Assignment 2

Import packages for data manipulation and visualization.

Note: These packages are just for linearly algebra operations, data wrangling, and visualization. No ML methods are used.

In [393]:
```
import pandas as pd
import numpy as np
import pprint as pp
from matplotlib import pyplot as plt
```

Pull data from file

In [394]:
```
train_data = pd.read_csv('./data/body_measurements.csv')
train_data.loc[(train_data['Gender'] == 1.0), 'Gender']='M'
train_data.loc[(train_data['Gender'] == 2.0), 'Gender']='F'
```

Let's take a look at some traits of our data, just to get a feel for it

In [395]:
```
print(train_data.dtypes)
```

```
Gender                object
Age                    int64
HeadCircumference      int64
ShoulderWidth          int64
ChestWidth             int64
Belly                  int64
Waist                  int64
Hips                   int64
ArmLength              int64
ShoulderToWaist        int64
WaistToKnee            int64
LegLength              int64
TotalHeight            int64
dtype: object
```

We can see that we are working with numerical data, and two gender classes (male and female). I chose this data for this reason, as numerical data is ideal for PCA/LDA analysis.

Let's dive a little deeper and get some ranges and averages for the data.

In [396]:
```
uncat_averages = train_data.mean(0)
cat_data = train_data.groupby('Gender')
cat_averages = cat_data.mean()
cat_max = cat_data.max()
cat_min = cat_data.min()

print('Averages')
cat_averages
```

```
Averages

/var/folders/qz/cmcq5ghx3xsbqnsy89f5q8gw0000gn/T/ipykernel_16952/967840
528.py:1: FutureWarning: Dropping of nuisance columns in DataFrame redu
ctions (with 'numeric_only=None') is deprecated; in a future version th
is will raise TypeError.  Select only valid columns before calling the
reduction.
  uncat_averages = train_data.mean(0)
```

Out[396]:

Gender	Age	HeadCircumference	ShoulderWidth	ChestWidth	Belly	Waist	H
F	13.067901	20.632716	13.854938	14.240741	21.160494	19.354938	19.861
M	17.240409	20.526854	14.703325	14.851662	19.401535	19.179028	19.000

Although we see a lot of variation in the dataset

In [397]:
```
print('Max')
cat_max
```

```
Max
```

Out[397]:

Gender	Age	HeadCircumference	ShoulderWidth	ChestWidth	Belly	Waist	Hips	ArmLength	Sl
F	54	80	87	37	213	52	63	41	
M	68	29	28	38	47	91	46	66	

In [398]:
```
print('Min')
cat_min
```

```
Min
```

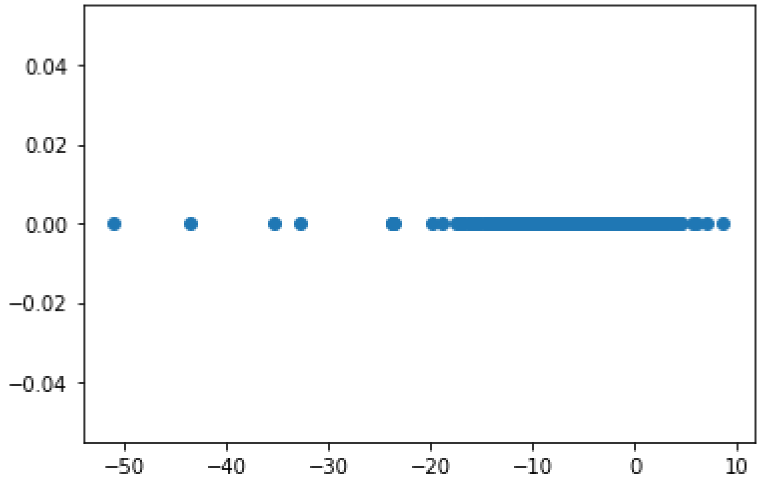
Out[398]:

Gender	Age	HeadCircumference	ShoulderWidth	ChestWidth	Belly	Waist	Hips	ArmLength	Sl
F	1	5	4	6	5	6	7	7	
M	1	9	5	6	6	2	7	6	

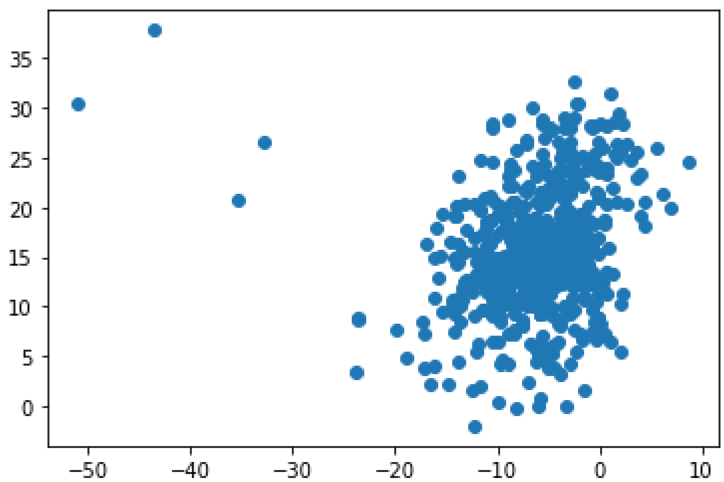
PCA

In [399]:
```
# Create a function that performs PCA on a given dataframe
def PCA(input_data, class_column, num_components, plot=True, test_data =
None):
    dropped_data = input_data.drop(class_column, axis=1)
    normalized_data = (dropped_data-dropped_data.min())/(dropped_data.ma
x()-dropped_data.min())
    transposed_data = np.transpose(normalized_data)
    cov_matrix = np.cov(transposed_data)
    w,v = np.linalg.eigh(cov_matrix)
    sort_indices = np.flip(np.argsort(w))
    eigenvectors = []
    eigenvalues = []
    for i in range(num_components):
        eigenvectors.append(v[sort_indices[i]])
        eigenvalues.append(w[sort_indices[i]])
    eigenvectors = np.array(eigenvectors).transpose()
    eigenvalues = np.abs(np.array(eigenvalues))
    train_projected = np.dot(dropped_data, eigenvectors)

    if test_data:
        test_projected = np.dot(test_data, eigenvectors)

    if plot and num_components == 1:
        fig, ax = plt.subplots()
        ax.scatter(train_projected[:,0], len(train_projected) * [0])
        if test_data:
            ax.scatter(test_projected[:, 0], len(test_projected) * [0],
marker='x')
        plt.show()
    if plot and num_components == 2:
        fig, ax = plt.subplots()
        ax.scatter(train_projected[:,0], train_projected[:, 1])
        if test_data:
            ax.scatter(train_projected[:,0], train_projected[:, 1], mark
er='x')
        plt.show()
    if plot and num_components == 3:
        fig = plt.figure()
        ax = plt.axes(projection='3d')
        ax.scatter3D(train_projected[:,0], train_projected[:, 1], train_
projected[:, 2])
        if test_data:
            ax.scatter3D(train_projected[:,0], train_projected[:, 1], tr
ain_projected[:, 2], marker='x')
        plt.show()
    print(f'{sum(eigenvalues) / sum(w) * 100}% of variance accounted for
with {num_components} PCs')
```

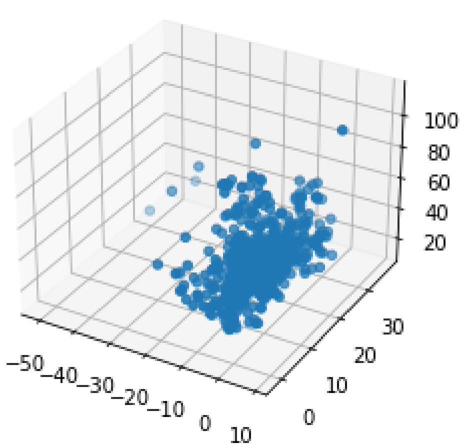
In [400]:
```
PCA(train_data, 'Gender', 1)
PCA(train_data, 'Gender', 2)
PCA(train_data, 'Gender', 3)
```




```
52.49432119115092% of variance accounted for with 1 PCs
```




```
63.903152634459616% of variance accounted for with 2 PCs
```




```
73.79452410656182% of variance accounted for with 3 PCs
```

Now we can see that PCA did seem to successfully find the axes with the highest variance, and that adding more components results in a higher percentage of the variance being accounted for. These results indicate that even with 3 PCs, there is still a large chunk of the variance that is unaccounted for, which means that the data that we are able to visualize isn't a great substitute for the original data.

LDA

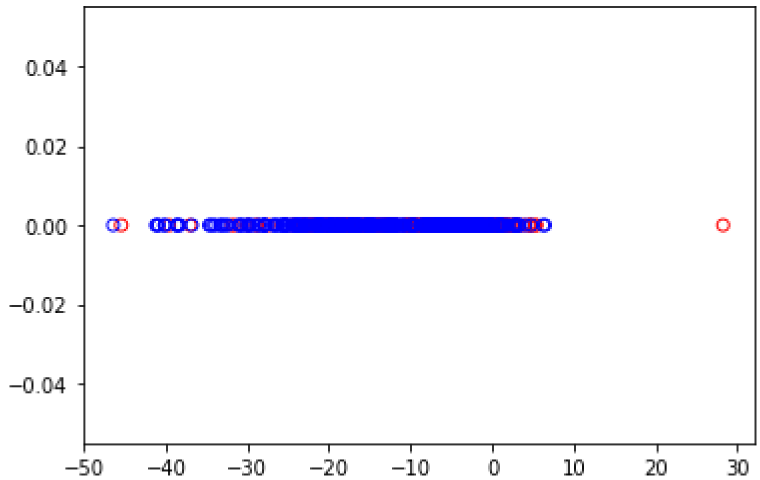
This [tutorial (https://www.youtube.com/watch?v=9lDXYHhAfGA&t=396s&ab_channel=PythonEngineer)](https://www.youtube.com/watch?v=9lDXYHhAfGA&t=396s&ab_channel=PythonEngineer) was a huge help to me for implementing the first section of the PCA algorithm in Python. The second half is extremely similar to the PCA algorithm.

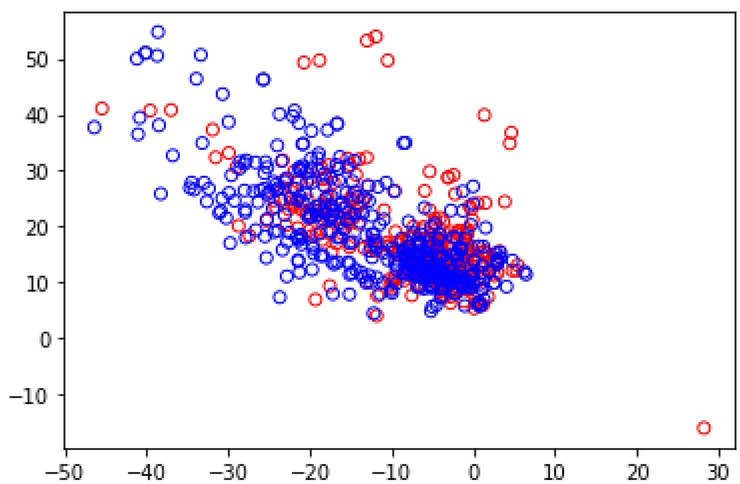
First we calculate the within class scatter matrix:

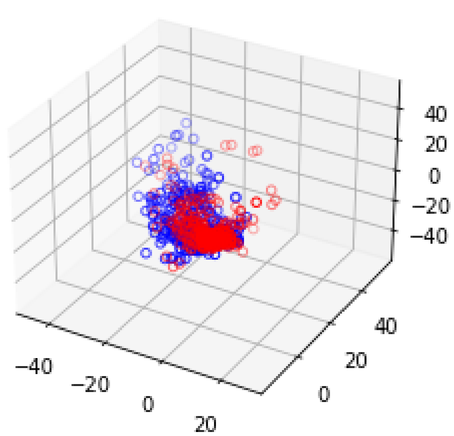
In [405]:
```
def LDA(input_data, class_column, num_components, plot=True, test_data=N
one):
    dropped_data = input_data.drop(class_column, axis=1).to_numpy()
    normalized_data = (dropped_data-dropped_data.min())/(dropped_data.ma
x()-dropped_data.min())
    n_features = normalized_data.shape[1]
    categorized_data = input_data.groupby(class_column)
    classes = list(categorized_data.indices.keys())
    SW = np.zeros((n_features, n_features))
    SB = np.zeros((n_features, n_features))
    all_mean = np.mean(normalized_data, axis=0)
    for c in classes:
        XC = normalized_data[input_data[class_column] == c]
        c_mean = XC.mean(axis=0)
        SW += np.cov(XC.T)
        c_samples = XC.shape[0]
        mean_diff = (c_mean - all_mean).reshape(n_features, 1)
        SB += c_samples * mean_diff @ mean_diff.T

    scatters = np.linalg.inv(SW) @ SB
    w,v = np.linalg.eigh(scatters)
    v = v.T
    sort_indices = np.flip(np.argsort(np.abs(w)))
    eigenvectors = []
    eigenvalues = []

    for i in range(num_components):
        eigenvectors.append(v[sort_indices[i]])
        eigenvalues.append(w[sort_indices[i]])
    eigenvectors = np.array(eigenvectors).T
    eigenvalues = np.abs(np.array(eigenvalues))
    train_projected = np.dot(dropped_data, eigenvectors)

    colors = ['r', 'b', 'k']

    if test_data:
        test_projected = np.dot(test_data, eigenvectors)

    if plot and num_components == 1:
        fig, ax = plt.subplots()
        for i, c in enumerate(classes):
            data = train_projected[input_data[class_column] == c]
            color = colors[i]
            ax.scatter(data[:,0], len(data) * [0], edgecolors=color, fac
ecolors='none')
            if test_data:
                t_data = test_projected.loc(test_projected[class_column]
== c)
                ax.scatter(t_data[:,0], len(data) * [0], edgecolors=colo
r, facecolors='none', marker='x')
        plt.show()
    if plot and num_components == 2:
        fig, ax = plt.subplots()
        for i, c in enumerate(classes):
            data = train_projected[input_data[class_column] == c]
```

```
            color = colors[i]
            ax.scatter(data[:,0], data[:, 1], edgecolors=color, facecolo
rs='none')
            if test_data:
                t_data = test_projected.loc(test_projected[class_column]
== c)
                ax.scatter(t_data[:,0], len(t_data) * [0], edgecolors=co
lor, facecolors='none', marker='x')
        plt.show()
    if plot and num_components == 3:
        fig = plt.figure()
        ax = plt.axes(projection='3d')
        for i, c in enumerate(classes):
            data = train_projected[input_data[class_column] == c]
            color = colors[i]
            ax.scatter3D(data[:,0], data[:, 1], data[:, 2], edgecolors=c
olor, facecolors='none')
            if test_data:
                t_data = test_projected.loc(test_projected[class_column]
== c)
                ax.scatter3D(t_data[:,0], t_data[:, 1], t_data[:, 2], ed
gecolors=color, facecolors='none', marker='x')
        plt.show()
```

In [406]:
```
LDA(train_data, 'Gender', 1)
LDA(train_data, 'Gender', 2)
LDA(train_data, 'Gender', 3)
```







Results

The results of the LDA analysis were pretty disappointing, and I am not sure if I have a typo or a logical error in the data (any feedback would be appreciated). There is a single eigenvalue that is an order of magnitude larger than the others, so this axis is immediately the primary. The others don't contribute to better separation of the data. This implies to me that I made a mistake, but after a few hours of combing code I haven't been able to clear the issue up.

The PCA results looked like they correctly depicted the PCs with the highest variance, so I believe that this implementation was correct

Conclusion

Although the LDA method did not yield quality results, the assignment did provide a solid understanding of PCA and LDA implentations. The lectures were great for understanding the source of the methods, and the assignment required a concrete implentation, research, and understanding.