report.md

Assignment 3: Clustering

Tobias Lindroth: 20 hrs

Robert Zetterlund: 18 hrs

Question 1 - Show the distributions

These tasks are straightforward so we present the code rather than explain it.

Scatterplot

```
df.plot.scatter(x=PHI, y=PSI)
```

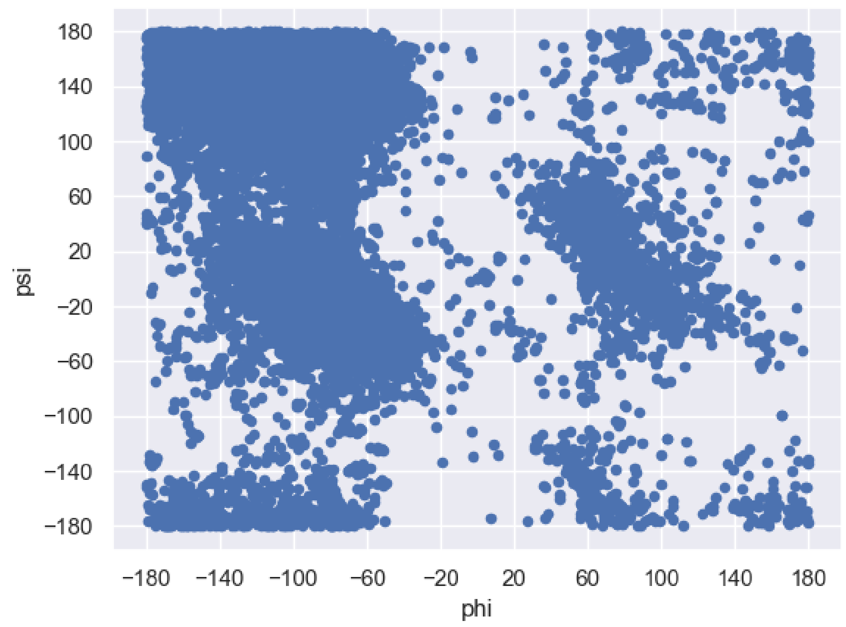

Figure 1: Scatterplot of psi and phi values

Heatmap

Code for creating square heatmap, show left below.

```
# Square heatmap

# In this program we create a 18x18 matrix of integers, and calculate index by floor division

# create matrix
matrix = np.zeros((nr_boxes, nr_boxes), dtype=int)

# Calculate index by using
df = df.apply(lambda row: row // BOX_SIZE + INDEX_OFFSET )

# increment based on value
# increment values in matrix based on index
```

```
for row in df.itertuples(index=False):
    matrix[row[1]][row[0]] += 1

# create heatmap
ax = sns.heatmap(matrix, xticklabels=X_TICKS, yticklabels=Y_TICKS)
```

Code for creating smooth heatmap, shown right below:

```
# smooth heatmap
sns.kdeplot(x=df[PHI], y=df[PSI], fill=True, cmap="rocket", cbar=True, thresh=0, levels=50)
```

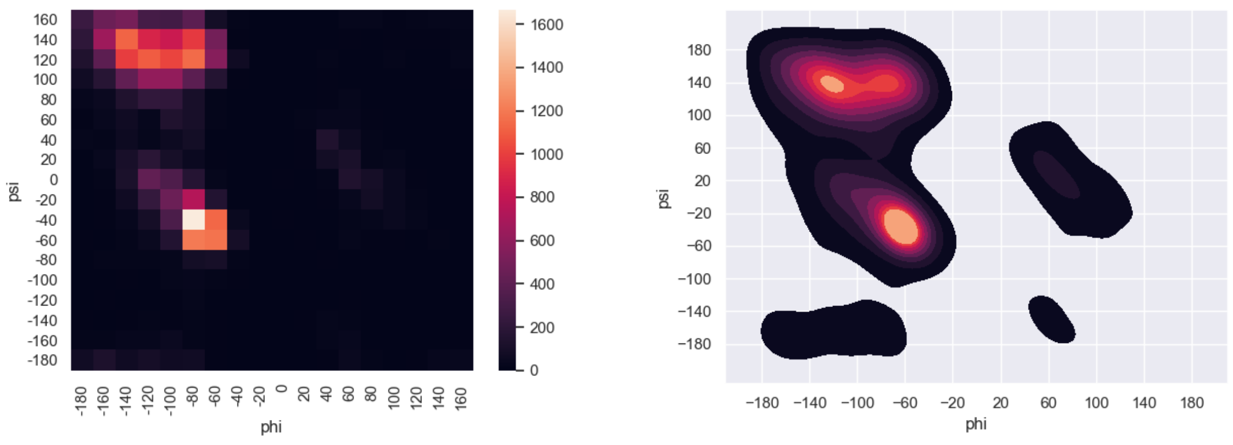

Figure 2: Heatmap for dataset using two different representations

Question 2 -

By trying out different k-values we notice that k=3 or k=4 have the most reasonable clusters.

With a k < 3, the clusters do not explain the variation. For example, in `k = 2` both clusters range over all possible phi values and include the interval `phi E (-30,20)` which has very few data points within it. Signaling that perhaps this divide should not be included in a cluster. For example, the blue centroid is seemingly the closest centroid for 4 different clusters.

With a k > 4 it seems like we are trying to create clusters where there should not be clusters, i.e we are overfitting. For example, in `k = 5` the red and green clusters in the top left corner create an awkward split that looks artificial and wrong. We also see by comparing it to other plots that it has split the red centroid by creating two centroids instead in the top left corner.

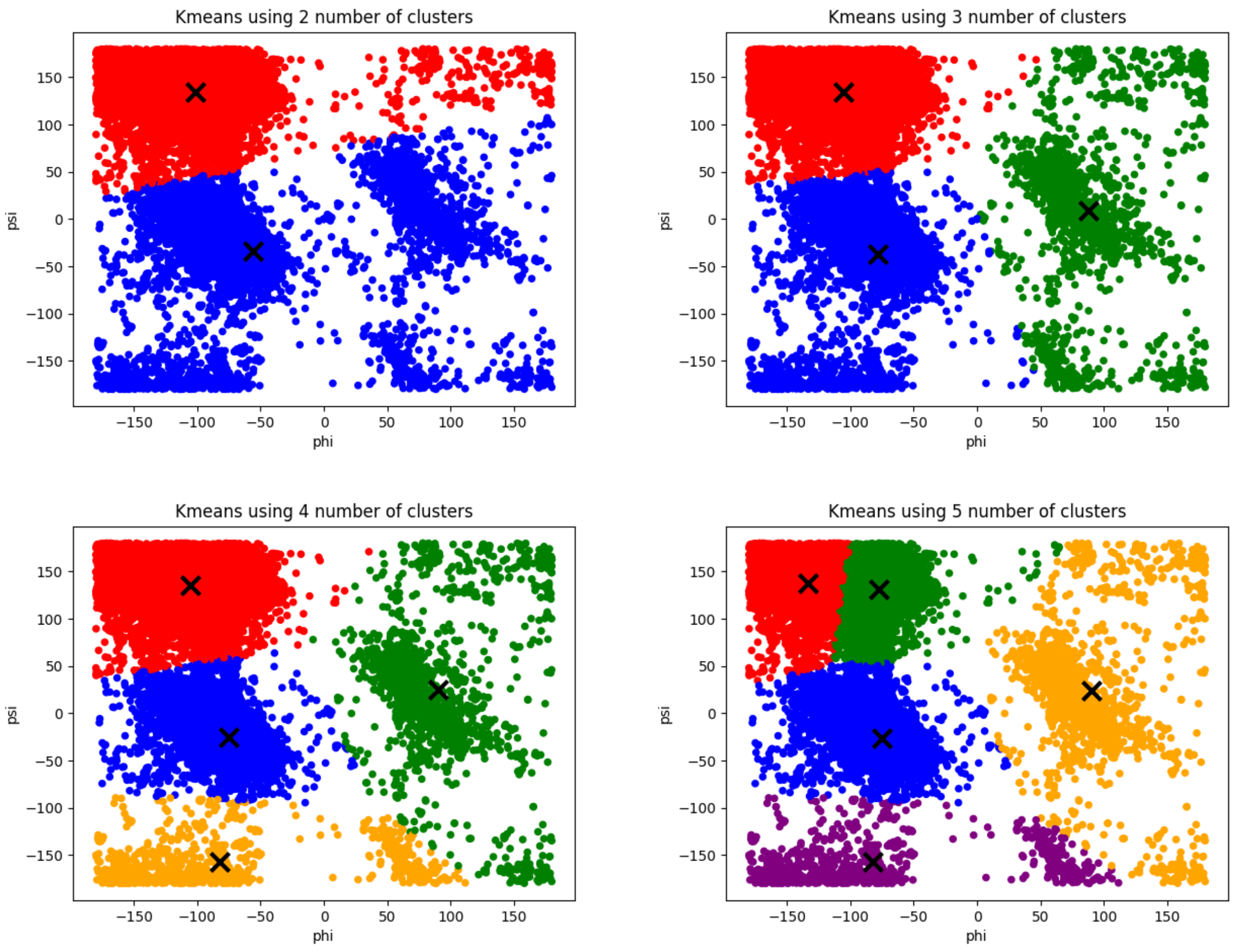

Figure 3: The clusters created using k-means for different k. The X:s indicates centroids.

To decide whether k=3 or k=4 should be used we use an elbow curve as it can give an indication to what k-value fits the data best, that is, how many clusters we should use. In the figure below we can clearly see an elbow at k=3, hence the elbow curves indicate that 3 clusters is the best fit.

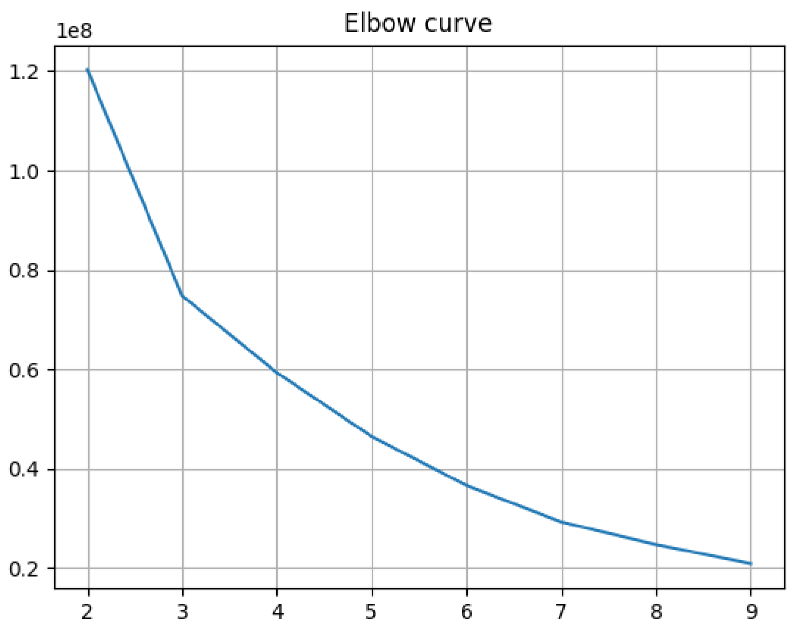

Figure 4: The elbow curve indicating we should use 3 clusters.

The calculation of the elbow curve was done by:

For each k (k=2 to k=10), taking the sum of the squared distances of samples to the nearest cluster centre. See the code snippet below.

```
distorsions = []
for k in range(2, 10):
    kmeans = KMeans(n_clusters=k)
    kmeans.fit(X)
    distorsions.append(kmeans.inertia_)

#inertia is the sum of squared distances of samples to    their closest cluster center.
```

Since both our own experiment with different k-values and the elbow curved indicate k=3 to be a good value, we deem 3 to be the most suitable k-value for this task.

Validation

We validate the clusters by checking if the clusters still are stable even if we remove a proportion of the points. We remove a random 25% of the points and examine if the labeling remains similar. We choose 25% as we believe that is enough to have an effect on the clusters, but not change the dataset dramatically.

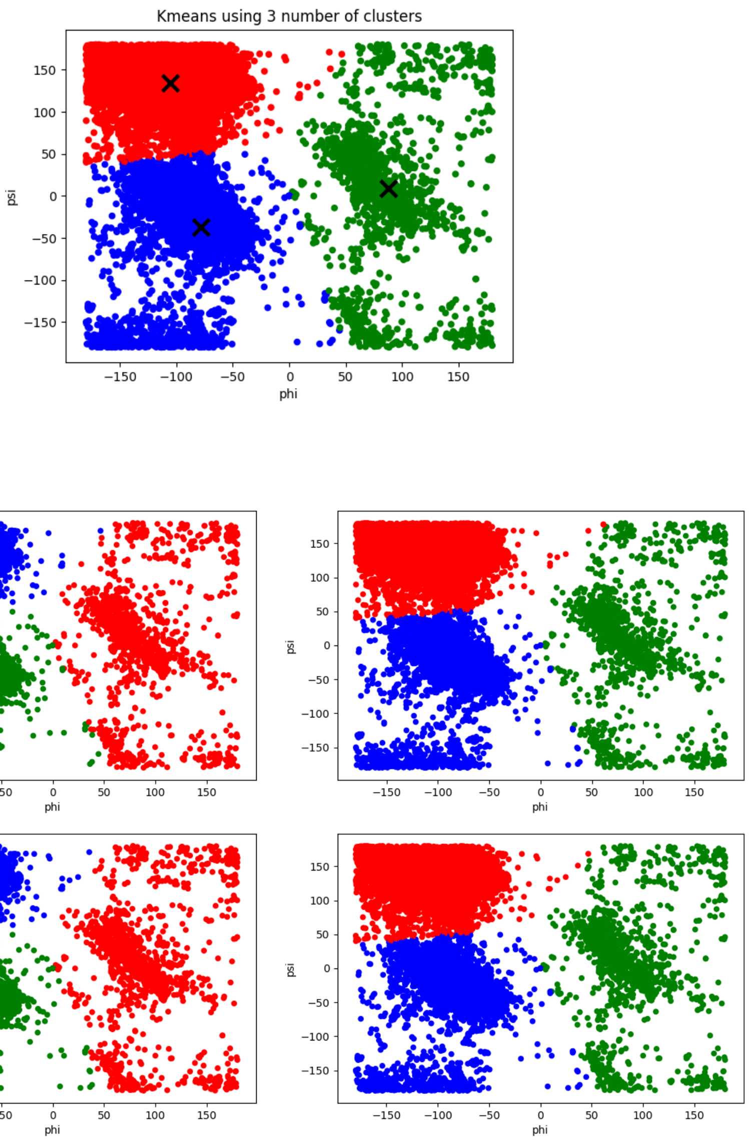

Figure 5: The original clusters compared to the clusters created when removing a random 25% of the points.

In the figures above we see the original clusters compared to 4 figures with clusters created after removing a random 25% of the points. We notice that the shapes of the clusters almost do not change at all. There are minor changes but it is still the same clusters. This indicates that the clusters are stable.

We also notice that the colors of the clusters change in different runs. This does however not mean that the clusters are unstable since the initial centroids are different each time. Suppose that points nearest to “centre 1” are shown in blue, those nearest “centre 2” are shown in green and those nearest “centre 3” is shown in red. Which of the clusters we have found happens to be blue might change from one run to the next since the centres start off in different places and end up in different places, “centre 1” from one run might end up near to where “centre 2” ended up in another run.

So, we always find roughly the same clusters when performing k-means with k=3 on different subsets of the dataset and hence we deem these clusters to be stable.

Do the clusters found seem reasonable?

By looking at the figure below we notice that there are some aspects of the clustering that does not seem very reasonable. For example, the bottom left points, see label 1 in figure below, should probably not be in the blue set, but rather in the red. This is because psi=-180 is the same as psi=180 and hence the points in the bottom left should actually be in the red cluster as they are more connected to that cluster.

Furthermore, it seems unreasonable for the green cluster to be so scattered, see label 2 in the figure below. It would probably be more reasonable if it was divided into two clusters. The points at the top and bottom in one cluster, and the points in the middle as another.

One could also argue that the points in the absolute bottom right corner, see label 3 in the figure below (and possible some in the top right corner) should belong to the red set, as phi=-180 is the same as phi=180 (and again psi=-180 is the same as psi=180).

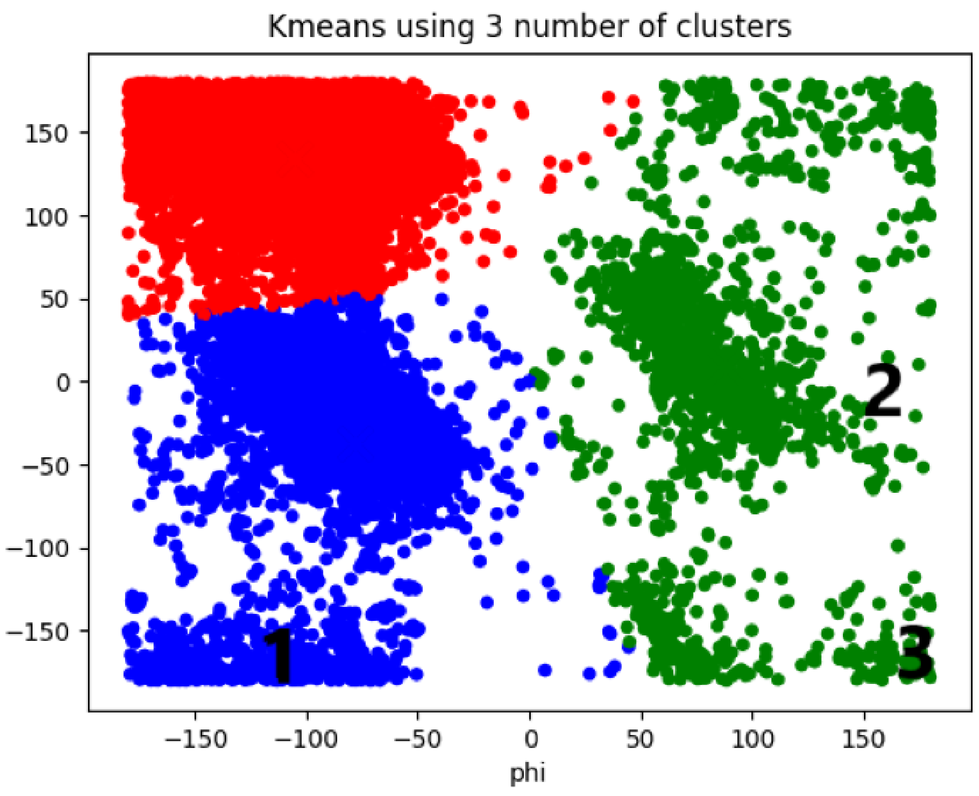

Figure 6: The clusters created using k-means with k=3.

Can you change the data to get better results?

By looking at the plots above, we notice two "divides", one at approximately `phi=0` and `psi=-110`. Below is a plot with added lines.

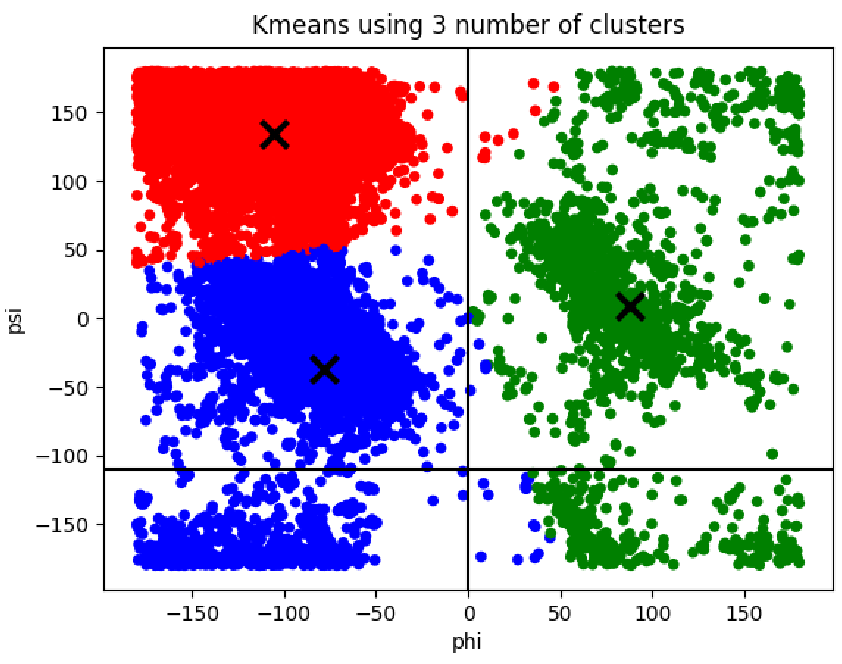

Figure 7: The clusters created using k-means with k=3. The X:s indicates centroids.

We reason that due to the periodic attributes we can show negative values as positive values by adding `360`, essentially shifting them. This would in practice revolve in the following result

		value	shift?	new value
phi		> 0	no	phi
phi		< 0	yes	phi+360
psi		> -110	no	psi
psi		< -110	yes	psi+360

```
# shift phi by 180, new range is 0 >-> 360
# shift psi by 70, new range is −110 >-> 250
df[PHI] = df[PHI].apply(lambda phi: phi + 360 if phi < 0 else phi)
df[PSI] = df[PSI].apply(lambda psi: psi + 360 if psi < −110 else psi)
```

What happens to the elbow curve, should we select new value for k? Lets see by comparing the two elbows curve, the elbow curve to the right is the one with shifted values.

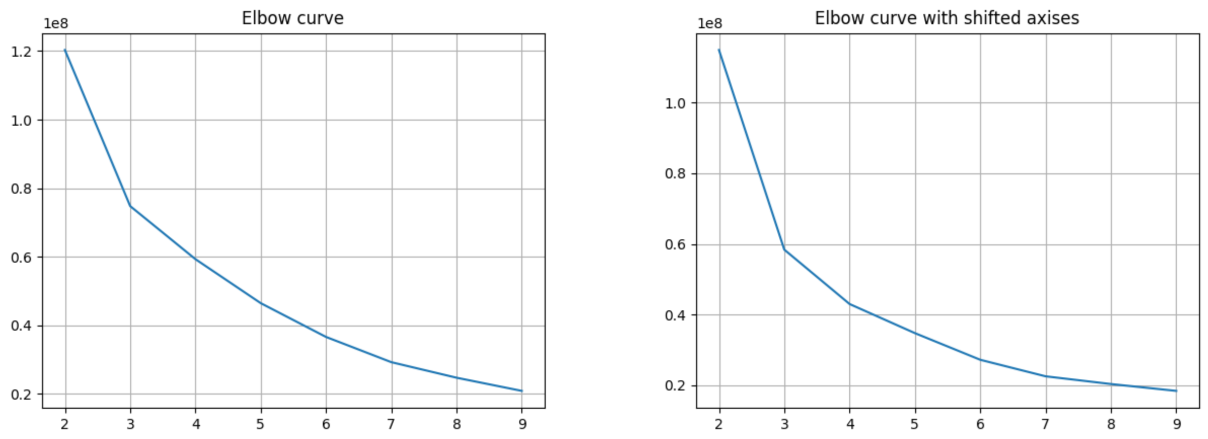

Figure 8: A comparison between the elbow curve created before shifting the data (left), and the elbow curve created after (right).

We see that we improve the cumulative sum of the neighbors (the y axis) but that the optimal value for k remains, `k = 3`.

We plot the datapoints again using kmeans with shifted axises and get a visually more reasonable clusters that does not bridge any divides.

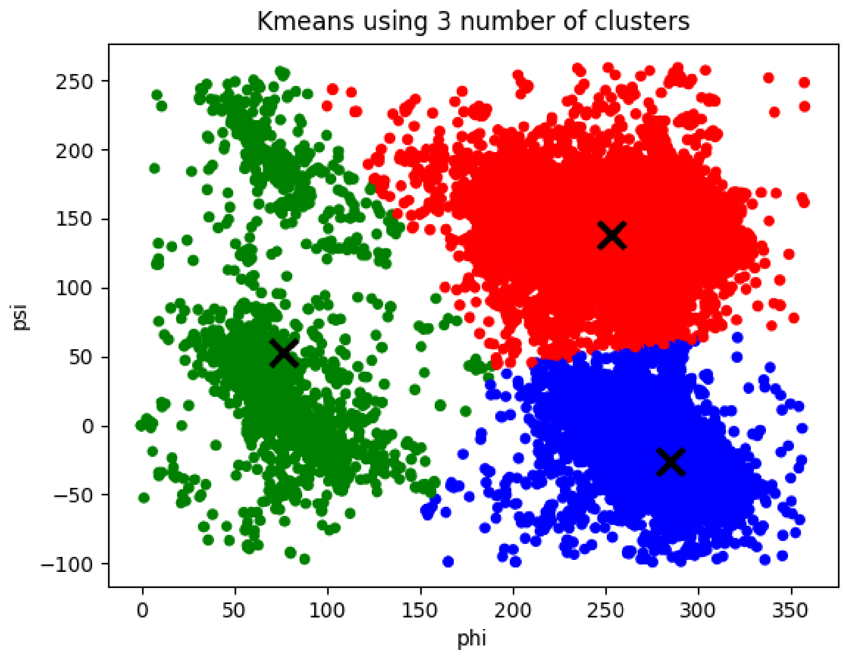

Figure 9: The clusters created using k-means with k=3 after shifting the data. The X:s indicates centroids.

Question 3

a - Motivate

i - the choice of the minimum number of samples in the neighbourhood for a point to be considered as a core point

[To preface, we are not entirely sure how to motivate this number, we have googled quite a bit but with no luck]

Things we had in mind when we chose the number:

- Do not form clusters of noise, hence we do not want to choose a to low minPts. Assisted by looking at our heatmaps.
- The value should be picked by someone with domain knowledge, i.e. not us.
- Given an arbitrarily chosen epsilon, we wanted to ensure that clusters we could visually motivate would be classified as such, and also that potential "bridges" across subsets of clusters.

Again, we did not really have any foundation when it came to this, we decided to test different values and ended up choosing 42. We decided that we would pick a suitable epsilon based on the value of minPts. It was noted online that this was an acceptable approach and we explore choosing this value more in depth in part d.

ii - the choice of the maximum distance between two samples belonging to the same neighbourhood (“eps” or “epsilon”).

The following lines of code uses the algorithm explained in this paper, via this medium article, but **we changed** so that it takes the largest distance within in all k neighbours. If it takes the nearest it will get identical results for all `k>0` .

We found that this methodology worked well, and intuitively we motivate the adjustment in by the following statements:

- We want to have core points within clusters, given a sufficiently dense cluster and value of `k` it does not really matter if the value of epsilon increases.
- When the largest distance to the "k":th neighbor changest the most, it is possible that we encounter noise. So by picking the value of largest distance, we have made a distinction between noise and clusters that is reasonable.
- In contrary to the article (from medium) in which we found the algorithm, it performs differently based on the value of kmeans.

Here is how it is implemented:

```
# init nearest neighbors with our dataset
neigh = NearestNeighbors(n_neighbors=n_neighbors)
nbrs = neigh.fit(X)

# get distances
(distances, _) = nbrs.kneighbors(X)

# Sort each node's distances to its closest n_neighbors neighbors
distances = np.sort(distances, axis=0)

# For each node, pick out the distance to the neighbor (out of closest n_neighbors) that is furthest away.
distances = distances[:, -1]

# find index of largest difference (make a distinction of 28750,
# since the plot looks exponental and we're only interested in "elbow" area.)
index = np.diff(distances[0:28750]).argmax()
```

Using `n_neighbors=42` , we get an epsilon of `19` , which we will use for the remainder of the analysis.

It is possible to argue our tweak to the algorithm, as this arbitrary distinction that noise begin at the largest jump in distance can have it's flaws. We also note that for large values of `n_neighbors` our approach performs poorly. But this proved effective for us and it seems that approaches are inherently data-dependent. We learn that it is important to have the amount of core points in mind and (although again, it is data dependent on the densisty) make sure that some non-core points are classified as well.

b - Highlight clusters found using DBSCAN, Barplot and outliers.

Scatterplot DBSCAN

Our code for creating a DBSCAN make use of masks.

```
# create array same size as dataset, init as all false
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
# Take array and make boolean true for indices of core nodes
core_samples_mask[db.core_sample_indices_] = True
# ...
for u_label, color in zip(unique_labels, colors):
    # u_label is -1 if not part of cluster, ie. noise
    isNoise = u_label == -1

    # make unassigned datapoints grey
    if isNoise:
        color = "grey"
        noiseMask = (labels == u_label)

    # create mask for labels, used for selecting which datapoints to plot
    class_member_mask = (labels == u_label)
    # create mask for non core datapoints within current loop's members
    non_core = X[class_member_mask & ~core_samples_mask]
    # create mask for core datapoints within current loop's members
    core = X[class_member_mask & core_samples_mask]

    # do regular scatterplots ...
```

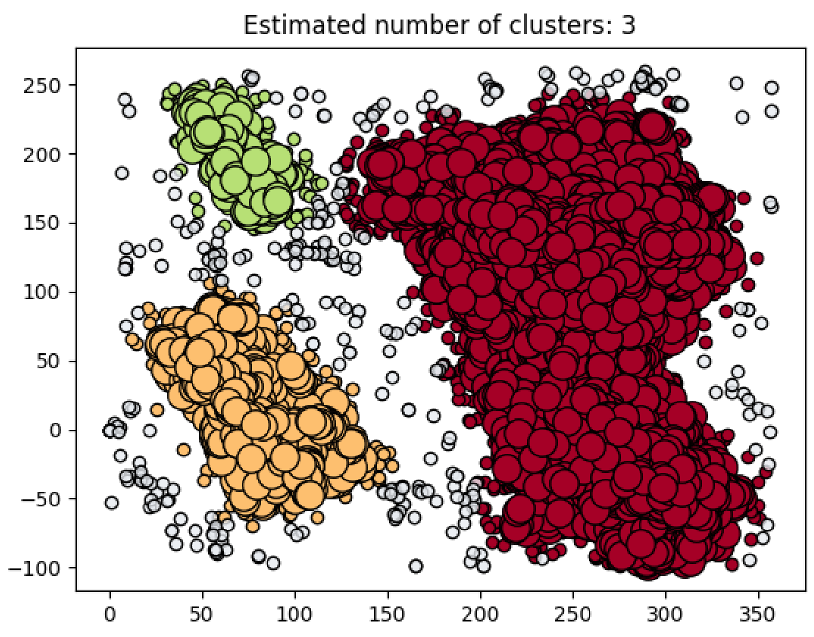

Figure 10: DBSCAN with epsilon=19, minPts=42

Barplot of noise

Using the same code as above, we use the `noisemask` and apply it to the dataframe. Then we count the occurence of each residue name and plot accordingly.

```
# apply noisemask to dataframe
df_noise = df[noiseMask]
# count number of each name occuring in df, create column counts to store value
df_noise = df_noise
                .groupby(['residue name'])
                .size()
                .reset_index(name='counts')

#create barplot
sns.barplot(x=df_noise["residue name"], y=df_noise["counts"])
```

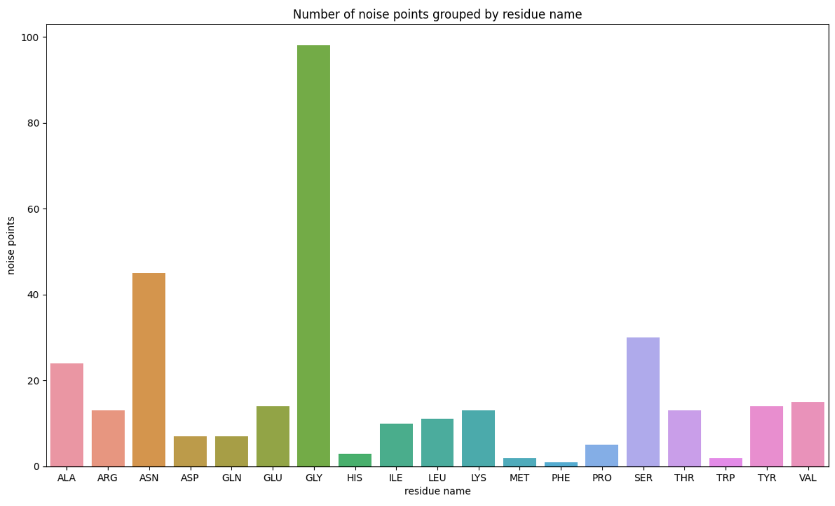

Figure 11: Barplot of noise of above DBSCAN

In total we have `327` noise points (outliers).

c - Compare DBSCAN and K-means

When looking at the clusters found by K-means and those found using DBSCAN we find some similarities, but mostly differences.

The clusters are similar in the way that both methods found 3 clusters (This is however largely affected by our choice of parameters).

But even though they have the same amount of clusters, the clusters are very different. K-means split the points to the right into two clusters and kept the ones at the left as a single cluster while DBSCAN did the opposite. The clusters found by DBSCAN seem more reasonable than those created by K-means. This is because when looking at the clusters, it is more clear that the points to the left should be divided into two clusters than that those to the right should be.

Furthermore, we can see that the clusters created by DBSCAN are more compact clusters as it leaves outliers out of clusters. K-means on the other hand puts every node into clusters even though they sometimes clearly should not be in a cluster.

An interesting note is that if we had done the clustering by hand, we would probably have chosen a mixture between K-means and DBSCAN. That is, we would have divided both the points to the rigth and the ones to left into two clusters, in total four clusters.

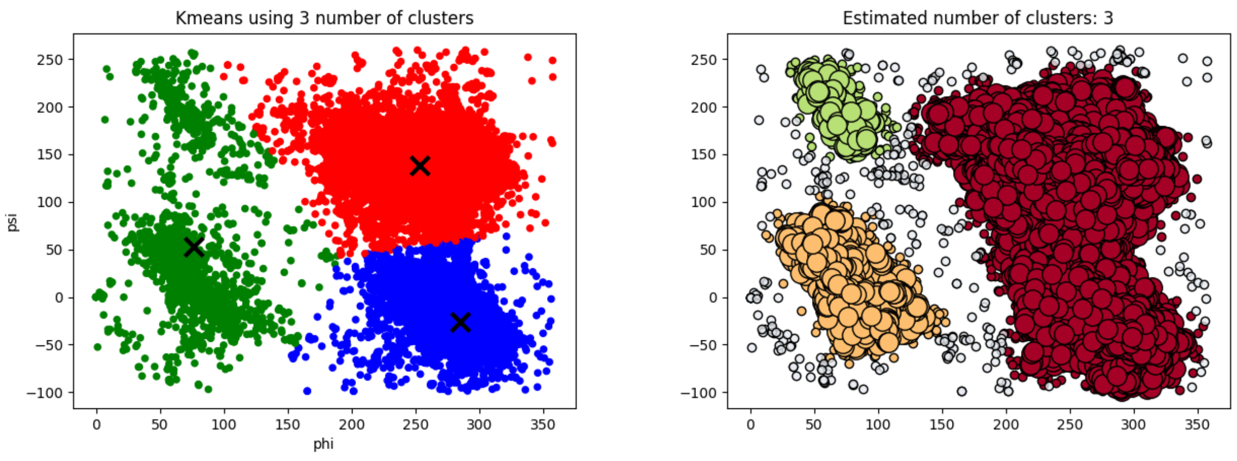

Figure 12: The clusters created by k-means to the left and the clusters created by dbscan to the right.

d - Robust to small changes?

By looking at what happens when epsilon or minPts is changed by a small value, one notices that dbscan can be very sensitive to these parameters.

When changing one parameter at a time, we see that the clusters we have found are very sensitive to an increase in the maximum distance between two samples belonging to the same neighbourhood a little. The top left cluster is engulfed into the large cluster. This is because epsilon now is large enough for a "bridge" to be created by the top left cluster and the large cluster.

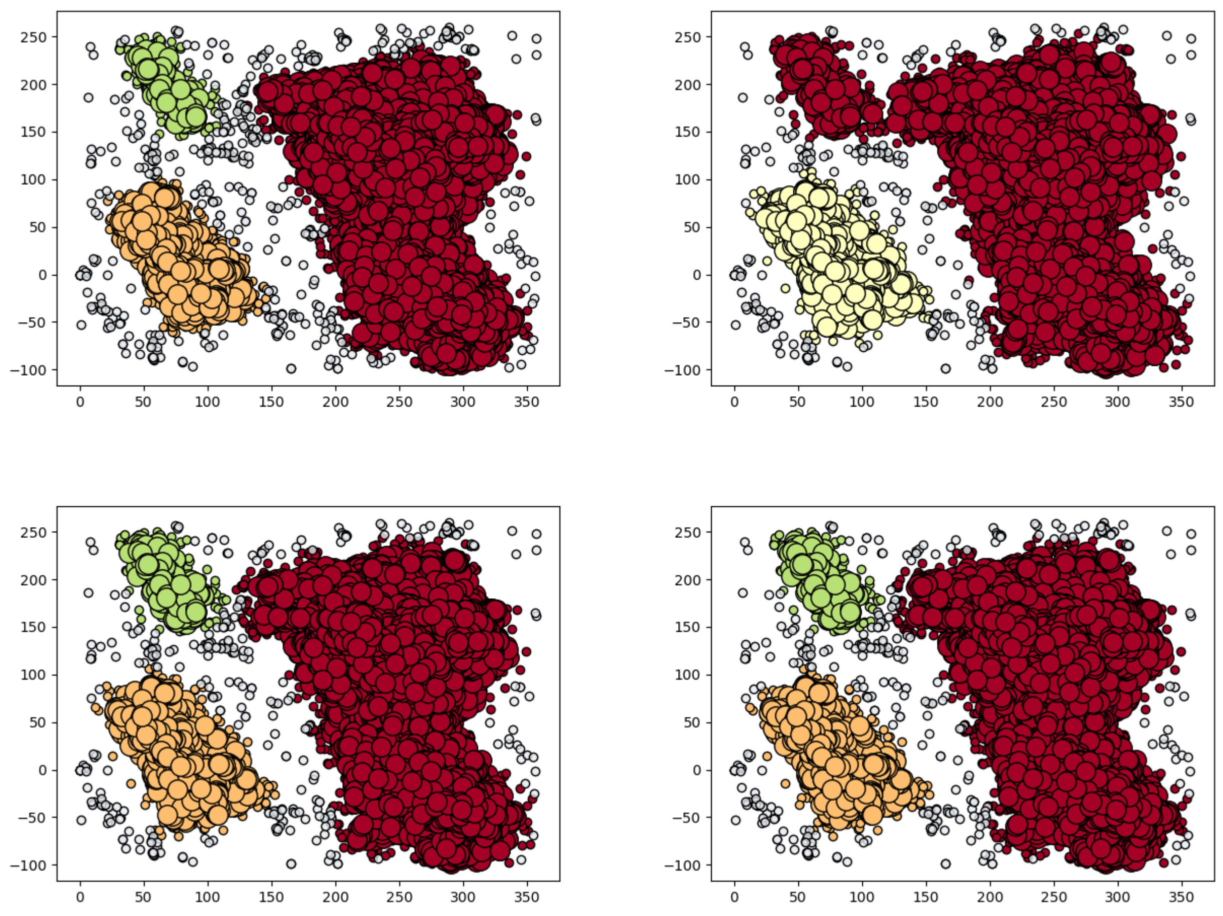

Figure 13: Top left (eps=17, minPts=42). Top Right (eps=21, minPts=42). Bottom left (eps=19, minPts=42). Bottom right (eps=19, minPts=44)

When changing both parameters at the same time, we see the same thing. The top left cluster is engulfed into the large cluster when epsilon is increased.

The clusters are of course also affected by changes in minPts, but in our case it was an increase in epsilon that created a large difference. In another case it might be a change in minPts that changes the clusters fundamentally.

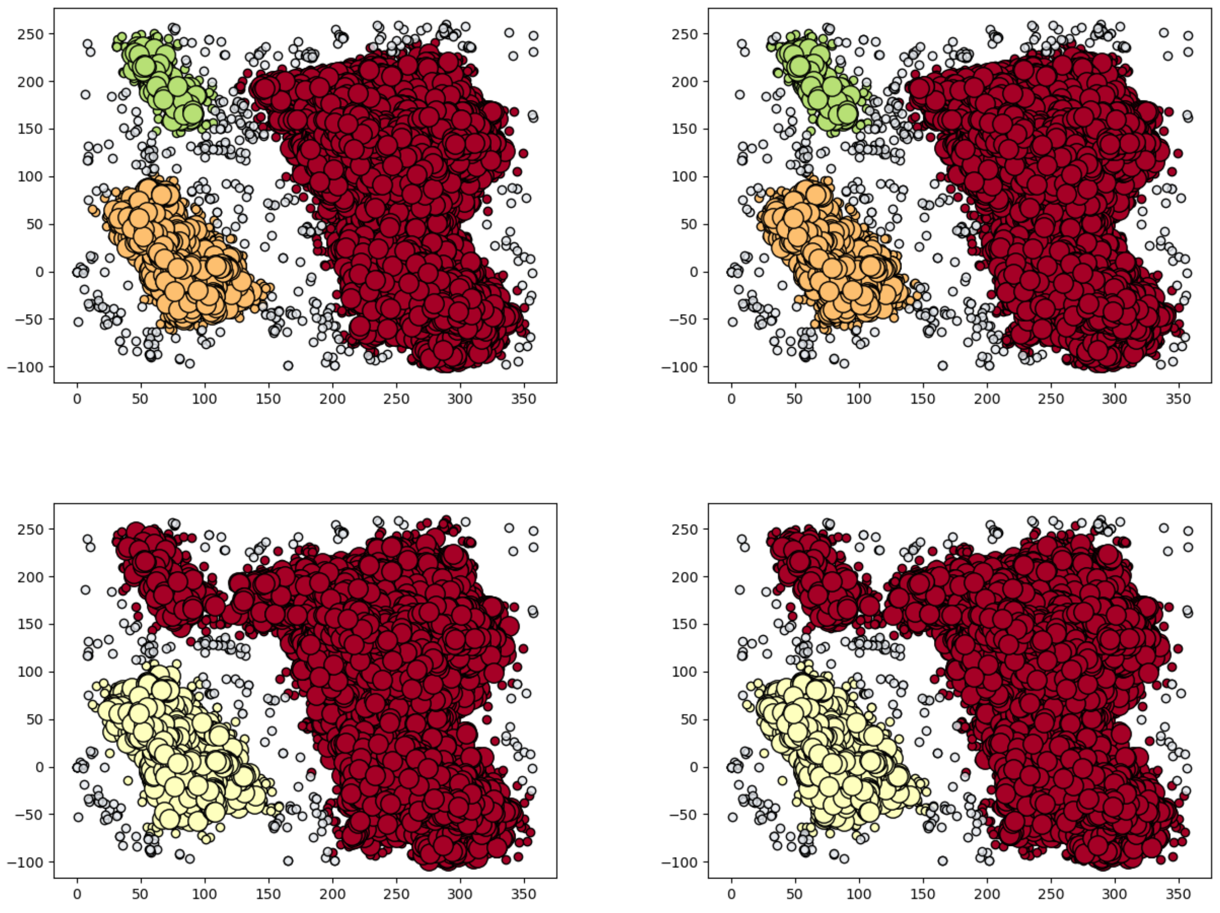

Figure 14: Top left (eps=17, minPts=40). Top Right (eps=17, minPts=44). Bottom left (eps=21, minPts=40). Bottom right (eps=21, minPts=44)

The fact that DBSCAN is so sensitive to the minimum number of samples in the neighbourhood for a point to be considered as a core point, and/or the choice of the maximum distance between two samples belonging to the same neighbourhood, shows how important it is to choose these parameters carefully. A small increase or decrease can change the clusters fundamentally.

Question 4

PRO

First we look at a scatterplot highlighting the amino acids with the PRO residue name.

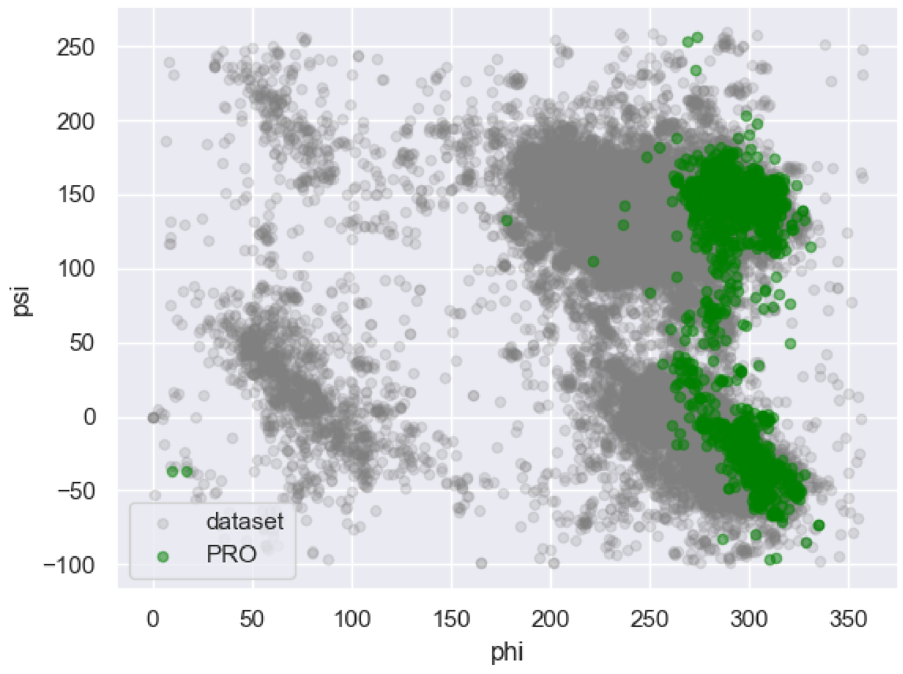

Figure 15: A scatterplot highlighting amino acids with residue name PRO.

We notice that amino acids with residue name PRO are can be classified into one or two clusters, since they are primarily in the phi range of 250 to 350, also they range somewhat consistently across psi -100 to 200. They are not evenly distributed through the entire dataset which lead us to believe that amino acids with residue name PRO are similar in characteristic and can be labeled with *somewhat* precision.

Now for a more analytical analysis, we find suitable k using elbow curve and find that PRO has a `k = 3`.

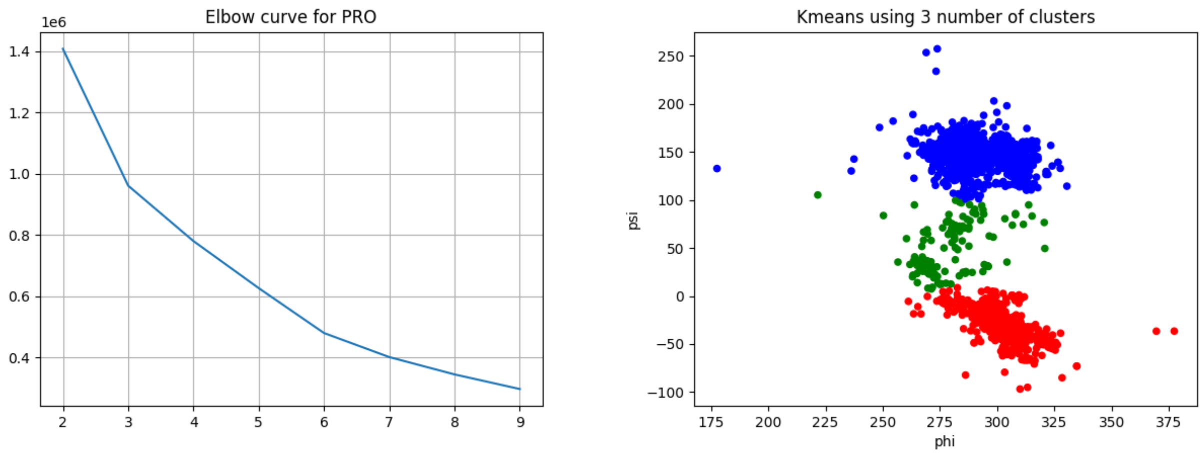

Figure 16: The elbow curve indicating 3 to be the most suitable k and the clusters created by k-means using k=3.

Comparatively little noise is noted when looking at the PRO dataset, verified by our DBSCAN below. We find that by using DBSCAN with reasonable chosen values of `epsilon` and `min_samples` provides one or two clusters.

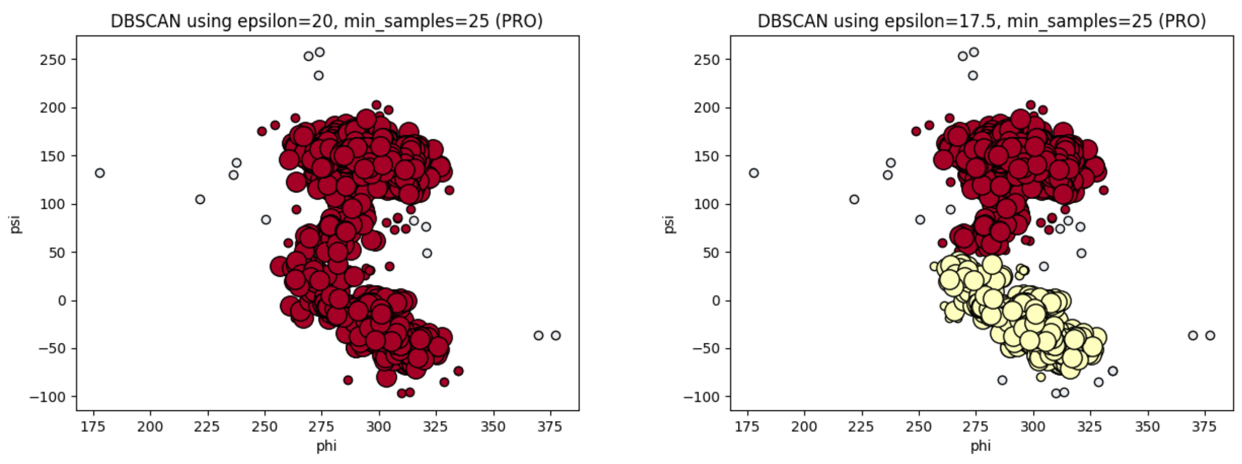

Figure 17: The clusters created when using different values of eps and minPts for DBSCAN.

We notice that the only similarity between the clusters found for PRO and the general clusters is that k-means found 3 clusters in both cases. Otherwise, the clusters found look completely different, both when it comes to shape and noise and both for k-means and DBSCAN. This seems however reasonable as the distribution of the points is so different.

GLY

First we look at a scatterplot highlighting the amino acids with the GLY residue name.

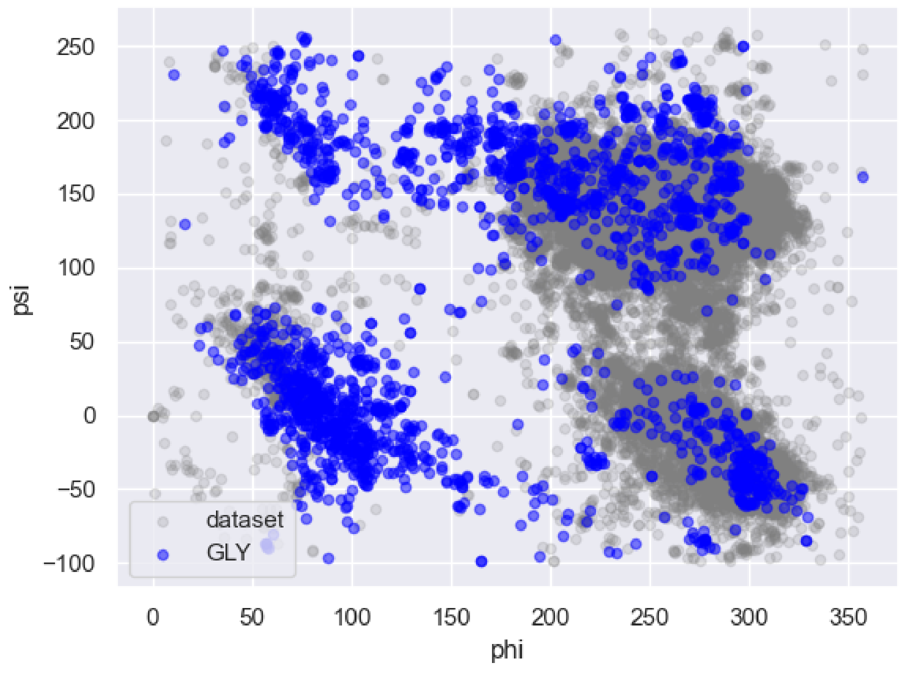

Figure 18: A scatterplot highlighting amino acids with residue name GLY.

The scatterplot shows how the distribution of GLY-points is quite similar to the distribution of all points. This indicates that the clusters of GLY might be quite similar to the general clusters. The GLY-points are quite scattered,with a lot of noise, which indicates that amino acids with residue name GLY are not similar in characteristic and therefore can be difficult to label with precision.

In order to compare the clusters, we find suitable k using elbow curve and find that GLY has a `k = 4`.

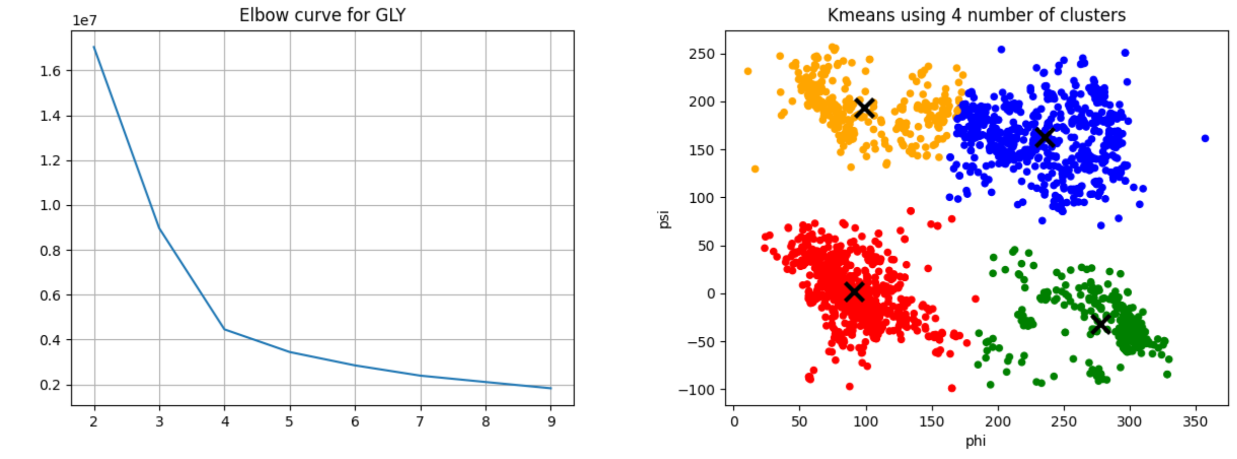

Figure 19: The elbow curve indicating 4 to be the most suitable k and the clusters created by k-means using k=4.

We find that by using DBSCAN with reasonable chosen values of `epsilon` and `min_samples` provides three or four clusters.

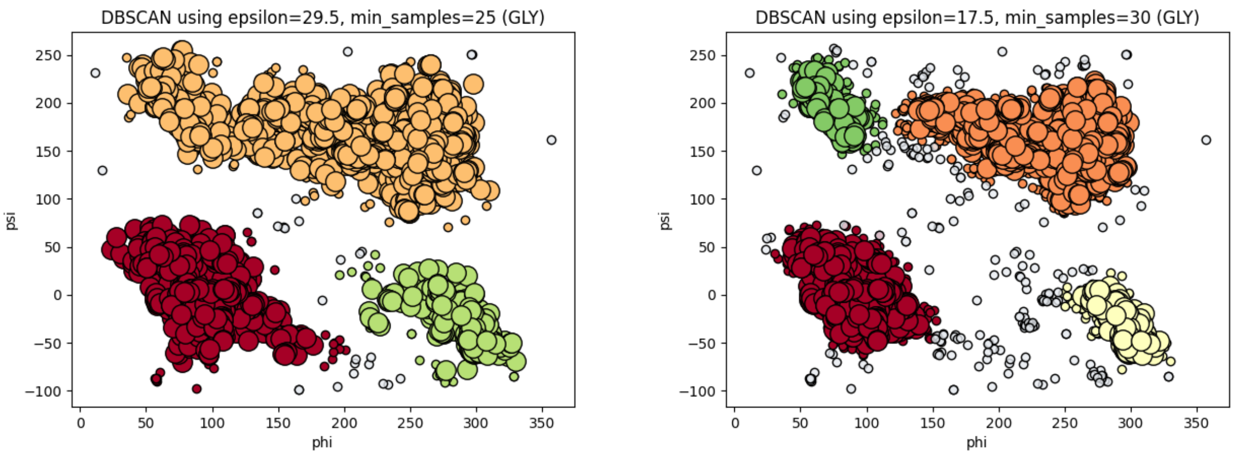

Figure 20: The clusters created when using different values of eps and minPts for DBSCAN.

In this case, we would argue that dbscan more accurately finds 3 clusters than kmeans did.

We notice that even though the distribution of GLY-points is quite similar to the general distribution, the clusters created by both DBSCAN and k-means are quite different from the general clusters. For example, k-means found 4 clusters for GLY whilst it found 3 general clusters. In DBSCAN the red and the green cluster from the general clusters (see figure 10) looks very different from the yellow and green cluster in figure 20.

We note however that the clusters found from GLY are much more alike the general clusters than the clusters found from PRO. For example, in DBSCAN (GLY) with 3 clusters, the bottom left cluster (red) is actually very similar to the bottom left cluster in the general clusters found by DBSCAN.

This seems reasonable as the distribution of GLY-points is a lot more similar to the general distribution than the distribution of the PRO-points.