In [118…
```
import arviz as az
import bambi as bmb
import numpy as np
import pandas as pd
import pymc as pm
import pytensor.tensor as pt
import pymssql

from bambi.families.univariate import UnivariateFamily
```

Generate our data.

Note that the true underlying coefficient between score ~ x1 is 1 and the higher x1 is, the higher the sample size will be.

In [151…
```
rng = np.random.default_rng(1234)
data = pd.DataFrame(
    {
        "x1": np.random.normal(loc=92.5, scale=12.1, size = 1000000)
    }
)
data['score'] = data['x1'] + np.random.normal(loc=0, scale=136, size=1000000)
data['weights'] = data['x1'] * np.random.rand(1000000) * 25 + np.random.uniform(low=0, high=100)
```

In [154…
```
data.mean()
```

Out[154]:
```
x1           92.487743
score        92.556893
weights    1244.125388
dtype: float64
```

In [172…
```
np.average(data['score'], weights=data['weights'])
```

Out[172]:
```
93.91753953120396
```

In [173…
```
np.average(data['x1'], weights=data['weights'])
```

Out[173]:
```
93.96342373249527
```

In [155…
```
data.std()
```

Out[155]:
```
x1          12.108526
score      136.714648
weights    690.430043
dtype: float64
```

Weighted Normal bambi code.

In [156…
```
def logp(value, mu, sigma, weights):
    # This is the same as the logp of a normal, but we have the weights multiplying
    return weights * pm.Normal.logp(value, mu, sigma)

def random(mu, sigma, rng=None, size=None):
    # Weights don't make sense when generating new observations.
    # They are a property of actual observations
    return rng.normal(loc=mu, scale=sigma, size=size)

class WeightedNormal:
    def __new__(self, name, mu, sigma, weights, **kwargs):
        return pm.CustomDist(name, mu, sigma, weights, logp=logp, **kwargs)

    @classmethod
    def dist(cls, mu, sigma, **kwargs):
        return pm.CustomDist.dist(
            mu, sigma, class_name="WeightedNormal", random=random, **kwargs
        )

# Create custom family.
# We need to use the weights in a different way, so we need 'transform_backend_kwargs'.
class WeightedNormalFamily(UnivariateFamily):
    SUPPORTED_LINKS = {"mu": ["identity", "log"], "sigma": ["log"]}
    @staticmethod
    def transform_backend_kwargs(kwargs):
        if "observed" in kwargs:
            observed = kwargs["observed"]
            kwargs["observed"] = observed[:, 0]
            kwargs["weights"] = observed[:, 1]
        return kwargs


# Create the custom family instance
likelihood = bmb.Likelihood("WeightedNormal", params=["mu", "sigma"], parent="mu", dist=WeightedNormal)
links = {"mu": "identity", "sigma": "log"}
wn_family = WeightedNormalFamily("weighted-normal", likelihood, links)
```

First, let's show a weighted regression with just the intercept as the independent variable.

In [157…
```
priors = {
    "Intercept" : bmb.Prior("Normal", mu=92.5, sigma=10),
    "sigma": bmb.Prior("HalfNormal", sigma=136),
}
model1 = bmb.Model("c(score, weights) ~ 1", data, family=wn_family, priors=priors)
```

In [158…
```
model1
```

Out[158]:
```
       Formula: c(score, weights) ~ 1
        Family: weighted-normal
          Link: mu = identity
  Observations: 1000000
        Priors:
    target = mu
        Common-level effects
            Intercept ~ Normal(mu: 92.5, sigma: 10.0)

        Auxiliary parameters
            c(score, weights)_sigma ~ HalfNormal(sigma: 136.0)
```

In [159…
```
idata_m1 = model1.fit(inference_method='vi')
```

100.00% [10000/10000 02:17<00:00 Average Loss = 7.9111e+09]

```
Finished [100%]: Average Loss = 7.9112e+09
```

In [160…
```
approx_sample = idata_m1.sample(10000)
coefs = az.summary(approx_sample)
coefs
```

```
arviz - WARNING - Shape validation failed: input_shape: (1, 10000), minimum_shape: (chains=2, draws=4)
```

Out[160]:

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
Intercept	93.911	0.192	93.55	94.279	0.002	0.001	10225.0	9153.0	NaN
c(score, weights)_sigma	139.981	18.790	105.59	175.742	0.185	0.131	10310.0	9716.0	NaN

Second, let's show an unweighted regression with the intercept and x1 as independent variables.

In [161…
```
priors = {
    "Intercept": bmb.Prior("Normal", mu=0, sigma=10),
    "x1": bmb.Prior("Normal", mu=1, sigma=0.1),
    "sigma": bmb.Prior("HalfNormal", sigma=136)
}
model2 = bmb.Model("score ~ x1", data, priors=priors)
```

In [162…
```
model2
```

Out[162]:
```
       Formula: score ~ x1
        Family: gaussian
          Link: mu = identity
  Observations: 1000000
        Priors:
    target = mu
        Common-level effects
            Intercept ~ Normal(mu: 0.0, sigma: 10.0)
            x1 ~ Normal(mu: 1.0, sigma: 0.1)

        Auxiliary parameters
            score_sigma ~ HalfNormal(sigma: 136.0)
```

In [163…
```
idata_m2 = model2.fit(inference_method='vi')
```

100.00% [10000/10000 03:55<00:00 Average Loss = 6.5353e+06]

```
Finished [100%]: Average Loss = 6.5351e+06
```

In [164…
```
approx_sample = idata_m2.sample(10000)
coefs = az.summary(approx_sample)
coefs
```

```
arviz - WARNING - Shape validation failed: input_shape: (1, 10000), minimum_shape: (chains=2, draws=4)
```

Out[164]:

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
Intercept	2.651	0.692	1.326	3.917	0.007	0.005	9204.0	9882.0	NaN
x1	1.005	0.134	0.742	1.243	0.001	0.001	10010.0	10003.0	NaN
score_sigma	166.475	22.527	124.495	208.041	0.228	0.161	9652.0	9277.0	NaN

Last, let's show a weighted regression with the intercept and x1 as dependent variables.

In [165…
```
priors = {
    "Intercept": bmb.Prior("Normal", mu=0, sigma=10),
    "x1": bmb.Prior("Normal", mu=1, sigma=0.1),
    "sigma": bmb.Prior("HalfNormal", sigma=136)
}
model3 = bmb.Model("c(score, weights) ~ x1", data, family=wn_family, priors=priors)
```

In [166…
```
model3
```

Out[166]:
```
       Formula: c(score, weights) ~ x1
        Family: weighted-normal
          Link: mu = identity
  Observations: 1000000
        Priors:
    target = mu
        Common-level effects
            Intercept ~ Normal(mu: 0.0, sigma: 10.0)
            x1 ~ Normal(mu: 1.0, sigma: 0.1)

        Auxiliary parameters
            c(score, weights)_sigma ~ HalfNormal(sigma: 136.0)
```

In [167…
```
idata_m3 = model3.fit(inference_method='vi')
```

100.00% [10000/10000 04:17<00:00 Average Loss = 8.1268e+09]

```
Finished [100%]: Average Loss = 8.1267e+09
```

In [168…
```
approx_sample = idata_m3.sample(10000)
coefs = az.summary(approx_sample)
coefs
```

```
arviz - WARNING - Shape validation failed: input_shape: (1, 10000), minimum_shape: (chains=2, draws=4)
```

Out[168]:

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
Intercept	2.627	0.684	1.352	3.904	0.007	0.005	10225.0	9178.0	NaN
x1	1.915	0.161	1.623	2.227	0.002	0.001	9808.0	9794.0	NaN
c(score, weights)_sigma	166.295	22.371	125.923	208.628	0.229	0.162	9576.0	8827.0	NaN

This same larger mean coefficient does not happen with sklearn linear regression.

In [169…
```
from sklearn.linear_model import LinearRegression

reg = LinearRegression().fit(X=data[['x1']], y=data['score'], sample_weight=data['weights'])
```

In [170…
```
print(reg.intercept_, reg.coef_)
```

```
-0.7225752528107989 [1.00720164]
```

In [ ]: