Import relevant packages here.

```
# Imports applicable libraries.
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
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.

```
# Loads the csv file and shows the first 10 rows.
data = pd.read_csv(r'C:\Users\oskar\OneDrive\Documents\TUD\TIL6022 TIL
Python Programming\Assignment 5\cf_data.csv')
data.head(10)
```

```
         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
```

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

```
# Creates the grid size.
dv = np.linspace(-10, 10, 41)
s = np.linspace(0, 200, 21)
a = np.zeros((21,41))
```

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.

```python
# Creates 3 seperate numpy arrays for the 3 columns in the csv file.
DV = data.dv.to_numpy()
S = data.s.to_numpy()
A = data.a.to_numpy()
```

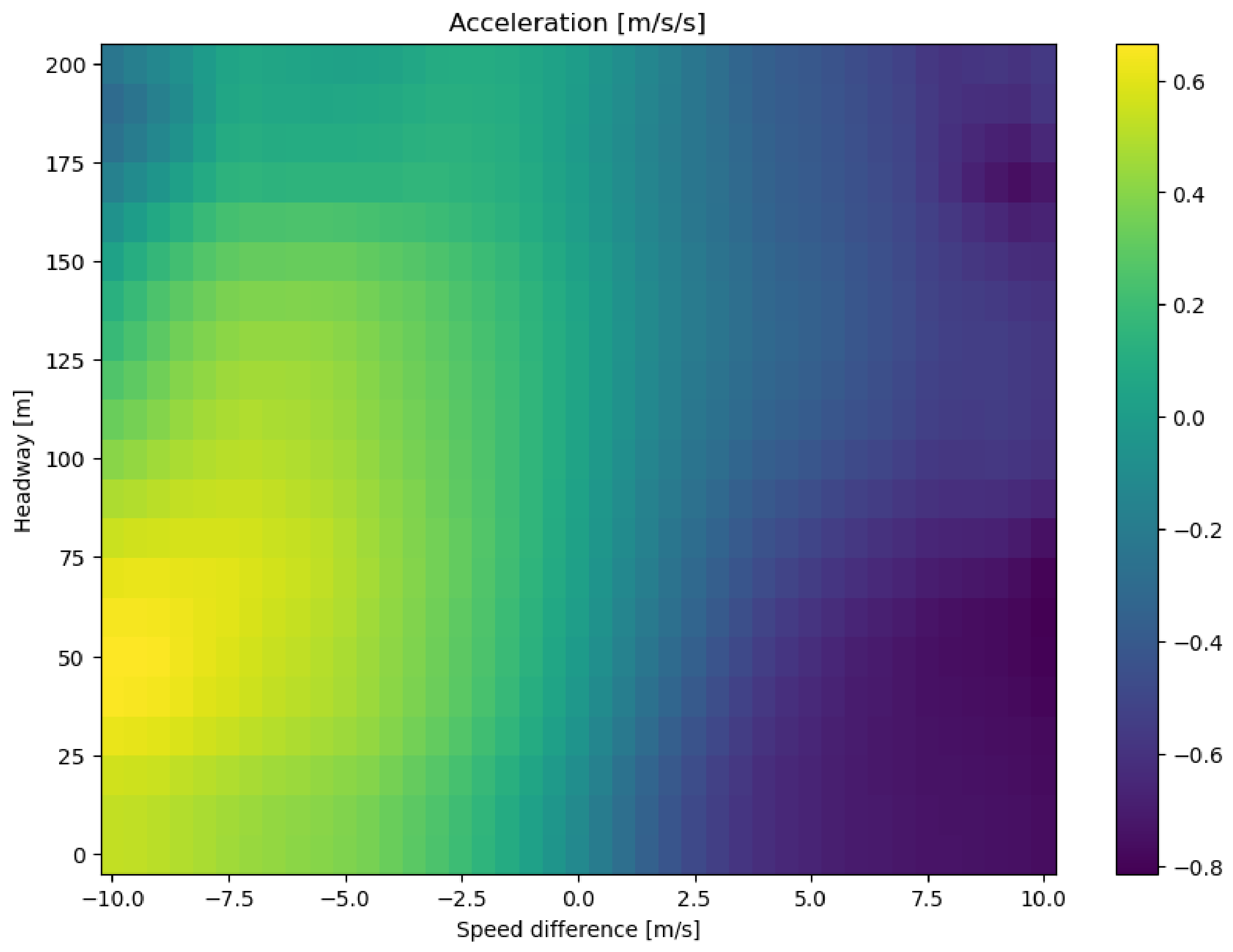
Create an algorithm that calculates all the acceleration values and stores them in the grid. The algorithm is described visually in the last part of the lecture. At each grid point, it calculates a weighted mean of all measurements. The weights are given by an exponential function, based on the 'distance' 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.

```python
upsilon = 1.5   #m/s
sigma = 30      #m

# Iterations over every row of data for every gridpoint.
for i in range(len(dv)):
    print(f'This is iteration {i+1}/41')

    for j in range(len(s)):
        # Resets the numerator and denominator for every new grid
point.
        mean_a_numerator = 0
        mean_a_denominator = 0

        for k in range(len(DV)):
            # Calculates the weight based on the proximity, and thus
the components to calculate the mean weighted acceleration.
            omega = math.exp(-abs(dv[i]-DV[k])/upsilon-abs(s[j]-
S[k])/sigma)
            mean_a_numerator += omega*A[k]
            mean_a_denominator += omega

        # Calculates the mean weighted acceleration.
        a[j,i] = mean_a_numerator/mean_a_denominator
```

```
This is iteration 1/41
This is iteration 2/41
This is iteration 3/41
This is iteration 4/41
This is iteration 5/41
This is iteration 6/41
This is iteration 7/41
This is iteration 8/41
This is iteration 9/41
This is iteration 10/41
This is iteration 11/41
This is iteration 12/41
This is iteration 13/41
This is iteration 14/41
This is iteration 15/41
This is iteration 16/41
This is iteration 17/41
This is iteration 18/41
This is iteration 19/41
This is iteration 20/41
This is iteration 21/41
This is iteration 22/41
This is iteration 23/41
This is iteration 24/41
This is iteration 25/41
This is iteration 26/41
This is iteration 27/41
This is iteration 28/41
This is iteration 29/41
This is iteration 30/41
This is iteration 31/41
This is iteration 32/41
This is iteration 33/41
This is iteration 34/41
This is iteration 35/41
This is iteration 36/41
This is iteration 37/41
This is iteration 38/41
This is iteration 39/41
This is iteration 40/41
This is iteration 41/41
```

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?

```
# Plots the data.
X, Y = np.meshgrid(dv, s)
axs = plt.axes()
p = axs.pcolor(X, Y, a, shading='nearest')
axs.set_title('Acceleration [m/s/s]')
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/s]
Headway [m]
Speed difference [m/s]
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
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8