Code Quality in Different Programming Languages

Probabilistic Programming 2024 Exam by Raúl Pardo and Andrzej Wąsowski
version 1.0.0 2024-05-01 10:00

The group consists of 3 members:

- Anders Hjulmand (`ahju@itu.dk`)
- Andreas Flensted (`frao@itu.dk`)
- Eisuke Okuda (`eiok@itu.dk`)

We hereby declare that we have answered these exam questions ourselves without any outside help.

Outline / table of contents

Introduction

This project examines software quality in a dataset of $N = 1127$ records of fragments of GitHub projects that are written in different programming languages. The software quality is defined as the number of commits classified as bugs. We examine whether the choice of programming language affects the number of bugs,

and in particular whether the language Haskell is more prone to contain bugs compared to other langauges. In addition we investegate if the age of a project/language combination has a real effect on the number of bugs, or whether the relationship between age and bugs is a spurious relationship caused by commits.

The hypotheses are as follows:

- **H1** - Haskell code is less prone to contain bugs (B). In other words, the distribution on the number of bugs (B) for Haskell gives high probability to the lowest number of bugs among all programming languages (L).
- **H2** - Age (A) has a positive impact on number of bugs (B) for all programming languages (L). That is, projects of old age (A) have larger number of bugs (B).
- **H3** - Number of commits (C) does not impact the effect of age (A) on the number of bugs (B) for any programming language (L). That is, the effect of age (A), conditioned on number of commits (C), on number of bugs (B) is the same as the direct effect of age (A) on number of bugs (B).

Utils

In [ ]:
```
# imports
import numpy as np
import pandas as pd
import arviz as az
import pymc as pm
import xarray as xr
import matplotlib.pyplot as plt
from scipy.stats import poisson
import seaborn as sns
from scipy.special import expit
from causalgraphicalmodels import CausalGraphicalModel

#warnings
import warnings
warnings.filterwarnings('ignore')
```

In [ ]:
```
#style
az.style.use("arviz-whitegrid")
```

In [ ]:
```
#seed
RANDOM_SEED = 8927
RNG = np.random.default_rng(RANDOM_SEED)
```

Helper Functions

In [ ]:
```
def standardize_column(column):
    '''
    Transforms column by its z-score.
    '''
    standardized_column = (column - column.mean()) / column.std()
```

```
    return standardized_column

def transform_exp(x):
    '''
    Transform a value to its exponential value.
    '''
    return np.exp(x)
```

Data Cleaning & Exploratory Data Analysis

In [ ]:
```
## load data
df = pd.read_csv('dataset.csv')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1127 entries, 0 to 1126
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   project       1127 non-null   object
 1   language      1127 non-null   object
 2   commits       1127 non-null   int64 
 3   insertions    1127 non-null   int64 
 4   age           1127 non-null   int64 
 5   bugs          1127 non-null   int64 
 6   project_type  1127 non-null   object
 7   devs          1127 non-null   int64 
dtypes: int64(5), object(3)
memory usage: 70.6+ KB
```

We note that none of the columns in the dataset contain missing values.

Standardization

We z-score standardize `age`, `commits` and `bugs` to have mean centered variables and unit standard deviation.

In [ ]:
```
# standardize columns
df['age_std'] = standardize_column(df['age'])
df['commits_std'] = standardize_column(df['commits'])
df['bugs_std'] = standardize_column(df['bugs'])
```

Catagorical Feature exploration

In [ ]:
```
df_cat = df.select_dtypes(include = ['O'])
for column in df_cat.columns:
  print('============================')
  print(f'Column name: {column}')
  print('============================')
  print(f'# of unique values: {len(df_cat[column].value_counts())}')
  print(f'# of average counts: {df_cat[column].value_counts().mean()}')
  print()
```

```
============================
Column name: project
============================
# of unique values: 729
# of average counts: 1.5459533607681757

============================
Column name: language
============================
# of unique values: 17
# of average counts: 66.29411764705883

============================
Column name: project_type
============================
# of unique values: 7
# of average counts: 161.0
```

In [ ]:
```
## factorize categorical
df["language"], languages = pd.factorize(df["language"])
df["project_type"], project_types = pd.factorize(df["project_type"])
```

Main Analysis

H1

Poisson Regression

H1 - Haskell code is less prone to contain bugs (B). In other words, the distribution on the number of bugs (B) for Haskell gives high probability to the lowest number of bugs among all programming languages (L).

We model the number of bugs with a poisson regression that includes an intercept for each language, thus getting the expected count of bugs for some language with the following model:

$$B_i \sim Poisson(\lambda_i)$$
$$log(\lambda_i) = \alpha_L$$
$$\alpha_L = \text{not yet determined}$$

Where :

$$L \in \mathcal{L}, where\ \mathcal{L} = \{Python, Java.., C\}$$
$$|\mathcal{L}| = 17$$

Prior Predictive Checks

Now what is a good choice of priors for α_L?

- One thing we can we say for sure is that the number of counts cannot be negative but the question is how many bugs an average project might have? First of all, there need to be some software pushed to the project, which is denoted by the number of commits **C**. As soon as a project has one commit, there must also be some probability that there is some bug in that pushed code. Based on own work experience, the number commits depending on what type of project, can range from a few commits to a 100. Some projects in the dataset we believe are much bigger meaning they have many more commits. A naive liveable space for the expected number of bugs would be in the interval $\lambda_i \in [0, \infty)$. Given our own experience, say that our project has a 100 commits and we are terrible programmers, this project would have a 100 bugs, so this is still a very large upper bound for the number of bugs. A more reasonable expected number of bugs could be a 1000.

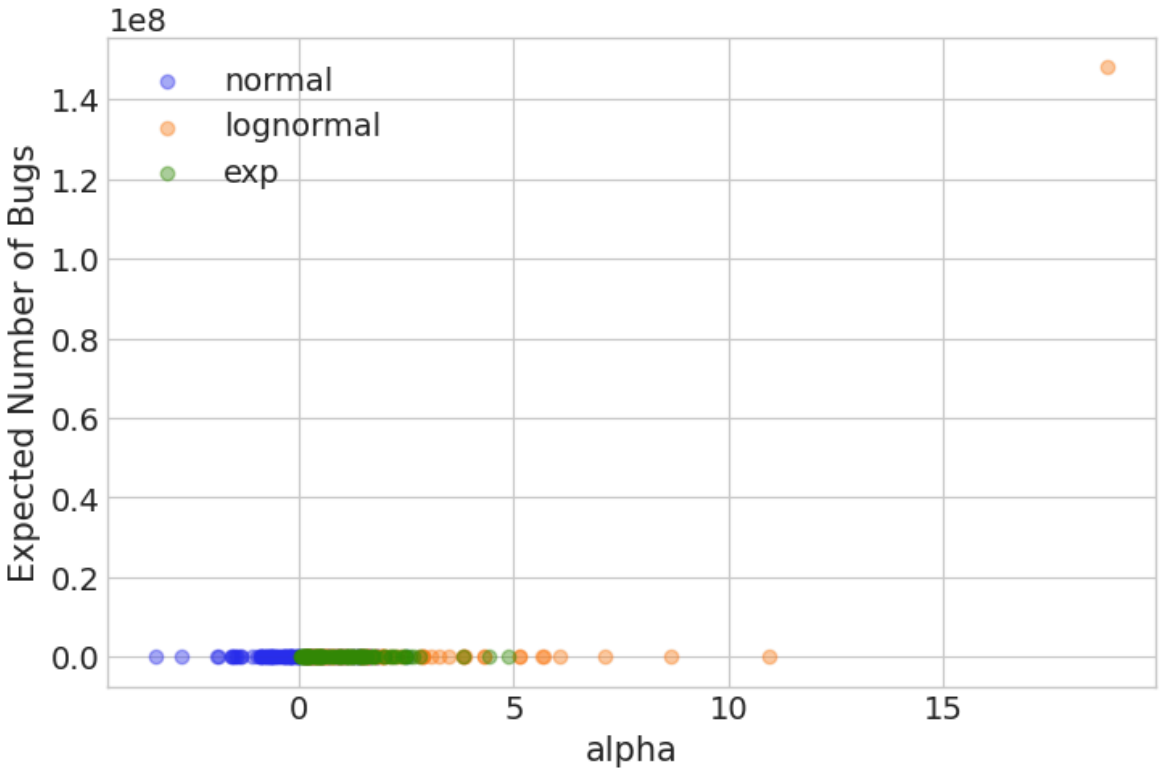
- This belief about λ_i is aimed to be reflected in the choice of prior for α_L that is the average number of bugs for a project.

- Since the expected value of λ has a logarithmic relationship to α_L, the choice of priors can lead to exploding number of bugs using a prior with high variance. Another common choice of prior distribution, the standard normal $N(0, 1)$, could lead to negative samples, that are mapped to expected value of bugs around 0. This amount of bugs in a project might also seem to good to be true.

- We explore the logarthmic relationship between α_L and the number of bugs with the following candidate prior distributions for α_L:

  - $\alpha_L \sim Exp(1)$
  - $\alpha_L \sim N(0, 1)$
  - $\alpha_L \sim log\ N(0, 1)$

In [ ]:
```
# First Initial Check
n_draws = np.random.normal(0, 1, 100)
ln_draws = np.random.lognormal(0,1,100)
exp_draws = np.random.exponential(1, 100)

plt.scatter(n_draws,np.exp(n_draws),label="normal", alpha=0.4)
plt.scatter(ln_draws,np.exp(ln_draws),label="lognormal", alpha=0.4)
plt.scatter(exp_draws,np.exp(exp_draws), label="exp",alpha=0.4)
plt.xlabel("alpha")
plt.ylabel("Expected Number of Bugs")
plt.legend()
plt.show()
```



From the plot above, it is already clear that the lognormal distribution yields exploding values for the number of bugs on average.

We can also see, that the normal distribution yields (as expected) a bunch of negative α values, that results in the average number of bugs to be very close to 0. The exponential distribution's heavy tailed characteristic shows from this initialisation a greater tendency than the standard normal $N(0, 1)$ to yield exploding values.

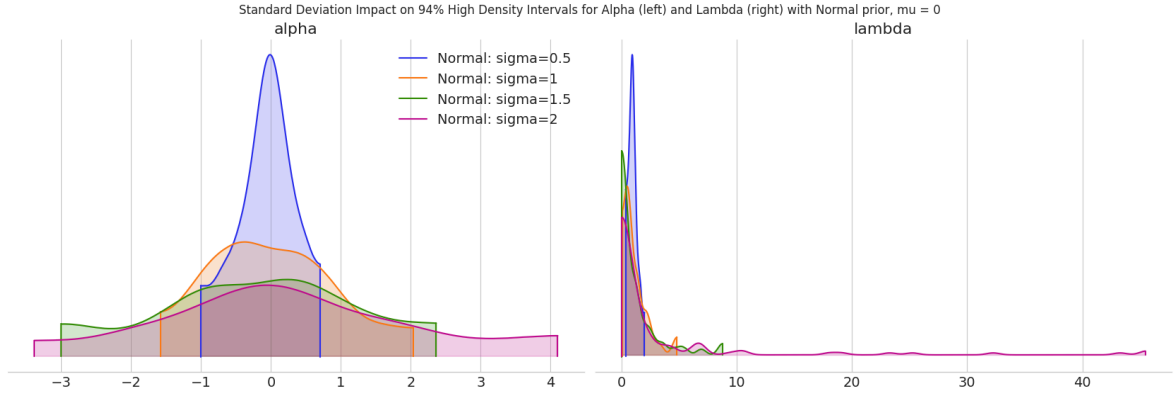
Next we run the prior predictive checks comparing the normal and exponential distribution to find a good informative prior. This is done by fixing the normal prior $\mu = 0$ and varying σ, as well as varying the rate λ for the $Exp(\lambda)$

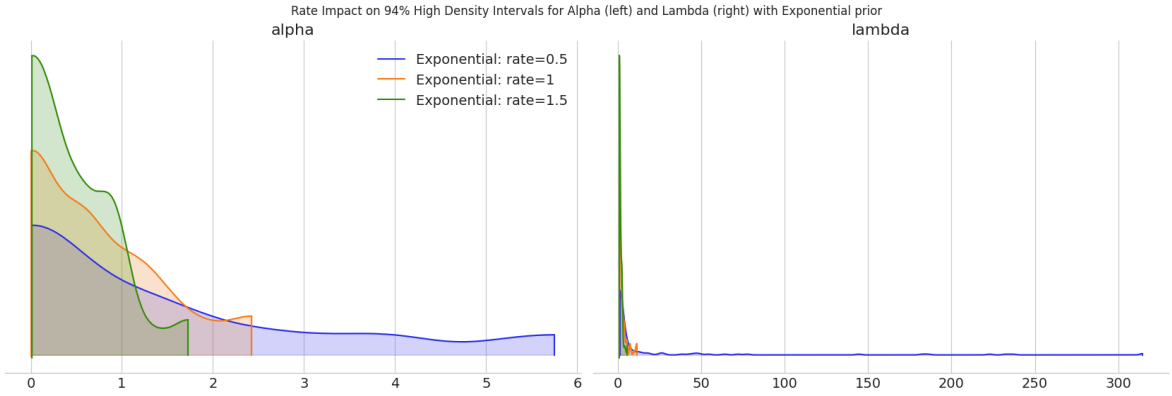
In [ ]:
```
def prior_predictive_check_alpha_normal(alpha_mu_prior, alpha_sigma_prior
    with pm.Model() as model:
        alpha = pm.Normal("alpha", mu=alpha_mu_prior, sigma=alpha_sigma_p
        lam = pm.Deterministic('lambda', pm.math.exp(alpha))
        trace = pm.sample_prior_predictive(samples=100)

    return trace, model

def prior_predictive_check_alpha_exp(alpha_lambda_prior):
    with pm.Model() as model:
        alpha = pm.Exponential("alpha", alpha_lambda_prior)
        lam = pm.Deterministic('lambda', pm.math.exp(alpha))
        trace = pm.sample_prior_predictive(samples=100)

    return trace, model
```

In [ ]:
```
# Fixed Mu, varying sigmas
# Varying rate/lambda parameters
mu = 0
n_sigmas = [0.5, 1, 1.5, 2]
exp_lambdas = [0.5, 1, 1.5]
```

```
normal_traces, exponential_traces = [], []
normal_labels, exponential_labels = [], []


for sigma in n_sigmas:
    trace, model = prior_predictive_check_alpha_normal(mu, sigma)
    normal_traces.append(trace.prior)
    normal_labels.append(f"Normal: sigma={sigma}")

for exp in exp_lambdas:
    trace, model = prior_predictive_check_alpha_exp(exp)
    exponential_traces.append(trace.prior)
    exponential_labels.append(f"Exponential: rate={exp}")

axes = az.plot_density(
    normal_traces,
    data_labels=normal_labels,
    var_names=["alpha", "lambda"],
    shade=0.2,
    point_estimate=None,
)

fig = axes.flatten()[0].get_figure()
fig.suptitle("Standard Deviation Impact on 94% High Density Intervals for

axes = az.plot_density(
    exponential_traces,
    data_labels=exponential_labels,
    var_names=["alpha", "lambda"],
    shade=0.2,
    point_estimate=None,
)

fig = axes.flatten()[0].get_figure()
fig.suptitle("Rate Impact on 94% High Density Intervals for Alpha (left)

plt.show()
```

```
Sampling: [alpha]
Sampling: [alpha]
Sampling: [alpha]
Sampling: [alpha]
Sampling: [alpha]
Sampling: [alpha]
Sampling: [alpha]
```

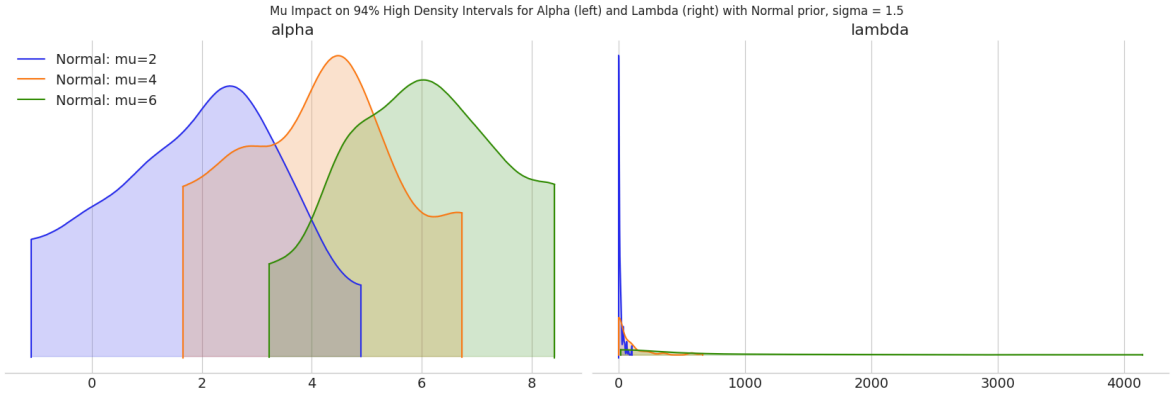




- Based on the above KDE plots and prior predictive checks for Normal distribution with varying σ and fixed $\mu = 0$ and an exponential distribution with varying rates, it is clear that small changes in the rate λ parameter of an exponential distribution is overly sensitive to changes in terms of pushing expected number of bugs too close to 0 or stretches it too far out, as it is seen with $\lambda = 0.5$. The normal distribution seem to be more a suitable choice of priors for the α, although even with a $\sigma = 2$ the simulated number of bugs gets a very heavy tail. Now to find the best prior, we fix the standard deviation of 1.5, and try to rid the models from most of the negative values by increasing the μ parameter.

In [ ]:
```
#Fixed Sigma, varying mu
sigma = 1.5
n_mus = [2,4,6]

traces, labels = [], []

for mu in n_mus:
    trace, model = prior_predictive_check_alpha_normal(mu, sigma)
    traces.append(trace.prior)
    labels.append(f"Normal: mu={mu}")

axes = az.plot_density(
    traces,
    data_labels=labels,
    var_names=["alpha", "lambda"],
    shade=0.2,
    point_estimate=None,
)

fig = axes.flatten()[0].get_figure()
fig.suptitle("Mu Impact on 94% High Density Intervals for Alpha (left) an
plt.show()
```

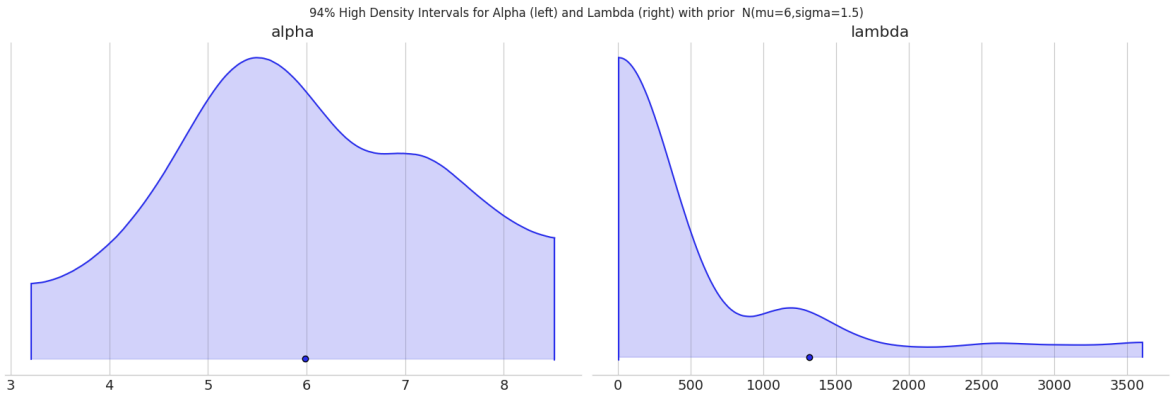
```
Sampling: [alpha]
Sampling: [alpha]
Sampling: [alpha]
```



Based on the varying means in the prior predictive sampling for the parameter α and its influence on the outcome being the expected count of bugs denoted by λ. From these observations we set the prior for $\alpha_L \sim N(6, 1.5)$. As highlighted below, the probability mass of λ_i lies within the range of $[0, \approx 5000)$. This is alot more than our own personal projects have had of commits and bugs, but these have also been tiny projects, and therefore not representative of what we expect from the population of code projects.

In [ ]:
```
axes = az.plot_density(
    [prior_predictive_check_alpha_normal(6, 1.5)[0].prior],
    data_labels=["Prior Alpha selection = mu=4, sigma=1.5"],
    var_names=["alpha", "lambda"],
    shade=0.2,
    point_estimate="mean",
)

fig = axes.flatten()[0].get_figure()
fig.suptitle("94% High Density Intervals for Alpha (left) and Lambda (rig
plt.show()
```

```
Sampling: [alpha]
```



Model Fitting

We model the expected number of bugs as follows:

$$B_i \sim Poisson(\lambda_i)$$
$$log(\lambda_i) = \alpha_L$$
$$\alpha_L \sim N(6, 1.5)$$

In [ ]:
```
## Poisson Modeling
with pm.Model() as h1_poisson_model:

      #Data
      language = pm.Data("language", df.language, mutable=True)
      # Priors
      alpha = pm.Normal("alpha", mu=6, sigma=1.5, shape=len(languages))

      #f(theta, x)
      lam= pm.Deterministic('lambda', pm.math.exp(alpha[language]))

      #likelihood
      B = pm.Poisson('B', mu=lam, observed=df.bugs)

      h1_poisson_trace = pm.sample(2000, tune=2000,idata_kwargs={'log_lik
```

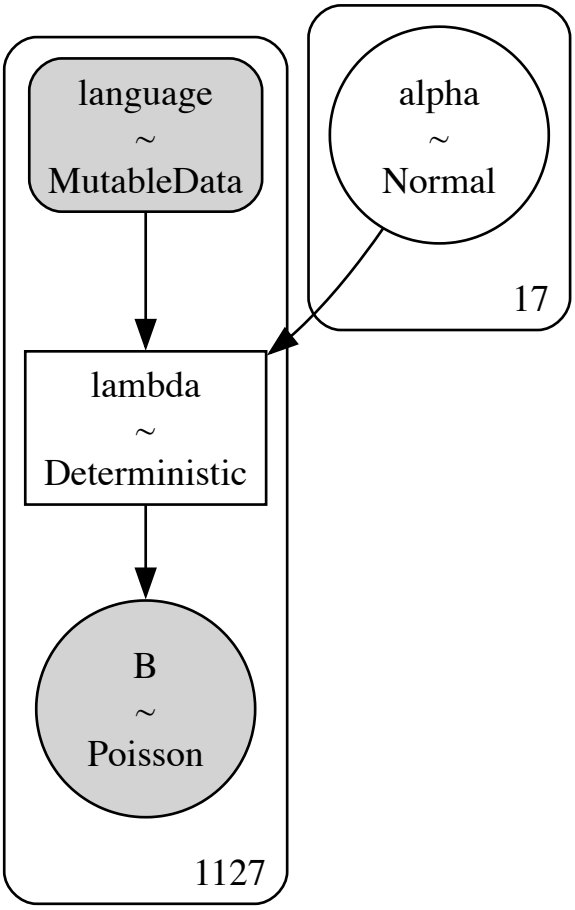
```
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [alpha]
```

24.31% [3889/16000 00:00<00:02
Sampling 4 chains, 0 divergences]

```
Sampling 4 chains for 2_000 tune and 2_000 draw iterations (8_000 + 8_000
draws total) took 3 seconds.
```

In [ ]:
```
pm.model_to_graphviz(model=h1_poisson_model)
```

Out[ ]:



Below is the trace convergence for the parameter α_L.

The number of generated samples is 4 chains * 2000 samples = 8000 samples.

As seen in the table below, the Monte Carlo Standard Error mean `mcse_mean` is 0 for all parameters. These indicate effective sampling of the mode of the posterior distributions in all 4 chains, whilst the equivalent entries in `mcse_sd` show the chains could sample from the mode effectively without deviation. This is also shown by the high number of `ess_bulk` and `ess_tail` indicating that new samples provided new information about the posterior distribution.

In addition, all the `rhat` values are 1.0 which means the 4 chains converged. This is also reflected in the trace plot below, which shows good mixing of the chains.

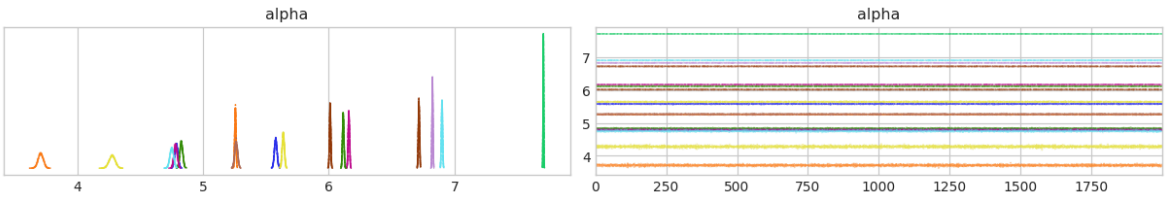
In summary, the convergence of the trace for the poisson regression is good and shows no signs of inefficient of inaccuracte sampling. We can therefore use the posterior distributions with high confidence.

In [ ]:
```
pm.summary(h1_poisson_trace, var_names=["alpha"], round_to=2)[['mcse_mean
       'ess_tail', 'r_hat']]
```

Out[ ]:

	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
alpha[0]	0.0	0.0	15659.27	5916.26	1.0
alpha[1]	0.0	0.0	14574.74	5987.23	1.0
alpha[2]	0.0	0.0	16902.96	5686.36	1.0
alpha[3]	0.0	0.0	14147.68	6175.94	1.0
alpha[4]	0.0	0.0	14967.98	6093.07	1.0
alpha[5]	0.0	0.0	13266.80	5517.90	1.0
alpha[6]	0.0	0.0	15032.16	6049.67	1.0
alpha[7]	0.0	0.0	16271.04	6222.98	1.0
alpha[8]	0.0	0.0	15412.74	5925.11	1.0
alpha[9]	0.0	0.0	16591.55	5495.09	1.0
alpha[10]	0.0	0.0	14423.99	6033.72	1.0
alpha[11]	0.0	0.0	12542.17	5575.47	1.0
alpha[12]	0.0	0.0	15256.05	5645.60	1.0
alpha[13]	0.0	0.0	15240.04	5801.58	1.0
alpha[14]	0.0	0.0	12700.50	5799.93	1.0
alpha[15]	0.0	0.0	13848.04	5610.27	1.0
alpha[16]	0.0	0.0	15384.85	5463.54	1.0

In [ ]:
```
pm.plot_trace(h1_poisson_trace, var_names=['alpha']);
```



Below is a forest plot and a numerical representation of the means and standard deviations (sd) from the posterior distributions of α_L. The posterior estimates estimates of the α_L means, have moved away from the prior mean of 6 while also reducing their estimated standard deviations from 1.5 to ≈ 0.01. This shows that the expected number of bugs was overestimated by the prior for some languages while for others it was underestimated.

We see notable differences in the effect between languages on the expected number of bugs. For example that `C` is estimated to have the highest number of bugs while `Typescript` is estimated to have the least amount of expected bugs.

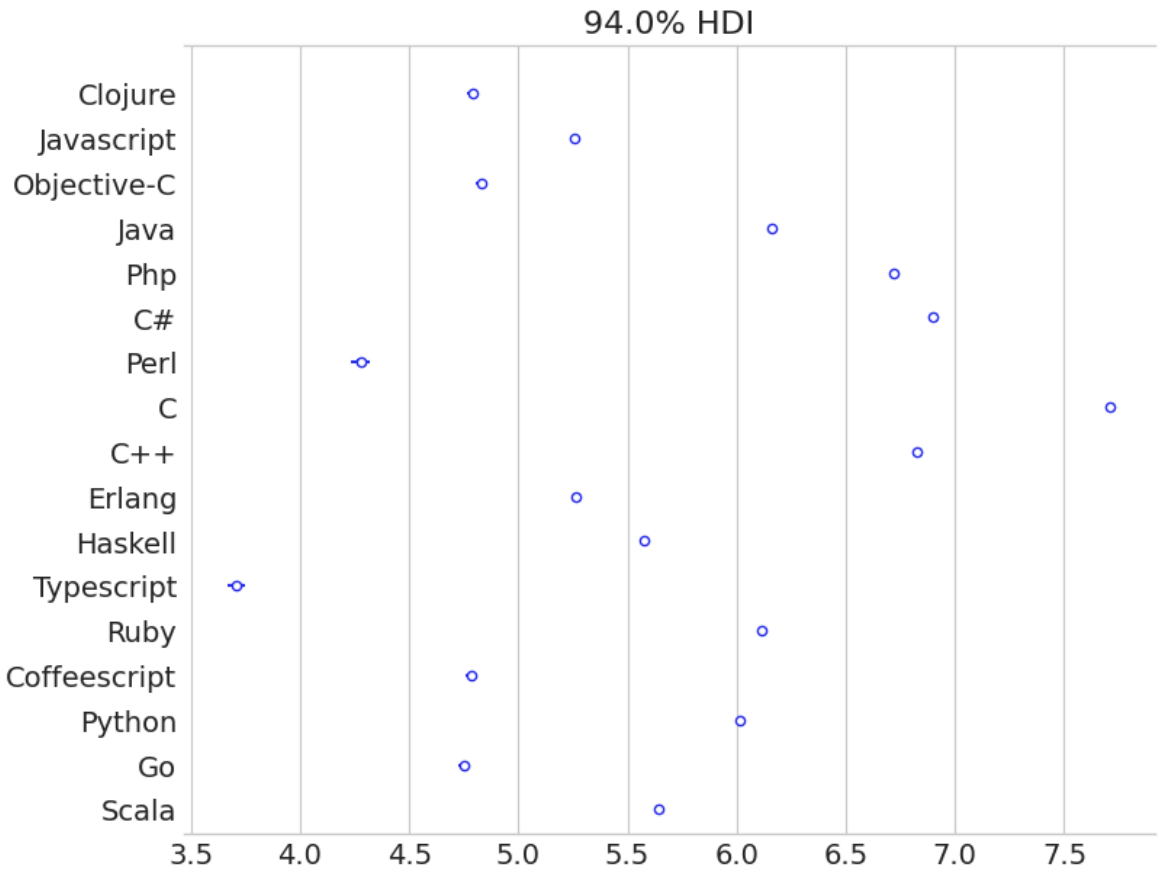
In [ ]:
```
summary_alpha_poisson = pm.summary(h1_poisson_trace, var_names=['alpha'],
summary_alpha_poisson
summary_alpha_poisson["language"] = languages.take(summary_alpha_poisson.
summary_alpha_poisson
```

Out[ ]:

	parameter	mean	sd	hdi_3%	hdi_97%	language
0	alpha[0]	4.7895	0.0127	4.7659	4.8138	Clojure
1	alpha[1]	5.2579	0.0050	5.2484	5.2675	Javascript
2	alpha[2]	4.8284	0.0116	4.8063	4.8502	Objective-C
3	alpha[3]	6.1601	0.0053	6.1503	6.1703	Java
4	alpha[4]	6.7174	0.0045	6.7089	6.7259	Php
5	alpha[5]	6.9008	0.0044	6.8925	6.9093	C#
6	alpha[6]	4.2779	0.0239	4.2320	4.3215	Perl
7	alpha[7]	7.7069	0.0023	7.7027	7.7114	C
8	alpha[8]	6.8251	0.0036	6.8183	6.8315	C++
9	alpha[9]	5.2643	0.0112	5.2432	5.2852	Erlang
10	alpha[10]	5.5788	0.0101	5.5596	5.5978	Haskell
11	alpha[11]	3.7103	0.0203	3.6703	3.7475	Typescript
12	alpha[12]	6.1155	0.0056	6.1053	6.1266	Ruby
13	alpha[13]	4.7826	0.0128	4.7588	4.8063	Coffeescript
14	alpha[14]	6.0117	0.0049	6.0023	6.0206	Python
15	alpha[15]	4.7522	0.0148	4.7248	4.7806	Go
16	alpha[16]	5.6389	0.0088	5.6222	5.6555	Scala

In [ ]:
```
_,ax = plt.subplots(figsize=(8,6))
az.plot_forest(h1_poisson_trace, var_names=["alpha"], combined=True, figs
```

```
forest_languages = [int(i.get_text()[1:-1]) if len(i.get_text()) < 6 else
ax.set_yticklabels(languages.take(forest_languages));
```



Posterior Predictive Check

In [ ]:
```
pareto_k_h1 = az.loo(h1_poisson_trace, pointwise=True).pareto_k.values
print("Max PSIS value for alpha_language: ", max(pareto_k_h1))
```

```
Max PSIS value for alpha_language:  67.51666091041223
```

We get the following warning:

"Estimated shape parameter of Pareto distribution is greater than 0.7 for one or more samples. You should consider using a more robust model, this is because importance sampling is less likely to work well if the marginal posterior and LOO posterior are very different. This is more likely to happen with a non-robust model and highly influential observations"

This warning indicates that some projects are highly influential on the model, and that taking them out at random, changes the posterior distribution.

We sample 1127 entries with randomly assigned languages and use those for posterior predictions:

In [ ]:
```
n_points = len(df)
x_seq = np.random.randint(size=n_points, low=0, high=len(languages))

with h1_poisson_model:
    pm.set_data({"language": x_seq})
    post_pred_h1 = pm.sample_posterior_predictive(h1_poisson_trace, var_n
```

```
post_pred_h1_mean = post_pred_h1.mean(["chain", "draw"])
```

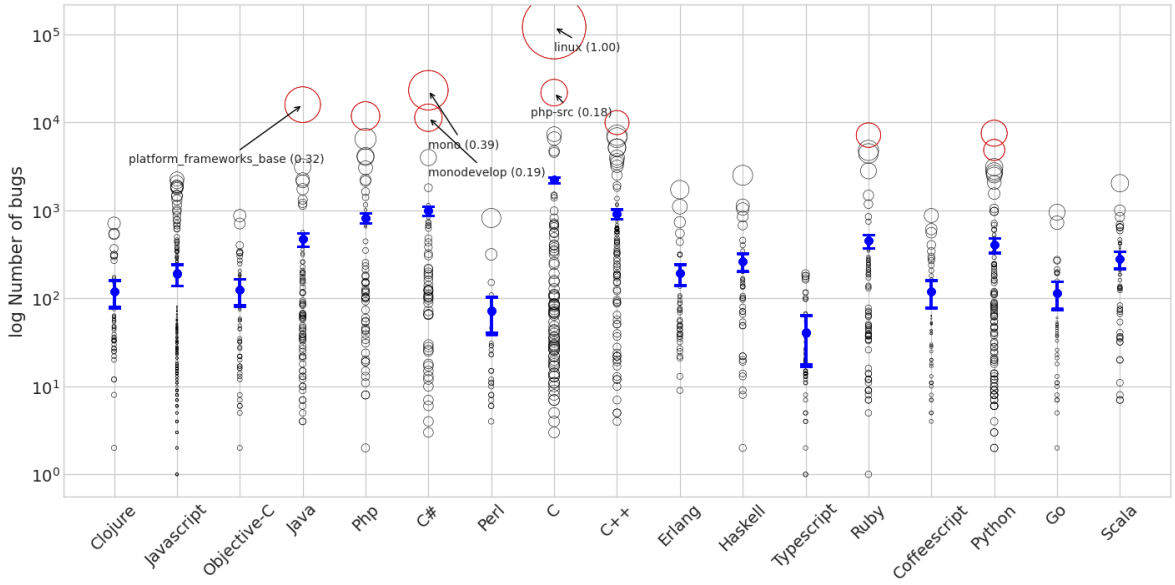
```
Sampling: [B]
```

To investigate the posterior predictions and the highly influential datapoints, below is plotted the `h1_poisson_model` mean prediction and 94% HDI as well as the individual observations for each datapoint. Each entry in the plot is sized by its pareto smooth importance value.

It is clear from the plot, that some projects like `Linux` written in `C` has more bugs than twice the order of magnitude of the other projects. This is also denoted by the red circle and its value of 1, that originally was (as seen above) 66.56. The other projects relative `PSIS` values have equally been normalized, and required a scaling in size of 3000, to be used for the plot below. This indicates that some projects in combination with some languages have a great outlying number of bugs compared to the average estimation.

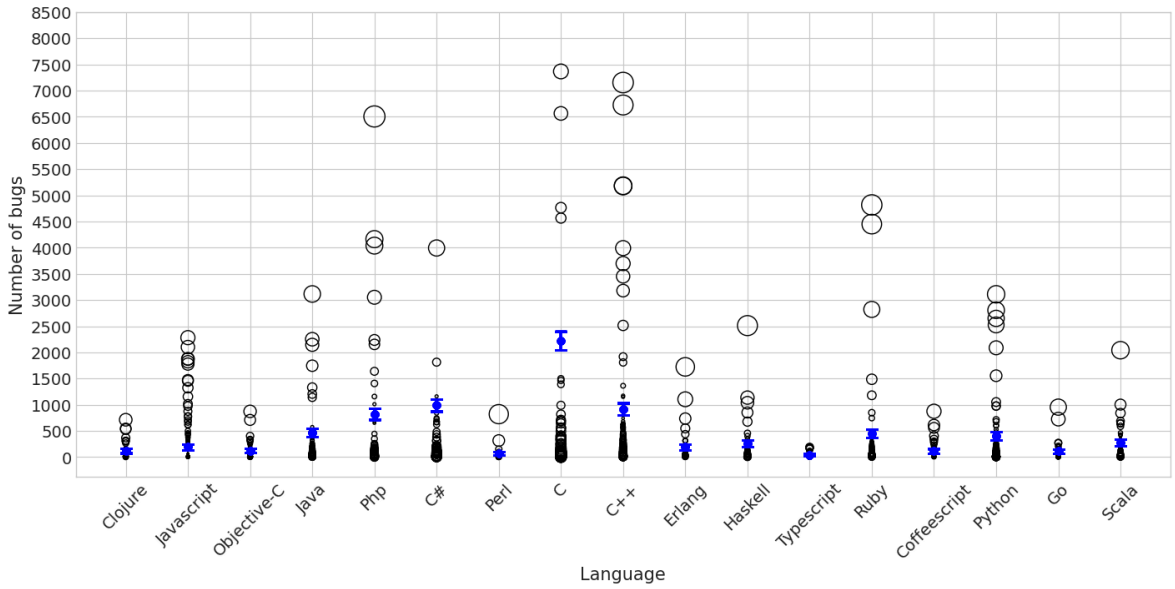
In [ ]:
```
#scale pareto-k values by their max and changing make them nice to plot
pareto_k_h1 /= pareto_k_h1.max()
pareto_k_h1_size = 3000 * pareto_k_h1

#Get the 10 data points with the largest pareto_k_values
top_indices = np.argsort(pareto_k_h1)[-10:][::-1]
top_data_points = df.iloc[top_indices]

#compute error bars of 94 HDI
hdi = az.hdi(post_pred_h1)
lower_bound, upper_bound = hdi.data_vars["B"].to_numpy()[:,0], hdi.data_v
diffs = upper_bound - lower_bound

fig, axes = plt.subplots(figsize=(14,7))
axes.set_yscale("log")

axes.errorbar(x_seq, post_pred_h1_mean, yerr=diffs, fmt='o', color='b', e
axes.scatter(df.language, df.bugs, s = pareto_k_h1_size, facecolors='none
axes.scatter(top_data_points.language, top_data_points.bugs, s = pareto_k
ticks = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]
axes.set_ylabel("log Number of bugs")
axes.set_xticks(ticks=ticks)
axes.set_xticklabels(labels=languages, rotation=45)

#Plotting text for data points with high pareto-k values. We manually adj
for index, row in top_data_points.iterrows():
    # print(row)
    dont_annotate = False
    if row['language'] == 7 and row['project'] == 'linux':
        position=(0,-20)
    elif row['language'] == 1:
        position=(-70,15)
    elif row['language'] == 5:
        position=(0,-50)
    elif row['language'] == 7 and row['project'] == 'php-src':
        position=(-20,-20)
    elif row["language"] == 3:
```

```
        position = (-150,-50)
    else:
        dont_annotate = True
    if dont_annotate == False:
        axes.annotate(f"{row['project']} ({pareto_k_h1[index]:.2f})", xy=
plt.show()
```



As seen above, the 10 projects with the greatest `PSIS` values, have been plotted in red. Below is the same plot but without these 10 influential datapoints and on a linear scale.

In [ ]:
```
fig,axes = plt.subplots(figsize=(14,7))

pareto_k_size_without_top = np.delete(pareto_k_h1_size, top_indices)
df_without_top = df.drop(top_indices)

axes.errorbar(x_seq, post_pred_h1_mean, yerr=diffs, fmt='o', color='b', e
axes.scatter(df_without_top.language, df_without_top.bugs, facecolors='no
axes.set_xlabel('Language')
axes.set_ylabel('Number of bugs')
ticks = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]
axes.set_xticks(ticks=ticks)
axes.set_xticklabels(labels=languages, rotation=45)
plt.yticks(np.arange(0, 9000, 500));
```



From the plotted mean prediction and 94% HDI interval of `h1_poisson_model`, most of the languages have an expected number of bugs somewhere between 0 and 1000. An outlier is the prediction for `C` which has a mean of 2224 bugs.

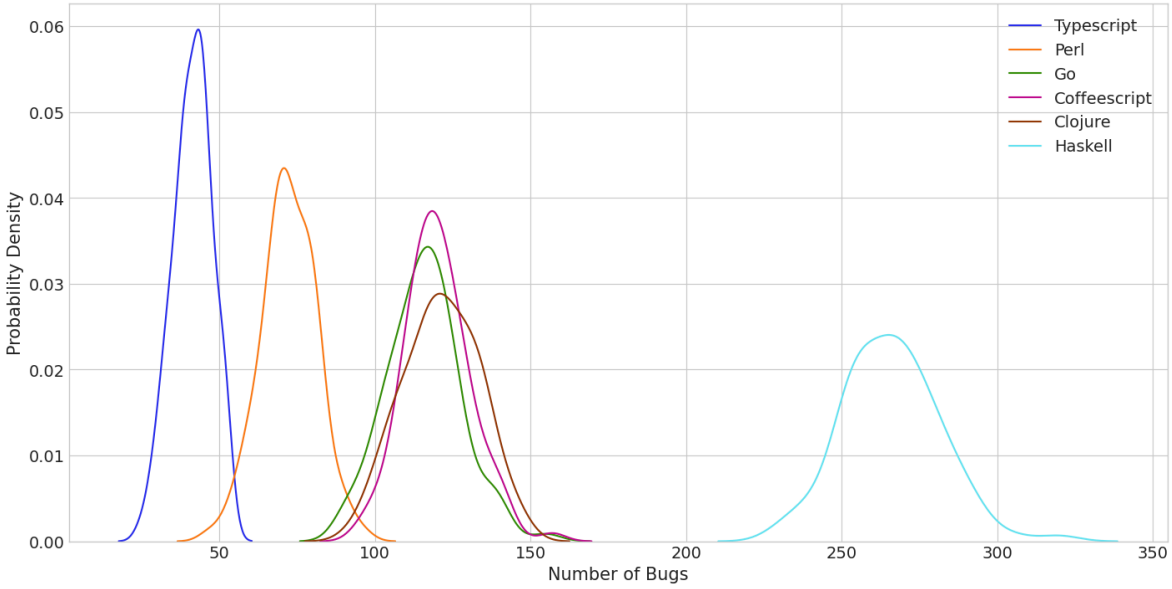
From the plot it is not clear which of the posterior distributions of λ_L is the smallest. Specifically we are interested in whether programming in Haskell yields the smallest probability to bugs given the model parameters and the data. To explore this, we look at the numerical estimates of the posterior λ_L means. Secondly, we plot the Kernel Density Estimates from each of the top 5 λ_L posterior distributions that yields the highest probability to the smallest number of bugs. This plot is also extended with the Haskell Kernel Density Estimate for comparison.

In [ ]:
```
n_points = len(df)
x_seq = np.array(range(len(languages)))

with h1_poisson_model:
    pm.set_data({"language": x_seq})
    post_pred_h1_plot = pm.sample_posterior_predictive(h1_poisson_trace,

post_pred_h1_mean_plot = post_pred_h1_plot.mean(["chain", "draw"])

seaborn_plot_data = pd.DataFrame({"lambdas": post_pred_h1_mean_plot, "lan
seaborn_plot_data.sort_values("lambdas", ascending=True, inplace=True)
print("Numerically ranked languages by expected number of bugs:\n \n" , s
_, axes = plt.subplots(figsize=(14, 7))

#Generating Poisson data
for language, expected_count in zip(seaborn_plot_data["language"].to_list
    sample = poisson.rvs(mu=expected_count, size=100)
    sns.kdeplot(x=sample, fill=False, label=f'{language}', ax=axes)

sample = poisson.rvs(seaborn_plot_data[seaborn_plot_data["language"]=="Ha
sns.kdeplot(x=sample, fill=False, label="Haskell", ax=axes)

axes.set_xlabel('Number of Bugs')
axes.set_ylabel('Probability Density')
plt.legend()
plt.show()
```

```
Sampling: []
```

100.00% [8000/8000 00:00<00:00]

```
Numerically ranked languages by expected number of bugs:

         lambdas      language
0     40.873363    Typescript
1     72.112336          Perl
2    115.857082            Go
3    119.424441   Coffeescript
4    120.248315       Clojure
5    125.013254   Objective-C
6    192.079344    Javascript
7    193.318562        Erlang
8    264.771203       Haskell
9    281.165680         Scala
10   408.167375        Python
11   452.845789          Ruby
12   473.486962          Java
13   826.660347           Php
14   920.656450           C++
15   993.095005            C#
16  2223.692969             C
```



Part Conclusion H1

Based on the above results, we can see that Haskell is not in the top 5 languages with lowest number of expected bugs. Based on these results we reject H1. The programming language that would fulfill that hypothesis based on `h1_poisson_model` would be Typescript with its $\lambda = 41$ i.e. the expected bugs for an average project. This is also seen from the posterior mean of $\alpha_{typescript} = 3.7111$ which is the smallest value for α_L.

Can we model this in any other way?

As of now we have only modelled the probability of the least amount of bugs among all programming languages with `h1_poisson_model` to which H1 is rejected. The danger of a GLM poisson regression with this dataset is that if a certain language is used in larger projects in terms of commits, the model will pick up on some

misleading relationships between languages and the number of bugs, simply because the language has been used in projects with many bugs. This hypothesis could also be modelled by a **binomial regression**, where the probability p of a success - a bug - is modelled by k number of successes (*bugs*) in N number of bernoulli trials (*Commits*).

Now follows the same procedure as above, to see whether using a binomial regression changes our conclusion to **H1**.

Binomial Regression

The model includes an α_L to see the average effect of a language on the probability of a bug.

$$\begin{aligned} B_i &\sim \text{Binomial}(N_i, p_i) \\ \text{logit}(p_i) &= \alpha_L \\ \alpha_L &\sim \text{Not yet Determined} \end{aligned}$$

We took inspiration for prior predictive checks from lecture 09. Here it is important to stress, that the probability mass for the outcome variable p_i has to be as evenly distributed between $(0, 1)$, such that the prior assumption is neither only successes or only failures for each independent bernoulli trial of commits.

Prior Predictive Checks

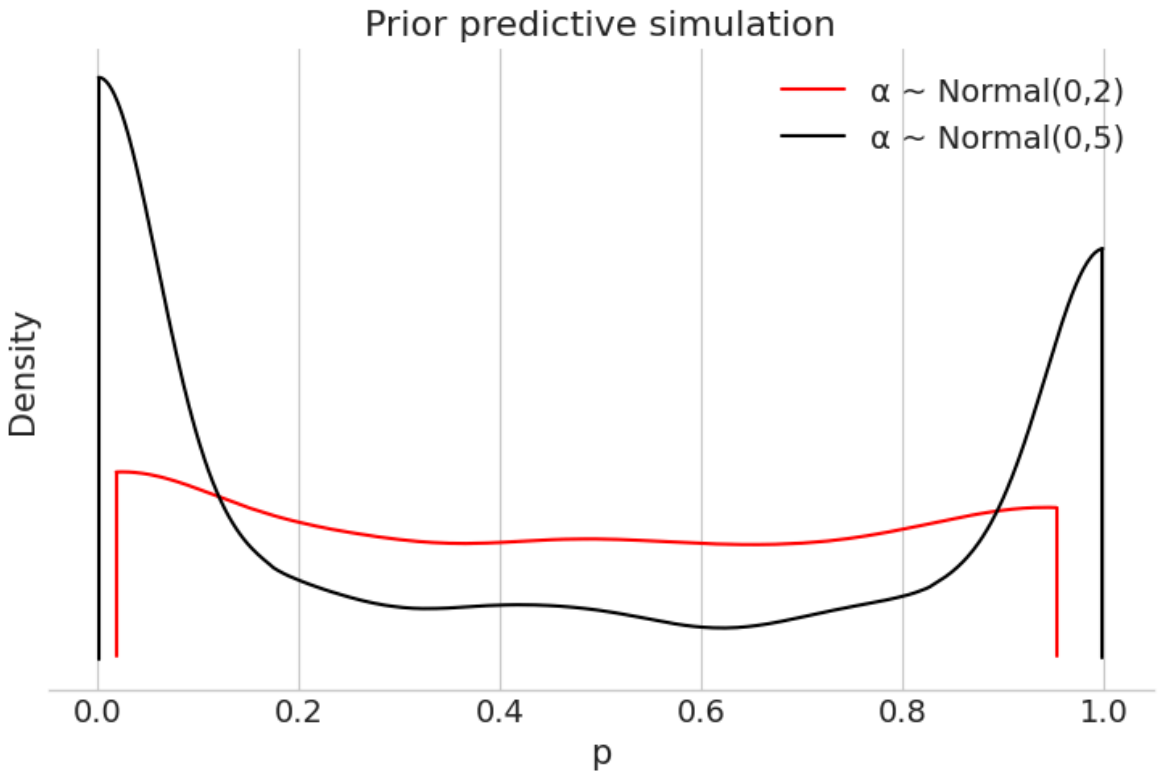
Using the inspiration from lecture 9, we show the prior assumptions from two Normal priors with for α one standard normal $N(\mu = 0, \sigma = 2)$ and one with a larger value for $\sigma = 5$

In [ ]:
```
with pm.Model() as standard_binomial_prior:

    alpha = pm.Normal("alpha", mu=0, sigma=2)
    p = pm.Deterministic("p", pm.math.invlogit(alpha))
    y_i = pm.Binomial("y", n=1, p=p, observed=df.bugs)



with pm.Model() as sigma_binomial_prior:

    alpha = pm.Normal("alpha", mu=0, sigma=5)
    p = pm.Deterministic("p", pm.math.invlogit(alpha))
    y_i = pm.Binomial("y", n=1, p=p, observed=df.bugs)

prior_standard  = pm.sample_prior_predictive(model=standard_binomial_prio
prior_sigma = pm.sample_prior_predictive(model=sigma_binomial_prior).prio
```

```
Sampling: [alpha, y]
Sampling: [alpha, y]
```

In [ ]:
```
az.plot_density(
    [prior_standard["p"], prior_sigma["p"]],
    data_labels=['α ~ Normal(0,2)','α ~ Normal(0,5)'],
```

```
    colors=['red','black'],point_estimate=None)
plt.title('Prior predictive simulation')
plt.xlabel('p')
plt.ylabel('Density');
```



We will chose the following prior for α_L

$$\alpha_L \sim N(0, 2)$$

Model Fitting

$$\begin{aligned} B_i &\sim \text{Binomial}(N_i, p_i) \\ \text{logit}(p_i) &= \alpha_L \\ \alpha_L &\sim N(0, 2) \end{aligned}$$

In [ ]:
```
with pm.Model() as h1_binomial_model:
      #Data
      language = pm.Data("language", df.language, mutable=True)

      # Priors
      alpha = pm.Normal("alpha", mu=0, sigma=2, shape=len(languages))

      #f(theta , x)
      p = pm.Deterministic("p", pm.math.invlogit(alpha[language]))

      # likelihood
      y_i = pm.Binomial("y", n=df.commits, p=p, observed=df.bugs)
      h1_binomial_trace = pm.sample(2000, tune=1000, idata_kwargs={'log_l
```

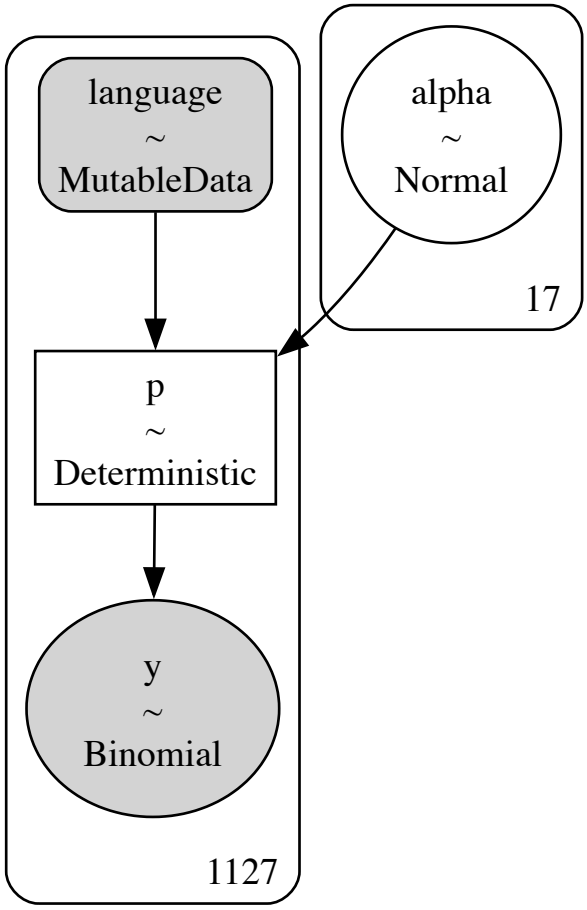
```
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [alpha]
```

100.00% [12000/12000 00:02<00:00 Sampling 4 chains, 0 divergences]

```
Sampling 4 chains for 1_000 tune and 2_000 draw iterations (4_000 + 8_000 draws total) took 3 seconds.
```

In [ ]:
```
pm.model_to_graphviz(model=h1_binomial_model)
```

Out[ ]:



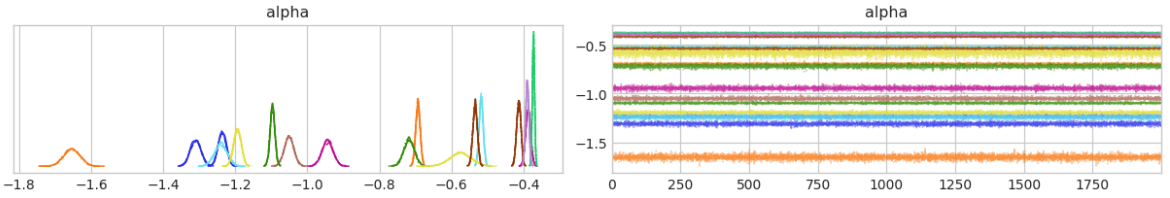
The trace summary looks valid in estimating the posterior distributions of the probability of a bug for a given datapoint, since the `r_hat` for all parameters are 1.0 . Additionally the `mcse_mean` and `mcse_sd` indicate that both samplings from the mode of the posterior could be done successfully.

In [ ]:
```
pm.summary(h1_binomial_trace, var_names=["alpha"], round_to=2)[['mcse_mea
       'ess_tail', 'r_hat']]
```

Out[ ]:

	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
alpha[0]	0.0	0.0	17180.44	5584.83	1.0
alpha[1]	0.0	0.0	14370.82	6030.80	1.0
alpha[2]	0.0	0.0	16184.49	5797.23	1.0
alpha[3]	0.0	0.0	15564.73	6169.54	1.0
alpha[4]	0.0	0.0	15120.46	6127.20	1.0
alpha[5]	0.0	0.0	16173.03	5474.52	1.0
alpha[6]	0.0	0.0	14940.95	5785.23	1.0
alpha[7]	0.0	0.0	15090.39	6156.03	1.0
alpha[8]	0.0	0.0	15443.69	6049.92	1.0
alpha[9]	0.0	0.0	14981.42	5982.45	1.0
alpha[10]	0.0	0.0	15864.97	5511.87	1.0
alpha[11]	0.0	0.0	15423.55	5908.05	1.0
alpha[12]	0.0	0.0	17343.75	6181.61	1.0
alpha[13]	0.0	0.0	14243.09	5690.45	1.0
alpha[14]	0.0	0.0	16109.47	6365.12	1.0
alpha[15]	0.0	0.0	15237.65	6381.65	1.0
alpha[16]	0.0	0.0	16020.41	5674.67	1.0

In [ ]:
```
pm.plot_trace(h1_binomial_trace, var_names=['alpha']);
```



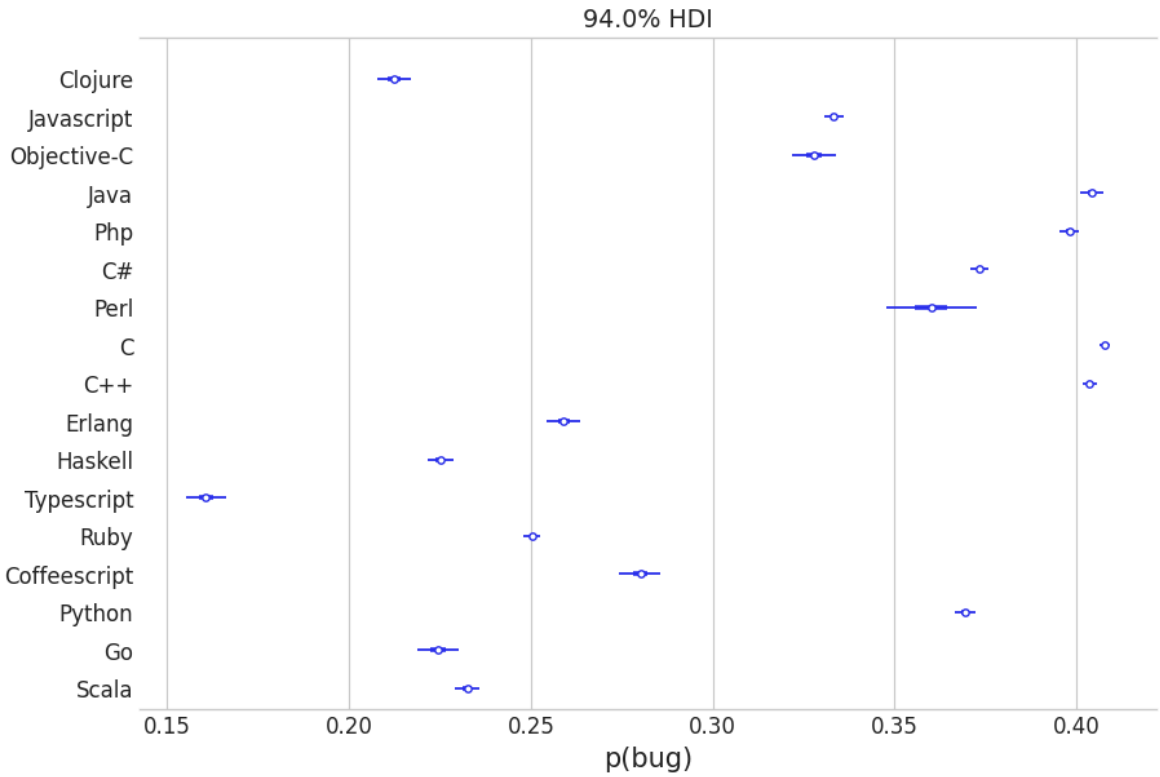
In [ ]:
```
pm.summary(h1_binomial_trace, var_names=["alpha"], round_to=2)[["mean","s
```

Out[ ]:

	mean	sd
alpha[0]	-1.31	0.01
alpha[1]	-0.69	0.01
alpha[2]	-0.72	0.01
alpha[3]	-0.39	0.01
alpha[4]	-0.41	0.01
alpha[5]	-0.52	0.01
alpha[6]	-0.58	0.03
alpha[7]	-0.37	0.00
alpha[8]	-0.39	0.00
alpha[9]	-1.05	0.01
alpha[10]	-1.24	0.01
alpha[11]	-1.65	0.02
alpha[12]	-1.10	0.01
alpha[13]	-0.94	0.01
alpha[14]	-0.53	0.01
alpha[15]	-1.24	0.02
alpha[16]	-1.19	0.01

In [ ]:

```
_, ax = plt.subplots(figsize=(9,6))
az.plot_forest(h1_binomial_trace, var_names=["alpha"], ax=ax, combined=Tr
forest_languages = [int(i.get_text()[1:-1]) if len(i.get_text()) < 6 else
ax.set_xlabel("p(bug)");
ax.set_yticklabels(languages.take(forest_languages));
```

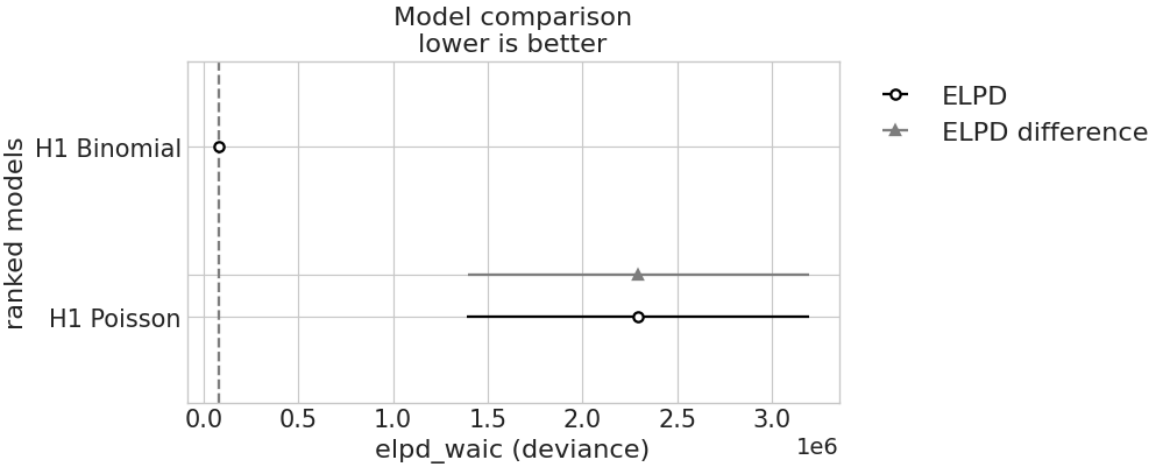


Based on the above forest plot, the `h1_binomial_model` shows similar results to `h1_poisson_model`. In particular we see that the language `C` has the highest estimated probability of a bug. Similarly, `Typescript` has the smallest estimated probability of bug, which aligns with the results of `h1_poisson_model`. Contrary to the poisson regression where ´Haskell´ was the 9th ranked language in terms of least number of bugs. It has now moved to being the language with the 4'th least probability of a bug.

We also compare the poisson and the binomial model using information criteria. We note that the binomial model has a better fit (and is not overfit) than the poisson.

In [ ]:
```
comp = pm.compare({'H1 Poisson': h1_poisson_trace,
            'H1 Binomial': h1_binomial_trace},
            ic='waic', scale='deviance')

pm.plot_compare(comp, insample_dev=False, figsize=(10,4))
```

Out[ ]:
```
<Axes: title={'center': 'Model comparison\nlower is better'}, xlabel='el
pd_waic (deviance)', ylabel='ranked models'>
```



In [ ]:
```
comp
```

Out[ ]:

	rank	elpd_waic	p_waic	elpd_diff	weight	s
H1 Binomial	0	8.151378e+04	4838.374974	0.000000e+00	0.5	12374.97354
H1 Poisson	1	2.294761e+06	118341.103230	2.213247e+06	0.5	902206.43928

Conclusion H1

Based on the results from `h1_poisson_model` and `h1_binomial_model`, we reject H1. `Haskell` does **neither** give the highest probability to the lowest number of bugs **nor** does it give the smallest probability to a bug given a commit. This means that `Haskell` is not less prone to containing bugs than any other languages based on our results. It was in fact `Typescript` that gave the highest probability to the lowest number of bugs.

H2

- **H2** - Age (A) has a positive impact on number of bugs (B) for all programming languages (L). That is, projects of old age (A) have larger number of bugs (B).

Poisson Regression

We model the number of bugs B with a Poisson distribution.

$$B_i \sim \text{Poisson}(\lambda_i)$$

This means that the log-link function is used to connect λ to the predictors. We use standardized Age A as a continious predictor. We also add an intercept α. When using standardized age, α corresponds to the $ln(\lambda)$ when age is equal to mean age.

$$log(\lambda_i) = \alpha + \beta_A A_i$$

We re-use the prior for α from H1:

$$\alpha \sim N(6, 1.5)$$

The prior for β_A is not yet defined. In the following sections we determine a reasonable prior

$$\beta_A \sim \text{Not determined yet}$$

Determine prior for β_A

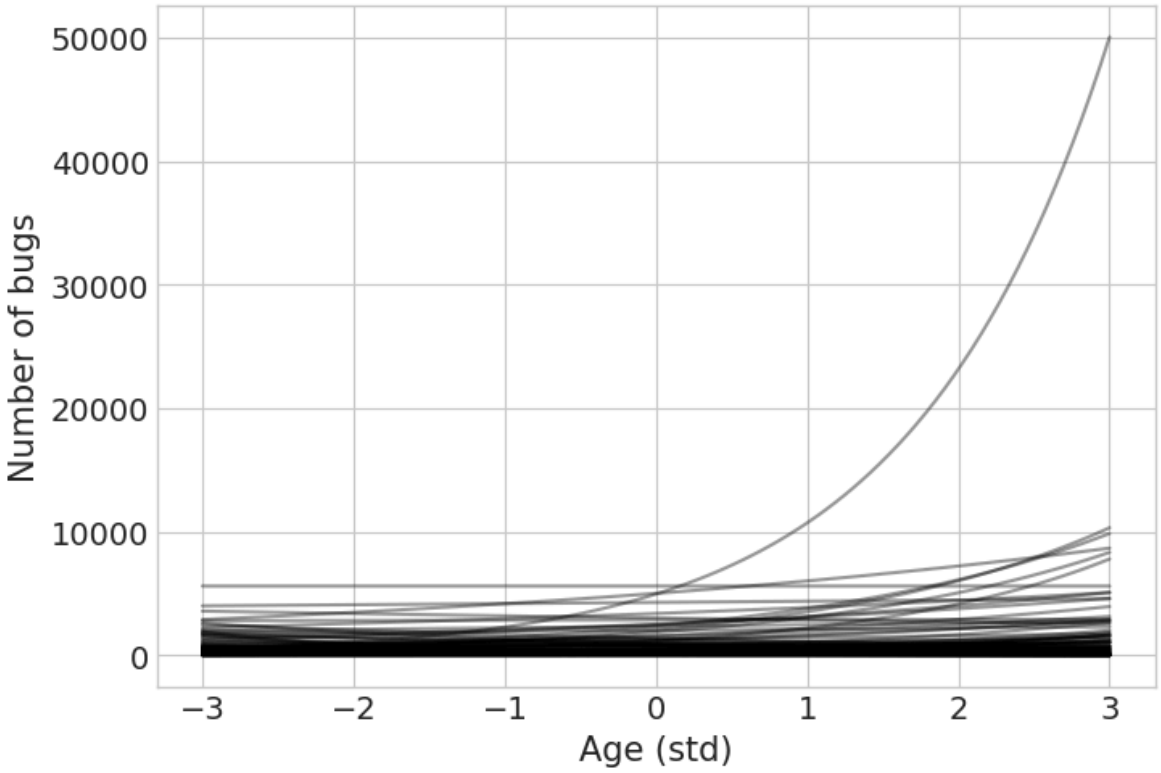
Next, we determine the prior for β_A.

We simulate the effect of varying the prior for β_A by sampling N alphas with the fixed prior that we determined earlier, and sampling N betas where we can vary the prior.

We then plot the effect of β_A by varying standardized age from -3 to 3, corresponding to 3 standard deviations below and above the mean of age. We can then see how strong (or not strong) the prior effect of β_A on number of bugs is.

In [ ]:
```
#Inspiration from https://github.com/pymc-devs/pymc-resources/blob/main/R

def plot_prior_continuous(b_mean: int = 0, b_std: int = 0, N: int = 100):
    plt.figure()

    alphas = np.random.normal(6, 1.5, N)
    betas = np.random.normal(b_mean, b_std, N)

    x_seq_std = np.linspace(-3, 3, N) # on std. scale

    for a, b in zip(alphas, betas):
        plt.plot(x_seq_std, np.exp(a + b * x_seq_std), "k", alpha=0.4)
    plt.xlabel("Age (std)")
    plt.ylabel('Number of bugs')

    return plt
```

In [ ]:
```
plot_prior_continuous(0, 0.3);
```



These effects of age seem plausible, and they allow for high values in number of bugs. They also allow for both positive and negative impact of age on the number of bugs. Many of them are rather flat, representing a conservative prior, meaning that we dont expect a particular direction of the effect of age. But we also dont expect explosive or unrealistic effects of age.

We determine the prior to be $\beta_A \sim N(0, 0.3)$

Model Fitting

We fit the following model:

$$B_i \sim \text{Poisson}(\lambda_i)$$

$$log(\lambda_i) = \alpha + \beta_A A_i$$

$$\alpha \sim N(6, 1.5)$$

$$\beta_A \sim N(0, 0.3)$$

In [ ]:
```
with pm.Model() as h2_poisson_model:
    #Priors
    alpha = pm.Normal('alpha', mu=6, sigma=1.5)
    beta_age = pm.Normal('beta_age', mu=0, sigma=0.3)

    #We add age_std here so we can do predictive plots with it later
    age_std = pm.Data("age_std", df.age_std, mutable=True)

    #Compuate lambda and sample number of bugs
    lam = pm.Deterministic('lam', pm.math.exp(alpha + beta_age * age_std))
    B = pm.Poisson('B', mu=lam, observed=df.bugs)
```

```
    #Sample posterior and prior predictive
    h2_poisson_trace = pm.sample(2000, tune=1000, idata_kwargs={'log_like
```

```
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [alpha, beta_age]
```

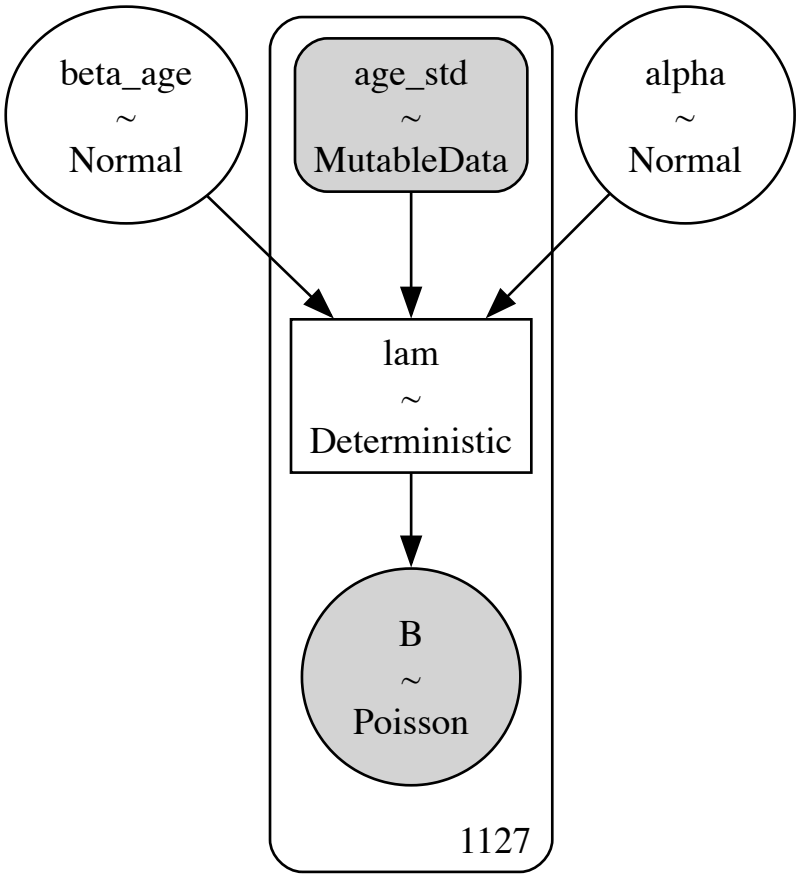
100.00% [12000/12000 00:01<00:00 Sampling 4 chains, 0 divergences]

```
Sampling 4 chains for 1_000 tune and 2_000 draw iterations (4_000 + 8_000
draws total) took 1 seconds.
```

The model has a single estimate for α and β_A. Together with standardized age, they determine 1.127 λ and B, one for each observation of a combination of project/language.

In [ ]:
```
pm.model_to_graphviz(h2_poisson_model)
```

Out[ ]:



Below is the trace convergence for the parameters `alpha` and `beta_age`.

The number of generated samples is 4 chains * 2000 samples = 8000 samples.

The effective sample sizes (`ess_bulk` , `ess_tail`) approach the number of generated samples, which indicates that the samples both in the bulk and in the tails of the posteriors have low-autocorrelation. This means most of the generated samples are sufficiently different from the previous sample, thereby adding information about the posterior, leading to high accuracy of each chain.

We also note that the Monte Carlo Standard Error `msce_mean` and `msce_sd` are 0.0, which also indicates good accuracy in the chains.

In addition, both the rhat values `r_hat` are 1.0, which means that the 4 chains have a good mixing. This is also reflected in the plot below, which shows good mixing of the chains.

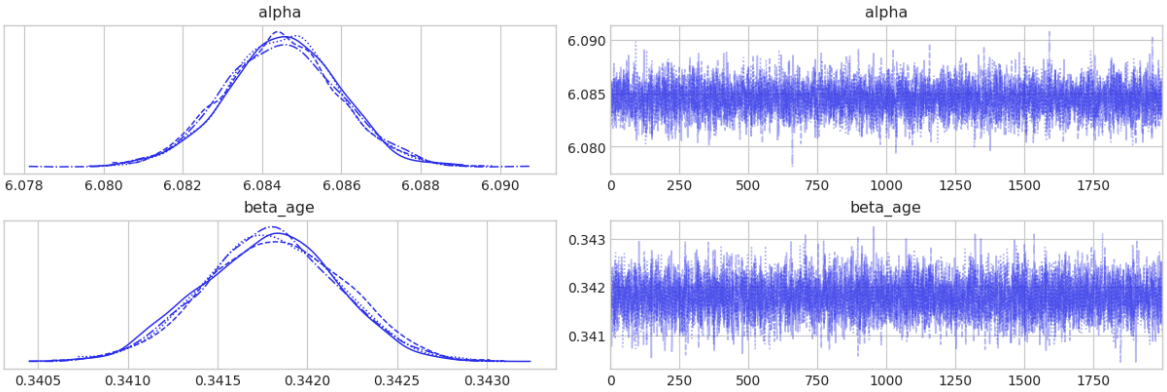
In summary, the convergence of the trace for model 2 is good and shows no signs of inefficient of inaccuracte sampling. We can therefore use the posterior distributions with high confidence.

In [ ]:
```
pm.summary(h2_poisson_trace, var_names=['alpha', 'beta_age'],round_to=2)[
       'ess_tail', 'r_hat']]
```

Out[ ]:

	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
alpha	0.0	0.0	6544.59	5848.49	1.0
beta_age	0.0	0.0	7086.38	5959.77	1.0

In [ ]:
```
az.plot_trace(h2_poisson_trace, var_names = ['alpha','beta_age'])
plt.show()
```



The forest plot shows that the standard deviations for the estimates for both `alpha` and `beta_age` are very small. We also note that the effect of `alpha` is much larger than the effect of `beta_age`.

The posterior mean of `alpha` is close to the prior we defined $N(6, 1.5)$, but it has become a lot more confident - it has a posterior sd of 0.0014.

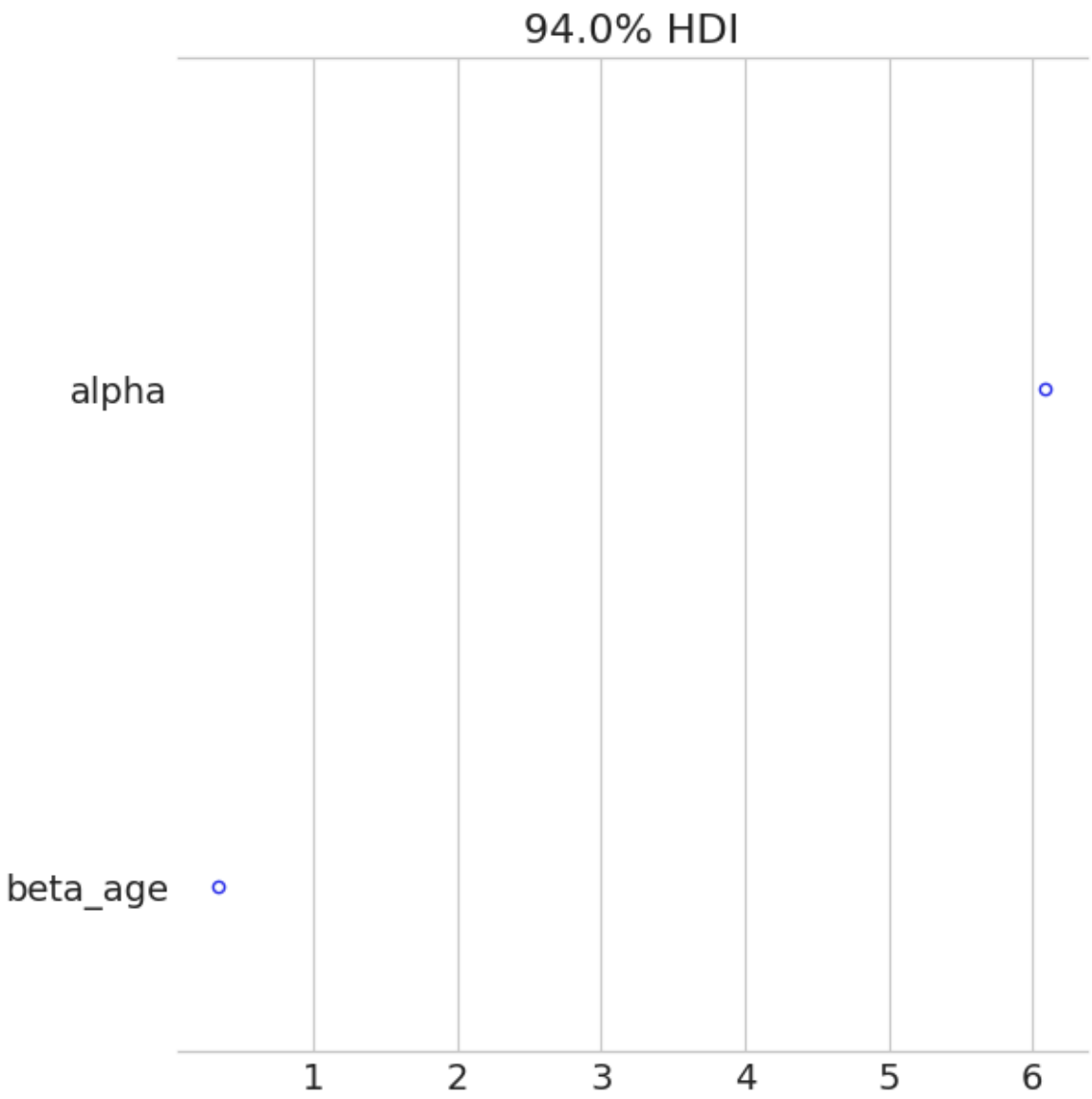
The posterior mean `beta_age` has moved in positive direction from its prior of $N(0, 0.3)$. The posterior sd is a lot more confident than the prior sd.

In [ ]:
```
pm.summary(h2_poisson_trace, var_names=['alpha', 'beta_age'],round_to=4)[
```

Out[ ]:

	mean	sd	hdi_3%	hdi_97%
alpha	6.0845	0.0014	6.0818	6.0872
beta_age	0.3418	0.0004	0.3411	0.3425

In [ ]:
```
az.plot_forest(h2_poisson_trace, var_names = ['alpha', 'beta_age'], combi
```



Posterior Predictive Check

Next we use the posterior estimates to make posterior predictive check and see how the model fits the data.

We calculate pareto k values for the data-points, to see which are influential ones. We get a warning that some samples have a pareto k value higher than 0.7, and are therefore considered highly influential.

In [ ]:
```
pareto_k_h2 = az.loo(h2_poisson_trace, pointwise=True).pareto_k.values
print("Max PSIS value for beta_age: ", max(pareto_k_h2))
```

```
Max PSIS value for beta_age:  49.82549529394249
```

We get the following warning:

"Estimated shape parameter of Pareto distribution is greater than 0.7 for one or more samples. You should consider using a more robust model, this is because importance sampling is less likely to work well if the marginal posterior and LOO posterior are very different. This is more likely to happen with a non-robust model and highly influential observations"

This warning indicates that some projects are highly influential on the model, and that taking them out at random, changes the posterior distribution.

We sample 1.127 points with equal spacing from values between -1.5 and 14 (corresponding to min/max age std.), and use these to sample the posterior predictive estimates.

In [ ]:
```
print(f"Min age std: {df.age_std.min():.2f}")
print(f"Max age std: {df.age_std.max():.2f}")
```

```
Min age std: -1.14
Max age std: 13.29
```

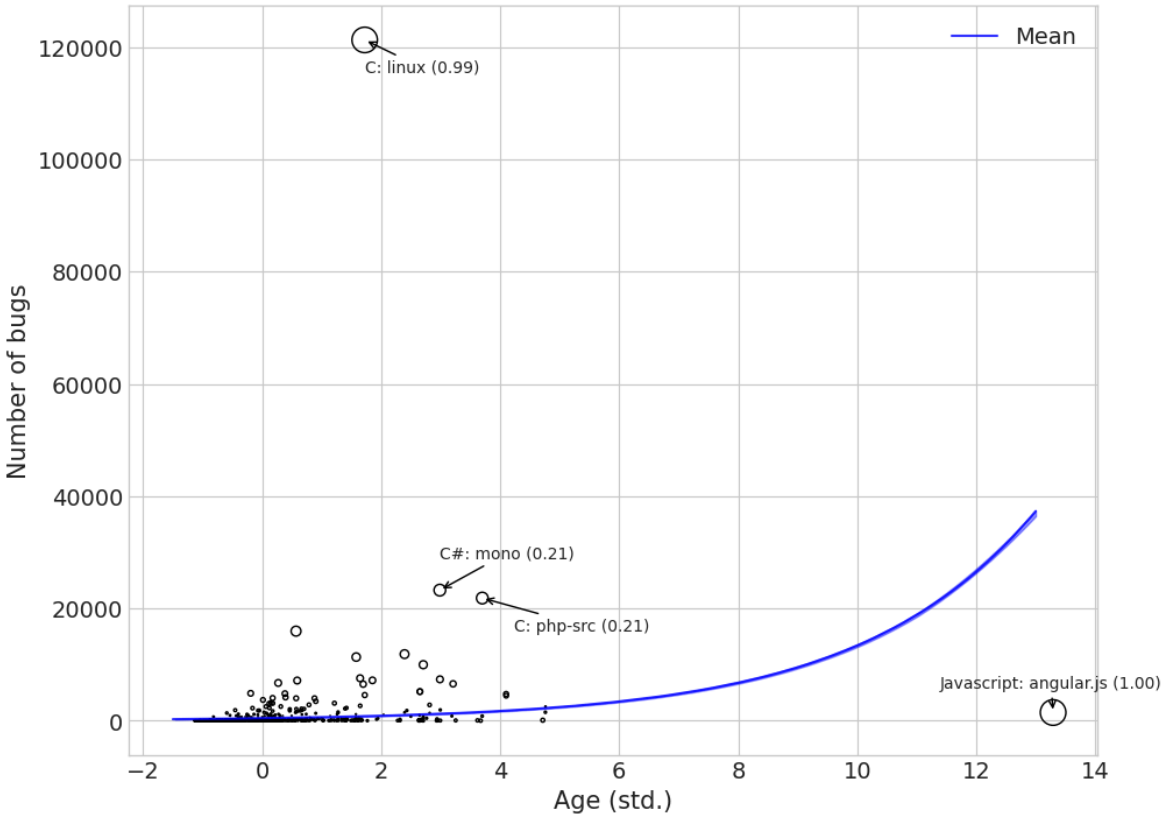
In [ ]:
```
n_points = len(df)
print(f'Number of points sampled: {n_points}')
x_seq_std = np.linspace(-1.5, 13, n_points)
x_seq = x_seq_std * df['age'].std() + df['age'].mean()  # Convert to orig


with h2_poisson_model:
    pm.set_data({"age_std": x_seq_std})
    post_pred_h2 = pm.sample_posterior_predictive(h2_poisson_trace, var_n

post_pred_h2_mean = post_pred_h2.mean(["chain", "draw"])
```

```
Sampling: [B]
Number of points sampled: 1127
```

30.81% [2465/8000 00:00<00:00]

In [ ]:
```
_, ax = plt.subplots(figsize=(10,7))

#scale pareto-k values by their max and changing make them nice to plot
pareto_k_h2 /= pareto_k_h2.max()
pareto_k_h2_size = 250 * pareto_k_h2

#Get the 10 data points with the largest pareto_k_values
top_indices = np.argsort(pareto_k_h2)[-10:][::-1]
top_data_points = df.iloc[top_indices]

az.plot_hdi(x_seq_std, post_pred_h2, color = 'b', fill_kwargs={'alpha': 0
ax.plot(x_seq_std, post_pred_h2_mean, color = 'b', alpha=0.8, label='Mean
ax.scatter(df.age_std, df.bugs, s = pareto_k_h2_size, facecolors='none',

#Plotting text for data points with high pareto-k values. We manually adj
for index, row in top_data_points.iterrows():
    dont_annotate = False
    if row['language'] == 7 and row['project'] == 'linux':
        position=(0,-20)
    elif row['language'] == 1:
        position=(-70,15)
    elif row['language'] == 5 and row['project'] == 'mono':
        position=(0,20)
    elif row['language'] == 7 and row['project'] == 'php-src':
        position=(20,-20)
    else:
        dont_annotate = True
    if dont_annotate == False:
        ax.annotate(f"{languages[row['language']]}: {row['project']} ({pa
```

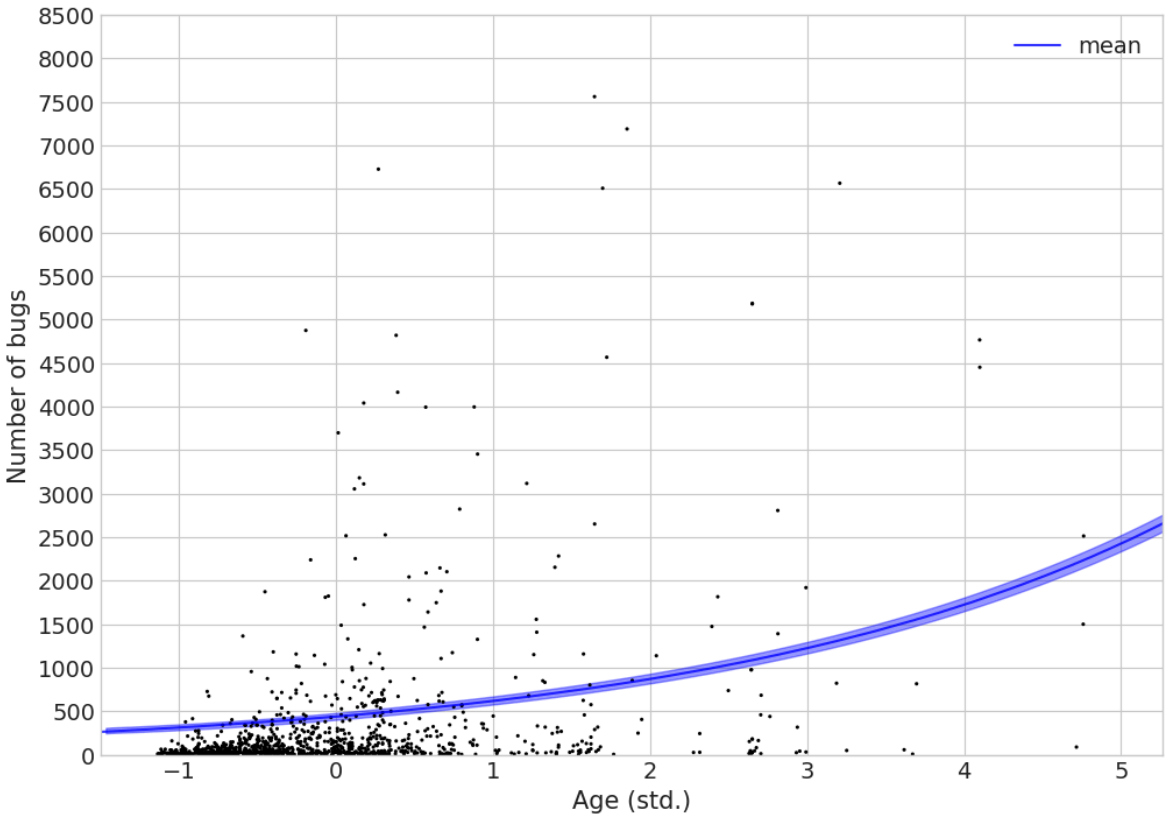
```
                    textcoords='offset points', arrowprops=dict(arrowstyl

ax.legend()
ax.set_xlabel('Age (std.)')
ax.set_ylabel('Number of bugs')
plt.show()
```



What jumps out of this figure is the existance of very influential data-points. In particular `C: Linux` which has many more bugs for its age compared to the other data-points, and `Javascript: angular.js` which is an old project/language but still has few bugs. It is important to note that we should not discard these data-points.

However, to better vizualise the posterior predictions we "zoom in" on the data-points by removing the influential data-points from the figure.

In [ ]:
```
plt.figure(figsize=(10,7))

pareto_k_size_without_top = np.delete(pareto_k_h2_size, top_indices)
df_without_top = df.drop(top_indices)

az.plot_hdi(x_seq_std, post_pred_h2, color = 'b', fill_kwargs={'alpha': 0
plt.plot(x_seq_std, post_pred_h2_mean, color = 'b', alpha=0.8, label='mea
plt.scatter(df_without_top.age_std, df_without_top.bugs, facecolors='none

plt.legend()
plt.xlim(-1.5, max(df_without_top.age_std)+0.5)
plt.ylim(0,8500)
plt.xlabel('Age (std.)')
plt.ylabel('Number of bugs')
plt.yticks(np.arange(0, 9000, 500));
```

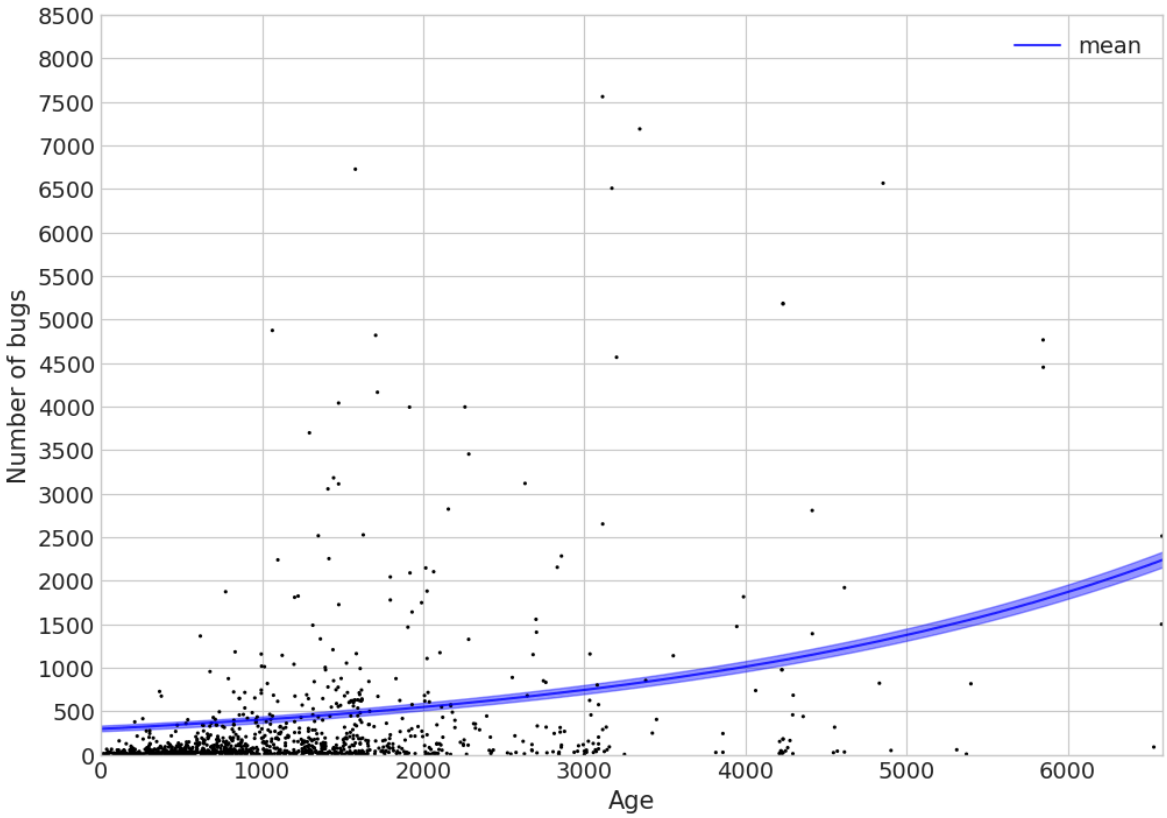


We see a positive effect of age, which was also reflected in the posterior estimate of `beta_age` which was 0.34. In addition we see that the number of bugs is around 450 when standardized age is 0. This is given by the `alpha` = 6.08 which corresponds to $exp(6.08) = 437$.

The plot also shows small uncertainty of the mean, although the uncertainty grows slighly as standardized age increases.

Next we plot the effect of non-standardized age, also excluding the most influential data-points. We assume that the unit for age is days.

In [ ]:
```
plt.figure(figsize=(10,7))

pareto_k_size_without_top = np.delete(pareto_k_h2_size, top_indices)
df_without_top = df.drop(top_indices)

az.plot_hdi(x_seq, post_pred_h2, color = 'b', fill_kwargs={'alpha': 0.4},
plt.plot(x_seq, post_pred_h2_mean, color = 'b', alpha=0.8, label='mean')
plt.scatter(df_without_top.age, df_without_top.bugs, facecolors='none', s

plt.legend()
plt.xlim(-1.5, max(df_without_top.age)+0.5)
plt.ylim(0,8500)
plt.xlabel('Age')
plt.ylabel('Number of bugs')
plt.yticks(np.arange(0, 9000, 500));
```



We note that the intercept of the model seems unrealistic. The model expects that a project/language of age 0 will have 297 bugs. This is a pitfall of using an intercept (α) in a Poisson model as a free parameter which does not reflect the natural zero of an unstarted project where age and bugs are both 0.

In [ ]:
```
y_at_0 = np.interp(0, x_seq, post_pred_h2_mean)
y_at_365 = np.interp(365, x_seq, post_pred_h2_mean)
y_at_1825 = np.interp(1825, x_seq, post_pred_h2_mean)
print(f"Expected number of bugs at 0 days: {y_at_0:.2f}")
print(f"Expected number of bugs at 365 days: {y_at_365:.2f}")
print(f"Expected number of bugs at 1825 days: {y_at_1825:.2f}")
```

```
Expected number of bugs at 0 days: 296.49
Expected number of bugs at 365 days: 331.80
Expected number of bugs at 1825 days: 519.06
```

Conclusion H2

Based on the above results, we find that age has a positive impact on number of bugs, thereby confirming hypothesis 2. A project/language with older age have a larger number of bugs. An increase in age from 365 days (1 year) to 1825 days (5 years) increases the expected number of bugs from 332 to 519. We consider this a rather modest effect.

However, we suspect that the association of age with bugs could be a result of omitted variable bias. In hypothesis 3 we examine whether stratifying by commits will lower the strength of the association with age on bugs.

It is also important to note that this model, which only includes a single effect of age on bugs does not model the effect of age for each individual language. The model

assumes that all languages can be modelled together and that there is an overall effect of age on bugs.

H3

- **H3** - Number of commits (C) does not impact the effect of age (A) on the number of bugs (B) for any programming language (L). That is, the effect of age (A), conditioned on number of commits (C), on number of bugs (B) is the same as the direct effect of age (A) on number of bugs (B).

Poisson Regression

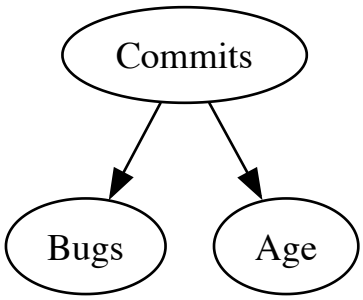
In hypothesis 3 we examine whether the effect of age (A), conditioned on number of commits (C), on number of bugs (B) is the same as the direct effect of age (A) on number of bugs (B).

We treat commits as a confounding variable that is affecting age and bugs through a "fork-effect".

In hypothesis 2 we saw that age (X) was associated with bugs (Y). In hypothesis 3 we strafiy by commits (Z) and examine whether the effect of age changes compared to hypothesis 2. To stratify by commits we simply add it as a predictor in the linear combination.

In [ ]:
```
CausalGraphicalModel(
    nodes=["Commits", "Bugs", "Age"],
    edges=[
        ("Commits", "Bugs"),
        ("Commits", "Age"),
    ]
).draw()
```

Out[ ]:



As before we model the number of bugs B with a Poisson distribution.

$$B_i \sim \text{Poisson}(\lambda_i)$$

This means that the log-link function is used to connect λ to the predictors. We use standardized Age A and standardized commits C as a continuous predictors. We also add an intercept α.

$$log(\lambda_i) = \alpha + \beta_A A_i + \beta_C C_i$$

We re-use the priors for α and β_A that we determined in H2.

$$\alpha \sim N(6, 1.5)$$

$$\beta_A \sim N(0, 0.3)$$

We determine the prior for β_C in the following section.

$$\beta_C \sim \text{Not yet determined}$$

Determine prior for β_C

We use standardized commits, which makes the coefficients easier to interpret, while also allowing us to remove the effect of commits in the posterior predictive check.

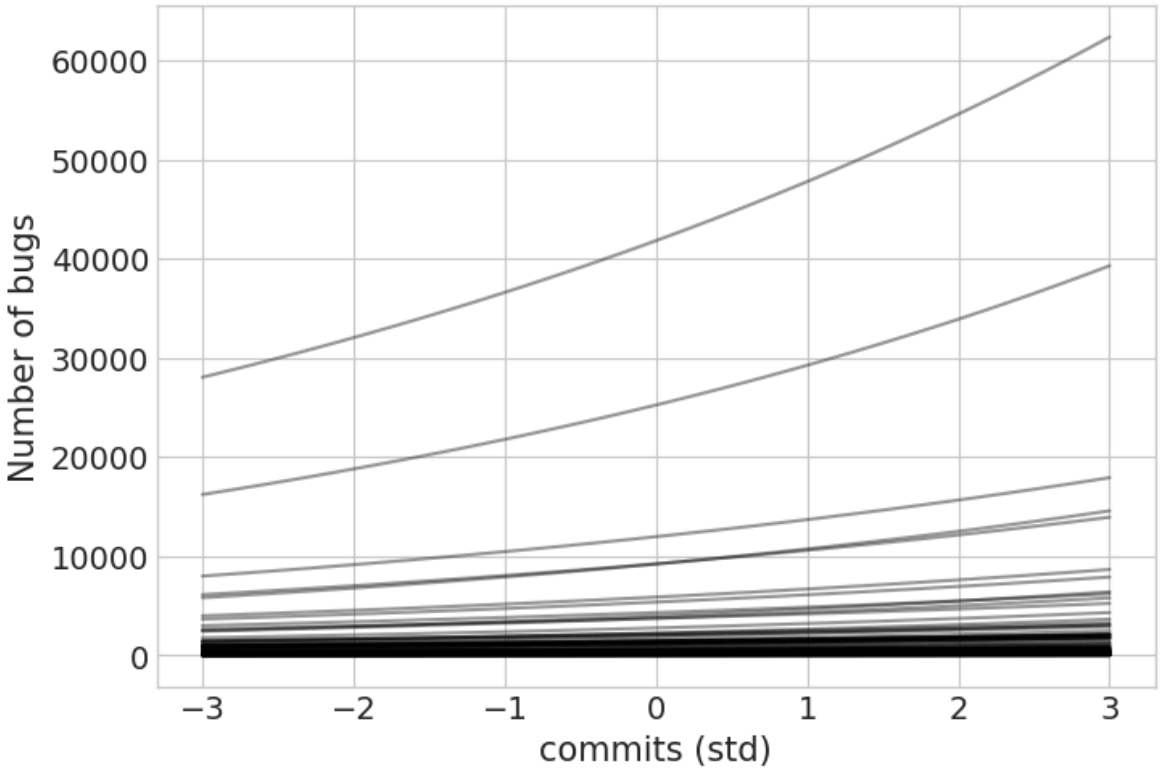
In the prior for β_A we allowed for both positive and negative impact of age on the number of bugs, with many priors being quite flat.

We dont want the same behavior in the prior for the effect of number of commits on number of bugs: β_C. This is because we know that bugs is defined as the number of commits classified as bugs. This means that commits can *not* have a negative impact on the number of bugs. This is simply impossible. So we want a prior for β_C that only allows positive relationships.

We choose to use a log-normal distribution for β_C which only has positive values.

In [ ]:
```
def plot_prior_continuous_lognormal(b_mean: float = 0, b_std: float = 0,
    plt.figure()

    alphas = np.random.normal(6, 1.5, N)
    betas = np.random.lognormal(mean=b_mean, sigma=b_std, size=N)
    x_seq_std = np.linspace(-3, 3, N) # on std. scale

    for a, b in zip(alphas, betas):
        plt.plot(x_seq_std, np.exp(a + b * x_seq_std), "k", alpha=0.4)
    plt.xlabel("commits (std)")
    plt.ylabel('Number of bugs')

    return plt
```

In [ ]:
```
plot_prior_continuous_lognormal(b_mean=-2, b_std=0.1);
```



These effects of commits seem plausible. They are all positive, and some of them allow for quite strong relationships.

We determine the prior to be $\beta_C \sim \text{LogNormal}(-2, 0.1)$.

Model Fitting

We fit the model using the determined priors:

$$B_i \sim \text{Poisson}(\lambda_i)$$

$$log(\lambda_i) = \alpha + \beta_A A_i$$

$$\alpha \sim N(6, 1.5)$$

$$\beta_A \sim N(0, 0.3)$$

$$\beta_C \sim \text{LogNormal}(-2, 0.1)$$

In [ ]:
```
with pm.Model() as h3_poisson_model:
    #Priors
    alpha = pm.Normal('alpha', mu=6, sigma=1.5)
    beta_age = pm.Normal('beta_age', mu=0, sigma=0.3)
    beta_commits = pm.Lognormal('beta_commits', mu=-2, sigma=0.1)

    #We add age_std and com_std here so we can do predictive plots with i
    age_std = pm.Data("age_std", df.age_std, mutable=True)
    commits_std = pm.Data("commits_std", df.commits_std, mutable=True)

    #Compuate lambda and sample number of bugs
    lam = pm.Deterministic('lam', pm.math.exp(alpha + beta_commits*commit
    B = pm.Poisson('B', mu=lam, observed=df.bugs)
```

```
    #Sample posterior and prior predictive
    h3_poisson_trace = pm.sample(2000, tune=1000, idata_kwargs={'log_like
```

```
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [alpha, beta_age, beta_commits]
```

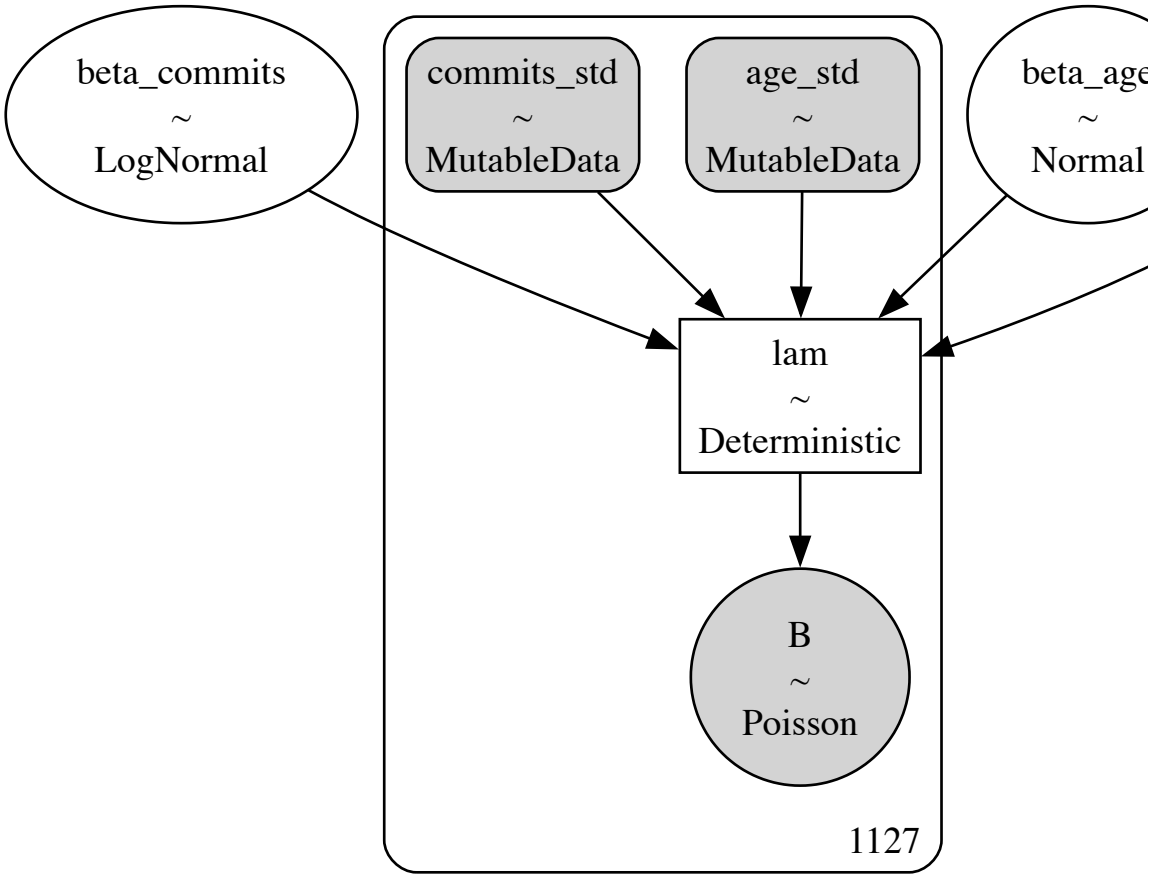
100.00% [12000/12000 00:01<00:00 Sampling 4 chains, 0 divergences]

```
Sampling 4 chains for 1_000 tune and 2_000 draw iterations (4_000 + 8_000
draws total) took 2 seconds.
```

The model has a single estimate for α, β_A and β_C . Together with standardized age and commits, they determine 1.127 λ and B, one for each observation of a combination of project/language.

In [ ]:
```
pm.model_to_graphviz(h3_poisson_model)
```

Out[ ]:



Below is the trace convergence for the parameters `alpha` , `beta_age` , and `beta_commits`

The effective sample sizes (`ess_bulk` , `ess_tail`) approach the number of generated samples, which indicates that the samples both in the bulk and in the tails of the posteriors have low-autocorrelation.

We also note that the Monte Carlo Standard Error `msce_mean` and `msce_sd` are 0.0, which also indicates good accuracy in the chains.

In addition, the rhat values `r_hat` are 1.0, which means that the 4 chains have converged well. This is also reflected in the plot below, which shows good mixing of the chains.

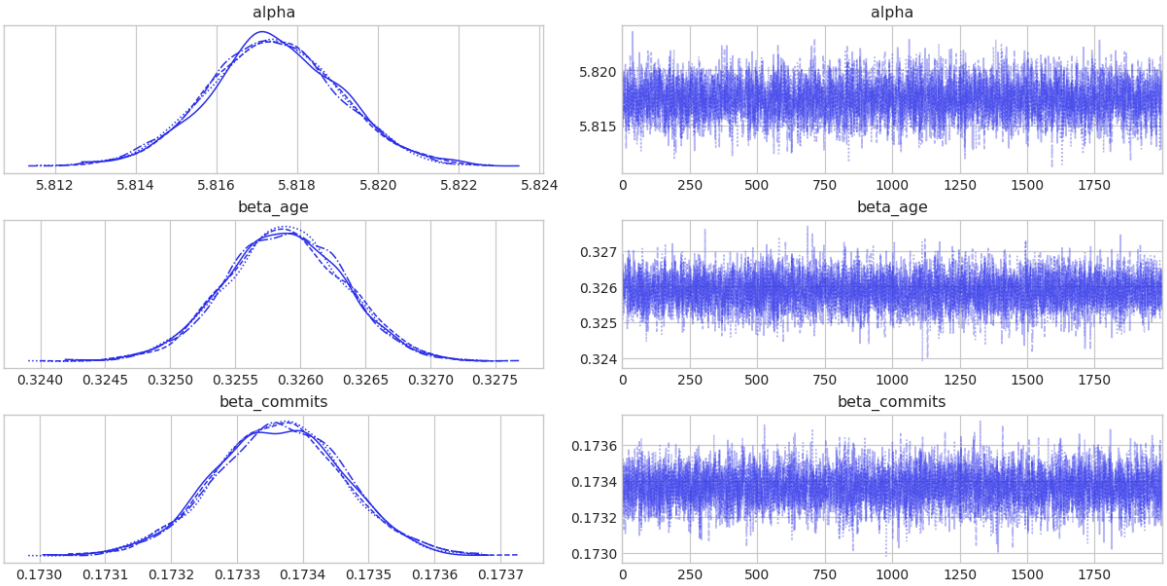
In summary, the convergence of the trace for model 3 is good and shows no signs of inefficient of inaccuracte sampling. We can therefore use the posterior distributions with high confidence.

In [ ]:
```
pm.summary(h3_poisson_trace, var_names=['alpha', 'beta_age', 'beta_commit
       'ess_tail', 'r_hat']]
```

Out[ ]:

	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
alpha	0.0	0.0	5014.76	5879.54	1.0
beta_age	0.0	0.0	6341.30	5816.96	1.0
beta_commits	0.0	0.0	6030.65	5506.42	1.0

In [ ]:
```
az.plot_trace(h3_poisson_trace, var_names = ['alpha','beta_age', 'beta_co
plt.show()
```



The forest plot below shows that the standard deviations for the estimates for `alpha`, `beta_age`, and `beta_commits` are very small. We also note that `alpha` is much larger than `beta_age` and `beta_commits`.

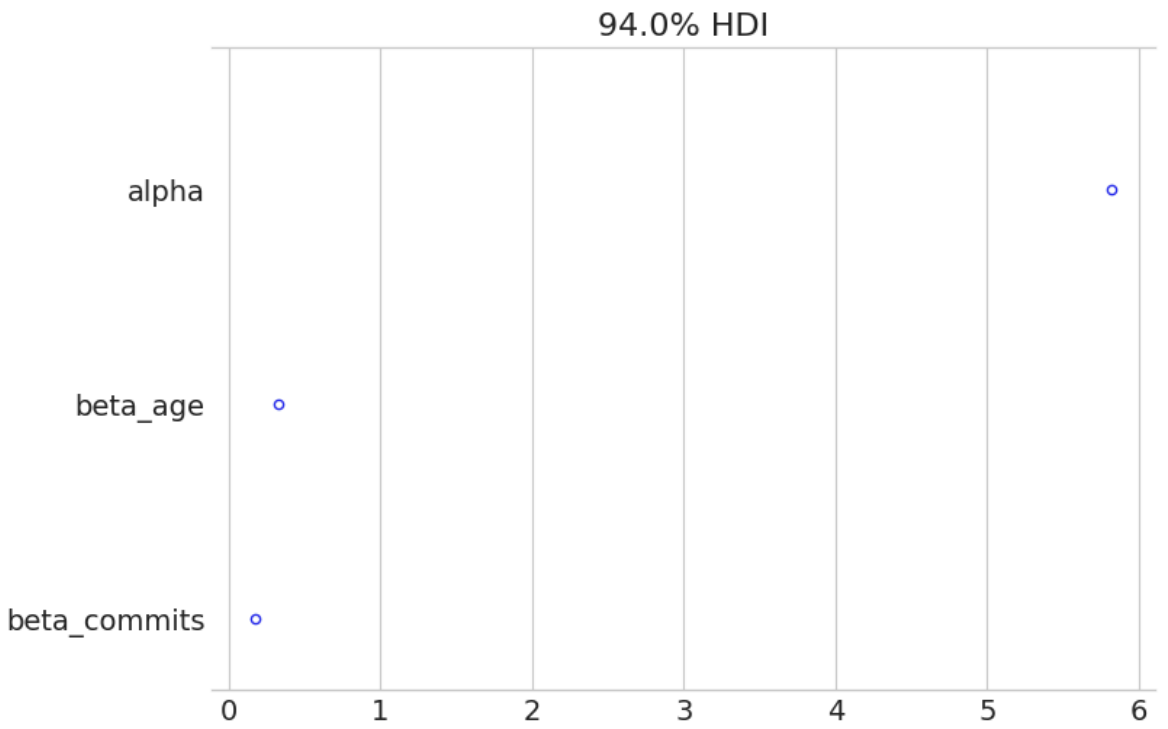
Contrary to what we expected, `beta_age` actually has a larger effect than `beta_commits`. We expected `beta_commits` to have a larger effect because it is closely associated with number of bugs (bugs is defined as the number of commits classified as a bug).

In [ ]:
```
pm.summary(h3_poisson_trace, var_names=['alpha', 'beta_age', 'beta_commit
```

Out[ ]:

	mean	sd	hdi_3%	hdi_97%
alpha	5.8174	0.0016	5.8142	5.8203
beta_age	0.3259	0.0005	0.3251	0.3268
beta_commits	0.1734	0.0001	0.1732	0.1735

In [ ]:
```
az.plot_forest(h3_poisson_trace, var_names = ['alpha','beta_age', 'beta_c
```



Next we compare the effect of age on number of bugs without conditioning on the number of commits (H2) and with conditioning on the number of commits (H3).

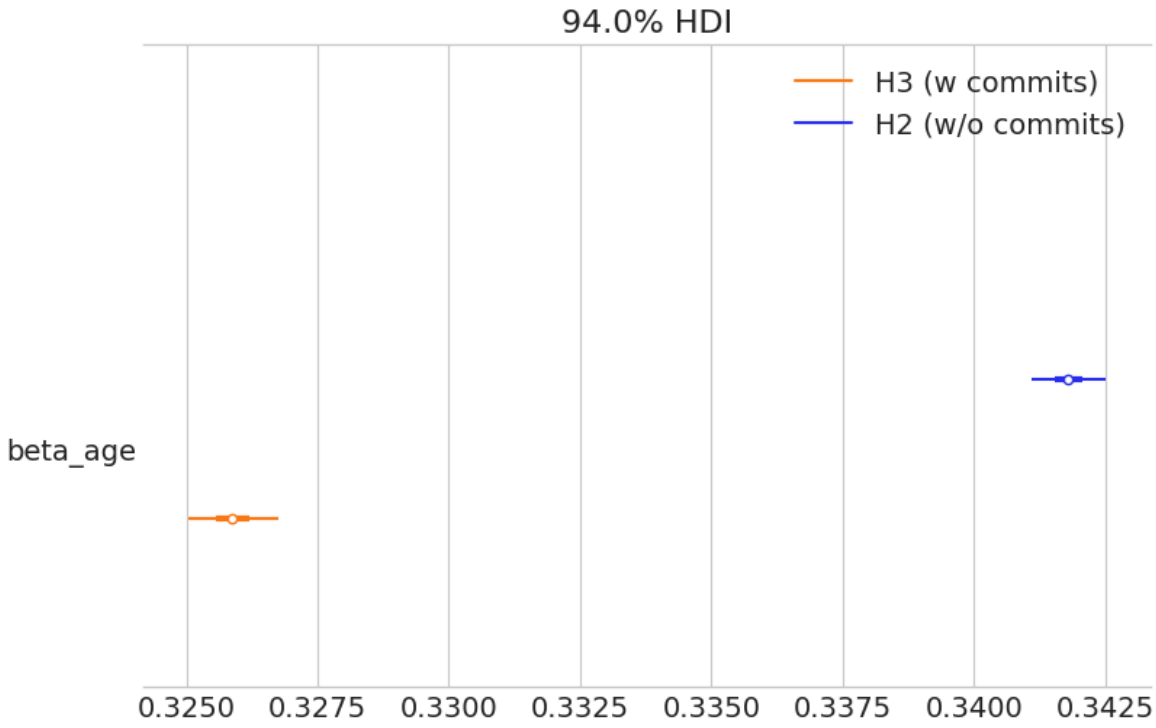
We see a notable difference between the effect of age in the two models in the forest plot. The effect of age on the number of bugs conditioned by the number of commits is confidently lower than the direct effect of age.

Nonetheless, the difference seems quite small when looking at the estimates. It will be easier to compare them in the next section of posterior predictions.

In [ ]:
```
print(f"Decrease in posterior mean of beta_age from H2 to H3: {round(pm.s
```

```
Decrease in posterior mean of beta_age from H2 to H3: -0.0159
```

In [ ]:
```
az.plot_forest(
    data=[h2_poisson_trace, h3_poisson_trace],
    model_names = ['H2 (w/o commits)','H3 (w commits)'],
    var_names=['beta_age'],
    combined=True,
    figsize=(8,5));
```



Posterior Predictive Checks for Age

We calculate pareto k values for the data-points, to see influential data-points. We get the same warning as in H2.

In [ ]:
```
pareto_k_h3 = az.loo(h3_poisson_trace, pointwise=True).pareto_k.values
```

For standardized age, we sample 1.127 points linearly from values between -1.5 and 14, exactly like in H2.

For standardized commits, we sample 1.127 points of 0. This is simply to remove the effect of `beta_commits`.

This means that we can compare the posterior predictions of the model in H3 with the model in H2 when varying standardized age.

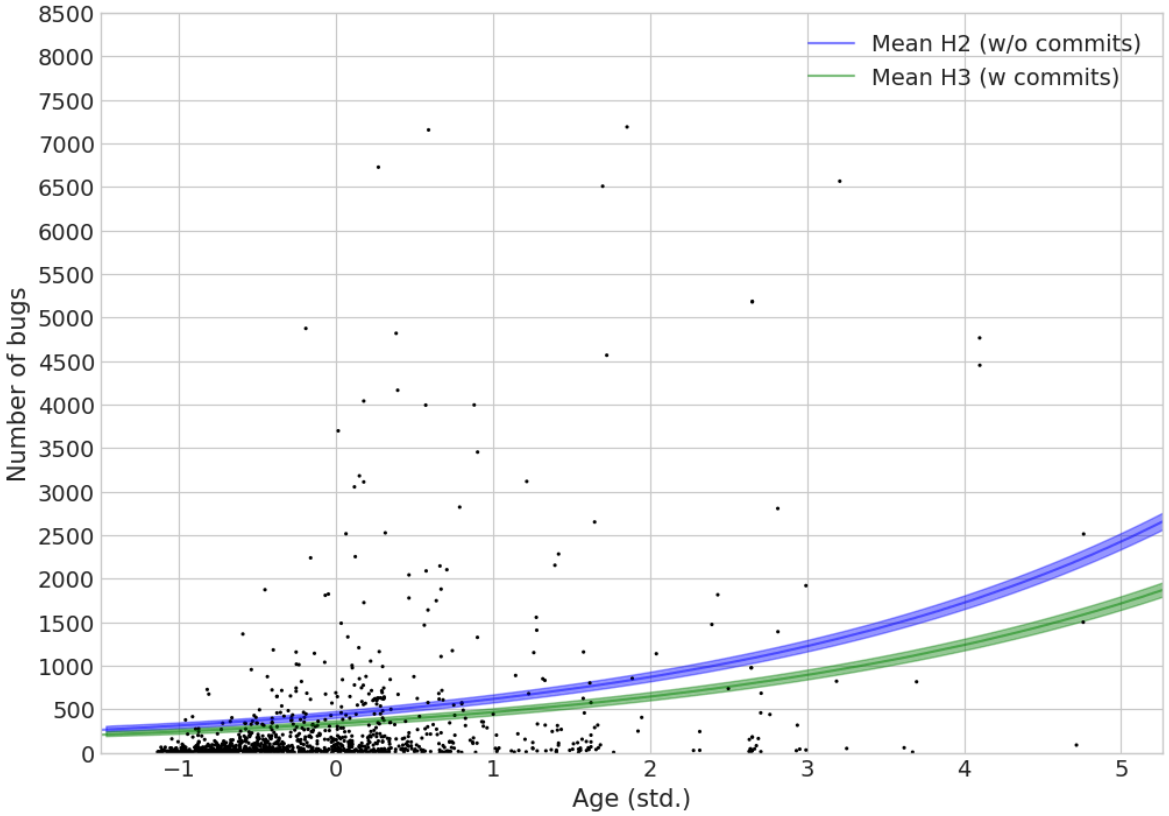
In [ ]:
```
n_points = len(df)
print(f'Number of points sampled: {n_points}')
x_seq = np.linspace(-1.5, 13, n_points)

with h3_poisson_model:
    pm.set_data({"age_std": x_seq, "commits_std": np.repeat(0, n_points)}
    post_pred_h3 = pm.sample_posterior_predictive(h3_poisson_trace, var_n

post_pred_h3_mean = post_pred_h3.mean(["chain", "draw"])
```

```
Sampling: [B]
Number of points sampled: 1127
```

22.11% [1769/8000 00:00<00:00]

We remove the influential data-points from the figure so it is easier to compare the two models.

In [ ]:
```
plt.figure(figsize=(10,7))

pareto_k_h3 /= pareto_k_h3.max()
pareto_k_h3_size = 250 * pareto_k_h3

#Get the 10 data points with the largest pareto_k_values
top_indices = np.argsort(pareto_k_h3)[-10:][::-1]

pareto_k_size_without_top = np.delete(pareto_k_h3_size, top_indices)
df_without_top = df.drop(top_indices)

az.plot_hdi(x_seq, post_pred_h2, color = 'b', fill_kwargs={'alpha': 0.4},
plt.plot(x_seq, post_pred_h2_mean, color = 'b', alpha=0.5, label='Mean H2

az.plot_hdi(x_seq, post_pred_h3, color = 'g', fill_kwargs={'alpha': 0.4},
plt.plot(x_seq, post_pred_h3_mean, color = 'g', alpha=0.5, label='Mean H3

plt.scatter(df_without_top.age_std, df_without_top.bugs, facecolors='none

plt.legend()
plt.xlim(-1.5, max(df_without_top.age_std)+0.5)
plt.ylim(0,8500)
plt.xlabel('Age (std.)')
plt.ylabel('Number of bugs')
plt.yticks(np.arange(0, 9000, 500));
```



Part Conclusion H3

The blue line corresponds to the effect of age in `h2_poisson_model` and is the direct effect of age on number of bugs. The green line is the effect of age conditioned on the number of commits. We note that the effect of age conditioned on commits is *smaller* than the direct effect of age. This gives evidence to discard

hypothesis 3. Nonetheless, there is still a substantial effect of age after conditioning on commits. We expected to see a smaller effect.

The alpha for model 3 is slightly lower than the alpha for model 2. Subtracting the exponential of their posterior means gives a difference of $exp(6.08) - exp(5.82) = 100$.

Posterior Predictive Checks for Commits

Next, we make posterior predictions for commits while removing the effect of age.

For standardized commits, we sample 1.127 points linearly from from values between -0.15 and 32

For standardized age, we sample 1.127 points of 0. This is simply to remove the effect of `beta_age`.

In [ ]:
```
print(f"Min commits std: {df.commits_std.min():.2f}")
print(f"Max commits std: {df.commits_std.max():.2f}")
```

```
Min commits std: -0.14
Max commits std: 31.42
```

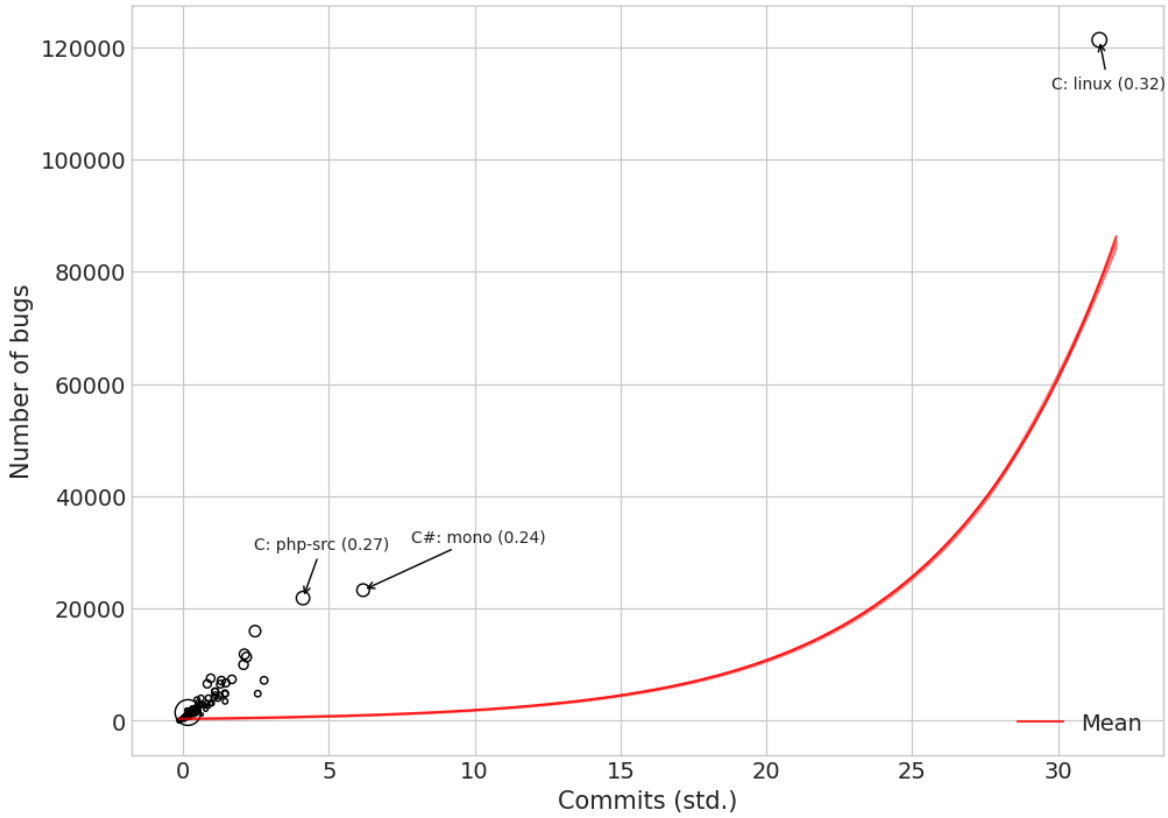
In [ ]:
```
n_points = len(df)
print(f'Number of points sampled: {n_points}')
x_seq_std = np.linspace(-0.15, 32, n_points)
x_seq = x_seq_std * df['commits'].std() + df['commits'].mean()  # Convert

with h3_poisson_model:
    pm.set_data({"commits_std": x_seq_std, "age_std": np.repeat(0, n_poin
    post_pred_h3_commits = pm.sample_posterior_predictive(h3_poisson_trac

post_pred_h3_commits_mean = post_pred_h3_commits.mean(["chain", "draw"])
```

```
Sampling: [B]
Number of points sampled: 1127
```

```
3.27% [262/8000 00:00<00:00]
```

In [ ]:
```
plt.figure(figsize=(10,7))

#scale pareto-k values by their max and changing make them nice to plot
pareto_k_h3 /= pareto_k_h3.max()
pareto_k_h3_size = 250 * pareto_k_h3
pareto_k_h3_size += abs(min(pareto_k_h3_size))+0.001 #fix error with nega

#Get the 10 data points with the largest pareto_k_values
top_indices = np.argsort(pareto_k_h3)[-10:][::-1]
top_data_points = df.iloc[top_indices]


az.plot_hdi(x_seq_std, post_pred_h3_commits, color = 'r', fill_kwargs={'a
plt.plot(x_seq_std, post_pred_h3_commits_mean, color = 'r', alpha=0.8, la
plt.scatter(df.commits_std, df.bugs, s = pareto_k_h3_size, facecolors='no

#Plotting text for data points with high pareto-k values. We manually adj
for index, row in top_data_points.iterrows():
```

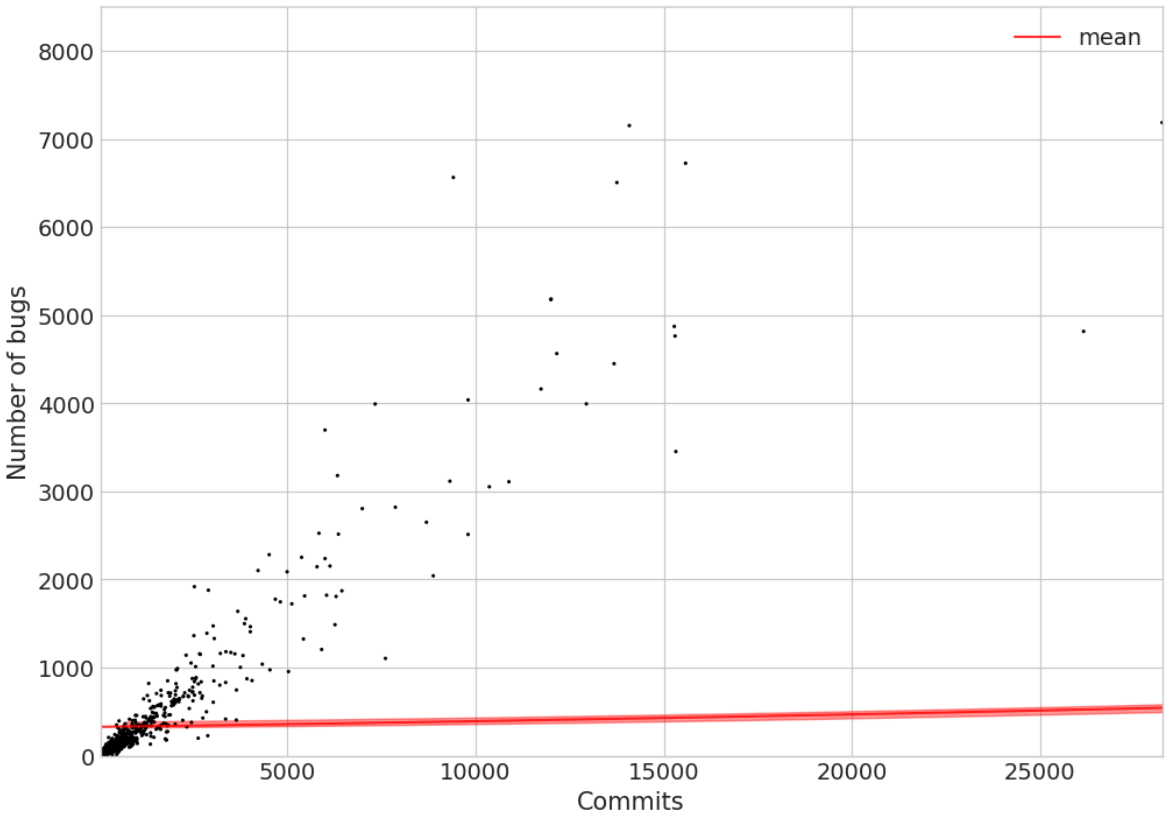
```
    dont_annotate = False
    if row['language'] == 7 and row['project'] == 'linux':
        position=(-30,-30)
    elif row['language'] == 5 and row['project'] == 'mono':
        position=(30,30)
    elif row['language'] == 7 and row['project'] == 'php-src':
        position=(-30,30)
    else:
        dont_annotate = True
    if dont_annotate == False:
         plt.annotate(f"{languages[row['language']]}: {row['project']} ({
                     textcoords='offset points', arrowprops=dict(arrowsty

plt.legend(loc = 'lower right');
plt.xlabel('Commits (std.)');
plt.ylabel('Number of bugs');
```



Like in H2, we see very influential data-points. To better vizualise posterior predictions of commits we "zoom-in" on data-points by removing the influential data-points from the figure. We also use non-standardized commits for the x-axis.

In [ ]:
```
plt.figure(figsize=(10,7))

pareto_k_size_without_top = np.delete(pareto_k_h3_size, top_indices)
df_without_top = df.drop(top_indices)

az.plot_hdi(x_seq, post_pred_h3_commits, color = 'r', fill_kwargs={'alpha
plt.plot(x_seq, post_pred_h3_commits_mean, color = 'r', alpha=0.8, label=
plt.scatter(df_without_top.commits, df_without_top.bugs, facecolors='none

plt.legend()
plt.xlim(min(df_without_top.commits), max(df_without_top.commits)+0.5)
plt.ylim(0,8500)
plt.xlabel('Commits')
```

```
plt.ylabel('Number of bugs')
plt.yticks(np.arange(0, 9000, 1000));
```



The effect of commits on number of bugs is clearly not fitted well. We would have expected to see a much larger effect of commits, which would have aligned the mean prediction to the data-points. We dont know why we get such a poor fit. We tried to use different priors for β_C but that did not help.

In [ ]:
```
y_at_0 = np.interp(0, x_seq, post_pred_h3_commits_mean)
y_at_1000 = np.interp(1000, x_seq, post_pred_h3_commits_mean)
y_at_10000 = np.interp(10000, x_seq, post_pred_h3_commits_mean)
print(f"Expected number of bugs at 0 commits: {y_at_0:.2f}")
print(f"Expected number of bugs at 1000 commits: {y_at_1000:.2f}")
print(f"Expected number of bugs at 10000 commits: {y_at_10000:.2f}")
```

```
Expected number of bugs at 0 commits: 327.65
Expected number of bugs at 1000 commits: 334.04
Expected number of bugs at 10000 commits: 392.54
```

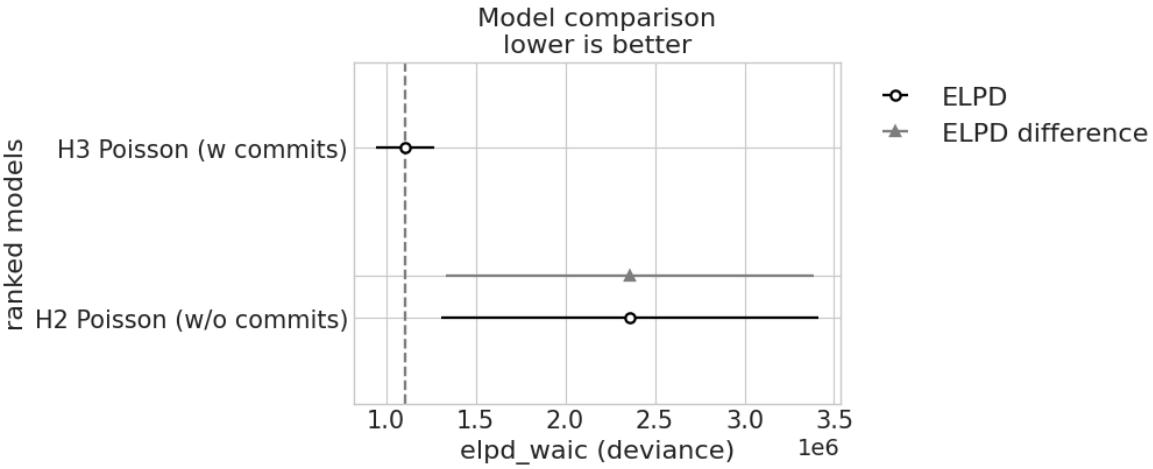
We see again that the increase in number of bugs in relation to commits is simply too low according to the model . An increase in commits from 0 to 10000 only increases the expected number of bugs from 328 to 392. This is unrealistic and doesnt match with the data.

In regard to hypothesis 3, since we expect the true effect of commits to be larger than the one we estimated, we also suspect that the effect of age conditioned on commits to be smaller than what we estimated.

This is why we try a different approach in the next section where we model bugs as a real value, i.e. model it with a normal distribution.

Finally, we use information criteria to compare the models for H2 and H3. We note that the waic is much lower in the model for H3 than the model for H2. This means that adding commits as a predictor increases the fit to the data without overfitting.

In [ ]:
```
comp = pm.compare({'H2 Poisson (w/o commits)': h2_poisson_trace,
                   'H3 Poisson (w commits)': h3_poisson_trace},
                   ic='waic', scale='deviance')

pm.plot_compare(comp, insample_dev=False, figsize=(10,4))
plt.show()
```



Additional Analysis

In this section we analyse H2 and H3 using different approaches.

H2

- **H2** - Age (A) has a positive impact on number of bugs (B) for all programming languages (L). That is, projects of old age (A) have larger number of bugs (B).

Normal Regression

This time we attempt to model Hypothesis 2 by standardizing number of bugs (B) and model it using a normal regression. We believe that this implementation makes bugs compatible with the scale of the predictors (age (A) and commits (C)) while also aiding prior predictive checks and the overall interpretation of the posterior parameter distributions.

Contrary to the previous `h2_poisson_model` for H2, we here include an effect of age on bugs for each programming language. This way we can see whether the effect of age on bugs differs among programming languages.

The model is defined as follows:

$$B_i \sim \text{Normal}(\mu_i, \sigma)$$

$$\mu_i = \alpha + \beta_{\text{A}[L]} A_i$$

$$\sigma \sim \text{Not yet determined}$$

$$\alpha \sim \text{Not yet determined}$$

$$\beta_{\text{A}[L]} \sim \text{Not yet determined}$$

where

$$L \in \mathcal{L}, where\ \mathcal{L} = \{Python, Java.., C\}$$
$$|\mathcal{L}| = 17$$

As for the prior for σ, α, and $\beta_{\text{A}[L]}$, we'll set plausible values in the next section.

Determine priors

- Determine priors for α and σ

We choose the prior of α to follow a normal distribution. Assuming that a standardized age value takes 0, we expect the corresponding standardized bugs to be 0 as we do not have prior knowledge of the relationship. In other words we expect the number of bugs to have a mean value when age also has a mean value. Thus we set $\mu = 0$ as the prior mean of α.

As for the standard deviation for α, we will make adjustments in prior predictive check so that the number of bugs follows reasonable values with the $\beta_{\text{A}[L]}$ standard deviation.

The same procedure is followed for σ.

- Determine prior $\beta_{\text{A}[L]}$

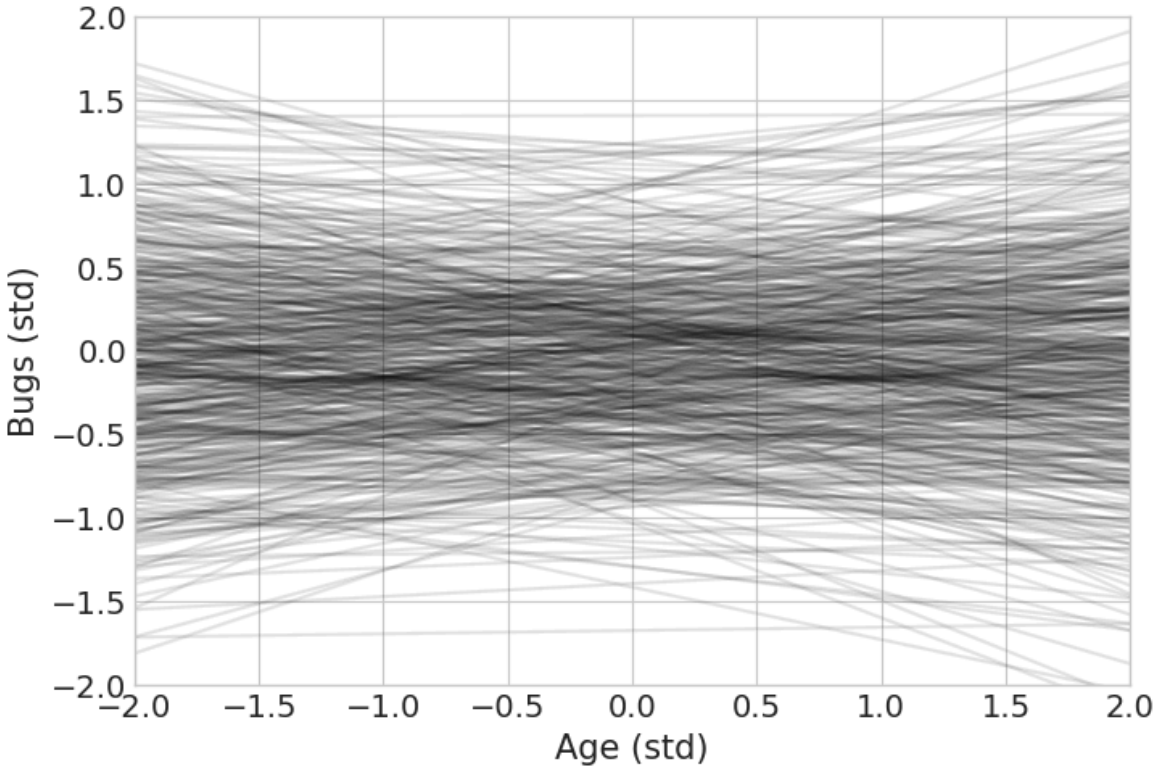
As in the previous prior predictive checks for `h2_poisson_model` we set a normal distribution as prior for the $\beta_{\text{A}[L]}$. The mean of the prior is set to 0, such that it does not assume any positive nor negative direction of the effect of age. In addition, we choose the standard the deviation such that the effect of age is on bugs is reasonable.

In [ ]:
```
with pm.Model() as h2_normal_model:
    alpha = pm.Normal("alpha", 0, 0.5)
    beta_age = pm.Normal("beta_age", 0, 0.2, shape=languages.size)
    sigma = pm.Exponential("sigma", 1.0)

    #We add language_ids and age_std as mutable data so we can change it
    language_ids = pm.Data('language_ids', df['language'], mutable=True)
    age_std = pm.Data("age_std", df.age_std, mutable=True)

    # Note the "deterministic" distribution node,
```

```
    # that beta_agesically encodes equality from the mathematical model
    mu = pm.Deterministic("mu",
                alpha + beta_age[language_ids] * age_std)

    # B = Number of Bugs Standardized
    B = pm.Normal("B", mu = mu, sigma = sigma, observed = df.bugs_std.val

    # prior predictive is a distribution of data
    # (but for each data point we also get its parameters so we can plot
    h2_normal_trace_prior = pm.sample_prior_predictive(samples = 500, ran
```

```
Sampling: [B, alpha, beta_age, sigma]
```

In [ ]:
```
# The index 0 gives us the first chain
alpha_  = h2_normal_trace_prior.prior.alpha[0]
beta_age_ = h2_normal_trace_prior.prior.beta_age[0][:,0]

fig, ax = plt.subplots()
# We use xArray, as multiplication of numpy and pandas does not do what w
xx  = xr.DataArray(np.linspace(-2, 2, 3), dims="plot_dim")
# xr does some weird matching, but effectively plus is pointwise
yy  = alpha_ + beta_age_ * xx
# transpose yy, because matplotlib wants it this way.
ax.plot(xx, yy.T, c = "k", alpha = 0.1)

ax.set_xlabel("Age (std)")
ax.set_ylabel("Bugs (std)")
ax.set_ylim(-2.0, 2.0)
ax.set_xlim(-2.0, 2.0)
plt.show()
```



The result looks fairly reasonable. Most lines are more or less flat and centered around at 0 for the standardized bugs. This aligns with our expectation on the effect of standardized age on the number of bugs.

We thereby proceed with $\beta_{\text{A}[L]} \sim \text{Normal}(0, 0.2)$.

Model fitting

We update the model as follows:

$$B_i \sim \text{Normal}(\mu_i, \sigma)$$

$$\mu_i = \alpha + \beta_{\text{A}[L]} A_i$$

$$\sigma \sim \text{Exp}(1)$$

$$\alpha \sim \text{Normal}(0, 0.5)$$

$$\beta_{\text{A}[L]} \sim \text{Normal}(0, 0.2)$$

where

$$L \in \mathcal{L}$$

$$\mathcal{L} = \{Python, Java.., C\}$$

$$|\mathcal{L}| = 17$$

In [ ]:
```
with h2_normal_model:
    h2_normal_trace = pm.sample(2000,tune=2000,idata_kwargs={"log_likelih
```

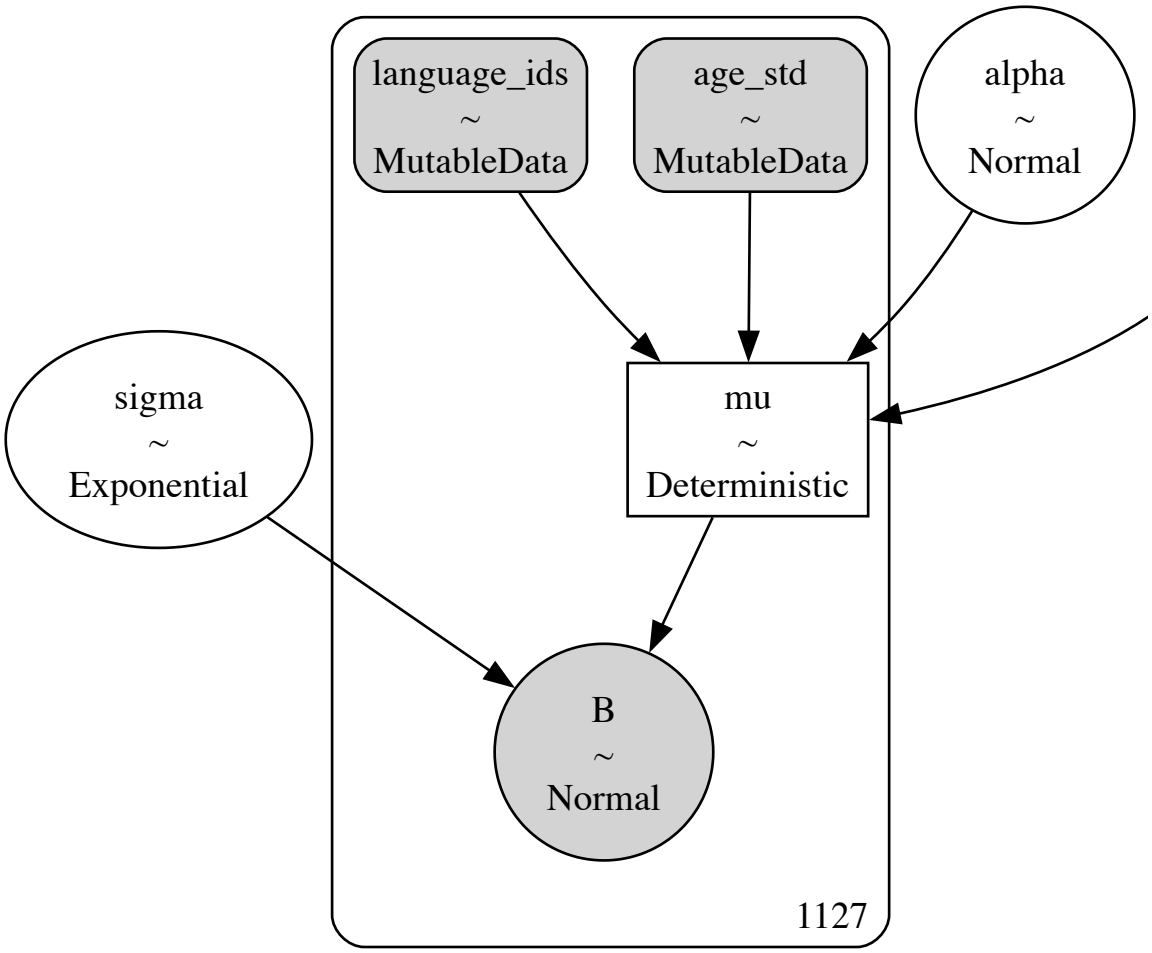
```
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [alpha, beta_age, sigma]
```

100.00% [16000/16000 00:02<00:00 Sampling 4 chains, 0 divergences]

```
Sampling 4 chains for 2_000 tune and 2_000 draw iterations (8_000 + 8_000
draws total) took 3 seconds.
```

In [ ]:
```
pm.model_graph.model_to_graphviz(h2_normal_model)
```

Out[ ]:



Below is the trace convergence for the parameters .

The effective sample sizes (`ess_bulk` , `ess_tail`) are high which indicates that the samples both in the bulk and in the tails of the posteriors have low-autocorrelation.

We also note that the Monte Carlo Standard Error `msce_mean` and `msce_sd` are 0.0, which also indicates good accuracy in the chains.

In addition, both the rhat values `r_hat` are 1.0, which means that the 4 chains have a good mixing. This is also reflected in the plot below, which shows good mixing of the chains.

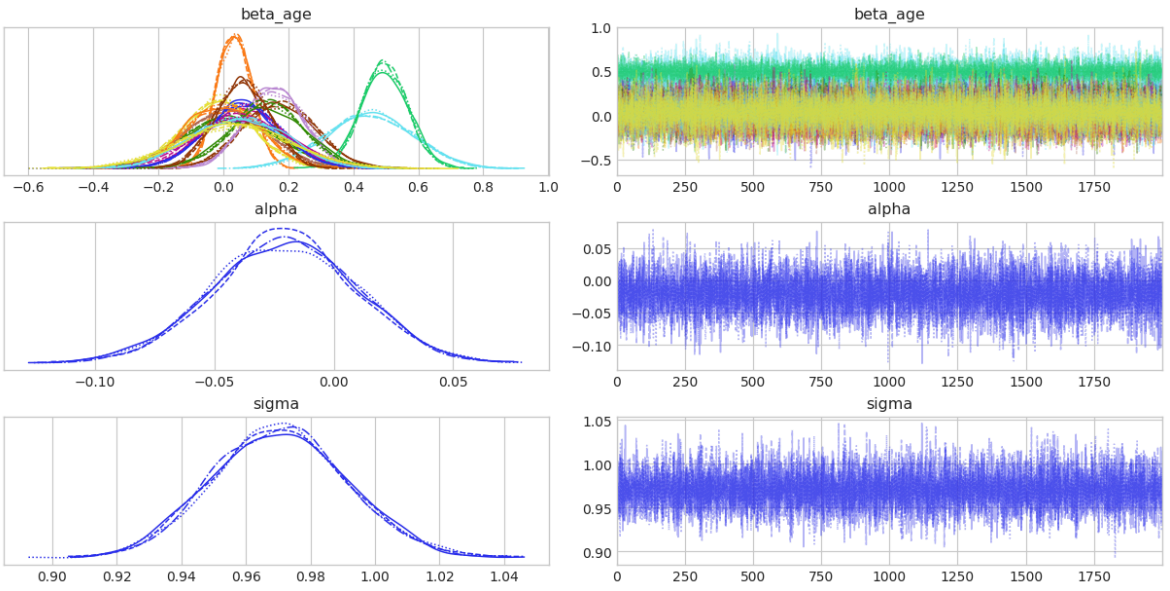
We conclude that we can proceed with this posterior.

In [ ]:
```
pm.summary(h2_normal_trace, var_names=["beta_age", "alpha", "sigma"],roun
       'ess_tail', 'r_hat']]
```

Out[ ]:

	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
beta_age[0]	0.0	0.0	15336.64	5405.02	1.0
beta_age[1]	0.0	0.0	12925.07	5815.03	1.0
beta_age[2]	0.0	0.0	13409.89	6412.93	1.0
beta_age[3]	0.0	0.0	16831.98	6298.78	1.0
beta_age[4]	0.0	0.0	13206.28	5978.21	1.0
beta_age[5]	0.0	0.0	15278.66	5303.77	1.0
beta_age[6]	0.0	0.0	15767.59	6649.09	1.0
beta_age[7]	0.0	0.0	17138.33	6176.63	1.0
beta_age[8]	0.0	0.0	14937.23	5927.27	1.0
beta_age[9]	0.0	0.0	15238.58	5930.74	1.0
beta_age[10]	0.0	0.0	17216.24	5689.61	1.0
beta_age[11]	0.0	0.0	16579.91	6171.67	1.0
beta_age[12]	0.0	0.0	15089.71	5568.40	1.0
beta_age[13]	0.0	0.0	15839.55	5843.34	1.0
beta_age[14]	0.0	0.0	15850.77	6167.91	1.0
beta_age[15]	0.0	0.0	14772.74	5973.56	1.0
beta_age[16]	0.0	0.0	15399.85	5329.24	1.0
alpha	0.0	0.0	16221.30	6601.41	1.0
sigma	0.0	0.0	14831.11	5537.20	1.0

In [ ]:
```
az.plot_trace(h2_normal_trace, var_names=["beta_age", "alpha", "sigma"])
plt.show()
```



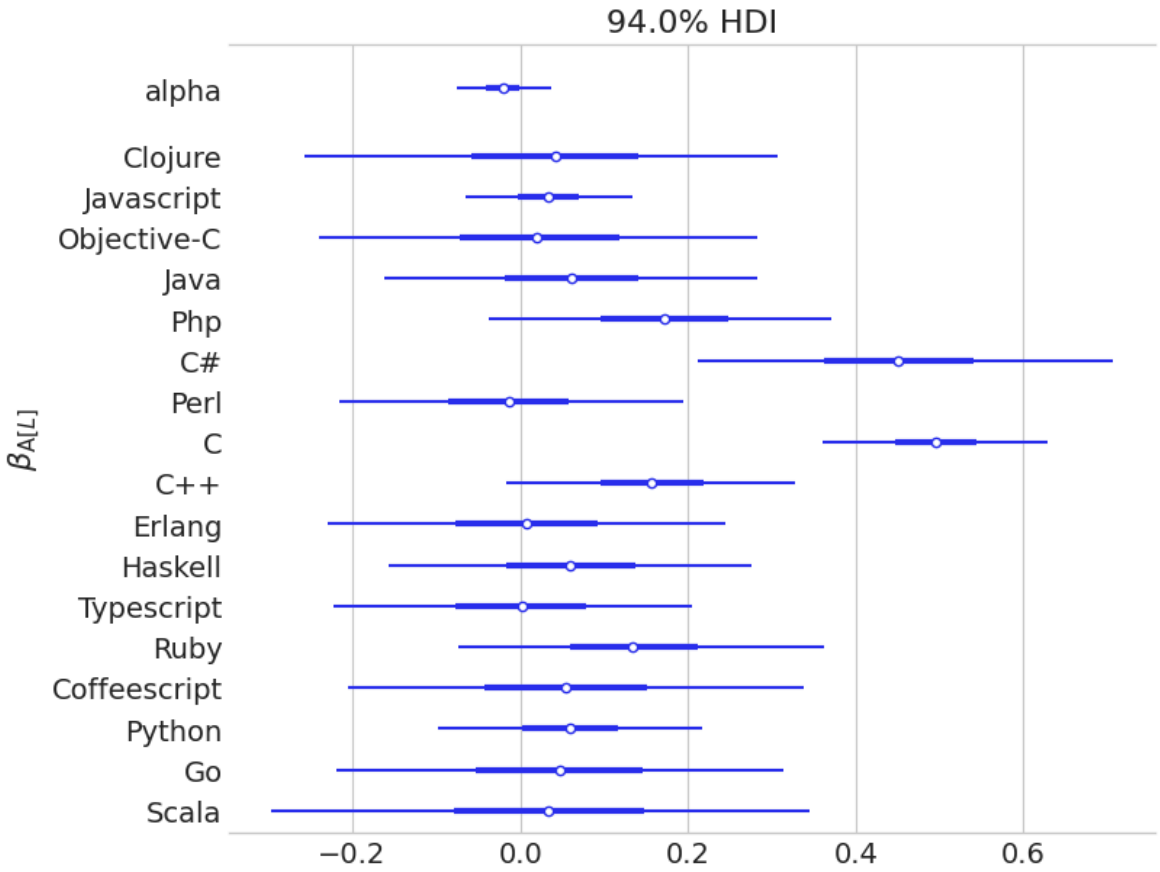
In [ ]:
```
pm.summary(h2_normal_trace, var_names=["beta_age", "alpha", "sigma"],roun
```

Out[ ]:

	mean	sd	hdi_3%	hdi_97%
beta_age[0]	0.04	0.15	-0.26	0.31
beta_age[1]	0.03	0.05	-0.07	0.13
beta_age[2]	0.02	0.14	-0.24	0.28
beta_age[3]	0.06	0.12	-0.16	0.28
beta_age[4]	0.17	0.11	-0.04	0.37
beta_age[5]	0.45	0.13	0.21	0.71
beta_age[6]	-0.01	0.11	-0.22	0.19
beta_age[7]	0.50	0.07	0.36	0.63
beta_age[8]	0.16	0.09	-0.02	0.33
beta_age[9]	0.01	0.13	-0.23	0.25
beta_age[10]	0.06	0.12	-0.16	0.28
beta_age[11]	0.00	0.11	-0.22	0.21
beta_age[12]	0.14	0.12	-0.07	0.36
beta_age[13]	0.05	0.14	-0.21	0.34
beta_age[14]	0.06	0.08	-0.10	0.22
beta_age[15]	0.05	0.14	-0.22	0.31
beta_age[16]	0.03	0.17	-0.30	0.34
alpha	-0.02	0.03	-0.08	0.04
sigma	0.97	0.02	0.93	1.01

In [ ]:

```
_,ax = plt.subplots(figsize=(8,6))
pm.plot_forest(h2_normal_trace, var_names=["alpha",'beta_age'], combined=
forest_languages = list(range(16, -1, -1))
ax.set_yticklabels(list(languages.take(forest_languages))+ ["alpha"]);
ax.set_ylabel(r"$\beta_{\mathrm{A}[L]}$")
plt.show()
```



Base on the summary and forest plot, the effect of age for many languages features a wide range of HDI suggesting that the estimates of the effects of age on bugs in each language have high uncertainty. All estimates except `C` and `C#` crosses 0 meaning that the effect of age on bugs in most of the languages are neither positive or negative, but rather that age has no effect in these languages. In contrast, the estimates of age in `C#` and `C` languages have a positive impact on bugs that does not cross 0, but we speculate that this could be due to an ommited-variable bias.

In addition, we also see that α has stayed close to its prior mean of 0 but now has a higher certainty.

Posterior predictive check

Finally, we perform posterior predictive check to see if the model fits the data well. Due to the model restriction, we visualize the results per language.

We first sampled 1127 points from `age_std` values between -1.5 to 5 given a specified language (`Perl`) and visualized them along with the actual datapoints.

In [ ]:
```
pareto_k_h2 = az.loo(h2_normal_trace, pointwise=True).pareto_k.values
print("Max PSIS value for beta_age: ", max(pareto_k_h2))
```

```
Max PSIS value for beta_age:  6.738907382675382
```

In [ ]:
```
# No. 6 for Perl
print(f"Perl \n")
print("Minimum project age:",df[df['language']==6].age_std.min())
print("Maximum project age:",df[df['language']==6].age_std.max())
print()
```

```
print("Minimum number of bugs:",df[df['language']==6].bugs_std.min())
print("Maximum number of bugs:",df[df['language']==6].bugs_std.max())
```

```
Perl

Minimum project age: -0.9181804910047063
Maximum project age: 3.1876957819621703

Minimum number of bugs: -0.12811232331968753
Maximum number of bugs: 0.08261848654565365
```

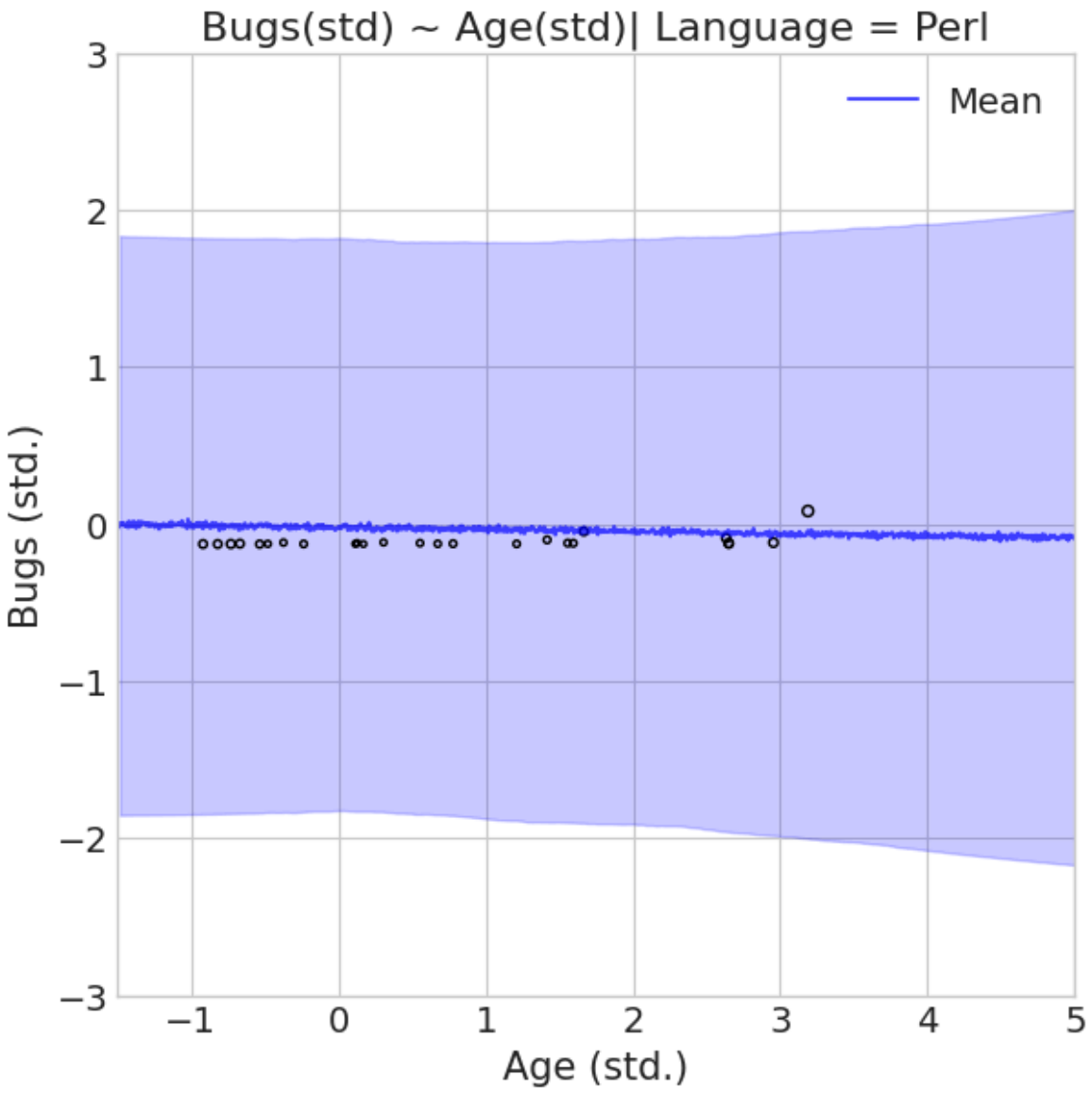
In [ ]:
```
n_points = len(df)
x_seq = np.linspace(-1.5, 5, n_points)

with h2_normal_model:
    pm.set_data({"age_std": x_seq, "language_ids": np.repeat(6, n_points)
    post_pred_B_h2_perl = pm.sample_posterior_predictive(h2_normal_trace,

post_pred_B_h2_mean_perl = post_pred_B_h2_perl.mean(["chain", "draw"])
```

```
Sampling: [B]
```

The results for `Perl` language is shown below.

- We see a near flat line at `bug_std` = 0. This disproves **H2** and indicates that regardless of projects' age, the `bug_std` get an expected value of 0.
- The posterior also seem to take a wide range of values from -2.0 to 2.0. The model doesn't fit to these data points well.

In [ ]:
```
plt.figure(figsize=(6,6))

pareto_k_h2 /= pareto_k_h2.max()
pareto_k_h2_size = 250 * pareto_k_h2
pareto_k_h2_size += abs(min(pareto_k_h2_size))+0.001 #fix error with nega

pareto_k_h2_size_p = []
for i in df[df['language']==6].index:
    pareto_k_h2_size_p.append(pareto_k_h2_size[i])

az.plot_hdi(x_seq, post_pred_B_h2_perl, color = 'b', fill_kwargs={'alpha'
plt.plot(x_seq, post_pred_B_h2_mean_perl, color = 'b', alpha=0.7, label='

# No. 6 for Perl
plt.scatter(df[df['language']==6].age_std, df[df['language']==6].bugs_std

plt.legend()
plt.xlim(-1.5,5.0)
#plt.ylim(0,20000)
plt.ylim(-3.0,3.0)
plt.xlabel('Age (std.)')
plt.ylabel('Bugs (std.)')
plt.title("Bugs(std) ~ Age(std)| Language = Perl");
```



We also sampled 1127 points from `age_std` values between -1.5 to 5 given a specified language (`C`) and and visualized them along with the actual datapoints.

In [ ]:
```
# No. 7 for C
print("C \n")
print("Minimum project age:",df[df['language']==7].age_std.min())
print("Maximum project age:",df[df['language']==7].age_std.max())
print()
print("Minimum number of bugs:",df[df['language']==7].bugs_std.min())
print("Maximum number of bugs:",df[df['language']==7].bugs_std.max())
```

```
C

Minimum project age: -1.1174164544142466
Maximum project age: 4.759147006872553

Minimum number of bugs: -0.1283702557675031
Maximum number of bugs: 31.15857773181636
```

In [ ]:
```
n_points = len(df)
x_seq = np.linspace(-1.5, 5, n_points)

with h2_normal_model:
    pm.set_data({"age_std": x_seq, "language_ids": np.repeat(7, n_points)})
```

```
    post_pred_B_h2_C = pm.sample_posterior_predictive(h2_normal_trace, va

post_pred_B_h2_mean_C = post_pred_B_h2_C.mean(["chain", "draw"])
```

```
Sampling: [B]
```

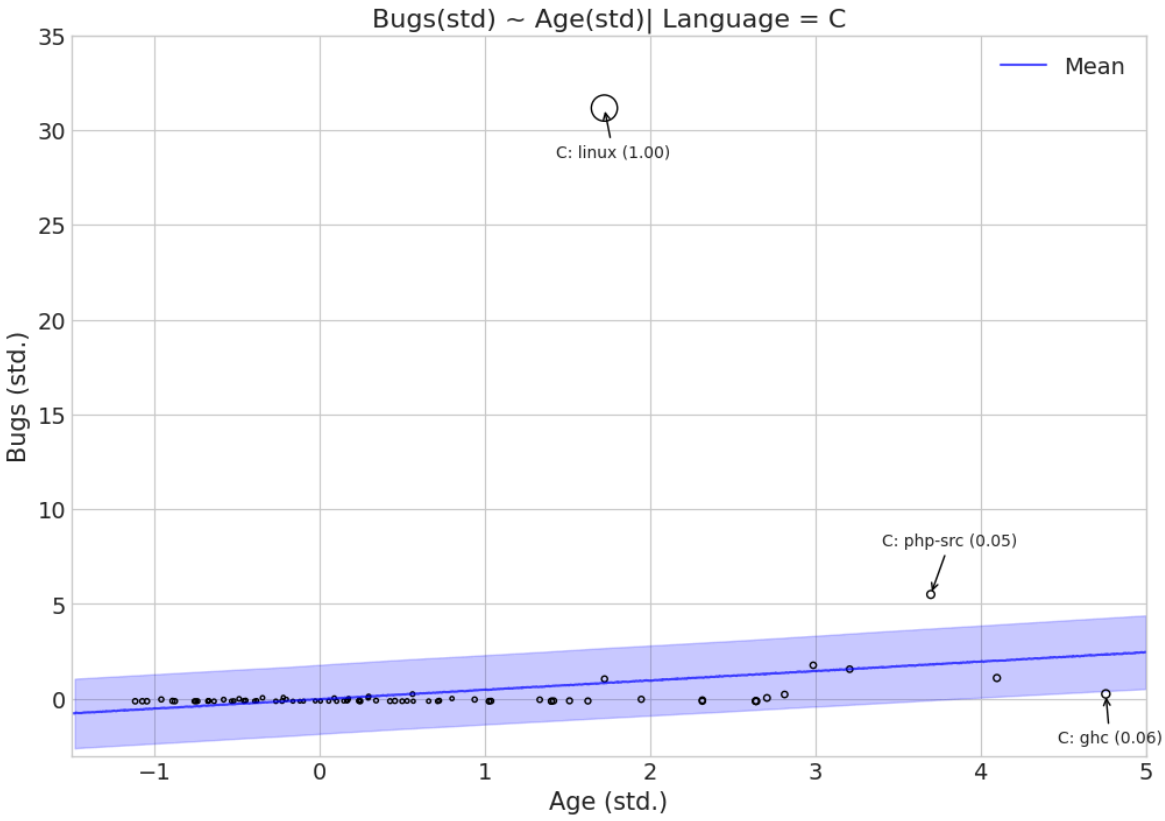
```
3.58% [286/8000 00:00<00:00]
```

The results for `C` language is shown below.

- We observe a positive impact of age on bugs with high confidence. The posterior, `beta_age[7]`, was estimated to have mean of0.49, meaning a change of 1 in `age_std` in the language `C` is linked to a positive change in `bugs_std`.
- As the influential datapoint `Linux` is written in `C`, the estimated impact from age in `C` on bugs could be an artifact of this particular project. This is also shown in the plot below.

In [ ]:
```
plt.figure(figsize=(10,7))

pareto_k_h2 /= pareto_k_h2.max()
pareto_k_h2_size = 250 * pareto_k_h2
pareto_k_h2_size += abs(min(pareto_k_h2_size))+0.001 #fix error with nega

pareto_k_h2_size_c = []
for i in df[df['language']==7].index:
    pareto_k_h2_size_c.append(pareto_k_h2_size[i])

#Get the 50 data points with the largest pareto_k_values
top_indices = np.argsort(pareto_k_h2)[-50:][::-1]
top_data_points = df.iloc[top_indices]

az.plot_hdi(x_seq, post_pred_B_h2_C, color = 'b', fill_kwargs={'alpha': 0
plt.plot(x_seq, post_pred_B_h2_mean_C, color = 'b', alpha=0.7, label='Mea

# No. 7 for C
plt.scatter(df[df['language']==7].age_std, df[df['language']==7].bugs_std

#Plotting text for data points with high pareto-k values. We manually adj
for index, row in top_data_points.iterrows():
    dont_annotate = False
    if row['language'] == 7 and row['project'] == 'linux':
        position=(-30,-30)
    elif row['language'] == 7 and row['project'] == 'php-src':
        position=(-30,30)
    elif row['language'] == 7 and row['project'] == 'ghc':
        position=(-30,-30)
    else:
        dont_annotate = True
    if dont_annotate == False:
         plt.annotate(f"{languages[row['language']]}: {row['project']} ({
                     textcoords='offset points', arrowprops=dict(arrowsty

plt.legend()
plt.xlim(-1.5,5.0)
#plt.ylim(0,20000)
plt.ylim(-3.0,35.0)
plt.xlabel('Age (std.)')
```

```
plt.ylabel('Bugs (std.)')
plt.title("Bugs(std) ~ Age(std)| Language = C");
```



Conclusion H2

Based on the results, we can reject **H2** and conclude that age doesn't have a positive effect for *all* languages.

`C#` and `C` languages are estimated with a positive coefficient, but we believe that it may be caused by the an omitted variable bias.

These results are different from the `h2_poisson_model` which found a reliable effect of age on bugs *across* languages. We believe that there may be too few data points for each language in the dataset, to reliably estiamate the effects of age on bugs in each language. But when aggregating by language, we have all datapoints available to estimate the effect of age on bug.

H3

- **H3** - Number of commits (C) does not impact the effect of age (A) on the number of bugs (B) for any programming language (L). That is, the effect of age (A), conditioned on number of commits (C), on number of bugs (B) is the same as the direct effect of age (A) on number of bugs (B).

Normal Regression

We condition the effect of age by including number of commits as a predictor in the model. As shown in the fork-shaped DAG of `h3_poisson_model`, we believe that conditioning on commits will remove some of the effect we saw in `h2_normal_model` for age on number of bugs. This is done by adding an effect of commits on bugs for each language, just like the effect of age seen in `h2_normal_model`.

We reuse the priors for σ, α, and $\beta_{\mathrm{A}[L]}$ as defined in `h2_normal_model`. In the next section we determine the prior for $\beta_{C[L]}$. The model is defined as follows:

$$B_i \sim \mathrm{Normal}(\mu_i, \sigma)$$

$$\mu_i = \alpha + \beta_{\mathrm{A}[L]} A_i + \beta_{\mathrm{C}[L]} C_i$$

$$\sigma \sim \mathrm{Exp}(1)$$

$$\alpha \sim \mathrm{Normal}(0, 0.5)$$

$$\beta_{\mathrm{A}[L]} \sim \mathrm{Normal}(0, 0.2)$$

$$\beta_{\mathrm{C}[L]} \sim \text{Not yet determined}$$

where

$$L \in \mathcal{L}$$

$$\mathcal{L} = \{Python, Java.., C\}$$

$$|\mathcal{L}| = 17$$

Determine prior for $\beta_{C[L]}$

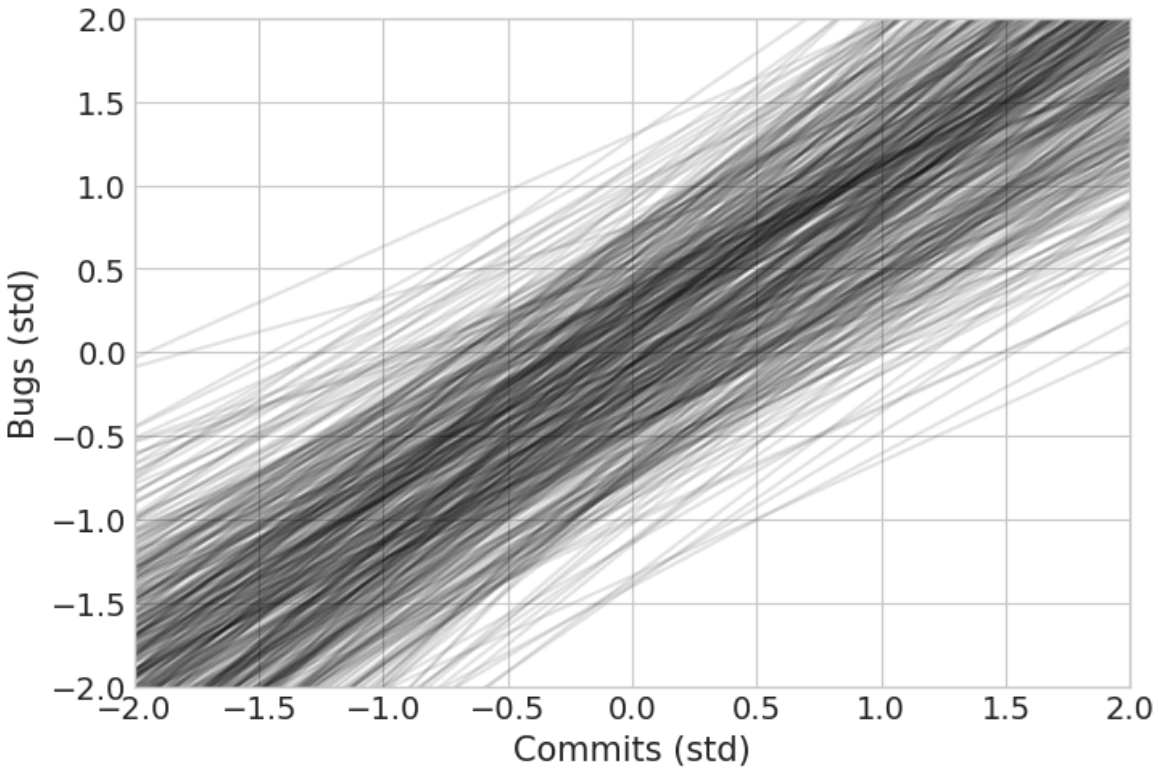
We expect the $\beta_{\mathrm{C}[L]}$ to have a positive effect on number of bugs as explained in `h3_poisson_model` and to follow a normal distribution. Thus we use a positive mean and simulate the effects of varying standard deviations below.

In [ ]:
```
with pm.Model() as h3_normal_model:
    alpha = pm.Normal("alpha", 0, 0.5)
    beta_age = pm.Normal("beta_age", 0, 0.2, shape=languages.size)
    beta_commits = pm.Normal("beta_commits", 1, 0.2, shape=languages.size
    sigma = pm.Exponential("sigma", 1)

    #We add language_ids and age_std, com_std as mutable data so we can c
    language_ids = pm.Data('language_ids', df['language'], mutable=True)
    age_std = pm.Data("age_std", df.age_std, mutable=True)
    commits_std = pm.Data("commits_std", df.commits_std, mutable=True)

    # Note the "deterministic" distribution node,
    # that basically encodes equality from the mathematical model
    mu = pm.Deterministic("mu",
                alpha + beta_age[language_ids] * age_std + beta_commits[l

    # B = Number of Bugs Standardized
    B = pm.Normal("B", mu = mu, sigma = sigma, observed = df.bugs_std.val
```

```
    # prior predictive is a distribution of data
    # (but for each data point we also get its parameters so we can plot
    h3_normal_trace_prior = pm.sample_prior_predictive(samples = 500, ran
```

```
Sampling: [B, alpha, beta_age, beta_commits, sigma]
```

In [ ]:
```
# The index 0 gives us the first chain
# (there is only one chain for the prior anyway)
alpha_  = h3_normal_trace_prior.prior.alpha[0]
beta_commits = h3_normal_trace_prior.prior.beta_commits[0][:,0]

fig, ax = plt.subplots()
# We use xArray, as multiplication of numpy and pandas does not do what w
xx  = xr.DataArray(np.linspace(-2, 2, 3), dims="plot_dim")
# xr does some weird matching, but effectively plus is pointwise
# and multiplication is bA_.T * xx (so matrix multiplication with the fir
# We get an array of rows (one per sample from the prior) and the rows co
# (the size of xx)
yy  = alpha_ + beta_commits * xx
# transpose yy, because matplotlib wants it this way.
ax.plot(xx, yy.T, c = "k", alpha = 0.1)

ax.set_xlabel("Commits (std)")
ax.set_ylabel("Bugs (std)")
ax.set_ylim(-2.0, 2.0)
ax.set_xlim(-2.0, 2.0)
plt.show()
```



All lines have a positive slope for number of commits, which reflect our assumption that the number of commits has a positive impact on the number of bugs. In addition, we also note that the effects lie within a non exploding range.

Thus we proceed with $\beta_{\mathrm{C}[L]} \sim \mathrm{Normal}(1, 0.2)$.

Model Fitting

We update the model as follows:

$$B_i \sim \text{Normal}(\mu_i, \sigma)$$

$$\mu_i = \alpha + \beta_{\text{A}[L]} A_i + \beta_{\text{C}[L]} C_i$$

$$\sigma \sim \text{Exp}(1)$$

$$\alpha \sim \text{Normal}(0, 0.5)$$

$$\beta_{\text{A}[L]} \sim \text{Normal}(0, 0.2)$$

$$\beta_{\text{C}[L]} \sim \text{Normal}(1, 0.2)$$

where

$$L \in \mathcal{L}$$

$$\mathcal{L} = \{Python, Java.., C\}$$

$$|\mathcal{L}| = 17$$

In [ ]:
```
with h3_normal_model:
    h3_normal_trace = pm.sample(2000, tune=2000, idata_kwargs={"log_likel
```

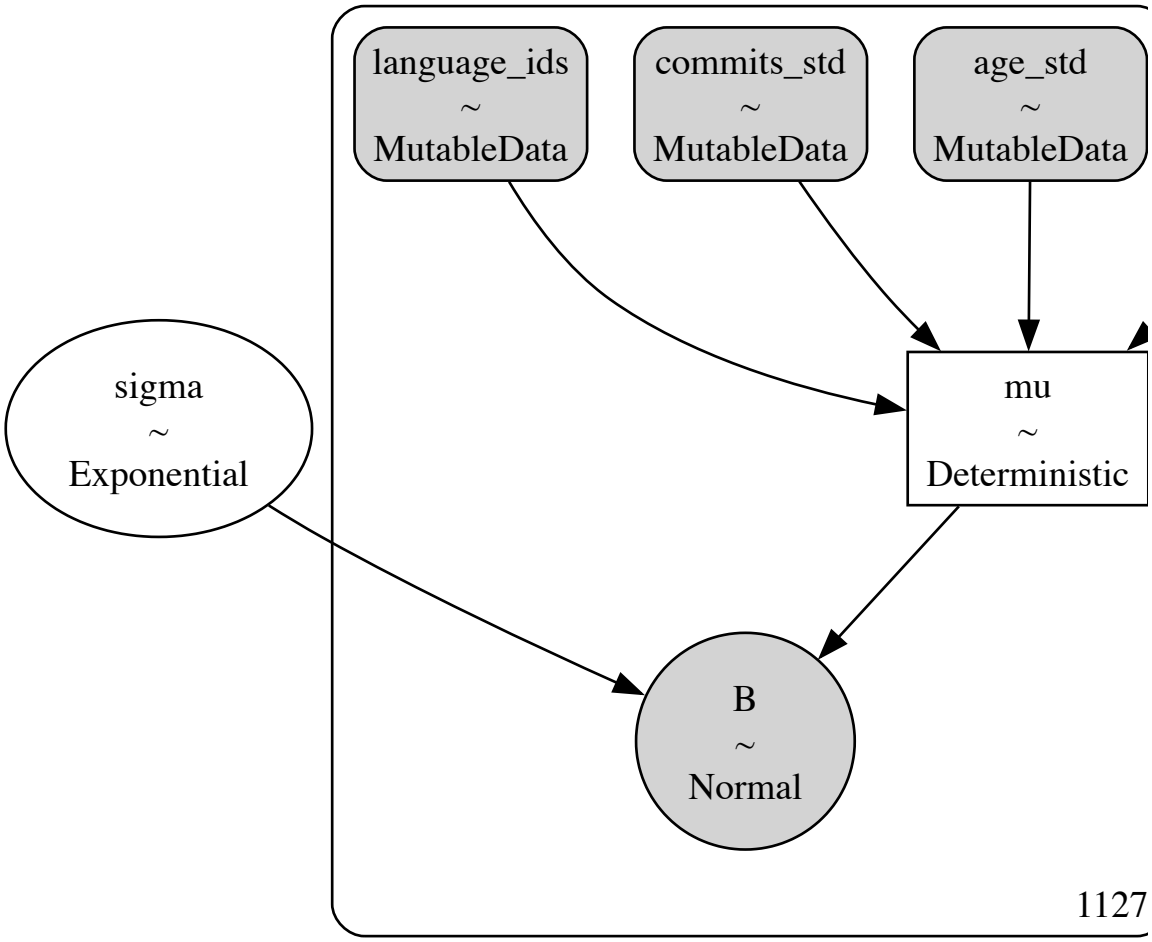
```
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [alpha, beta_age, beta_commits, sigma]
```

100.00% [16000/16000 00:04<00:00 Sampling 4 chains, 0 divergences]

```
Sampling 4 chains for 2_000 tune and 2_000 draw iterations (8_000 + 8_000
draws total) took 4 seconds.
```

In [ ]:
```
pm.model_graph.model_to_graphviz(h3_normal_model)
```

Out[ ]:



Below is the trace convergence for the parameters.

The effective sample sizes (`ess_bulk` , `ess_tail`) are high which indicates that the samples both in the bulk and in the tails of the posteriors have low-autocorrelation.

We also note that the Monte Carlo Standard Error `msce_mean` and `msce_sd` are 0.0, which also indicates good accuracy in the chains.

In addition, both the rhat values `r_hat` are 1.0, which means that the 4 chains have a good mixing. This is also reflected in the plot below, which shows good mixing of the chains.

We conclude that we can proceed with this posterior.

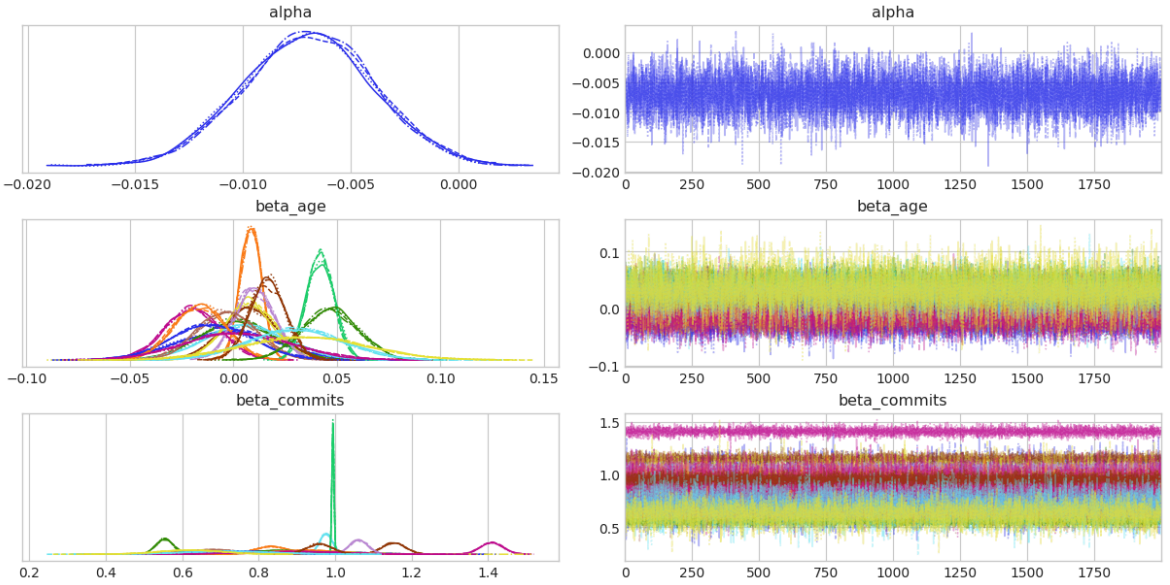
In [ ]:
```
pm.summary(h3_normal_trace, var_names=["alpha", "beta_age", "beta_commits
       'ess_tail', 'r_hat']]
```

Out[ ]:

	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
alpha	0.0	0.0	9613.83	6380.05	1.0
beta_age[0]	0.0	0.0	8727.13	6511.35	1.0
beta_age[1]	0.0	0.0	13174.01	6004.98	1.0
beta_age[2]	0.0	0.0	10487.92	6228.18	1.0
beta_age[3]	0.0	0.0	13122.30	5533.38	1.0
beta_age[4]	0.0	0.0	10664.79	6518.87	1.0
beta_age[5]	0.0	0.0	7530.48	5990.43	1.0
beta_age[6]	0.0	0.0	11740.00	5723.51	1.0
beta_age[7]	0.0	0.0	11572.24	6081.26	1.0
beta_age[8]	0.0	0.0	11480.00	7278.00	1.0
beta_age[9]	0.0	0.0	13054.66	6247.32	1.0
beta_age[10]	0.0	0.0	7149.85	6237.17	1.0
beta_age[11]	0.0	0.0	13996.78	5907.96	1.0
beta_age[12]	0.0	0.0	9846.68	6633.62	1.0
beta_age[13]	0.0	0.0	7622.67	6439.41	1.0
beta_age[14]	0.0	0.0	12454.32	6371.82	1.0
beta_age[15]	0.0	0.0	8053.40	6578.03	1.0
beta_age[16]	0.0	0.0	10348.87	6151.23	1.0
beta_commits[0]	0.0	0.0	8345.93	6143.35	1.0
beta_commits[1]	0.0	0.0	11320.79	7006.65	1.0
beta_commits[2]	0.0	0.0	11753.53	6188.69	1.0
beta_commits[3]	0.0	0.0	12613.10	5875.08	1.0
beta_commits[4]	0.0	0.0	9588.20	6810.14	1.0
beta_commits[5]	0.0	0.0	6982.88	6062.48	1.0
beta_commits[6]	0.0	0.0	12816.45	6474.77	1.0
beta_commits[7]	0.0	0.0	10755.54	5475.54	1.0
beta_commits[8]	0.0	0.0	12167.16	6660.02	1.0
beta_commits[9]	0.0	0.0	13282.54	6067.54	1.0
beta_commits[10]	0.0	0.0	7266.73	6385.94	1.0
beta_commits[11]	0.0	0.0	12435.93	5533.71	1.0
beta_commits[12]	0.0	0.0	9570.07	7118.33	1.0
beta_commits[13]	0.0	0.0	7572.56	6509.53	1.0
beta_commits[14]	0.0	0.0	11365.21	6408.70	1.0
beta_commits[15]	0.0	0.0	8094.07	6645.60	1.0

	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
beta_commits[16]	0.0	0.0	9744.77	6139.54	1.0

In [ ]:
```
az.plot_trace(h3_normal_trace, var_names=["alpha", "beta_age", "beta_comm
plt.show()
```



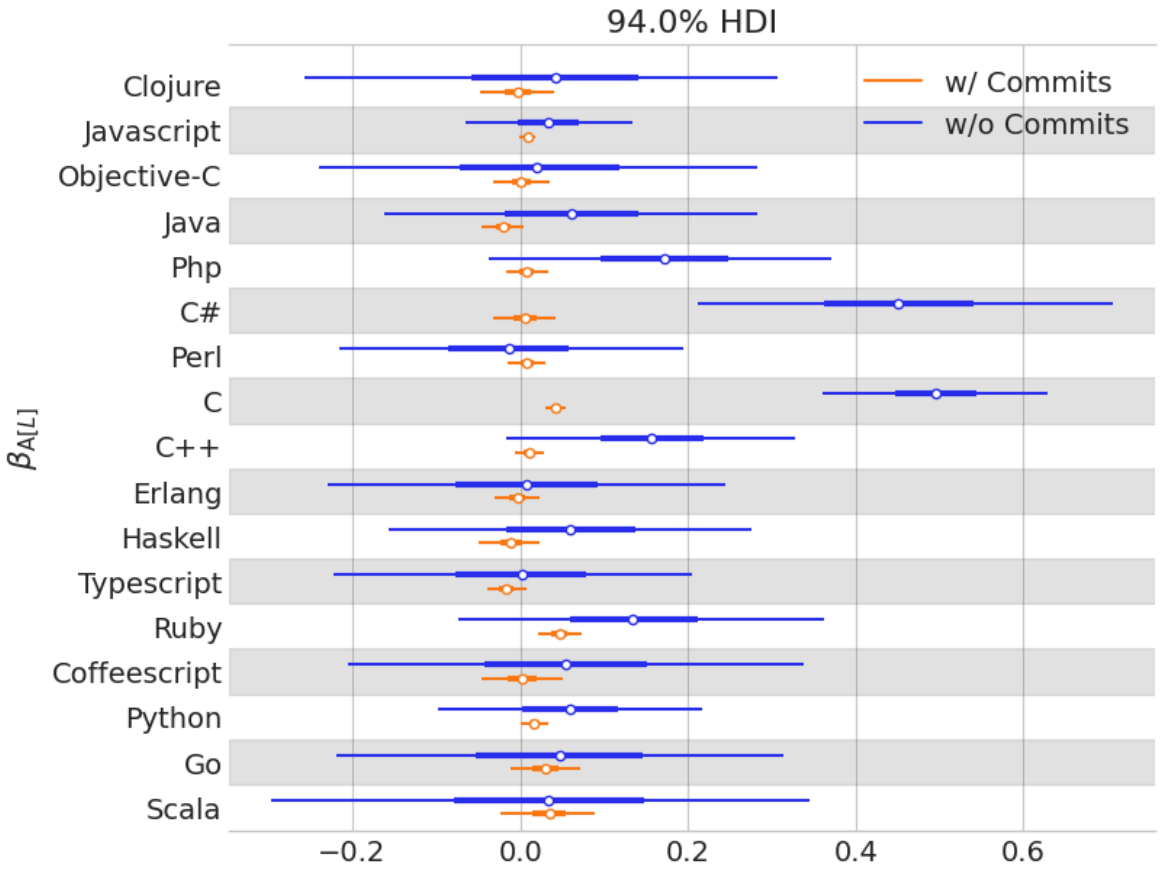
In [ ]:
```
pm.summary(h3_normal_trace, var_names=["alpha", "beta_age", "beta_commits
```

Out[ ]:

	mean	sd	hdi_3%	hdi_97%
alpha	-0.01	0.00	-0.01	-0.00
beta_age[0]	-0.00	0.02	-0.05	0.04
beta_age[1]	0.01	0.01	-0.00	0.02
beta_age[2]	0.00	0.02	-0.03	0.03
beta_age[3]	-0.02	0.01	-0.05	0.00
beta_age[4]	0.01	0.01	-0.02	0.03
beta_age[5]	0.01	0.02	-0.03	0.04
beta_age[6]	0.01	0.01	-0.01	0.03
beta_age[7]	0.04	0.01	0.03	0.06
beta_age[8]	0.01	0.01	-0.01	0.03
beta_age[9]	-0.00	0.01	-0.03	0.02
beta_age[10]	-0.01	0.02	-0.05	0.02
beta_age[11]	-0.02	0.01	-0.04	0.01
beta_age[12]	0.05	0.01	0.02	0.07
beta_age[13]	0.00	0.03	-0.05	0.05
beta_age[14]	0.02	0.01	0.00	0.03
beta_age[15]	0.03	0.02	-0.01	0.07
beta_age[16]	0.03	0.03	-0.02	0.09
beta_commits[0]	0.94	0.12	0.71	1.17
beta_commits[1]	0.83	0.04	0.75	0.92
beta_commits[2]	0.86	0.09	0.69	1.03
beta_commits[3]	1.41	0.03	1.35	1.46
beta_commits[4]	1.15	0.03	1.10	1.21
beta_commits[5]	0.98	0.02	0.94	1.01
beta_commits[6]	0.93	0.12	0.70	1.15
beta_commits[7]	0.99	0.00	0.99	1.00
beta_commits[8]	1.06	0.02	1.02	1.11
beta_commits[9]	0.68	0.08	0.53	0.83
beta_commits[10]	0.73	0.11	0.53	0.93
beta_commits[11]	0.92	0.09	0.75	1.07
beta_commits[12]	0.55	0.02	0.51	0.60
beta_commits[13]	0.88	0.13	0.64	1.11
beta_commits[14]	0.96	0.03	0.89	1.02
beta_commits[15]	0.71	0.11	0.50	0.93

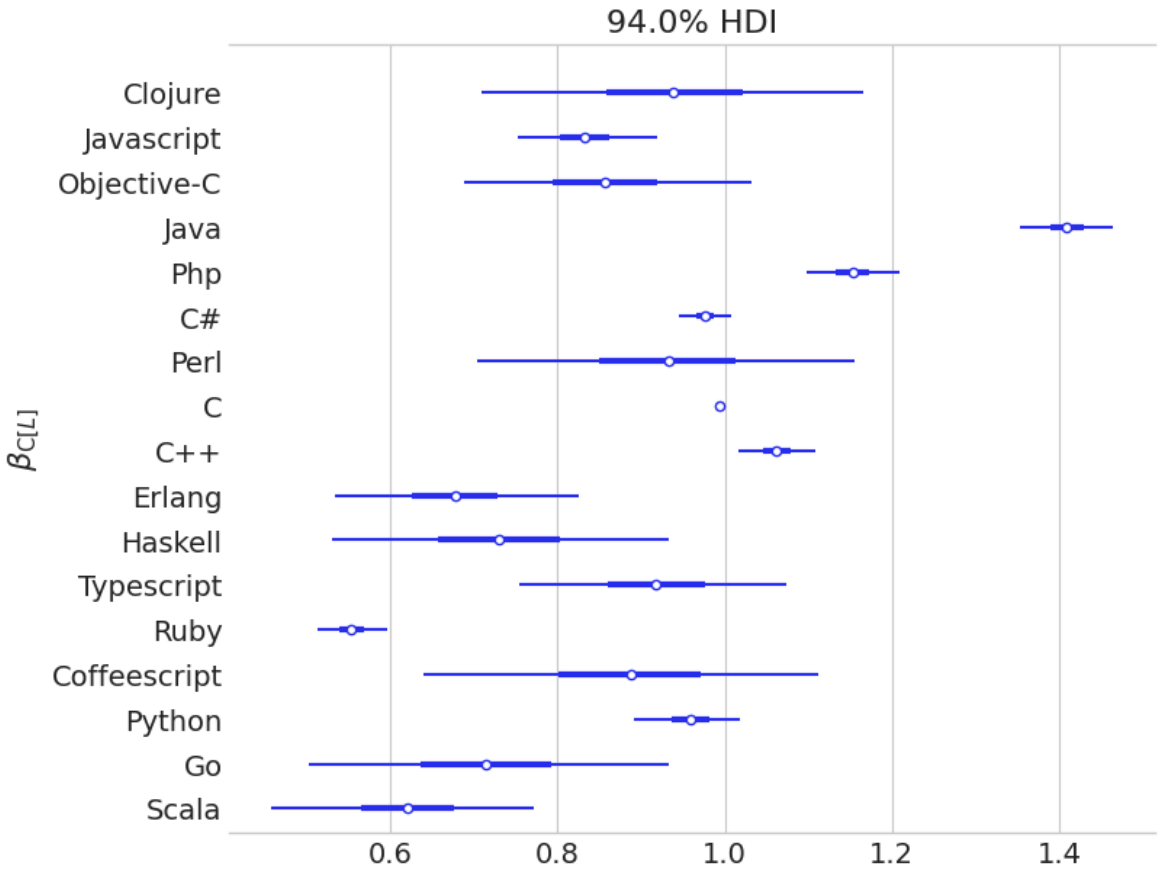
	mean	sd	hdi_3%	hdi_97%
beta_commits[16]	0.62	0.08	0.46	0.77

In [ ]:
```
_,ax = plt.subplots(figsize=(8,6))
pm.plot_forest((h2_normal_trace, h3_normal_trace), model_names=['w/o Comm
forest_languages = list(range(16, -1, -1))
ax.set_yticklabels(languages.take(forest_languages));
ax.set_ylabel(r"$\beta_{\mathrm{A}[L]}$")
plt.show()
```



From the plot, it is clear that after conditioning on the number of commits, the effect of age is pulled even closer towards 0 with a more confident estimation. However, we still see that the langugae `C` is confidently estimated to be positive. This brings evidence that disproves **H3**. This result aligns with our assumption that the effect of age on bugs is primarily rooted in the effect of commits (see fork-shaped DAG in `h3_poisson_model`).

In [ ]:
```
_,ax = plt.subplots(figsize=(8,6))
pm.plot_forest(h3_normal_trace, var_names=['beta_commits'], combined=True
forest_languages = list(range(16, -1, -1))
ax.set_yticklabels(languages.take(forest_languages));
ax.set_ylabel(r"$\beta_{\mathrm{C}[L]}$")
plt.show()
```



We see all $\beta_{\mathrm{C}[L]}$ get a positive coefficient although there is a large variation in the HDIs among the coefficients.

Posterior predictive check

Again, we do posterior predictive check to evaluate the model fitting.

Due to the model restriction, we can only visualize the results per language. We sampled 1127 points from `age_std` values between -1.5 to 5 given a specified language (`Perl`) and visualize them along with the actual data points.

In [ ]:
```
pareto_k_h3 = az.loo(h3_normal_trace, pointwise=True).pareto_k.values
print("Max PSIS value for beta_age: ", max(pareto_k_h3))
```

```
Max PSIS value for beta_age:  2.4691969801483973
```

In [ ]:
```
n_points = len(df)
x_seq = np.linspace(-1.5, 5, n_points)

with h3_normal_model:
    pm.set_data({"age_std": x_seq, "commits_std": np.repeat(0, n_points),
    post_pred_B_h3_perl = pm.sample_posterior_predictive(h3_normal_trace,

post_pred_B_h3_mean_perl = post_pred_B_h3_perl.mean(["chain", "draw"])
```

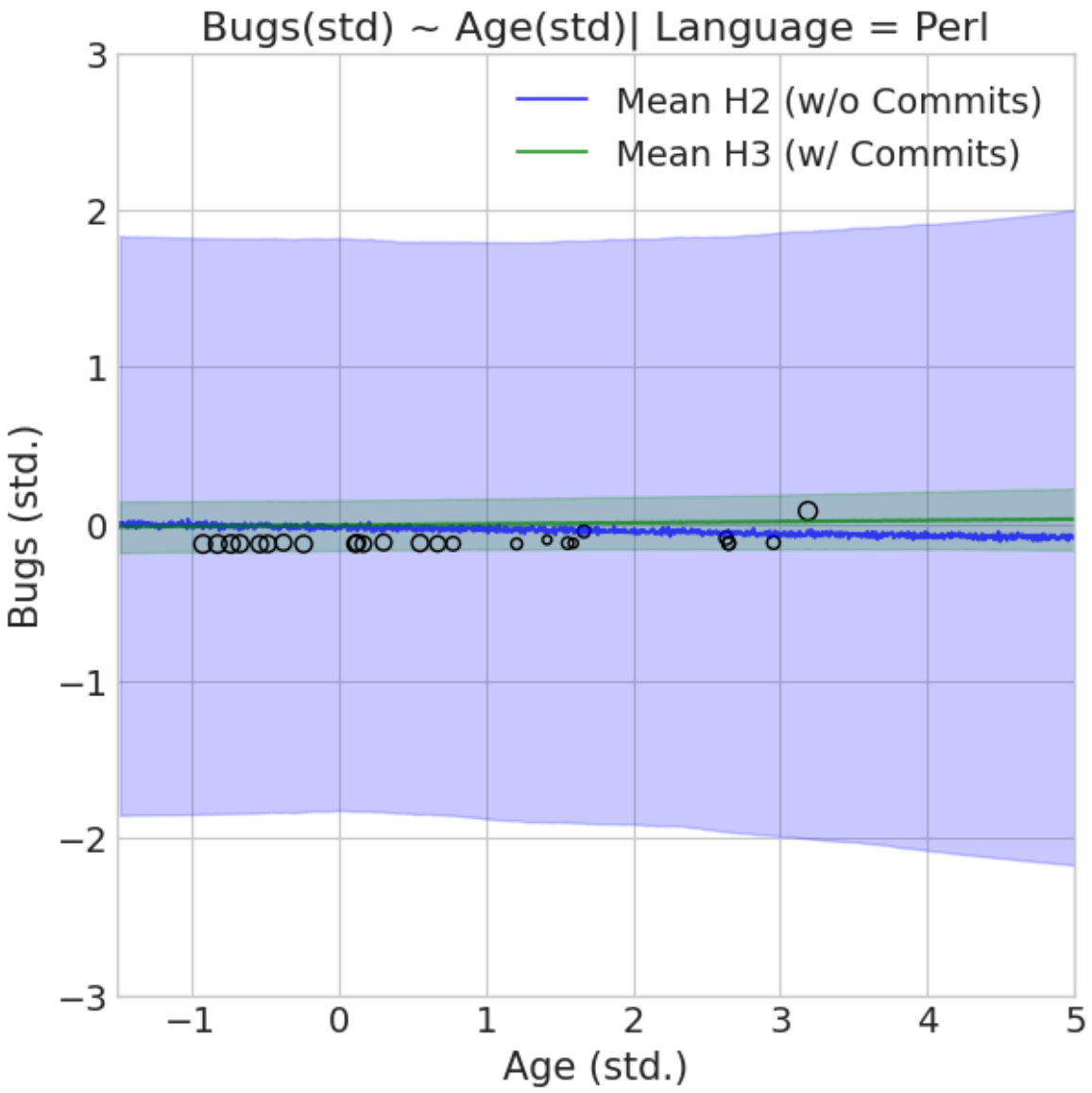
```
Sampling: [B]
```

18.29% [1463/8000 00:00<00:00]

The results for Perl language is shown below.

- We still see a near flat line (green) at `bug_std` = 0. This indicates that regardless of projects' age, the standardized bugs get an expected value of 0.
- The new posterior has a more narrow HDI (green), which makes the model more certain about the age effect.

In [ ]:
```
plt.figure(figsize=(6,6))

pareto_k_h3 /= pareto_k_h3.max()
pareto_k_h3_size = 250 * pareto_k_h3
pareto_k_h3_size += abs(min(pareto_k_h3_size))+0.001 #fix error with nega

pareto_k_h3_size_p = []
for i in df[df['language']==6].index:
    pareto_k_h3_size_p.append(pareto_k_h3_size[i])

az.plot_hdi(x_seq, post_pred_B_h2_perl, color = 'b', fill_kwargs={'alpha'
plt.plot(x_seq, post_pred_B_h2_mean_perl, color = 'b', alpha=0.7, label='
az.plot_hdi(x_seq, post_pred_B_h3_perl, color = 'g', fill_kwargs={'alpha'
plt.plot(x_seq, post_pred_B_h3_mean_perl, color = 'g', alpha=0.7, label='
# No.6 for Perl
plt.scatter(df[df['language']==6].age_std, df[df['language']==6].bugs_std

plt.legend()
plt.xlim(-1.5,5.0)
#plt.ylim(0,20000)
plt.ylim(-3.0,3.0)
plt.xlabel('Age (std.)')
plt.ylabel('Bugs (std.)')
plt.title("Bugs(std) ~ Age(std)| Language = Perl");
```



We also sampled 1127 points from `age_std` values between -1.5 to 5 given a specified language `C` and and visualized them along with the actual datapoints.

In [ ]:
```
n_points = len(df)
x_seq = np.linspace(-1.5, 5, n_points)

with h3_normal_model:
    pm.set_data({"age_std": x_seq, "commits_std": np.repeat(0, n_points),
    post_pred_B_h3_C = pm.sample_posterior_predictive(h3_normal_trace, va

post_pred_B_h3_mean_C = post_pred_B_h3_C.mean(["chain", "draw"])
```

```
Sampling: [B]
```

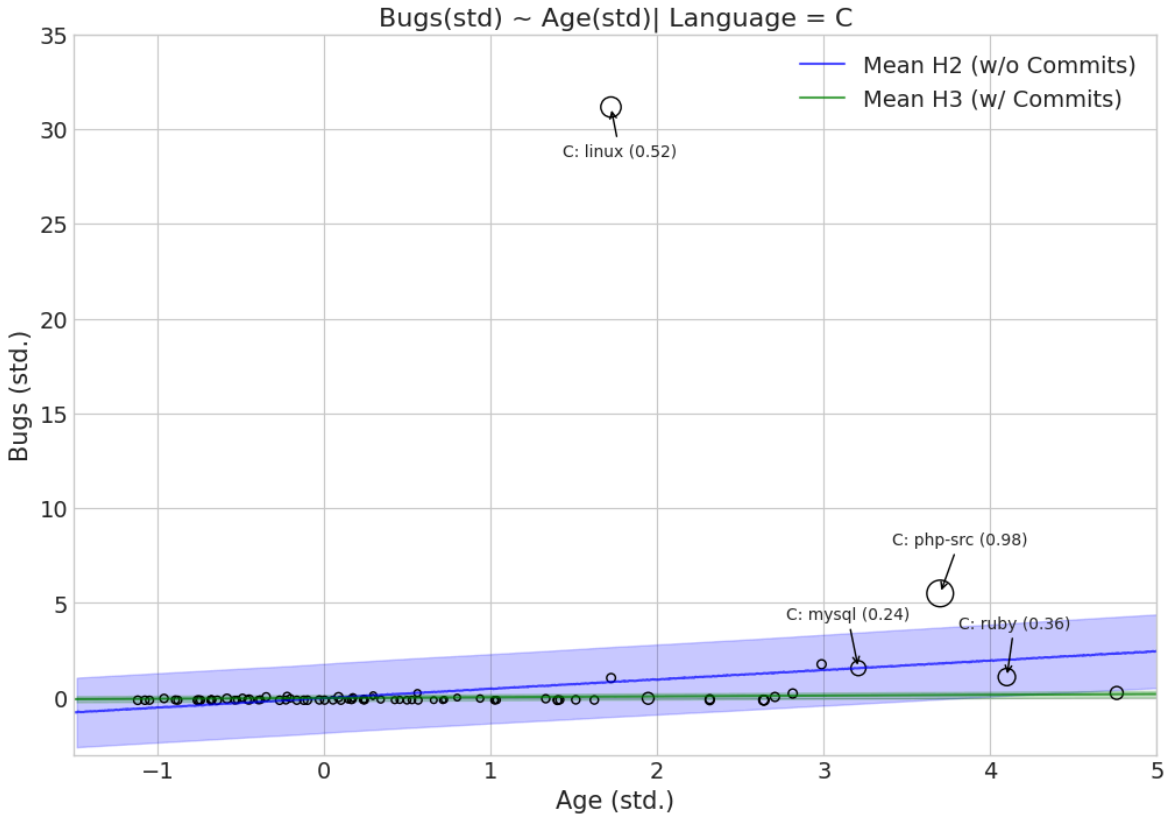
100.00% [8000/8000 00:00<00:00]

The results for `C` language is shown below.

- After conditioned on commits, the effect of age was reduced to be close to 0. The posterior `beta_age[7]` mean was estimated at 0.04 which is a significant drop from 0.49 in the `h2_normal_model`.
- There is a shift in terms of the pareto-k influence. For example, the `linux` datapoint is not as influential as the `php-src` compared to the influence in the

`h2_normal_model`.

In [ ]:
```
plt.figure(figsize=(10,7))

pareto_k_h3 /= pareto_k_h3.max()
pareto_k_h3_size = 250 * pareto_k_h3
pareto_k_h3_size += abs(min(pareto_k_h3_size))+0.001 #fix error with nega

pareto_k_h3_size_c = []
for i in df[df['language']==7].index:
    pareto_k_h3_size_c.append(pareto_k_h3_size[i])

#Get the 25 data points with the largest pareto_k_values
top_indices = np.argsort(pareto_k_h3)[-25:][::-1]
top_data_points = df.iloc[top_indices]

az.plot_hdi(x_seq, post_pred_B_h2_C, color = 'b', fill_kwargs={'alpha': 0
plt.plot(x_seq, post_pred_B_h2_mean_C, color = 'b', alpha=0.7, label='Mea
az.plot_hdi(x_seq, post_pred_B_h3_C, color = 'g', fill_kwargs={'alpha': 0
plt.plot(x_seq, post_pred_B_h3_mean_C, color = 'g', alpha=0.7, label='Mea

# No. 7 for C
plt.scatter(df[df['language']==7].age_std, df[df['language']==7].bugs_std

#Plotting text for data points with high pareto-k values. We manually adj
for index, row in top_data_points.iterrows():
    dont_annotate = False
    if row['language'] == 7 and row['project'] == 'linux':
        position=(-30,-30)
    elif row['language'] == 7 and row['project'] == 'php-src':
        position=(-30,30)
    elif row['language'] == 7 and row['project'] == 'mysql':
        position=(-45,30)
    elif row['language'] == 7 and row['project'] == 'ruby':
        position=(-30,30)
    else:
        dont_annotate = True
    if dont_annotate == False:
         plt.annotate(f"{languages[row['language']]}: {row['project']} ({
                     textcoords='offset points', arrowprops=dict(arrowsty

plt.legend()
plt.xlim(-1.5,5.0)
#plt.ylim(0,20000)
plt.ylim(-3.0,35.0)
plt.xlabel('Age (std.)')
plt.ylabel('Bugs (std.)')
plt.title("Bugs(std) ~ Age(std)| Language = C");
```



Lastly, we sampled 1127 points from `commits_std` values between -1 to 5 in the language `C` and visualized them along with the actual datapoints to see the effect of commits.

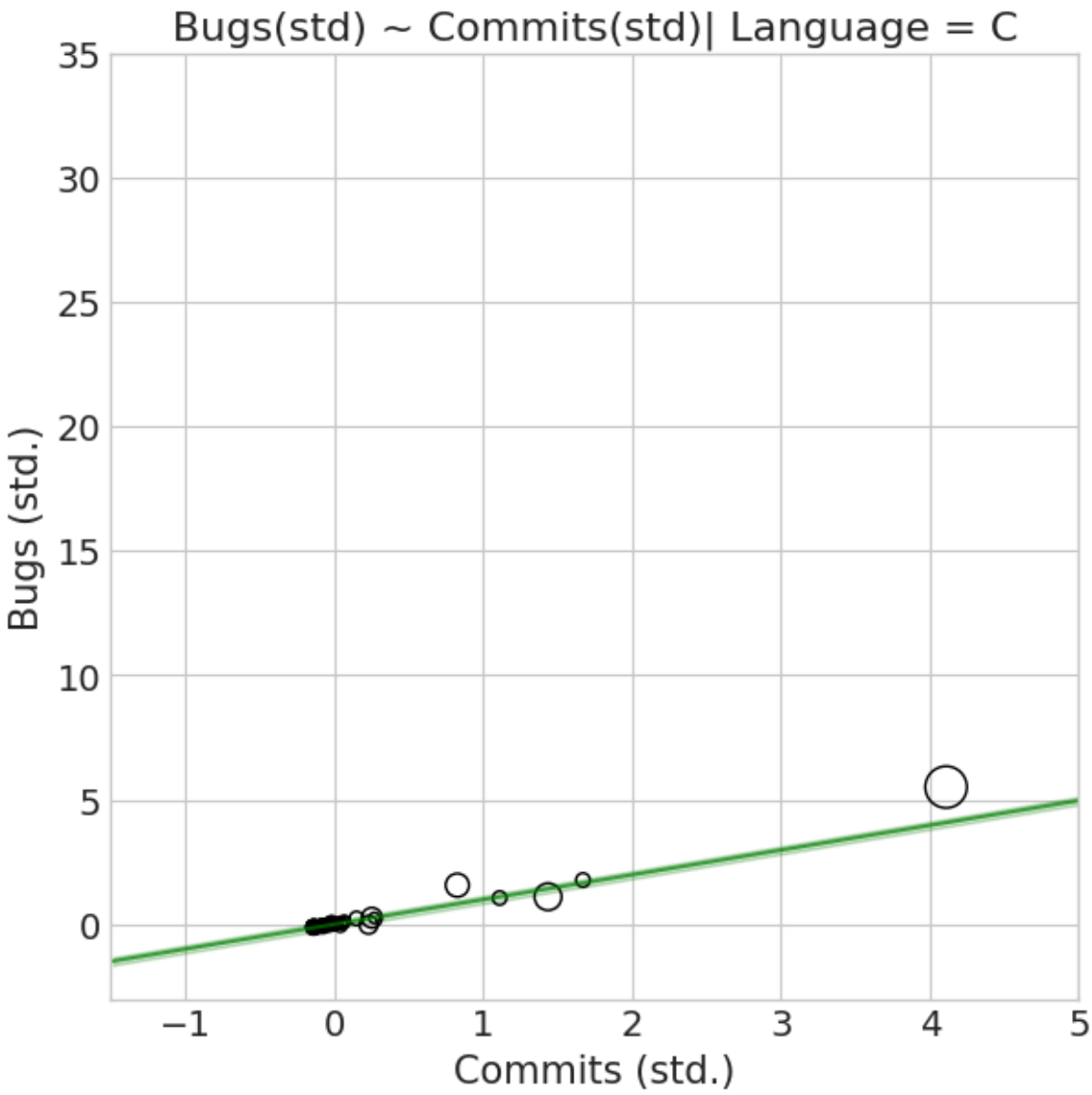
In [ ]:
```
n_points = len(df)
x_seq = np.linspace(-1.5, 5, n_points)

with h3_normal_model:
    pm.set_data({"commits_std": x_seq, "age_std": np.repeat(0, n_points),
    post_pred_B_h3_commits = pm.sample_posterior_predictive(h3_normal_tra

post_pred_B_h3_mean_commits = post_pred_B_h3_commits.mean(["chain", "draw
```

```
Sampling: [B]
```

100.00% [8000/8000 00:00<00:00]

In [ ]:
```
plt.figure(figsize=(6,6))

pareto_k_h3 /= pareto_k_h3.max()
pareto_k_h3_size = 250 * pareto_k_h3
pareto_k_h3_size += abs(min(pareto_k_h3_size))+0.001 #fix error with nega

pareto_k_h3_size_c = []
for i in df[df['language']==7].index:
    pareto_k_h3_size_c.append(pareto_k_h3_size[i])

az.plot_hdi(x_seq, post_pred_B_h3_commits, color = 'g', fill_kwargs={'alp
plt.plot(x_seq, post_pred_B_h3_mean_commits, color = 'g', alpha=0.7)
# No.6 for Perl
plt.scatter(df[df['language']==7].commits_std, df[df['language']==7].bugs
plt.xlim(-1.5,5.0)
plt.ylim(-3.0,35.0)
plt.xlabel('Commits (std.)')
```

```
plt.ylabel('Bugs (std.)')
plt.title("Bugs(std) ~ Commits(std)| Language = C");
```



Based on the plot, we see a strong association between number of commits and the number of bugs in the language `C`.

Conclusion H3

Based on the results we reject **H3**. Conditioned by commits, age experienced a drop in the effect on number of bugs for **all** languages. This means that the association found between age and bugs in `h2_poisson_model` was in fact a spurious relationship actually caused by commits. Although we get the same conclusion for **H3** in `h3_poisson_model`, we do see that the effect of age after conditioning on commits is remarkebly smaller in `h3_normal_model` compared to `h3_poisson_model`. We believe that the bad fit of commits in the `h3_poisson_model` makes the effect of age after conditioning on commits less reliable than in the `h3_normal_model`.

H3

Multilevel Normal Regression

We design a multilevel model for the normal distribution implementation of hypothesis 3.

In hypothesis 3, we examine whether the effect of age (A), conditioned on number of commits (C), on number of bugs (B) is the same as the direct effect of age (A) on number of bugs (B).

In the normal distribution implementation from earlier, we saw that the effect of age drastically reduced when adding commits as a predictor. This effect was consistent for all languages and many effects were close to 0.

In this context we think it is reasonable to introduce to our model that the effect of age on bugs in one language can inform the effect of age in other languages.

In the multilevel model, we expect to see a shrinkage effect, meaning that the effect of age for each language will be closer to the overall mean of age effect.

We create a multilevel model for hypothesis 3 with a normal distribution implementation which is similar `h3_normal_model`.

The difference is that the multilevel model is extended with two additional population level parameters (for the population of languages):

- $\bar{\beta}$ - effect of age across languages
- γ - dispersion in effect of age across languages

In this context, we note that the `h3_normal_model` does not include any population level parameters for the population of languages and can therefore be considered a no-pooling model.

We model the number of standardized bugs B with a Normal distribution.

$$B_i \sim \mathcal{N}(\mu_i, \sigma)$$

$$\mu_i = \alpha + \beta_{A[L]} A_i + \beta_{C[L]} C_i$$

We re-use the priors for the effect of commits, alpha, and sigma that we determined earlier in `h3_normal_model`.

$$\beta_{C[L]} \sim N(1, 0.2)$$

$$\alpha \sim N(0, 0.5)$$

$$\sigma \sim \text{Exponential}(1)$$

The difference from the previous model is, that information learned from a $\beta_{A[L]}$ in one language informs the others through the shared hyper-parameters $\bar{\beta}$ and γ.

$$\beta_{A[L]} \sim N(\bar{\beta}, \gamma)$$

We define the hyperpriors:

$$\bar{\beta} \sim N(0, 0.2)$$

$$\gamma \sim \text{Exponential}(1)$$

In [ ]:
```
with pm.Model() as h3_ml_model:

    #Hyperpriors
    beta_bar = pm.Normal('beta_bar', 0, 0.2)
    beta_sigma = pm.Exponential('beta_sigma', 1)

    alpha = pm.Normal("alpha", 0, 0.5)
    beta_commits = pm.Normal("beta_commits", 1, 0.2, shape=languages.size
    beta_age = pm.Normal("beta_age", beta_bar, beta_sigma, shape=language

    sigma = pm.Exponential("sigma", 1)

    #We add language_ids and age_std as mutable data so we can change it
    language_ids = pm.Data('language_ids', df.language, mutable=True)
    age_std = pm.Data("age_std", df.age_std, mutable=True)
    commits_std = pm.Data("commits_std", df.commits_std, mutable=True)

    mu = pm.Deterministic("mu",
                          alpha + beta_age[language_ids] * age_std + be

    # B = Number of Bugs Standardized
    B = pm.Normal("B", mu = mu, sigma = sigma, observed = df.bugs_std.val

    h3_ml_trace = pm.sample(2000, tune=1000, idata_kwargs={'log_likelihoo
```

```
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [beta_bar, beta_sigma, alpha, beta_commits, beta_age, sigma]
```

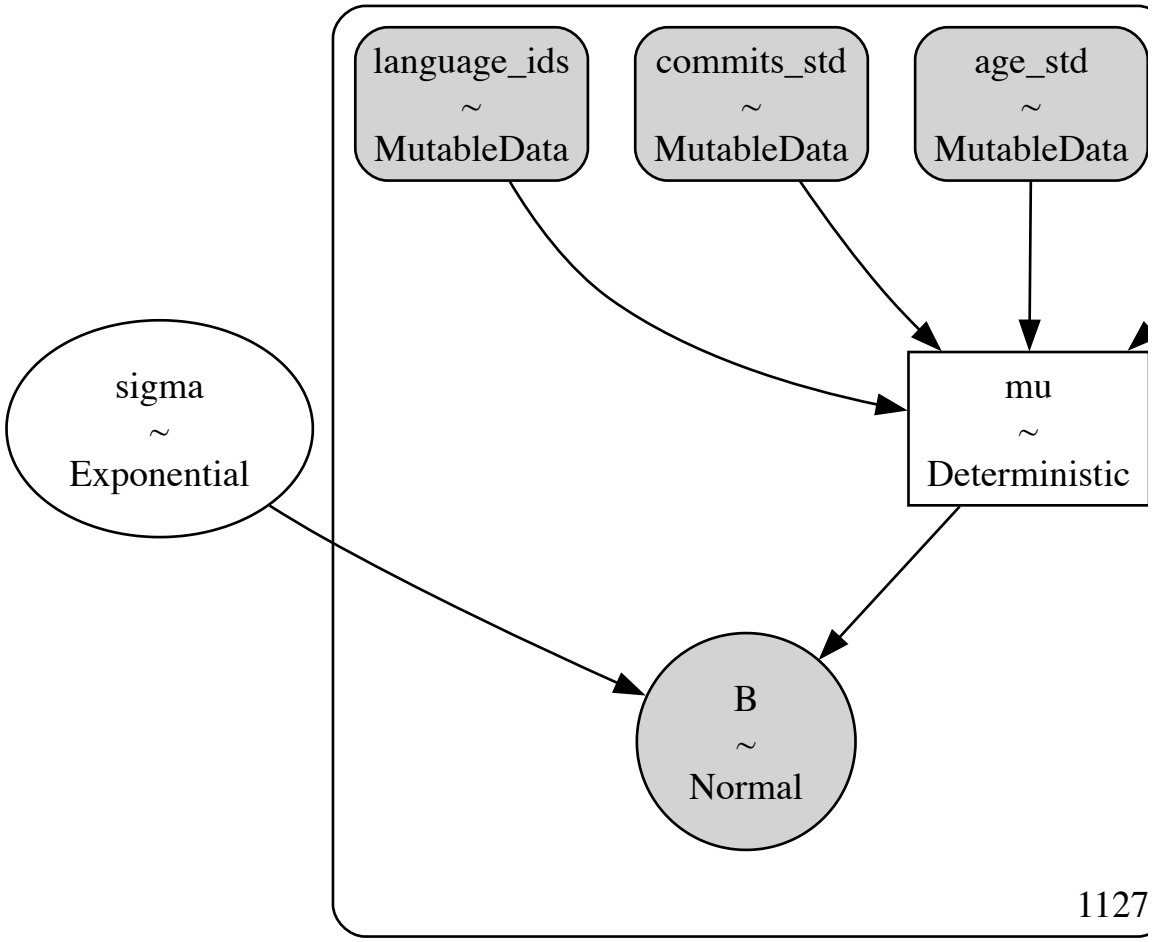
100.00% [12000/12000 00:13<00:00 Sampling 4 chains, 0 divergences]

```
Sampling 4 chains for 1_000 tune and 2_000 draw iterations (4_000 + 8_000
draws total) took 14 seconds.
```

In the diagram of the model parameters we note that `beta_age` now has has two hyper-parameters `beta_bar` and `beta_sigma`. Besides that, the model is the same as `h3_normal_model`.

In [ ]:
```
pm.model_to_graphviz(h3_ml_model)
```

Out[ ]:



Below is the trace convergence of the parameters .

The effective sample sizes (`ess_bulk` , `ess_tail`) are high which indicates that the samples both in the bulk and in the tails of the posteriors have low-autocorrelation.

We also note that the Monte Carlo Standard Error `msce_mean` and `msce_sd` are 0.0, which also indicates good accuracy in the chains.

In addition, both the rhat values `r_hat` are 1.0, which means that the 4 chains have a good mixing. This is also reflected in the plot below, which shows good mixing of the chains.

Multilevel models often have trouble to converge, but that is not the case for this model.

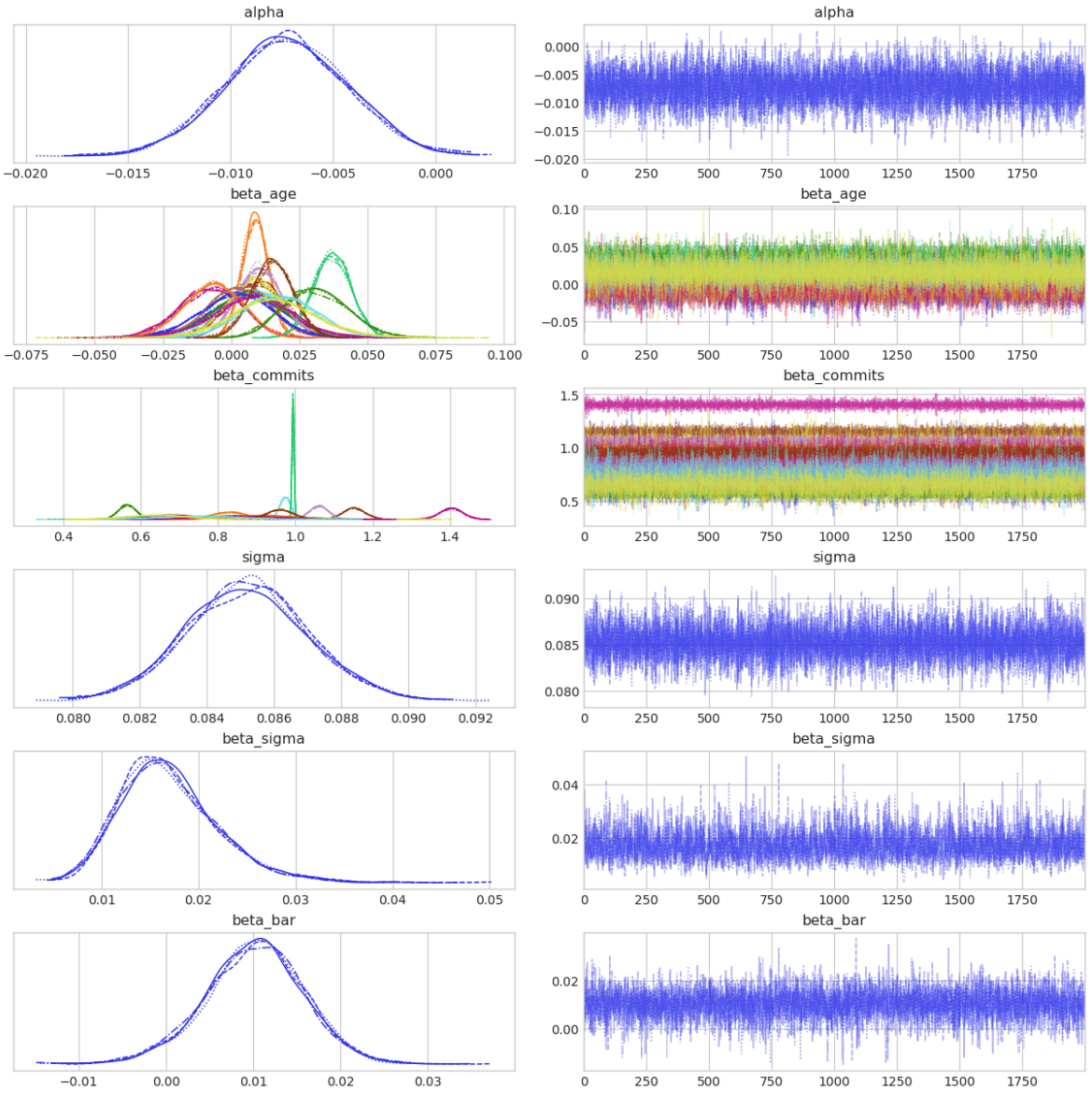
In [ ]:
```
pm.summary(h3_ml_trace, var_names=['alpha', 'beta_age', 'beta_commits', '
       'ess_tail', 'r_hat']]
```

Out[ ]:

	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
alpha	0.0	0.0	11564.63	6512.48	1.0
beta_age[0]	0.0	0.0	8462.80	5079.49	1.0
beta_age[1]	0.0	0.0	12729.39	6150.59	1.0
beta_age[2]	0.0	0.0	10267.00	6401.06	1.0
beta_age[3]	0.0	0.0	7368.93	5504.37	1.0
beta_age[4]	0.0	0.0	9220.76	6785.37	1.0
beta_age[5]	0.0	0.0	7981.55	5953.60	1.0
beta_age[6]	0.0	0.0	11104.68	5656.29	1.0
beta_age[7]	0.0	0.0	8663.70	5505.31	1.0
beta_age[8]	0.0	0.0	10001.00	6445.21	1.0
beta_age[9]	0.0	0.0	9973.21	5926.84	1.0
beta_age[10]	0.0	0.0	7069.68	6228.19	1.0
beta_age[11]	0.0	0.0	8450.74	6229.18	1.0
beta_age[12]	0.0	0.0	7644.30	6605.96	1.0
beta_age[13]	0.0	0.0	7616.15	5908.15	1.0
beta_age[14]	0.0	0.0	12627.47	6430.66	1.0
beta_age[15]	0.0	0.0	9363.93	7034.50	1.0
beta_age[16]	0.0	0.0	9770.96	6203.72	1.0
beta_commits[0]	0.0	0.0	11337.43	6286.35	1.0
beta_commits[1]	0.0	0.0	10267.49	6120.72	1.0
beta_commits[2]	0.0	0.0	13767.85	6137.49	1.0
beta_commits[3]	0.0	0.0	16495.15	5265.67	1.0
beta_commits[4]	0.0	0.0	10831.10	6448.38	1.0
beta_commits[5]	0.0	0.0	9118.48	6559.00	1.0
beta_commits[6]	0.0	0.0	12727.67	5821.95	1.0
beta_commits[7]	0.0	0.0	13129.94	6256.40	1.0
beta_commits[8]	0.0	0.0	11369.49	5831.97	1.0
beta_commits[9]	0.0	0.0	12771.69	5359.52	1.0
beta_commits[10]	0.0	0.0	8305.78	5992.85	1.0
beta_commits[11]	0.0	0.0	12635.80	5007.20	1.0
beta_commits[12]	0.0	0.0	10241.07	6243.00	1.0
beta_commits[13]	0.0	0.0	9805.27	6610.09	1.0
beta_commits[14]	0.0	0.0	12570.72	5590.70	1.0
beta_commits[15]	0.0	0.0	10401.09	6453.47	1.0

	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
beta_commits[16]	0.0	0.0	14841.17	6329.17	1.0
sigma	0.0	0.0	15569.15	5460.04	1.0
beta_sigma	0.0	0.0	3358.72	3831.76	1.0
beta_bar	0.0	0.0	5441.23	5058.25	1.0

In [ ]:
```
az.plot_trace(h3_ml_trace, var_names=['alpha', 'beta_age', 'beta_commits'
plt.show()
```



In [ ]:
```
pm.summary(h3_ml_trace, var_names=['alpha', 'beta_age', 'beta_commits', '
```

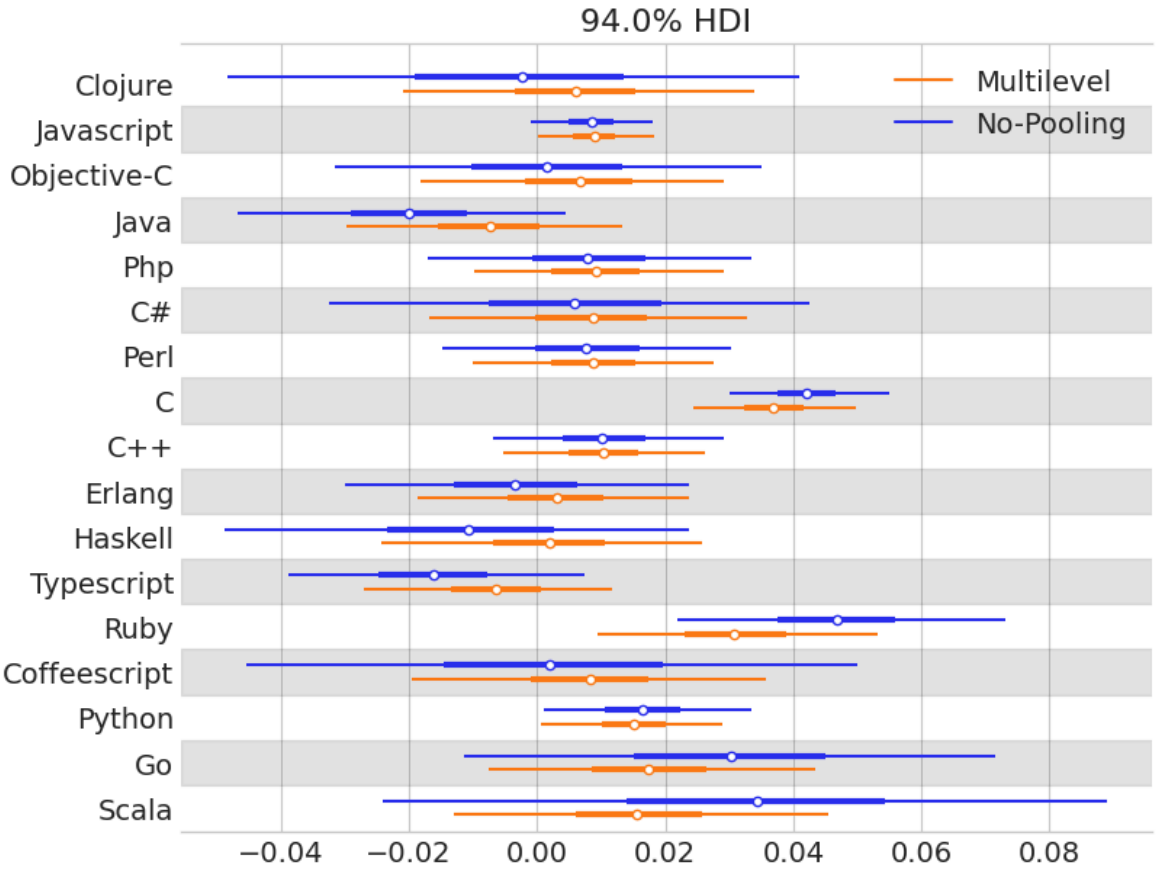
Out[ ]:

	mean	sd	hdi_3%	hdi_97%
alpha	-0.01	0.00	-0.01	-0.00
beta_age[0]	0.01	0.01	-0.02	0.03
beta_age[1]	0.01	0.00	0.00	0.02
beta_age[2]	0.01	0.01	-0.02	0.03
beta_age[3]	-0.01	0.01	-0.03	0.01
beta_age[4]	0.01	0.01	-0.01	0.03
beta_age[5]	0.01	0.01	-0.02	0.03
beta_age[6]	0.01	0.01	-0.01	0.03
beta_age[7]	0.04	0.01	0.02	0.05
beta_age[8]	0.01	0.01	-0.01	0.03
beta_age[9]	0.00	0.01	-0.02	0.02
beta_age[10]	0.00	0.01	-0.02	0.03
beta_age[11]	-0.01	0.01	-0.03	0.01
beta_age[12]	0.03	0.01	0.01	0.05
beta_age[13]	0.01	0.01	-0.02	0.04
beta_age[14]	0.02	0.01	0.00	0.03
beta_age[15]	0.02	0.01	-0.01	0.04
beta_age[16]	0.02	0.02	-0.01	0.05
beta_commits[0]	0.91	0.11	0.71	1.11
beta_commits[1]	0.83	0.05	0.74	0.91
beta_commits[2]	0.85	0.09	0.68	1.02
beta_commits[3]	1.40	0.03	1.35	1.46
beta_commits[4]	1.15	0.03	1.10	1.21
beta_commits[5]	0.97	0.01	0.95	1.00
beta_commits[6]	0.93	0.12	0.71	1.14
beta_commits[7]	0.99	0.00	0.99	1.00
beta_commits[8]	1.06	0.02	1.02	1.11
beta_commits[9]	0.67	0.08	0.52	0.81
beta_commits[10]	0.68	0.09	0.52	0.85
beta_commits[11]	0.91	0.08	0.75	1.06
beta_commits[12]	0.57	0.02	0.52	0.61
beta_commits[13]	0.86	0.10	0.66	1.05
beta_commits[14]	0.96	0.03	0.90	1.03
beta_commits[15]	0.75	0.10	0.55	0.93

	mean	sd	hdi_3%	hdi_97%
beta_commits[16]	0.64	0.08	0.50	0.79
sigma	0.09	0.00	0.08	0.09
beta_sigma	0.02	0.01	0.01	0.03
beta_bar	0.01	0.01	-0.00	0.02

Next we compare the posterior estimates for `beta_age` in `h3_normal_model` (i.e. the no-pooling model) and in the multilevel model `h3_ml_model`.

In [ ]:
```
_,ax = plt.subplots(figsize=(8,6))
az.plot_forest([h3_normal_trace,h3_ml_trace], var_names=["beta_age"], mod
forest_languages = [int(i.get_text()[1:-1]) if len(i.get_text()) < 6 else
ax.set_yticklabels(languages.take(forest_languages));
```



We see a notable effects of including population level parameters in the mulilevel model compared to the no-pooling model.

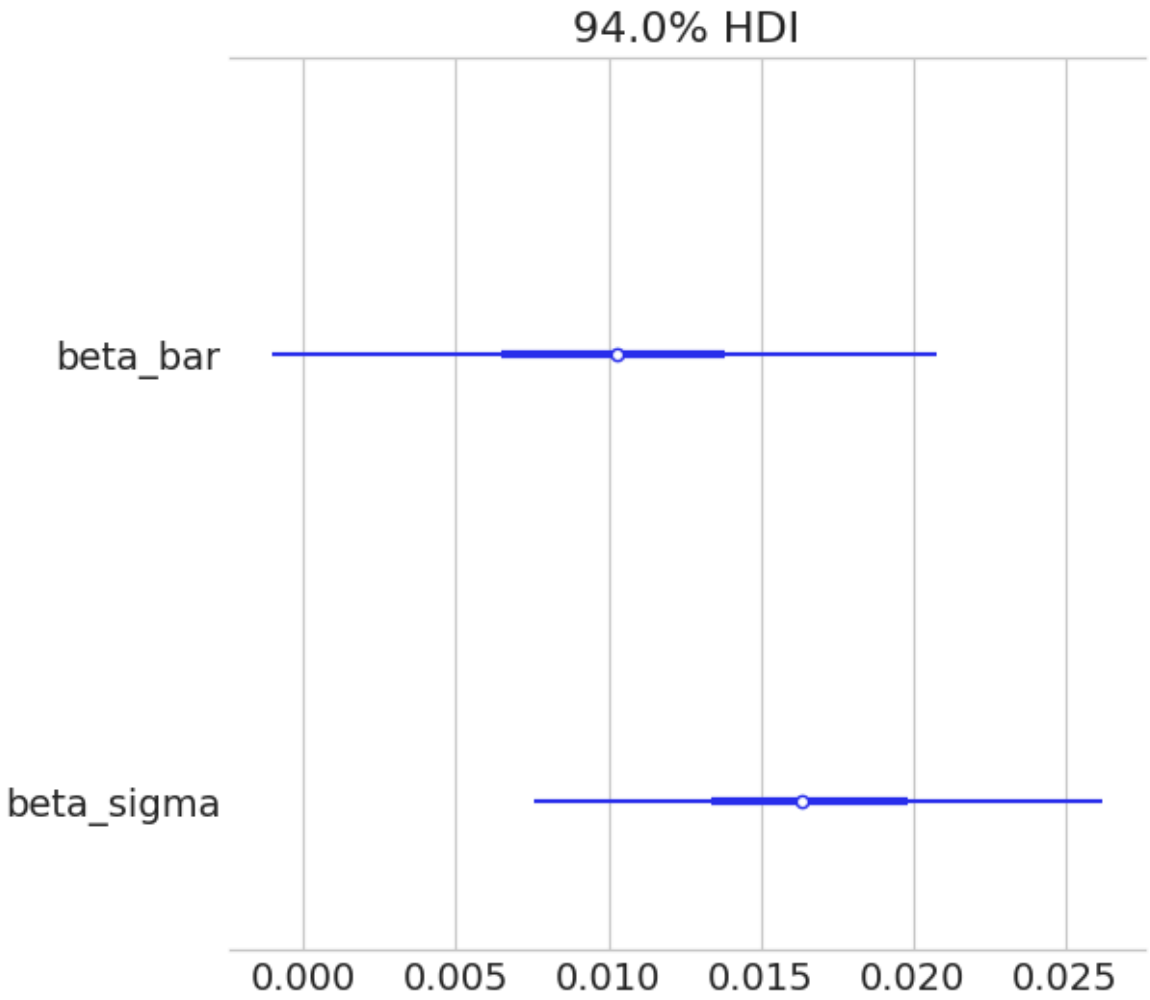
- Multilevel estimates are closer to 0
- Multilevel estimates generally have smaller HDIs

These are desirable effects of the multilevel model that regularizes the estimates based on the observations.

Below is the forest plot for $\bar{\beta}$ and γ. The effect of age across all languages $\bar{\beta}$ just reaches 0 at the 94% HDI but is still very small. The dispersion in the effect of age

across all languages γ is also small meaning that there is some amount of shrinkage effect.

In [ ]:
```
az.plot_forest(h3_ml_trace, var_names = ['beta_bar', 'beta_sigma'], combi
```

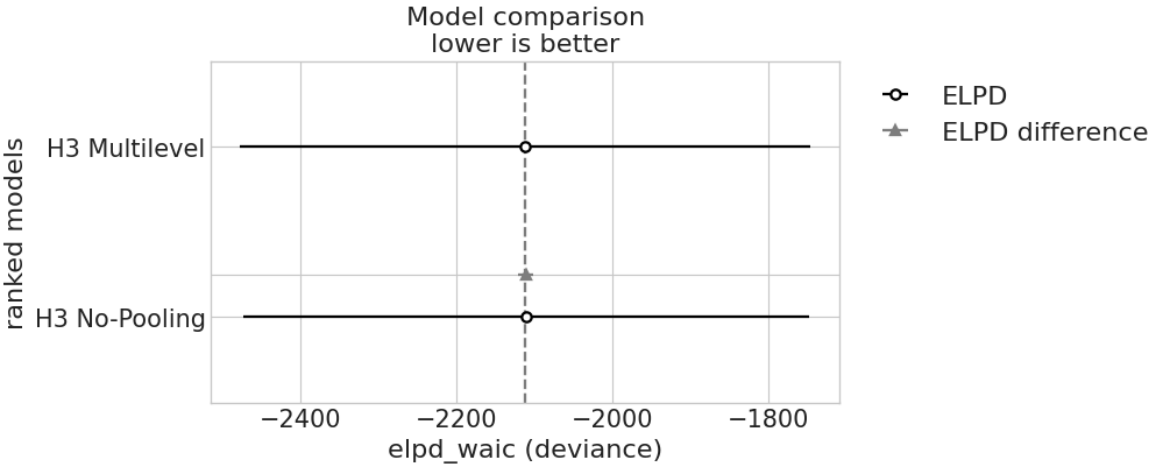


Lastly we compare the no-pooling model and the multilevel model.

Note the small reduction of `p_waic` value for the `h3_ml_model` compared to the `h3_normal_model`. This reduction is due to the effect of population level parameters $\bar{\beta}$ and γ, and is a beneficial consequence of using a multilevel model.

In addition, the multilevel model has a slightly smaller `elpd_waic` value, but the overlap is very large meaning we cannot state which model to prefer based on the information criteria.

In [ ]:
```
comp = pm.compare({'H3 Multilevel': h3_ml_trace,
            'H3 No-Pooling': h3_normal_trace},
            ic='waic', scale='deviance')

pm.plot_compare(comp, insample_dev=False, figsize=(10,4))
plt.show()
```



In [ ]:
```
comp
```

Out[ ]:

	rank	elpd_waic	p_waic	elpd_diff	weight	se	
H3 Multilevel	0	-2111.452354	191.049643	0.000000	0.609642	365.990745	0.000
H3 No-Pooling	1	-2110.474243	188.420014	0.978111	0.390358	362.952987	9.369

Overall Conclusion

Based on the results from `h1_poisson_model` and `h1_binomial_model`, we reject **H1**. `Haskell` does **neither** give the highest probability to the lowest number of bugs **nor** does it give the smallest probability to a bug given a commit. This means that `Haskell` is *not* less prone to contain less bugs than any other programming language in the dataset. We believe that future work could include the variable `project_type` as a predictor. This would ensure that the effect of number of bugs for a language is not caused by this language being consistently used in project types with a high number of bugs.

In regards to **H2** we found varying results. When modeling the impact of age on bugs *across* all languages in `h2_poisson_model`, we found that projects/language combinations of old age was confidently estimated to have a larger number of bugs. In contrast, when modelling the impact of age on bugs seperately for each language in `h2_normal_model`, we found that age had a reliable positive effect on the number of bugs for only two out 17 languages.

Based on the results from `h3_poisson_model` and `h3_normal_model` we reject **H3**. In both models we found that the direct/unconditioned effect of age on bugs was larger than the same effect when including commits as a confounding predictor. This means that the effect of age on bugs found in **H2** was in fact a spurious relationship caused by commits. Nonetheless, we did see that the reduction in the age effect after conditioning on commits was remarkebly larger in `h3_normal_model` compared to `h3_poisson_model`. We believe this to be a consequence of the poor

fit of commits in the `h3_poisson_model` which made the reduction in age effect less reliable compared to the reduced effect in the `h3_normal_model`.