Part II C3 Petrology - Practical 20: MORB Global Systematics - ANSWERS

Simon Matthews (sm905@cam.ac.uk), Lent 2023

In this practical you will extract and analyse geochemical and geophysical data for the global mid-ocean ridge system with the aim of identifying what the primary controls on the composition of MORB are. The practical will also demonstrate how we can use computers and relatively simple python scripts to handle large datasets efficiently and perform otherwise time consuming calculations.

All the practicals in this part of the course are contained in Jupyter Notebooks which can be run in the cloud on a service called myBinder. This means they should work on *any* computer, provided it has a modern web browser. The notebooks allow you to run code alongside text and images and allow you to conveniently save the output from the code. Whole scientific papers (e.g. https://mybinder.org/v2/gh/kaylai/vesical-binder/HEAD?filepath=Manuscript.ipynb) can be written in Jupyter Notebooks! **However, all the work you do is stored on a server in the cloud and will disappear when you shut the window, so make sure you download a copy!** To do this, click `File` in the top-left corner, and find `Download` on the menu that appears.

A summary of the practical will be available online after the end of the practical class, but don't rely on this, use the demonstrators! Some of the tasks are deliberately designed to require discussions with other students and the demonstrators.

1. Retrieving and plotting bathymetry data using pyGMT

There are multiple ways of interacting with these datasets but we will use a package called pyGMT, which is a python interface to the GMT (Generic Mapping Tools) software. GMT is widely used in Earth and Marine Sciences. You can read more about the bathymetry data here.

First we must import the `pygmt` package so the notebook has access to its code and mapping know-how:

In [2]:
```
import pygmt
```

Now we will extract some of the data GMT is packaged with and assign it to variables so that we can use it more conveniently.

The first dataset is a grid of altitude/bathymetry covering the whole planet

In [3]:
```
grid = pygmt.datasets.load_earth_relief()
grid
```

Out[3]:
xarray.DataArray 'elevation' (**lat**: 181, **lon**: 361)

array([[2862. , 2862. , 2862. , ..., 2862. , 2862. , 2862.],
[3073.5, 3072.5, 3071.5, ..., 3075. , 3074.5, 3073.5],
[3140. , 3141.5, 3143. , ..., 3138.5, 3139.5, 3140.],
...,
[-3791.5, -3710. , -3678.5, ..., -3843. , -3832. , -3791.5],
[-2233.5, -2180. , -2142. , ..., -2309. , -2275.5, -2233.5],
[-4198. , -4198. , -4198. , ..., -4198. , -4198. , -4198.]],
dtype=float32)

▾ Coordinates:

lon	(lon)	float64	-180.0 -179.0 ... 179.0 180.0	
lat	(lat)	float64	-90.0 -89.0 -88.0 ... 89.0 90.0	

▾ Attributes:

long_name :	Earth elevation relative to the geoid
units :	meters
vertical_datum :	EGM96
horizontal_datu...	WGS84

The second dataset records the location of spreading ridges:

In [4]:
```
points = pygmt.datasets.load_sample_data(name='ocean_ridge_points')
points
```


Out[4]:

	longitude	latitude
0	-32.2971	37.4118
1	-32.3909	37.1394
2	-32.6448	37.1760
3	-32.7066	37.0349
4	-32.9468	37.0643
...	...	...
4141	-77.6612	-45.7623
4142	-76.6762	-45.6414
4143	-75.8527	-45.5741
4144	-75.7026	-46.0728
4145	-75.5410	-46.5777

4146 rows × 2 columns

Now we can combine both of these datasets to extract the bathymetry along the ridge axes:

In [5]:
```
track = pygmt.grdtrack(points=points, grid=grid, newcolname="bathymetry")
track
```

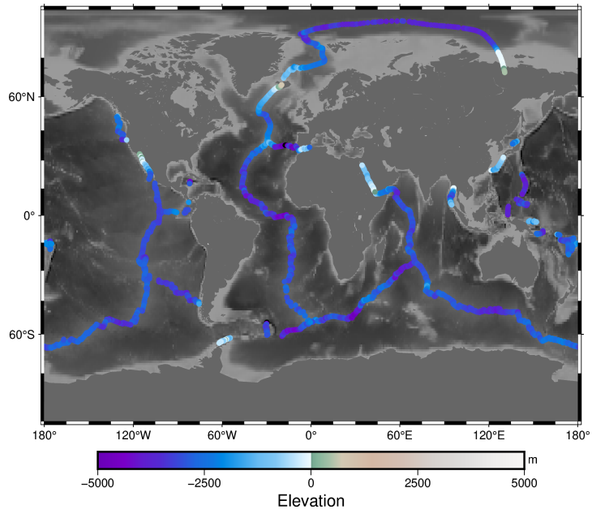
Out[5]:

	longitude	latitude	bathymetry
0	-32.2971	37.4118	-1691.475635
1	-32.3909	37.1394	-1764.711568
2	-32.6448	37.1760	-1899.662953
3	-32.7066	37.0349	-1956.343874
4	-32.9468	37.0643	-2048.090389
...	...	...	...
4141	-77.6612	-45.7623	-2905.372615
4142	-76.6762	-45.6414	-2930.035331
4143	-75.8527	-45.5741	-2455.096540
4144	-75.7026	-46.0728	-1926.114648
4145	-75.5410	-46.5777	-1548.465330

4146 rows × 3 columns

Let's see what this looks like on a map:

In [6]:
```
# Create a GMT figure and store it in variable fig:
fig = pygmt.Figure()

# Plot the earth relief grid on Cylindrical Stereographic projection
fig.basemap(region="g", frame=True, projection="Cyl_stere/0/45/15c") # this sets the projection and the region a
fig.grdimage(grid=grid, cmap="gray") # the plots up the ocean bathymetry with a boring gray colour scale
fig.coast(land="#666666") # Mask land areas

# Generate a color map for the bathymetry
pygmt.makecpt(cmap='terra', series=[-5000, 5000])

# Plot using circles (c) of 0.15 cm, the sampled bathymetry points
# Points are colored using elevation values (normalized for visual purposes)
fig.plot(
    x=track.longitude,
    y=track.latitude,
    style="c0.15c",
    cmap=True, # Use the colormap generated above
    fill=track.bathymetry, # Colour according to bathymetry
)
fig.colorbar(frame=["a2500", "x+lElevation", "y+lm"])
fig.show()
```



You should visit the pygmt manual pages and look at the information about projections and setting the region to get a quick feel for what the `projection` and `region` arguments mean. Please ask if this does not make sense.

Most of the plots we make in these practicals (and probably when you use python to make plots in general) uses a library called **matplotlib**. Unfortunately, pyGMT works quite differently to matplotlib, and (in my opinion) is far less intuitive than matplotlib. It does produce much higher quality maps and provides us with convenient access to bathymetry datasets, so it is worth the extra hassle!

2. Extract and plot chemical data for oceanic basalts from PETDB

In the practical folder is a spreadsheet containing a large compilation of chemical data collected from rocks at present-day spreading ridges. The spreadsheet was downloaded from the PETDB website. Have a look at the website and try running a search for some part of the Earth, and have a look at the data that is returned.

Q2.1: What categories of rocks in the database are likely to be the most and least useful for understanding magmatic processes? Ask a demonstrator if you're not sure where to find this information.

🟢 *The database contains metamorphic and sedimentary rocks which are unlikely to be helpful. The full range of igneous rocks may be useful for understanding magmatic processes in general, but the ones that are easiest to interpret mantle processes from are the mafic and ultramafic volcanics.<*

The **pandas** python package provides a toolkit for importing and manipulating large datasets:

In [7]:
```
import pandas as pd
```

Adding `as pd` after the import command means that we can access all the functions using the name `pd`, saving us from typing out `pandas` each time!

We can use pandas to read in an Excel file (telling it to skip the first 5 rows which hold a title and some information from PETDB):

In [8]:
```
petdb = pd.read_excel("spreading_ridge_rocks.xlsx", header=5)
```

It's always a good idea to check the data import has worked correctly, so we can look at the very top of the table:

In [9]:
```
petdb.head()
```


Out[9]:

	SAMPLE ID	IGSN	EXPEDITION ID	REFERENCES	LATITUDE	LONGITUDE	LOC PREC	MIN ELEVATION	MAX ELEVATION	TECTONIC SETTING	...	FEO
0	*VG1586	NaN	nr	MELSON, 2003	5.60	61.89	0.01	-2360.0	-2820.0	SPREADING_CENTER	...	NaN
1	*VG968	NaN	nr	ITO, 1980	28.90	-43.32	0.01	-3692.0	-3701.0	SPREADING_CENTER	...	NaN
2	078_I5.27N	NaN	nr	COHEN, 1982	5.45	61.83	0.01	NaN	NaN	SPREADING_CENTER	...	10.06
3	09N039W-UDM-HOST	NaN	nr	SOBOLEV, 1993	9.00	-39.50	0.10	NaN	NaN	SPREADING_CENTER	...	NaN
4	108DR1	NaN	nr	PRINZHOFER, 1989	11.43	-104.00	0.01	NaN	NaN	SPREADING_CENTER	...	NaN

5 rows × 34 columns

While this gives us a convenient view, it skips some of the columns. To see all of the columns that exist in the table:

In [10]:
```
petdb.columns
```

Out[10]:
```
Index(['SAMPLE ID', 'IGSN', 'EXPEDITION ID', 'REFERENCES', 'LATITUDE',
       'LONGITUDE', 'LOC PREC', 'MIN ELEVATION', 'MAX ELEVATION',
       'TECTONIC SETTING', 'MIN AGE', 'AGE', 'MAX AGE', 'METHOD',
       'ANALYZED MATERIAL', 'ROCK TYPE', 'ROCK NAME', 'MINERAL', 'SIO2',
       'TIO2', 'AL2O3', 'CR2O3', 'FE2O3', 'FE2O3T', 'FEO', 'FEOT', 'NIO',
       'MNO', 'MGO', 'CAO', 'NA2O', 'K2O', 'P2O5', 'LOI'],
      dtype='object')
```

Importantly for us, we have the latitude, longitude, and the oxide composition (in wt%).

Q2.2: How have the columns FE2O3, FE2O3T, FEO, and FEOT been populated in the spreadsheet, and what is their meaning?

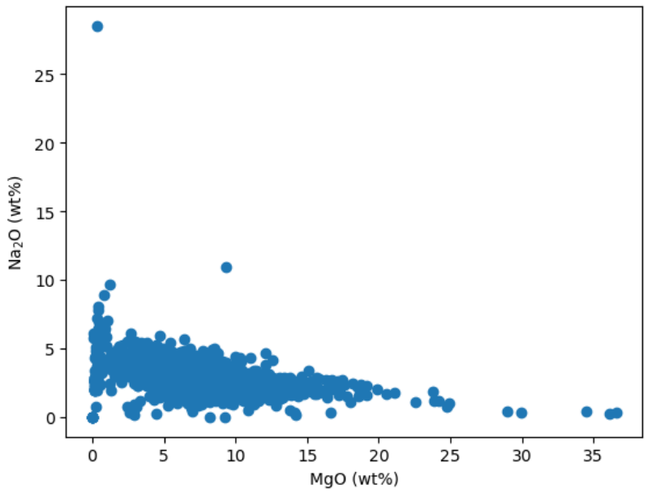
🟢 *Sometimes (rarely) samples will have had both their ferrous and ferric iron determined, so their records will be populated with both FeO and Fe$_2$O$_3$. Many of the standard analytical techniques we use to obtain rock compositions don't provide information on the oxidation state of Fe, so more often it is lumped all together under the assumption that it is all FeO (FeOT) or Fe_2O_3 (Fe2O3T). In basalts these assumptions are almost never correct, but it ensures the data is reported with clarity.*

To plot the data in this part of the practical we will use matplotlib, which is conventially imported like this:

In [11]:
```
import matplotlib.pyplot as plt
```

We can plot the whole dataset at once, which will allow us to check data quality as well as get an overview of it. Here is an example of one plot using matplotlib, with a little bit of customisation:

In [12]:
```
fig, ax = plt.subplots(dpi=100) # Make a figure, and put a set of axes on it, set the display resolution as 350 l

# Add some scatter points to the axis
ax.scatter(petdb.MGO, petdb.NA2O)

# Set the axis labels (notice how you make a subscript appear):
ax.set_xlabel('MgO (wt%)')
ax.set_ylabel('Na$_2$O (wt%)')

# Display the plot
plt.show()
```



The things you can do to customise how matplotlib displays a plot are virtually endless. Have a look at the matplotlib examples gallery to get an idea for this.

Q2.3: Do you notice any issues with the data?

You might want to change the plot settings to zoom into parts of the dataset. You can do this with the lines `ax.set_xlim(..., ...)` *and* `ax.set_ylim(..., ...)`.

🟢 *Some of the records are missing MgO values and are plotting at 0.0. Some samples have anomalously high Na_2O. Sometimes it is worth filtering the dataset for such issues before using it, but it shouldn't cause a problem for us today.*

We should bear these issues in mind when we start using the dataset. They might be things we can ignore, or we might want to filter the data before we use it.

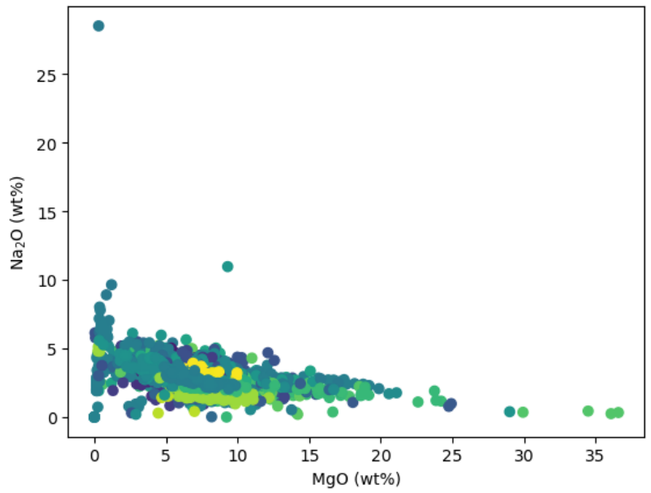
Q2.4: Is there any pattern you notice in the global dataset when MgO is plotted against Na$_2$O?

🟢 *Na_2O is negatively correlated with MgO. As rocks become more evolved their MgO concentration drops (it is compatible), but since Na_2O is generally incompatible it increases.*

Q2.5: Produce versions of the plot coloured for longitude and latitude

To add a colour scale to the points you can use the argument `c=...` *where you enter the sequence of data in place of "...". Try working out how to do this from the code above, and ask a demonstrator if you get stuck.*

In [14]:
```
fig, ax = plt.subplots(dpi=100) # Make a figure, and put a set of axes on it, set the display resolution as 350 l

# Add some scatter points to the axis
ax.scatter(petdb.MGO, petdb.NA2O, c=petdb.LATITUDE)

# Set the axis labels (notice how you make a subscript appear):
ax.set_xlabel('MgO (wt%)')
ax.set_ylabel('Na$_2$O (wt%)')

# Display the plot
plt.show()
```

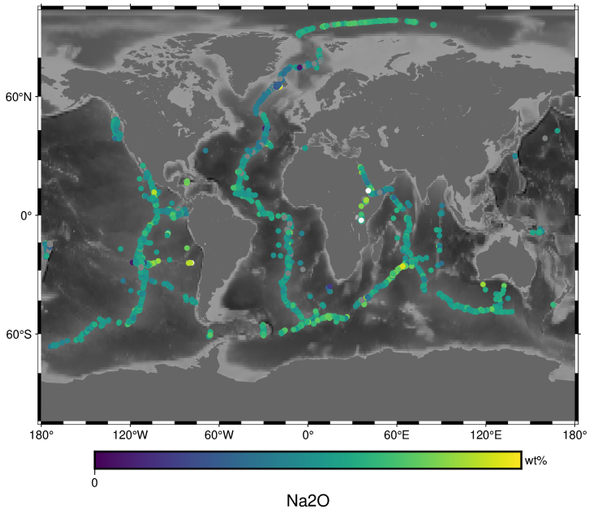



Though its tricky to pick out all the data on these plots, you should notice that there is some systematic variation between where the data sits in Na$_2$O vs MgO and its location.

Q2.6: Perhaps a better way of viewing it is to colour code the symbols on a map. Repurpose the code that produced the map above to make such a map.

The colormap used above is great for plotting topography and bathymetry, but not so great for other variables. Find a better one by using the GMT colour scale reference

In [18]:
```
# Create a GMT figure and store it in variable fig:
fig = pygmt.Figure()

# Plot the earth relief grid on Cylindrical Stereographic projection
fig.basemap(region="g", frame=True, projection="Cyl_stere/0/45/15c") # this sets the projection and the region ar
fig.grdimage(grid=grid, cmap="gray") # the plots up the ocean bathymetry with a boring gray colour scale
fig.coast(land="#666666") # Mask land areas

# Generate a color map for the bathymetry
pygmt.makecpt(cmap='viridis', series=[0, 5])

# Plot using circles (c) of 0.15 cm, the sampled bathymetry points
# Points are colored using elevation values (normalized for visual purposes)
fig.plot(
    x=petdb.LONGITUDE,
    y=petdb.LATITUDE,
    style="c0.15c",
    cmap=True, # Use the colormap generated above
    fill=petdb.NA2O, # Colour according to bathymetry
)
fig.colorbar(frame=["a2500", "x+lNa2O", "y+lwt%"])
fig.show()
```



Q2.7 Is there any coherence between Na_2O content and bathymetry?

🟢 *Yes, at least to some degree. The swell around Iceland is associated with low Na_2O, for example.*

3. Controls on the Na$_2$O concentration in MORB

You should have found some systematic pattern between Na_2O and bathymetry. If you didn't, speak to a demonstrator. This section will explore this in a bit more detail and in a more rigorous fashion.

These relationships between the major element composition of MORB and geophysical indicators of the extent of melting (such as crustal thickness or depth of seafloor at the spreading ridge axis) were used by Charlie Langmuir and co-workers to argue for the presence of a global array in MORB composition that is controlled by mantle properties (e.g. Klein & Langmuir, JGR, 1987).

In order to isolate chemical variations associated with mantle, rather than crustal, processes these authors devised a chemical property known as Na_8. Take a look at their Figure 1a and the surrounding text to see how this property is defined and used.

Q3.1: Briefly describe why comparison of Na_8 between different parts of the spreading ridge system should provide a clearer picture of mantle melting variations than the use of Na_2O from above.

🟢 *Correcting the raw Na_2O contents back to a predicted value at 8 wt% MgO will reduce the variation that is linked to fractional crystallisation, meaning that the signal of Na_8 variation will be more clearly linked to variation in mantle processes, including mantle melting.*

Q3.2: Why might it be better to average over rift segments rather than using individual rock compositions and the bathymetry at the point they were collected?

An individual rift segment is likely to sample melts from a broad region of the mantle beneath. Any variability in Na_8 is likely to be related to melting and melt transport, for example incomplete mixing of near-fractional melts (see later lectures for more on this). However, on the level of a segment, we can get a good idea of the average Na_8, and therefore get a value that is more representative of the accumulated fractional melts, which is much easier for us to interpret and model! Likewise, small scale variation in bathymetry is unlikely to be related to the mantle.

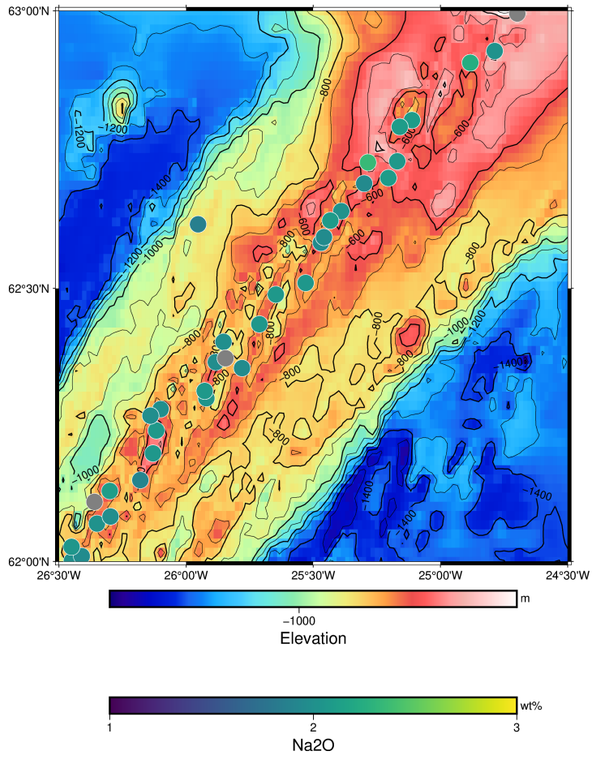
In the rest of the practical we will use the up-to-date datasets we imported above to produce a new version of the Na_8 -- bathymetry array. I will demonstrate how we might do this for one segment first...

To start with we will look at the Reykjanes Ridge, just to the south of Iceland. We will use a version of the plots above (combined with some data filtering) to extract representative values for the bathymetry and Na_8.

First, we can plot a map of the bathymetry for this segment. Notice we can use a higher resolution version of the bathymetry grid. To see the range of resolutions that GMT can call on, look at the documentation page here. We will also plot the locations of the rock samples in the database for the segment. To do this we must filter the database by location:

In [89]:
```
## In PetDB, longitude goes from -180 to +180.
petsam = petdb[(petdb.LONGITUDE > -26.5) & (petdb.LONGITUDE < -24.5) & (petdb.LATITUDE > 62.) & (petdb.LATITUDE <
```

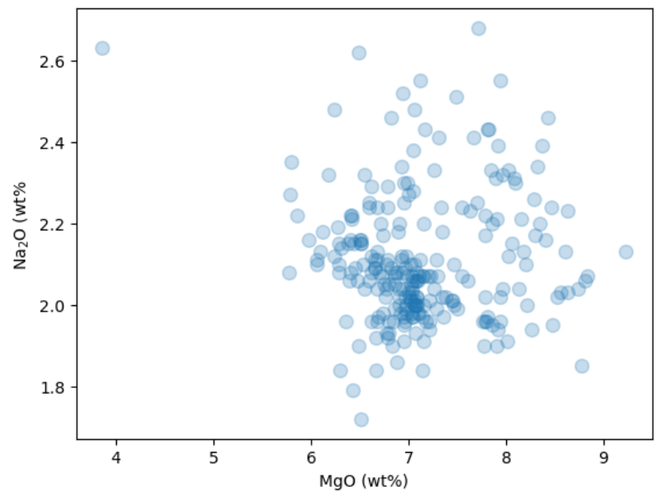
In [90]:
```
## first of all, make another data grid using a subset of the global data.
## The region defines the longitude and latitude bounds in the order WESN
## The resolution is 01m - which is 1 arcminute grid spacing. This should be sufficient
grid = pygmt.datasets.load_earth_relief(resolution="01m", region=[-26.5, -24.5, 62, 63])



fig = pygmt.Figure()
fig.grdimage(grid=grid, projection="M15c", frame="a", cmap="haxby") # haxby is a colorscale that is OK for oceans
fig.grdcontour(grid=grid) # plot up contours
fig.colorbar(frame=["a1000", "x+lElevation", "y+lm"])
pygmt.makecpt(cmap="viridis", series=[1., 3.], background = "i") # color scale for Na2O - change series limits
# Plot using circles (c) of 0.5 cm, the sampled bathymetry points

fig.plot(
    x=petsam.LONGITUDE,
    y=petsam.LATITUDE,
    style="c0.5c",
    pen="white",
    cmap=True,
    fill=petsam.NA2O, # fill circle according color-scale
)


fig.colorbar(frame=["a1.0", "x+lNa2O", "y+lwt%"],position="JBC+o0c/4c")
fig.show()
```



So, this has generated a contoured and coloured bathymetry chart of the region of interest, along with the position of available basalt samples, these colour-coded by their Na_2O contents. You can put the map in context by comparing to a global plot from GMT, the GMRT website or even Google Earth.

Now we need to estimate Na_8 for the segment, which we could also by eye from a plot:

In [91]:
```
f, a = plt.subplots()

a.scatter(petsam.MGO, petsam.NA2O, alpha=0.25, s=70)

a.set_xlabel('MgO (wt%)')
a.set_ylabel('Na$_2$O (wt%')

plt.show()
```

Notice on the plot above that each symbol is slightly transparent. I did this to see how dense the data is- there are many points overlapping with each other in a big cluster.

Q3.3: Estimate (by eye) a value for Na_8 for this data.

This is a little subjective. From the Klein and Langmuir paper you looked at earlier in the practical we know how we would expect Na_2O to vary with MgO during crystal fractionation. There's a higher density of rocks around a region that would fit with such a trend, and it looks like it would meet a Na_2O content of 1.9 wt% at MgO = 8 wt%. The Na_8 value in this case would be 1.9 wt%.

We could also do this a bit more rigorously by regressing the data to MgO = 8 wt%. A really simple way to do this is to run a linear regression over the whole dataset. Of course, someone has already written code that can do this in python, and made it available in a python package SciPy, which we can import here:

In [92]:
```
from scipy.stats import linregress
import numpy as np
```

In this case I import a specific function from the library, rather than the whole library (as we did for pandas, for example). I also imported the library numpy, which provides lots of useful mathematical operations, and lets us define matrices etc.

We can look up how to use the linregress function with a handy Jupyter Lab trick:

In [93]:
```
linregress?
```

```
Signature: linregress(x, y=None, alternative='two-sided')
Docstring:
Calculate a linear least-squares regression for two sets of measurements.

Parameters
----------
x, y : array_like
    Two sets of measurements.  Both arrays should have the same length.  If
    only `x` is given (and ``y=None``), then it must be a two-dimensional
    array where one dimension has length 2.  The two sets of measurements
    are then found by splitting the array along the length-2 dimension. In
    the case where ``y=None`` and `x` is a 2x2 array, ``linregress(x)`` is
    equivalent to ``linregress(x[0], x[1])``.
alternative : {'two-sided', 'less', 'greater'}, optional
    Defines the alternative hypothesis. Default is 'two-sided'.
    The following options are available:

    * 'two-sided': the slope of the regression line is nonzero
    * 'less': the slope of the regression line is less than zero
    * 'greater':  the slope of the regression line is greater than zero

    .. versionadded:: 1.7.0

Returns
-------
```

```
result : ``LinregressResult`` instance
    The return value is an object with the following attributes:

    slope : float
        Slope of the regression line.
    intercept : float
        Intercept of the regression line.
    rvalue : float
        The Pearson correlation coefficient. The square of ``rvalue``
        is equal to the coefficient of determination.
    pvalue : float
        The p-value for a hypothesis test whose null hypothesis is
        that the slope is zero, using Wald Test with t-distribution of
        the test statistic. See `alternative` above for alternative
        hypotheses.
    stderr : float
        Standard error of the estimated slope (gradient), under the
        assumption of residual normality.
    intercept_stderr : float
        Standard error of the estimated intercept, under the assumption
        of residual normality.

See Also
--------
scipy.optimize.curve_fit :
    Use non-linear least squares to fit a function to data.
scipy.optimize.leastsq :
    Minimize the sum of squares of a set of equations.

Notes
-----
Missing values are considered pair-wise: if a value is missing in `x`,
the corresponding value in `y` is masked.

For compatibility with older versions of SciPy, the return value acts
like a ``namedtuple`` of length 5, with fields ``slope``, ``intercept``,
``rvalue``, ``pvalue`` and ``stderr``, so one can continue to write::

    slope, intercept, r, p, se = linregress(x, y)

With that style, however, the standard error of the intercept is not
available.  To have access to all the computed values, including the
standard error of the intercept, use the return value as an object
with attributes, e.g.::

    result = linregress(x, y)
    print(result.intercept, result.intercept_stderr)

Examples
--------
>>> import matplotlib.pyplot as plt
>>> from scipy import stats
>>> rng = np.random.default_rng()

Generate some data:

>>> x = rng.random(10)
>>> y = 1.6*x + rng.random(10)

Perform the linear regression:

>>> res = stats.linregress(x, y)

Coefficient of determination (R-squared):

>>> print(f"R-squared: {res.rvalue**2:.6f}")
R-squared: 0.717533

Plot the data along with the fitted line:

>>> plt.plot(x, y, 'o', label='original data')
>>> plt.plot(x, res.intercept + res.slope*x, 'r', label='fitted line')
>>> plt.legend()
>>> plt.show()

Calculate 95% confidence interval on slope and intercept:

>>> # Two-sided inverse Students t-distribution
>>> # p - probability, df - degrees of freedom
```



```
>>> from scipy.stats import t
>>> tinv = lambda p, df: abs(t.ppf(p/2, df))

>>> ts = tinv(0.05, len(x)-2)
>>> print(f"slope (95%): {res.slope:.6f} +/- {ts*res.stderr:.6f}")
slope (95%): 1.453392 +/- 0.743465
>>> print(f"intercept (95%): {res.intercept:.6f}"
...       f" +/- {ts*res.intercept_stderr:.6f}")
intercept (95%): 0.616950 +/- 0.544475
File:      ~/opt/anaconda3/lib/python3.9/site-packages/scipy/stats/_stats_mstats_common.py
Type:      function
```

Let's try running it:

In [94]:
```
linregress(petsam.MGO, petsam.NA2O)
```

Out[94]:
```
LinregressResult(slope=nan, intercept=nan, rvalue=nan, pvalue=nan, stderr=nan, intercept_stderr=nan)
```

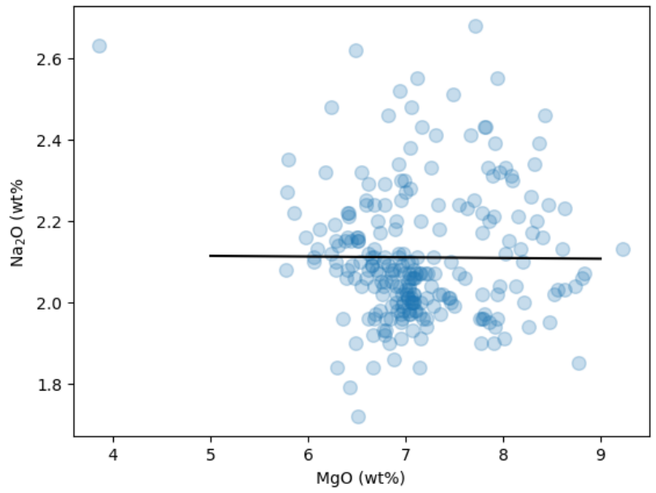
Damn! It didn't work. It turns out this is because some of the samples in the dataset have empty entried for some of the oxides. We could have filtered this out when we selected the data earlier, or we can do this when we call the function:

In [95]:
```
linreg_result = linregress(petsam.MGO[(petsam.MGO>0)&(petsam.NA2O>0)], petsam.NA2O[(petsam.MGO>0)&(petsam.NA2O>0)
linreg_result
```

Out[95]:
```
LinregressResult(slope=-0.0017564779583879213, intercept=2.122959602580933, rvalue=-0.00760025470063826, pvalue=
0.906757343366542, stderr=0.014980059401239799, intercept_stderr=0.10747636856256197)
```

We can see what this regression looks like by adding it to the plot below:

In [96]:
```
f, a = plt.subplots()

a.scatter(petsam.MGO, petsam.NA2O, alpha=0.25, s=70)

x = np.array([5, 9])
y = linreg_result[0] * x + linreg_result[1]

a.plot(x, y, c='k')

a.set_xlabel('MgO (wt%)')
a.set_ylabel('Na$_2$O (wt%')

plt.show()
```



Q3.4: How does this compare with the gradient of the lines used by Klein & Langmuir? Do you think this is a reliable way of estimating Na_8?

🟢 *The gradient is not consistent with a liquid line of descent. Instead, the fitting routine is trying to accomodate the full range of data, including the points that plot away from the trend defined by the greatest data density. Really, I have formulated the problem badly. The question we really want the answer to is what LLD best represents the data (i.e., fixing the gradient to that given in Klein & Langmuir). However, this highlights the potential pitfalls of trying to automate calculations- it is important to know exactly what the code is doing! We could instead define a function that returns a straight line with fixed gradient and used a SciPy method to find the minimum misfit. This is the way I would do it if I was analysing the whole dataset myself.*

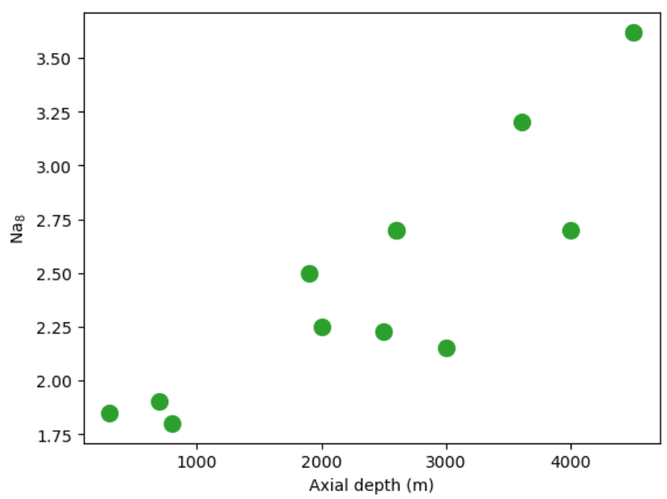
The table below lists some spreading centres which will help us build a global axial depth - Na_8 array. How you populate this is **up to you**, but I suggest you work together as a class, each taking one segment (or work in pairs).

The Reykjanes peninsula has already been filled in with my favoured values, but maybe you disagree?

Segment Name	Longitude W	Longitude E	Latitude S	Latitude N	Crustal Thickness (km)	Axial Depth (m)	Na_8
Reykjanes Example	-26.5	-24.5	62.0	63.0	12	700	1.9
EPRR10	-104.4	-104.2	9.1	10.1	6	2600	2.7
JUAN8	-129.5	-128.9	46.54	47.5	7	2500	2.23
GALA6	-97.8	-96.5	2.1	2.2	5.5	3000	2.15
GALA11	-92.1	-90.8	1.85	2.15	8	1900	2.5
MARR23	-18.4	-17.6	67.9	68.9	9.5	800	1.8
MARR24	-18.8	-18.5	66.9	67.9	12	300	1.85
MARR86	-32.4	-32.2	37	37.5	6	2000	2.25
SWIR67SWIR66	68	69.9	-26.4	-25.6	3.5	4000	2.7
GAKK15	0	2.8	84.1	84.5	4	3600	3.2
SEIR40	100.2	101.4	-47.7	-47.2	6	2600	2.7
SEIR56SEIR57	116.5	117.1	-49.4	-49.2	4	4500	3.62

I've provided code to produce a plot with the values, once you have entered them here:

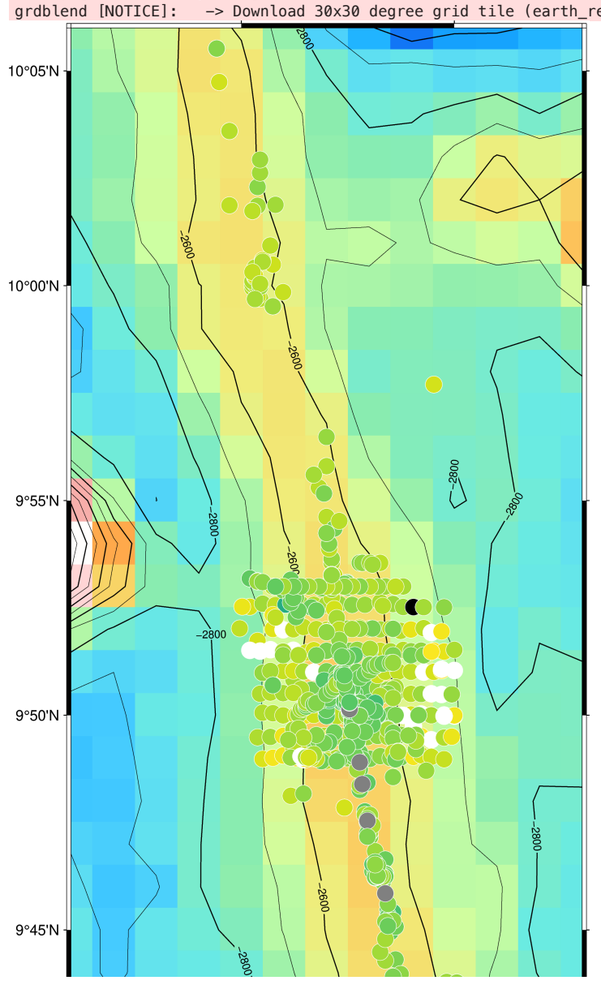
In [74]:
```
axial_depths = [700.0, 2600.0, 2500.0, 3000.0, 1900.0, 800, 300, 2000, 4000, 3600, 2600, 4500]
Na8 = [1.9, 2.7, 2.23, 2.15, 2.5, 1.8, 1.85, 2.25, 2.7, 3.2, 2.7, 3.62]


fig, ax = plt.subplots(dpi=100)

ax.scatter(axial_depths, Na8, c='C2', s=100)

ax.set_xlabel('Axial depth (m)')
ax.set_ylabel('Na$_8$')

plt.show()
```

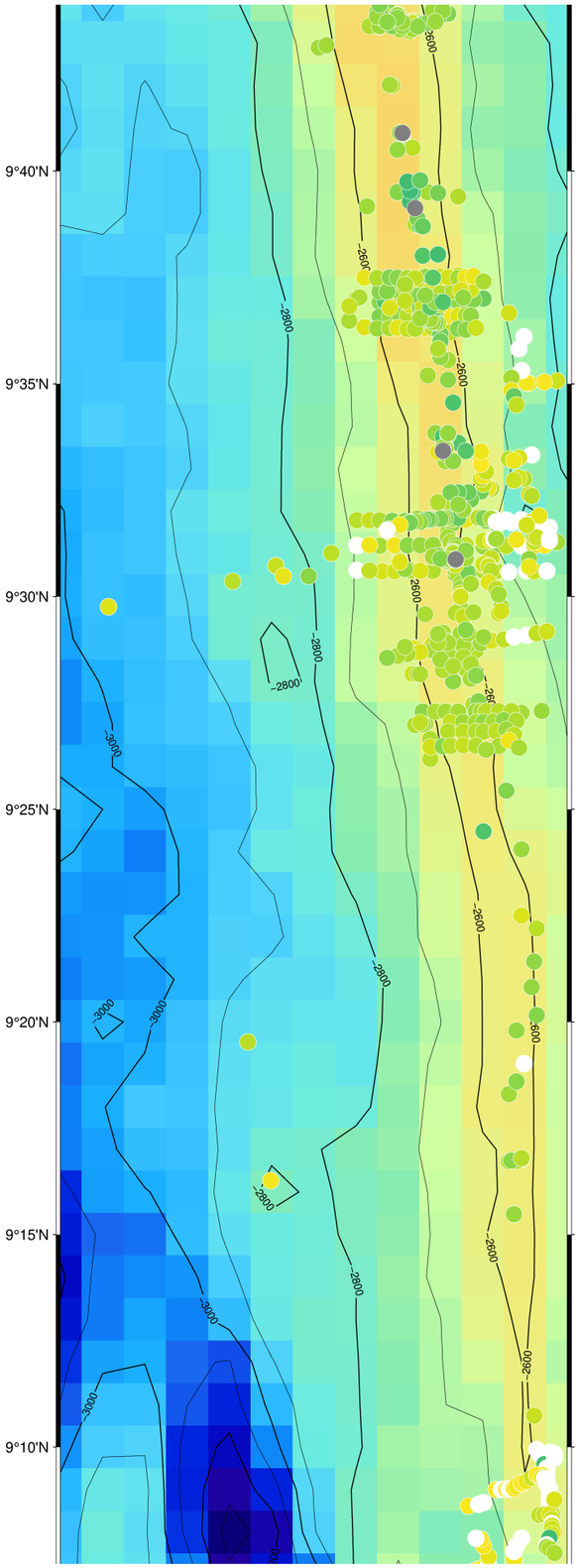


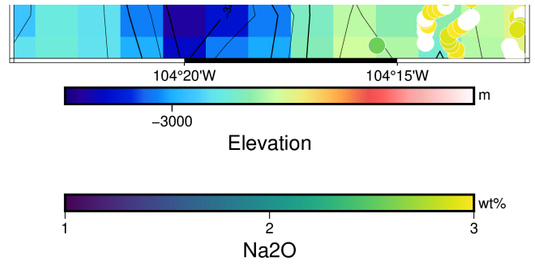
EPRR10

In [21]:
```
petsam = petdb[(petdb.LONGITUDE > -104.4) & (petdb.LONGITUDE < -104.2) & (petdb.LATITUDE > 9.1) & (petdb.LATITUDE

grid = pygmt.datasets.load_earth_relief(resolution="01m", region=[-104.4, -104.2, 9.1, 10.1])

fig = pygmt.Figure()
fig.grdimage(grid=grid, projection="M15c", frame="a", cmap="haxby") # haxby is a colorscale that is OK for oceans
fig.grdcontour(grid=grid) # plot up contours
fig.colorbar(frame=["a1000", "x+lElevation", "y+lm"])
pygmt.makecpt(cmap="viridis", series=[1., 3.], background = "i") # color scale for Na2O - change series limits
# Plot using circles (c) of 0.5 cm, the sampled bathymetry points

fig.plot(
    x=petsam.LONGITUDE,
    y=petsam.LATITUDE,
    style="c0.5c",
    pen="white",
    cmap=True,
    fill=petsam.NA2O, # fill circle according color-scale
)


fig.colorbar(frame=["a1.0", "x+lNa2O", "y+lwt%"],position="JBC+o0c/4c")
fig.show()
```

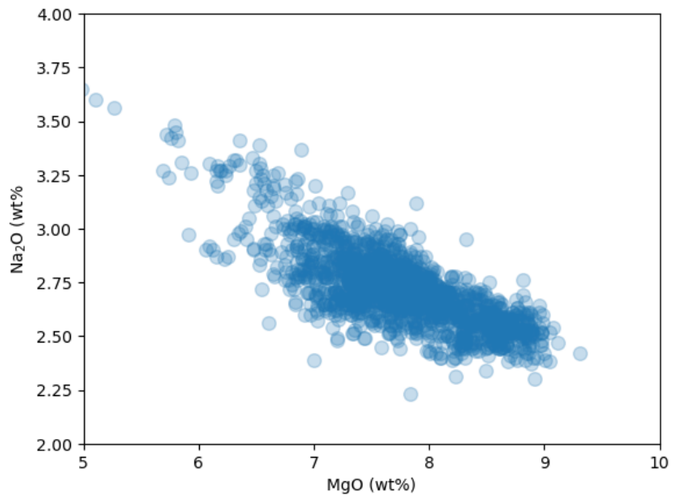
```
grdblend [NOTICE]: Remote data courtesy of GMT data server oceania [http://oceania.generic-mapping-tools.org]
grdblend [NOTICE]: SRTM15 Earth Relief at 01x01 arc minutes reduced by Gaussian Cartesian filtering (1.9 km full
width) [Tozer et al., 2019].
grdblend [NOTICE]:   -> Download 30x30 degree grid tile (earth_relief_01m_g): N00W120
```







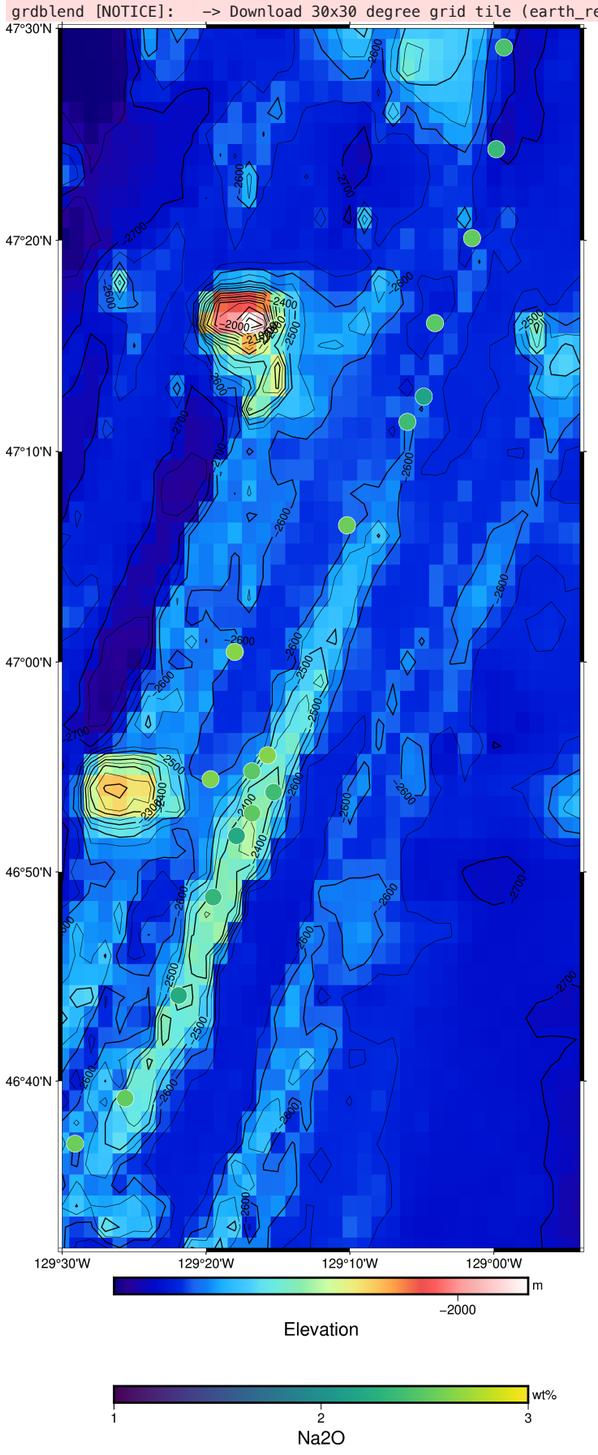
In [24]:
```
f, a = plt.subplots()

a.scatter(petsam.MGO, petsam.NA2O, alpha=0.25, s=70)

a.set_xlabel('MgO (wt%)')
a.set_ylabel('Na$_2$O (wt%')

a.set_xlim(5, 10)
a.set_ylim(2, 4)

plt.show()
```



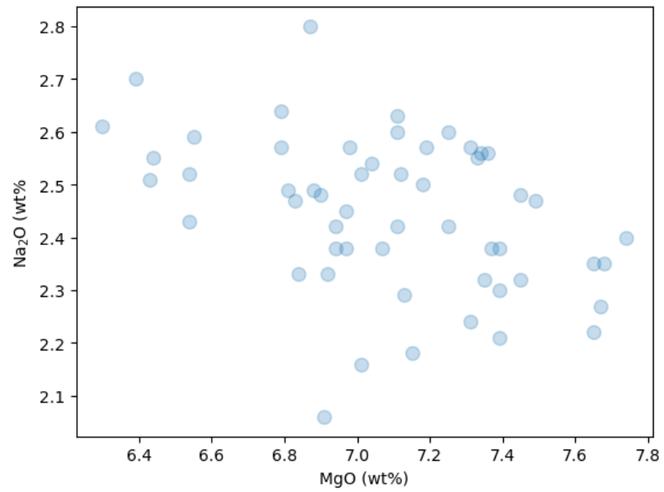
JUAN8

In [27]:
```
petsam = petdb[(petdb.LONGITUDE > -129.5) & (petdb.LONGITUDE < -128.9) & (petdb.LATITUDE > 46.54) & (petdb.LATITU

grid = pygmt.datasets.load_earth_relief(resolution="01m", region=[-129.5, -128.9, 46.54, 47.5])

fig = pygmt.Figure()
fig.grdimage(grid=grid, projection="M15c", frame="a", cmap="haxby") # haxby is a colorscale that is OK for oceans
fig.grdcontour(grid=grid) # plot up contours
fig.colorbar(frame=["a1000", "x+lElevation", "y+lm"])
pygmt.makecpt(cmap="viridis", series=[1., 3.], background = "i") # color scale for Na2O - change series limits
# Plot using circles (c) of 0.5 cm, the sampled bathymetry points

fig.plot(
    x=petsam.LONGITUDE,
    y=petsam.LATITUDE,
    style="c0.5c",
    pen="white",
    cmap=True,
    fill=petsam.NA2O, # fill circle according color-scale
)


fig.colorbar(frame=["a1.0", "x+lNa2O", "y+lwt%"],position="JBC+o0c/4c")
fig.show()
```

```
grdcut [WARNING]: (s - y_min) must equal (NY + eps) * y_inc), where NY is an integer and |eps| <= 0.0001.
grdcut [WARNING]: s reset from 46.54 to 46.5333333333
grdblend [NOTICE]: Remote data courtesy of GMT data server oceania [http://oceania.generic-mapping-tools.org]
grdblend [NOTICE]: SRTM15 Earth Relief at 01x01 arc minutes reduced by Gaussian Cartesian filtering (1.9 km full
width) [Tozer et al., 2019].
grdblend [NOTICE]:   -> Download 30x30 degree grid tile (earth_relief_01m_g): N30W150
```

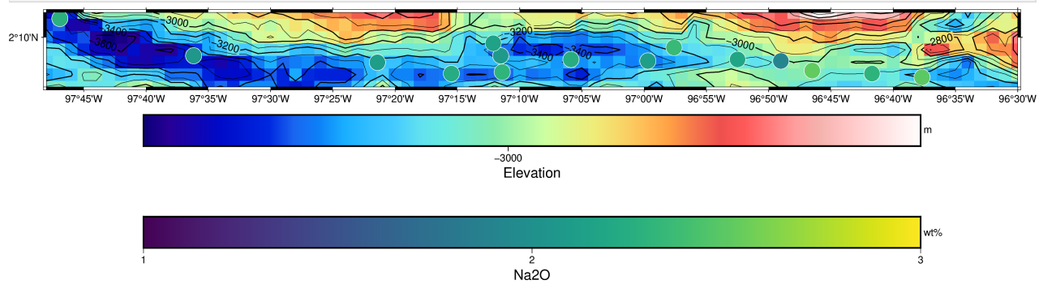


In [28]:
```
f, a = plt.subplots()

a.scatter(petsam.MGO, petsam.NA2O, alpha=0.25, s=70)

a.set_xlabel('MgO (wt%)')
a.set_ylabel('Na$_2$O (wt%')

# a.set_xlim(5, 10)
# a.set_ylim(2, 4)

plt.show()
```

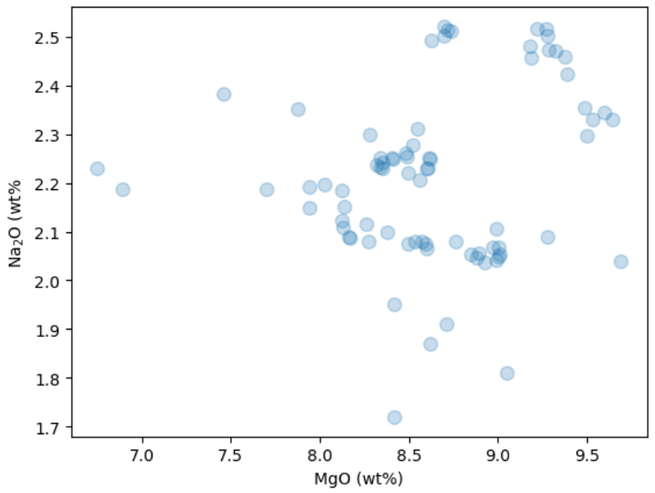


GALA6

In [73]:
```
petsam = petdb[(petdb.LONGITUDE > -97.8) & (petdb.LONGITUDE < -96.5) & (petdb.LATITUDE > 2.1) & (petdb.LATITUDE <

grid = pygmt.datasets.load_earth_relief(resolution="01m", region=[-97.8, -96.5, 2.1, 2.2])

fig = pygmt.Figure()
fig.grdimage(grid=grid, projection="M30c", frame="a", cmap="haxby") # haxby is a colorscale that is OK for oceans
fig.grdcontour(grid=grid) # plot up contours
fig.colorbar(frame=["a1000", "x+lElevation", "y+lm"])
pygmt.makecpt(cmap="viridis", series=[1., 3.], background = "i") # color scale for Na2O - change series limits
# Plot using circles (c) of 0.5 cm, the sampled bathymetry points

fig.plot(
    x=petsam.LONGITUDE,
    y=petsam.LATITUDE,
    style="c0.5c",
    pen="white",
    cmap=True,
    fill=petsam.NA2O, # fill circle according color-scale
)


fig.colorbar(frame=["a1.0", "x+lNa2O", "y+lwt%"],position="JBC+o0c/4c")
fig.show(width=1000)
```

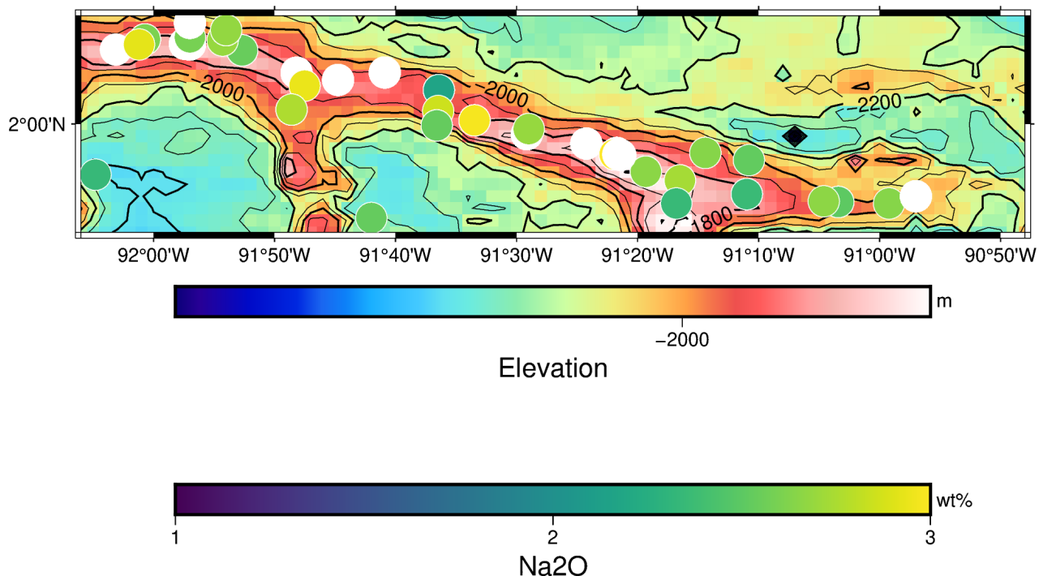


In [35]:
```
f, a = plt.subplots()

a.scatter(petsam.MGO, petsam.NA2O, alpha=0.25, s=70)

a.set_xlabel('MgO (wt%)')
a.set_ylabel('Na$_2$O (wt%')

# a.set_xlim(5, 10)
# a.set_ylim(2, 4)

plt.show()
```

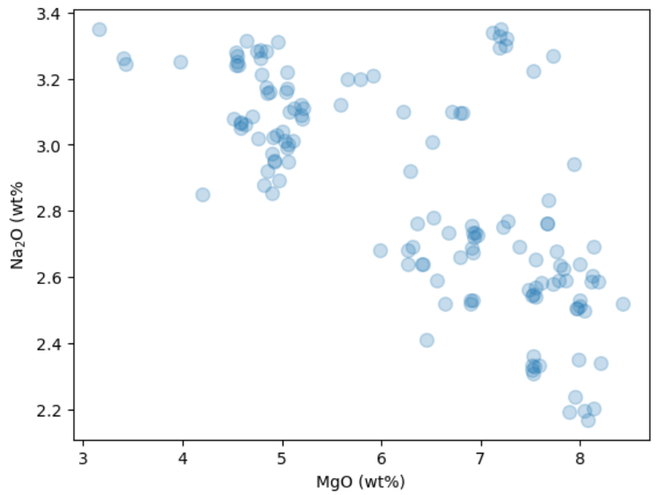


GALA11

In [40]:
```
petsam = petdb[(petdb.LONGITUDE > -92.1) & (petdb.LONGITUDE < -90.8) & (petdb.LATITUDE > 1.85) & (petdb.LATITUDE

grid = pygmt.datasets.load_earth_relief(resolution="01m", region=[-92.1, -90.8, 1.85, 2.15])

fig = pygmt.Figure()
fig.grdimage(grid=grid, projection="M15c", frame="a", cmap="haxby") # haxby is a colorscale that is OK for oceans
fig.grdcontour(grid=grid) # plot up contours
fig.colorbar(frame=["a1000", "x+lElevation", "y+lm"])
pygmt.makecpt(cmap="viridis", series=[1., 3.], background = "i") # color scale for Na2O - change series limits
# Plot using circles (c) of 0.5 cm, the sampled bathymetry points

fig.plot(
    x=petsam.LONGITUDE,
    y=petsam.LATITUDE,
    style="c0.5c",
    pen="white",
    cmap=True,
    fill=petsam.NA2O, # fill circle according color-scale
)


fig.colorbar(frame=["a1.0", "x+lNa2O", "y+lwt%"],position="JBC+o0c/4c")
fig.show(width=1000)
```

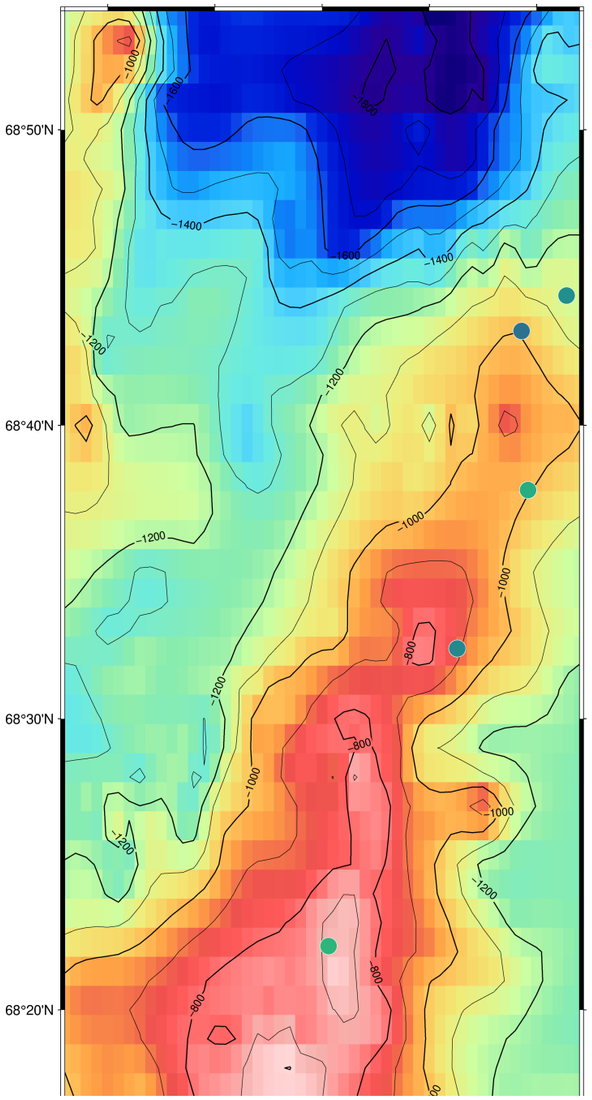



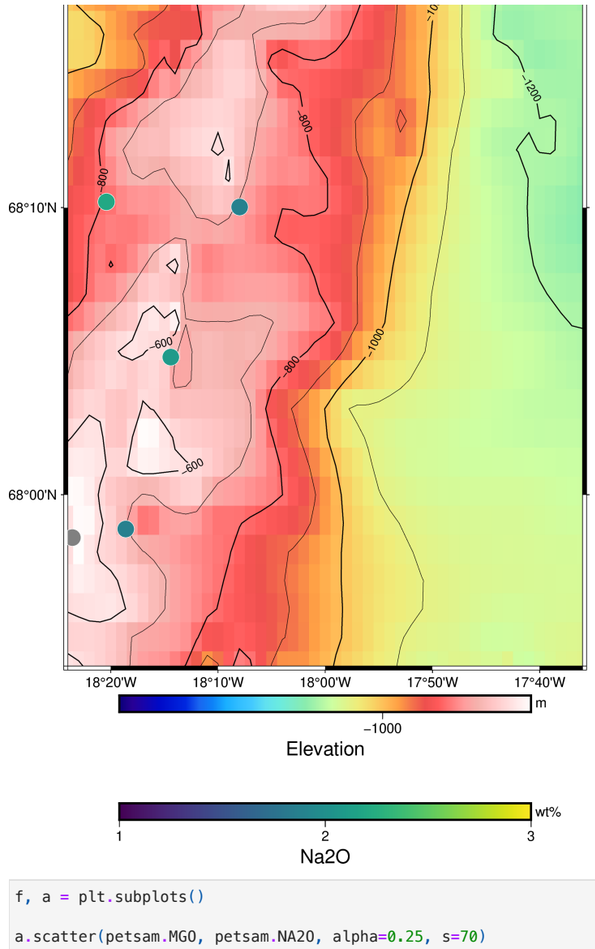
In [38]:
```
f, a = plt.subplots()

a.scatter(petsam.MGO, petsam.NA2O, alpha=0.25, s=70)

a.set_xlabel('MgO (wt%)')
a.set_ylabel('Na$_2$O (wt%')

# a.set_xlim(5, 10)
# a.set_ylim(2, 4)

plt.show()
```



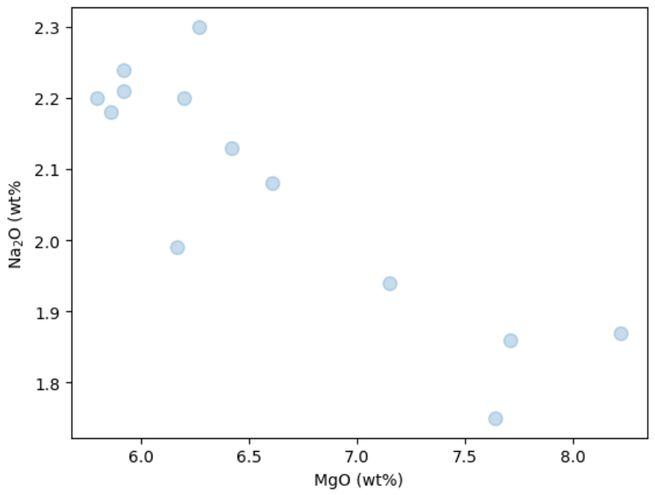
MARR23

In [97]:
```
petsam = petdb[(petdb.LONGITUDE > -18.4) & (petdb.LONGITUDE < -17.6) & (petdb.LATITUDE > 67.9) & (petdb.LATITUDE

grid = pygmt.datasets.load_earth_relief(resolution="01m", region=[-18.4, -17.6, 67.9, 68.9])

fig = pygmt.Figure()
fig.grdimage(grid=grid, projection="M15c", frame="a", cmap="haxby") # haxby is a colorscale that is OK for oceans
fig.grdcontour(grid=grid) # plot up contours
fig.colorbar(frame=["a1000", "x+lElevation", "y+lm"])
pygmt.makecpt(cmap="viridis", series=[1., 3.], background = "i") # color scale for Na2O - change series limits
# Plot using circles (c) of 0.5 cm, the sampled bathymetry points

fig.plot(
    x=petsam.LONGITUDE,
    y=petsam.LATITUDE,
    style="c0.5c",
    pen="white",
    cmap=True,
    fill=petsam.NA2O, # fill circle according color-scale
)


fig.colorbar(frame=["a1.0", "x+lNa2O", "y+lwt%"],position="JBC+o0c/4c")
fig.show()
```



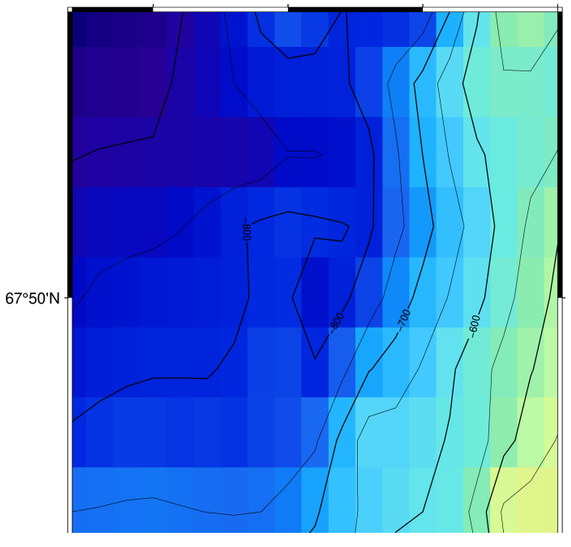


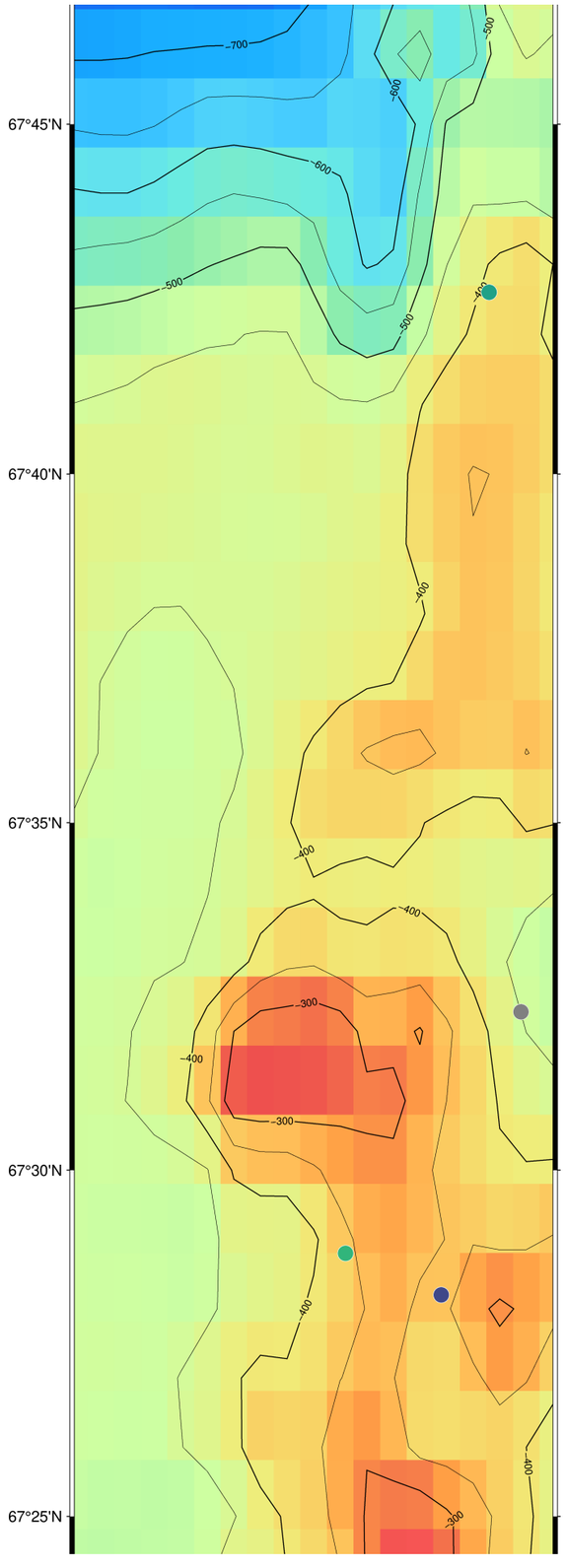
In [42]:
```
f, a = plt.subplots()

a.scatter(petsam.MGO, petsam.NA2O, alpha=0.25, s=70)

a.set_xlabel('MgO (wt%)')
a.set_ylabel('Na$_2$O (wt%')

# a.set_xlim(5, 10)
# a.set_ylim(2, 4)

plt.show()
```

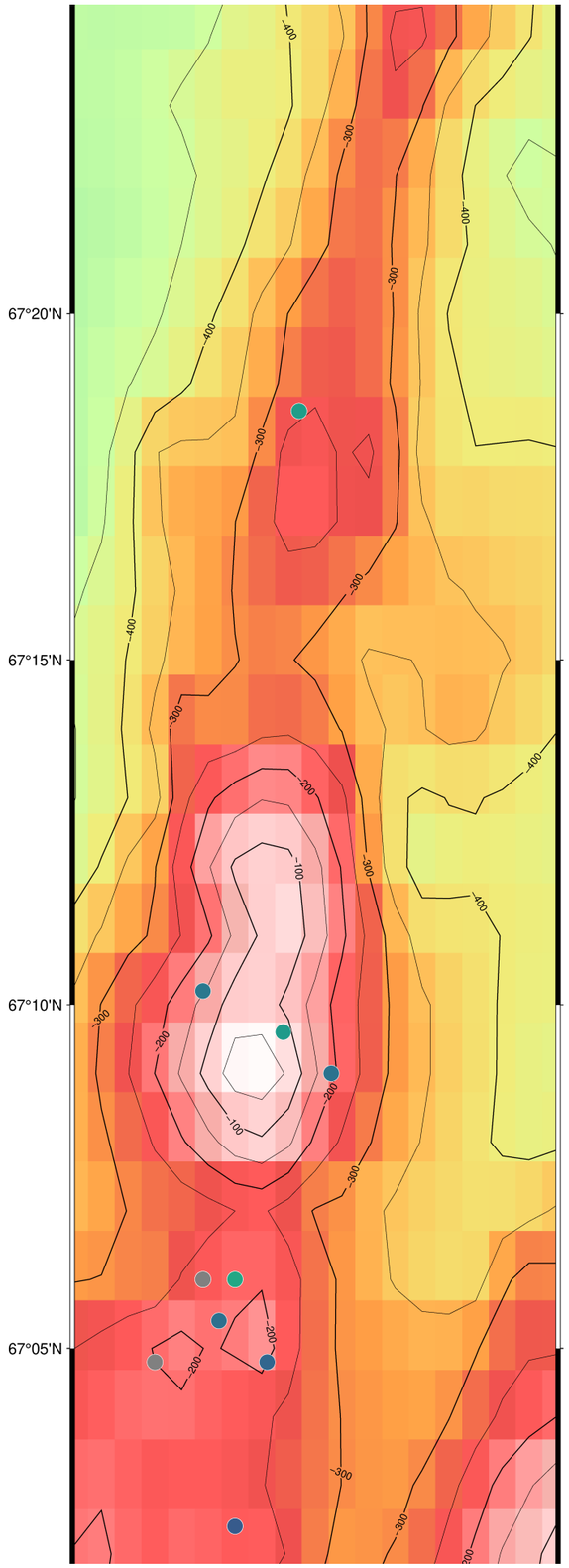


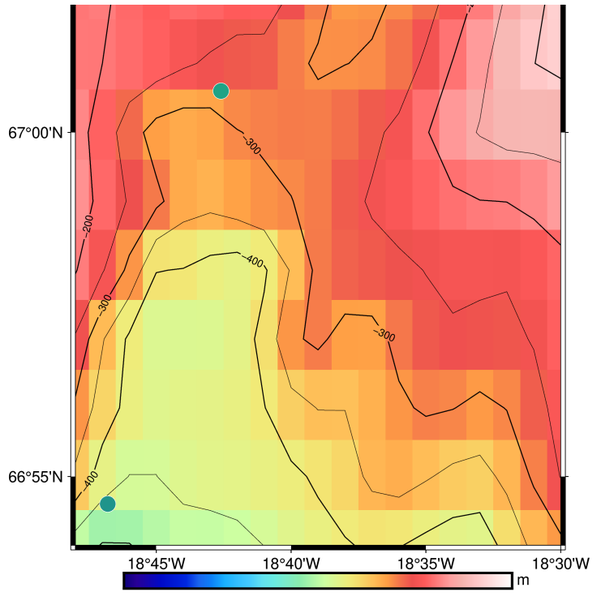
MARR24

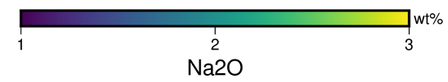
In [45]:
```
petsam = petdb[(petdb.LONGITUDE > -18.8) & (petdb.LONGITUDE < -18.5) & (petdb.LATITUDE > 66.9) & (petdb.LATITUDE

grid = pygmt.datasets.load_earth_relief(resolution="01m", region=[-18.8, -18.5, 66.9, 67.9])

fig = pygmt.Figure()
fig.grdimage(grid=grid, projection="M15c", frame="a", cmap="haxby") # haxby is a colorscale that is OK for oceans
fig.grdcontour(grid=grid) # plot up contours
fig.colorbar(frame=["a1000", "x+lElevation", "y+lm"])
pygmt.makecpt(cmap="viridis", series=[1., 3.], background = "i") # color scale for Na2O - change series limits
# Plot using circles (c) of 0.5 cm, the sampled bathymetry points

fig.plot(
    x=petsam.LONGITUDE,
    y=petsam.LATITUDE,
    style="c0.5c",
    pen="white",
    cmap=True,
    fill=petsam.NA2O, # fill circle according color-scale
)


fig.colorbar(frame=["a1.0", "x+lNa2O", "y+lwt%"],position="JBC+o0c/4c")
fig.show()
```



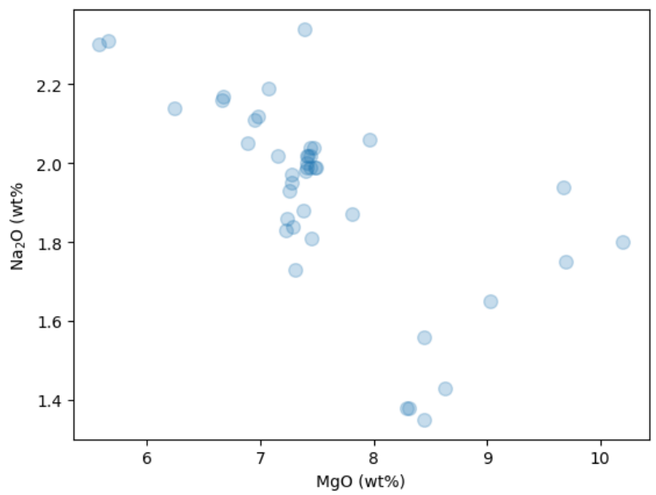








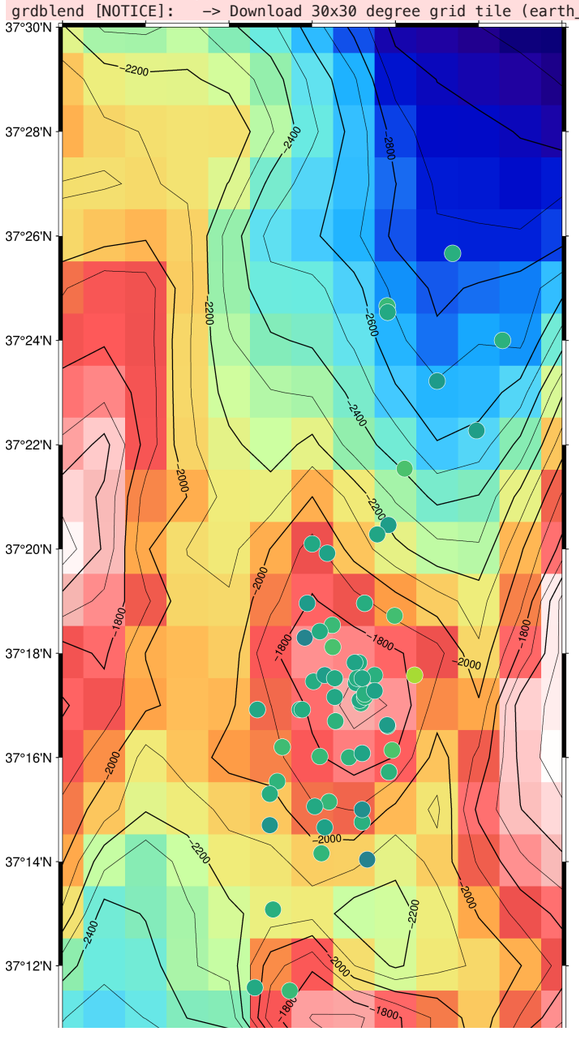
In [47]:
```
f, a = plt.subplots()

a.scatter(petsam.MGO, petsam.NA2O, alpha=0.25, s=70)

a.set_xlabel('MgO (wt%)')
a.set_ylabel('Na$_2$O (wt%')

# a.set_xlim(5, 10)
# a.set_ylim(2, 4)

plt.show()
```

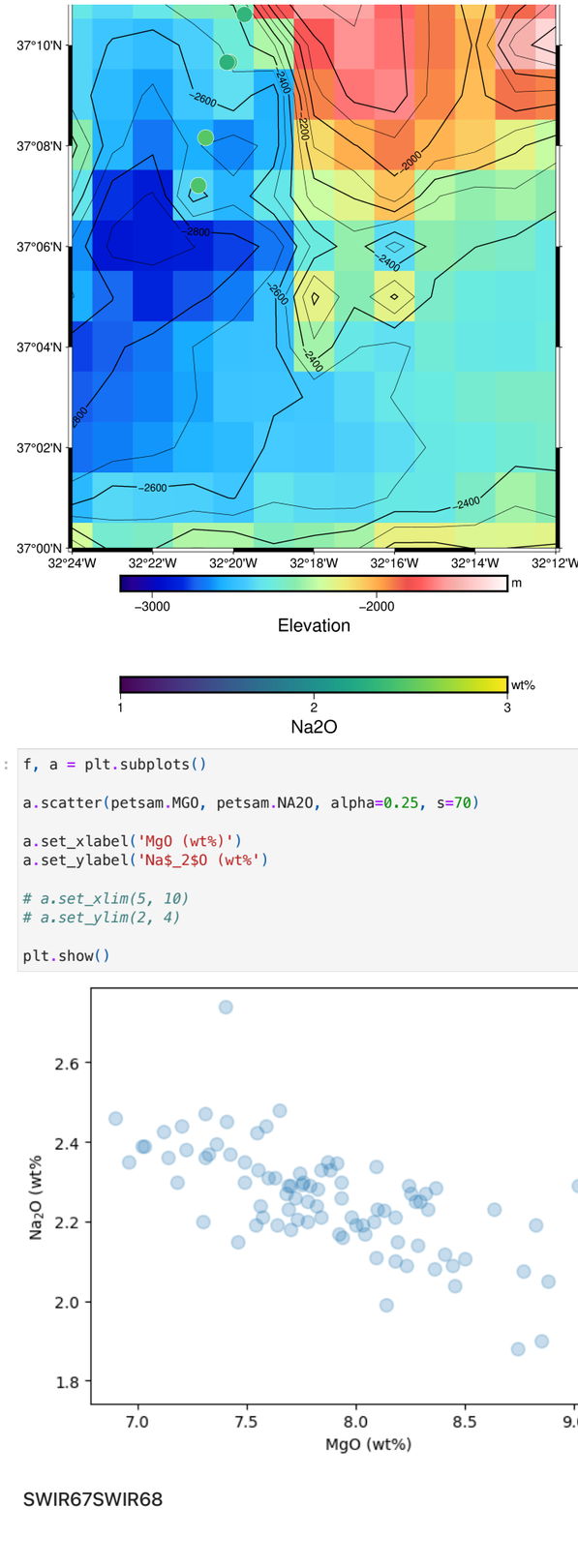


MARR86

In [49]:
```
petsam = petdb[(petdb.LONGITUDE > -32.4) & (petdb.LONGITUDE < -32.2) & (petdb.LATITUDE > 37) & (petdb.LATITUDE <

grid = pygmt.datasets.load_earth_relief(resolution="01m", region=[-32.4, -32.2, 37, 37.5])

fig = pygmt.Figure()
fig.grdimage(grid=grid, projection="M15c", frame="a", cmap="haxby") # haxby is a colorscale that is OK for oceans
fig.grdcontour(grid=grid) # plot up contours
fig.colorbar(frame=["a1000", "x+lElevation", "y+lm"])
pygmt.makecpt(cmap="viridis", series=[1., 3.], background = "i") # color scale for Na2O - change series limits
# Plot using circles (c) of 0.5 cm, the sampled bathymetry points

fig.plot(
    x=petsam.LONGITUDE,
    y=petsam.LATITUDE,
    style="c0.5c",
    pen="white",
    cmap=True,
    fill=petsam.NA2O, # fill circle according color-scale
)


fig.colorbar(frame=["a1.0", "x+lNa2O", "y+lwt%"],position="JBC+o0c/4c")
fig.show()
```

```
grdblend [NOTICE]: Remote data courtesy of GMT data server oceania [http://oceania.generic-mapping-tools.org]
grdblend [NOTICE]: SRTM15 Earth Relief at 01x01 arc minutes reduced by Gaussian Cartesian filtering (1.9 km full
width) [Tozer et al., 2019].
grdblend [NOTICE]:   -> Download 30x30 degree grid tile (earth_relief_01m_g): N30W060
```

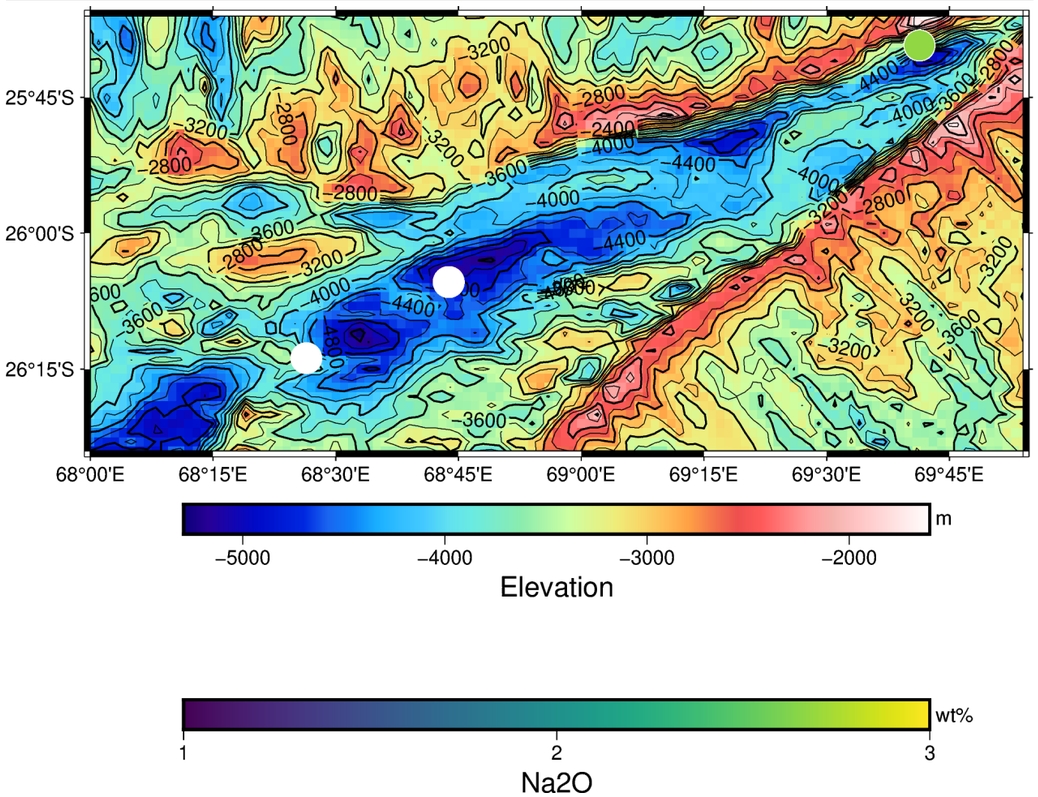




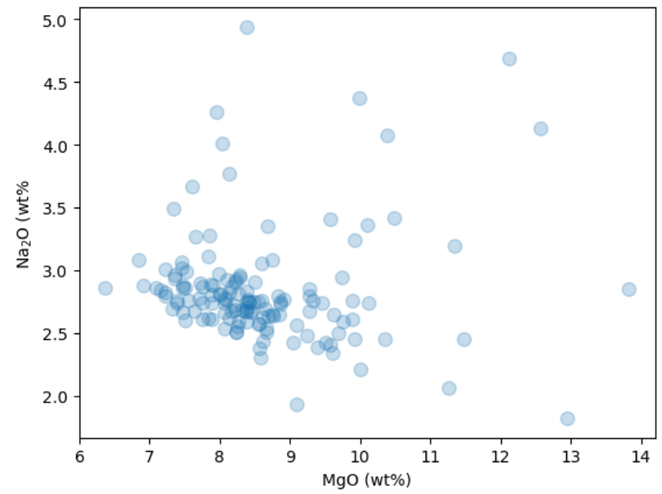
In [50]:
```
f, a = plt.subplots()

a.scatter(petsam.MGO, petsam.NA2O, alpha=0.25, s=70)

a.set_xlabel('MgO (wt%)')
a.set_ylabel('Na$_2$O (wt%')

# a.set_xlim(5, 10)
# a.set_ylim(2, 4)

plt.show()
```


SWIR67SWIR68

In [54]:
```
petsam = petdb[(petdb.LONGITUDE > 68) & (petdb.LONGITUDE < 69.9) & (petdb.LATITUDE > -26.4) & (petdb.LATITUDE < -

grid = pygmt.datasets.load_earth_relief(resolution="01m", region=[68, 69.9, -26.4, -25.6])

fig = pygmt.Figure()
fig.grdimage(grid=grid, projection="M15c", frame="a", cmap="haxby") # haxby is a colorscale that is OK for oceans
fig.grdcontour(grid=grid) # plot up contours
fig.colorbar(frame=["a1000", "x+lElevation", "y+lm"])
pygmt.makecpt(cmap="viridis", series=[1., 3.], background = "i") # color scale for Na2O - change series limits
# Plot using circles (c) of 0.5 cm, the sampled bathymetry points

fig.plot(
    x=petsam.LONGITUDE,
    y=petsam.LATITUDE,
    style="c0.5c",
    pen="white",
    cmap=True,
    fill=petsam.NA2O, # fill circle according color-scale
)


fig.colorbar(frame=["a1.0", "x+lNa2O", "y+lwt%"],position="JBC+o0c/4c")
fig.show(width=1000)
```



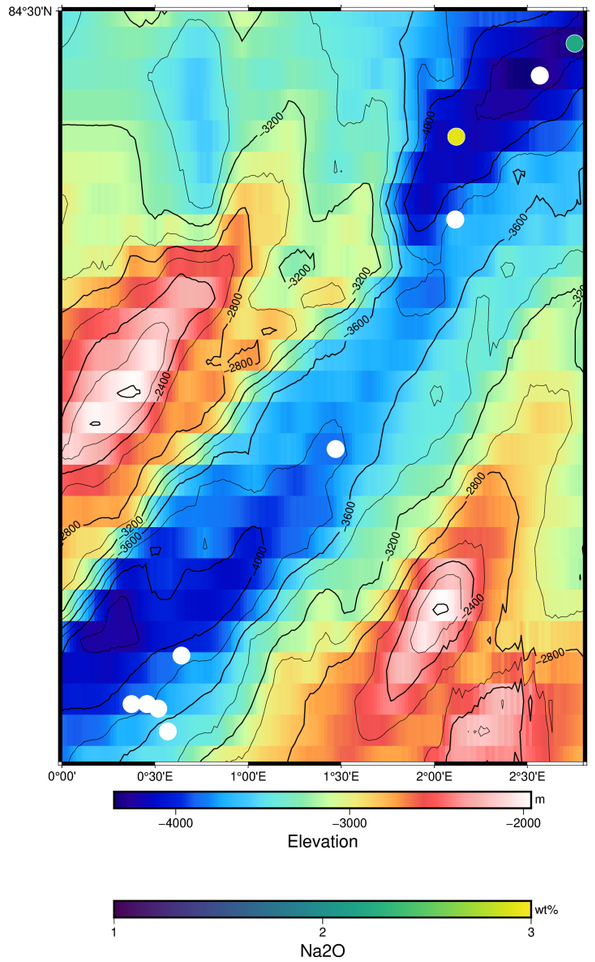
In [55]:
```
f, a = plt.subplots()

a.scatter(petsam.MGO, petsam.NA2O, alpha=0.25, s=70)

a.set_xlabel('MgO (wt%)')
a.set_ylabel('Na$_2$O (wt%')

# a.set_xlim(5, 10)
# a.set_ylim(2, 4)

plt.show()
```

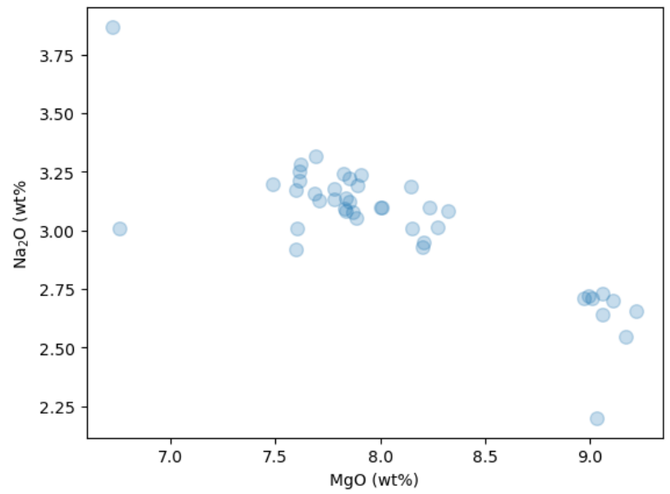
GAKK15

In [58]:
```
petsam = petdb[(petdb.LONGITUDE > 0) & (petdb.LONGITUDE < 2.8) & (petdb.LATITUDE > 84.1) & (petdb.LATITUDE < 84.5

grid = pygmt.datasets.load_earth_relief(resolution="01m", region=[0, 2.8, 84.1, 84.5])

fig = pygmt.Figure()
fig.grdimage(grid=grid, projection="M15c", frame="a", cmap="haxby") # haxby is a colorscale that is OK for oceans
fig.grdcontour(grid=grid) # plot up contours
fig.colorbar(frame=["a1000", "x+lElevation", "y+lm"])
pygmt.makecpt(cmap="viridis", series=[1., 3.], background = "i") # color scale for Na2O - change series limits
# Plot using circles (c) of 0.5 cm, the sampled bathymetry points

fig.plot(
    x=petsam.LONGITUDE,
    y=petsam.LATITUDE,
    style="c0.5c",
    pen="white",
    cmap=True,
    fill=petsam.NA2O, # fill circle according color-scale
)


fig.colorbar(frame=["a1.0", "x+lNa2O", "y+lwt%"],position="JBC+o0c/4c")
fig.show()
```

```
grdblend [NOTICE]: Remote data courtesy of GMT data server oceania [http://oceania.generic-mapping-tools.org]
grdblend [NOTICE]: SRTM15 Earth Relief at 01x01 arc minutes reduced by Gaussian Cartesian filtering (1.9 km full
width) [Tozer et al., 2019].
grdblend [NOTICE]:   -> Download 30x30 degree grid tile (earth_relief_01m_g): N60E000
```



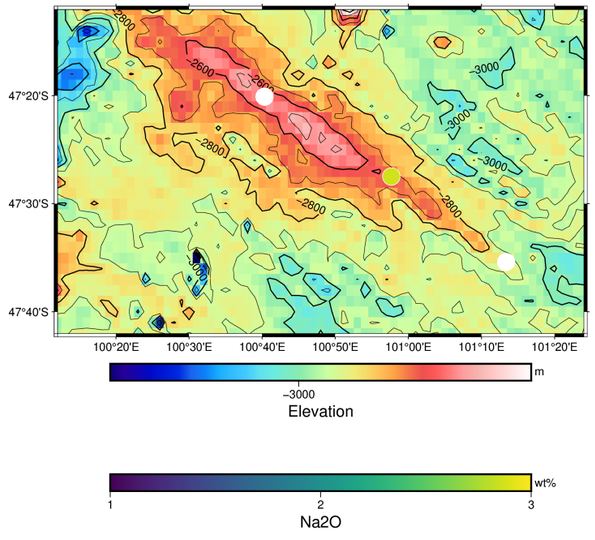
In [59]:
```
f, a = plt.subplots()

a.scatter(petsam.MGO, petsam.NA2O, alpha=0.25, s=70)

a.set_xlabel('MgO (wt%)')
a.set_ylabel('Na$_2$O (wt%')

# a.set_xlim(5, 10)
# a.set_ylim(2, 4)

plt.show()
```

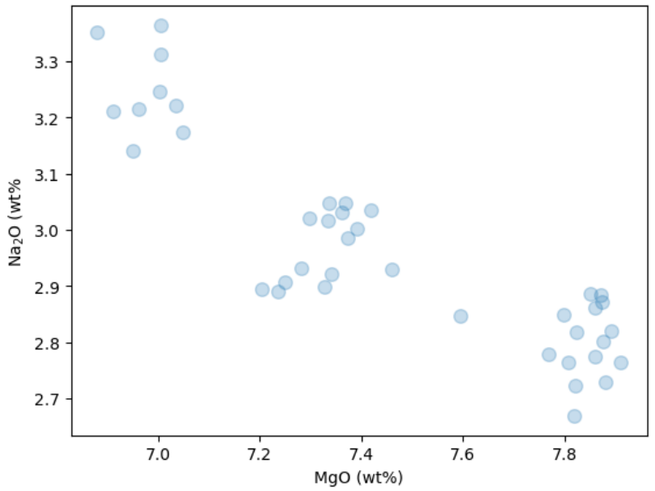
SEIR40

In [63]:
```
petsam = petdb[(petdb.LONGITUDE > 100.2) & (petdb.LONGITUDE < 101.4) & (petdb.LATITUDE > -47.7) & (petdb.LATITUDE

grid = pygmt.datasets.load_earth_relief(resolution="01m", region=[100.2, 101.4, -47.7, -47.2])

fig = pygmt.Figure()
fig.grdimage(grid=grid, projection="M15c", frame="a", cmap="haxby") # haxby is a colorscale that is OK for oceans
fig.grdcontour(grid=grid) # plot up contours
fig.colorbar(frame=["a1000", "x+lElevation", "y+lm"])
pygmt.makecpt(cmap="viridis", series=[1., 3.], background = "i") # color scale for Na2O - change series limits
# Plot using circles (c) of 0.5 cm, the sampled bathymetry points

fig.plot(
    x=petsam.LONGITUDE,
    y=petsam.LATITUDE,
    style="c0.5c",
    pen="white",
    cmap=True,
    fill=petsam.NA2O, # fill circle according color-scale
)


fig.colorbar(frame=["a1.0", "x+lNa2O", "y+lwt%"],position="JBC+o0c/4c")
fig.show()
```

```
grdblend [NOTICE]: Remote data courtesy of GMT data server oceania [http://oceania.generic-mapping-tools.org]
grdblend [NOTICE]: SRTM15 Earth Relief at 01x01 arc minutes reduced by Gaussian Cartesian filtering (1.9 km full
width) [Tozer et al., 2019].
grdblend [NOTICE]:   -> Download 30x30 degree grid tile (earth_relief_01m_g): S60E090
```

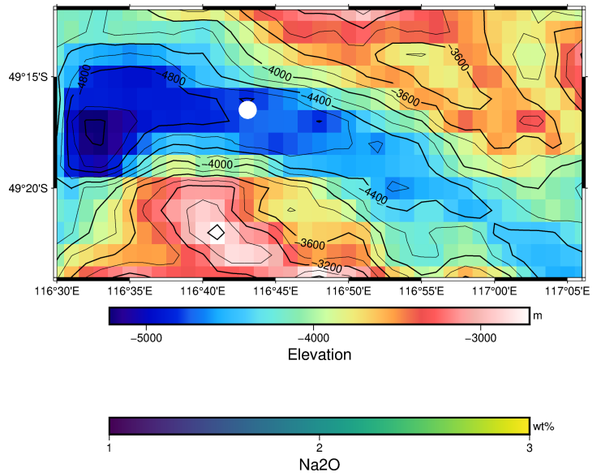


In [64]:
```
f, a = plt.subplots()

a.scatter(petsam.MGO, petsam.NA2O, alpha=0.25, s=70)

a.set_xlabel('MgO (wt%)')
a.set_ylabel('Na$_2$O (wt%')

# a.set_xlim(5, 10)
# a.set_ylim(2, 4)

plt.show()
```

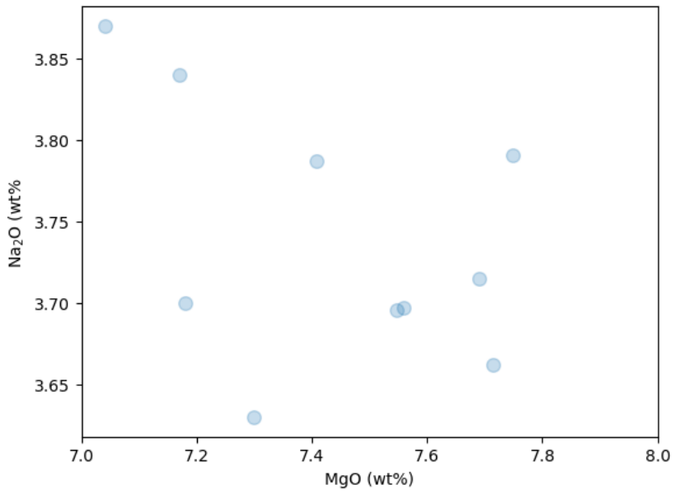


SEIR56SEIR57

In [67]:
```
petsam = petdb[(petdb.LONGITUDE > 116.5) & (petdb.LONGITUDE < 117.1) & (petdb.LATITUDE > -49.4) & (petdb.LATITUDE

grid = pygmt.datasets.load_earth_relief(resolution="01m", region=[116.5, 117.1, -49.4, -49.2])

fig = pygmt.Figure()
fig.grdimage(grid=grid, projection="M15c", frame="a", cmap="haxby") # haxby is a colorscale that is OK for oceans
fig.grdcontour(grid=grid) # plot up contours
fig.colorbar(frame=["a1000", "x+lElevation", "y+lm"])
pygmt.makecpt(cmap="viridis", series=[1., 3.], background = "i") # color scale for Na2O - change series limits
# Plot using circles (c) of 0.5 cm, the sampled bathymetry points

fig.plot(
    x=petsam.LONGITUDE,
    y=petsam.LATITUDE,
    style="c0.5c",
    pen="white",
    cmap=True,
    fill=petsam.NA2O, # fill circle according color-scale
)


fig.colorbar(frame=["a1.0", "x+lNa2O", "y+lwt%"],position="JBC+o0c/4c")
fig.show()
```



In [69]:
```
f, a = plt.subplots()

a.scatter(petsam.MGO, petsam.NA2O, alpha=0.25, s=70)

a.set_xlabel('MgO (wt%)')
a.set_ylabel('Na$_2$O (wt%')

a.set_xlim(7, 8)
# a.set_ylim(2, 4)

plt.show()
```



In [ ]:

In [ ]:

In [ ]:

4. Modelling the global array

It has been suggested that the global array you have just obtained a version of is controlled by variations in mantle temperature. By constructing a simple model, we can test whether this is a feasible explanation.

We will add a line to the plot above where each point on the line represents a different value of T_p.

The model will need to take mantle potential temperature T_p as an input, and estimate the Na>sub>2O content of the primary melt and the axial depth of the ridge. In the lectures we have covered the equations needed to do this. The first step is to calculate the melt fraction generated for a given T_p at a spreading centre. There are many ways of doing this (including reading numbers off the figures from the lecture!), but there is a convenient python package that can help us:

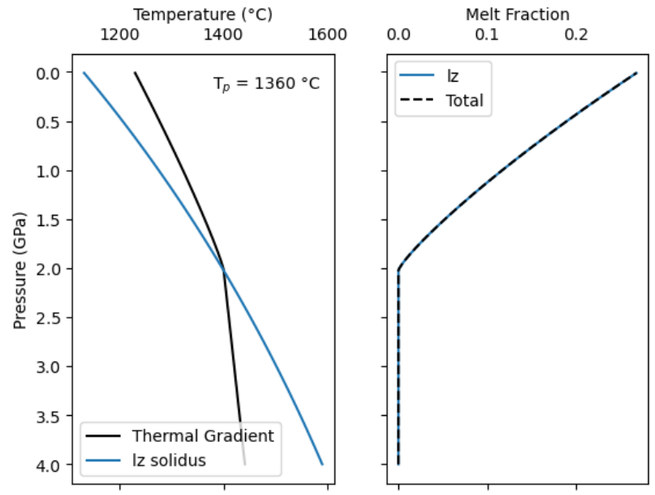
In [78]:
```
import pyMelt as m
```

The way the pyMelt package is structured requires us to define a mantle lithology and build a mantle object from it:

In [79]:
```
# Only need to run this once!
lz = m.lithologies.matthews.klb1()
mantle = m.mantle([lz], [1.0], ['lz'])
```

This tells pyMelt to use a set of equations that paramterise KLB1 lherzolite melting published by Matthews et al. (2021). Then we can perform an adiabatic melting calculation at a given T_p (in ˚C), and produce a plot to visualise the results:

In [80]:
```
# This line does the calculation:
column = mantle.adiabaticMelt(1360.0, Pstart=4.0)

# This line makes a nice plot
f, a = column.plot()
```

Q4.1: Find in your lecture notes the equation for obtaining crustal thickness from the information plotted above. Can you make a rough estimate of what it should be?

🟢 *Since there are melts being contributed from a triangular melting region we need to sum each melt along the curve (the distant wings of the melting region have the low F melts, the centre of the ridge has the high F part.*

🟢 *In this case the curve makes an almost triangle shape, so we can use the area of a triangle.*

🟢 *Remember to convert from pressure to depth!*

🟢 *I calculate 7.6 km*

In [81]:
```
# You might want to write some python code to help!

0.5 * (2e9 / 3300 / 10) * 0.25
```

Out[81]: 7575.757575757576

We can have pyMelt do the calculation too, but first we must tell it to creating a spreading centre:

In [82]:
```
mor = m.geosettings.spreadingCentre(column)
ref_tc = mor.tc
ref_tc
```

Out[82]: 6.927764192167202

Q4.2: Does the calculated value differ from your own answer? Why?

🟢 *Yes! The approximation of a triangle is likely to make only a small difference. The thing my calculation neglected is that there is already a crust on top of the melting region. I integrated to 0 GPa, but in reality the mantle could only upwell to ~0.25 GPa before the crust starts. To incorporate this into the calculation, pyMelt integrates iteratively, doing one step at a time, until the pressure of the step equals the pressure exerted by the crust already integrated.*

We expect that mid-ocean ridges are close to isostatic equilibrium, so variations in axial depth should be driven by density differences in the rock beneath them. There will be two main components to this: i) variations in crustal thickness and ii) variations in mantle temperature. To make the calculations easier we will ignore the contribution of mantle temperature and focus on crustal thickness.

Q4.3: Derive an expression linking a change in ridge bathymetry to a change in crustal thickness.

Assume uniform densities (values below) for the crust, ocean, and mantle. | Material | Density (kg m^{-3}) | |---|---| | Water | 1000 | | Crust | 3000 | | Convecting Mantle | 3300 |

Doing the isostatic balance and rearranging:

$ \Delta w = \Delta t_c \frac{\rho_c - \rho_m}{\rho_m - \rho_w} $

Using pyMelt we can calculate the thickness of oceanic crust as a function of mantle T_p, and therefore the change in bathymetry from a reference value. We will use a T_p of 1360˚C as the reference case (for which we calculated the crustal thickness above).

Q4.4: By modifying the code below, create a loop that calculates the crustal thickness for a range of T_p:

In [120…
```
# Create an empty list to store the answers in
tcs = []

# This is for Q4.6:
Fmax = []

for Tp in np.linspace(1300, 1600, 7): # Run the code below for 7 values of Tp spaced evenly between 1300 and 160

    result = 0.0 # Change this bit!

    columncalc = mantle.adiabaticMelt(Tp, Pstart=4.0)
    morcalc = m.geosettings.spreadingCentre(columncalc)
    result = morcalc.tc

    tcs.append(result)

    # Code for Q4.6 can go here:
    result = morcalc.F.max()
    Fmax.append(result)

print(tcs)
```

```
/Users/sm905/opt/anaconda3/lib/python3.9/site-packages/pyMelt/mantle_class.py:450: UserWarning: Super solidus st
art
  _warn("Super solidus start")
/Users/sm905/opt/anaconda3/lib/python3.9/site-packages/pyMelt/mantle_class.py:450: UserWarning: Super solidus st
art
  _warn("Super solidus start")
[3.5412736870509636, 6.269182014104794, 9.884386541561335, 14.280999384511631, 19.14423292099851, 24.40491627346
7503, 29.730501928983546]
```

We can convert this to a numpy array to make calculations easier:

In [121…
```
tcs = np.array(tcs)
Fmax = np.array(Fmax)
```

Q4.5: Now use the expression you derived above to calculate the bathymetry relative to the T_p = 1360˚C case:

In [122…
```
# Your code here...

relative_bathymetry = (tcs - ref_tc) * -0.1304

relative_bathymetry
```

Out[122]:
```
array([ 0.44159836,  0.08587912, -0.38554355, -0.95886187, -1.59302752,
       -2.27902063, -2.973477  ])
```

We must now calculate the primary Na_2O content of the magmas associated with each T_p.

If we use a highly simplified melting model, the Na_2O content of the mantle melts can be calculated from the following expression for accumulated fractional melts which was developed by Plank et al., 1995, a paper which is worth a look.

$$ C_L = \frac{2C_0}{X_{max}} \left[1 + \frac{D \left[(1 - X_{max})^{((1/D) - 1))} -1 \right]}{X_{max}(D+1)} \right] $$
In this case, C_L is the liquid composition, C_0 is the mantle source composition (you can try 0.3 wt% for Na_2O and D, the partition coefficient during melting (try a constant of 0.01 for Na).

You can extract the melt fraction at the top of the melting column (i.e., the maximum melt fraction) from the pyMelt results using:

In [123…
```
mor.F.max()
```

Out[123]:
```
0.23304347465452885
```


Q4.6: Add Fmax into the loop above so you extract Fmax and crustal thickness at the same time. Check Fmax makes sense.

In [124…
```
Fmax
```

Out[124]:
```
array([0.17426217, 0.22365055, 0.26896055, 0.28350108, 0.29181602,
       0.31016789, 0.34267905])
```

Q4.7: Use these values of Fmax to calculate Na_2O of the magma produced for each T_p:

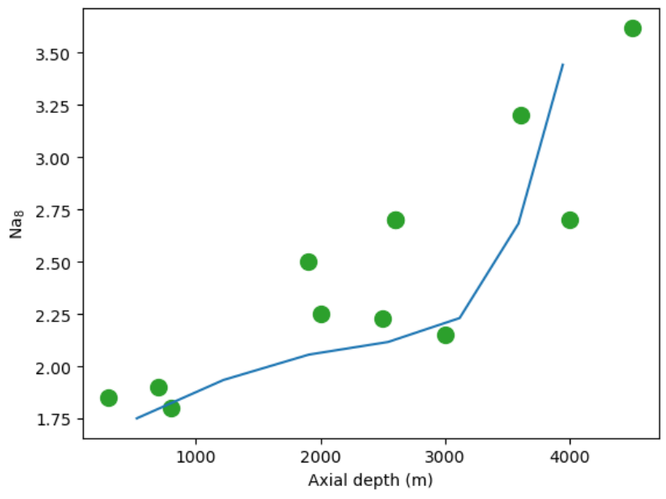
In [125…
```
Na2O = 2 * 0.3 / Fmax * (1 + (0.01 * (1 - Fmax)**((1/0.01)-1)) / (Fmax * (0.01 + 1)))
Na2O
```

Out[125]:
```
array([3.44308809, 2.68275666, 2.23081044, 2.11639406, 2.05609002,
       1.93443623, 1.75090949])
```

Q4.8: Create a version of the global array plot with the model superimposed.

*Remember that the bathymetrys calculated are relative** bathymetrys. You will need to add an offset to it to match the data.

In [126…
```
fig, ax = plt.subplots(dpi=100)

ax.scatter(axial_depths, Na8, c='C2', s=100)

ax.set_xlabel('Axial depth (m)')
ax.set_ylabel('Na$_8$')


ax.plot(relative_bathymetry*1000 + 3500, Na2O)

plt.show()
```



Q4.9: What range of T_p is required to explain the global array? What might affect your answer?

The model curve above is for T_p 1300-1600˚C which encompasses most of the data. One of the big factors that could affect the results is our choice of mantle melting model. Different paramterisations will report different melt fractions as a function of T_p and will therefore cause a systematic offset in the model results. Lithological heterogeneity could also affect the results. A recent study that incorporated the effects of lithological heterogeneity obtained a T_p of 1530˚C for Iceland, so perhaps the upper end of the range is too high. Our estimate of mantle Na_2O concentration might be (probably is) wrong too, and it probably varies globally. You also have the effects of conductive cooling at slow spreading ridges, active upwelling at near-plume localities, and a whole host of other things.

Please take a look at Gale et al., 2014 and Plank et al., 1995 to gain a fuller understanding of the nature and origin of the global correlation between MORB compositions and oceanic crustal thicknesses.

In [ ]: