Homework 3

Lauren Bassett
DS 6040

Honor Pledge: On my honor, I pledge that I have neither given nor recieved help on this assignment.

Part 1: Changepoint detection and samplers (50 points)

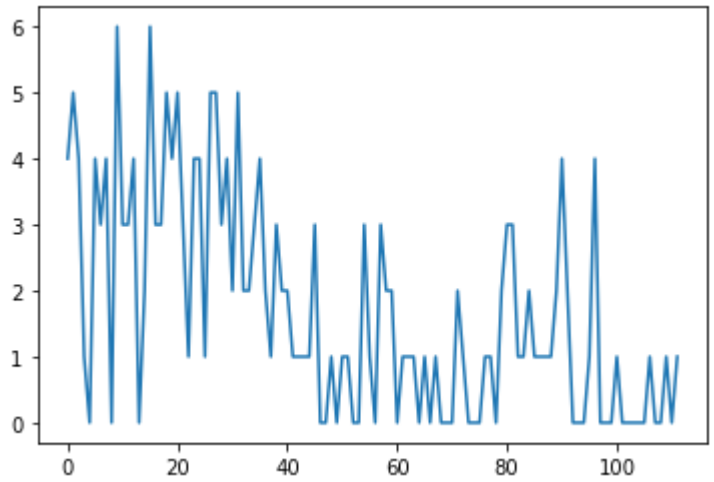
1. (50 points) With the above information, complete the Gibbs sampler in the accompanying notebook. You should only need to complete the update steps for the μ and λ (called lambdap in the notebook) parameters. Run the Gibbs sampler, plot the posterior densities and calculate the EAP estimates with 95% (equal tailed) credible intervals for μ and λ. Provide the top 5 most probable values of k.

In [ ]:
```
from scipy.stats import gamma, norm, poisson
from scipy.special import logsumexp
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
```

In [ ]:
```
#Import dataset
coal_dat = pd.read_csv("coaldisasters-ds6040.csv")
coal_dat['Count'].plot()
coal_dat.head()
```

Out[ ]:

	Year	Count
0	1851	4
1	1852	5
2	1853	4
3	1854	1
4	1855	0



In [ ]:
```
def log_like(x_data, mu, lamb, k):
    n = len(x_data)
    first_chunks = poisson.logpmf(x_data[0:k], mu)
    second_chunks = poisson.logpmf(x_data[(k):n], lamb)
    return np.sum(first_chunks) + np.sum(second_chunks)

# example call
log_like(np.random.choice(4, size=3), 1, 2, 3)
```

Out[ ]:
```
-4.386294361119891
```

In [ ]:
```
# example sampling from a gamma
# make sure you use the correct arguments
rate = 10
gamma.rvs(2, scale = 1/rate)
```

Out[ ]:
```
0.42837537845854384
```

In [ ]:
```
def samp_k(x_data, mu, lamb):
    n = len(x_data)
    possible_ks = np.arange(n)

    log_unnorm_weights = [log_like(x_data, mu, lamb, k) for k in possible_ks]
#get array of loglikelihood for each possible k

    log_denominator = logsumexp(log_unnorm_weights)
    log_norm_weights = log_unnorm_weights - log_denominator

    return np.random.choice(n, size=1, p=np.exp(log_norm_weights))[0]

samp_k(coal_dat['Count'], 2, 2)
```

Out[ ]: 29

In [ ]:

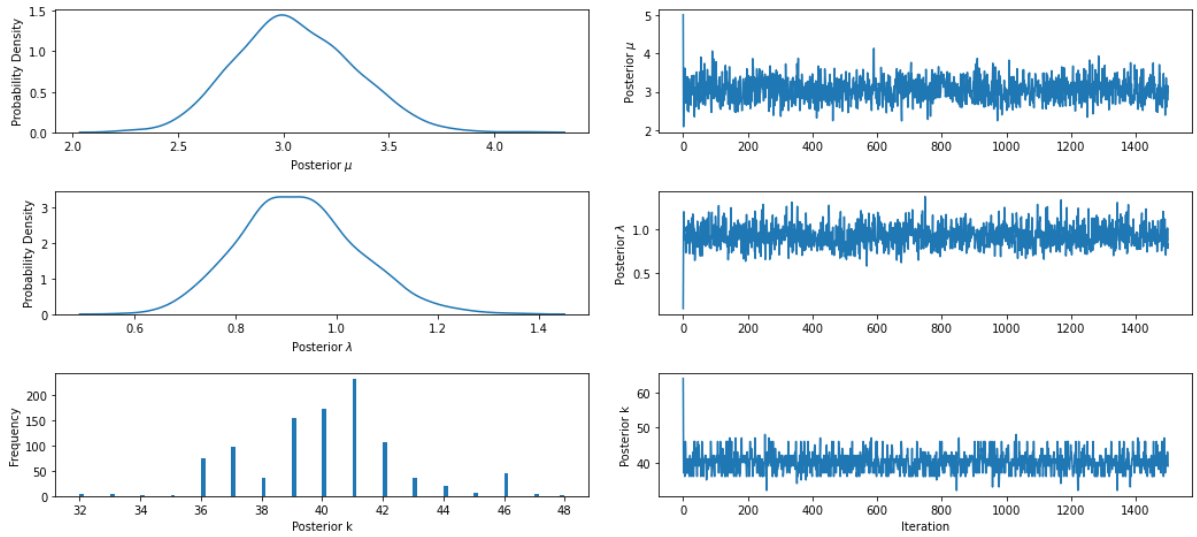
```
class CoalDisasterModel():
    """Gibbs Sampler"""
    def __init__(self, data, a_mu, b_mu, a_lambda, b_lambda,
                 start_mu = 5, start_lambda = .1, start_k = 64, iter_num = 100
0, burnin = 500):

        #Read in data and priors, and make them contained class variables
        self.data = data
        self.a_mu = a_mu
        self.b_mu = b_mu
        self.a_lambda = a_lambda
        self.b_lambda = b_lambda
        self.iter_num = iter_num
        self.burnin = burnin

        #Initalize sampling containers
        self.mu = np.zeros(iter_num+burnin+1)
        self.lambdap = np.zeros(iter_num+burnin+1)
        self.k = np.zeros(iter_num+burnin+1)

        #Put in starting values
        self.mu[0] = start_mu
        self.lambdap[0] = start_lambda
        self.k[0] = start_k

        for i in np.arange(iter_num+burnin):
            if i % 100 == 0:
                print("Iteration " + str(i))


            #This is where you modify the sampler
            self.mu[i+1] =  gamma.rvs(a_mu+sum(data[:int(self.k[i])]),scale=1/
(self.k[i]+b_mu))
            self.lambdap[i+1] = gamma.rvs(a_lambda + sum(data[int(self.k[i
]):]),scale=1/( len(data)- self.k[i]+b_lambda))
            self.k[i+1] = samp_k(self.data, self.mu[i+1], self.lambdap[i+1])

    def plot_posteriors(self):
        f, axs = plt.subplots(3,2, figsize = (15, 7))
        f.tight_layout(pad = 3)
        sns.kdeplot(self.mu[self.burnin:],ax =axs[0,0])
        axs[0,0].set_xlabel(r'Posterior $\mu$')
        axs[0,0].set_ylabel('Probability Density')
        sns.kdeplot(self.lambdap[self.burnin:], ax =axs[1,0])
        axs[1,0].set_xlabel(r'Posterior $\lambda$')
        axs[1,0].set_ylabel('Probability Density')
        axs[2,0].hist(self.k[self.burnin:], bins = 112)
        axs[2,0].set_xlabel(r'Posterior k')
        axs[2,0].set_ylabel('Frequency')
        axs[0,1].plot(np.arange(self.iter_num+self.burnin+1),self.mu, '-')
        axs[0,1].set_xlabel('')
        axs[0,1].set_ylabel(r'Posterior $\mu$')
        axs[1,1].plot(np.arange(self.iter_num+self.burnin+1),self.lambdap,'-')
        axs[1,1].set_xlabel('')
        axs[1,1].set_ylabel(r'Posterior $\lambda$')
```

```
        axs[2,1].plot(np.arange(self.iter_num+self.burnin+1),self.k,'-')
        axs[2,1].set_xlabel('Iteration')
        axs[2,1].set_ylabel('Posterior k')

    def get_rate_estimates(self):
        to_return =pd.DataFrame(columns = ['Posterior EAP', 'Posterior Cred In
terval Lower', 'Posterior Cred Interval Upper'])
        to_return.loc[r'mu', 'Posterior EAP'] = self.mu[self.burnin:].mean()
        to_return.loc[r'lambda', 'Posterior EAP'] = self.lambdap[self.burnin:]
.mean()
        to_return.loc[r'mu', 'Posterior Cred Interval Lower'] = np.quantile(se
lf.mu[self.burnin:],.025)
        to_return.loc[r'lambda', 'Posterior Cred Interval Lower'] = np.quantil
e(self.lambdap[self.burnin:],.025)
        to_return.loc[r'mu', 'Posterior Cred Interval Upper'] = np.quantile(se
lf.mu[self.burnin:],.975)
        to_return.loc[r'lambda', 'Posterior Cred Interval Upper'] = np.quantil
e(self.lambdap[self.burnin:],.975)
        return to_return

    def get_k_probs(self):
        freq = np.bincount(abs(self.k.astype('int'))[self.burnin:])
        ii = np.nonzero(freq)[0]
        freq = np.vstack((ii,freq[ii])).T
        freq = freq.astype("float16")
        freq[:,1] = freq[:,1]/(freq[:,1].sum())
        freq = pd.DataFrame(freq, columns=['k', 'Probability of k'])

        return freq
```

In [ ]:
```
test = CoalDisasterModel(coal_dat['Count'], 1,1,1,1)
```

```
Iteration 0
Iteration 100
Iteration 200
Iteration 300
Iteration 400
Iteration 500
Iteration 600
Iteration 700
Iteration 800
Iteration 900
Iteration 1000
Iteration 1100
Iteration 1200
Iteration 1300
Iteration 1400
```

In [ ]:
```
test.plot_posteriors()
```



In [ ]:
```
test.get_rate_estimates()
```

Out[ ]:

	Posterior EAP	Posterior Cred Interval Lower	Posterior Cred Interval Upper
mu	3.053165	2.548502	3.598784
lambda	0.925168	0.711589	1.172141

In [ ]:
```
most_probable = test.get_k_probs()
most_probable.sort_values(by=["Probability of k"], ascending=False).head(5)
```

Out[ ]:

	k	Probability of k
9	41.0	0.232788
8	40.0	0.172852
7	39.0	0.155884
10	42.0	0.106873
5	37.0	0.097900

In [ ]:
```
most_probable['year'] =[int(y+1851) for y in most_probable['k']]
most_probable.sort_values(by=["Probability of k"], ascending=False).head(5)
```

Out[ ]:

	k	Probability of k	year
9	41.0	0.232788	1892
8	40.0	0.172852	1891
7	39.0	0.155884	1890
10	42.0	0.106873	1893
5	37.0	0.097900	1888

Then do the the following:

a) Describe your findings. What do these EAP and credible intervals imply? And what was the most likely year of the changepoint?

The posterior distributions show that the most likely date for the changepoint is centered around 40 (1892). The data is clearly on either side of this break, which mirrors what we see in the original plot. The EAP is the expected value of changepoint, and the credible interval is the range of dates most likely to be the changepoint.

The year most likely to be the changepoint is 1892.

b) Why is an EAP or credible interval not neccesarily the most appropriate thing to report for the year of the changepoint?

The EAP is the expected value, and in this case, it might not be the most appropriate measure since we are interested in the overall shape of the distribution, and when things began to change.

For this reason, the credible interval is probably more appropriate, however, if we wanted a basis for a single point in time, the EAP would be better to report the year of the changepoint.

Part 2: Bayesian Logisitic Regression with PyMC3

1. Load the data ansd create a new binary variable where the new quality of the wine is 0 if the wine recieved a C or an F, and a 1 if the wine recieved an A.

In [ ]:
```
wine = pd.read_csv('whitewine-training-ds6040.csv')
```

In [ ]:
```
wine['quality_binary'] = [1 if x == "A" else 0 for x in wine['wine_quality']]
```

In [ ]:
```
overall_best_A = "wine['wine_quality'] ~ wine['fixed.acidity']+wine['free.sulfur.dioxide']+wine['alcohol']"
```

In [ ]:
```
import pymc3 as pm
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

```
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

In [ ]:
```
###The following code is heavily based or taken directly from: https://www.kaggle.com/billbasener/pymc3-bayesian-logistic-regression-classification

def plot_traces(traces, retain=0):
    '''
    Convenience function:
    Plot traces with overlaid means and values
    '''

    ax = pm.traceplot(traces[-retain:],
                      lines=tuple([(k, {}, v['mean'])
                                   for k, v in pm.summary(traces[-retain:]).iterrows()]))

    for i, mn in enumerate(pm.summary(traces[-retain:])['mean']):
        ax[i,0].annotate('{:.2f}'.format(mn), xy=(mn,0), xycoords='data'
                    ,xytext=(5,10), textcoords='offset points', rotation=90
                    ,va='bottom', fontsize='large', color='#AA0022')
```

In [ ]:
```
wine.columns = wine.columns.str.replace(".", "_", regex=False)
wine_best = wine[['volatile_acidity', 'residual_sugar', 'alcohol', 'quality_binary']]
wine_best_A = wine[['fixed_acidity','free_sulfur_dioxide','alcohol', 'quality_binary']]
```

In [ ]:

```
#This is the general form for the logistic. You will need to change the model
 formula to match the white wine data.
with pm.Model() as model:
    pm.glm.GLM.from_formula(formula = "quality_binary ~ volatile_acidity + res
idual_sugar + alcohol",
                            data = wine_best,
                            family = pm.glm.families.Binomial())

    trace = pm.sample(1000)


#You can obtain forest plots with:
pm.plots.forestplot(trace, figsize = (12,5), rope=(0, 0))
pm.plot_trace(trace)
#You can get a table of summary statistics out using:
pm.summary(trace)
```

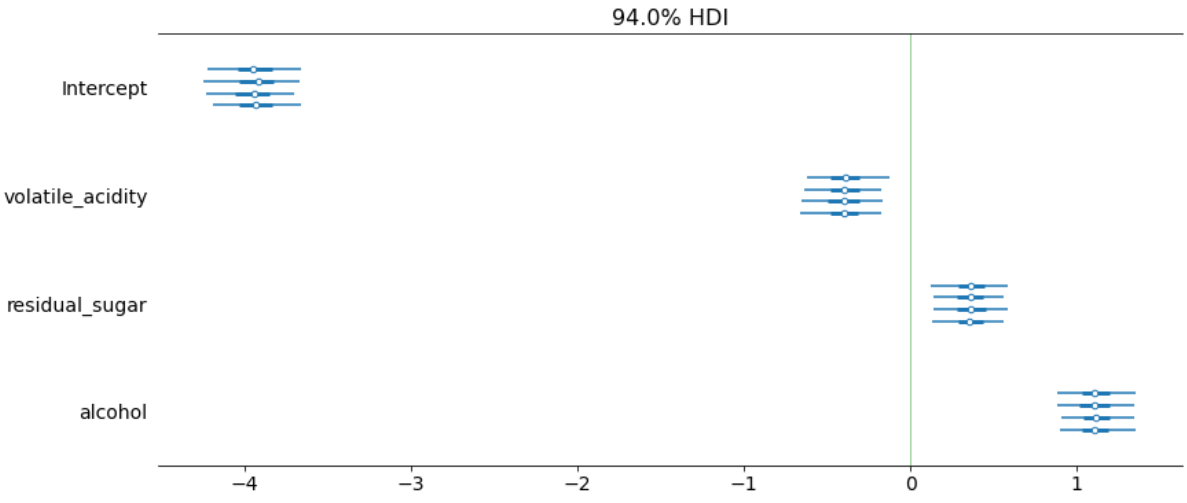
```
The glm module is deprecated and will be removed in version 4.0
We recommend to instead use Bambi https://bambinos.github.io/bambi/
C:\Users\laure\AppData\Local\Temp\ipykernel_26092\2126371662.py:7: FutureWarn
ing: In v4.0, pm.sample will return an `arviz.InferenceData` object instead o
f a `MultiTrace` by default. You can pass return_inferencedata=True or return
_inferencedata=False to be safe and silence this warning.
  trace = pm.sample(1000)
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [alcohol, residual_sugar, volatile_acidity, Intercept]
```

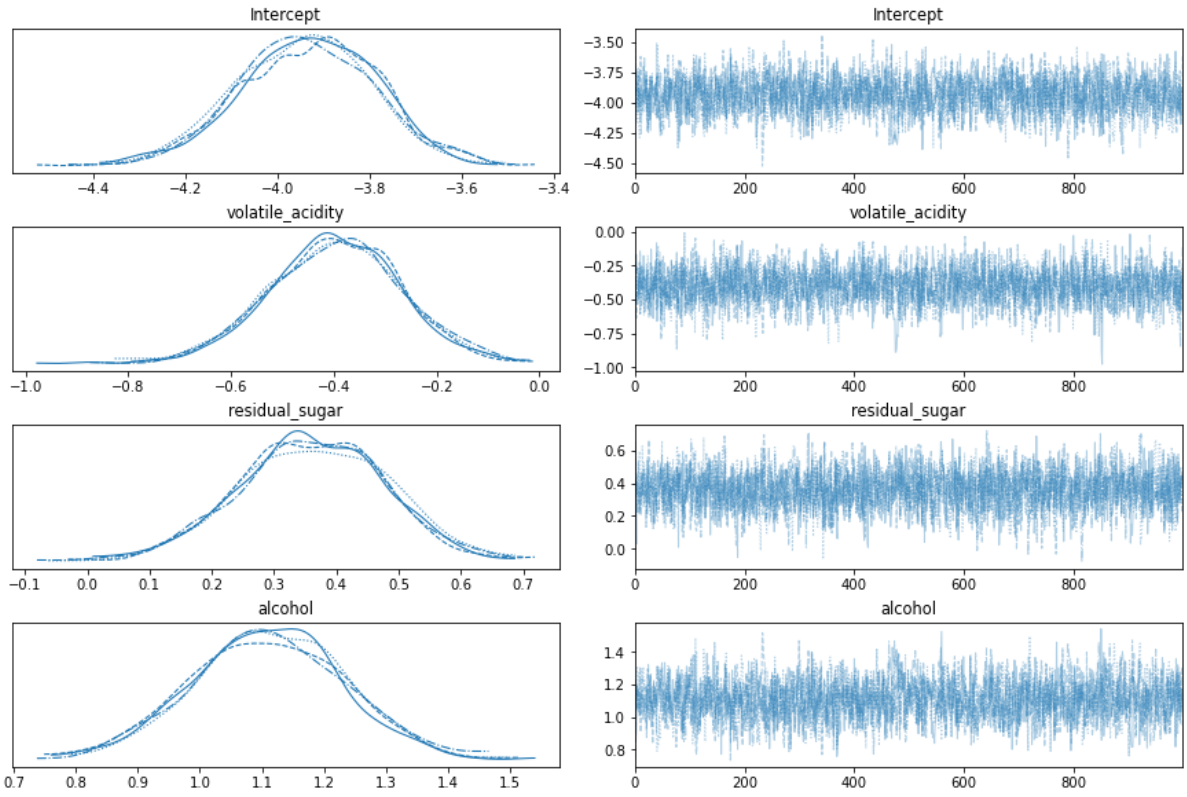
100.00% [8000/8000 00:11<00:00 Sampling 4 chains, 0 divergences]

```
Sampling 4 chains for 1_000 tune and 1_000 draw iterations (4_000 + 4_000 dra
ws total) took 27 seconds.
C:\Users\laure\AppData\Local\Temp\ipykernel_26092\2126371662.py:11: Deprecati
onWarning: The function `forestplot` from PyMC3 is just an alias for `plot_fo
rest` from ArviZ. Please switch to `pymc3.plot_forest` or `arviz.plot_forest
`.
  pm.plots.forestplot(trace, figsize = (12,5), rope=(0, 0))
c:\Users\laure\anaconda3\envs\pymc_env\lib\site-packages\arviz\data\io_pymc3.
py:96: FutureWarning: Using `from_pymc3` without the model will be deprecated
in a future release. Not using the model will return less accurate and less u
seful results. Make sure you use the model argument or call from_pymc3 within
a model context.
  warnings.warn(
c:\Users\laure\anaconda3\envs\pymc_env\lib\site-packages\arviz\data\io_pymc3.
py:96: FutureWarning: Using `from_pymc3` without the model will be deprecated
in a future release. Not using the model will return less accurate and less u
seful results. Make sure you use the model argument or call from_pymc3 within
a model context.
  warnings.warn(
c:\Users\laure\anaconda3\envs\pymc_env\lib\site-packages\arviz\data\io_pymc3.
py:96: FutureWarning: Using `from_pymc3` without the model will be deprecated
in a future release. Not using the model will return less accurate and less u
seful results. Make sure you use the model argument or call from_pymc3 within
a model context.
  warnings.warn(
```

Out[ ]:

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
Intercept	-3.938	0.147	-4.218	-3.666	0.003	0.002	1794.0	1906.0	1.0
volatile_acidity	-0.399	0.129	-0.654	-0.167	0.003	0.002	2222.0	2524.0	1.0
residual_sugar	0.360	0.119	0.137	0.581	0.002	0.002	2334.0	2416.0	1.0
alcohol	1.111	0.122	0.888	1.346	0.003	0.002	1730.0	1658.0	1.0


94.0% HDI
Intercept
volatile_acidity
residual_sugar
alcohol
−4
−3
−2
−1
0
1


Intercept
volatile_acidity
residual_sugar
alcohol

In [ ]:

```
#This is the general form for the logistic. You will need to change the model
 formula to match the white wine data.
with pm.Model() as model:
    pm.glm.GLM.from_formula(formula = "quality_binary ~ fixed_acidity + free_s
ulfur_dioxide + alcohol",
                            data = wine_best_A,
                            family = pm.glm.families.Binomial())

    traceA = pm.sample(1000)


#You can obtain forest plots with:
pm.plots.forestplot(traceA, figsize = (12,5), rope=(0,0))
pm.plot_trace(traceA)
#You can get a table of summary statistics out using:
pm.summary(traceA)
```

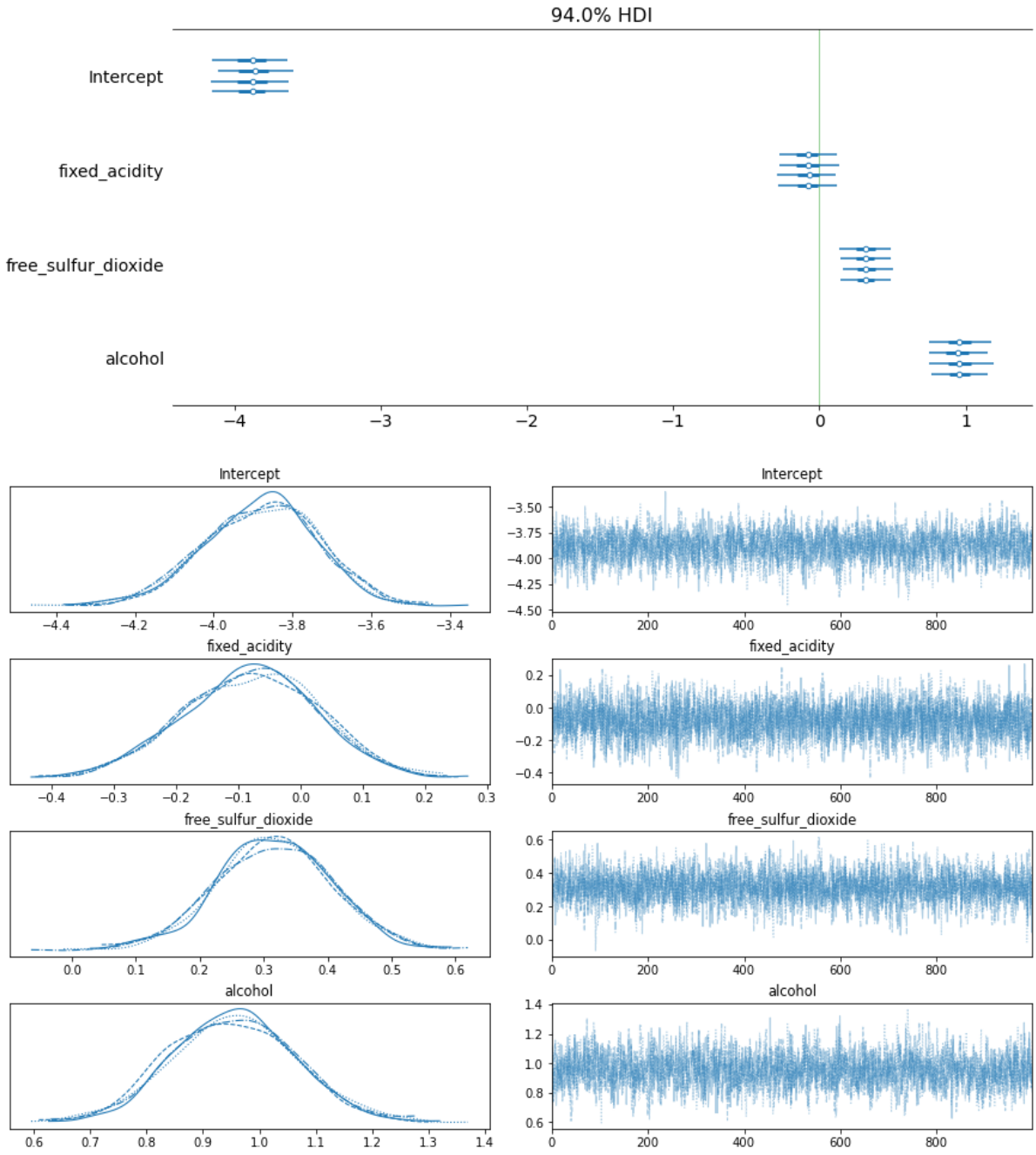
```
The glm module is deprecated and will be removed in version 4.0
We recommend to instead use Bambi https://bambinos.github.io/bambi/
C:\Users\laure\AppData\Local\Temp\ipykernel_26092\1593436193.py:7: FutureWarn
ing: In v4.0, pm.sample will return an `arviz.InferenceData` object instead o
f a `MultiTrace` by default. You can pass return_inferencedata=True or return
_inferencedata=False to be safe and silence this warning.
  traceA = pm.sample(1000)
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [alcohol, free_sulfur_dioxide, fixed_acidity, Intercept]
```

100.00% [8000/8000 00:10<00:00 Sampling 4 chains, 0 divergences]

```
Sampling 4 chains for 1_000 tune and 1_000 draw iterations (4_000 + 4_000 dra
ws total) took 26 seconds.
C:\Users\laure\AppData\Local\Temp\ipykernel_26092\1593436193.py:11: Deprecati
onWarning: The function `forestplot` from PyMC3 is just an alias for `plot_fo
rest` from ArviZ. Please switch to `pymc3.plot_forest` or `arviz.plot_forest
`.
  pm.plots.forestplot(traceA, figsize = (12,5), rope=(0,0))
c:\Users\laure\anaconda3\envs\pymc_env\lib\site-packages\arviz\data\io_pymc3.
py:96: FutureWarning: Using `from_pymc3` without the model will be deprecated
in a future release. Not using the model will return less accurate and less u
seful results. Make sure you use the model argument or call from_pymc3 within
a model context.
  warnings.warn(
c:\Users\laure\anaconda3\envs\pymc_env\lib\site-packages\arviz\data\io_pymc3.
py:96: FutureWarning: Using `from_pymc3` without the model will be deprecated
in a future release. Not using the model will return less accurate and less u
seful results. Make sure you use the model argument or call from_pymc3 within
a model context.
  warnings.warn(
c:\Users\laure\anaconda3\envs\pymc_env\lib\site-packages\arviz\data\io_pymc3.
py:96: FutureWarning: Using `from_pymc3` without the model will be deprecated
in a future release. Not using the model will return less accurate and less u
seful results. Make sure you use the model argument or call from_pymc3 within
a model context.
  warnings.warn(
```

Out[ ]:

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r
Intercept	-3.882	0.143	-4.156	-3.626	0.003	0.002	2251.0	2888.0	
fixed_acidity	-0.080	0.108	-0.280	0.124	0.002	0.002	3143.0	2405.0	
free_sulfur_dioxide	0.316	0.091	0.142	0.486	0.002	0.001	3186.0	2603.0	
alcohol	0.955	0.111	0.755	1.172	0.002	0.002	2165.0	2563.0	


94.0% HDI
Intercept
fixed_acidity
free_sulfur_dioxide
alcohol
−4
−3
−2
−1
0
1
Intercept
Intercept
fixed_acidity
fixed_acidity
free_sulfur_dioxide
free_sulfur_dioxide
alcohol
alcohol

In [ ]:

Discuss your findings

for the overall fit: y = -3.933 + -0.399 (volatile acidity) + 0.358 (residual sugar) + 1.108(alcohol)

for the A-type wines: y = -3.887 + -0.080(fixed acidity) + 0.315 (free sulphur dioxide) + 0.959(alcohol)

The corresponding equations are part of the prediction model to predict whether a wine is likely to be classified as 'C' or 'F', or 'A'.

Interpreting the Intercept: The intercept for both models is close to -4. This is the 'base' quality of the wine before the other factors are considered.

The other factors increase or decrease a likelihood that a wine is rated 'A' Quality. The closer to 1, the more likely it is that a wine is rated A quality. Acid decreases this value, where alcohol increases it.

Each parameter has a range where the value is likely to truly affect the response. Fixed acidity (a-type wine) crosses 0, which means that it is uncertain if Fixed acidititty truly increases or decreases the likelihood of a wine being 'A' Quality.

In [ ]: