>2.969</td>
<td>8.042</td>
<td>0.040</td>
<td>0.028</td>
<td>1182.0</td>
<td>1229.0</td>
<td>1.0</td>
</tr>
<tr>
<td>b</td>
<td>2.189</td>
<td>0.791</td>
<td>0.659</td>
<td>3.624</td>
<td>0.024</td>
<td>0.018</td>
<td>1106.0</td>
<td>1153.0</td>
<td>1.0</td>
</tr>
<tr>
<td>sigma</td>
<td>1.366</td>
<td>0.378</td>
<td>0.788</td>
<td>2.075</td>
<td>0.010</td>
<td>0.007</td>
<td>1452.0</td>
<td>1722.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>
Trace plots

```python
1 ax = az.plot_trace(trace)
2 plt.show()
```
Posterior plots

```python
1 ax = az.plot_posterior(trace, ref_val=[6,2,1], grid=(1,3))
2 plt.show()
```
Regression line posterior draws

```python
plt.scatter(x, y, s=30, label='data')
post_m = trace.posterior['m'].sel(chain=0, draw=slice(0,None,10))
post_b = trace.posterior['b'].sel(chain=0, draw=slice(0,None,10))
plt.figure(layout="constrained")
plt.scatter(x, y, s=30, label='data')
for m, b in zip(post_m.values, post_b.values):
    plt.plot(x, m*x + b, c='gray', alpha=0.1)
plt.plot(x, 6*x + 2, label='true regression line', lw=3., c='red')
plt.legend(loc='best')
plt.show()
```
Posterior predictive draws

Draws for observed variables can also be generated (posterior predictive draws) via the `sample_posterior_predictive()` method.

```python
with lm:
    pp = pm.sample_posterior_predictive(trace, progressbar=False)
```

Sampling: [y]

```python
pp
```

arviz.InferenceData

- `posterior_predictive`
- `observed_data`

```python
pp.posterior_predictive
```

<xarray.Dataset>
Dimensions: (chain: 4, draw: 1000, y_dim_2: 11)
Coordinates:
  * chain (chain) int64 0 1 2 3
  * draw (draw) int64 0 1 2 3 4 5 6 7 8 ... 992 993 994 995 996 997 998 999
  * y_dim_2 (y_dim_2) int64 0 1 2 3 4 5 6 7 8 9 10
Data variables:
  y (chain, draw, y_dim_2) float64 2.857 2.19 4.669 ... 7.195 5.406
Attributes:
  created_at: 2023-03-24T16:28:16.241911
xarray.Dataset

arviz_version: 0.15.1
inference_library: pymc
inference_library_version: 5.1.2
Dimensions:

\[(\text{chain: 4, draw: 1000, y\_dim\_2: 11})\]

Coordinates:

<table>
<thead>
<tr>
<th>chain</th>
<th>(chain)</th>
<th>int64</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>draw</th>
<th>(draw)</th>
<th>int64</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 ... 995 996 997 998 999</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>y_dim_2</th>
<th>(y_dim_2)</th>
<th>int64</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Data variables:

<table>
<thead>
<tr>
<th>y</th>
<th>(chain, draw, y_dim_2)</th>
<th>float64</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.857 2.19 4.669 ... 7.195 5.406</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Plotting the posterior predictive distribution

```python
1 az.plot_ppc(pp, num_pp_samples=500)
2 plt.show()
```
```python
plt.figure(layout="constrained")
plt.scatter(x, y, s=30, label='data')
plt.plot(x, pp.posterior_predictive['y'].sel(chain=0).T, c="grey", alpha=0.01)
plt.plot(x, np.mean(pp.posterior_predictive['y'].sel(chain=0).T, axis=1), c='red', label="PP mean")
plt.legend()
plt.show()
```
plt.figure(layout="constrained")
plt.scatter(x, y, s=30, label='data')
plt.plot(x, np.mean(pp.posterior_predictive['y'].sel(chain=0).T, axis=1), c='red', label="PP mean")
az.plot_hdi(x, pp.posterior_predictive['y'])
plt.legend()
plt.show()
Model revision

```python
with pm.Model() as lm2:
    m = pm.Normal('m', mu=0, sigma=50)
    b = pm.Normal('b', mu=0, sigma=50)
    sigma = pm.HalfNormal('sigma', sigma=5)

    y_hat = pm.Deterministic("y_hat", m*x + b)

    pm.Normal('y', mu=y_hat, sigma=sigma, observed=y)

trace = pm.sample(random_seed=1234, progressbar=False)
pp = pm.sample_posterior_predictive(trace, var_names=['y_hat'], progressbar=False)
```

Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [m, b, sigma]
Sampling 4 chains for 1_000 tune and 1_000 draw iterations (4_000 + 4_000 draws total) took 1 seconds.
Sampling: []
<table>
<thead>
<tr>
<th></th>
<th>mean</th>
<th>sd</th>
<th>hdi_3%</th>
<th>hdi_97%</th>
<th>mcse_mean</th>
<th>mcse_sd</th>
<th>ess_bulk</th>
<th>ess_tail</th>
<th>r_hat</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>5.573</td>
<td>1.344</td>
<td>2.969</td>
<td>8.042</td>
<td>0.040</td>
<td>0.028</td>
<td>1182.0</td>
<td>1229.0</td>
<td>1.0</td>
</tr>
<tr>
<td>b</td>
<td>2.189</td>
<td>0.791</td>
<td>0.659</td>
<td>3.624</td>
<td>0.024</td>
<td>0.018</td>
<td>1106.0</td>
<td>1153.0</td>
<td>1.0</td>
</tr>
<tr>
<td>sigma</td>
<td>1.366</td>
<td>0.378</td>
<td>0.788</td>
<td>2.075</td>
<td>0.010</td>
<td>0.007</td>
<td>1452.0</td>
<td>1722.0</td>
<td>1.0</td>
</tr>
<tr>
<td>y_hat[0]</td>
<td>2.189</td>
<td>0.791</td>
<td>0.659</td>
<td>3.624</td>
<td>0.024</td>
<td>0.018</td>
<td>1106.0</td>
<td>1153.0</td>
<td>1.0</td>
</tr>
<tr>
<td>y_hat[1]</td>
<td>2.746</td>
<td>0.681</td>
<td>1.427</td>
<td>4.009</td>
<td>0.021</td>
<td>0.015</td>
<td>1155.0</td>
<td>1245.0</td>
<td>1.0</td>
</tr>
<tr>
<td>y_hat[2]</td>
<td>3.303</td>
<td>0.583</td>
<td>2.228</td>
<td>4.424</td>
<td>0.017</td>
<td>0.012</td>
<td>1254.0</td>
<td>1319.0</td>
<td>1.0</td>
</tr>
<tr>
<td>y_hat[3]</td>
<td>3.861</td>
<td>0.501</td>
<td>2.933</td>
<td>4.821</td>
<td>0.013</td>
<td>0.010</td>
<td>1481.0</td>
<td>1786.0</td>
<td>1.0</td>
</tr>
<tr>
<td>y_hat[4]</td>
<td>4.418</td>
<td>0.446</td>
<td>3.543</td>
<td>5.218</td>
<td>0.010</td>
<td>0.007</td>
<td>2031.0</td>
<td>2364.0</td>
<td>1.0</td>
</tr>
<tr>
<td>y_hat[5]</td>
<td>4.975</td>
<td>0.427</td>
<td>4.187</td>
<td>5.791</td>
<td>0.007</td>
<td>0.005</td>
<td>3266.0</td>
<td>2804.0</td>
<td>1.0</td>
</tr>
<tr>
<td>y_hat[6]</td>
<td>5.532</td>
<td>0.449</td>
<td>4.671</td>
<td>6.389</td>
<td>0.007</td>
<td>0.005</td>
<td>4478.0</td>
<td>2810.0</td>
<td>1.0</td>
</tr>
<tr>
<td>y_hat[7]</td>
<td>6.090</td>
<td>0.508</td>
<td>5.090</td>
<td>7.017</td>
<td>0.008</td>
<td>0.006</td>
<td>3929.0</td>
<td>2889.0</td>
<td>1.0</td>
</tr>
<tr>
<td>y_hat[8]</td>
<td>6.647</td>
<td>0.591</td>
<td>5.568</td>
<td>7.810</td>
<td>0.011</td>
<td>0.008</td>
<td>2963.0</td>
<td>2707.0</td>
<td>1.0</td>
</tr>
<tr>
<td>y_hat[9]</td>
<td>7.204</td>
<td>0.691</td>
<td>5.924</td>
<td>8.542</td>
<td>0.014</td>
<td>0.010</td>
<td>2389.0</td>
<td>2337.0</td>
<td>1.0</td>
</tr>
<tr>
<td>y_hat[10]</td>
<td>7.761</td>
<td>0.801</td>
<td>6.358</td>
<td>9.405</td>
<td>0.018</td>
<td>0.013</td>
<td>2087.0</td>
<td>2291.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>
1 plt.figure(layout="constrained")
2 plt.plot(x, pp.posterior_predictive['y_hat'].sel(chain=0).T, c="grey", alpha=0.01)
3 plt.scatter(x, y, s=30, label='data')
4 plt.show()
Demo 2 - Bayesian Lasso

```python
n = 50
k = 100

np.random.seed(1234)
X = np.random.normal(size=(n, k))

beta = np.zeros(shape=k)
beta[[10, 30, 50, 70]] = 10
beta[[20, 40, 60, 80]] = -10

y = X @ beta + np.random.normal(size=n)
```

Based on Bayesian Sparse Regression and Lasso regression with block updating
Naive model

```python
with pm.Model() as bayes_naive:
    b = pm.Flat("beta", shape=k)
    s = pm.HalfNormal('sigma', sigma=2)

    pm.Normal("y", mu=X @ b, sigma=s, observed=y)

trace = pm.sample(progressbar=False, random_seed=12345)
```

Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [beta, sigma]
Sampling 4 chains for 1_000 tune and 1_000 draw iterations (4_000 + 4_000 draws total) took 70 seconds.
The rhat statistic is larger than 1.01 for some parameters. This indicates problems during sampling. See https://arxiv.org/abs/1903.08008 for details
The effective sample size per chain is smaller than 100 for some parameters. A higher number is needed for rhat and ess computation. See https://arxiv.org/abs/1903.08008 for details
There were 113 divergences after tuning. Increase `target_accept` or reparameterize.
Chain 0 reached the maximum tree depth. Increase `max_treedepth`, increase `target_accept` or reparameterize
Chain 1 reached the maximum tree depth. Increase `max_treedepth`, increase `target_accept` or reparameterize
Chain 2 reached the maximum tree depth. Increase `max_treedepth`, increase `target_accept` or reparameterize
Chain 3 reached the maximum tree depth. Increase `max_treedepth`, increase `target_accept` or reparameterize

Sta 663 - Spring 2023
<table>
<thead>
<tr>
<th></th>
<th>mean</th>
<th>sd</th>
<th>hdi_3%</th>
<th>hdi_97%</th>
<th>mcse_mean</th>
<th>mcse_sd</th>
<th>ess_bulk</th>
<th>ess_tail</th>
</tr>
</thead>
<tbody>
<tr>
<td>beta[0]</td>
<td>643.620</td>
<td>606.139</td>
<td>-622.272</td>
<td>1402.288</td>
<td>261.586</td>
<td>196.065</td>
<td>6.0</td>
<td>14.0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>beta[96]</td>
<td>420.720</td>
<td>1056.327</td>
<td>-1260.449</td>
<td>2382.936</td>
<td>487.813</td>
<td>369.004</td>
<td>5.0</td>
<td>12.0</td>
</tr>
<tr>
<td>beta[97]</td>
<td>-143.535</td>
<td>1203.422</td>
<td>-2170.522</td>
<td>1927.823</td>
<td>574.093</td>
<td>436.428</td>
<td>5.0</td>
<td>14.0</td>
</tr>
<tr>
<td>beta[98]</td>
<td>-818.080</td>
<td>685.229</td>
<td>-2423.653</td>
<td>190.270</td>
<td>313.078</td>
<td>245.187</td>
<td>6.0</td>
<td>12.0</td>
</tr>
<tr>
<td>beta[99]</td>
<td>261.936</td>
<td>912.018</td>
<td>-1070.744</td>
<td>1629.130</td>
<td>440.188</td>
<td>335.261</td>
<td>5.0</td>
<td>15.0</td>
</tr>
<tr>
<td>sigma</td>
<td>2.432</td>
<td>1.316</td>
<td>0.461</td>
<td>4.800</td>
<td>0.467</td>
<td>0.343</td>
<td>7.0</td>
<td>27.0</td>
</tr>
</tbody>
</table>

[101 rows x 9 columns]
Weakly informative model

```python
with pm.Model() as bayes_weak:
    b = pm.Normal("beta", mu=0, sigma=10, shape=k)
    s = pm.HalfNormal('sigma', sigma=2)
    pm.Normal("y", mu=X @ b, sigma=s, observed=y)

trace = pm.sample(progressbar=False, random_seed=12345)
```

Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [beta, sigma]
Sampling 4 chains for 1_000 tune and 1_000 draw iterations (4_000 + 4_000 draws total) took 71 seconds.
The rhat statistic is larger than 1.01 for some parameters. This indicates problems during sampling. See http://arxiv.org/abs/1903.08008 for details
The effective sample size per chain is smaller than 100 for some parameters. A higher number is needed for rhat and ess computation. See https://arxiv.org/abs/1903.08008 for details
There were 40 divergences after tuning. Increase `target_accept` or reparameterize.
Chain 2 reached the maximum tree depth. Increase `max_treedepth`, increase `target_accept` or reparameterize.
```python
az.summary(trace)
```

<table>
<thead>
<tr>
<th></th>
<th>mean</th>
<th>sd</th>
<th>hdi_3%</th>
<th>hdi_97%</th>
<th>mcse_mean</th>
<th>mcse_sd</th>
<th>ess_bulk</th>
<th>ess_tail</th>
<th>r_hat</th>
</tr>
</thead>
<tbody>
<tr>
<td>beta[0]</td>
<td>-0.077</td>
<td>6.620</td>
<td>-12.749</td>
<td>12.799</td>
<td>0.200</td>
<td>0.548</td>
<td>830.0</td>
<td>1795.0</td>
<td>1.07</td>
</tr>
<tr>
<td>beta[1]</td>
<td>0.527</td>
<td>5.875</td>
<td>-10.243</td>
<td>12.733</td>
<td>0.693</td>
<td>0.525</td>
<td>63.0</td>
<td>2169.0</td>
<td>1.13</td>
</tr>
<tr>
<td>beta[3]</td>
<td>-0.756</td>
<td>6.533</td>
<td>-14.000</td>
<td>10.236</td>
<td>0.958</td>
<td>0.681</td>
<td>43.0</td>
<td>1570.0</td>
<td>1.06</td>
</tr>
<tr>
<td>beta[4]</td>
<td>0.399</td>
<td>7.131</td>
<td>-12.064</td>
<td>14.597</td>
<td>1.102</td>
<td>0.785</td>
<td>40.0</td>
<td>1274.0</td>
<td>1.06</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>beta[96]</td>
<td>0.085</td>
<td>6.255</td>
<td>-10.955</td>
<td>12.770</td>
<td>0.510</td>
<td>0.462</td>
<td>162.0</td>
<td>1586.0</td>
<td>1.03</td>
</tr>
<tr>
<td>beta[97]</td>
<td>-2.220</td>
<td>6.716</td>
<td>-14.092</td>
<td>11.428</td>
<td>1.059</td>
<td>0.754</td>
<td>40.0</td>
<td>234.0</td>
<td>1.07</td>
</tr>
<tr>
<td>beta[99]</td>
<td>-1.003</td>
<td>6.763</td>
<td>-14.138</td>
<td>10.319</td>
<td>0.633</td>
<td>0.449</td>
<td>125.0</td>
<td>1170.0</td>
<td>1.02</td>
</tr>
<tr>
<td>sigma</td>
<td>1.606</td>
<td>1.172</td>
<td>0.013</td>
<td>3.646</td>
<td>0.393</td>
<td>0.288</td>
<td>7.0</td>
<td>11.0</td>
<td>1.57</td>
</tr>
</tbody>
</table>

[101 rows x 9 columns]
```python
az.summary(trace).iloc[[10,20,30,40,50,60,70,80]]
```

<table>
<thead>
<tr>
<th></th>
<th>mean</th>
<th>sd</th>
<th>hdi_3%</th>
<th>hdi_97%</th>
<th>mcse_mean</th>
<th>mcse_sd</th>
<th>ess_bulk</th>
<th>ess_tail</th>
<th>r_hat</th>
</tr>
</thead>
<tbody>
<tr>
<td>beta[10]</td>
<td>4.967</td>
<td>6.752</td>
<td>-9.493</td>
<td>15.785</td>
<td>1.221</td>
<td>0.872</td>
<td>31.0</td>
<td>2230.0</td>
<td>1.09</td>
</tr>
<tr>
<td>beta[20]</td>
<td>-4.413</td>
<td>6.705</td>
<td>-15.987</td>
<td>8.051</td>
<td>1.157</td>
<td>0.825</td>
<td>34.0</td>
<td>1600.0</td>
<td>1.08</td>
</tr>
<tr>
<td>beta[30]</td>
<td>4.986</td>
<td>5.798</td>
<td>-6.198</td>
<td>15.950</td>
<td>0.606</td>
<td>0.560</td>
<td>95.0</td>
<td>1727.0</td>
<td>1.06</td>
</tr>
<tr>
<td>beta[40]</td>
<td>-3.742</td>
<td>7.697</td>
<td>-19.201</td>
<td>9.502</td>
<td>1.128</td>
<td>0.802</td>
<td>48.0</td>
<td>263.0</td>
<td>1.05</td>
</tr>
<tr>
<td>beta[50]</td>
<td>6.186</td>
<td>6.122</td>
<td>-6.222</td>
<td>16.288</td>
<td>0.635</td>
<td>0.451</td>
<td>122.0</td>
<td>1311.0</td>
<td>1.04</td>
</tr>
<tr>
<td>beta[60]</td>
<td>-6.317</td>
<td>6.212</td>
<td>-17.585</td>
<td>6.570</td>
<td>0.327</td>
<td>0.231</td>
<td>364.0</td>
<td>1647.0</td>
<td>1.10</td>
</tr>
<tr>
<td>beta[70]</td>
<td>4.856</td>
<td>6.688</td>
<td>-6.852</td>
<td>18.421</td>
<td>0.461</td>
<td>0.423</td>
<td>217.0</td>
<td>1495.0</td>
<td>1.04</td>
</tr>
</tbody>
</table>

Sta 663 - Spring 2023
1 \texttt{ax = az.plot_forest(trace)}
2 \texttt{plt.tight_layout()}
3 \texttt{plt.show()}
def plot_slope(trace, prior="beta", chain=0):
    post = (trace.posterior[prior]
        .to_dataframe()
        .reset_index()
        .query(f"chain == {chain}"))

    sns.catplot(x="beta_dim_0", y="beta", data=post, kind="boxen", linewidth=0, color='blue', aspect=2, s
plt.tight_layout()
plt.xticks(range(0, 110, 10))
plt.show()
plot_slope(trace)
with pm.Model() as bayes_lasso:
    b = pm.Laplace("beta", 0, 1, shape=k)
    s = pm.HalfNormal('sigma', sigma=1)
    pm.Normal("y", mu=X @ b, sigma=s, observed=y)
trace = pm.sample(progressbar=False, random_seed=1234)

Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [beta, sigma]
Sampling 4 chains for 1_000 tune and 1_000 draw iterations (4_000 + 4_000 draws total) took 23 seconds.
The rhat statistic is larger than 1.01 for some parameters. This indicates problems during sampling. See http://arxiv.org/abs/1903.08008 for details
The effective sample size per chain is smaller than 100 for some parameters. A higher number is needed for rhat and ess computation. See https://arxiv.org/abs/1903.08008 for details
There were 239 divergences after tuning. Increase `target_accept` or reparameterize.
```
az.summary(trace)

<table>
<thead>
<tr>
<th></th>
<th>mean</th>
<th>sd</th>
<th>hdi_3%</th>
<th>hdi_97%</th>
<th>mcse_mean</th>
<th>mcse_sd</th>
<th>ess_bulk</th>
<th>ess_tail</th>
<th>r_hat</th>
</tr>
</thead>
<tbody>
<tr>
<td>beta[0]</td>
<td>0.067</td>
<td>0.785</td>
<td>-1.402</td>
<td>1.716</td>
<td>0.011</td>
<td>0.018</td>
<td>4345.0</td>
<td>1972.0</td>
<td>1.02</td>
</tr>
<tr>
<td>beta[1]</td>
<td>0.308</td>
<td>0.829</td>
<td>-1.179</td>
<td>1.710</td>
<td>0.152</td>
<td>0.119</td>
<td>38.0</td>
<td>533.0</td>
<td>1.07</td>
</tr>
<tr>
<td>beta[2]</td>
<td>-0.045</td>
<td>0.783</td>
<td>-1.634</td>
<td>1.478</td>
<td>0.012</td>
<td>0.015</td>
<td>4101.0</td>
<td>2327.0</td>
<td>1.02</td>
</tr>
<tr>
<td>beta[3]</td>
<td>-0.215</td>
<td>0.804</td>
<td>-1.865</td>
<td>1.157</td>
<td>0.075</td>
<td>0.053</td>
<td>95.0</td>
<td>2069.0</td>
<td>1.03</td>
</tr>
<tr>
<td>beta[4]</td>
<td>0.063</td>
<td>0.775</td>
<td>-1.508</td>
<td>1.570</td>
<td>0.012</td>
<td>0.013</td>
<td>4381.0</td>
<td>2580.0</td>
<td>1.11</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>beta[96]</td>
<td>0.173</td>
<td>0.774</td>
<td>-1.322</td>
<td>1.427</td>
<td>0.136</td>
<td>0.097</td>
<td>38.0</td>
<td>2541.0</td>
<td>1.07</td>
</tr>
<tr>
<td>beta[97]</td>
<td>-0.139</td>
<td>0.721</td>
<td>-1.532</td>
<td>1.351</td>
<td>0.011</td>
<td>0.014</td>
<td>3961.0</td>
<td>2368.0</td>
<td>1.11</td>
</tr>
<tr>
<td>beta[98]</td>
<td>0.244</td>
<td>0.708</td>
<td>-1.038</td>
<td>1.689</td>
<td>0.028</td>
<td>0.020</td>
<td>570.0</td>
<td>2040.0</td>
<td>1.01</td>
</tr>
<tr>
<td>beta[99]</td>
<td>-0.277</td>
<td>0.750</td>
<td>-1.850</td>
<td>1.061</td>
<td>0.020</td>
<td>0.016</td>
<td>1298.0</td>
<td>1953.0</td>
<td>1.01</td>
</tr>
<tr>
<td>sigma</td>
<td>0.825</td>
<td>0.522</td>
<td>0.171</td>
<td>1.790</td>
<td>0.129</td>
<td>0.093</td>
<td>10.0</td>
<td>8.0</td>
<td>1.30</td>
</tr>
</tbody>
</table>

[101 rows x 9 columns]
```
```python
az.summary(trace).iloc[[10,20,30,40,50,60,70,80]]
```

<table>
<thead>
<tr>
<th></th>
<th>mean</th>
<th>sd</th>
<th>hdi_3%</th>
<th>hdi_97%</th>
<th>mcse_mean</th>
<th>mcse_sd</th>
<th>ess_bulk</th>
<th>ess_tail</th>
<th>r_hat</th>
</tr>
</thead>
<tbody>
<tr>
<td>beta[10]</td>
<td>8.306</td>
<td>1.184</td>
<td>6.066</td>
<td>10.577</td>
<td>0.025</td>
<td>0.018</td>
<td>2301.0</td>
<td>2176.0</td>
<td>1.04</td>
</tr>
<tr>
<td>beta[20]</td>
<td>-8.298</td>
<td>1.253</td>
<td>-10.672</td>
<td>-5.908</td>
<td>0.029</td>
<td>0.021</td>
<td>1860.0</td>
<td>2316.0</td>
<td>1.01</td>
</tr>
<tr>
<td>beta[30]</td>
<td>8.673</td>
<td>0.979</td>
<td>6.683</td>
<td>10.340</td>
<td>0.050</td>
<td>0.035</td>
<td>551.0</td>
<td>2361.0</td>
<td>1.01</td>
</tr>
<tr>
<td>beta[40]</td>
<td>-8.781</td>
<td>1.517</td>
<td>-11.710</td>
<td>-5.931</td>
<td>0.031</td>
<td>0.022</td>
<td>2408.0</td>
<td>2496.0</td>
<td>1.12</td>
</tr>
<tr>
<td>beta[50]</td>
<td>9.076</td>
<td>0.981</td>
<td>7.097</td>
<td>10.773</td>
<td>0.041</td>
<td>0.029</td>
<td>1055.0</td>
<td>2812.0</td>
<td>1.01</td>
</tr>
<tr>
<td>beta[60]</td>
<td>-9.313</td>
<td>1.133</td>
<td>-11.377</td>
<td>-7.166</td>
<td>0.133</td>
<td>0.098</td>
<td>69.0</td>
<td>2392.0</td>
<td>1.04</td>
</tr>
<tr>
<td>beta[70]</td>
<td>8.636</td>
<td>1.228</td>
<td>6.214</td>
<td>10.546</td>
<td>0.167</td>
<td>0.125</td>
<td>53.0</td>
<td>2107.0</td>
<td>1.05</td>
</tr>
<tr>
<td>beta[80]</td>
<td>-9.984</td>
<td>0.888</td>
<td>-11.761</td>
<td>-8.366</td>
<td>0.017</td>
<td>0.012</td>
<td>2629.0</td>
<td>2425.0</td>
<td>1.05</td>
</tr>
</tbody>
</table>
plot_slope(trace)