CPS 803 Assignment 4 Report

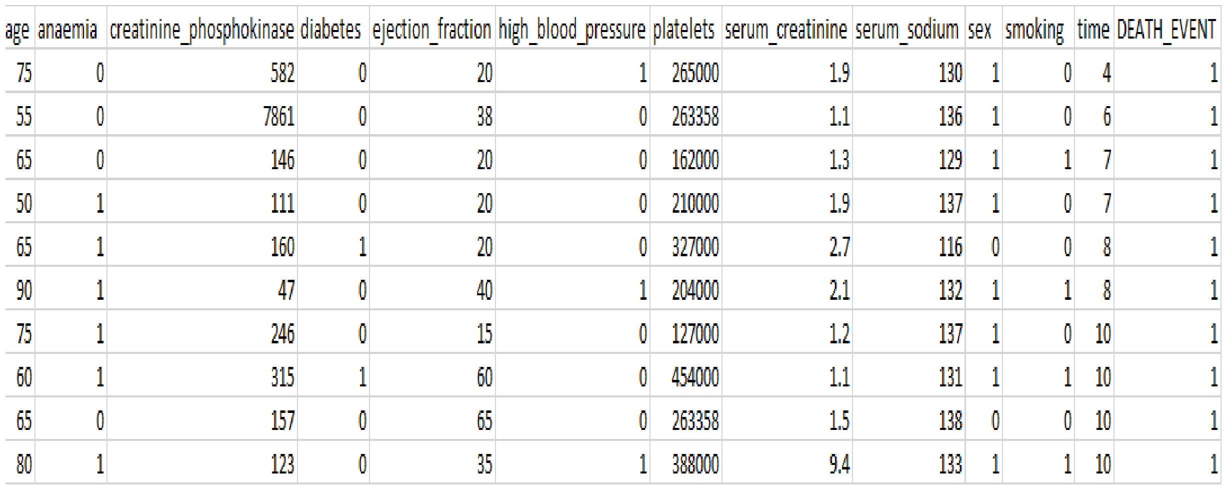
Mahyar Tajeri

Professor Elodie Lugez

December 8, 2022

Background

For this assignment, I had to find a dataset to use with machine learning, using clustering, as well as preprocessing. I decided to use a dataset containing heart failure clinical records, which included information about patients suffering from heart failure. [1] The dataset has 13 features, including age, high blood pressure (yes/no), smoking (yes/no), and other medical information. Below is a screenshot of all the features:



age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_pressure	platelets	serum_creatinine	serum_sodium	sex	smoking	time	DEATH_EVENT
75	0	582	0	20	1	265000	1.9	130	1	0	4	1
55	0	7861	0	38	0	263358	1.1	136	1	0	6	1
65	0	146	0	20	0	162000	1.3	129	1	1	7	1
50	1	111	0	20	0	210000	1.9	137	1	0	7	1
65	1	160	1	20	0	327000	2.7	116	0	0	8	1
90	1	47	0	40	1	204000	2.1	132	1	1	8	1
75	1	246	0	15	0	127000	1.2	137	1	0	10	1
60	1	315	1	60	0	454000	1.1	131	1	1	10	1
65	0	157	0	65	0	263358	1.5	138	0	0	10	1
80	1	123	0	35	1	388000	9.4	133	1	1	10	1

I thought this would be a good potential dataset for clustering because it was categorized as such on the UCI Machine Learning Repository website. [1] I also chose this because finding patterns and groups in heart failure patients can provide insight into what might be causing/symptoms to look out for regarding heart failures.

For this particular task, I chose K-means clustering because it's the most general one [2], and I didn't want to use something like DBSCAN which does better on dense clusters but is poor on sparse data. [2] For the dataset itself, I downloaded it as a csv file and looked at it on excel for visualization.

Methods

For this task, I used `sklearn`, `pandas`, `matplotlib`, and `seaborn` libraries in python. First, I created a pandas DataFrame object by reading the csv with the dataset I downloaded. I printed its shape to get an idea of what I'm working with:

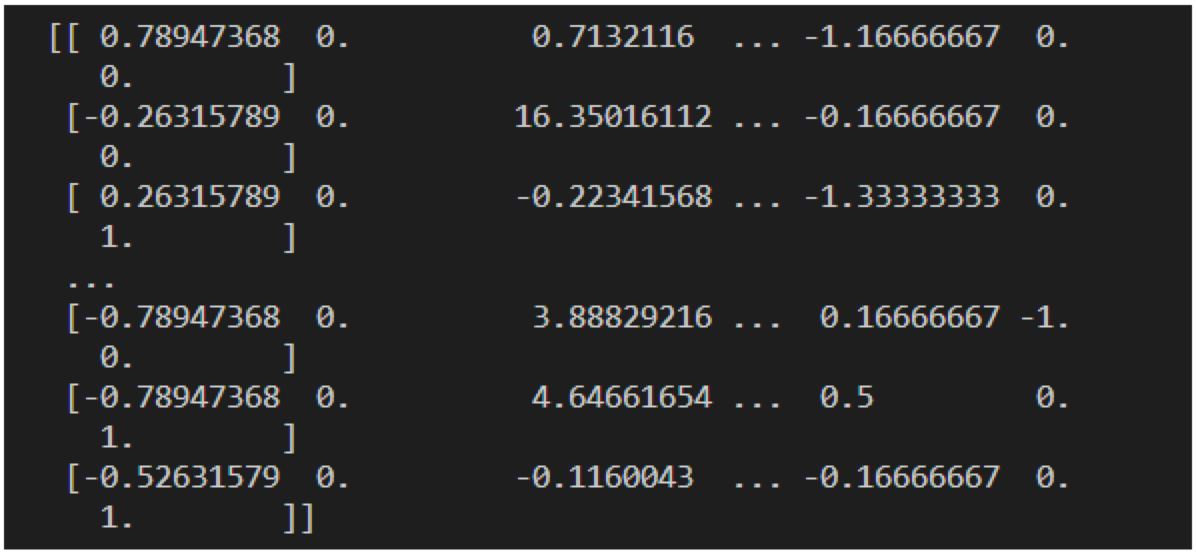
```
(299, 13)
```

Preprocessing

I decided that I didn't want the `time` and `DEATH_EVENT` features for my data because they concern heart failure patients AFTER a diagnosis. I removed these columns with `DataFrame.drop()` and then printed the shape of the data again:

```
(299, 11)
```

Moving forward with this 11 feature data, I decided to scale the data because of the varying ranges for the features. For example, the `platelets` column has values in the hundreds of thousands, whereas the `serum_creatinine` column has values in the tens. Scaling the data will help prevent one feature from dominating the effect
on the clustering model. This is especially important for K-means clustering because K-means uses euclidean distance [2], and an over-represented feature could skew this calculation. [3] I used `RobustScaler` from `sklearn` to scale the data, using the `RobustScaler.fit_transform()` method. `RobustScaler` uses statistical calculations that are resistant to outliers, [4] and I had the best results using this scaler compared to `MinMaxScaler`, `StandardScaler`, or `MaxAbsScaler`. Here's what the data looks like after scaling. As you can see, the values are quite different from the excel screenshot shown before.

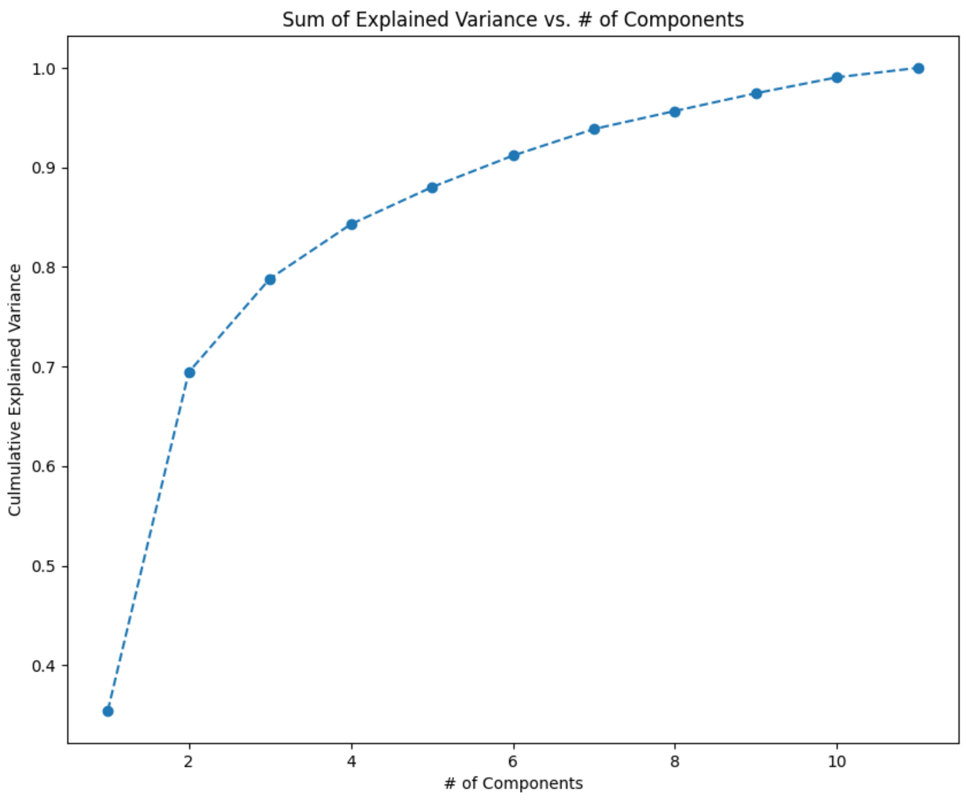


```
[[ 0.78947368  0.          0.7132116  ... -1.16666667  0.
   0.        ]
 [-0.26315789  0.         16.35016112 ... -0.16666667  0.
   0.        ]
 [ 0.26315789  0.         -0.22341568 ... -1.33333333  0.
   1.        ]
 ...
 [-0.78947368  0.          3.88829216 ...  0.16666667 -1.
   0.        ]
 [-0.78947368  0.          4.64661654 ...  0.5         0.
   1.        ]
 [-0.52631579  0.         -0.1160043  ... -0.16666667  0.
   1.        ]]
```

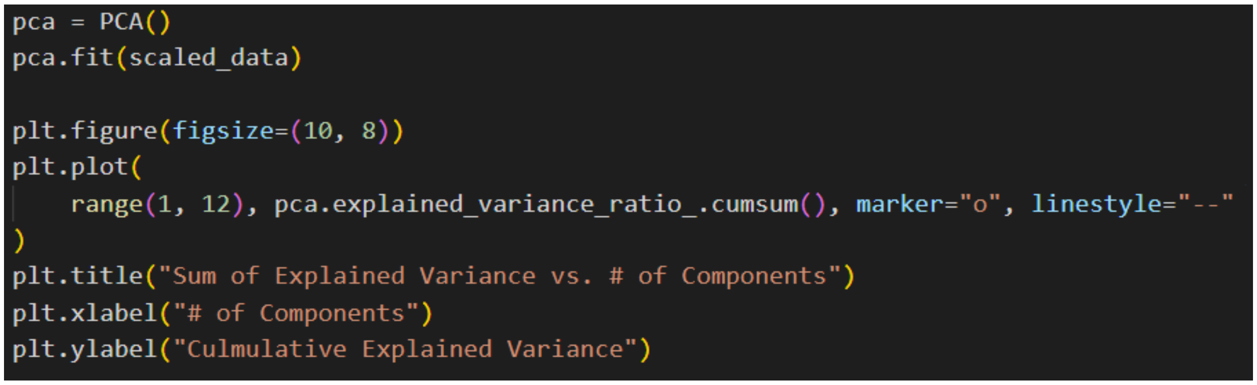
Another preprocessing task that I needed to handle was the number of features. Even after removing 2, I still had 11 features, and showing clustering in 11 dimensions seems inefficient. Another problem is the curse of dimensionality. [5] With 11 features, you would need a lot of data points for the effect of the features to be meaningful since their effects are so spread out. Since we only have about 300 instances, it's important to reduce the number of features.

One way to do this is through principal component analysis (PCA). [6] With PCA, you can compress the information of multiple features into one, without losing a lot of information. [6] Using linear algebra, it takes multiple features and produces linear combinations (new components) that are uncorrelated. [6] The first of these components will try to capture as much variance in the original data as possible, and the second one will try to capture as much as the remaining variance, and so on. [6]

To conduct PCA on my data, I used `PCA` from `sklearn.decomposition`. I used the `fit` method to align it with my scaled data, Then I plotted the cumulative explained variance of each additional component to determine how many components I'd need:



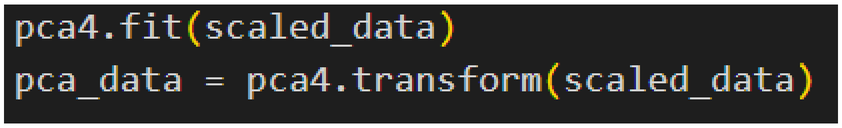
Shown above is how much total explained variance is acquired from forming components. As you can see, at about 4 components, we have surpassed 80% explained variance, which I decided was a good number to stop at. [7] Here is the code for this graph:



```python
pca = PCA()
pca.fit(scaled_data)

plt.figure(figsize=(10, 8))
plt.plot(
    range(1, 12), pca.explained_variance_ratio_.cumsum(), marker="o", linestyle="--"
)
plt.title("Sum of Explained Variance vs. # of Components")
plt.xlabel("# of Components")
plt.ylabel("Culmulative Explained Variance")
```

After I decided on 4 components, I created a new `pca` variable and used the `transform` method to create new data with only the principal components as features: [7]

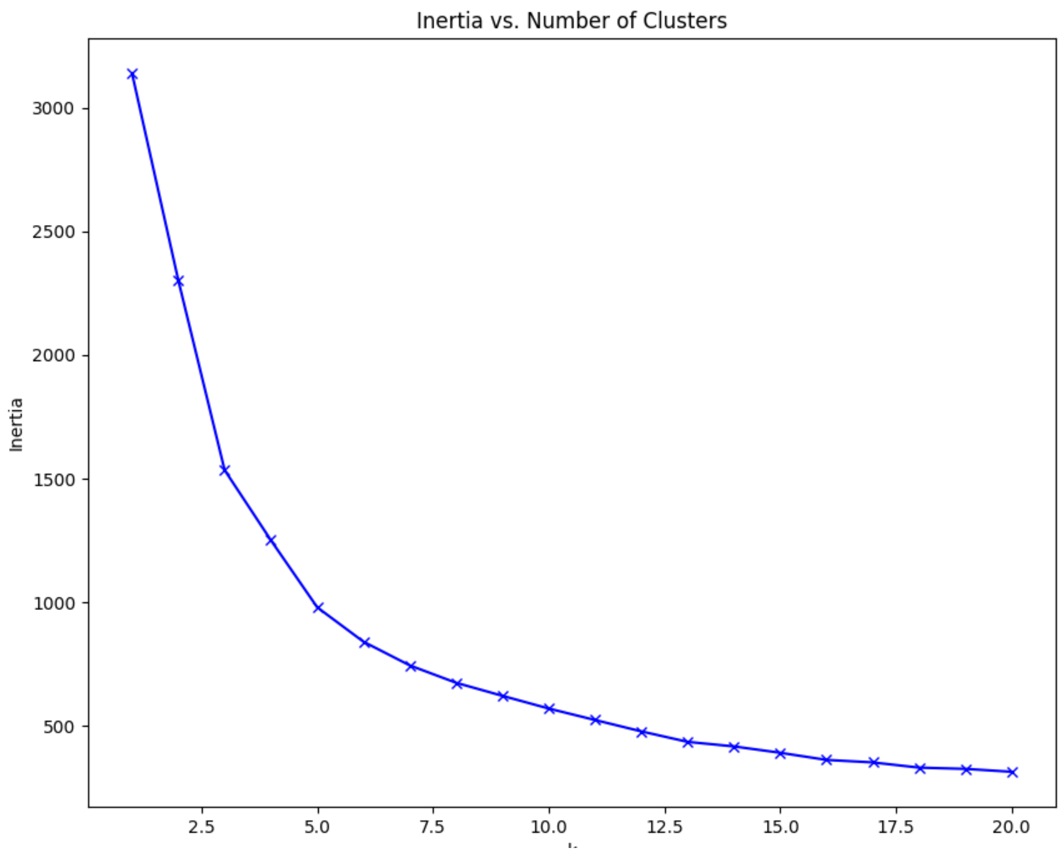


```python
pca4.fit(scaled_data)
pca_data = pca4.transform(scaled_data)
```

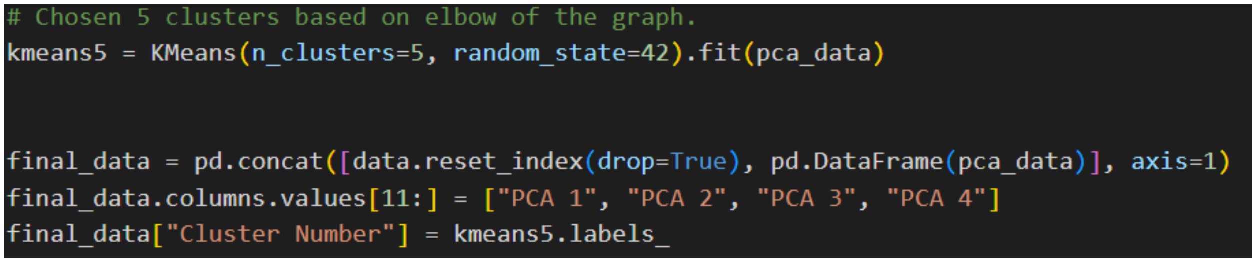
Clustering

Now that I had preprocessed the data, it was time to create the clustering model. I used `KMeans` from `sklearn.cluster` and used the `fit` method to create my model. I also used the parameter `random_state=42` so my models were consistent in case I had to run my program multiple times to debug.

I put this in a loop going from 1 to 20, using the `n_clusters` parameter to control how many clusters there were. I did this to see at which point creating more clusters would be redundant/overfitting: [7]



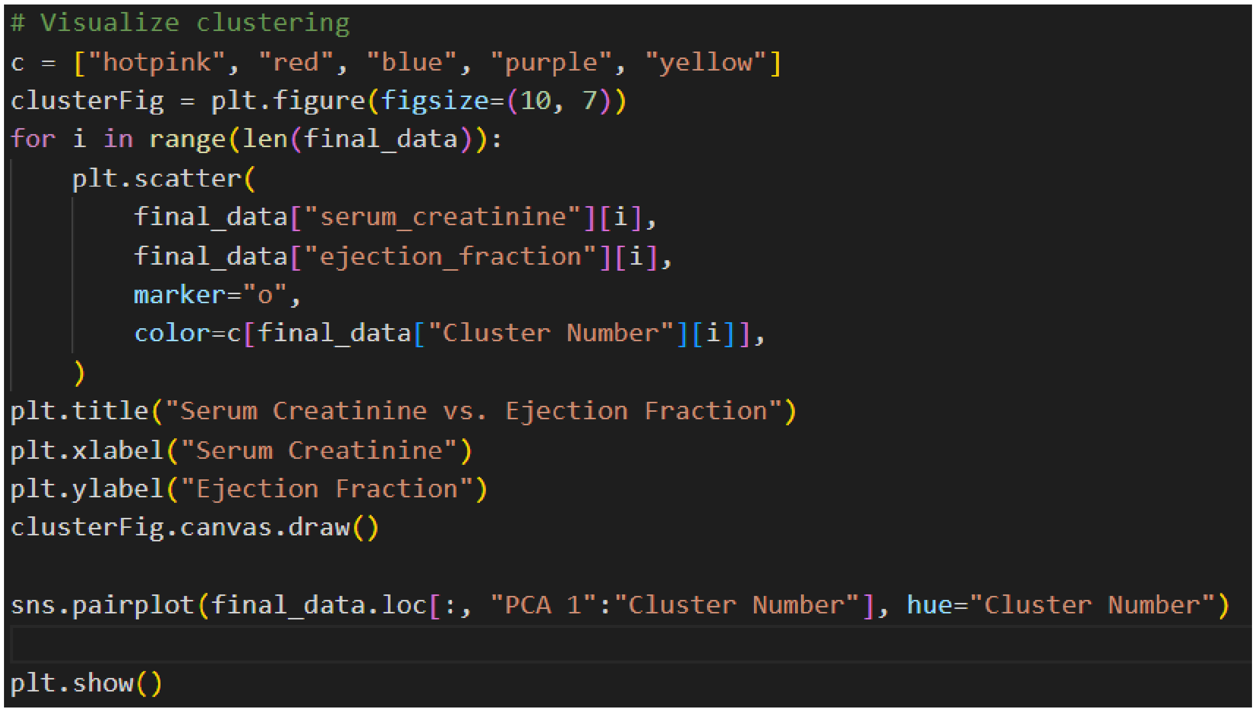
I decided that around 5 clusters was about the point where I started getting diminishing returns, [7] so I decided to move forward with k = 5 clusters. I then created a new `KMeans` model with `n_clusters=5`, and the same `random_state`. Now that I had my model, all that was left to do was organize the results. I created a new `pandas` `DataFrame` to store all of my findings plus the original data. I created 4 new column headers for each principal component created earlier and another column header for the cluster label (0-4, since there are 5 clusters):



```python
# Chosen 5 clusters based on elbow of the graph.
kmeans5 = KMeans(n_clusters=5, random_state=42).fit(pca_data)


final_data = pd.concat([data.reset_index(drop=True), pd.DataFrame(pca_data)], axis=1)
final_data.columns.values[11:] = ["PCA 1", "PCA 2", "PCA 3", "PCA 4"]
final_data["Cluster Number"] = kmeans5.labels_
```

All that is left is to visualize the data, and I used `matplotlib.pyplot` [8] and `seaborn` [9] to do that:



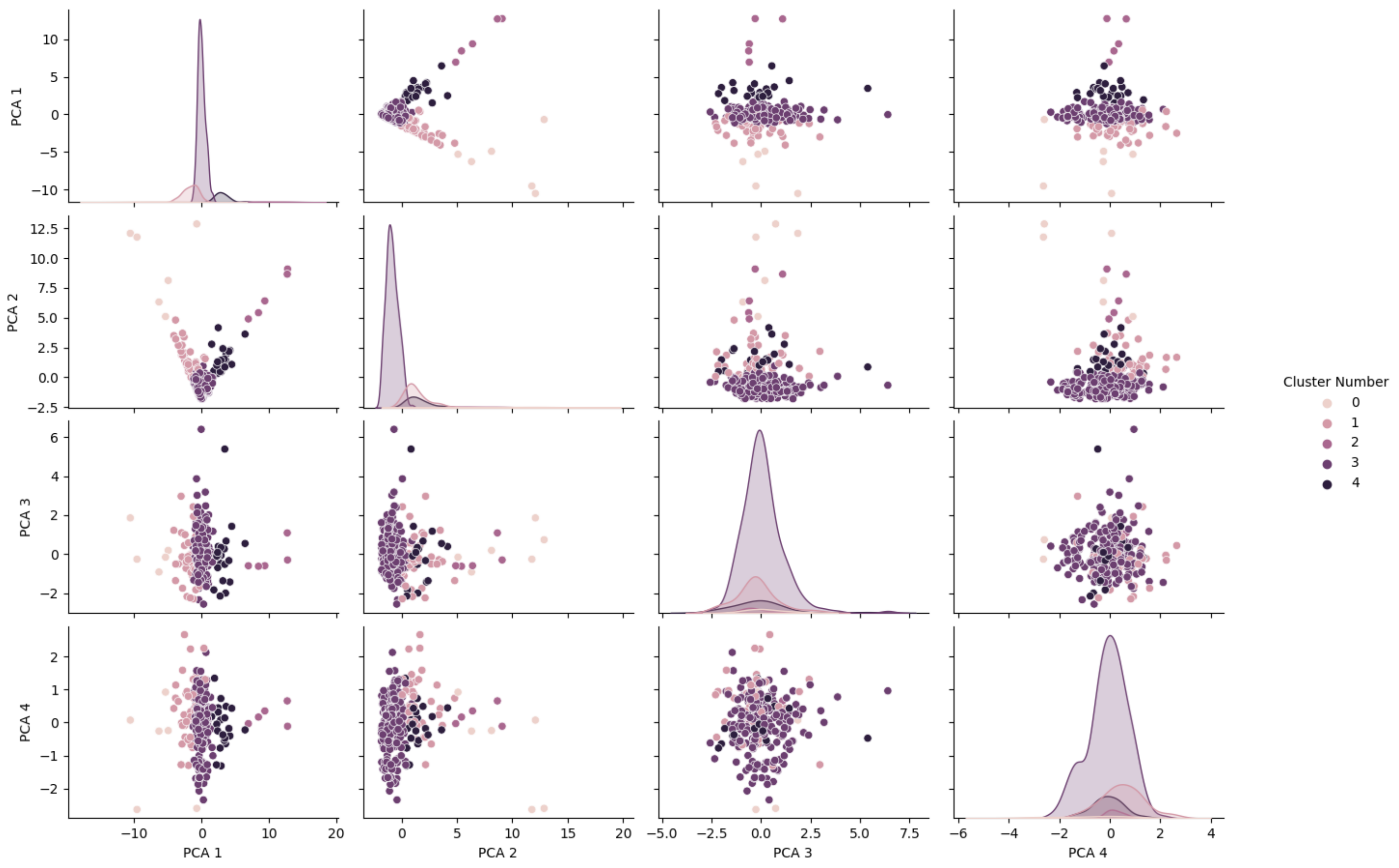
```python
# Visualize clustering
c = ["hotpink", "red", "blue", "purple", "yellow"]
clusterFig = plt.figure(figsize=(10, 7))
for i in range(len(final_data)):
    plt.scatter(
        final_data["serum_creatinine"][i],
        final_data["ejection_fraction"][i],
        marker="o",
        color=c[final_data["Cluster Number"][i]],
    )
plt.title("Serum Creatinine vs. Ejection Fraction")
plt.xlabel("Serum Creatinine")
plt.ylabel("Ejection Fraction")
clusterFig.canvas.draw()

sns.pairplot(final_data.loc[:, "PCA 1":"Cluster Number"], hue="Cluster Number")

plt.show()
```

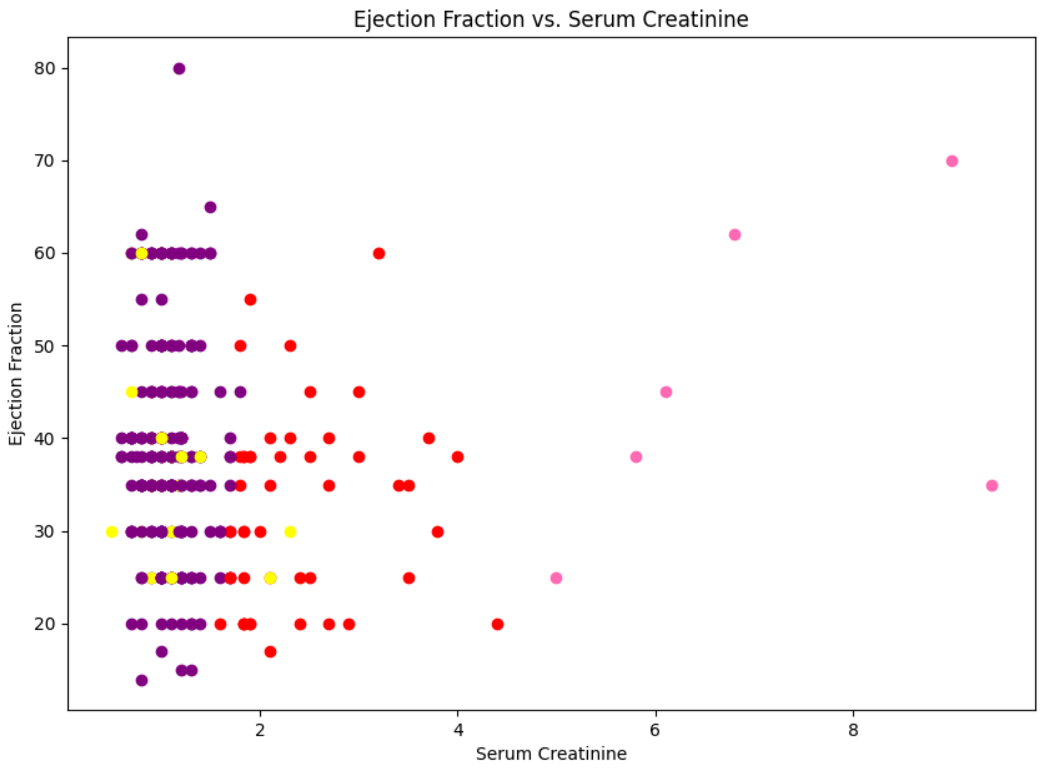
Results

The results of the clustering are as follows. Here is the result of each principal component compared with one another using `seaborn` and its `pairplot` method that makes a plot for each pair of features.



As you can see, the clustering on a lot of the pairings are very clear and make sense. The ones that are like this include `PCA 1` as one of the axes. The ones that seem random are the ones that have a high component number for both axes, such as the two plots using `PCA 3` and `PCA 4` for their axes. This makes sense because the later components have less variance captured from the original data set, and thus are less distinguishable when it comes to clustering. One plot that stood out to me is the one on the bottom left with `PCA 1` as the x-axis and `PCA 4` as the y-axis because the division in the clusters is prominent. Another good one is the second one from the top in the first column because it creates a nice V-shape with distinguishable clusters.

One other noteworthy result is the comparison between the `serum_creatinine` feature and the `ejection_fraction` feature:



This comparison is important because data scientists believe that dying from a heart failure could be accurately determined using these 2 features alone. [1] The clustering shown above isn't perfect, but there is a pretty clear pattern visible, even though we're using non principal components. A classification model would probably be better for determining a relationship for this specific case.

Conclusions

Based on the results discussed earlier, clustering this data set using K-means clustering has been successful in establishing different groups of patients with heart failure. Using the information on the pairwise plots, a clear clustering pattern is established, especially when using the principal components with greater variance. The greatest examples of this include the graph with `PCA 1` on its x-axis and `PCA 2` on its y-axis, as well as the plot with `PCA 4` on its x-axis and `PCA 1` on its y-axis.

Furthermore, using this clustering model on the two features that data scientists claim can predict deaths of heart failure patients, [1] I found a clear pattern, albeit imperfect, but impressive considering I wasn't using any of the principal components in this plot. Based on this finding along with the pairwise plots, I conclude that this clustering model is successful.

References

[1] Tanvir Ahmad, Assia Munir, Sajjad Haider Bhatti, Muhammad Aftab, and Muhammad Ali Raza, Heart Failure Clinical Records Data Set
https://archive.ics.uci.edu/ml/datasets/Heart+failure+clinical+records

[2] scikit-learn, Clustering, https://scikit-learn.org/stable/modules/clustering.html

[3] Ajay Jain, Practical Approach to KMeans Clustering – Python and Why Scaling is Important!
https://medium.com/analytics-vidhya/practical-approach-to-kmeans-clustering-python-and-why-scaling-is-important-44ac0b0fea47

[4] scikit-learn, sklearn.preprocessing.RobustScaler,
https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html

[5] Sashimi Karanam, Curse of Dimensionality - A "Curse" to Machine Learning,
https://towardsdatascience.com/curse-of-dimensionality-a-curse-to-machine-learning-c122ee33bfeb

[6] Saniya Parveezm, Roberto Iriondo, Principal Component Analysis (PCA) with Python Examples – Tutorial,
https://pub.towardsai.net/principal-component-analysis-pca-with-python-examples-tutorial-67a917bae9aa

[7] Elitsa Kaloyanova, How to Combine PCA and K-means Clustering in Python?,
https://365datascience.com/tutorials/python-tutorials/pca-k-means/

[8] matplotlib, matplotlib.pyplot, https://matplotlib.org/3.5.3/api/_as_gen/matplotlib.pyplot.html

[9] seaborn, seaborn.pairplot, https://seaborn.pydata.org/generated/seaborn.pairplot.html