Error Log/ Workings: Adk Affinity Chromatography Analysis

#Reading in the data#

```
data = "BslA 1 221019.csv"
import pandas as pd
data = pd.read_csv(data)
```

#But table output has incorrectly headed columns#

```
data = pd.read_csv(data, skiprows = 2, index_col = "ml")
```

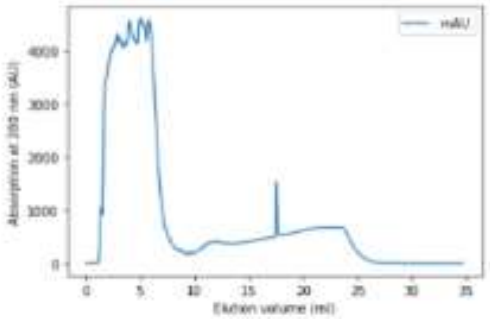
#Skips rows correctly and indexes by the “ml” column#

```
absorption_data = pd.DataFrame(data_to_use["mAU"])
```

#”KeyError: ‘mAU’”. Appears there is a space in front of mAU in excel file.

```
absorption_data = pd.DataFrame(data[" mAU"])
print(data)
absorption_plot = absorption_data.plot()
absorption_plot.set_ylabel("Absorption at 280 nm (AU)")
absorption_plot.set_xlabel("Elution volume (ml)"
```

ml	mAU	mL1	(Fractions)	mL2	(Injections)
0.00	-0.10	-0.16	F2	0.0	3.0
0.04	-0.28	-0.07	A1	NaN	NaN
0.07	-0.46	3.43	A2	NaN	NaN
0.11	-0.62	6.92	Waste	NaN	NaN
0.15	-0.77	7.06	Waste	NaN	NaN
...	...	...	...	...	...
34.57	-4.99	NaN	NaN	NaN	NaN
34.60	-5.00	NaN	NaN	NaN	NaN



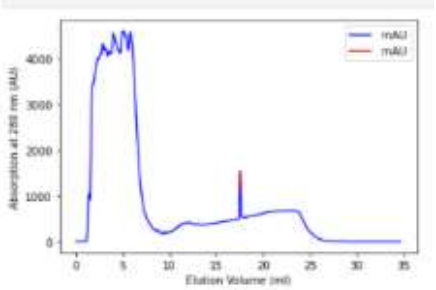
conda install plotly==4.13.0 #plotly has several useful tools for labelling graphs and detecting peaks#

#Peak detection requires this to be in a 1-D array: DataFrame in pandas not compatible with scipy find_peaks function. Instead, programmed a different way to find peaks manually#

```
elution_start_vol = 7.5
absorption_peaks_full = absorption_data[absorption_data[" mAU"] > 1000]
absorption_peaks_elute = absorption_peaks_full[absorption_peaks_full["ml"] > elution_start_vol]
```

#This will create two dataframes. The first is to identify peaks over the whole spectrum of absorption greater than 1000 AU. The second identifies these peaks that are in the collected eluted fractions, which begins after 7.5 ml in this case#

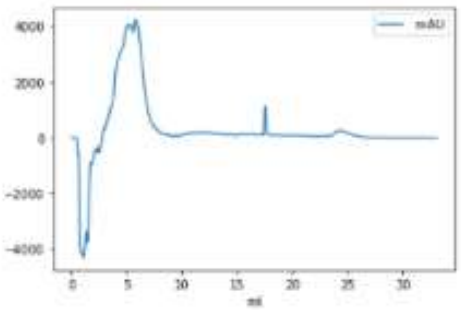
```
import matplotlib.pyplot as plt
fig,ax = plt.subplots()
absorption_data.plot("ml", " mAU", color="blue", ax=ax)
absorption_peaks_elute.plot("ml", " mAU", color="red", ax=ax)
ax.set_ylabel("Absorption at 280 nm (AU)")
ax.set_xlabel("Elution Volume (ml)")
plt.show()
```



#Creates an axis on which two graphs are plotted: the absorption graph (in blue) and a graph of peaks, in red. Hence, highlights peaks of the eluent > 1000 AU#

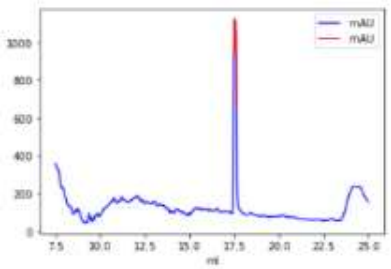
#However, there is a linear increase of absorption at 280 nm on our graph: this is as a result of imidazole absorbing waves of 280 nm: concentration is increasing to elute proteins. Must remove this by subtracting a control. This may later allow calculation of protein concentration in each fraction#

```
blank_data = "Adk 4 Blank Data Example.csv"

blank_data = pd.read_csv(blank_data, skiprows = 2)

print(blank_data)

blank_data.plot("ml", " mAU")

blank_absorption_data = blank_data[["ml", " mAU"]]
```



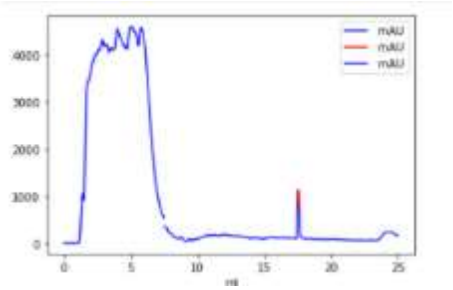
```
normalised_data = absorption_data

normalised_data[" mAU"] = normalised_data[" mAU"] - blank_absorption_data[" mAU"]

print(normalised_data)

normalised_data.plot("ml", " mAU")

normalised_elution_peaks = normalised_elution_region[normalised_elution_region[" mAU"]
>= 500]
```

#This normalises the data. However, the flowthrough region is not 0 as our control is not a true control: only data available, due to limited access to the lab, was a failed purification which does contain protein in the flowthrough. Therefore, I removed the flowthrough region from the normalised data. It was also decided that, due to normalisation, the absorption cut off to identify a peak as a true peak could be reduced from 1000 to 500 AU#



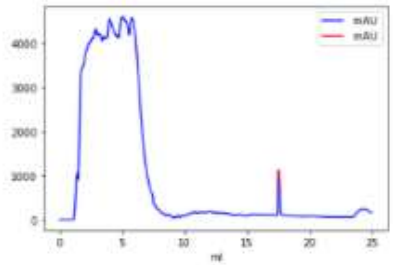
```
flowthrough = data[["ml", " mAU"]]

flowthrough = flowthrough[flowthrough["ml"] <= elution_start_vol]

fig,ax3 = plt.subplots()

normalised_elution_region.plot("ml", " mAU", color="blue", ax=ax3)

normalised_elution_peaks.plot("ml", " mAU", color="red", ax=ax3)

flowthrough.plot("ml", " mAU", color="blue", ax=ax3)

ax.set_ylabel("Absorption at 280 nm (AU)")

ax.set_xlabel("Elution Volume (ml)")

plt.show()
```

#The lines, however, do not meet, though should be. Dataframes of flowthrough and elution region are therefore appended to eachother#



```
flowthrough_normalised_eluent = pd.concat([flowthrough, normalised_elution_region])

fig,ax4 = plt.subplots()

flowthrough_normalised_eluent.plot("ml", " mAU", color="blue", ax=ax4)
```

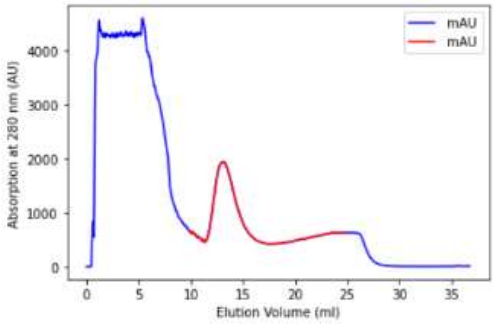
```
normalised_elution_peaks.plot("ml", " mAU", color="red", ax=ax4)
ax.set_ylabel("Absorption at 280 nm (AU)")
ax.set_xlabel("Elution Volume (ml)")
plt.show()
```



#Noticed there is a slight downward trend: this is due to the absorption
sample data and control data not matching in programs: one was produced on a 2.5ml sample loop and other on 5ml sample loop. Hence, this code will only be applicable to Adk samples for now (this, luckily, is what is mainly being worked on currently#

#Now found that the full data does not include the tail: which may still be important (after-eluent). These fractions are still collected so a calculation of protein concentration would be useful. This will require the normalised data. This also ensures that on the graph, only eluent peaks are highlighted.

```
after_eluent = normalised_data[["ml", " mAU"]]
after_eluent = after_eluent[after_eluent["ml"] >= elution_end_vol]
flowthrough_normalised_eluent = pd.concat([flowthrough, normalised_elution_region, after_eluent])
```



#Now needed to calculate the start and end points of each fraction. This showed that, for some sample data, the wrong volumes were being pulled from the csv. This was fixed by changing to the correct column being pulled: “ml.2”#

```
import numpy as np
from scipy.integrate import simps
from numpy import trapz
A1_start = collected_fractions.at["A1", "ml.2"]
#etc.
```

#Calculating A13 end volume wasn’t as easy as the others: the “fraction” below A13 is labelled as “Waste”, but there are at least 3 others labelled as waste. To get the correct “Waste”, used the below:

```
fractions.reset_index(inplace = True)
A13_end_index = fractions.iloc[fractions[fractions["(Fractions)"] == "A13"].index + 1]
A13_end_index.reset_index(inplace = True)
A13_end_index = pd.DataFrame(A13_end_index)
A13_end = A13_end_index.at[0, "ml.2"]
fractions.set_index("(Fractions)", inplace=True)
```

#Now created subsets of data that included only the data in each fraction. This originally used the eluent region only: but it was of interest to calculate protein concentration in all fractions so used the whole normalised absorption spectrum (flowthrough + normalised eluent + normalised “after-eluent”)

#Numpy arrays created for each fraction and area calculated by two different methods: trapezoidal and Simpsons’ methods. Appended to a table and the mean value calculated and appended to the same table, with fraction sizes

#Protein concentration is calculated from this. Originally used the protein extinction coefficient in each calculation, but later created an element and placed at top of code as a potential input data: could be changed. Appended to same dataframe as above, and eluent protein concentrations plotted on a graph

```
A1_mean_absorption_per_ml = (fraction_areas.at['A1', 'Mean Area']/1000) /
(fraction_areas.at['A1', 'Fraction Size'])

A1_protein_concentration_mM = (A1_mean_absorption_per_ml / (
Protein_Extinction_Coefficient * 0.002)) *1000

#etc
```

#Moved all input data elution start/end fractions. eluent fractions, blank data, sample data and protein extinction coefficient to top of code for ease of access to user. Defined the code as “Adk_aff_chrom_analysis with arguments of data input and peak_au_sensitivity value (i.e. the minimum AU value that will identify as peaks). Defining the end volume of eluent required pulling of the start volume of the fraction above, as the “end fraction” is used as an input. The below code pulled the end volume of eluent:

```
collected_fractions.reset_index(inplace = True)

elution_end_fraction_index =
collected_fractions.iloc[collected_fractions[collected_fractions["(Fractions)"] ==
elution_end_fraction].index + 1]

elution_end_fraction_index.reset_index(inplace = True)

elution_end_fraction_index = pd.DataFrame(elution_end_fraction_index)

elution_end_vol = elution_end_fraction_index.at[0, "ml.2"]
```

#Future aims for this code is to use for loops, instead of repeating instructions. I would also like to get a true control dataset that can be applied and hence applicable to all data samples, not just Adk. I would also like to integrate more of the input data into the arguments of the function.