Import relevant packages here.

In [236...
```
import matplotlib.pyplot as plt
import pandas as pd
import chardet
import math
```

Load the data and verify it is loaded correctly.

- Print it (head, tail, or specific rows, choose a sensible number of rows).
- Compare it to the source file.

In [237...
```
file_path = 'cf_data.csv'
with open(file_path , 'rb') as file:
    result = chardet.detect(file.read(100000))
result

df = pd.read_csv(file_path , encoding = 'ascii')
```

In [238...
```
dq = df[0:51]
dq
```

Out[238]:

	dv	s	a
0	-0.743240	53.5427	1.242570
1	-0.557230	53.6120	1.777920
2	-0.454769	53.6541	0.544107
3	-0.525396	53.7030	-0.294755
4	-0.601285	53.7592	-0.290961
5	-0.682448	53.8232	-0.283414
6	-0.768859	53.8957	-0.271604
7	-0.860452	53.9770	-0.133532
8	-0.832777	54.0678	0.243356
9	-0.576125	54.1436	0.406759
10	-0.423120	54.1830	0.132934
11	-0.482842	54.2282	-0.036750
12	-0.546495	54.2796	0.252901
13	-0.666451	54.3375	1.016250
14	-0.889424	54.4128	1.348450
15	-1.077630	54.5154	0.801785
16	-1.185320	54.6284	0.457602
17	-1.298860	54.7524	0.539608
18	-1.297390	54.7758	0.628472
19	-1.048450	54.8996	0.724280
20	-0.896299	54.9855	0.827068
21	-0.971692	55.0788	0.936818
22	-1.048590	55.1798	1.053460
23	-1.045840	55.2885	1.176890
24	-0.826029	55.3890	1.306950
25	-0.595237	55.4537	0.952171
26	-0.377534	55.5080	-0.325259
27	-0.179251	55.5292	-0.586766
28	-0.177712	55.4315	0.904121
29	-0.428135	55.4524	1.319240
30	-0.520055	55.5171	0.915649
31	-0.368377	55.5564	1.500660
32	-0.289445	55.5908	2.104120
33	-0.090540	55.6143	2.301710
34	0.161186	55.6089	2.006080
35	0.429164	55.5821	0.773532

	dv	s	a
36	0.605079	55.5231	-0.052977
37	0.680615	55.4611	-0.421825
38	0.634645	55.3870	-1.510650
39	0.515043	55.3341	-1.435200
40	0.330924	55.2840	1.494930
41	0.444507	55.2680	3.767860
42	1.010490	55.1951	2.147950
43	1.118990	55.0659	0.447902
44	-0.981924	27.1318	0.238812
45	-0.836237	27.2270	1.174030
46	-0.680288	27.2991	1.272390
47	-0.514144	27.3631	0.886180
48	-0.435262	27.4019	-0.433038
49	-0.533494	27.4501	-1.360100
50	-0.641372	27.5086	-1.446930

In the ensuing, you will use `numpy`.

Let's create a grid for the values to plot. But first create **two arrays named `dv` and `s`** using `numpy.linspace` that hold the grid values at the relevant indices in their respective dimension of the grid.

Create a **grid named `a`** with zeros using `numpy.zeros` in to which calculated acceleration values can be stored.

Let the grid span:

- Speed difference `dv` [m/s]
  - From -10 till 10
  - With 41 evenly spaced values
- Headway `s` [m]
  - From 0 till 200
  - With 21 evenly spaced values

In [239…

```
import numpy as np
dv = np.linspace(-10,10,num = 41)
# print(dv)
s = np.linspace(0,200,num = 21)
# print(s)
a = np.zeros((21,41)) # (rows , columns)
# print(a)
```

Create from the imported data 3 separate `numpy` arrays for each column `dv`, `s` and `a`. (We do this for speed reasons later.)

- Make sure to name them differently from the arrays that belong to the grid as above.
- You can access the data of each column in a `DataFrame` using `data.xxx` where `xxx` is the column name (not as a string).
- Use the method `to_numpy()` to convert a column to a `numpy` array.

In [240…

```
DV = df.dv.to_numpy()
print(DV)
S = df.s.to_numpy()
print(S)
A =df.a.to_numpy()
print(A)
```

```
[-0.74324  -0.55723  -0.454769 ...  5.13764   5.15348   5.25868 ]
[ 53.5427  53.612   53.6541 ... 115.118  114.599  113.112 ]
[ 1.24257   1.77792   0.544107 ...  0.232283  0.262078 -0.61244 ]
```

Create an algorithm that calculates all the acceleration values and stores them in the grid. The algorithm is described visually in the last part of the lecture. At each grid point, it calculates a weighted mean of all measurements. The weights are given by an exponential function, based on the 'distance' between the grid point, and the measurement values of `dv` and `s`. To get you started, how many `for` -loops do you need?

For this you will need `math`.
Use an *upsilon* of 1.5m/s and a *sigma* of 30m.

Warning: This calculation may take some time. So:

- Print a line for each iteration of the outer-most `for` -loop that shows you the progress.
- Test you code by running it only on the first 50 measurements of the data.

In [241…

```
def average_acc(dv, s):
    w_dv = np.exp(-np.abs(df['dv'] - dv)/1.5)
    w_s = np.exp(-np.abs(df['s'] - s)/30)
    w = w_dv*w_s
    w_ai = w*df['a']
    return w_ai.sum()/w.sum()
```

In [242…

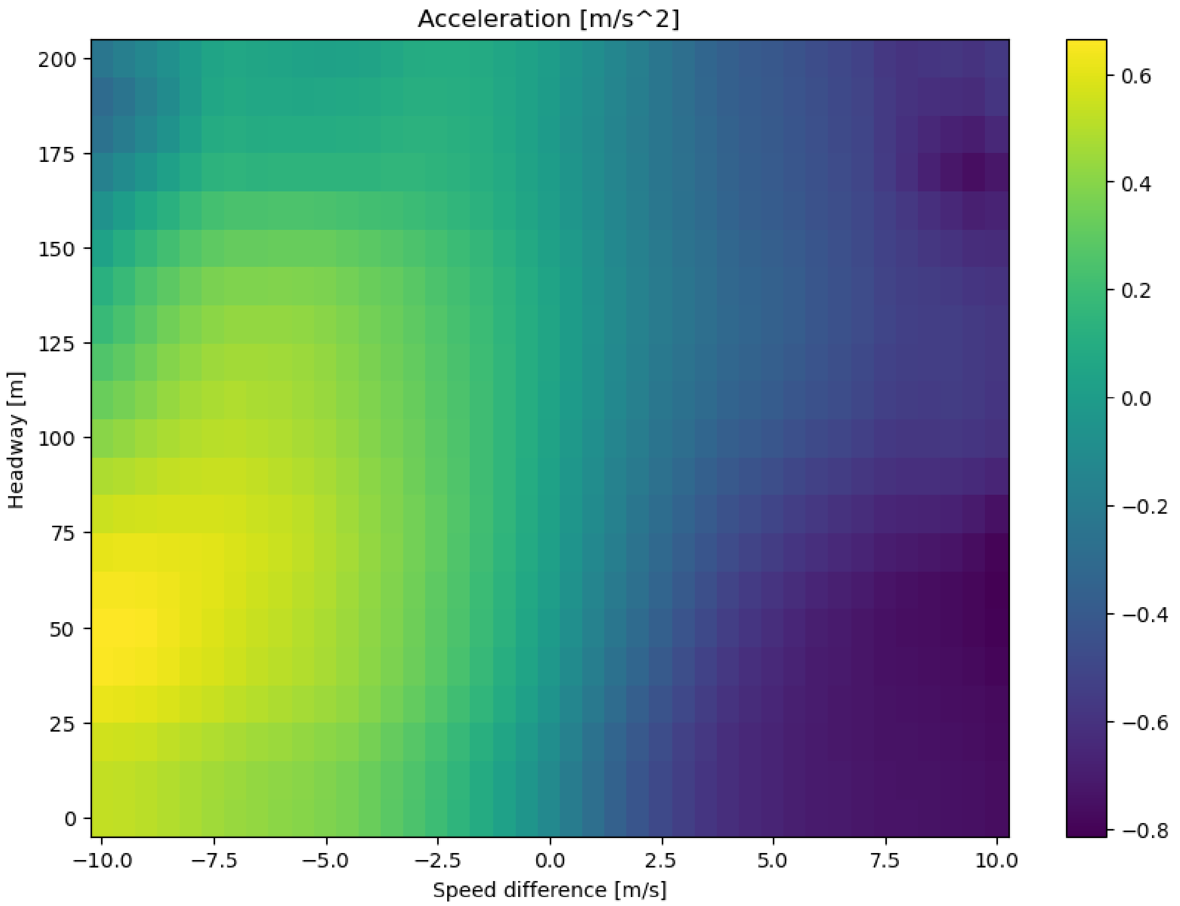
```
for i in range(41):
    for j in range(21):
        x = dv[i]
        y = s[j]
        a[j, i] = average_acc(x, y)
```

The following code will plot the data for you. Does it make sense when considering:

- Negative (slower than leader) and positive (faster than leader) speed differences?
- Small and large headways?

In [244…

```
X, Y = np.meshgrid(dv, s)
axs = plt.axes()
p = axs.pcolor(X, Y, a, shading='nearest')
axs.set_title('Acceleration [m/s^2]')
axs.set_xlabel('Speed difference [m/s]')
axs.set_ylabel('Headway [m]')
axs.figure.colorbar(p);
axs.figure.set_size_inches(10, 7)
```


Acceleration [m/s^2]
Headway [m]
200
175
150
125
100
75
50
25
0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
Speed difference [m/s]
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8

In [ ]:

In [ ]: