# Homeworks 4 and 5

This homework will practice several tools that we cover over the past 2 weeks. We will practice handling geopandas, plotting these data on maps, reading/writing in netcdf, and spectral analysis of spatial and temporal data.

Make sure you started your notebook with the `uwdsgeo` environment.

**1. Terrestrial Glacier data base** (15 points):

practice geopandas, practice plotting on maps, practice measuring means and correlations, linear regressions.

**2. Ice-shelf seismograms** (15 points)

Time-domain filtering, 1D Fourier transform.

**3. 2D Crustal model** (10 points)

practice reading netcdf, making maps and exploring 2D spectral content.

# 1. Terrestrial Glaciers

We will look at ice thickness of global glaciers from Welty et al, 2021:\ Welty, E., Zemp, M., Navarro, F., Huss, M., Fürst, J.J., Gärtner-Roer, I., Landmann, J., Machguth, H., Naegeli, K., Andreassen, L.M. and Farinotti, D., 2020. Worldwide version-controlled database of glacier thickness observations. Earth System Science Data, 12(4), pp.3039-3055. https://doi.org/10.5194/essd-12-3039-2020

!git clone https://gitlab.com/wgms/glathida.git

## a) Import Python modules (1 point)

Import pandas, geopandas, plotting, raster files, numpy

In [1]:
```python
# solution
import folium
import geopandas as gpd
import matplotlib.pyplot as plt
import netCDF4 as nc
import numpy as np
import pandas as pd
import pycrs
import rasterio

from folium.plugins import MarkerCluster
from rasterio.mask import mask
from rasterio.plot import show
```

## b) Import data (2 points)

Read the glacier data from the file `glathida/data/T.csv` into a pandas data frame, and decribe briefly the dataframe content and its first few lines.

In [2]:
```
# solution
glacier = pd.read_csv('glathida-master/data/T.csv')
glacier
```

Out[2]:

| | GlaThiDa_ID | POLITICAL_UNIT | GLACIER_NAME | GLACIER_DB | GLACIER_ID | LAT |
|---|---|---|---|---|---|---|
| **0** | 1 | SE | ISFALLSGLAC | WGI | SE4B000E0006 | 67.91500 |
| **1** | 2 | SE | RABOTS GLACIAER | WGI | SE4B000E1016 | 67.91000 |
| **2** | 3 | SE | STORGLACIAEREN | WGI | SE4B000E0005 | 67.90000 |
| **3** | 4 | US | SOUTH CASCADE | WGI | US2M00264006 | 48.35698 |
| **4** | 5 | CA | ATHABASCA | FOG | 7 | 52.17540 |
| **...** | ... | ... | ... | ... | ... | ... |
| **5136** | 6627 | US | MOUNT ADAMS | NaN | NaN | 46.20240 |
| **5137** | 6628 | US | MOUNT ADAMS | NaN | NaN | 46.20240 |
| **5138** | 6629 | US | MOUNT ADAMS | NaN | NaN | 46.20398 |
| **5139** | 6630 | AT | MULLWITZKEES | OTH | 6054 | 47.08670 |
| **5140** | 6631 | AT | MULLWITZKEES | OTH | 6054 | 47.08670 |

5141 rows × 26 columns

## c) Convert Pandas to Geopandas (1 point)

You can create a Geopandas GeoDataFrame from a Pandas DataFrame if there is coordinate data in the DataFrame. In the data that you opened above, there are columns for the `X` (or longitude) and `Y` (or latitude) coordinates of each rock formation - with headers named `X` (or here LON) and `Y` (or LAT).

You can convert columns containing x,y coordinate data using the GeoPandas `points_from_xy()` function as follows:

`coordinates = gpd.points_from_xy(column-with-x-data.X, column-with-y-data.Y)`

Describe the new geopandas.

In [3]:
```
gdf = gpd.GeoDataFrame(
    glacier, geometry=gpd.points_from_xy(glacier.LON, glacier.LAT))

print(gdf.head())
```

```
   GlaThiDa_ID POLITICAL_UNIT     GLACIER_NAME GLACIER_DB    GLACIER_ID  \
0            1             SE      ISFALLSGLAC        WGI  SE4B000E0006
1            2             SE   RABOTS GLACIAER        WGI  SE4B000E1016
2            3             SE    STORGLACIAEREN        WGI  SE4B000E0005
3            4             US     SOUTH CASCADE        WGI  US2M00264006
4            5             CA         ATHABASCA        FOG             7

        LAT         LON  SURVEY_DATE  ELEVATION_DATE  AREA  ...   \
0  67.91500    18.56800   19790399.0      19799999.0   1.3  ...
1  67.91000    18.49600   19790399.0      19799999.0   4.1  ...
2  67.90000    18.57000   19790399.0      19799999.0   3.1  ...
3  48.35698 -121.05735   19759999.0      19759999.0   2.0  ...
4  52.17540 -117.28400          NaN             NaN   3.8  ...

   NUMBER_OF_SURVEY_POINTS  NUMBER_OF_SURVEY_PROFILES  \
0                      NaN                        NaN
1                      NaN                       10.0
2                      NaN                        NaN
3                      NaN                        NaN
4                      NaN                        NaN

   TOTAL_LENGTH_OF_SURVEY_PROFILES  INTERPOLATION_METHOD  \
0                              NaN                   NaN
1                              NaN                   NaN
2                              NaN                   NaN
3                              NaN                   NaN
4                              NaN                   NaN

            INVESTIGATOR      SPONSORING_AGENCY  \
0  Schytt V. and others  University of Iceland
1  Schytt V. and others  University of Iceland
2  Schytt V. and others  University of Iceland
3                   NaN                    NaN
4                   NaN                    NaN

                                REFERENCES  DATA_FLAG  REMARKS  \
0                 Björnsson, H., (1981). Geogr. Ann.        NaN      NaN
1                 Björnsson, H., (1981). Geogr. Ann.        NaN      NaN
2                 Björnsson, H., (1981). Geogr. Ann.        NaN      NaN
```

```
3  Driedger, C.L., and Kennard, P.M., (1986a). An...        NaN       NaN
4  Driedger, C.L., and Kennard, P.M., (1986a). An...        NaN       NaN

                   geometry
0    POINT (18.56800 67.91500)
1    POINT (18.49600 67.91000)
2    POINT (18.57000 67.90000)
3  POINT (-121.05735 48.35698)
4  POINT (-117.28400 52.17540)

[5 rows x 27 columns]
```

## d) Mapping geopandas points (3 points)

Import a nice background elevation map using a rasterIO image. Use the tutorial instructions and download the file from; https://www.naturalearthdata.com/downloads/50m-raster-data/50m-cross-blend-hypso/

In [4]:
```python
elevation = rasterio.open('HYP_50M_SR/HYP_50M_SR.tif')
```

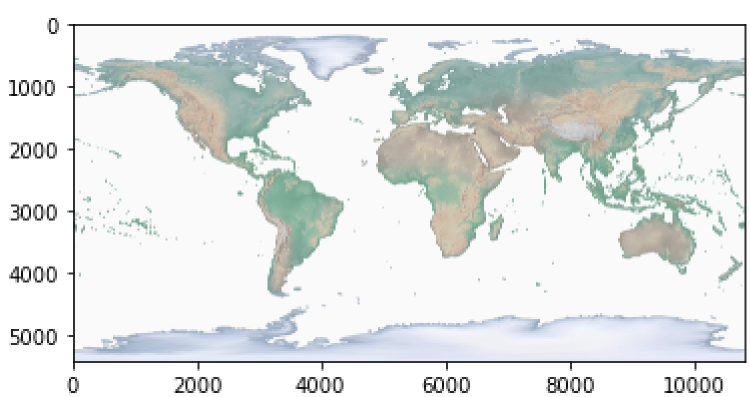
***Tips***: when plotting a image in `matplotlib` you need to add information about the physical dimensions of the image. You can calculate the `bounds` .

In [5]:
```python
bounds = (elevation.bounds.left, elevation.bounds.right, \
          elevation.bounds.bottom, elevation.bounds.top)
```

We will use `matplotlib.pyplot` to show the raster image in the background (tips: use `imshow()` . The raster image in matplotlib can only import one frame and not three (R, G, B) frames. We will first stack the three images together.

In [6]:
```python
red = elevation.read(1)
green = elevation.read(2)
blue = elevation.read(3)
pix = np.dstack((red, green, blue))
plt.imshow(pix)
```

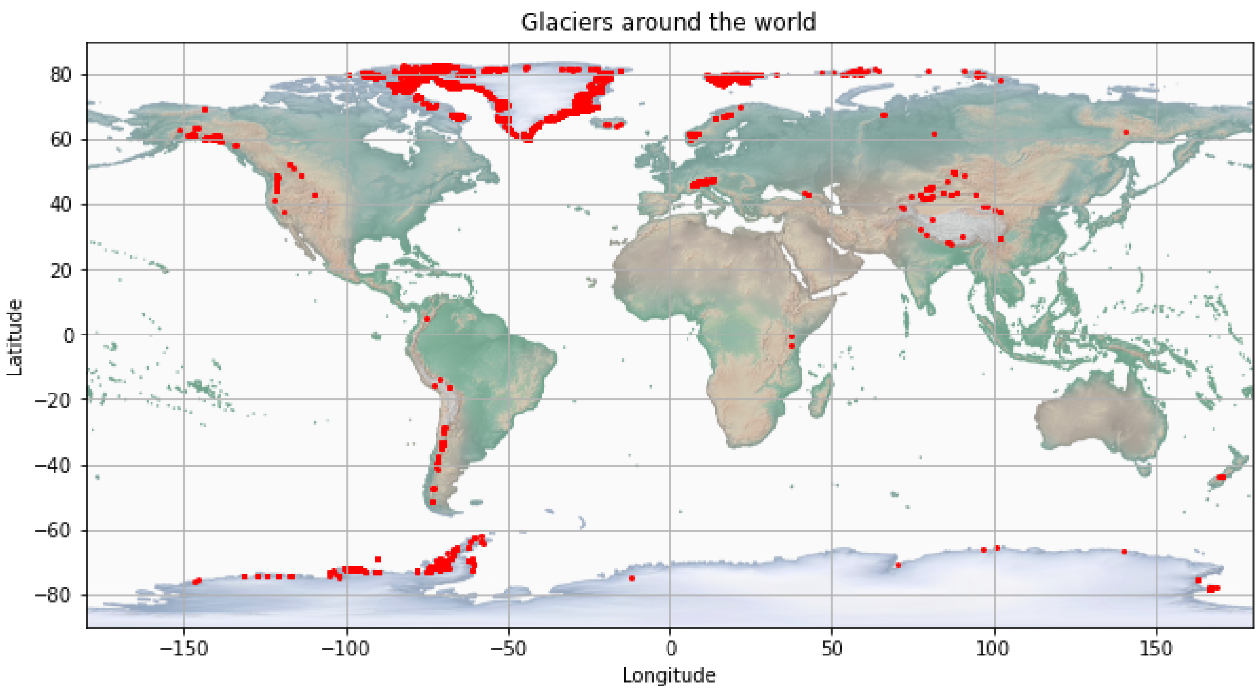
Out[6]:
```
<matplotlib.image.AxesImage at 0x7f8e5e438b80>
```




Then we will use `pix` as the first layer of the plot. Because `pix` only contains pixel dimension, you can add the physical dimension using the argument `extent=bounds` in your first plot. Then add the Geopandas points using the geopandas `plot()` function and customize the marker size, style, and color using your artistic talents. Please anotate the figure with x and y labels, a title, and save the figure into a PNG. The figure should be saved into an

11x8 inch plot, and fontsize should be at least 14 points. You can set your default values for all of your plots using the `rc.Params.update` parameters we tested in the week3_lab1 tutorial.

In [7]:
```
# set default parameters
params = {'legend.fontsize': 14, \
          'xtick.labelsize':14, \
          'ytick.labelsize':14, \
          'font.size':14}
fig, ax = plt.subplots(figsize=(11, 8))
ax.set_xlabel('Longitude')
ax.set_ylabel('Latitude')
ax.set_title('Glaciers around the world')
ax.grid(True)

gdf.plot(ax=ax, color='red', markersize=4)
plt.imshow(pix,extent=bounds)
plt.savefig('week4_plot1.png')
```




## e) Explore the data with vizualisation (3 points)

Before making any inference of models with the data, we will start by exploring basic correlations among parameters by plotting. In particular, we will focus on `MEAN_THICKNESS`, `AREA`, `MEAN_SLOPE` parameters.

The database may contain Nans and other "bad" values (welcome to the data world!). First we will clean the data by removing nans. We are mostly interested in the thickness, area, and slope

In [8]:
```
gdf2=gdf.dropna(subset=['MEAN_THICKNESS','AREA','MEAN_SLOPE'])
gdf2
```

Out[8]:

| | GlaThiDa_ID | POLITICAL_UNIT | GLACIER_NAME | GLACIER_DB | GLACIER_ID | LA |
|---|---|---|---|---|---|---|

| | GlaThiDa_ID | POLITICAL_UNIT | GLACIER_NAME | GLACIER_DB | GLACIER_ID | LA |
|---|---|---|---|---|---|---|
| **32** | 33 | US | EASTON | FOG | 1367 | 48.75000 |
| **33** | 34 | US | LEMON CREEK | FOG | 3334 | 58.38000 |
| **34** | 35 | RU | PRAVIY AKTRU | WGI | SU5A15106127 | 50.06134 |
| **35** | 36 | RU | MALIY AKTRU | WGI | SU5A15106126 | 50.04967 |
| **36** | 37 | RU | LEVIY AKTRU | WGI | SU5A15106128 | 50.08008 |
| **...** | ... | ... | ... | ... | ... | . |
| **620** | 2111 | SJ | BLEKUMBREEN | GLIMS | G016068E78246N | 78.24548 |
| **628** | 2119 | KZ | TSENTRALNIY TUYUKSU | GLIMS | G077080E43049N | 43.04365 |
| **632** | 2123 | SJ | ALDEGONDABREEN | RGI | RGI50-07.01079 | 77.97140 |
| **749** | 2240 | SJ | ARIEBREEN | RGI | RGI60-07.00209 | 77.02720 |
| **837** | 2328 | SJ | AUSTRE LOVENBREEN | RGI | RGI60-07.00496 | 78.87100 |

111 rows × 27 columns

Make plots to vizualise the correlation, or lack of, between all three data. Make at least three plots.

**Tips**:

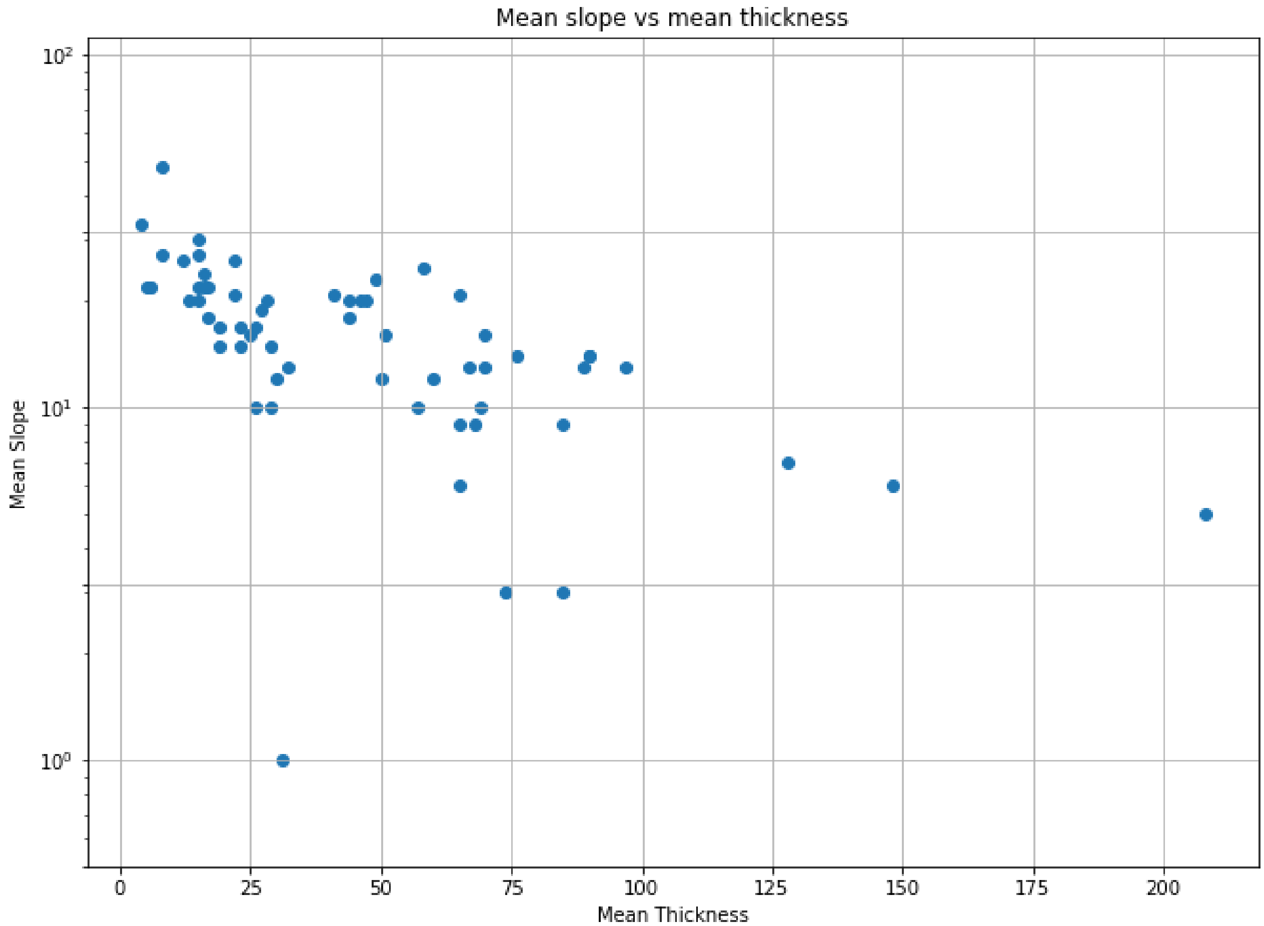
1. Use the function `scatter` to plot the values of mean thickness, mean slope, area, and latitude.
2. use one of the dataframe columns as a color using the argument `c`. You can also vary the `colormap` using the argument `cmap`. Help on colormaps can be found here: https://matplotlib.org/stable/tutorials/colors/colormaps.html. Be mindful of Color-Vision Deficient readers and read *Crameri, F., Shephard, G.E. and Heron, P.J., 2020. The misuse of colour in science communication. Nature communications, 11(1), pp.1-10. https://doi.org/10.1038/s41467-020-19160-7* (find it on the class Gdrive). You can add a third "data" by choosing a marker color that scales with an other parameter. For instance, try coloring your marker with the `LAT` parameter to look at systematic latitudinal trends from the equator to the poles.
3. Do not forget to adjust fontsize, figure size (at least 10,8), grid, labels with units. ou may also explore the *logarithmic* correlations by mapping the axis from linear to logarithmic scale `plt.xscale('log')`.

In [9]:

```
# Figure 1: Mean slope vs mean thickness
fig, ax = plt.subplots(figsize=(11, 8))
ax.set_xlabel('Mean Thickness')
ax.set_ylabel('Mean Slope')
ax.set_title('Mean slope vs mean thickness')
ax.grid(True)

ax.scatter(gdf2.MEAN_THICKNESS,gdf2.MEAN_SLOPE)
ax.set_yscale('log')
ax.set_yticks([0.5,1,3.15,10,31.6,100])
```

Out[9]:

```
[<matplotlib.axis.YTick at 0x7f8de45d2910>,
 <matplotlib.axis.YTick at 0x7f8de45d24f0>,
 <matplotlib.axis.YTick at 0x7f8de45cd970>,
 <matplotlib.axis.YTick at 0x7f8de45f4b50>,
 <matplotlib.axis.YTick at 0x7f8de45ff0a0>,
 <matplotlib.axis.YTick at 0x7f8de45ff5b0>]
```

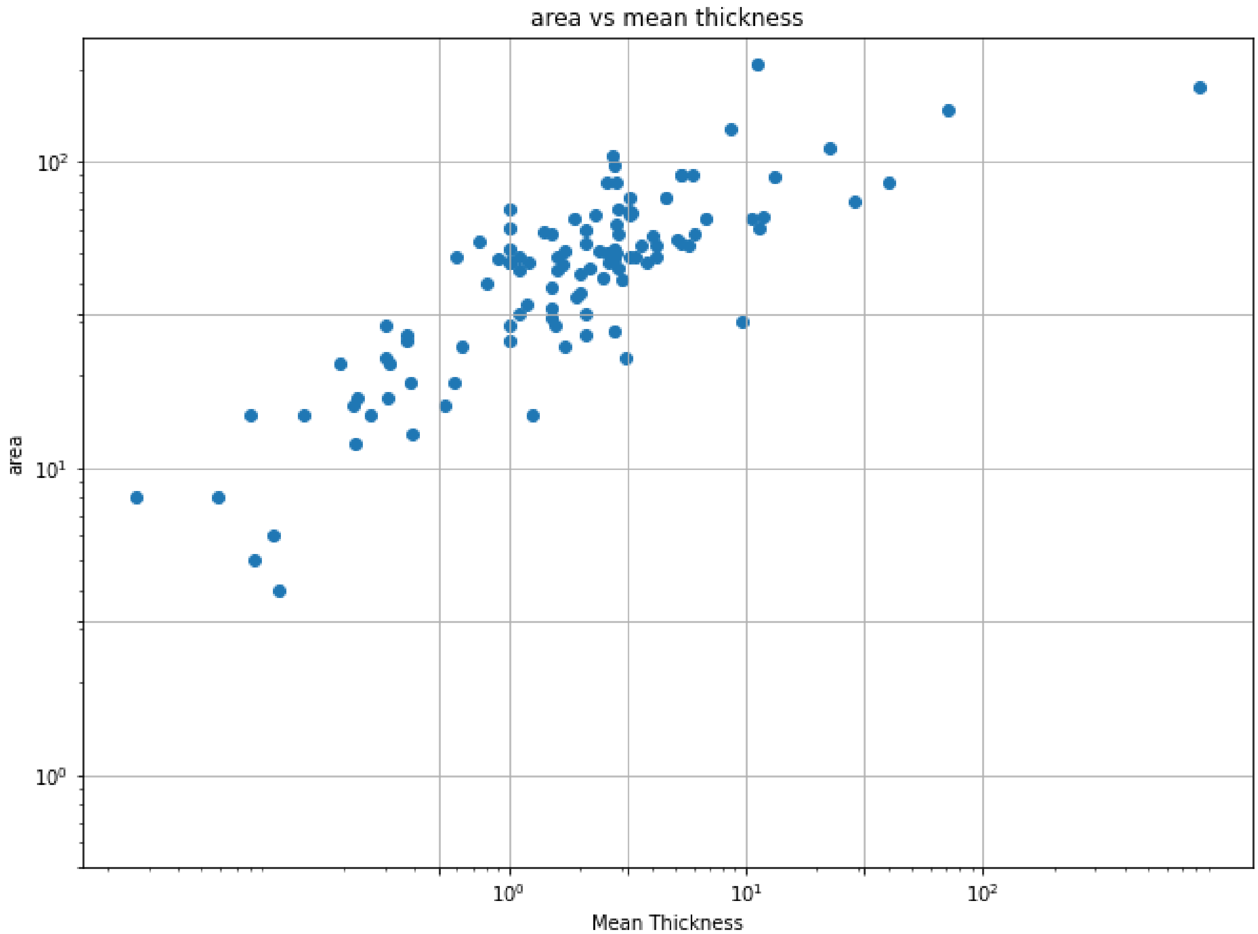



In [10]:

```
# Figure 2: area vs mean thickness
fig, ax = plt.subplots(figsize=(11, 8))
ax.set_xlabel('Mean Thickness')
ax.set_ylabel('area')
ax.set_title('area vs mean thickness')
ax.grid(True)

ax.scatter(gdf2.AREA,gdf2.MEAN_THICKNESS)
ax.set_xscale('log')
ax.set_yscale('log')
ax.set_yticks([0.5,1,3.15,10,31.6,100])
ax.set_xticks([0.5,1,3.15,10,31.6,100])
```

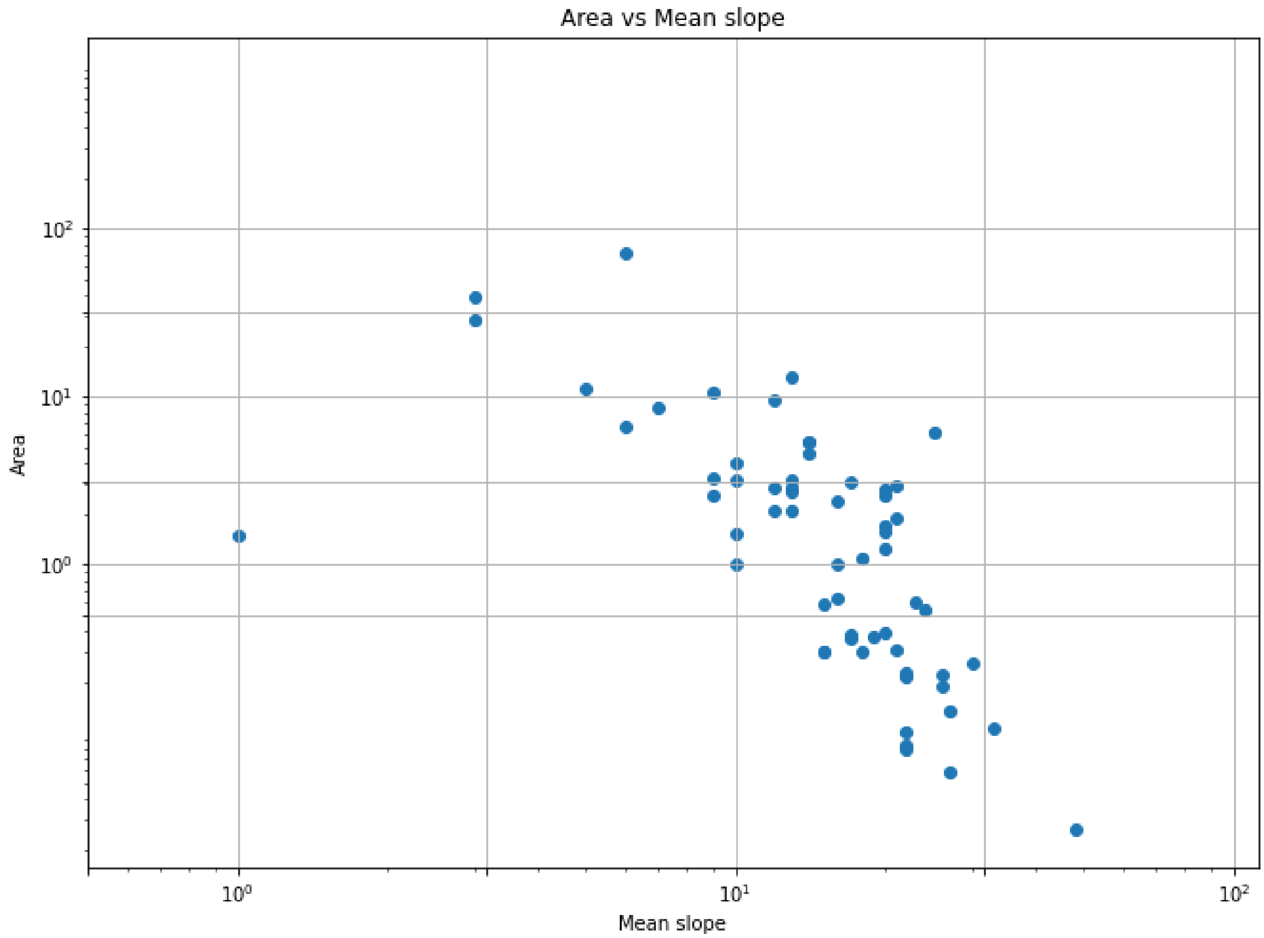
Out[10]:

```
[<matplotlib.axis.XTick at 0x7f8e5a866940>,
 <matplotlib.axis.XTick at 0x7f8de47372b0>,
 <matplotlib.axis.XTick at 0x7f8de4737700>,
 <matplotlib.axis.XTick at 0x7f8de46904f0>,
 <matplotlib.axis.XTick at 0x7f8de4690a00>,
 <matplotlib.axis.XTick at 0x7f8de4690f10>]
```




In [11]:
```
# Figure 2: area vs mean slope
fig, ax = plt.subplots(figsize=(11, 8))
ax.set_xlabel('Mean slope')
ax.set_ylabel('Area')
ax.set_title('Area vs Mean slope')
ax.grid(True)

ax.scatter(gdf2.MEAN_SLOPE,gdf2.AREA)
ax.set_yscale('log')
ax.set_xscale('log')
ax.set_yticks([0.5,1,3.15,10,31.6,100])
ax.set_xticks([0.5,1,3.15,10,31.6,100])
```

Out[11]:
```
[<matplotlib.axis.XTick at 0x7f8de46f4730>,
 <matplotlib.axis.XTick at 0x7f8e047c4850>,
 <matplotlib.axis.XTick at 0x7f8e047c4a00>,
 <matplotlib.axis.XTick at 0x7f8e044aaa90>,
 <matplotlib.axis.XTick at 0x7f8e044aae50>,
 <matplotlib.axis.XTick at 0x7f8de5549970>]
```




## f) Linear Regression (5 points total counted in the next section)

You found from basic data visualization that the three parameters `MEAN_SLOPE`, `MEAN_THICKNESS`, and `AREA` are correlated. It does make physical sense because a *steep* glaciers is likely to be in the high mountains regions, hanging on the mountain walls, and thus be constrained, and conversely, a flat glacier is either at its valley, ocean terminus or on ice sheets.

**1. Simple linear regression (1 point)** We will now perform a regression between the parameters (or their log!). Linear regressions are models that can be imported from scikit-learn. Log/exp functions in numpy as `np.log()` and `np.exp()`. Remember that a linear regression is finding $a$ and $b$ knowing both $x$ and the data $y$ in $y = Ax + b$. We want to **predict ice thickness from a crude estimate of the glacier area.**

**Tips**: a. make sure that the dimensions are correct and that there is no NaNs and zeros. b. Make sure to inport the scikit learn linear regression function and the error metrics.

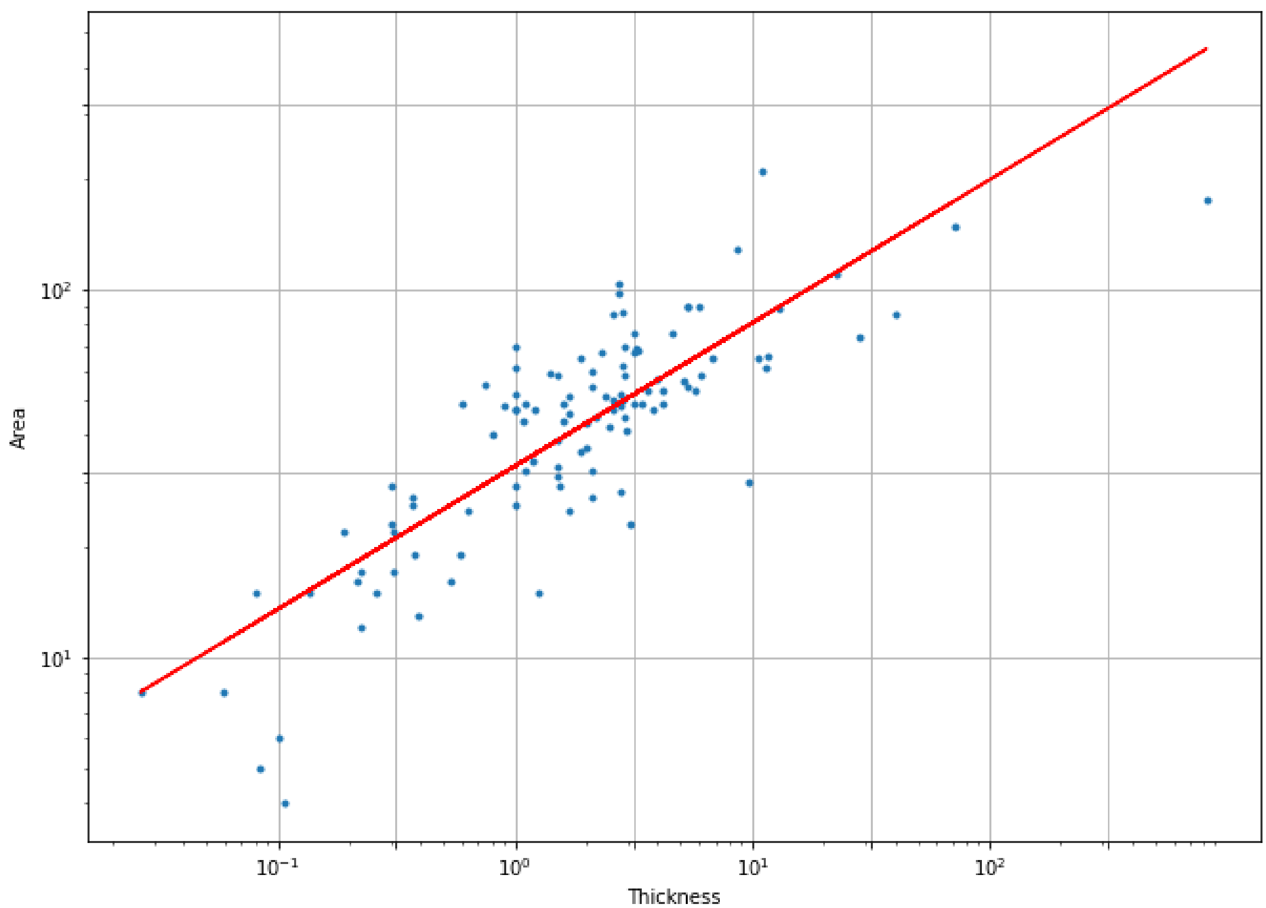
In [12]:
```
# solution
from sklearn.linear_model import LinearRegression
# convert the data into numpy arrays.
T = np.log(np.asarray(gdf2.MEAN_THICKNESS).reshape(-1, 1))
a = np.log(np.asarray(gdf2.AREA).reshape(-1, 1))# reshaping was necessary to be


# perform the linear regression. First we will use the entire available data
regr = LinearRegression()
```

```
# we will first perform the fit:
model = regr.fit(a,T)

# We will first predict the fit:
Epred=regr.predict(a)

# The coefficients
print('Coefficient /(Thickness/Area): ', regr.coef_[0][0])
```

```
Coefficient /(Thickness/Area):  0.3890224843292104
```

Make a plot of the data and the linear regression your just performed

In [13]:
```
fig, ax = plt.subplots(figsize=(11, 8))
plt.plot(np.exp(a),np.exp(T),'.')
ax.grid(True)
ax.set_ylabel('Area')
ax.set_xlabel('Thickness')
ax.set_yscale('log')
ax.set_xscale('log')
ax.set_yticks([0.1,0.31,1,3.15,10,31.6,100,316])
ax.set_xticks([0.1,0.31,1,3.15,10,31.6,100,316])
plt.plot(np.exp(a),np.exp(Epred),color="red")
plt.grid(True)
plt.show()
```




Briefly comment on the quality of your fit and a linear regression (1 point)

The R^2 is at 0.68, meaning that it doesn't fit all the datapoints, but looking at the overall trend this linear regression could explain it.

In [14]:

```
from sklearn.metrics import mean_squared_error, r2_score
# The mean squared error
print('Mean squared error (mm): %.2f'
      % mean_squared_error(T, Epred))
# The coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: %.2f'
      % r2_score(T, Epred))
```

```
Mean squared error (mm): 0.16
Coefficient of determination: 0.68
```

### 2. Leave One Out Cross Validation linear regression (1 point)

Perform the LOCCV on the `AREA` and `THICKNESS` values. Predict the `THICKNESS` value knowing a `AREA` value. Use material seen in class. Make a plot of your fit.

In [15]:

```
# we randomly select values and split the data between training and validation s
from sklearn.model_selection import ShuffleSplit
# we split once the data between a training and a validating set
n=1 # we do this selectio once
v_size = 0.3 # 30% of the data will be randomly selected to be the validation se

rs = ShuffleSplit(n_splits=n, test_size=.3, random_state=0)
for train_index, val_index in rs.split(T):
    T_train, T_val = T[train_index], T[val_index]
    a_train, a_val = a[train_index], a[val_index]

from sklearn.model_selection import LeaveOneOut
loo = LeaveOneOut()

vel = np.zeros(len(T)) # initalize a vector to store the regression values
mse = np.zeros(len(T))
r2s = np.zeros(len(T))
i=0
for train, test in loo.split(T):
    T_train, E_val = T[train_index], T[val_index]
    a_train, t_val = a[train_index], a[val_index]
    # now fit the data on the training set.
    regr = LinearRegression()
    regr_val = LinearRegression()
    # Fit on training data:
    regr.fit(a_train,T_train)
    # Fit on validation data:
    regr_val.fit(a_val,T_val)
    # We will first predict the fit:
    Epred=regr.predict(a)
    Epred_val=regr_val.predict(a)

    # The coefficients
    vel[i]= regr.coef_[0][0]
    mse[i]= mean_squared_error(Epred, Epred_val)
    r2s[i]=r2_score(Epred, Epred_val)
    i+=1

# the data shows cleary a trend, so the predictions of the trends are close to e
print("mean of the slope estimates %f4.2 and the standard deviation %f4.2"%(np.m
# the test error is the average of the mean-square-errors
print("CV = %f4.2"%(np.mean(mse)))
```

```
mean of the slope estimates 0.4148354.2 and the standard deviation 0.0000004.2
CV = 0.0308974.2
```

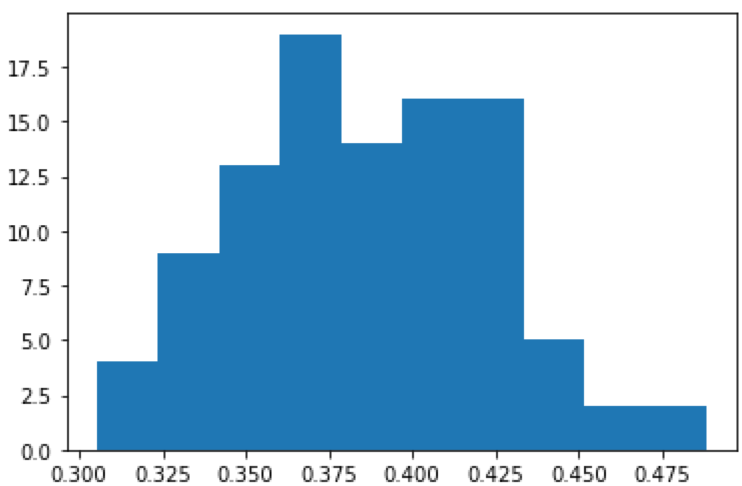
### 3. Bootstrapping (1 point)

Perform the same analysis but using a bootstrapping technique. Output the mean and standard deviation of the slope. An illustration with a histogram may help.

In [16]:
```
from sklearn.utils import resample
k=100
vel = np.zeros(k) # initalize a vector to store the regression values
mse = np.zeros(k)
r2s = np.zeros(k)
i=0
for iik in range(k):
    ii = resample(np.arange(len(T)),replace=True,n_samples=len(T))# new indices
    T_b, a_b = T[ii], a[ii]
    # now fit the data on the training set.
    regr = LinearRegression()
    regr_val = LinearRegression()
    # Fit on training data:
    regr.fit(a_b,T_b)
    Epred=regr.predict(a)

    # The coefficients
    vel[i]= regr.coef_[0][0]
    i+=1

# the data shows cleary a trend, so the predictions of the trends are close to e
print("mean of the slope estimates %f4.2 and the standard deviation %f4.2"%(np.m
plt.hist(vel)
```

```
mean of the slope estimates 0.3860904.2 and the standard deviation 0.0373154.2
```

Out[16]:
```
(array([ 4.,  9., 13., 19., 14., 16., 16.,  5.,  2.,  2.]),
 array([0.3053047 , 0.3236182 , 0.34193169, 0.36024519, 0.37855869,
        0.39687219, 0.41518568, 0.43349918, 0.45181268, 0.47012618,
        0.48843967]),
 <BarContainer object of 10 artists>)
```



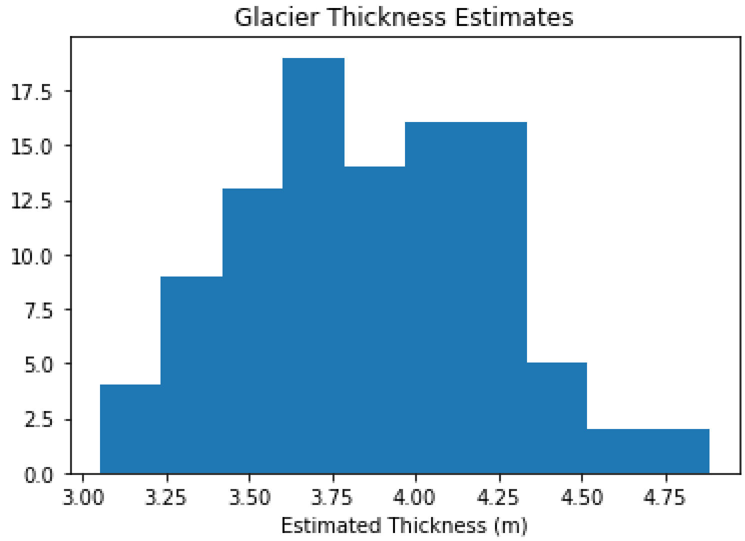

### 4. Predict the thickness of a glacier (2 points)

Let assume that you measure a glacier of area 10 km$^2$. Can you use your bootstrap regression framework to provide a distribution of possible values of the ice thickness ? Output the mean and standard deviation of the predicted ice thickness.

In [17]:
```
# solution
```

```
glacierarea = 10
estimates = glacierarea * vel
plt.hist(estimates)
plt.title("Glacier Thickness Estimates")
plt.xlabel("Estimated Thickness (m)")
print("mean of the thickness estimates %f4.2 and the standard deviation %f4.2"%(
```

```
mean of the thickness estimates 3.8608954.2 and the standard deviation 0.373149
4.2
```




## 2) Spectrogram analysis of iceshelf vibrations (15 points total)

We will explore the spectral content of the vibrations felt on iceshelves. We first download seismic data, then filter it at different frequency bandwidths, then plot the spectrogram and comment on the data.

The seismic data is handled by the Obspy package. Review the obspy tutorial that Ariane. We will download the data presented in: Aster, R.C., Lipovsky, B.P., Cole, H.M., Bromirski, P.D., Gerstoft, P., Nyblade, A., Wiens, D.A. and Stephen, R., 2021. Swell-Triggered Seismicity at the Near-Front Damage Zone of the Ross Ice Shelf. Seismological Research Letters. https://doi.org/10.1785/0220200478

**Tips**:

1. Check out the SciPy filtering help here: https://scipy-cookbook.readthedocs.io/items/ButterworthBandpass.html. Obspy has built in functions as well, but for the sake of practicing, explore the scipy filtering functions.

2. The usual steps to handling seismic data are: data download ( `get_waveforms` ) & removing the instrumental response ( `remove_response` ).

**a. Import the relevant Obspy python modules (1 point).**

In [18]:

```
#solution:
from obspy import read
import obspy.clients.fdsn.client as fdsn
```

```python
from obspy import read
from obspy import read_inventory
from obspy import UTCDateTime
from obspy.core.stream import Stream
from obspy.signal.cross_correlation import correlate
```

**b. Data download (2 points)**

We will now download the data from station "DR01" from seismic network "XH", channel "LHN" from 1/1/2015 until 3/31/2015. The client will be the "IRIS" data center. Obspy functions take on UTCDateTime formatted obspy datetime object, be sure to call or import that specific function. (1 point)

In [19]:
```python
#solution
networks = "XH"
stations = "DR01"
channels = "LHN"
Tstart = UTCDateTime(year = 2015, month = 1, day = 1)
Tend = UTCDateTime(year = 2015, month = 3, day = 31)

fdsn_client = fdsn.Client('IRIS')

Dtmp = fdsn_client.get_waveforms(network = networks, station = stations, locatio
                                 endtime = Tend, attach_response = True)
```

In [20]:
```python
# how many days did we download?
dt=Tend-Tstart # in seconds
Ndays = int(dt/86400) # in days
print(dt,'sec ', Ndays,'Days')
```

```
7689600.0 sec  89 Days
```

**c. Time series filtering (1 point)**

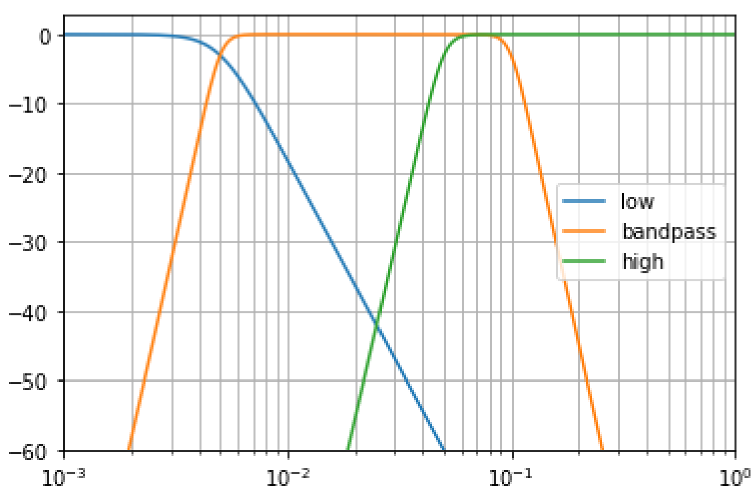
Now we will filter the trace to explore its frequency content. We will apply 3 filters:

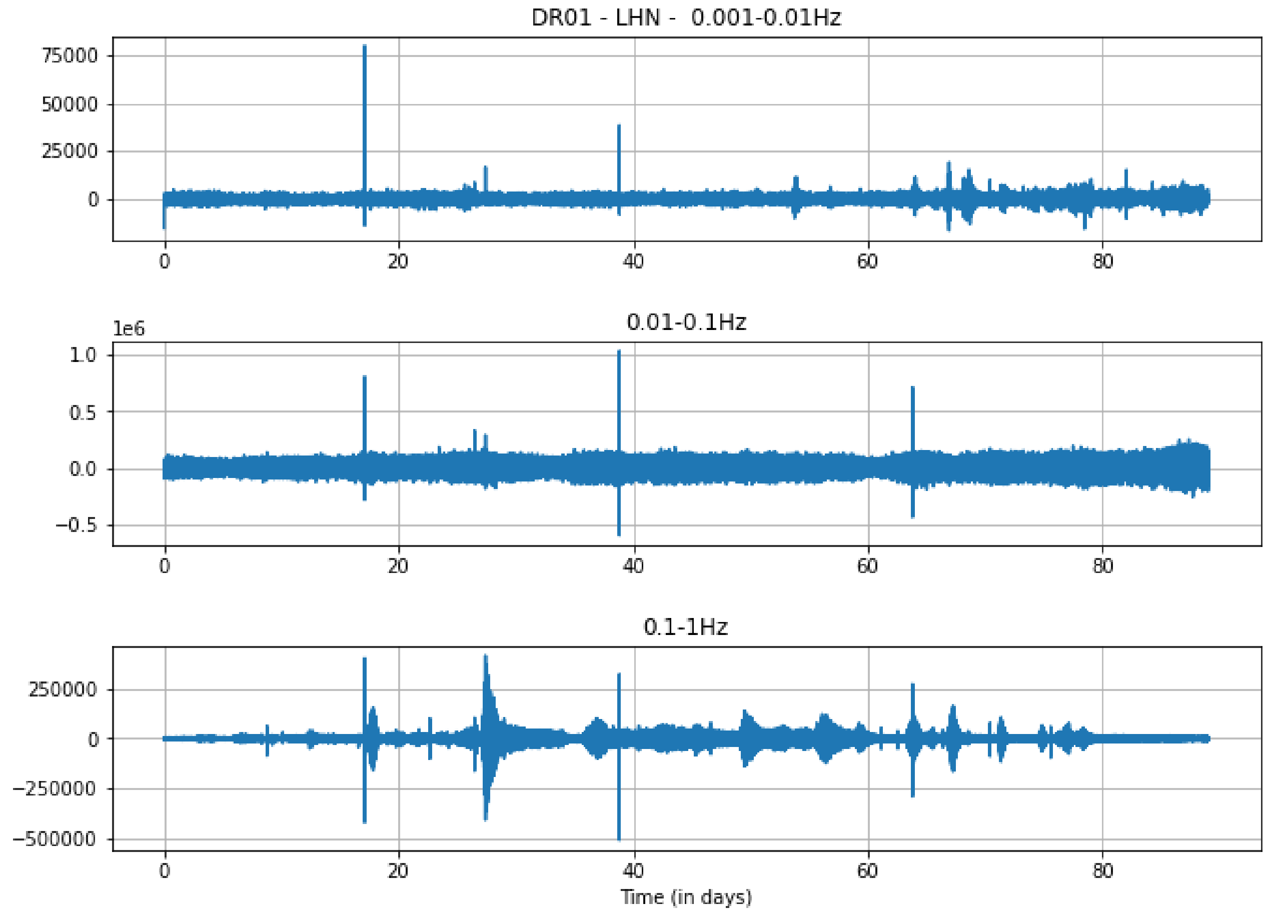
1. a `lowpass` filter to look at seismic frequencies below 0.01Hz, or 100 s period
2. a `bandpass` filter to look at seismic frequencies between 0.01Hz-0.1 Hz (10-100s)
3. a `highpass` filter to look at seismic frequencies higher than 0.1 Hz (10s) and until the time series Nyquist frequency (0.5Hz since the data is sampled at 1 Hz).

In [21]:
```python
from scipy.signal import butter,buttord,  sosfiltfilt, freqs


# Here we use a Butterworth filter to select the spectral content of the wavefor
# I like to use Buttord because it finds the order of the filter that meets the
# it's a lot more intuitive! https://docs.scipy.org/doc/scipy/reference/generate

N1, Wn1 = buttord(0.005, 0.001, 3, 40, True)
b1, a1 = butter(N1, Wn1, 'low', True)
N2, Wn2 = buttord([0.005, 0.1], [0.001, 0.2], 3, 40, True)
b2, a2 = butter(N2, Wn2, 'band', True)
N3, Wn3 = buttord(0.05, 0.1, 3, 40, True)
b3, a3 = butter(N3, Wn3, 'high', True)

w1, h1 = freqs(b1, a1, np.logspace(-3, 0, 500))
```

```
w2, h2 = freqs(b2, a2, np.logspace(-3, 0, 500))
w3, h3 = freqs(b3, a3, np.logspace(-3, 0, 500))
plt.semilogx(w1, 20 * np.log10(abs(h1)))
plt.semilogx(w2, 20 * np.log10(abs(h2)))
plt.semilogx(w3, 20 * np.log10(abs(h3)))
plt.legend(['low','bandpass','high'])
plt.axis([0.001, 1, -60, 3])
plt.grid(which='both', axis='both')


## we have to use the second order sections when filtering
## this will help us avoid transfer function errors.
sos1 = butter(N1, Wn1, 'low', output="sos")
sos2 = butter(N2, Wn2, 'band', output="sos")
sos3 = butter(N3, Wn3, 'high', output="sos")

# filter data
Z1 = sosfiltfilt(sos1, Dtmp[0].data )
Z2 = sosfiltfilt(sos2, Dtmp[0].data)
Z3 = sosfiltfilt(sos3, Dtmp[0].data)


fig,ax=plt.subplots(3,1,figsize=(11,8))
plt.subplots_adjust(hspace = 0.5)
t=np.linspace(0,Ndays,len(Dtmp[0].data))
ax[0].plot(t,Z1);ax[0].set_title('DR01 - LHN -  0.001-0.01Hz');ax[0].grid(True)
ax[1].plot(t,Z2);ax[1].set_title('0.01-0.1Hz');ax[1].grid(True)
ax[2].plot(t,Z3);ax[2].set_title('0.1-1Hz');ax[2].grid(True)
plt.xlabel('Time (in days)')
```

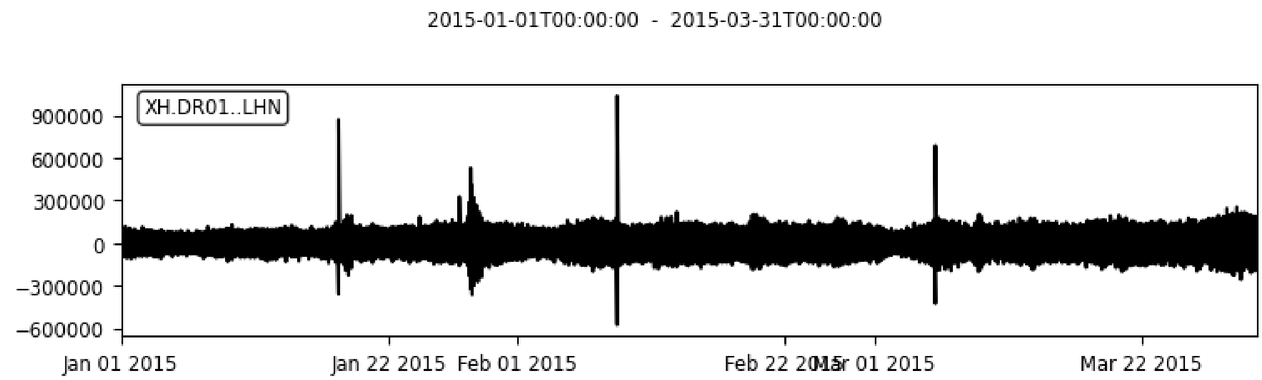
```
Out[21]: Text(0.5, 0, 'Time (in days)')
```

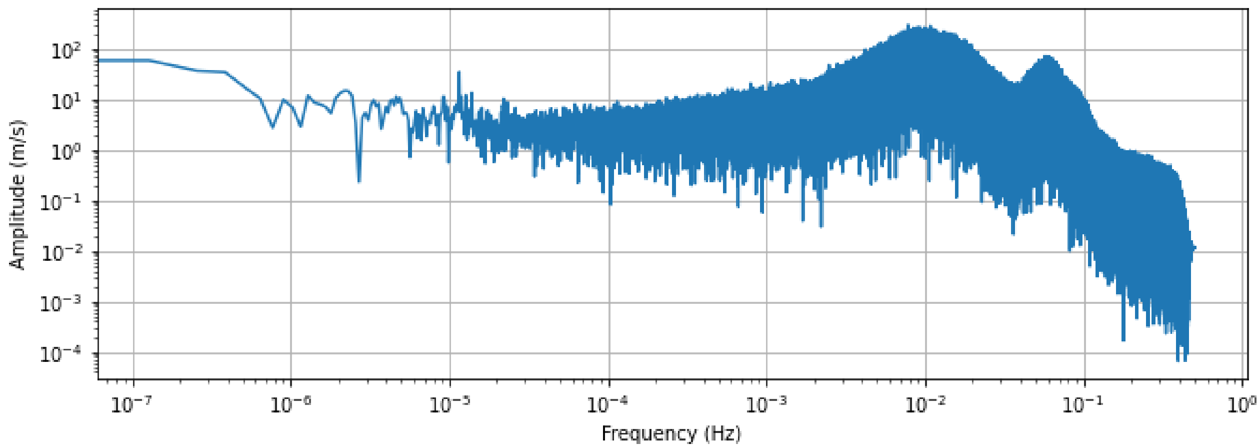


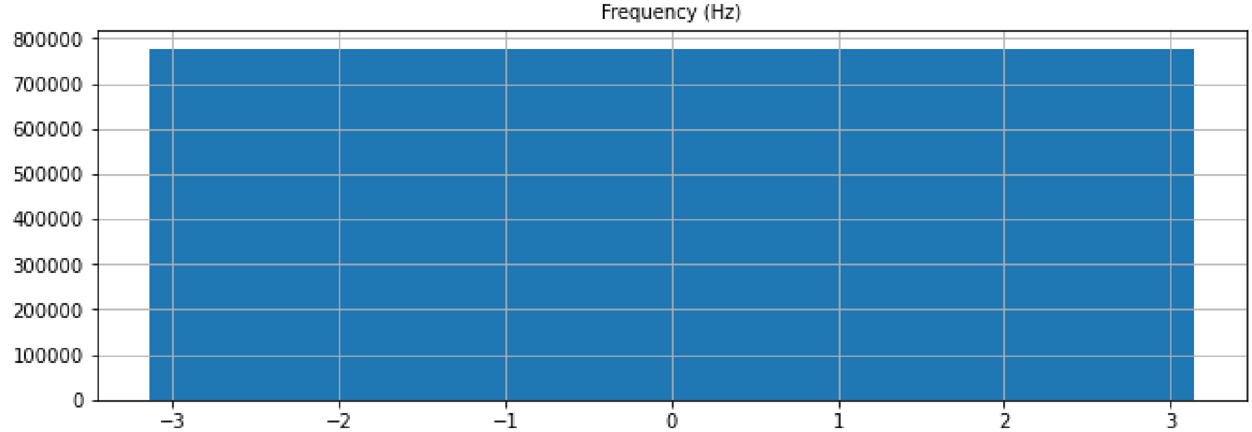



**c. Fourier transform (3 points)** Perform and the Fourier amplitude spectrum of the seismogram. Don't forget to label the figure properly! Use the Fourier frequency vector for x-axis. Use the tutorials for inspirtion.

In [22]:
```
# solution
from scipy.fftpack import fft, ifft, fftfreq, next_fast_len

npts = Dtmp[0].stats.npts
## Take FFT of the signals
Nfft = next_fast_len(int(Dtmp[0].data.shape[0]))
freqVec = fftfreq(Nfft, d=Dtmp[0].stats.delta)[:Nfft//2]
Zhat = fft(Dtmp[0].data,n=Nfft)#/np.sqrt(Z[0].stats.npts)

print()
Dtmp.plot()

fig,ax=plt.subplots(2,1,figsize=(11,8))
ax[0].plot(freqVec,np.abs(Zhat[:Nfft//2])/Nfft)
ax[0].grid(True)
ax[0].set_xscale('log');ax[0].set_yscale('log')
ax[0].set_xlabel('Frequency (Hz)');ax[0].set_ylabel('Amplitude (m/s)')
ax[1].hist(np.angle(Zhat))
ax[1].grid(True)
```




Comment on the spectral content of the seismograms. How does the relative contribution of the low, intermediate, and high frequency signal compares with the relative amplitude observed in the bandpass filtered time series?

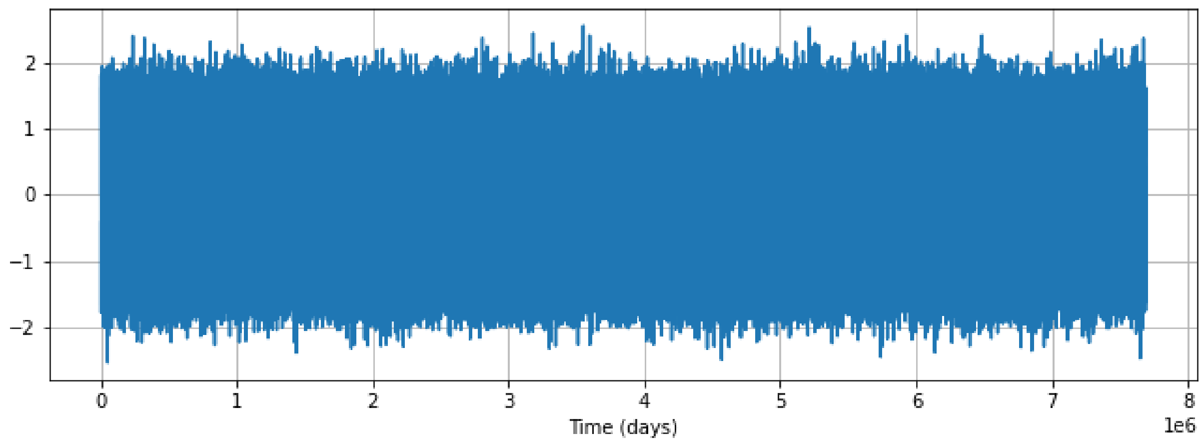
### d. Synthetic noise (3 points)

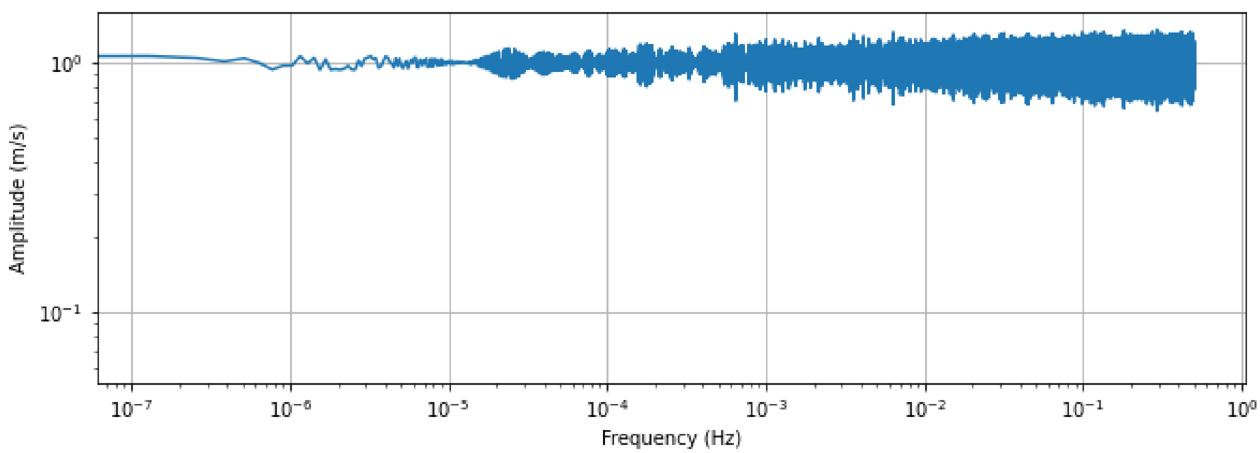
We have now a good idea of what the amplitude of seismic waves are at this station. Now create a noise signal using the Fourier amplitude spectrum of the seismic signal, and with a random phase. You can use the notes from our first Numpy example (week3_lab1.ipynb)

In [23]:
```
# solution
from numpy import random

noise = 2*random.rand(npts) - 1
T = np.linspace(0,(Tend-Tstart), Dtmp[0].stats.npts)
noisep = np.zeros(Nfft,dtype=complex)
noise = np.zeros(Nfft)
for i in range(1,Nfft//2):
```

```python
    c=2*random.rand(1)-1
    noisep[i]=  np.exp(1j*np.pi*c[0])
noisep[Nfft//2:] = np.conj(noisep[:Nfft//2-1:-1])
noise = ifft(noisep)[:npts]*np.sqrt(Nfft)

Zhat2 = fft(noise, n = Nfft)

fig, ax =  plt.subplots(2, 1, figsize = (11,8))
ax[0].plot(T, noise)
ax[0].set_xlabel("Time (days)")
ax[0].grid(True)
ax[1].plot(freqVec, np.abs(Zhat2[:Nfft//2])/np.sqrt(Nfft))
ax[1].set_xscale("log"); ax[1].set_xlabel("Frequency (Hz)")
ax[1].set_yscale("log"); ax[1].set_ylabel("Amplitude (m/s)")

ax[1].grid(True)
plt.show()
```

```
/Users/earthnote/opt/anaconda3/lib/python3.8/site-packages/numpy/core/_asarray.p
y:83: ComplexWarning: Casting complex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
```



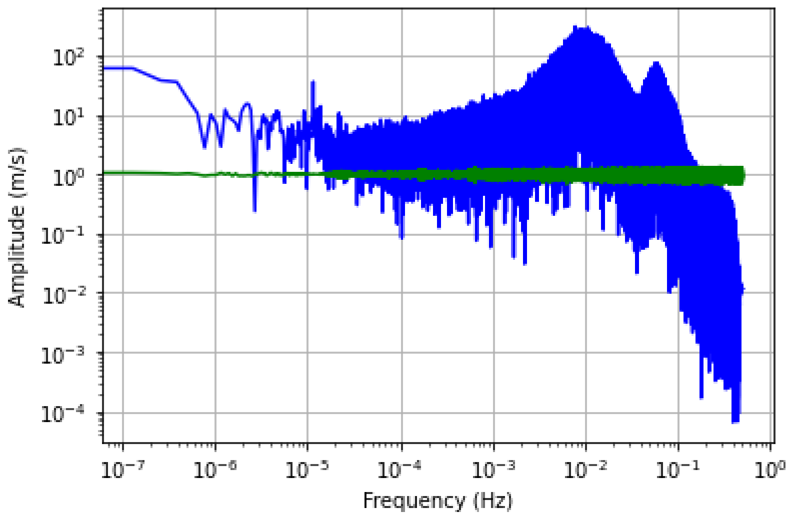

### e. !Sanity check! (1 point)

Check that the Fourier amplitude spectrum of the noise is that of the original window. Overlay them on a plot

In [24]:

```python
#solution
plt.plot(freqVec,np.abs(Zhat[:Nfft//2])/Nfft, color = 'b')
plt.plot(freqVec, np.abs(Zhat2[:Nfft//2])/np.sqrt(Nfft), color ='g')
plt.xscale('log')
plt.yscale('log')
```

```
plt.xlabel("Frequency (Hz)")
plt.ylabel("Amplitude (m/s)")
plt.grid(True)
```

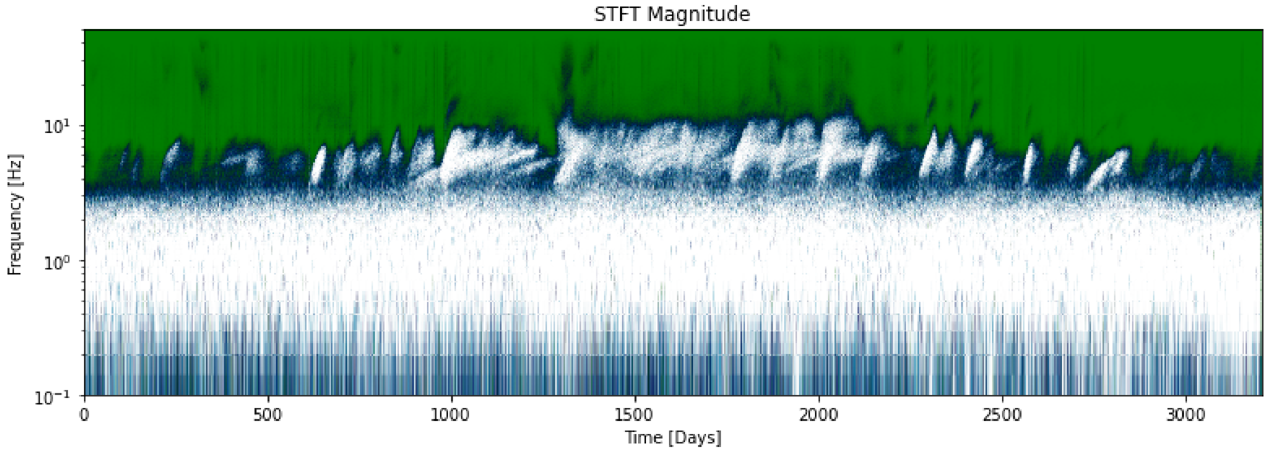


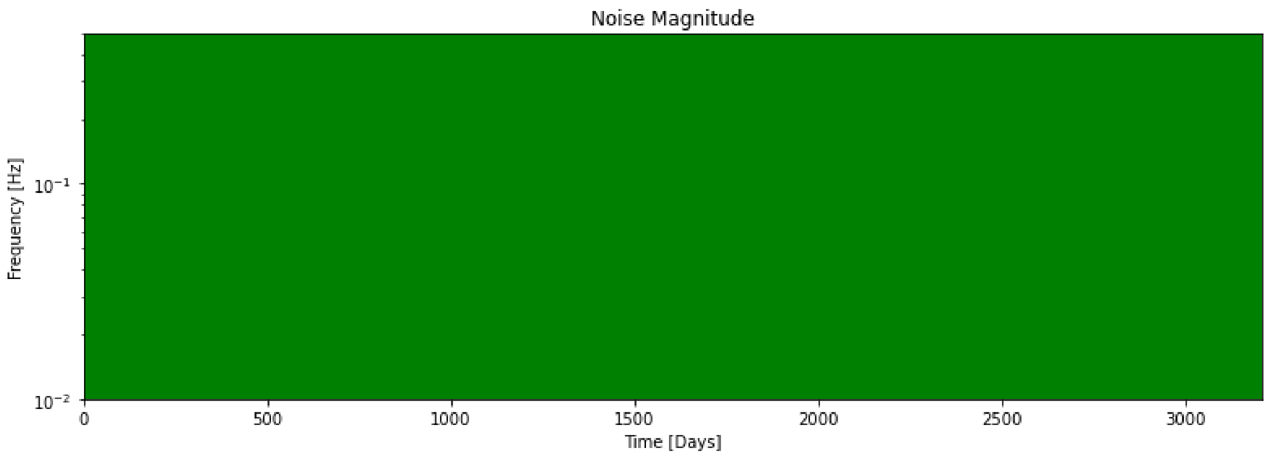

### f. Short Time Fourier Transform (4 points)

STFT are important transforms that are used in data science of time series. They are mainly used for denoising and for feature extraction. Spectrograms are STFT with window overlap.

In [25]:
```
from scipy.signal import stft

nperseg=1000

#solution
f, t, Zxx = stft(Dtmp[0].data, fs=100, nperseg=nperseg,noverlap=20)
print(np.max(np.max(np.abs(Zxx))))
fig,ax=plt.subplots(2,1,figsize=(11,8))
ax[0].pcolormesh(t/24, f, np.abs(Zxx), vmin=1, vmax=1E3, shading='gouraud',cmap=
ax[0].set_title('STFT Magnitude')
ax[0].set_ylabel('Frequency [Hz]')
ax[0].set_xlabel('Time [Days]');ax[0].set_yscale('log');ax[0].set_ylim(0.1,50)

f, t, Zxx = stft(noise, fs=100, nperseg=nperseg,noverlap=20)
print(np.max(np.max(np.abs(Zxx))))
ax[1].pcolormesh(t/24, f, np.abs(Zxx), vmin=0.1, vmax=1E2, shading='gouraud',cma
ax[1].set_title('Noise Magnitude')
ax[1].set_ylabel('Frequency [Hz]')
ax[1].set_xlabel('Time [Days]');ax[1].set_yscale('log');ax[1].set_ylim(0.01,0.5)
plt.tight_layout()
```

```
57468.256447
/Users/earthnote/opt/anaconda3/lib/python3.8/site-packages/scipy/signal/spectra
l.py:1812: UserWarning: Input data is complex, switching to return_onesided=Fals
e
  warnings.warn('Input data is complex, switching to '
0.157421068717
```




Now you have created a 2D image of a time series! Many seismologists use that as input to convolutional neural networks.

## 2) 2D Spectral analysis of geological models (10 points)

In this exercise we will correlate water table level with surface elevation. Please download the 3D Geologic framework https://www.sciencebase.gov/catalog/item/5cfeb4cce4b0156ea5645056 and https://www.sciencebase.gov/catalog/item/5e287112e4b0d3f93b03fa7f

In the following we will prepare our data.

In [26]:
```
import netCDF4 as nc
file1 = '3DGeologicFrame/NCM_GeologicFrameworkGrids.nc' # mmake sure that the fo
file2 = '3DGeologicFrame/NCM_SpatialGrid.nc'
file3 = '3DGeologicFrame/NCM_AuxData.nc'
geology = nc.Dataset(file1)
grid = nc.Dataset(file2)
watertable = nc.Dataset(file3)
```

In [27]:
```
print(grid)
```

```
<class 'netCDF4._netCDF4.Dataset'>
root group (NETCDF4 data model, file format HDF5):
    dimensions(sizes): dim1(3201), dim2(4901), dim3(311), dim4(662), dim9(28)
    variables(dimensions): |S1 Notes(dim9), float32 x(dim2, dim1), float32 y(dim
2, dim1), float64 Latitude vector(dim3), float64 Longitude vector(dim4), float64
```

```
Index j grid(dim4, dim3), float64 Index k grid(dim4, dim3)
    groups:
```

In [28]:
```python
print(geology)
```

```
<class 'netCDF4._netCDF4.Dataset'>
root group (NETCDF4 data model, file format HDF5):
    dimensions(sizes): dim1(3201), dim2(4901), dim9(31)
    variables(dimensions): |S1 Notes(dim9), float32 Surface Elevation(dim2, dim
1), float32 Bedrock Elevation(dim2, dim1), float32 Bottom Cenozoic Elevation(dim
2, dim1), float32 Bottom Phanerozoic Elevation(dim2, dim1), float32 Mid Crustal
Elevation(dim2, dim1), float32 Moho Elevation(dim2, dim1), float32 Top Ocean Pla
te Elevation(dim2, dim1)
    groups:
```

In [29]:
```python
print(watertable)
```

```
<class 'netCDF4._netCDF4.Dataset'>
root group (NETCDF4 data model, file format HDF5):
    dimensions(sizes): dim1(3201), dim2(4901), dim3(212), dim4(164), dim7(47), d
im8(26), dim9(37)
    variables(dimensions): |S1 Water table depth file(dim7), |S1 Slopes file(dim
8), |S1 Notes(dim9), float32 Water Table Depth(dim2, dim1), float32 Calibration
Slope(dim3), float32 Calibration Offset(dim4, dim3)
    groups:
```

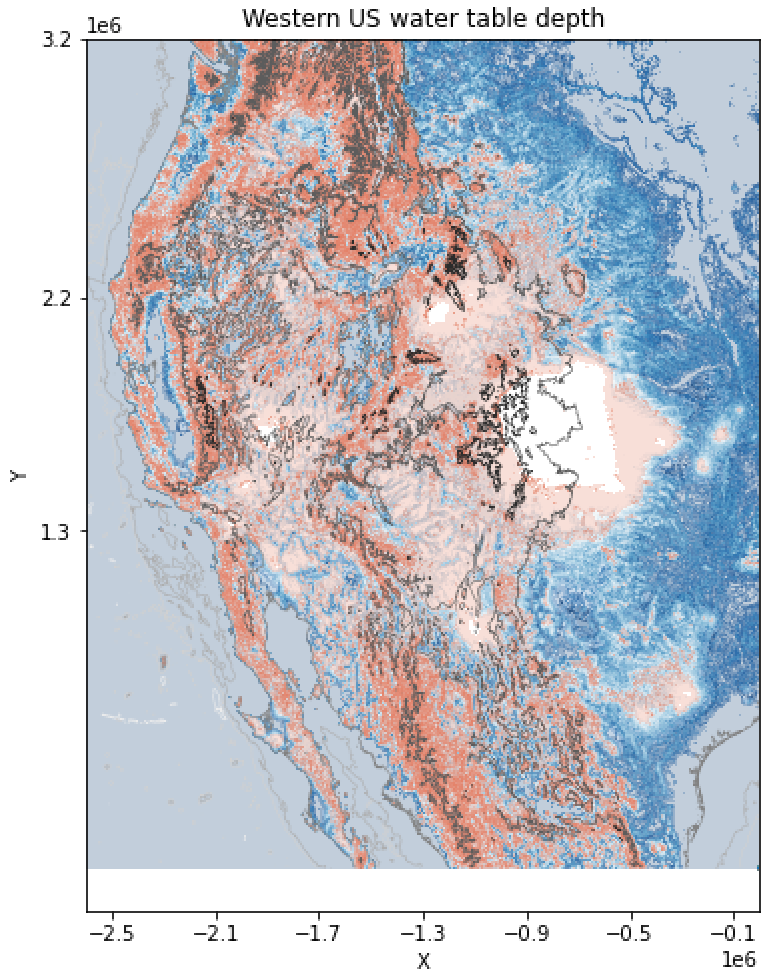
In [30]:
```python
x = grid['x'][0:4901, 0:3201]
y = grid['y'][0:4901, 0:3201]
y_ticks = grid['Index k grid'][0:4901, 0]
y_labels = grid['Latitude vector'][:]
# recreate the lat long vectors.
minlat,maxlat = min(grid['Latitude vector'][:]),max(grid['Latitude vector'][:])
minlon,maxlon = min(grid['Longitude vector'][:]),max(grid['Longitude vector'][:]
xlat = np.linspace(minlat,maxlat,3201)
xlon = np.linspace(minlon,maxlon,4901)
```

In [31]:
```python
geology['Surface Elevation'][3246, 1234]
elevation = geology['Surface Elevation'][0:4901, 0:3201]
bedrock = geology['Bedrock Elevation'][0:4901, 0:3201]
WT = watertable.variables['Water Table Depth'][0:4901, 0:3201]
```

**a. Plot (2 points)** Plot the data `WT` and `elevation`. Use `contourf`, `x` and `y` as lat-long variables. You can use `levels` to split the color map, and `alpha` less than 1 to increase transparency.

In [38]:
```python
fig = plt.figure(figsize=(11,8))
ax = fig.add_subplot(111)
ax.plot(WT, elevation)
ax.contourf(x, y, WT,cmap="RdBu_r",levels=[0,10,20,30,40,50,60,70,80,90,100,200]
ax.contour(x, y, elevation,cmap="Greys",linewidths=0.5)
ax.set_aspect('equal','box')
ax.set_xlim(-2.6E6,0);
ax.set_xticks((-2.5e+06,-2.1e+06,-1.7e+06,-1.3e+06,-0.9e+06,-0.5e+06,-0.1e+06))
ax.set_yticks((1.3e+06,2.2e+06,3.2e+06))
ax.set_title('Western US water table depth')
ax.set_ylabel('Y')
ax.set_xlabel('X')
```

Out[38]:
```
Text(0.5, 0, 'X')
```

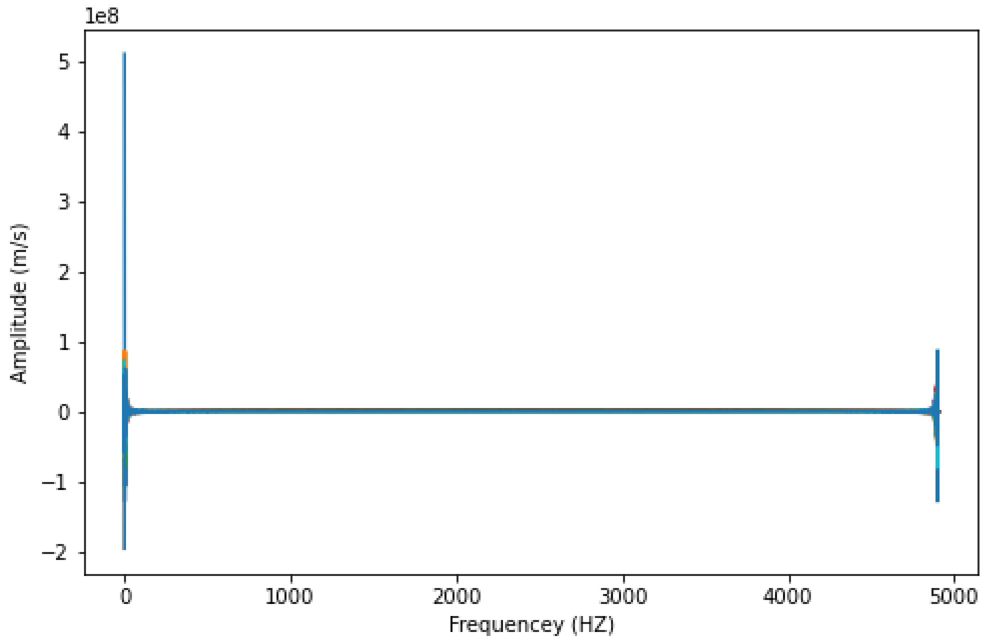


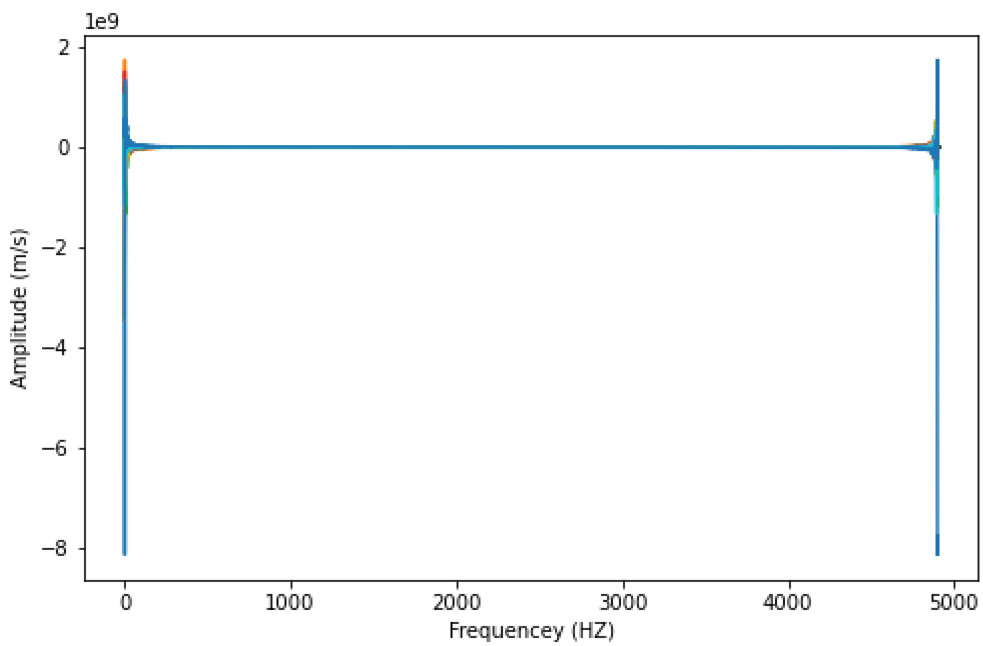

**b. Perform and plot the 2D Fourier transforms (4 points)**

In [33]:
```
from scipy.fftpack import fft2, fftfreq,fftshift
#solution

wt_fourier = fft2(WT)
ele_fourier = fft2(elevation)

fig, ax = plt.subplots(2,1, figsize = (8,11))
ax[0].plot(wt_fourier)

ax[0].set_xlabel('Frequencey (HZ)')
ax[0].set_ylabel("Amplitude (m/s)")
ax[1].plot(ele_fourier)

ax[1].set_xlabel('Frequencey (HZ)')
ax[1].set_ylabel("Amplitude (m/s)")
```

```
/Users/earthnote/opt/anaconda3/lib/python3.8/site-packages/numpy/core/_asarray.p
y:83: ComplexWarning: Casting complex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
/Users/earthnote/opt/anaconda3/lib/python3.8/site-packages/numpy/core/_asarray.p
y:83: ComplexWarning: Casting complex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order)
```

Out[33]:
```
Text(0, 0.5, 'Amplitude (m/s)')
```




**c. Interpretation (1 point)** Comment on the wavelengths that dominate the DEM and the water table wavelengths

dominate wavelenths, DEM and the watertable are very large and minute wavelengths. Wavelengths are inversely propotional to frequency. As a result, frequencies near 0 are very large wavelengths, and the frequencies near 5000 are at much smaller wavelengths.

**d. 2D filtering (3 points)** Find a way to low pass filter the image (spectral filtering or convolution)

In [37]:

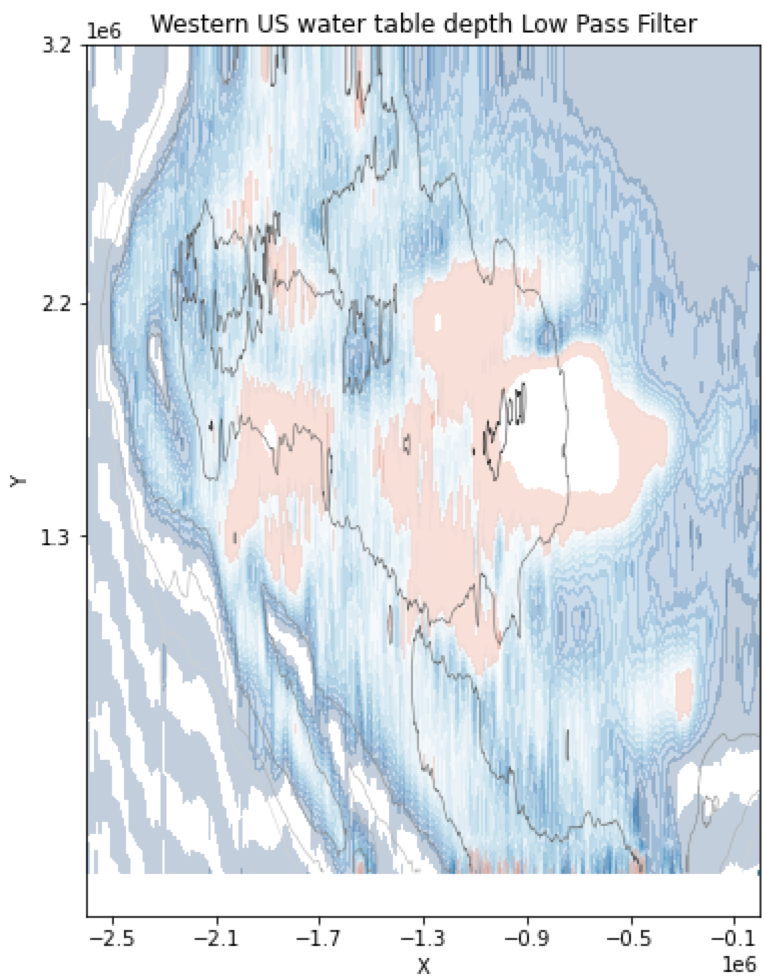
```
from scipy.signal import butter,buttord,  sosfiltfilt, freqs
N1, Wn1 = buttord(0.005, 0.001, 3, 40, True)
sos1 = butter(N1, Wn1, 'low', output="sos")
WT_sos = sosfiltfilt(sos1, WT.data)
Elevation_1 = sosfiltfilt(sos1, elevation.data)

fig = plt.figure(figsize=(11,8))
ax = fig.add_subplot(111)
```

```python
ax.plot(WT_sos, Elevation_1)
ax.contourf(x, y, WT_sos,cmap="RdBu_r",levels=[0,10,20,30,40,50,60,70,80,90,100,
ax.contour(x, y, Elevation_1,cmap="Greys",linewidths=0.5)
print(y.max())
ax.set_aspect('equal','box')
ax.set_xlim(-2.6E6,0);
ax.set_xticks((-2.5e+06,-2.1e+06,-1.7e+06,-1.3e+06,-0.9e+06,-0.5e+06,-0.1e+06))
ax.set_yticks((1.3e+06,2.2e+06,3.2e+06))
ax.set_ylabel('Y')
ax.set_xlabel('X')
ax.set_title('Western US water table depth Low Pass Filter')
```

```
3.2e+06
```

Out[37]:

```
Text(0.5, 1.0, 'Western US water table depth Low Pass Filter')
```




In [ ]: