In [ ]:
```
import stan

import nest_asyncio # https://pystan.readthedocs.io/en/latest/faq.html
nest_asyncio.apply()
```

In [ ]:
```
import warnings
warnings.filterwarnings('once')
```

In [ ]:
```
# reload packages from notebook whenever needed
%load_ext autoreload
%autoreload 2
```

In [ ]:
```
%matplotlib inline

import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt

import arviz as az # For visualization and loo
import seaborn as sns
```

In this homework, you're going to go through an exercise in going from a problem description, to writing down a statistical model, to fitting it both using raw Python and in Stan

**Model**: We have two independent variables (covariates), $x1$ and $x2$, and two unknown latent parameters $\alpha$ and $\beta$. The independent data $y$ is generated as follows. First, a coin is flipped and if it is heads, then $y = 0$. Otherwise, $y$ is drawn according to a Poisson random variable. This is typically called a "zero-inflated Poisson" model.

The coin flip is heads with probability $logit^{-1}(\alpha x_1)$. The Poisson random variable has rate $e^{\beta x_2}$.

Our prior is that $\alpha, \beta$ are both drawn from a uniform random variable between -1 and 1. The covariates are both drawn from a Normal distribution with mean 0 and variance 1.

# Conceptual: Describe a research setting that conceivably corresponds to this data generating process

In [ ]:
```

```

# Write the statistical model down in math.

In other words, what are the distributions from which $x_1, x_2, \alpha, \beta, y$ are sampled?

Either using LaTeX within markdown, or inserting an image with math into the markdown.

# Generate data from the true model, given fixed parameters $\alpha, \beta$

In [ ]:
```
from scipy import stats
from scipy.stats import bernoulli, norm, poisson

def inv_logit(v):
    return 1/(1+np.exp(-v))

# TODO finish this function
# the output should be a dataframe with columns x1, x2, y
def generate_data(theta, N):
    alpha, beta = theta #as in lecture, theta = (alpha, beta)
    pass
```

In [ ]:
```
def plot_generated_data(df):
    sns.histplot(df.y)
    plt.yscale('log')
    plt.show()

    cuts = pd.DataFrame({str(feature) + 'Bin' : pd.qcut(df[feature], 5) for feature
    dfplot = pd.concat([df, cuts], axis=1)[['y', 'x1Bin', 'x2Bin']].groupby(['x1Bin
    dfpivot = dfplot.pivot(index='x1Bin', columns='x2Bin', values='y')
    plt.clf()
    sns.heatmap(dfpivot)
    plt.title('Means of y vs Features x1 and x2')
    plt.tight_layout()
    plt.show()
```

The below will plot the data once you have filled out the above function

In [ ]:
```
alpha, beta = (1, 1)
df = generate_data((alpha, beta), 1000)
plot_generated_data(df)
```

In [ ]:
```
alpha, beta = (.5, -1)
df = generate_data((alpha, beta), 1000)
plot_generated_data(df)
```

# Calculate posterior using your own code

As we did in Lecture 2 and Lecture 4

In [ ]:
```
```

In [ ]:
```
# Prior for alpha, beta from the statistical model

#TODO finish this function for the pdf of theta. Hint: what is the area of the rect
def p_theta(theta):
    pass
```

In [ ]:
```
from scipy.stats import norm

# TODO finish this function for P({y} | theta, {x1}, {x2})
# Bonus: do this in log space to avoid underflow
def p_data_given_theta(theta, data, do_log = False):
    alpha, beta = theta
    ys = data['y']

    #TODO finish this function for P(y | theta, x1, x2)
    # Hint: there are 2 cases, y_i = 0 and y_i > 0
    # Hint: you'll use the inv_logit function defined above
    # Hint: you'll use the poisson distribution from scipy.stats, and the pmf metho
    def single_y_likelihood(y, x1, x2):
        pass

    vals = [single_y_likelihood(y, x1, x2) for y, x1, x2 in zip(ys, data['x1'], dat
    df['vals'] = vals

    # TODO finish below
    if not do_log:
        pass
    else:
        pass

# TODO finish this function for P(y | x1, x2)
# bonus: do this in log space to avoid underflow
def p_data(data, possible_theta_values, do_log = False):
    """
    This function is the marginal likelihood of the data.
    It is the integral of the likelihood function over possible values of theta, we
    """
    if not do_log:
        pass
    else:
        pass
```

Grid sampling

In [ ]:
```
# TODO finish this function to sample possible_theta_values, p_theta_given_data usi
# bonus: do this in log space to avoid underflow
def grid_sampling(data, gridnum = 1000, do_log = False):
    possible_alpha_values = np.linspace(-1, 1, gridnum)
    possible_beta_values = np.linspace(-1, 1, gridnum)
    possible_theta_values = [(alpha, beta) for alpha in possible_alpha_values for b

    #TODO finish
    p_theta_given_data = None

    return possible_theta_values, p_theta_given_data
```

In [ ]:
```
true_alpha, true_beta = (.25, .5)
df = generate_data((true_alpha, true_beta), 100)
```

Below will plot posterior for alpha, beta using grid sampling

In [ ]:
```
possible_theta_values, p_theta_given_data = grid_sampling(df, gridnum = 10)
dfplot = pd.DataFrame(possible_theta_values, columns=['alpha', 'beta'])
dfplot['p_theta_given_data'] = p_theta_given_data
from matplotlib.colors import LogNorm, Normalize
sns.heatmap(dfplot.pivot('alpha', 'beta', 'p_theta_given_data'))
```

Bonus: Below will plot log posterior for alpha, beta using grid sampling in log space.

In [ ]:
```
possible_theta_values, p_theta_given_data = grid_sampling(df, gridnum = 10, do_log=
dfplot = pd.DataFrame(possible_theta_values, columns=['alpha', 'beta'])
dfplot['log_p_theta_given_data'] = p_theta_given_data
sns.heatmap(dfplot.pivot('alpha', 'beta', 'log_p_theta_given_data'))
plt.title('log_p_theta_given_data')
```

In log space, can do with larger datasets without underflow (though P(data) is still troublesome...)

In [ ]:
```
true_alpha, true_beta = (.25, .5)
df = generate_data((true_alpha, true_beta), 500)
possible_theta_values, p_theta_given_data = grid_sampling(df, gridnum = 10, do_log=
dfplot = pd.DataFrame(possible_theta_values, columns=['alpha', 'beta'])
dfplot['log_p_theta_given_data'] = p_theta_given_data
sns.heatmap(dfplot.pivot('alpha', 'beta', 'log_p_theta_given_data'))
plt.title('log_p_theta_given_data')
```

With more data points, we get much tighter posteriors (observe the color scale)

# Now, fit a Stan model for this dataset

In [ ]:
```
true_alpha, true_beta = (.25, .5)
df = generate_data((true_alpha, true_beta), 1000)
```

## First, just fit a Poisson regression, ignoring the zero inflation

First, we'll fit a "mis-specified" model -- suppose we just fit a Poisson regression, i.e.:

$$y \sim Poisson(e^{intercept + \alpha * x_1 + \beta * x_2})$$

The following will be useful: https://mc-stan.org/docs/functions-reference/poisson-log-glm.html. You'll likely need to directly increment the `target` parameter.

In [ ]:
```

```

In [ ]:
```
stan_data = {'N': df.shape[0], 'y': df.y.values, 'X': df[['x1', 'x2']].values}
```

TODO: write a Stan model and save it in poissonregression.stan

In [ ]:
```
stan_folder = './'
stan_file = stan_folder + 'poissonregression.stan'
with open(stan_file) as file:
    model_code = file.read()
print(model_code)
```

In [ ]:
```
compiled_model = stan.build(model_code, data=stan_data)
```

In [ ]:
```
fit = compiled_model.sample(num_chains=2, num_warmup = 100, num_samples= 500)
```

We wrote some of the code for you to analyze the model fit

In [ ]:
```
idata = az.from_pystan(posterior=fit, posterior_model=compiled_model)
```

In [ ]:
```
summary = az.summary(fit)
summary
```

In [ ]:
```
az.plot_posterior(idata)
```

In [ ]:
```
az.plot_trace(fit, compact=False, legend=True)
```

In [ ]:
```
az.plot_pair(idata)
```

In [ ]:
```
# TODO: In your poissonregression.stan file, use the generated quantities block to
# from the posterior predictive distribution and plot the posterior predictive dist
# Display it in the same plot as the true $y$ data, and comment on comparing the tw
```

In [ ]:
```

```

In [ ]:
```
# TODO bonus: use arviz to plot the posterior predictive distribution
```

In [ ]:
```

```

# Now, fit a correct Stan model reflecting the true data generating process

Now, fit a zero inflated Poisson in stan.

Hint: you'll need to directly increment the `target` parameter.

Hint: you'll likely need the following functions: `log_sum_exp`, `bernoulli_logit_lpmf`, `poisson_log_glm_lpmf`

Hint: I had to play around a bit with the input parameter formatting (e.g., put x2 inside its own matrix)

In [ ]:
```
stan_data = {}
```

In [ ]:
```
stan_folder = './'
stan_file = stan_folder + 'zeroinflatedpoisson.stan'
with open(stan_file) as file:
    model_code = file.read()
print(model_code)
```

In [ ]:
```
compiled_model = stan.build(model_code, data=stan_data)
```

In [ ]:
```
fit = compiled_model.sample(num_chains=2, num_warmup = 100, num_samples= 500)
```

In [ ]:
```
idata = az.from_pystan(posterior=fit, posterior_model=compiled_model)
```

In [ ]:
```
summary = az.summary(fit)
summary
```

In [ ]:
```
az.plot_posterior(idata)
```

In [ ]:
```
az.plot_trace(fit, compact=False, legend=True)
```

In [ ]:
```
az.plot_pair(idata)
```

In [ ]:
```
# TODO: As before, In your stan file, use the generated quantities block to sample
# from the posterior predictive distribution and plot the posterior predictive dist
# Display it in the same plot as the true $y$ data, and comment on comparing the tw
```

In [ ]:
```

```