PythonNotebook8_solution_2023

January 26, 2024

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from datetime import datetime
import matplotlib
```

[2]:
```
ice = pd.read_csv('T1_ice.csv')
# if the file cannot be found, you will get an error message here.
```

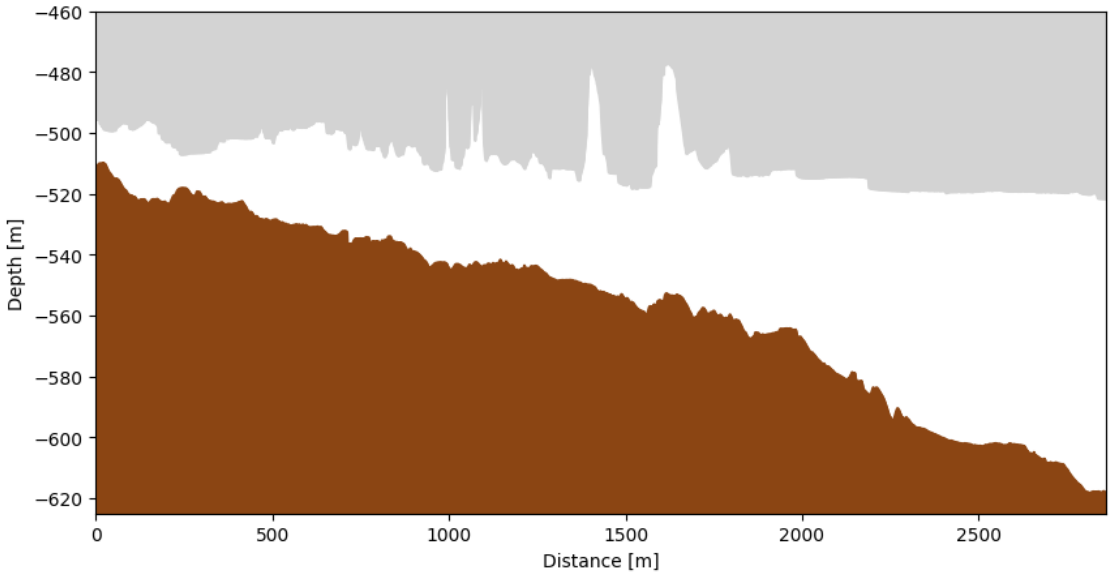
[3]:
```
seafloor = pd.read_csv('T1_seafloor.csv')
```

0.1 Exercise 8.1.1 Improve the above plot to make it easier to understand

- use `plt.fill_between` to color everything below the seafloor line brown
- use `plt.fill_between` to color everything above the ice base line grey
- adjust the limits of the x- and y-axis to make the plot look nice

[4]:
```
plt.figure(figsize=(10,5))
plt.plot(ice['distance(m)'],ice['ice(m)'], color='lightgrey')
plt.plot(seafloor['distance(m)'],seafloor['seafloor(m)'], color='saddlebrown')
plt.xlabel('Distance [m]')
plt.ylabel('Depth [m]')
plt.xlim(np.min(seafloor['distance(m)']),np.max(seafloor['distance(m)']))

# SOLUTION #
plt.fill_between(seafloor['distance(m)'],seafloor['seafloor(m)'],␣
 ↪-625,color='saddlebrown')
plt.fill_between(ice['distance(m)'],ice['ice(m)'], -460,color='lightgrey')
plt.ylim(-625,-460)
```

[4]:
```
(-625.0, -460.0)
```



0.2 Exercise 8.1.2 Load another csv file and plot the data onto the plot from 8.1.1

- use `pd.load_csv` to import `T1_ocean_1Hz.csv` into a dataframe called `ocean`
- check what the dataframe looks like
- drop columns so that you are only left with: `time(UTC),distance(m),depth(m),Absolute_Salinity(g/kg)`
- use `plt.scatter` to plot the depth as a function of distance
- use the `s` argument to change the size of the plotted markers

[5]:
```
# Load and check data

# SOLUTION #
ocean = pd.read_csv('T1_ocean_1Hz.csv')
ocean
```

[5]:

	time(UTC)	latitude	longitude	distance(m)	depth(m)	\
0	10-Jan-2020 20:16:56	-75.2071	-104.8253	2861.5250	-613.3393	
1	10-Jan-2020 20:16:56	-75.2071	-104.8253	2861.4987	-613.3446	
2	10-Jan-2020 20:16:56	-75.2071	-104.8253	2861.4575	-613.4449	
3	10-Jan-2020 20:16:56	-75.2071	-104.8253	2861.4022	-613.5580	
4	10-Jan-2020 20:16:56	-75.2071	-104.8253	2861.3292	-613.6709	
…	…	…	…	…	…	
17954	09-Jan-2020 20:34:22	-75.2222	-104.7969	0.5095	-501.9610	
17955	09-Jan-2020 20:34:22	-75.2222	-104.7969	0.3128	-501.8992	
17956	09-Jan-2020 20:34:22	-75.2222	-104.7969	0.1738	-501.8257	
17957	09-Jan-2020 20:34:22	-75.2222	-104.7969	0.0737	-501.7498	
17958	09-Jan-2020 20:34:22	-75.2222	-104.7969	-0.0000	-501.7370	

```
       pressure(db)  insitu_temperature(C)  conductivity(mS/cm)  \
0          620.7120                 0.0508              28.9044
1          620.7174                 0.0509              28.9044
2          620.8190                 0.0509              28.9039
3          620.9336                 0.0512              28.9036
4          621.0481                 0.0514              28.9032
…             …                       …                    …
17954      507.8577                -0.7238              27.9853
17955      507.7951                -0.7238              27.9853
17956      507.7207                -0.7238              27.9845
17957      507.6437                -0.7241              27.9842
17958      507.6308                -0.7243              27.9839

       Conservative_Temperature(C)  Absolute_Salinity(g/kg)  \
0                           0.0284                  34.5637
1                           0.0284                  34.5636
2                           0.0285                  34.5629
3                           0.0287                  34.5621
4                           0.0290                  34.5612
…                            …                      …
17954                      -0.7362                  34.2727
17955                      -0.7362                  34.2727
17956                      -0.7362                  34.2717
17957                      -0.7365                  34.2717
17958                      -0.7367                  34.2714

       thermal_driving(C)  density_anomaly(kg/m^3)  potential_temperature(C)  \
0                  2.3921                  27.6228                    0.0144
1                  2.3922                  27.6227                    0.0144
2                  2.3923                  27.6222                    0.0145
3                  2.3925                  27.6215                    0.0147
4                  2.3928                  27.6208                    0.0149
…                   …                      …                          …
17954              1.5218                  27.4251                   -0.7498
17955              1.5217                  27.4251                   -0.7498
17956              1.5216                  27.4243                   -0.7498
17957              1.5213                  27.4244                   -0.7501
17958              1.5210                  27.4241                   -0.7503

       practical_salinity(psu)  speed_of_sound(m/s)  dissolved_oxygen(ml/l)
0                      34.3945            1458.6991                  4.4275
1                      34.3944            1458.6994                  4.4277
2                      34.3937            1458.7004                  4.4282
3                      34.3929            1458.7023                  4.4287
4                      34.3920            1458.7042                  4.4290
…                        …                  …                        …
17954                  34.1051            1452.8557                  4.5015
```

```
17955                34.1051            1452.8546                4.5015
17956                34.1041            1452.8521                4.5015
17957                34.1041            1452.8494                4.5017
17958                34.1038            1452.8478                4.5017

[17959 rows x 16 columns]
```

[6]:
```
# Drop columns

# SOLUTION #
ocean = ocean.
 ↪drop(columns=['latitude','longitude','pressure(db)','insitu_temperature(C)','conductivity(m
 ↪cm)','Conservative_Temperature(C)','thermal_driving(C)','density_anomaly(kg/
 ↪m^3)','potential_temperature(C)','practical_salinity(psu)','speed_of_sound(m/
 ↪s)','dissolved_oxygen(ml/l)'])
ocean
```

[6]:

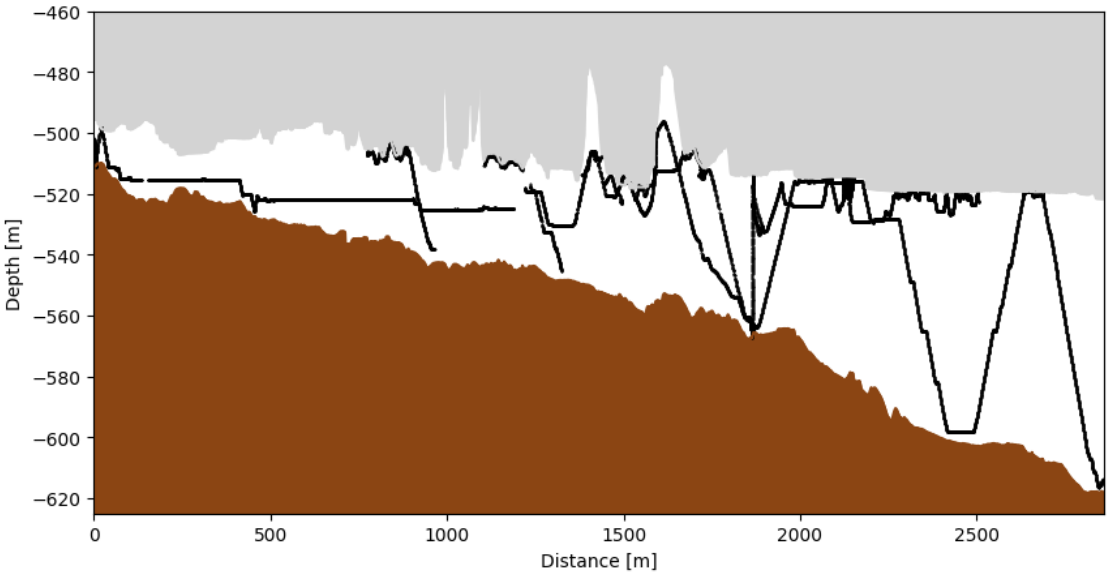
	time(UTC)	distance(m)	depth(m)	Absolute_Salinity(g/kg)
0	10-Jan-2020 20:16:56	2861.5250	-613.3393	34.5637
1	10-Jan-2020 20:16:56	2861.4987	-613.3446	34.5636
2	10-Jan-2020 20:16:56	2861.4575	-613.4449	34.5629
3	10-Jan-2020 20:16:56	2861.4022	-613.5580	34.5621
4	10-Jan-2020 20:16:56	2861.3292	-613.6709	34.5612
…	…	…	…	…
17954	09-Jan-2020 20:34:22	0.5095	-501.9610	34.2727
17955	09-Jan-2020 20:34:22	0.3128	-501.8992	34.2727
17956	09-Jan-2020 20:34:22	0.1738	-501.8257	34.2717
17957	09-Jan-2020 20:34:22	0.0737	-501.7498	34.2717
17958	09-Jan-2020 20:34:22	-0.0000	-501.7370	34.2714

[17959 rows x 4 columns]

[7]:
```
# Plot the depth as function of distance and use `s` to change ther marker size

plt.figure(figsize=(10,5))
plt.plot(ice['distance(m)'],ice['ice(m)'], color='lightgrey')
plt.plot(seafloor['distance(m)'],seafloor['seafloor(m)'], color='saddlebrown')
plt.xlabel('Distance [m]')
plt.ylabel('Depth [m]')
plt.xlim(np.min(seafloor['distance(m)']),np.max(seafloor['distance(m)']))

# Fill in the solution from 6.2.2 here #
plt.fill_between(seafloor['distance(m)'],seafloor['seafloor(m)'],␣
 ↪-625,color='saddlebrown')
plt.fill_between(ice['distance(m)'],ice['ice(m)'], -460,color='lightgrey')
plt.ylim(-625,-460)
```

```
# SOLUTION #

# Add scatter plot here#
plt.scatter(ocean['distance(m)'],ocean['depth(m)'],s=.2, color='k')
```

[7]: <matplotlib.collections.PathCollection at 0x269d62c7a00>



0.3 Exercise 8.1.3 dip deeper into the scatter plot

Now that we have plotted the journey of ICEFIN it would be nice to plot some of the components that it has been measuring on it's journey below the ice shelf.

With scatter plots we are able to color the markers based on data. In this exercise we will use the `Absolute_Salinity(g/kg)` column as data to be colored.

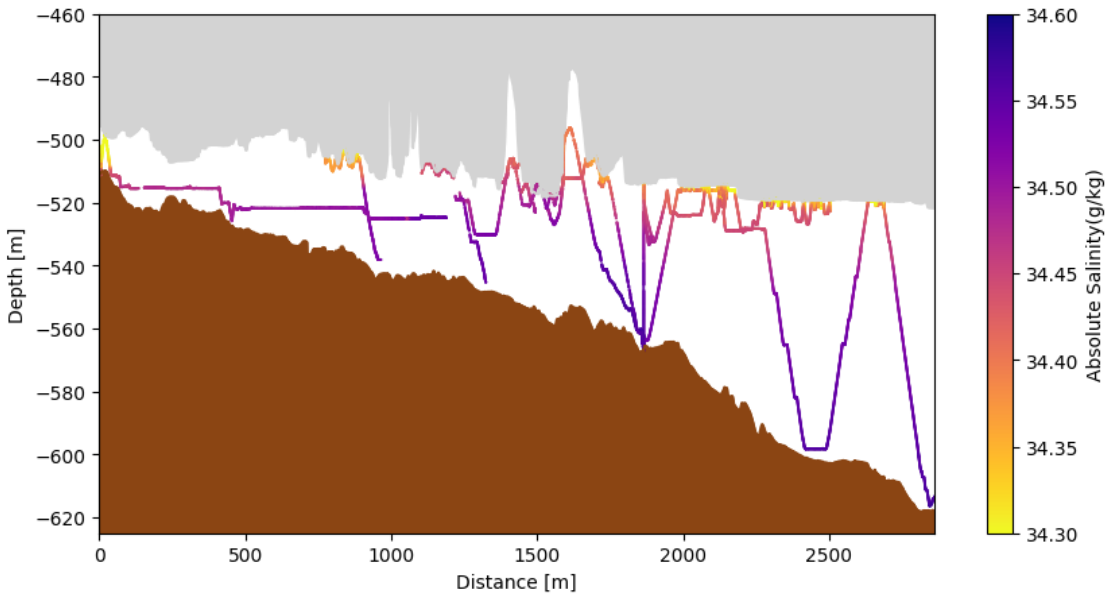
Have a look at the scatter plot documentation!

- copy and use the same scatter plot as above
- parse the `Absolute_Salinity(g/kg)` column to the `c` argument of the scatter plot
- use the `cmap` argument to add a colormap of choice
- use the `vmin` and `vmax` arguments to control the range of the colormap (`vmin = 34.3` and `vmax = 34.6` should be good)
- add a colorbar to the plot

[8]:
```
plt.figure(figsize=(10,5))
plt.plot(ice['distance(m)'],ice['ice(m)'], color='lightgrey')
plt.plot(seafloor['distance(m)'],seafloor['seafloor(m)'], color='saddlebrown')
plt.xlabel('Distance [m]')
plt.ylabel('Depth [m]')
```

```
plt.xlim(np.min(seafloor['distance(m)']),np.max(seafloor['distance(m)']))

# Fill in the solution from 6.2.2 here #
plt.fill_between(seafloor['distance(m)'],seafloor['seafloor(m)'],␣
↪-625,color='saddlebrown')
plt.fill_between(ice['distance(m)'],ice['ice(m)'], -460,color='lightgrey')
plt.ylim(-625,-460)

# SOLUTION #

# Add scatter plot here #
# cm = plt.cm.get_cmap('plasma_r')
cm = matplotlib.colormaps.get_cmap('plasma_r')
vmin = 34.3
vmax = 34.6
plt.scatter(ocean['distance(m)'],ocean['depth(m)'],c=ocean['Absolute_Salinity(g/
↪kg)'], s=.2, cmap=cm,vmin=vmin,vmax = vmax)
plt.colorbar(label='Absolute Salinity(g/kg)')
```

[8]: <matplotlib.colorbar.Colorbar at 0x269d6380550>



[9]:
```
ocean_mean = ocean.groupby(['time(UTC)'],as_index=False).mean()
```

0.4 Exercise 8.2.1 creating a column with distances

In the example below we create a new column `time_delta` which contains the time it takes from ICEFIN to get from point `i` to `i+1` and the result is stored at `i+1`.

It is now your task to create a ned column which contains the distance travelled between point `i` and `i+1`, and store the result at `i+1`.

- create a new column called `travelled_distance` with NaN values in every row
- use the columns `distance(m)` and `depth(m)` to calculate the eucledian distance and store the result in `travelled_distance`

Euclidean distance:

$$dist = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

[10]:
```
# Creating a new column with NaN values
ocean_mean['time_delta'] = np.nan
```

[11]:
```
# Loop through the dataframe and fill in the new column

for i in range(len(ocean_mean)-1):
    date_format = '%d-%b-%Y %H:%M:%S'
    time2 = datetime.strptime(ocean_mean['time(UTC)'].iloc[i+1], date_format)
    time1 = datetime.strptime(ocean_mean['time(UTC)'].iloc[i], date_format)
    ocean_mean['time_delta'].at[i+1] = (time2-time1).seconds
ocean_mean
```

[11]:

	time(UTC)	distance(m)	depth(m)	Absolute_Salinity(g/kg)	\
0	09-Jan-2020 05:30:02	1864.504000	-519.591700	34.443100	
1	09-Jan-2020 05:30:11	1864.504000	-522.580738	34.458025	
2	09-Jan-2020 05:30:20	1864.504000	-527.018589	34.476633	
3	09-Jan-2020 05:30:28	1864.504000	-531.471256	34.509967	
4	09-Jan-2020 05:30:37	1864.504000	-535.751850	34.521138	
…	…	…	…	…	
2563	11-Jan-2020 23:10:53	2499.645878	-520.624211	34.376311	
2564	11-Jan-2020 23:11:02	2500.899763	-519.747450	34.299400	
2565	11-Jan-2020 23:11:11	2502.174011	-520.237222	34.317922	
2566	11-Jan-2020 23:11:19	2503.573775	-521.522025	34.372900	
2567	11-Jan-2020 23:11:28	2504.304733	-522.464750	34.400467	

	time_delta
0	NaN
1	9.0
2	9.0
3	8.0
4	9.0
…	…
2563	8.0
2564	9.0
2565	9.0
2566	8.0
2567	9.0

[2568 rows x 5 columns]

[12]:
```
# Create a new column with NaN values

# SOLUTION #

ocean_mean['travelled_distance'] = np.nan
ocean_mean
```

[12]:

	time(UTC)	distance(m)	depth(m)	Absolute_Salinity(g/kg)	\
0	09-Jan-2020 05:30:02	1864.504000	-519.591700	34.443100	
1	09-Jan-2020 05:30:11	1864.504000	-522.580738	34.458025	
2	09-Jan-2020 05:30:20	1864.504000	-527.018589	34.476633	
3	09-Jan-2020 05:30:28	1864.504000	-531.471256	34.509967	
4	09-Jan-2020 05:30:37	1864.504000	-535.751850	34.521138	
…	…	…	…	…	
2563	11-Jan-2020 23:10:53	2499.645878	-520.624211	34.376311	
2564	11-Jan-2020 23:11:02	2500.899763	-519.747450	34.299400	
2565	11-Jan-2020 23:11:11	2502.174011	-520.237222	34.317922	
2566	11-Jan-2020 23:11:19	2503.573775	-521.522025	34.372900	
2567	11-Jan-2020 23:11:28	2504.304733	-522.464750	34.400467	

	time_delta	travelled_distance
0	NaN	NaN
1	9.0	NaN
2	9.0	NaN
3	8.0	NaN
4	9.0	NaN
…	…	…
2563	8.0	NaN
2564	9.0	NaN
2565	9.0	NaN
2566	8.0	NaN
2567	9.0	NaN

[2568 rows x 6 columns]

[13]:
```
# Fill in the column with the euclidean distance

# SOLUTION #

for i in range(len(ocean_mean)-1):
    ocean_mean['travelled_distance'].at[i+1] = np.
 ↪sqrt((ocean_mean['distance(m)'].iloc[i+1] - ocean_mean['distance(m)'].
 ↪iloc[i]) ** 2 + (ocean_mean['depth(m)'].iloc[i+1] - ocean_mean['depth(m)'].
 ↪iloc[i]) ** 2)
ocean_mean
```

[13]:
```
                 time(UTC)  distance(m)    depth(m)  Absolute_Salinity(g/kg)  \
0     09-Jan-2020 05:30:02  1864.504000 -519.591700                34.443100
1     09-Jan-2020 05:30:11  1864.504000 -522.580738                34.458025
2     09-Jan-2020 05:30:20  1864.504000 -527.018589                34.476633
3     09-Jan-2020 05:30:28  1864.504000 -531.471256                34.509967
4     09-Jan-2020 05:30:37  1864.504000 -535.751850                34.521138
…                      …          …          …                      …
2563  11-Jan-2020 23:10:53  2499.645878 -520.624211                34.376311
2564  11-Jan-2020 23:11:02  2500.899763 -519.747450                34.299400
2565  11-Jan-2020 23:11:11  2502.174011 -520.237222                34.317922
2566  11-Jan-2020 23:11:19  2503.573775 -521.522025                34.372900
2567  11-Jan-2020 23:11:28  2504.304733 -522.464750                34.400467

      time_delta  travelled_distance
0            NaN                 NaN
1            9.0            2.989038
2            9.0            4.437851
3            8.0            4.452667
4            9.0            4.280594
…          …                 …
2563         8.0            1.678182
2564         9.0            1.530012
2565         9.0            1.365132
2566         8.0            1.900015
2567         9.0            1.192908

[2568 rows x 6 columns]
```

0.5 Exercise 8.2.2 calculating the velocity

In the example below we create a new column `time_delta` which contains the time it takes from ICEFIN to get from point `i` to `i+1` and the result is stored at `i+1`.

It is now your task to create a ned column which contains the distance travelled between point `i` and `i+1`, and store the result at `i+1`.

- create a new column with the velocity using `time_delta` and `travelled_distance`
- what is the maximum velocity?
- what is the minimum velocity?
- what is the mean velocity?

Hint: Since there might be NaN values it might be needed to use i.e. `np.nanmin()` instead of `np.min()`

[14]:
```
# Create a new column with the velocities

# SOLUTION #
```

```
ocean_mean['velocity'] = ocean_mean['travelled_distance']/
 ↪ocean_mean['time_delta']
ocean_mean
```

[14]:

	time(UTC)	distance(m)	depth(m)	Absolute_Salinity(g/kg)	\
0	09-Jan-2020 05:30:02	1864.504000	-519.591700	34.443100	
1	09-Jan-2020 05:30:11	1864.504000	-522.580738	34.458025	
2	09-Jan-2020 05:30:20	1864.504000	-527.018589	34.476633	
3	09-Jan-2020 05:30:28	1864.504000	-531.471256	34.509967	
4	09-Jan-2020 05:30:37	1864.504000	-535.751850	34.521138	
…	…	…	…	…	
2563	11-Jan-2020 23:10:53	2499.645878	-520.624211	34.376311	
2564	11-Jan-2020 23:11:02	2500.899763	-519.747450	34.299400	
2565	11-Jan-2020 23:11:11	2502.174011	-520.237222	34.317922	
2566	11-Jan-2020 23:11:19	2503.573775	-521.522025	34.372900	
2567	11-Jan-2020 23:11:28	2504.304733	-522.464750	34.400467	

	time_delta	travelled_distance	velocity
0	NaN	NaN	NaN
1	9.0	2.989038	0.332115
2	9.0	4.437851	0.493095
3	8.0	4.452667	0.556583
4	9.0	4.280594	0.475622
…	…	…	…
2563	8.0	1.678182	0.209773
2564	9.0	1.530012	0.170001
2565	9.0	1.365132	0.151681
2566	8.0	1.900015	0.237502
2567	9.0	1.192908	0.132545

[2568 rows x 7 columns]

[15]:
```
# Calculate the max, min and mean velocities

# SOLUTION #

min_vel = np.nanmin(ocean_mean['velocity'])
max_vel = np.nanmax(ocean_mean['velocity'])
mean_vel = np.nanmean(ocean_mean['velocity'])

print(f'The minimum velocity is {min_vel:.2f} m/s, the maximum velocity is␣
 ↪{max_vel:.2f} m/s, and the mean velocity is {mean_vel:.2f} m/s')
```

```
The minimum velocity is 0.00 m/s, the maximum velocity is 0.65 m/s, and the mean
velocity is 0.22 m/s
```