Introduction to Data Science and AI – Report – Assignment 3

Big Data Technologies, Sofia University "St. Kliment Ohridski"

Stefan Dimitrov Velev, 0MI3400521

In this assignment I had to work with a data set containing a list of phi and psi combinations that have been observed in a large set of proteins.

The packages I have used in this assignment are: ***Pandas*** (for reading and storing data), ***NumPy*** (for making statistics), ***Matplotlib*** (for visualisation), and ***Scikit-learn*** (machine learning library that supports supervised and unsupervised learning). The specific modules that I have used can be found in the following code snippet which is compulsory in the situations when external classes and functions are required:

```python
# Import required Python libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.cluster import KMeans
from sklearn.cluster import DBSCAN
```

The first step before all the tasks is to load the dataset using *Pandas* `read_csv(…)` function:

```python
# Read the given CSV file
df = pd.read_csv('./data/data_assignment3.csv', delimiter=',')
```

A following step is to remove all the rows with missing values. In that case, there are not any in the dataset. However, this step should always be considered whenever starting to work with unfamiliar data. It is not compulsory to remove these rows but the consideration process is a necessity.

```python
# Remove rows with missing values (if there are any)
df = df.dropna()
```

To gain an impression of the data, I used `df.describe()` for generating descriptive statistics. We can notice that there are 29 369 rows in the data set. This is not a small number and that means we should be careful and consider that in all our further tasks.

Task 1: Show the distribution of phi and psi combinations using:

a. A scatter plot

We have to take into consideration that phi and psi are angles in degrees, so they are in the interval (-180°, 180°). That is why I chose to limit the plot from -180 to 180 on both axes. Since the combinations are over 29 000, I had to depict the points smaller so that they can be seen clearly. In addition, I changed their opacity to 0.8 for better visualisation. Setting the tick locations is also important when working with such values. I chose intervals of length 60 on both axes for that purpose. Setting labels on both axes (with units of measurement) is momentous for making the correct interpretations. The final step was to put a title on the scatter plot.

```python
# Draw scatter plot of the data
plt.figure(figsize=(12, 12))
plt.scatter(df['phi'], df['psi'], s = 2, alpha = 0.8)
plt.xlim(-180, 180)
plt.ylim(-180, 180)
plt.xticks(range(-180, 181, 60))
plt.yticks(range(-180, 181, 60))
plt.xlabel('Phi Angle (in degrees)')
plt.ylabel('Psi Angle (in degrees)')

plt.title('Scatter Plot of Phi and Psi Combinations Observed in a Large Set of Proteins')
plt.show()
```

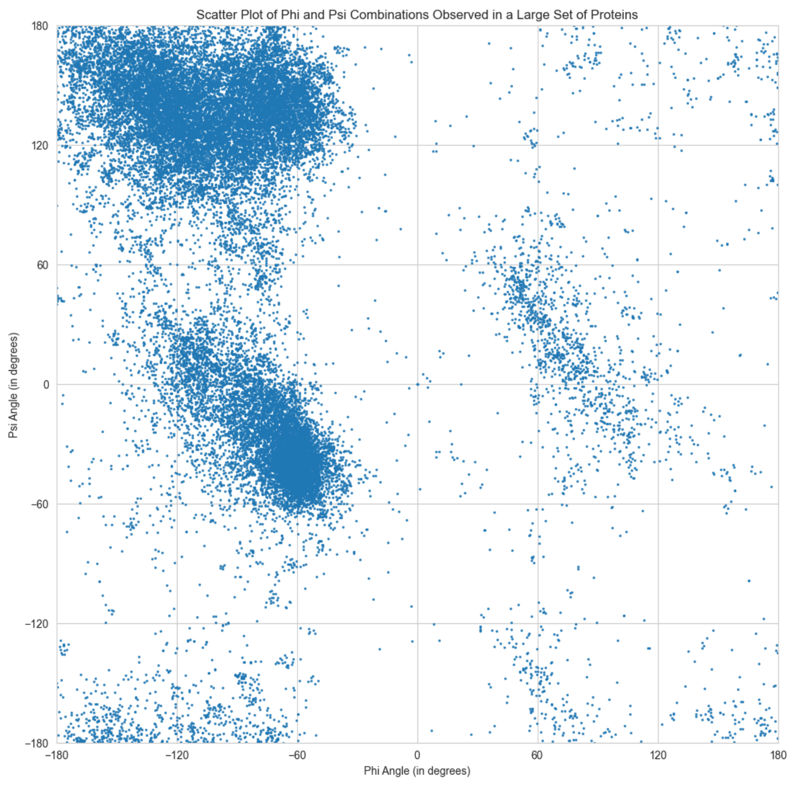

Figure 1: *Scatter Plot of Phi and Psi Combinations Observed in a Large Set of Proteins*

b. A 2D histogram

When we work with 2D histograms, we must consider some factors. First, we have to set the number of bins for the two dimensions. My choice is 70 since I have made several attempts and with that values the 2D histogram seems smoother. Another important decision is the colour scheme of the 2D histogram. I chose ‘ocean’ which depicts blue areas on green background getting darker in the areas where more points are located. The two colours are distinct enough for me to draw a conclusion of the distribution. Moreover, in 2D histograms we need colour bars that are visualisations of the mapping from scalar value to colours. In most cases, they display the colour scale from the minimum to the maximum values in the data, helping us understand the colour variations in the plot. In our case, it shows the number of combinations in the bins. The compulsory elements such as labelling the x-axis, labelling the y-axis and putting title of the diagram are considered as well.

```
# Draw 2D histogram of the data
plt.figure(figsize=(12, 10))
plt.hist2d(df['phi'], df['psi'], bins=70, cmap='ocean')
plt.xticks(range(-180, 181, 60))
plt.yticks(range(-180, 181, 60))
plt.colorbar(label = "Number of Combinations in Bin")
plt.xlabel('Phi Angle (in degrees)')
plt.ylabel('Psi Angle (in degrees)')
plt.title('2D Histogram of Phi and Psi Combinations Observed in a Large Set of Proteins')
plt.show()
```

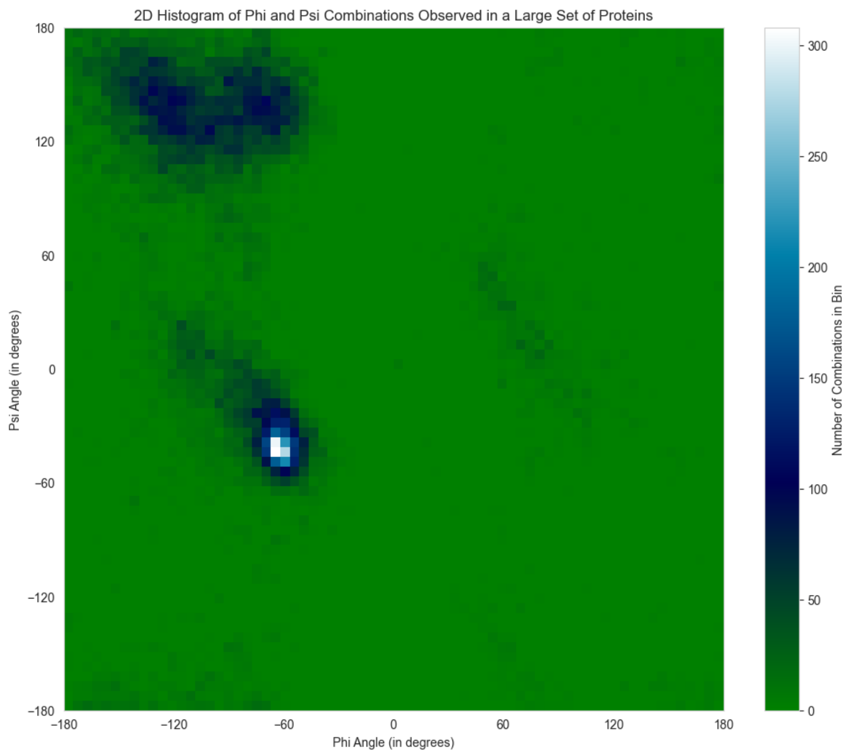

Figure 2: *2D Histogram of Phi and Psi Combinations Observed in a Large Set of Proteins*

Task 2: Use the K-means clustering method to cluster the phi and psi angle combinations in the data file:

a. Experiment with different values of K. Suggest an appropriate value of K for this task and motivate the choice.

First, we make a copy of the data with the phi and psi angle combinations only. After that, we create an empty list to use for storing the *inertia* – the sum of squared distances of samples to their closest cluster centre, weighted by the sample weights if present. This is data that I use later for applying the elbow method on the results. Since I have to run the algorithm many times for different values of K, I prefer to make a function for that. It accepts the number K that represents the desired number of clusters. I use the model from the ***Scikit-learn*** library which calculates the inertia for me as well. The colour map which I use is 'viridis'. The rules for better visualisation of a scatter plot apply for that situation as well.

```
X = df[['phi', 'psi']].copy()
inertia = []

def visualise_kmeans(k):
    kmeans = KMeans(n_clusters = k, n_init = 10)
    kmeans.fit(X)
    y_kmeans = kmeans.predict(X)
    inertia.append(kmeans.inertia_)

    plt.figure(figsize=(9, 9))
    plt.scatter(X['phi'], X['psi'], c = y_kmeans, s = 2, alpha = 0.8, cmap = 'viridis')
    plt.xlim(-180, 180)
    plt.ylim(-180, 180)
    plt.xticks(range(-180, 181, 60))
    plt.yticks(range(-180, 181, 60))
    plt.xlabel('Phi Angle (in degrees)')
    plt.ylabel('Psi Angle (in degrees)')

    plt.title(f'KMeans Clustering with k = {k} on Phi and Psi Combinations Observed in a Large Set of 
Proteins')
    plt.show()
```

To run the function, I construct a loop with values of k from 2 to 10. First, I want to assess the results visually. After that, I use the `inertia` list to make a plot of the elbow method on the results. The rules for labelling are the same.

```
for k in range(2, 11):
    visualise_kmeans(k)

plt.figure(figsize=(8, 8))
plt.plot(range(2, 11), inertia, marker='o')
plt.title('Elbow Method for KMeans Clustering on Phi and Psi Combinations Observed in a Large Set of
Proteins')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia (WCSS)')
plt.show()
```

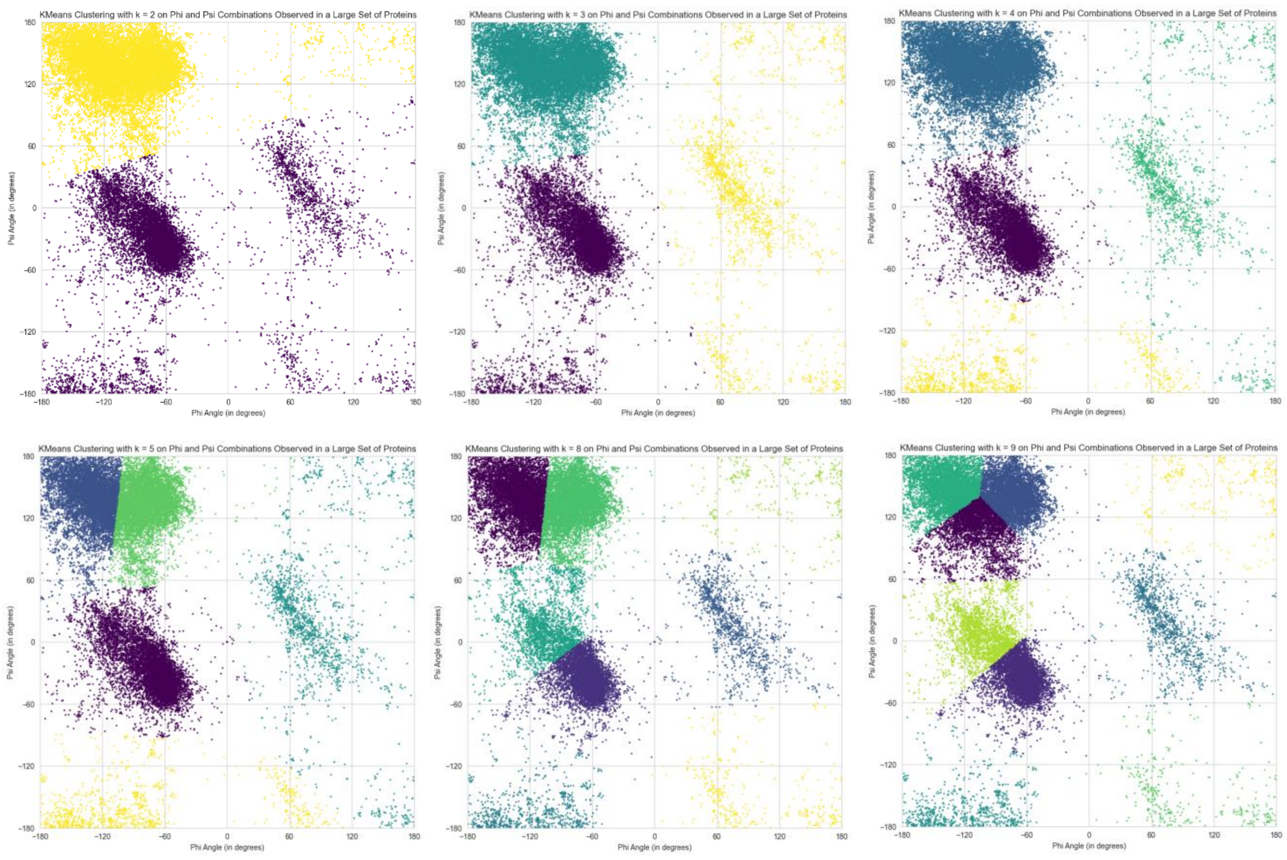

Figure 3: *K-means clustering for Different Values of K on Phi and Psi Coordinates Observed in a Large Set of Proteins*

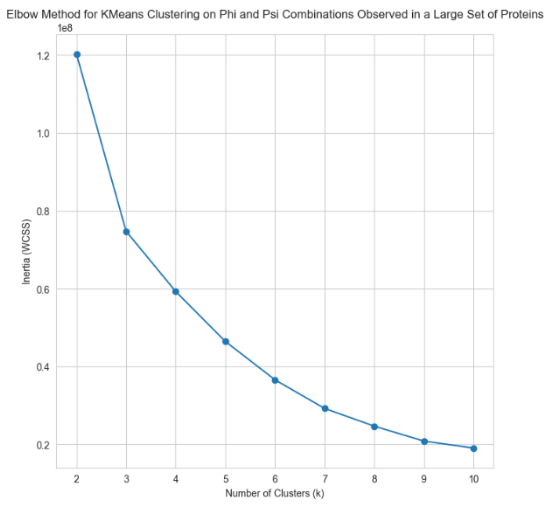

Figure 4: *Elbow Method for K-means Clustering on Phi and Psi Combinations Observed in a Large Set of Proteins*

I can suppose visually that the best results for me are for k = 3 and k = 4. I would prefer the number of clusters to be four as it is not a good idea the points that are on the bottom edge to be part of the middle clusters since the distance between them is big enough to say that they are separated.

We can see from the elbow method that k = 3 and k = 4 are the best options, slightly prevailing for three number of clusters. That is because of the scattered points all over the plot.

b. Do the clusters found in part (a) seem reasonable.

As the number of clusters increases, the motivation behind such choices decreases. However, the separation seems proportional which is a good feature of the K-means algorithm. For my preferred choice of four clusters, I think they seem reasonable at first glance. The clusters are well-formed. However, there are a lot of scattered points some of which seem to be outliers. Nevertheless, the algorithm is bound to assign them to the nearest cluster. On second reading, as we observe more carefully, we notice a curious phenomenon.

c. The top edge of a Ramachandran plot wraps round to the bottom edge, and the right edge wraps around to the left edge (we can think of the 2D Ramachandran plot being mapped onto the surface of a torus). Ideally, this should be considered when clustering the data points on a Ramachandran plot. Repeat questions (a) and (b) taking this into account.

It is true that the top edge wraps round to the bottom edge and vice versa. The same fact is true for the right and the left edge of the Ramachandran plot. In other words, this means, for instance, that we must consider the points on the bottom edge to be part of the cluster around the top edge since the surface resembles a torus. We can relate this observation to the concept of periodic functions that are functions which repeat themselves at regular intervals. For example, values of 359° for φ and ψ must be close to 0°. This means that the data points near the boundary of the plot have to be clustered together even though they may appear far apart in a regular 2D plane.

To handle this periodicity properly, we can transform the angles into polar coordinates and then perform K-means clustering. This allows us to treat the data points as if they are points on a torus. We convert the angles into radians using the `np.radians(…)` function. After that, we calculate the cosine and the sine of φ and ψ respectively. We stack the results in columns and perform the same K-means algorithm for the new `X_polar` ndarray. As I do not want to modify the original scatter plot, the points are preserved, but the way how they are interpreted by the model is different. As before, we make several experiments.

```python
# Transform to polar coordinates to tackle problem with periodicity
phi = df['phi'].values
psi = df['psi'].values

phi_cos = np.cos(np.radians(phi))
phi_sin = np.sin(np.radians(phi))
psi_cos = np.cos(np.radians(psi))
psi_sin = np.sin(np.radians(psi))

X_polar = np.column_stack((phi_cos, phi_sin, psi_cos, psi_sin))
inertia = []
```

```python
def visualise_kmeans_polar(k):
    kmeans = KMeans(n_clusters=k, n_init=10, random_state=98)
    kmeans.fit(X_polar)
    y_kmeans = kmeans.predict(X_polar)
    inertia.append(kmeans.inertia_)

    plt.figure(figsize=(9, 9))
    plt.scatter(X['phi'], X['psi'], c=y_kmeans, s=2, alpha=0.8, cmap='viridis')
    plt.xlim(-180, 180)
    plt.ylim(-180, 180)
    plt.xticks(range(-180, 181, 60))
    plt.yticks(range(-180, 181, 60))
    plt.xlabel('Phi Angle (in degrees)')
    plt.ylabel('Psi Angle (in degrees)')

    plt.title(f'KMeans Clustering with k = {k} based on Phi and Psi Polar Coordinates Observed in a
Large Set of Proteins')
    plt.show()
```

```
for k in range(2, 11):
    visualise_kmeans_polar(k)

plt.figure(figsize=(8, 8))
plt.plot(range(2, 11), inertia, marker='o')
plt.title('Elbow Method for KMeans Clustering based on Phi and Psi Polar Coordinates Observed in a
Large Set of Proteins')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia (WCSS)')
plt.show()
```

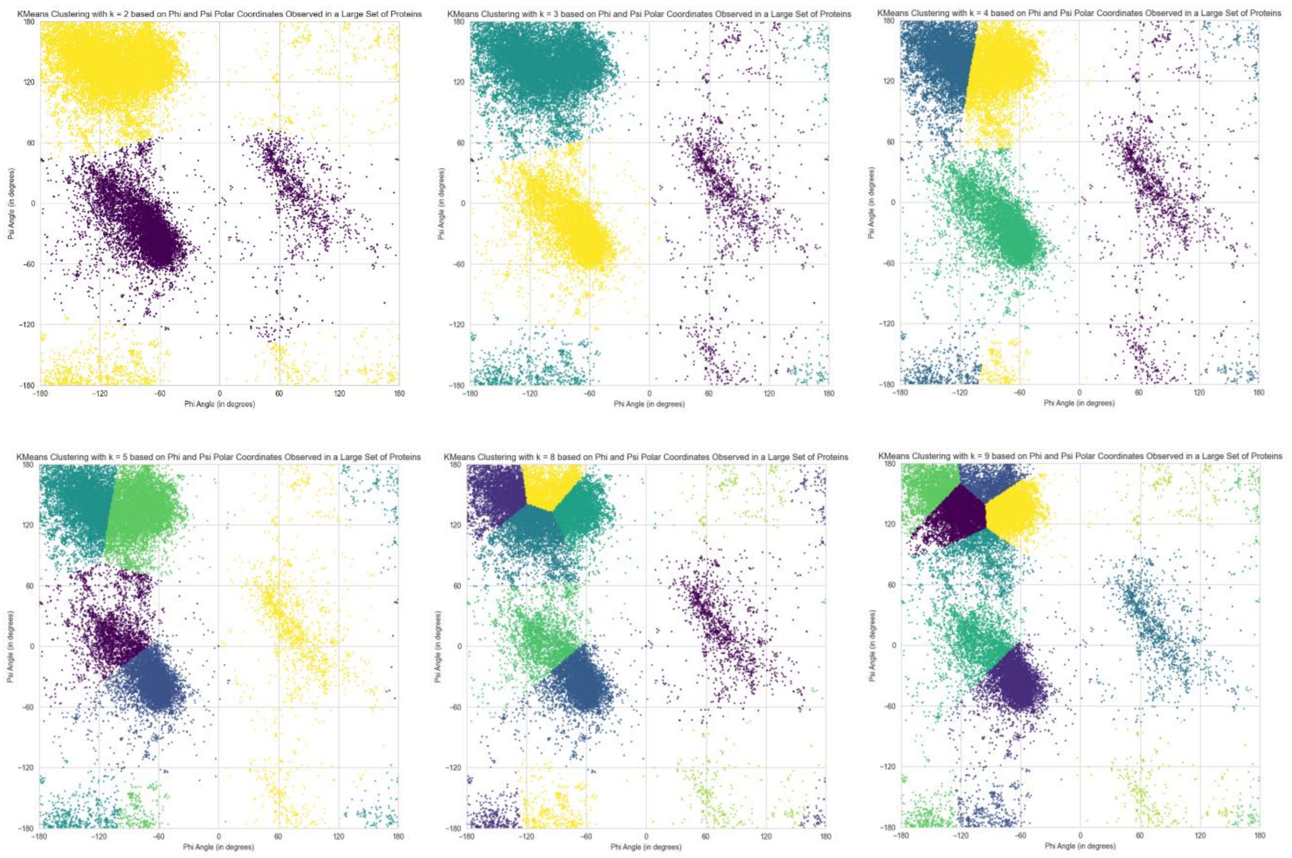

***Figure 5:** K-means for Different Values of K based on Phi and Psi Polar Coordinates Observed in a Large Set of Proteins*

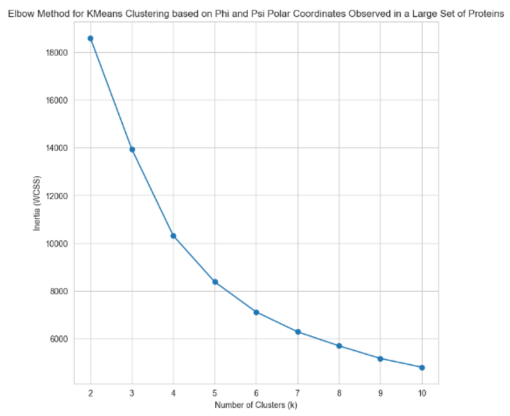

***Figure 6:** Elbow Method for K-means Clustering based on Phi and Psi Polar Coordinates Observed in a Large Set of Proteins*

I can suppose visually that for me the best result is for k = 3. As we can see now, the bottom left and the bottom right points are part of the upper left cluster which means that now we are considering them to be closer to one another although we cannot see that on the 2D plot. The result of the elbow method supports my choice.

Task 3: Use the DBSCAN method to cluster the phi and psi angle combinations in the data file.

a. Motivate the choice of the minimum number of samples in the neighbourhood for a point to be considered as a core point, and the maximum distance between two samples belonging to the same neighbourhood ('eps'). Compare the clusters found by DBSCAN with those found using K-means.

We will start with performing the DBSCAN algorithm for different values of `min_samples` and eps on the modified data with phi and psi in polar coordinates. The reason is that we want more objective results as we already have the previous consideration in mind. The values which we use for the attempts are: for *the minimum number of samples in the neighbourhood for a point to be considered as a core point* – 20, 30, 40, 50 and for each of the previous we choose *the maximum distance between two samples belonging to the same neighbourhood* – 0.2, 0.3, 0.4. The chosen values are in conformity with the modified polar coordinates variant where the distances are expected to be smaller as values. `db.core_sample_indices_` identifies indices of the core points in the data. A `core_samples_mask` array marks these core points while the `labels` array assigns cluster ids to each data point (noise is indicated by -1). The data is visualised using a scatter plot where points are coloured by their cluster assignment whereas outliers are plotted in black. The number of clusters and outliers for each parameter combination is printed, and the corresponding plot is displayed.

```
# Performing DBSCAN for different values of min_samples and eps on phi and psi polar coordinates data
for min_samples in [20, 30, 40, 50]:
    for eps in [0.2, 0.3, 0.4]:
        db = DBSCAN(eps=eps, min_samples=min_samples)
        y_db = db.fit_predict(X_polar)

        core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
        core_samples_mask[db.core_sample_indices_] = True
        labels = db.labels_

        num_clusters = len(set(labels)) - (1 if -1 in labels else 0)
        num_outliers = list(labels).count(-1)

        plt.figure(figsize=(9, 9))
        plt.scatter(X['phi'], X['psi'], c=y_db, cmap='viridis', s=2, alpha=0.8)

        outliers_mask = labels == -1
        plt.scatter(X['phi'][outliers_mask], X['psi'][outliers_mask], c='black', s=2, alpha=0.8)

        plt.xlim(-180, 180)
        plt.ylim(-180, 180)
        plt.xticks(range(-180, 181, 60))
        plt.yticks(range(-180, 181, 60))
        plt.xlabel('Phi Angle (in degrees)')
        plt.ylabel('Psi Angle (in degrees)')

        plt.title(f'DBSCAN Clustering with min samples = {min_samples} and eps = {eps} based on Phi and
Psi Polar Coordinates Observed in a Large Set of Proteins')
        plt.show()

        print(f'Number of clusters: {num_clusters}')
        print(f'Number of outliers: {num_outliers}')
```

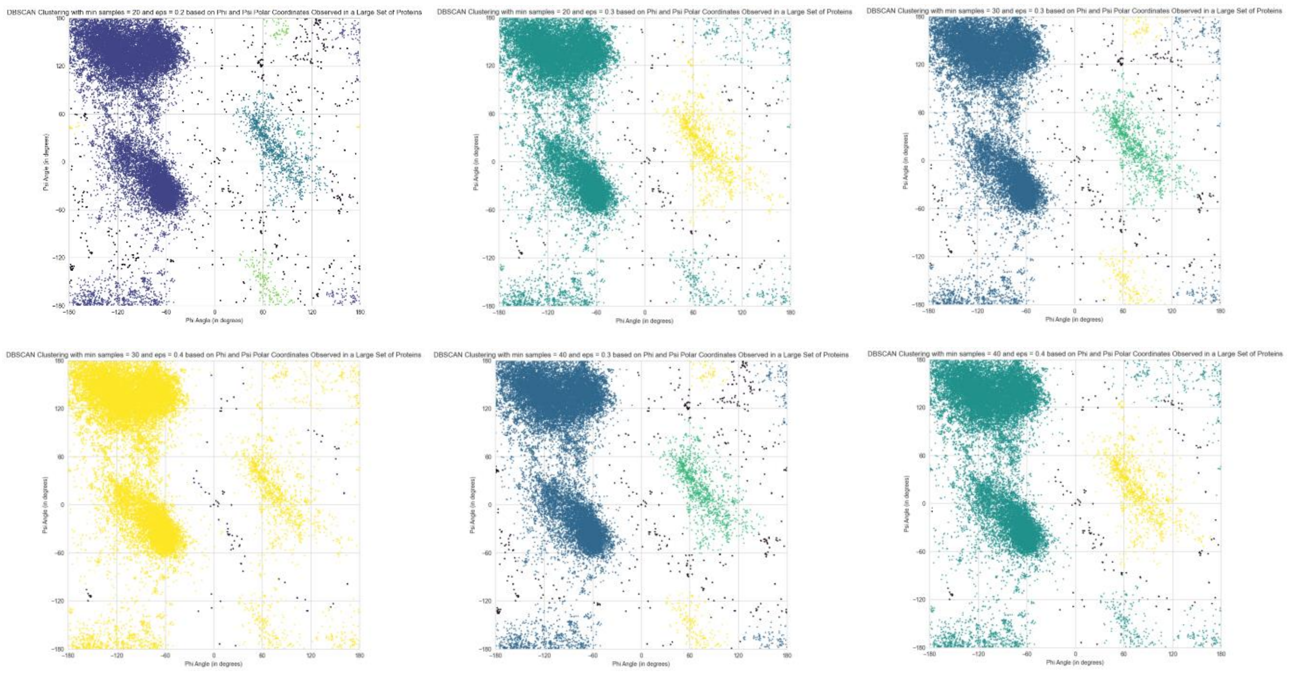

Figure 7: *DBSCAN Clustering with different values for min samples and eps based on Phi and Psi Polar Coordinates Observed in a Large Set of Proteins*

Some of the results are presented above. I can suppose visually that for me the best results are those with three clusters. The plot with eps = 0.3 and min samples = 30 seems to arrive at a compromise.

When working with the original data, the parameters of the DBSCAN algorithm must be changed since the distances are bigger. The scatter plot presented below is with min samples = 80 and eps = 30. In that situation, we see that 5 clusters are formed with 119 outliers. It is clear why the clusters are more than the previous case – the algorithm does not consider the unifications of the borders (as in this case we use the original Phi and Psi combination data).

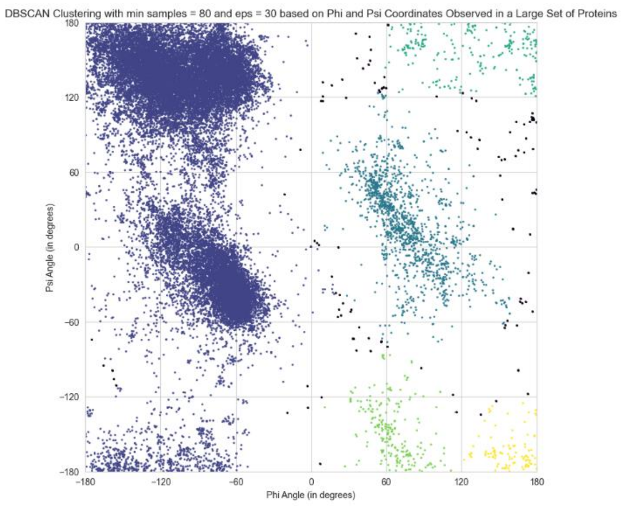

Figure 8: *DBSCAN Clustering with min samples = 80 and eps = 30 based on Phi and Psi Coordinates Observed in a Large Set of Proteins*

The key difference between clusters formed by K-means and DBSCAN lies in their definitions and flexibility. K-means creates clusters based on proximity to centroids, assuming clusters are spherical and requiring the user to predefine the number of clusters. It struggles with irregularly shaped clusters. DBSCAN, on the other hand, identifies clusters as dense regions of points separated by sparse areas, without needing to predefine the number of clusters. It excels at finding clusters of arbitrary shapes and automatically labels sparse regions as noise (outliers) while K-means does not mark points as outliers. We notice that the DBSCAN algorithm seems to find a lot of clusters easier without splitting the bigger clusters as the K-means model does.

b. Highlight the clusters found using DBSCAN and any outliers in a scatter plot.

On all of the above plots for that task we can see the outliers coloured in black and the clusters coloured by their clustered assignments. The choice of colours for the clusters is from the colormap 'viridis'.

c. How many outliers are found? Plot a bar chart to show how often each of the amino acid residue types are outliers.

For that task I choose to work on the DBSCAN plot with eps = 0.3 and min samples = 30 for the modified data in polar coordinates. The scatter plot is formed of 3 clusters and there are 280 outliers highlighted in black. Now we must check what amino acid residues type they are. I make a copy of the original data using the three columns – 'residue name', 'phi' and 'psi'. Then I add the provided labels from the algorithm as a fourth column named 'label'. I get all the noise points (the outliers) as a separate data frame. I use the method `value_counts()` for the column 'residue name' of these outliers which counts the occurrences of each unique value in that column. As a result, it produces a Series where the index represents the residue types and the values represent their respective counts. Then I create a bar chart where the x-axis represents residue types (`residue_counts.index`), and the y-axis represents their frequencies (`residue_counts.values`).

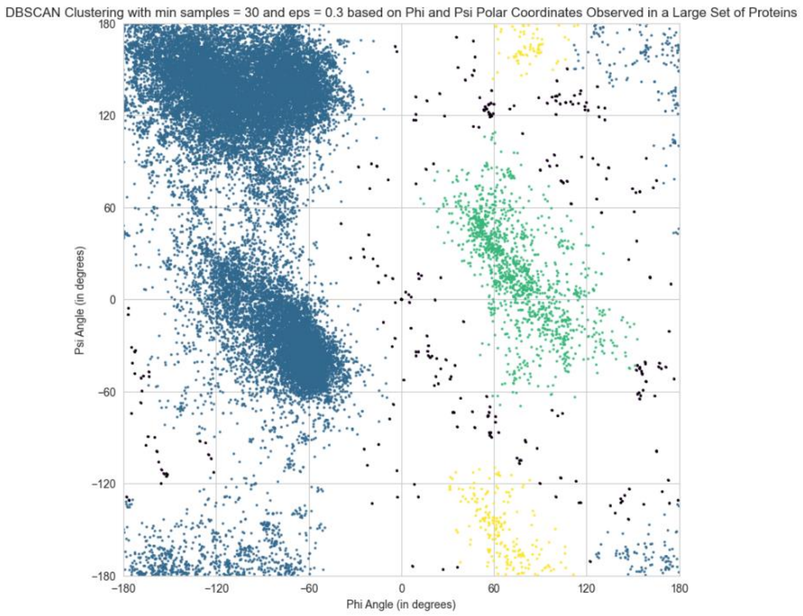

Figure 9: *DBSCAN Clustering with min samples = 30 and eps = 0.3 based on Phi and Psi Polar Coordinates Observed in a Large Set of Proteins*

```
X_residue = df[['residue name', 'phi', 'psi']].copy()
X_residue['label'] = y_db
outliers = X_residue[X_residue['label'] == -1]
```

```
residue_counts = outliers['residue name'].value_counts()

plt.figure(figsize=(14, 6))
residue_counts.plot(kind='bar', color='skyblue', edgecolor='black')
plt.title('Frequency of Residue Types in Outliers Found with DBSCAN (min samples = 30, eps = 0.3) based
on Phi and Psi Polar Combinations Observed in a Large Set of Proteins')
plt.xlabel('Residue Type')
plt.ylabel('Frequency')
plt.show()
```

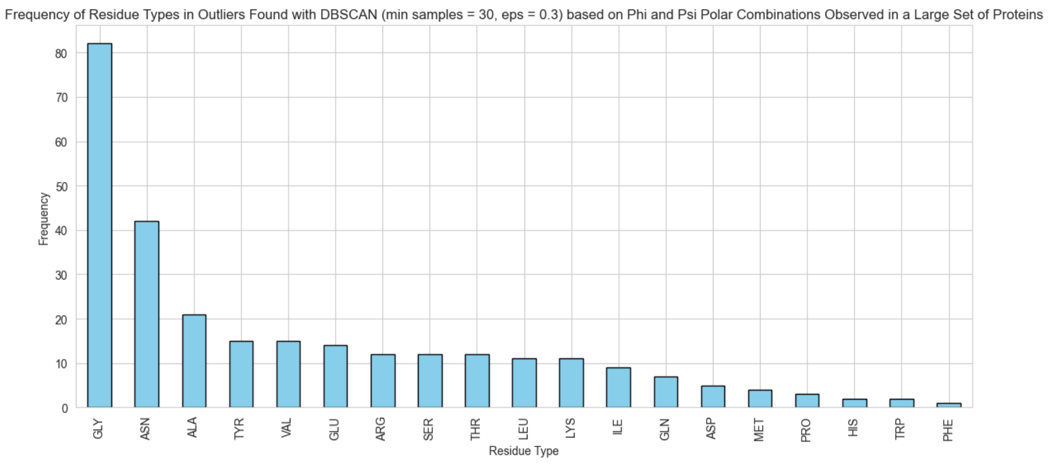

Figure 10: *Frequency of Residue Types in Outliers Found with DBSCAN (min samples = 30, eps = 0.3) based on Phi and Psi Polar Combinations Observed in a Large Set of Proteins*

Task 4: The data file can be stratified by amino acid residue type. Use DBSCAN to cluster the data that have residue type PRO. Investigate how the clusters found for amino acid residues of type PRO differ from the general clusters.

We make another copy of the original data with the three columns – 'residue name', 'phi', 'psi'. After that, we get all the combinations that have residue type name 'PRO'. Their number in the data set is 1 596. To visualise these points, we use a scatter plot:

```
# Draw scatter plot of the data
plt.figure(figsize=(9, 9))
plt.scatter(X_residue_pro['phi'], X_residue_pro['psi'], s = 2, alpha = 0.8)
plt.xlim(-180, 180)
plt.ylim(-180, 180)
plt.xticks(range(-180, 181, 60))
plt.yticks(range(-180, 181, 60))
plt.xlabel('Phi Angle (in degrees)')
plt.ylabel('Psi Angle (in degrees)')

plt.title('Scatter Plot of Phi and Psi Combinations for Residue Type PRO Observed in a Large Set of Proteins')
plt.show()
```

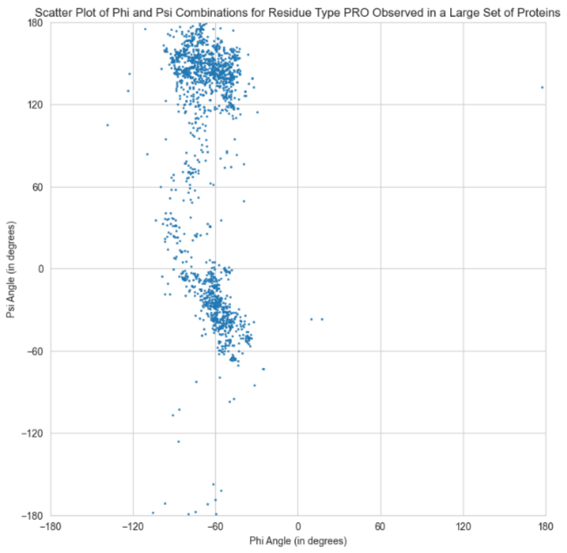

Figure 11: *Scatter Plot of Phi and Psi Combinations for Residue Type PRO Observed in a Large Set of Proteins*

We have to perform the DBSCAN algorithm only on that points. We do that with the original data and so choose *the minimum number of samples in the neighbourhood for a point to be considered as a core point* to be 10 and *the maximum distance between two samples belonging to the same neighbourhood* to be 10 as well.

```
X_residue_pro_coordinates = X_residue_pro[['phi', 'psi']].copy()
db_residue = DBSCAN(eps=10, min_samples=10)
y_db_pro = db_residue.fit_predict(X_residue_pro_coordinates)

core_samples_mask = np.zeros_like(db_residue.labels_, dtype=bool)
core_samples_mask[db_residue.core_sample_indices_] = True
labels = db_residue.labels_

num_clusters = len(set(labels)) - (1 if -1 in labels else 0)
num_outliers = list(labels).count(-1)

plt.figure(figsize=(9, 9))
plt.scatter(X_residue_pro['phi'], X_residue_pro['psi'], c=y_db_pro, cmap='viridis', s=2, alpha=0.8)

outliers_mask = labels == -1
plt.scatter(X_residue_pro['phi'][outliers_mask], X_residue_pro['psi'][outliers_mask], c='black', s=2,
alpha=0.8)

plt.xlim(-180, 180)
plt.ylim(-180, 180)
plt.xticks(range(-180, 181, 60))
plt.yticks(range(-180, 181, 60))
plt.xlabel('Phi Angle (in degrees)')
plt.ylabel('Psi Angle (in degrees)')

plt.title(f'DBSCAN Clustering with min samples = 30 and eps = 0.3 based on Residue Type PRO Phi and Psi
Coordinates Observed in a Large Set of Proteins')
plt.show()

print(f'Number of clusters: {num_clusters}')
print(f'Number of outliers: {num_outliers}')
```

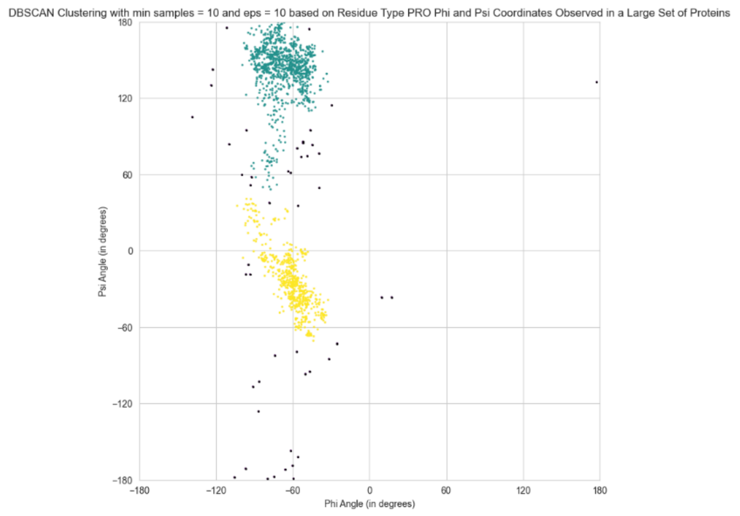

Figure 12: *DBSCAN Clustering with min samples = 10 and eps = 10 based on Residue Type Pro Phi and Psi Coordinates Observed in a Large Set of Proteins*

As a result, we get 2 clusters and 49 noise points (outliers). By comparison, we can see that these points are always part of one cluster in the scatter plots from question 3. For the K-means model from question 2 there are more variations. The reason lies in the different parameters (`eps` and `min_samples`) that we use for the models. On the other hand, domain knowledge is required for their choice which for that case is specifically in the field of bioinformatics. That is why we have so many different variations. Moreover, the cartesian coordinates and the polar coordinates form two entirely different tasks and as such many tests can and still need to be performed so as to be able to form thorough conclusions.