In [1]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pymc3 as pm
import pandas as pd
import theano
import arviz as az
import seaborn as sns
```

In [2]:
```
sales = pd.read_csv("data/sales-ds6040sum2021.csv")
sales.head()
```

Out[2]:

	sales	food	con	neur	store
0	1.363821	0	0.216553	1.290224	0
1	-1.119747	1	0.216553	1.290224	0
2	-0.180141	0	0.216553	1.290224	0
3	-2.282334	1	0.216553	1.290224	0
4	0.673304	0	0.216553	1.290224	0

Question 1 Bayesian Hierarchical Modelling

You have been hired by a regional chain of coffee shops to help improve sales and to examine how the personality characteristics of individual store managers might impact the sales numbers of both coffee and food. The client as the following questions they need answered:

- How does conscientiousness and neuroticism impact the sales of coffee and food, and are coffee and food impacted differently?
- Once you control for the personality characteristics of the store managers, what stores should be performing well? (i.e. the rest of the employees might be great, but the store manager might be bringing sales down)

IMPORTANT: This part of the assignment is meant to simulate the experience of a working data scientist. For this section, prepare your response as though you are preparing a report for your client. This means that it needs to be readable, well formatted, and detail your reasoning.

1. Problem Statement - What problem are you tackling?

The main problem statement here is to see how the personality characteristics of individual store managers might impact sales numbers for both coffee and food. Will conscientiousness and neuroticism impact the sales of coffee and food, if so how will they impact the sales? How are the stores performing. These questions are the questions that need to be answered from our analysis.

1. Approach- Describe the model you are using. Present it both in equation form, as well as a written description of the model. Importantly, you are not writing this for another data scientist, you are writing this for someone who is capable of understanding what a regression is, and what these sorts of models can provide, but has never worked with data analyses or statistics before (so smart, but without the same knowledge base you have.)

The sales data has the following columns. Sales, Food, con, neur, store. The data includes all the information for multiple stores. In some ways one can say that the data is nested since all stores' data is in the main data. Another question that we would like to tackle is the difference of sales for each store, more specficically how the stores conscientiousness and neuroticism impact the sales.

Taking a closer look at the data we can see that there are 20 stores.

In [3]:
```
sales.store.unique()
```

Out[3]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19])
```

Here the unique store ids are from 0:19, which means that there are 20 stores.

Along with this we can see that the con (conscientiousness) and neur (neuroticism) is kept the same for each store as shown below for store 0 and store 1.

In [4]:
```
pd.concat([sales[(sales['store'] == 0)].head(3),sales[(sales['store'] == 1)].he
```

Out[4]:

	sales	food	con	neur	store
0	1.363821	0	0.216553	1.290224	0
1	-1.119747	1	0.216553	1.290224	0
2	-0.180141	0	0.216553	1.290224	0
24	0.279672	0	-0.135665	-2.276493	1
25	-2.559676	1	-0.135665	-2.276493	1
26	-0.227531	0	-0.135665	-2.276493	1

Given this information the best modeling approach at this time will be the hierarchical modeling approach where the hierarchy is the stores.

From the below code we can notice that the estimation equation to find the likelihood will be the following.

Let β be the effect of con for coffee, and β_f be the difference in that effect for food.

Let γ be the effect of neur for coffee, and γ_f be the difference in that effect for food.

Let α is the hierarchical affect and also the overall intercept, level of sales at mean levels of con and neur.

$$\hat{\text{sales}} = \alpha_{store} + \alpha\_f_{store} + \beta * \text{con} + \beta_f * \text{con} + \gamma * \text{neur} + \gamma_f * \text{neur}$$

here we have two hierarchical effects in our model because within each store, the con and neur are the same. We would want to control the personality of the managers.

In [5]:
```
store_idx = sales.store.values
n_stores = len(sales.store.unique())

with pm.Model() as hierarchical_model:
    # Priors for the fixed effects
    # a - overall intercept, level of sales at mean levels of con and neur
    mu_a = pm.Normal('mu_a', mu=0., sd=1e5)
    sigma_a = pm.HalfCauchy('sigma_a', 5)
    mu_a_f = pm.Normal('mu_a_f', mu=0., sd=1e5)
    sigma_a_f = pm.HalfCauchy('sigma_a_f', 5)

    # Sales intercepts as offsets
    a_offset = pm.Normal('a_offset', mu=0, sd=1, shape=20)
    a = pm.Deterministic("a", mu_a + a_offset * sigma_a)
    a_offset_f = pm.Normal('a_offset_f', mu=0, sd=1, shape=20)
    a_f = pm.Deterministic("a_f", mu_a_f + a_offset_f * sigma_a_f)

    # Store level effect of con as offset
    b_offset = pm.Normal('b_offset', mu=0, sd=1)
    b = pm.Deterministic("b", b_offset)
    b_offset_f = pm.Normal('b_offset_f', mu=0, sd=1)
    b_f = pm.Deterministic("b_f", b_offset_f)

    # Store level effect of neur as offset
    c_offset = pm.Normal('c_offset', mu=0, sd=1)
    c = pm.Deterministic("c", c_offset)
    c_offset_f = pm.Normal('c_offset_f', mu=0, sd=1)
    c_f = pm.Deterministic("c_f", c_offset_f)


    # Model error
    eps = pm.HalfCauchy('eps', 5)

    # Linear regression
    food = sales.food.values
    con = sales.con.values
    neur = sales.neur.values

    sales_est = a[store_idx] + a_f[store_idx]*food \
            + b *con + b_f*con*food \
            + c *neur + c_f*neur*food

    # Data likelihood
    sales_like = pm.Normal('sales_like',
                           mu=sales_est,
                           sd=eps,
                           observed=sales.sales)
with hierarchical_model:
    hierarchical_trace = pm.sample(1000, n_init=50000, tune=1000, target_accept
```

```
/usr/local/lib/python3.9/site-packages/deprecat/classic.py:215: FutureWarning:
In v4.0, pm.sample will return an `arviz.InferenceData` object instead of a `M
ultiTrace` by default. You can pass return_inferencedata=True or return_infere
ncedata=False to be safe and silence this warning.
  return wrapped_(*args_, **kwargs_)
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (4 chains in 4 jobs)
NUTS: [eps, c_offset_f, c_offset, b_offset_f, b_offset, a_offset_f, a_offset,
sigma_a_f, mu_a_f, sigma_a, mu_a]
```

 100.00% [8000/8000 00:18<00:00 Sampling 4 chains, 0 divergences]

```
Sampling 4 chains for 1_000 tune and 1_000 draw iterations (4_000 + 4_000 draw
s total) took 29 seconds.
```

1. Prior Rationale- List your prior choices and why those were chosen.

The prior information for the α and α_f was chosen to be the following.

$$\mu = Normal(0, 10000)$$

$$\sigma = HalfCauchy(5)$$

$$\text{offset} = Normal(0, 1, \text{shape} = 20)$$

$$\alpha = \text{Deterministic}(\mu + \text{offset} * \sigma)$$

The reasoning for this was to keep a wide prior information was because there wasn't that clear information that could be used. In using these large priors we will be able to make a general idea about the behavior of the posterior distributions.

The prior information for β, β_f, γ, γ_f was chosen to be the following.

$$\text{offset} = Normal(0, 1)$$

$$\beta = \text{Deterministic}(\text{offset})$$

where \beta can be replaced by any of the following β, β_f, γ, γ_f for notation purposes.

Again, the reasoning for prior choices was set as such because there wasn't clear information about the data. Also, along with this there didn't need to be a shape input for these betas because within each store, the con and neur are the same.

1. Findings- This is where you present your findings.

The trace plots, forest plots, and summary of the data can be found in section 6.

From our summary dataframe shown in section 6, we noticed that all r hat values were either 1.00 or 1.01, which is a good indicator that our sampler is performing well. When using 4 chains and 1,000 sample and 1,000 tune and an target acceptance of 0.95 our model

converged well with 0 divergences as shown by the model output in section 2 along with the traceplots all showing convergence.

In [6]:
```
summary = pm.summary(hierarchical_trace)
```

```
Got error No model on context stack. trying to find log_likelihood in translat
ion.
/usr/local/lib/python3.9/site-packages/arviz/data/io_pymc3_3x.py:98: FutureWar
ning: Using `from_pymc3` without the model will be deprecated in a future rele
ase. Not using the model will return less accurate and less useful results. Ma
ke sure you use the model argument or call from_pymc3 within a model context.
  warnings.warn(
```

In [7]:
```
summary = summary.reset_index()
summary[(summary['index'].str.contains('b')) | (summary['index'].str.contains('
```

Out[7]:

	index	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_l
42	b_offset	0.294	0.456	-0.602	1.101	0.012	0.009	1415.0	1927.0	
43	b_offset_f	0.904	0.327	0.277	1.513	0.008	0.005	1818.0	2331.0	
44	c_offset	-0.359	0.285	-0.889	0.181	0.007	0.005	1823.0	2470.0	
45	c_offset_f	0.121	0.210	-0.265	0.526	0.005	0.003	2164.0	2532.0	
88	b	0.294	0.456	-0.602	1.101	0.012	0.009	1415.0	1927.0	
89	b_f	0.904	0.327	0.277	1.513	0.008	0.005	1818.0	2331.0	
90	c	-0.359	0.285	-0.889	0.181	0.007	0.005	1823.0	2470.0	
91	c_f	0.121	0.210	-0.265	0.526	0.005	0.003	2164.0	2532.0	

From the summary statistics we can see that generally the mean with food sales was higher than that of the means without food sales.

In [9]:
```
summary[summary['index'].str.contains('14')]
```

Out[9]:

	index	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail
16	a_offset[14]	2.480	0.578	1.396	3.521	0.015	0.010	1571.0	2389.0
36	a_offset_f[14]	0.956	0.598	-0.186	2.082	0.011	0.008	2741.0	2779.0
62	a[14]	3.574	0.443	2.741	4.385	0.009	0.007	2389.0	2448.0
82	a_f[14]	-0.069	0.438	-0.860	0.799	0.007	0.006	3452.0	2866.0

Looking at the store 14, we can say that the shop will generally perform well without the store manager's personality effect.

In [10]:
```
summary[(summary['index'].str.contains('a\['))].sort_values('mean', ascending=F
```

Out[10]:

	index	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
62	a[14]	3.574	0.443	2.741	4.385	0.009	0.007	2389.0	2448.0	1.0
60	a[12]	1.725	0.414	0.939	2.484	0.009	0.007	2030.0	2554.0	1.0
65	a[17]	1.574	0.360	0.864	2.204	0.008	0.005	2291.0	3180.0	1.0
61	a[13]	1.525	0.520	0.632	2.560	0.012	0.008	1936.0	2413.0	1.0
52	a[4]	1.271	0.531	0.334	2.316	0.014	0.010	1527.0	2130.0	1.0
51	a[3]	1.098	0.384	0.379	1.806	0.009	0.006	2043.0	2380.0	1.0
63	a[15]	1.069	0.283	0.597	1.659	0.004	0.003	4173.0	3139.0	1.0
55	a[7]	0.705	0.919	-1.124	2.353	0.025	0.018	1316.0	1964.0	1.0
53	a[5]	0.413	0.451	-0.412	1.278	0.010	0.007	2238.0	2620.0	1.0
54	a[6]	0.251	0.360	-0.420	0.942	0.007	0.005	2463.0	3028.0	1.0
66	a[18]	0.200	0.730	-1.066	1.664	0.016	0.012	1976.0	2561.0	1.0
48	a[0]	0.185	0.445	-0.634	1.025	0.009	0.007	2222.0	3148.0	1.0
56	a[8]	0.005	0.356	-0.657	0.687	0.007	0.005	2826.0	2837.0	1.0
50	a[2]	-0.056	0.415	-0.809	0.757	0.009	0.007	1990.0	2600.0	1.0
58	a[10]	-0.299	0.326	-0.915	0.314	0.006	0.005	2537.0	3384.0	1.0
67	a[19]	-0.358	0.302	-0.917	0.205	0.005	0.004	3511.0	3613.0	1.0
49	a[1]	-0.485	0.675	-1.802	0.713	0.016	0.011	1834.0	2498.0	1.0
59	a[11]	-0.592	0.357	-1.266	0.081	0.007	0.005	2290.0	2778.0	1.0
57	a[9]	-0.843	0.488	-1.760	0.067	0.010	0.008	2202.0	2453.0	1.0
64	a[16]	-1.000	0.418	-1.813	-0.245	0.009	0.007	2007.0	2583.0	1.0

In [11]:
```
summary[(summary['index'].str.contains('a_f'))].sort_values('mean', ascending=F
```

Out[11]:

	index	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_
47	sigma_a_f	0.849	0.190	0.506	1.197	0.005	0.003	1503.0	2725.0	
76	a_f[8]	0.066	0.399	-0.678	0.831	0.006	0.005	3874.0	3563.0	
83	a_f[15]	0.012	0.375	-0.671	0.744	0.005	0.006	5733.0	3186.0	
85	a_f[17]	-0.001	0.405	-0.729	0.777	0.006	0.006	4506.0	3346.0	
81	a_f[13]	-0.030	0.463	-0.936	0.821	0.008	0.007	3316.0	2887.0	
82	a_f[14]	-0.069	0.438	-0.860	0.799	0.007	0.006	3452.0	2866.0	
78	a_f[10]	-0.499	0.382	-1.157	0.246	0.006	0.004	4634.0	3618.0	
74	a_f[6]	-0.556	0.401	-1.307	0.200	0.006	0.005	4462.0	3389.0	
79	a_f[11]	-0.569	0.395	-1.273	0.199	0.006	0.005	4162.0	3049.0	
84	a_f[16]	-0.766	0.421	-1.554	0.020	0.007	0.005	3724.0	3485.0	
1	mu_a_f	-0.863	0.229	-1.319	-0.459	0.006	0.004	1710.0	2256.0	
72	a_f[4]	-0.887	0.478	-1.720	0.097	0.009	0.006	2994.0	2972.0	
71	a_f[3]	-0.892	0.414	-1.610	-0.068	0.007	0.005	4027.0	3278.0	
69	a_f[1]	-1.043	0.534	-2.095	-0.066	0.010	0.007	2650.0	2899.0	
75	a_f[7]	-1.112	0.668	-2.398	0.103	0.014	0.010	2214.0	2567.0	
86	a_f[18]	-1.174	0.555	-2.199	-0.073	0.010	0.008	2874.0	2994.0	
87	a_f[19]	-1.206	0.374	-1.877	-0.482	0.005	0.004	5103.0	2882.0	
77	a_f[9]	-1.265	0.447	-2.098	-0.410	0.008	0.006	3208.0	3228.0	
68	a_f[0]	-1.490	0.437	-2.311	-0.650	0.007	0.005	3790.0	3147.0	
73	a_f[5]	-1.546	0.441	-2.407	-0.765	0.007	0.005	3613.0	3326.0	
70	a_f[2]	-1.945	0.416	-2.716	-1.157	0.007	0.005	4073.0	3246.0	
80	a_f[12]	-2.317	0.441	-3.149	-1.509	0.008	0.005	3350.0	3079.0	

1. Summary- This is where you summarize and interpret what your analyses uncovered. Again, this is for the client, so it needs to be usable information for them.

Given that the coefficents are all the same for all 20 models, we can quickly look at the intercepts for the 20 models. Sorting by the mean, we can see that the best performing store for coffee sales would be 14 and the worst performing store would be 16. For food sales the best performing store will be store 8 while the worst is store 12.

Another interesting metric to look at is that almost all the a_f intercepts (hierarchical intercept for food sales) were negative. The coffee sales hierarchical intercepts actually had mostly postivie slopes, which means sales will go up. Genearlly speaking, from a store's perspective planning more towards coffee sales might be beneficial to the store revenue.

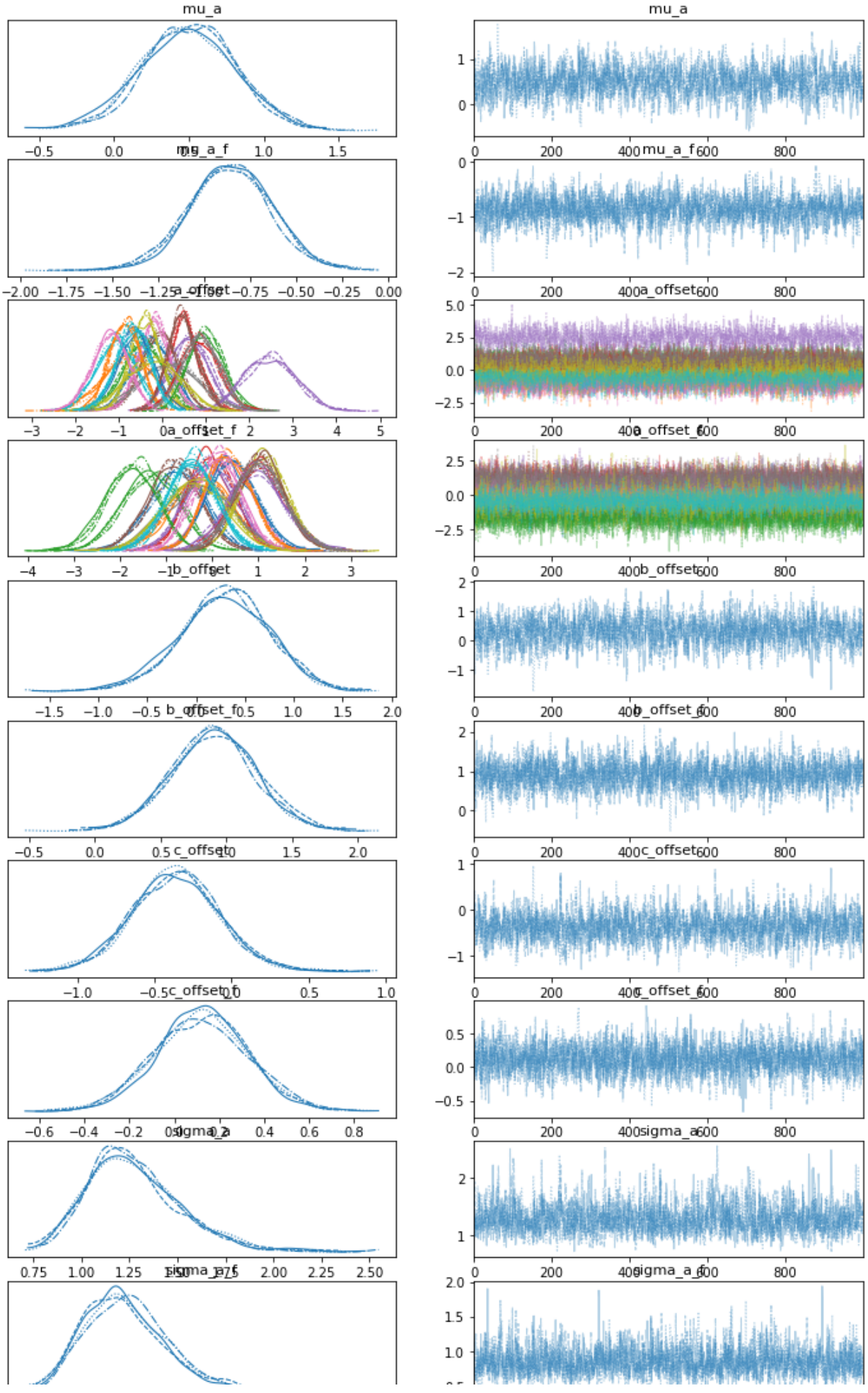
Also, from the posteriors, we can see which stores are performing poorly and which stores are doing the best. If the focus of store revenue is for coffee sales from a managing point of view it may be benefifical to look at the stores that have a negative mean and the same goes to food sales and looking at their means.

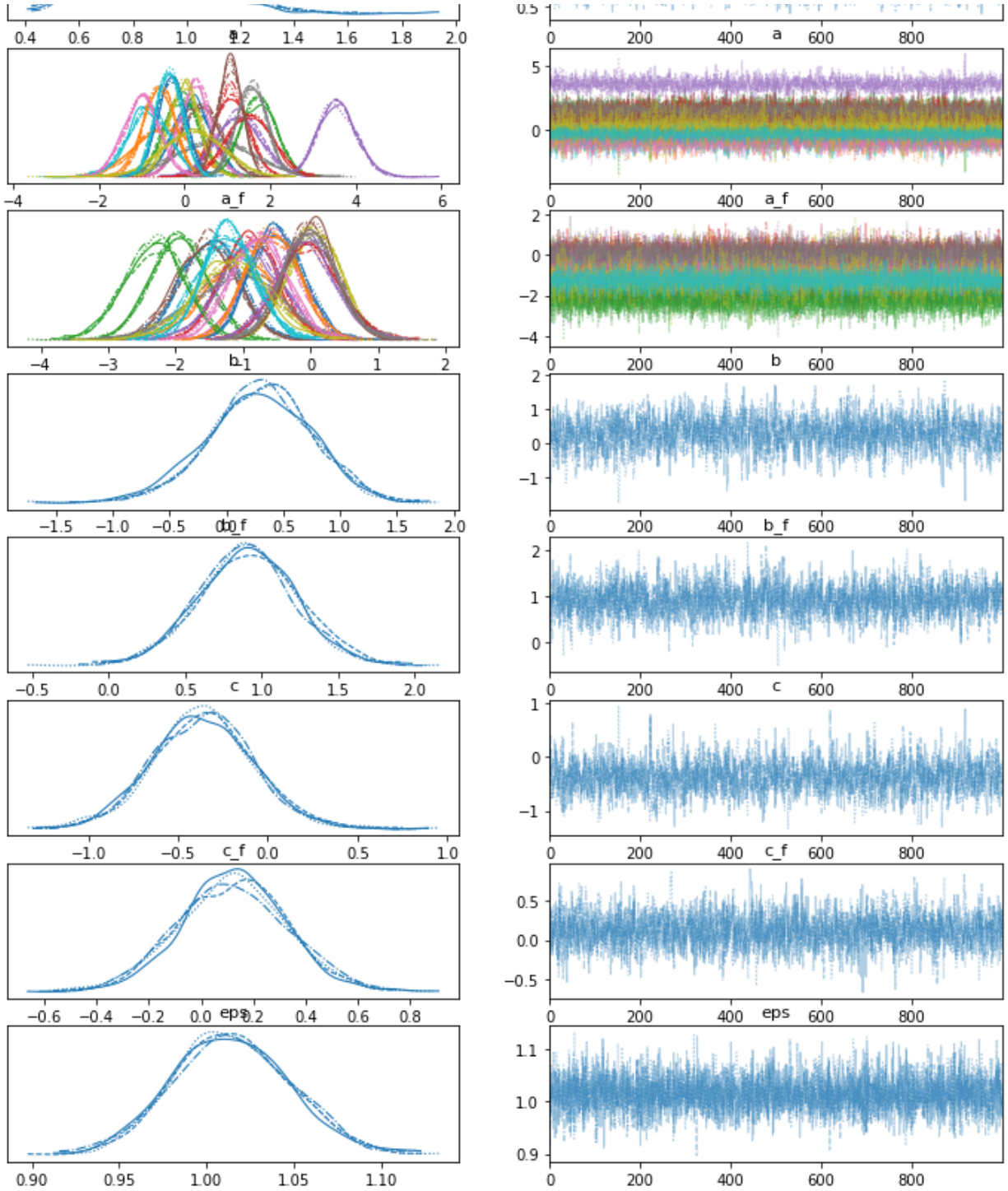
1. Diagnostics- This is where you put information/plots as to how the estimator performed. This is for technical reference (I like to always have these in the reports I create, but this information is not really for a client per say, more for another data scientist to validate your work.) This doesn't need to be long.

In [12]:
```
az.rcParams["plot.max_subplots"] = 50# This finishes without error
az.plot_trace(hierarchical_trace)
```

```
Got error No model on context stack. trying to find log_likelihood in translat
ion.
/usr/local/lib/python3.9/site-packages/arviz/data/io_pymc3_3x.py:98: FutureWar
ning: Using `from_pymc3` without the model will be deprecated in a future rele
ase. Not using the model will return less accurate and less useful results. Ma
ke sure you use the model argument or call from_pymc3 within a model context.
  warnings.warn(
Got error No model on context stack. trying to find log_likelihood in translat
ion.
```

Out[12]:
```
array([[<AxesSubplot:title={'center':'mu_a'}>,
        <AxesSubplot:title={'center':'mu_a'}>],
       [<AxesSubplot:title={'center':'mu_a_f'}>,
        <AxesSubplot:title={'center':'mu_a_f'}>],
       [<AxesSubplot:title={'center':'a_offset'}>,
        <AxesSubplot:title={'center':'a_offset'}>],
       [<AxesSubplot:title={'center':'a_offset_f'}>,
        <AxesSubplot:title={'center':'a_offset_f'}>],
       [<AxesSubplot:title={'center':'b_offset'}>,
        <AxesSubplot:title={'center':'b_offset'}>],
       [<AxesSubplot:title={'center':'b_offset_f'}>,
        <AxesSubplot:title={'center':'b_offset_f'}>],
       [<AxesSubplot:title={'center':'c_offset'}>,
        <AxesSubplot:title={'center':'c_offset'}>],
       [<AxesSubplot:title={'center':'c_offset_f'}>,
        <AxesSubplot:title={'center':'c_offset_f'}>],
       [<AxesSubplot:title={'center':'sigma_a'}>,
        <AxesSubplot:title={'center':'sigma_a'}>],
       [<AxesSubplot:title={'center':'sigma_a_f'}>,
        <AxesSubplot:title={'center':'sigma_a_f'}>],
       [<AxesSubplot:title={'center':'a'}>,
        <AxesSubplot:title={'center':'a'}>],
       [<AxesSubplot:title={'center':'a_f'}>,
        <AxesSubplot:title={'center':'a_f'}>],
       [<AxesSubplot:title={'center':'b'}>,
        <AxesSubplot:title={'center':'b'}>],
       [<AxesSubplot:title={'center':'b_f'}>,
        <AxesSubplot:title={'center':'b_f'}>],
       [<AxesSubplot:title={'center':'c'}>,
        <AxesSubplot:title={'center':'c'}>],
       [<AxesSubplot:title={'center':'c_f'}>,
        <AxesSubplot:title={'center':'c_f'}>],
       [<AxesSubplot:title={'center':'eps'}>,
        <AxesSubplot:title={'center':'eps'}>]], dtype=object)
```

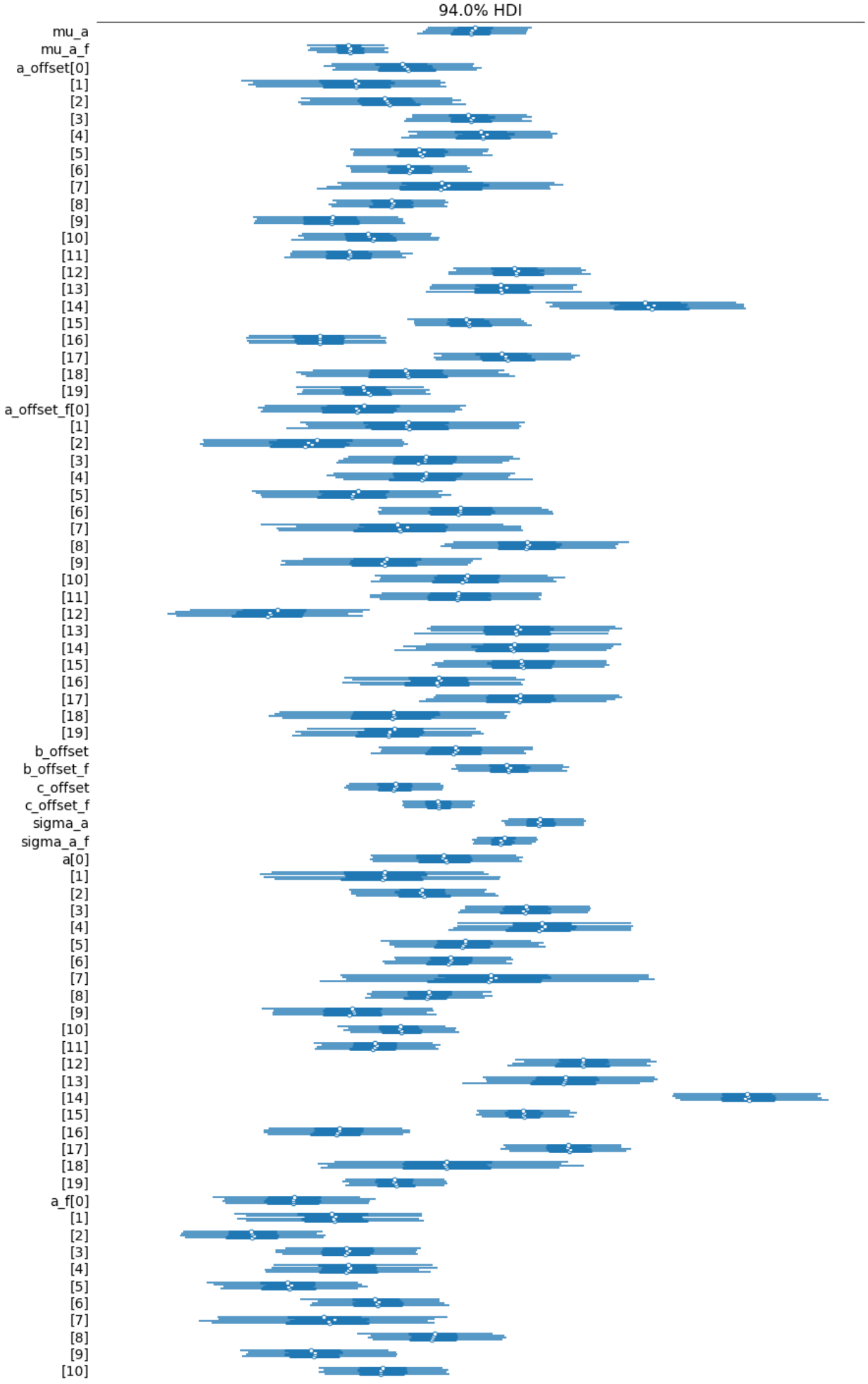

mu_a
mu_a_f
a_offset
a_offset_f
b_offset
b_offset_f
c_offset
c_offset_f
sigma_a
sigma_a_f

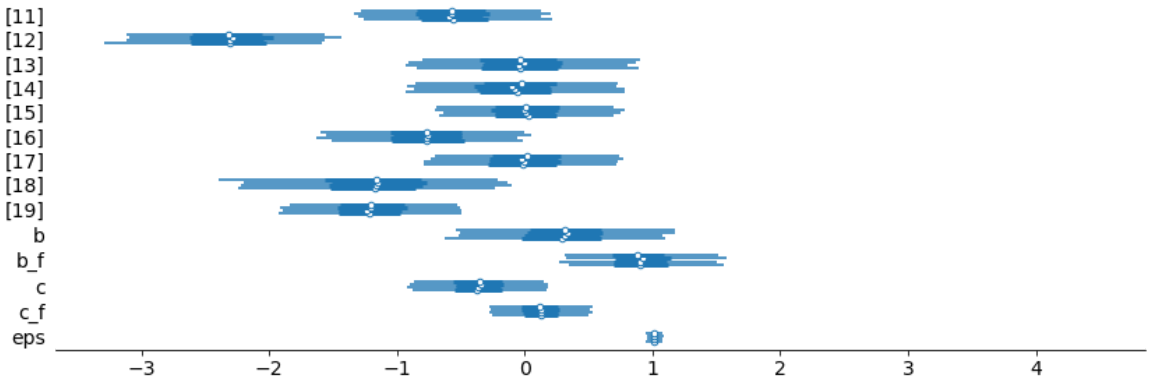


In [13]:
```
pm.plots.forestplot(hierarchical_trace, figsize = (14,30))
```

```
/var/folders/pn/dgy7ckd90nl7mlj6g6rc_1kw0000gn/T/ipykernel_84097/926719147.py:
1: DeprecationWarning: The function `forestplot` from PyMC3 is just an alias f
or `plot_forest` from ArviZ. Please switch to `pymc3.plot_forest` or `arviz.pl
ot_forest`.
  pm.plots.forestplot(hierarchical_trace, figsize = (14,30))
Got error No model on context stack. trying to find log_likelihood in translat
ion.
/usr/local/lib/python3.9/site-packages/arviz/data/io_pymc3_3x.py:98: FutureWar
ning: Using `from_pymc3` without the model will be deprecated in a future rele
ase. Not using the model will return less accurate and less useful results. Ma
ke sure you use the model argument or call from_pymc3 within a model context.
  warnings.warn(
```

Out[13]: array([<AxesSubplot:title={'center':'94.0% HDI'}>], dtype=object)


94.0% HDI
mu_a
mu_a_f
a_offset[0]
[1]
[2]
[3]
[4]
[5]
[6]
[7]
[8]
[9]
[10]
[11]
[12]
[13]
[14]
[15]
[16]
[17]
[18]
[19]
a_offset_f[0]
[1]
[2]
[3]
[4]
[5]
[6]
[7]
[8]
[9]
[10]
[11]
[12]
[13]
[14]
[15]
[16]
[17]
[18]
[19]
b_offset
b_offset_f
c_offset
c_offset_f
sigma_a
sigma_a_f
a[0]
[1]
[2]
[3]
[4]
[5]
[6]
[7]
[8]
[9]
[10]
[11]
[12]
[13]
[14]
[15]
[16]
[17]
[18]
[19]
a_f[0]
[1]
[2]
[3]
[4]
[5]
[6]
[7]
[8]
[9]
[10]



In [14]:
```
pm.summary(hierarchical_trace).sort_values('r_hat')
```

```
Got error No model on context stack. trying to find log_likelihood in translat
ion.
/usr/local/lib/python3.9/site-packages/arviz/data/io_pymc3_3x.py:98: FutureWar
ning: Using `from_pymc3` without the model will be deprecated in a future rele
ase. Not using the model will return less accurate and less useful results. Ma
ke sure you use the model argument or call from_pymc3 within a model context.
  warnings.warn(
```

Out[14]:

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_h
mu_a	0.497	0.313	-0.078	1.103	0.009	0.006	1228.0	1717.0	
a[18]	0.200	0.730	-1.066	1.664	0.016	0.012	1976.0	2561.0	
a[17]	1.574	0.360	0.864	2.204	0.008	0.005	2291.0	3180.0	
a[16]	-1.000	0.418	-1.813	-0.245	0.009	0.007	2007.0	2583.0	
a[15]	1.069	0.283	0.597	1.659	0.004	0.003	4173.0	3139.0	
...	...	...	...	...	...	...	...	...	
a_offset_f[4]	-0.021	0.553	-1.063	0.976	0.010	0.009	3364.0	2945.0	
a_offset_f[3]	-0.033	0.513	-0.960	0.954	0.009	0.008	3446.0	2591.0	
a_offset_f[2]	-1.317	0.611	-2.535	-0.271	0.013	0.009	2091.0	3023.0	
a_offset_f[11]	0.356	0.498	-0.532	1.319	0.009	0.007	3180.0	3043.0	
eps	1.015	0.034	0.950	1.075	0.000	0.000	4755.0	2762.0	

93 rows × 9 columns

Question 2

Continuing our adventures in classifying wine, in this section you will be applying (pseudo)-Bayesian Model Averaging with logistic regression.

1. First, revisit your HW2 and calculate the misclassification rate and the cross tabs for 3 variable models that used flat priors that performed best on the testing data. You will have 1 model for LDA and 1 model for QDA.

2. Next, use the provided BMA Wine class to fit a Bayesian Model Averaged logistic regression using the training data. Output the variable inclusion probabilities using the summary() function and interpret.

3. Finally, obtain the miss-classification rates and cross tabs for the BMA model applied to the training data and the testing data. Compare the performance of the BMA models to the performance of the best LDA and QDA models.

LDA, QDA, and Bayesian Model Averging Models

In [15]:
```
from mpmath import mp
import numpy as np
import pandas as pd
import statsmodels.api as sm
from statsmodels.tools import add_constant
from itertools import combinations
mp.dps = 50

#This class is based on the BMA class provided by Bill Basener in: https://www.
#It has been modified to allow for multinomial regression (logistic regression
#Specifically, I've hardcoded the model as a 3 category multinomial regression,
class BMA_Wine:

    def __init__(self, y, X, **kwargs):
        # Setup the basic variables.
        self.y = y
        self.X = X
        self.names = list(X.columns)
        self.nRows, self.nCols = np.shape(X)
        self.likelihoods = mp.zeros(self.nCols,1)
        self.likelihoods_all = {}
        self.coefficients_mp = mp.zeros(self.nCols,2)
        self.coefficients = np.zeros((self.nCols, 2))
        self.probabilities = np.zeros(self.nCols)
        # Check the max model size. (Max number of predictor variables to use i
        # This can be used to reduce the runtime but not doing an exhaustive sa
        if 'MaxVars' in kwargs.keys():
            self.MaxVars = kwargs['MaxVars']
        else:
            self.MaxVars = self.nCols
        # Prepare the priors if they are provided.
        # The priors are provided for the individual regressor variables.
        # The prior for a model is the product of the priors on the variables i
        if 'Priors' in kwargs.keys():
            if np.size(kwargs['Priors']) == self.nCols:
                self.Priors = kwargs['Priors']
            else:
                print("WARNING: Provided priors error.  Using equal priors inst
                print("The priors should be a numpy array of length equal tot h
                self.Priors = np.ones(self.nCols)
        else:
            self.Priors = np.ones(self.nCols)
        if 'Verbose' in kwargs.keys():
            self.Verbose = kwargs['Verbose']
        else:
            self.Verbose = False
```

```
        if 'RegType' in kwargs.keys():
            self.RegType = kwargs['RegType']
        else:
            self.RegType = 'LS'

    def fit(self):
        # Perform the Bayesian Model Averaging

        # Initialize the sum of the likelihoods for all the models to zero.
        # This will be the 'normalization' denominator in Bayes Theorem.
        likelighood_sum = 0

        # To facilitate iterating through all possible models, we start by iter
        # the number of elements in the model.
        max_likelihood = 0
        for num_elements in range(1,self.MaxVars+1):

            if self.Verbose == True:
                print("Computing BMA for models of size: ", num_elements)

            # Make a list of all index sets of models of this size.
            Models_current = list(combinations(list(range(self.nCols)), num_ele

            # Occam's window - compute the candidate models to use for the next
            # Models_previous: the set of models from the previous iteration th
            # Models_next:     the set of candidate models for the next iterati
            # Models_current:  the set of models from Models_next that can be c
            #                    to a model from Models_previous


            # Iterate through all possible models of the given size.
            for model_index_set in Models_current:

                # Compute the linear regression for this given model.
                model_X = self.X.iloc[:,list(model_index_set)]

                model_regr = sm.MNLogit(self.y, model_X).fit(disp=0)

                # Compute the likelihood (times the prior) for the model.
                model_likelihood = mp.exp(-model_regr.bic/2)*np.prod(self.Prior

                if self.Verbose == True:
                    pass
                    #print("Model Variables:",model_index_set,"likelihood=",mod
                self.likelihoods_all[str(model_index_set)] = model_likelihood

                # Add this likelihood to the running tally of likelihoods.
                likelighood_sum = mp.fadd(likelighood_sum, model_likelihood)
                # Add this likelihood (times the priors) to the running tally
                # of likelihoods for each variable in the model.
                for idx, i in zip(model_index_set, range(num_elements)):
                    self.likelihoods[idx] = mp.fadd(self.likelihoods[idx], mode
                    for j in np.arange(model_regr.params.shape[1]):

                        self.coefficients_mp[idx,j] = mp.fadd(self.coefficients
                max_likelihood = np.max([max_likelihood,model_likelihood]) # ge



        # Divide by the denominator in Bayes theorem to normalize the probabili
```

```
        # sum to one.
        self.likelighood_sum = likelighood_sum
        for idx in range(self.nCols):
            self.probabilities[idx] = mp.fdiv(self.likelihoods[idx],likelighooc
            for j in range(2):
                self.coefficients[idx,j] = mp.fdiv(self.coefficients_mp[idx,j],

        # Return the new BMA object as an output.
        return self


    def predict_MAP(self,true_class, data):
        data = np.asarray(data)
        result = np.zeros((data.shape[0],3))
        temp = sm.MNLogit(true_class, exog=np.asarray(data))
        result = temp.predict(params = self.coefficients, exog = np.asarray(dat
        result = pd.DataFrame(result, columns= ["A", "C", "F"])
        res_MAP = result.idxmax(axis=1)
        to_return = pd.DataFrame({'TrueClass':true_class, 'MAP':res_MAP})
        return to_return


    def misclass_rate(self, true_class, data):
        maps = self.predict_MAP(true_class, data)


        maps['Mis_class'] = maps['MAP']  == maps['TrueClass']

        mis_class =  1 - maps['Mis_class'].mean()

        return mis_class
    def misclass_xtabs(self, true_class, data):
        maps = self.predict_MAP(true_class, data)

        xtabs = pd.crosstab(maps['MAP'], maps['TrueClass'])
        return xtabs

    def summary(self):
        # Return the BMA results as a data frame for easy viewing.
        df = pd.DataFrame([self.names, list(self.probabilities), list(self.coef
             ["Variable Name", "Probability", "Avg. Coefficient"]).T
        return df
```

In [16]:

```
from scipy.stats import multivariate_normal

class LDA():
    def __init__(self, dataset, class_var, priors = None):
        n_class = len(dataset[class_var].unique())
        if priors is None:
            priors = np.repeat(1/n_class, n_class)
        self.priors =  np.asarray(priors)
        self.means = dataset.groupby(class_var).mean()
        self.sigma = dataset.cov()
        self.class_var = class_var
        self.training_data = dataset
    def predict_probs(self, data = None):
        if data is None:
            data = self.training_data
        data_temp = data.drop(self.class_var, axis = 1)
```

```
        dens_list = []
        col_names = []
        for ind, row in self.means.iterrows():
            col_names.append(ind)
            dens_list.append(multivariate_normal.pdf(data_temp, mean = np.asarr
        dens_list = pd.DataFrame(np.transpose(np.vstack(dens_list)),columns= co
        dens_list = dens_list.mul(self.priors, axis=1)
        dens_list = dens_list.div(dens_list.sum(axis=1), axis=0)
        dens_list['True Class'] = data[self.class_var]
        return dens_list
    def predict_MAP(self, data = None):
        if data is None:
            data = self.training_data
        dens_list = self.predict_probs(data).drop('True Class', axis = 1)
        map_list = dens_list.idxmax(axis = 1)
        maps = {'MAP Class': map_list}
        maps = pd.DataFrame(maps)
        maps['True Class'] = data[self.class_var]
        return maps
    def misclass_rate(self, data = None):
        if data is None:
            data = self.training_data
        maps = self.predict_MAP(data = data)


        maps['Mis_class'] = maps['MAP Class']  == maps['True Class']

        mis_class =  1 - maps['Mis_class'].mean()

        return mis_class
    def misclass_xtabs(self, data = None):
        if data is None:
            data = self.training_data
        maps = self.predict_MAP(data = data)

        xtabs = pd.crosstab(maps['MAP Class'], maps['True Class'])
        return xtabs
    def misclass_pairplot(self, data = None):
        if data is None:
            data = self.training_data
        maps = self.predict_MAP(data = data)
        temp_dat = data.copy(deep = True)
        temp_dat['Mis-Classified'] = maps['MAP Class']  != maps['True Class']
        plot = sns.pairplot(temp_dat,hue="Mis-Classified", height = 1.5, aspect
        return plot
```

In [17]:

```
class QDA(LDA):
    def __init__(self, dataset, class_var, priors = None):
        n_class = len(dataset[class_var].unique())
        if priors is None:
            priors = np.repeat(1/n_class, n_class)
        self.priors = np.asarray(priors)
        self.means = dataset.groupby(class_var).mean()
        gb = dataset.groupby(class_var)
        self.sigma = {x: gb.get_group(x).cov() for x in gb.groups}
        self.class_var = class_var
        self.training_data = dataset
    def predict_probs(self, data = None):
        if data is None:
```

```
            data = self.training_data
        data_temp = data.drop(self.class_var, axis = 1)
        dens_list = []
        col_names = []
        for ind, row in self.means.iterrows():
            col_names.append(ind)
            dens_list.append(multivariate_normal.pdf(data_temp, mean = np.asarr
        dens_list = pd.DataFrame(np.transpose(np.vstack(dens_list)),columns= cc
        dens_list = dens_list.mul(self.priors, axis=1)
        dens_list = dens_list.div(dens_list.sum(axis=1), axis=0)
        dens_list['True Class'] = data[self.class_var]
        return dens_list
    def predict_MAP(self, data = None):
        if data is None:
            data = self.training_data
        dens_list = self.predict_probs(data).drop('True Class', axis = 1)
        map_list = dens_list.idxmax(axis = 1)
        maps = {'MAP Class': map_list}
        maps = pd.DataFrame(maps)
        maps['True Class'] = data[self.class_var]
        return maps
```

Wine data Load

In [18]:
```
train = pd.read_csv("../HW2/data/whitewine-training-ds6040.csv")
train.head()
```

Out[18]:

	fixed.acidity	volatile.acidity	citric.acid	residual.sugar	chlorides	free.sulfur.dioxide	total.
0	0.183032	-0.088263	0.223977	2.798612	-0.038083	0.549032	
1	-0.640290	0.206999	0.056475	-0.946679	0.142355	-1.246502	
2	1.476825	0.010158	0.558982	0.092590	0.187465	-0.319775	
3	-0.757907	0.403840	-1.451043	0.112199	-0.038083	-0.319775	
4	0.183032	-0.088263	0.223977	2.798612	-0.038083	0.549032	

In [19]:
```
test = pd.read_csv("../HW2/data/whitewine-testing-ds6040.csv")
test.head()
```

Out[19]:

	fixed.acidity	volatile.acidity	citric.acid	residual.sugar	chlorides	free.sulfur.dioxide	total.
0	2.074706	2.095780	-0.370398	-0.442945	-0.924246	-0.963499	
1	0.752463	0.031237	0.597571	-1.000570	-0.077157	-1.447827	
2	-1.050596	-0.170181	0.032922	-1.000570	0.016964	-1.750531	
3	-1.892024	1.541879	-1.822352	-1.040400	-0.783065	-1.750531	
4	-0.569781	-0.371600	-0.370398	1.010863	0.252266	-0.176466	

Question 2 - 1

First, revisit your HW2 and calculate the misclassification rate and the cross tabs for 3 variable models that used flat priors that performed best on the testing data. You will have 1 model for LDA and 1 model for QDA.

In [20]:
```
import itertools
error_rates_combinations = []
for i in itertools.combinations(train.columns[:-1], 3):
    combination = list(i) + ["wine_quality"]
    flat_priors_lda = LDA(train[combination], 'wine_quality')
    training_error = flat_priors_lda.misclass_rate()
    testing_error = flat_priors_lda.misclass_rate(data = test[combination])
    error_rates_combinations.append([",".join(combination[:-1]), training_error
df_error_rates_combinations_lda = pd.DataFrame(error_rates_combinations)
df_error_rates_combinations_lda.columns = ["combinations",\
                                           "training error",\
                                           "testing error"]
```

In [21]:
```
df_error_rates_combinations_lda.sort_values('testing error')[0:1]
```

Out[21]:

	combinations	training error	testing error
77	volatile.acidity,density,alcohol	0.510699	0.487791

In [22]:
```
import itertools
error_rates_combinations = []
for i in itertools.combinations(train.columns[:-1], 3):
    combination = list(i) + ["wine_quality"]
    flat_priors_qda = QDA(train[combination], 'wine_quality')
    training_error = flat_priors_qda.misclass_rate()
    testing_error = flat_priors_qda.misclass_rate(data = test[combination])
    error_rates_combinations.append([",".join(combination[:-1]), training_error
df_error_rates_combinations_qda = pd.DataFrame(error_rates_combinations)
df_error_rates_combinations_qda.columns = ["combinations",\
                                           "training error",\
                                           "testing error"]
```

In [23]:
```
df_error_rates_combinations_qda.sort_values('testing error')[0:1]
```

Out[23]:

	combinations	training error	testing error
70	volatile.acidity,free.sulfur.dioxide,alcohol	0.463814	0.451163

Question 2 - 2

Next, use the provided BMA Wine class to fit a Bayesian Model Averaged logistic regression using the training data. Output the variable inclusion probabilities using the summary() function and interpret.

In [24]:
```
bma_model = BMA_Wine(train['wine_quality'], add_constant(train.iloc[:,:-1]), Re
```

In [25]:
```
bma_model.fit()
```

```
Computing BMA for models of size:  1
Computing BMA for models of size:  2
Computing BMA for models of size:  3
Computing BMA for models of size:  4
Computing BMA for models of size:  5
Computing BMA for models of size:  6
Computing BMA for models of size:  7
Computing BMA for models of size:  8
Computing BMA for models of size:  9
Computing BMA for models of size:  10
Computing BMA for models of size:  11
Computing BMA for models of size:  12
```

Out[25]:
```
<__main__.BMA_Wine at 0x158b07460>
```

In [26]:
```
bma_model.summary()
```

Out[26]:

	Variable Name	Probability	Avg. Coefficient
0	const	1.0	[3.5548050259163233, 2.5617193891280996]
1	fixed.acidity	0.050937	[0.006081316362694351, 0.012836499255148485]
2	volatile.acidity	1.0	[0.252005132198438, 0.9405138587589457]
3	citric.acid	0.00038	[1.7243943761242395e-05, 1.3338342052969118e-05]
4	residual.sugar	1.0	[-0.4297881784368994, -0.8224412258831187]
5	chlorides	0.000432	[5.931264746435702e-05, 6.510663234552933e-05]
6	free.sulfur.dioxide	0.237846	[-0.06733716611614582, -0.09003553653546773]
7	total.sulfur.dioxide	0.002822	[0.000523484644990109, 0.0008579730874492004]
8	density	0.238925	[0.17396196112903814, 0.2939245486737622]
9	pH	0.035454	[-0.008840094487387816, -0.012972530705769082]
10	sulphates	0.127061	[-0.006841192294011102, -0.030589318593530715]
11	alcohol	1.0	[-0.815932614640545, -2.0316664028216285]

From the summary we can see two numeric columns, Probability and Avg. Coefficient.

The probability is the probability that the predictor (variable) will be included in the true model. The average coefficient is the average coefficient for that predictor in the final true model. One interesting thing to notice is that the probability of volatile.acidity, residual.sugar, and alcohol all were one. From the BMA model, LDA, and QDA all three models utilized the volatile.acidity, and alcohol predictors.

Question 2 - 3

Finally, obtain the miss-classification rates and cross tabs for the BMA model applied to the training data and the testing data. Compare the performance of the BMA models to the performance of the best LDA and QDA models.

In [27]:
```
bma_model.misclass_rate(train['wine_quality'], add_constant(train.iloc[:,:-1]))
```

Out[27]:
```
0.2709251101321586
```

In [28]:
```
bma_model.misclass_xtabs(train['wine_quality'], add_constant(train.iloc[:,:-1])
```

Out[28]:

TrueClass	A	C	F
MAP			
C	98	1845	528
F	2	233	472

In [29]:
```
bma_model.misclass_rate(test['wine_quality'], add_constant(test.iloc[:,:-1]))
```

Out[29]:
```
0.30988372093023253
```

In [30]:
```
bma_model.misclass_xtabs(test['wine_quality'], add_constant(test.iloc[:,:-1]))
```

Out[30]:

TrueClass	A	C	F
MAP			
C	79	904	357
F	1	96	283

In [31]:
```
## LDA Model
lda_model = LDA(train[["volatile.acidity",
                       "density",
                       "alcohol",
                       "wine_quality"]],
                      'wine_quality')
print(lda_model.misclass_rate(data = test[["volatile.acidity",
                       "density",
                       "alcohol",
                       "wine_quality"]]))
lda_model.misclass_xtabs(data = test[["volatile.acidity",
                       "density",
                       "alcohol",
                       "wine_quality"]])
```

```
0.4877906976744186
```

Out[31]:

True Class	A	C	F
MAP Class			
A	55	335	48
C	17	390	156
F	8	275	436

In [32]:
```
## QDA Model
qda_model = QDA(train[["volatile.acidity",
                       "free.sulfur.dioxide",
                       "alcohol",
                       "wine_quality"]],
                      'wine_quality')
print(qda_model.misclass_rate(data = test[["volatile.acidity",
```

```
                        "free.sulfur.dioxide",
                        "alcohol",
                        "wine_quality"]]))
qda_model.misclass_xtabs(data = test[["volatile.acidity",
                        "free.sulfur.dioxide",
                        "alcohol",
                        "wine_quality"]])
```

```
0.4511627906976744
```

Out[32]:

True Class	A	C	F
MAP Class			
A	48	299	35
C	26	437	146
F	6	264	459

With flat priors, the misclassiciation rate for the QDA and LDA model were significantly higher than the BMA model. From the cross tabs, we can see where the rise in misclass rate is for the QDA and LDA model. In the BMA model there were no mapping to the A class and a lot of mappings to the C class. From the data we know this class had the most data points, which brings down the misclassification rates. If the main purpose was to put more emphasis on the A class classification the BMA model would not be the way to go, but for a general classifcation problem, and from the metrics shown above the BMA model seemed to be a good model in those aspects.

In [ ]:

```
```