chapter 2

bayesian p values

$$pB = p(T_{\text{sim}} \leq T_{\text{obs}} \mid \tilde{Y})$$

Diagnostics

```
import arviz as az
import matplotlib.pyplot as plt
import numpy as np
import pymc as pm
from scipy import stats
```

```
good_chains = stats.beta.rvs(2, 5, size=(2, 2000))
bad_chains0 = np.random.normal(np.sort(good_chains, axis=None), 0.05,
                               size=4000).reshape(2, -1)

bad_chains1 = good_chains.copy()
for i in np.random.randint(1900, size=4):
    bad_chains1[i%2:,i:i+100] = np.random.beta(i, 950, size=100)

chains = {"good_chains":good_chains,
          "bad_chains0":bad_chains0,
          "bad_chains1":bad_chains1}
```

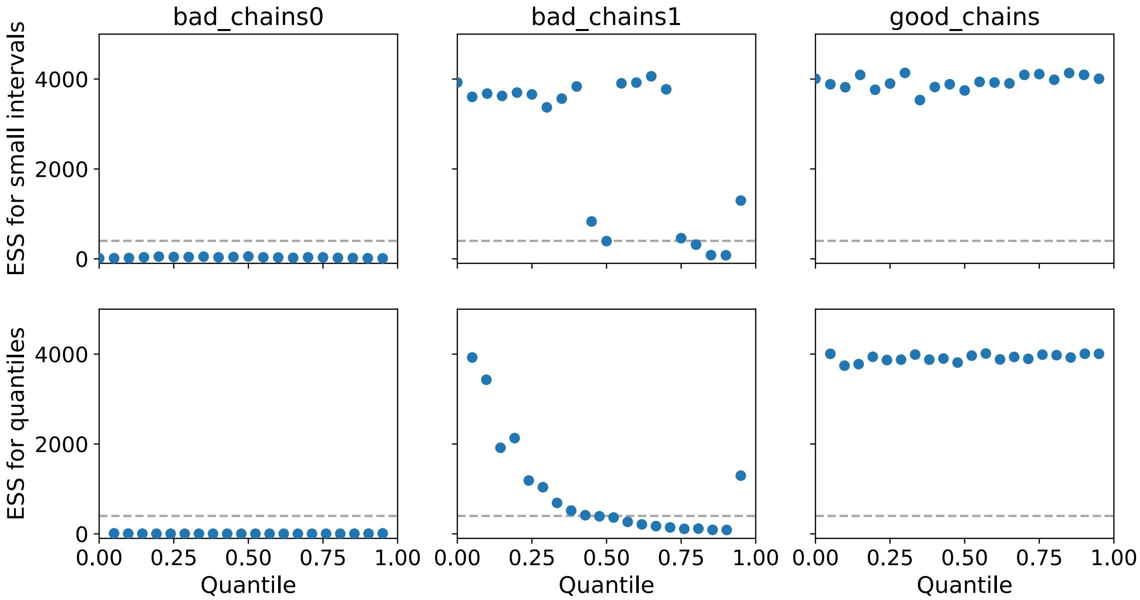
Effective sample size

```
import arviz as az

print(az.ess(chains))
```

```
<xarray.Dataset> Size: 24B
Dimensions:      ()
Data variables:
    bad_chains0  float64 8B 2.421
    bad_chains1  float64 8B 222.7
    good_chains  float64 8B 3.902e+03
```

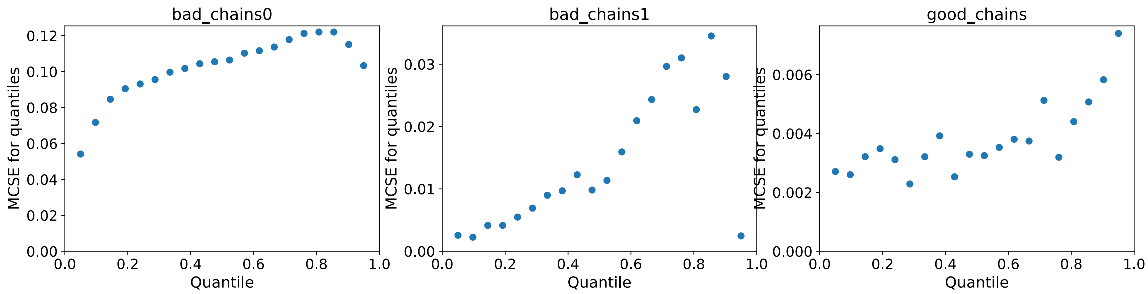
```
import matplotlib.pyplot as plt

_, axes = plt.subplots(2, 3, figsize=(12, 6), sharey=True, sharex=True)
```

```python
az.plot_ess(chains, kind="local", ax=axes[0])
az.plot_ess(chains, kind="quantile", ax=axes[1])

for ax_ in axes[0]:
    ax_.set_xlabel("")
for ax_ in axes[1]:
    ax_.set_title("")

for ax_ in axes[:,1:].ravel():
    ax_.set_ylabel("")
plt.ylim(-100, 5000)
```



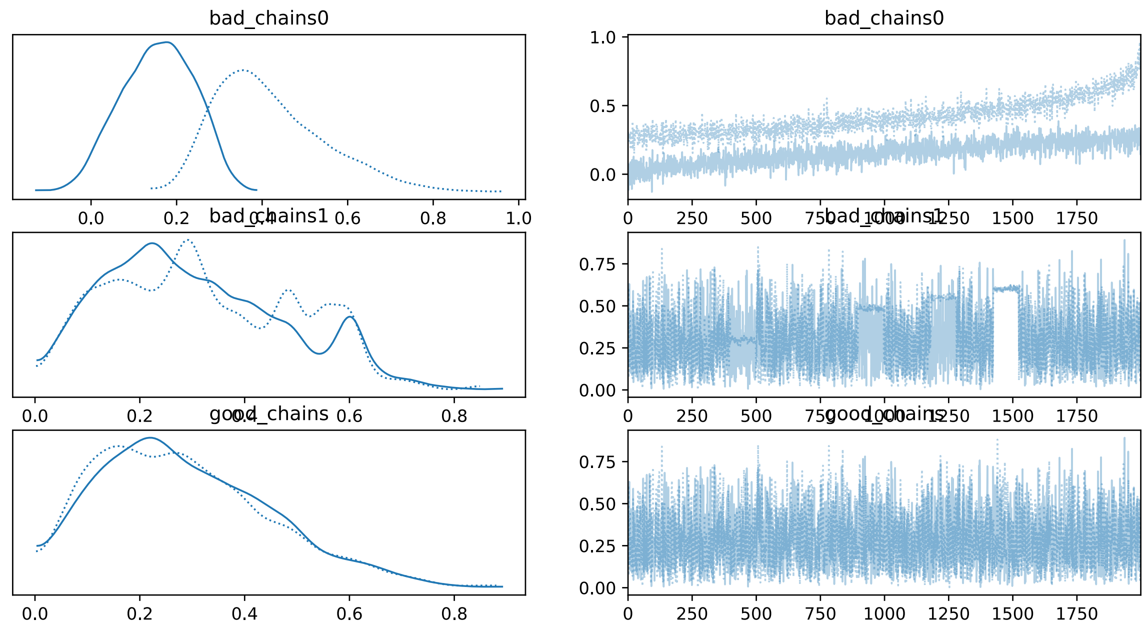

```python
# this doesn't have a shared Y axis by default,
#  probably becauase you normally would normally
# look at the parameters at basically the same tim
az.plot_mcse(chains)
```

```
array([<Axes: title={'center': 'bad_chains0'}, xlabel='Quantile', ylabel='MCSE
for quantiles'>,
       <Axes: title={'center': 'bad_chains1'}, xlabel='Quantile', ylabel='MCSE
for quantiles'>,
       <Axes: title={'center': 'good_chains'}, xlabel='Quantile', ylabel='MCSE
for quantiles'>],
      dtype=object)
```




```
az.plot_trace(chains)
```

```
array([[<Axes: title={'center': 'bad_chains0'}>,
        <Axes: title={'center': 'bad_chains0'}>],
       [<Axes: title={'center': 'bad_chains1'}>,
        <Axes: title={'center': 'bad_chains1'}>],
       [<Axes: title={'center': 'good_chains'}>,
        <Axes: title={'center': 'good_chains'}>]], dtype=object)
```

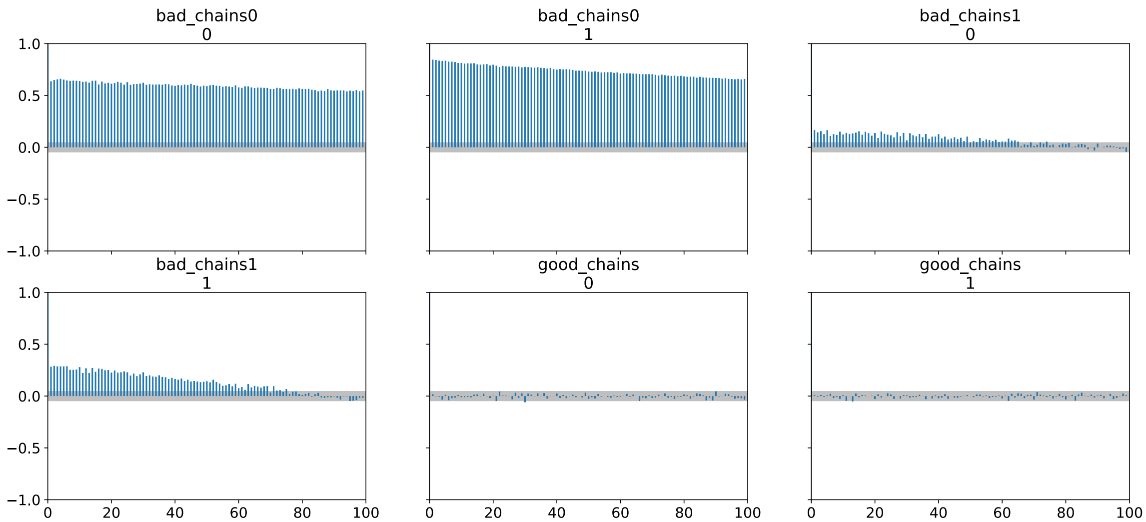


this is nice it shows chains for both realizations

```
az.plot_autocorr(chains)
```

```
array([[<Axes: title={'center': 'bad_chains0\n0'}>,
        <Axes: title={'center': 'bad_chains0\n1'}>,
        <Axes: title={'center': 'bad_chains1\n0'}>],
```

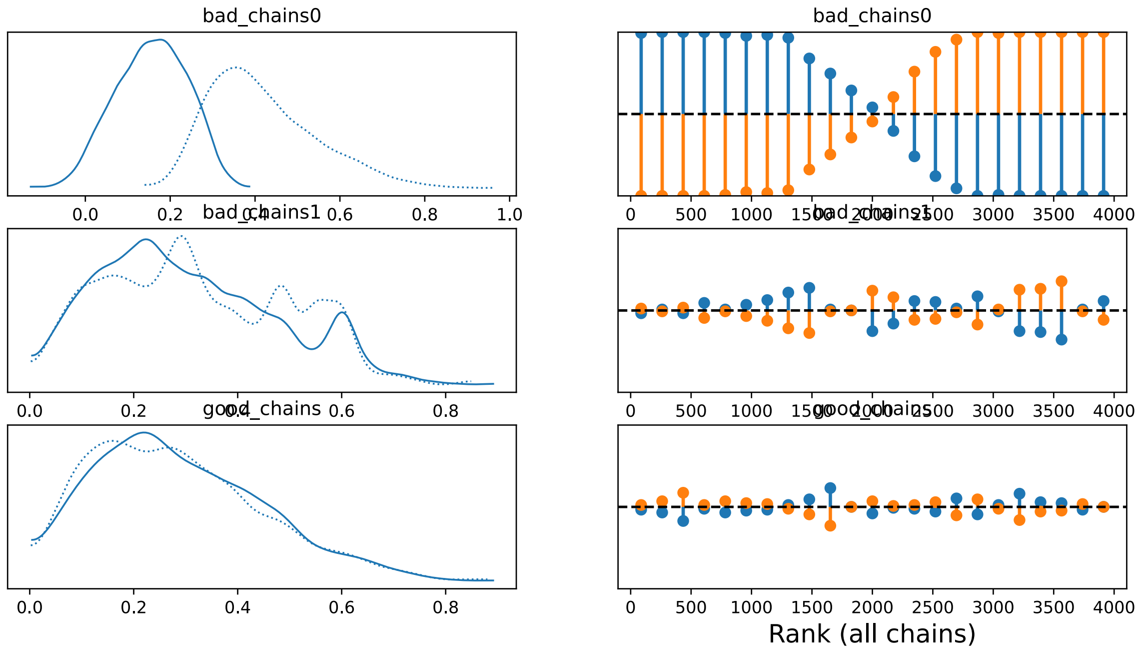
```
 [<Axes: title={'center': 'bad_chains1\n1'}>,
  <Axes: title={'center': 'good_chains\n0'}>,
  <Axes: title={'center': 'good_chains\n1'}>]], dtype=object)
```



I think that this is the best way to look at the chains

```
az.plot_trace(chains, kind="rank_vlines")
```

```
array([[<Axes: title={'center': 'bad_chains0'}>,
        <Axes: title={'center': 'bad_chains0'}, xlabel='Rank (all chains)'>],
       [<Axes: title={'center': 'bad_chains1'}>,
        <Axes: title={'center': 'bad_chains1'}, xlabel='Rank (all chains)'>],
       [<Axes: title={'center': 'good_chains'}>,
        <Axes: title={'center': 'good_chains'}, xlabel='Rank (all chains)'>]],
      dtype=object)
```



code 2.12

```
import pymc as pm

with pm.Model() as model_0:
    theta1 = pm.Normal("theta1", 0, 1, initval=0.1)
    theta2 = pm.Uniform("theta2", -theta1, theta1)
    idata_0 = pm.sample(return_inferencedata=True)
```

```
Initializing NUTS using jitter+adapt_diag...
/home/kirvanlewis/projects/bmcp/.venv/lib/python3.11/site-packages/pytensor/
tensor/elemwise.py:735: RuntimeWarning: invalid value encountered in log
  variables = ufunc(*ufunc_args, **ufunc_kwargs)
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [theta1, theta2]
```

```
/home/kirvanlewis/projects/bmcp/.venv/lib/python3.11/site-packages/rich/
live.py:231: UserWarning: install
"ipywidgets" for Jupyter support
  warnings.warn('install "ipywidgets" for Jupyter support')
```

```
Sampling 4 chains for 1_000 tune and 1_000 draw iterations (4_000 + 4_000 draws
total) took 2 seconds.
There were 2244 divergences after tuning. Increase `target_accept` or
reparameterize.
```

```
y_obs =  np.random.normal(0, 1, size=100)
idatas_cmp = {}

with pm.Model() as mA:
    σ = pm.HalfNormal("σ", 1)
    y = pm.SkewNormal("y", mu=0, sigma=σ, alpha=1, observed=y_obs)
    idataA = pm.sample(idata_kwargs={"log_likelihood":True})
    idataA.extend(pm.sample_posterior_predictive(idataA))
    idatas_cmp["mA"] = idataA

# zero mean, happens to be correct here

with pm.Model() as mB:
    σ = pm.HalfNormal("σ", 1)
    y = pm.Normal("y", 0, σ, observed=y_obs)
    idataB = pm.sample(idata_kwargs={"log_likelihood":True})
    idataB.extend(pm.sample_posterior_predictive(idataB))
    idatas_cmp["mB"] = idataB

# random mean

with pm.Model() as mC:
    μ = pm.Normal("μ", 0, 1)
    σ = pm.HalfNormal("σ", 1)
    y = pm.Normal("y", μ, σ, observed=y_obs)
    idataC = pm.sample(idata_kwargs={"log_likelihood":True})
    idataC.extend(pm.sample_posterior_predictive(idataC))
    idatas_cmp["mC"] = idataC

az.compare(idatas_cmp)
```

```
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [σ]
```

```
/home/kirvanlewis/projects/bmcp/.venv/lib/python3.11/site-packages/rich/
live.py:231: UserWarning: install
"ipywidgets" for Jupyter support
  warnings.warn('install "ipywidgets" for Jupyter support')
```

```
Sampling 4 chains for 1_000 tune and 1_000 draw iterations (4_000 + 4_000 draws
total) took 1 seconds.
Sampling: [y]
```

```
/home/kirvanlewis/projects/bmcp/.venv/lib/python3.11/site-packages/rich/
live.py:231: UserWarning: install
"ipywidgets" for Jupyter support
  warnings.warn('install "ipywidgets" for Jupyter support')
```

```

```

```
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [σ]
```

```
/home/kirvanlewis/projects/bmcp/.venv/lib/python3.11/site-packages/rich/
live.py:231: UserWarning: install
"ipywidgets" for Jupyter support
  warnings.warn('install "ipywidgets" for Jupyter support')
```

```

```

```
Sampling 4 chains for 1_000 tune and 1_000 draw iterations (4_000 + 4_000 draws
total) took 1 seconds.
Sampling: [y]
```

```
/home/kirvanlewis/projects/bmcp/.venv/lib/python3.11/site-packages/rich/
live.py:231: UserWarning: install
"ipywidgets" for Jupyter support
  warnings.warn('install "ipywidgets" for Jupyter support')
```

```

```

```
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [μ, σ]
```

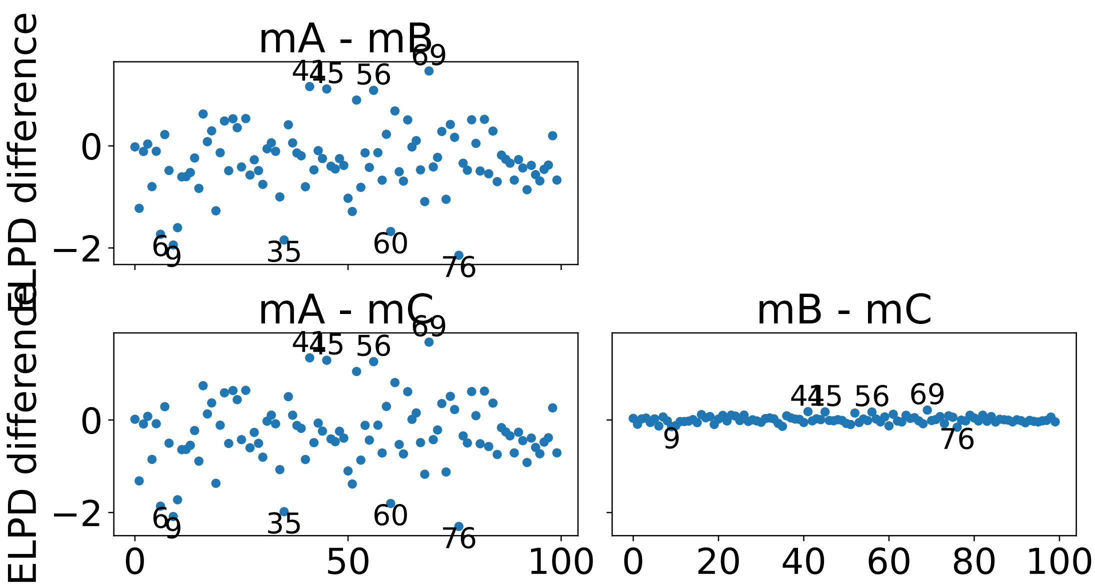
```
/home/kirvanlewis/projects/bmcp/.venv/lib/python3.11/site-packages/rich/
live.py:231: UserWarning: install
"ipywidgets" for Jupyter support
  warnings.warn('install "ipywidgets" for Jupyter support')
```

```
Sampling 4 chains for 1_000 tune and 1_000 draw iterations (4_000 + 4_000 draws
total) took 1 seconds.
Sampling: [y]
```

```
/home/kirvanlewis/projects/bmcp/.venv/lib/python3.11/site-packages/rich/
live.py:231: UserWarning: install
"ipywidgets" for Jupyter support
  warnings.warn('install "ipywidgets" for Jupyter support')
```

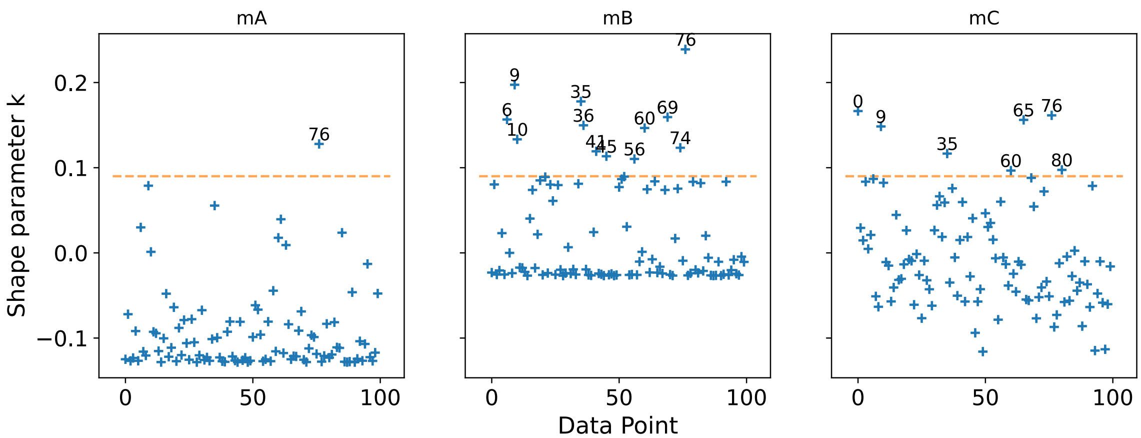
	rank	elpd_loo	p_loo	elpd_diff	weight	se	dse	warning	scale
mB	0	−149.944148	3.164403	0.000000	1.000000e+00	7.710005	0.00000	False	log
mC	1	−150.590992	2.051472	0.646851	4.440892e-16	7.642037	0.72549	False	log
mA	2	−179.649781	1.629871	29.705635	0.000000e+00	11.348048	6.60913	False	log

```
az.plot_elpd(idatas_cmp,    figsize=(10,    5),    plot_kwargs={"marker":"."},
threshold=2);
```



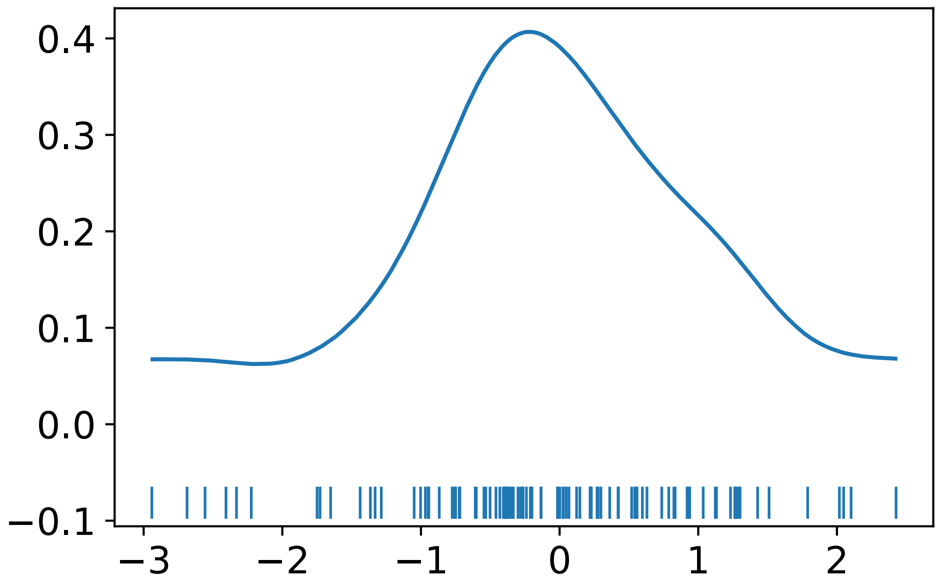
plot shape paremeter K for loo computation (value for pareto distribution) can be used to detect highly influential points (above 0.7 as a rule of thumb, none here)

```
_, axes = plt.subplots(1, 3, figsize=(12, 4), sharey=True)
for idx, (model, ax) in enumerate(zip(("mA", "mB", "mC"), axes)):
    loo_ = az.loo(idatas_cmp[model], pointwise=True)
             az.plot_khat(loo_,    ax=ax,    threshold=0.09,    show_hlines=True,
hlines_kwargs={"hlines":0.09, "ls":"--"})
    ax.set_title(model)
    if idx:
        axes[idx].set_ylabel("")
    if not idx % 2:
        axes[idx].set_xlabel("")
```



KDE with data

```
plt.close()

az.plot_kde(y_obs, rug=True)
```



Loo-pit

$$P_i = P(\tilde{y}_i \leq Y_i \mid y - i)$$

```
_, axes = plt.subplots(1, 3, figsize=(12, 4), sharey=True)
for model, ax in zip(("mA", "mB", "mC"), axes):
    az.plot_loo_pit(idatas_cmp[model], y="y", legend=False, use_hdi=True, ax=ax)
     ax.set_title(model)
     ax.set_xticks([0, 0.5, 1])
     ax.set_yticks([0, 1, 2])
```