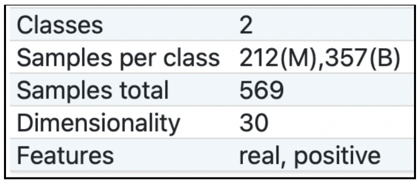
Data Analytics (IT350) - Lab 1

Annam Indhu Lekha
191IT207

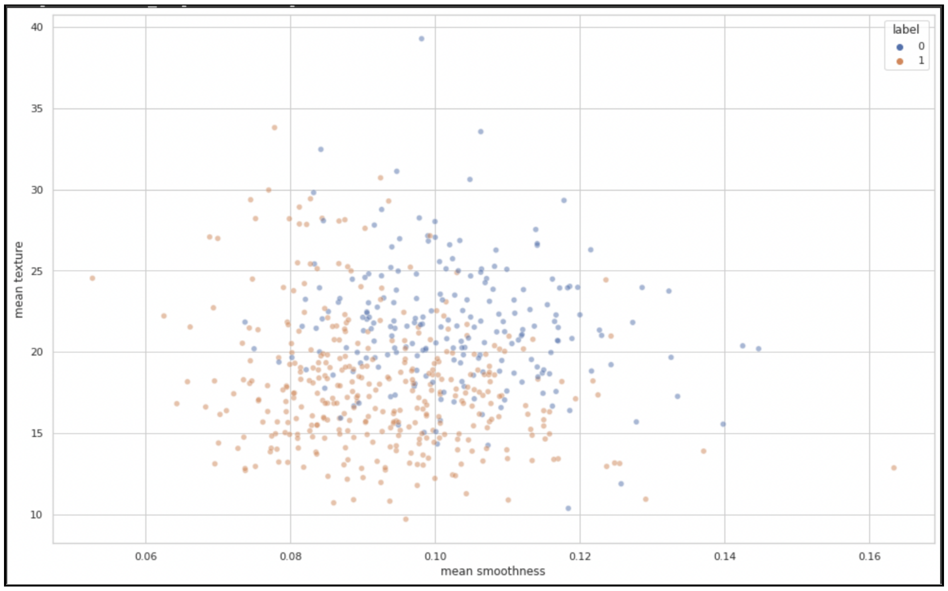
I have used the breast_caner dataset from sklearn.datasets. The data has 569 samples with thirty features, and each sample has a label associated with it (benign or malignant).

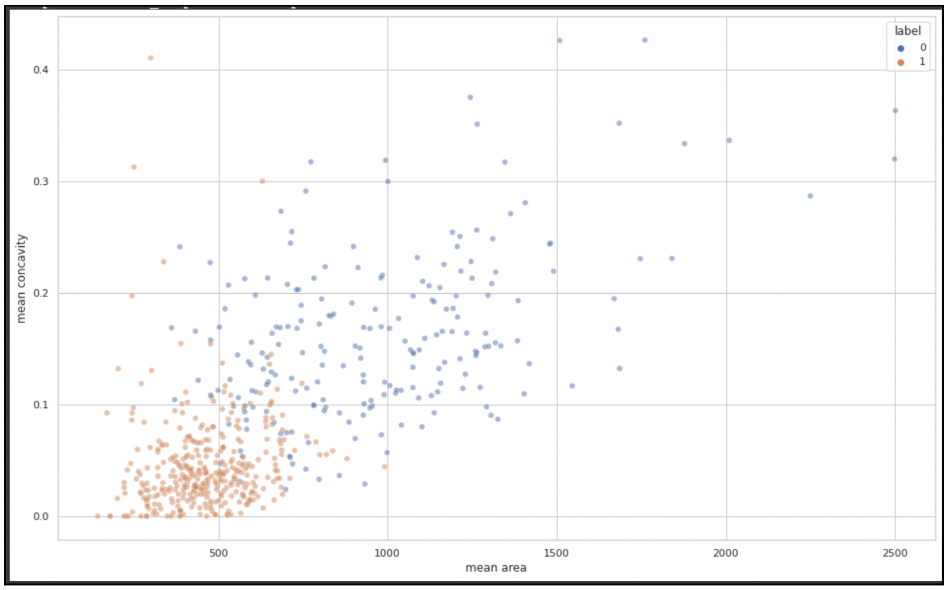


Classes	2
Samples per class	212(M),357(B)
Samples total	569
Dimensionality	30
Features	real, positive

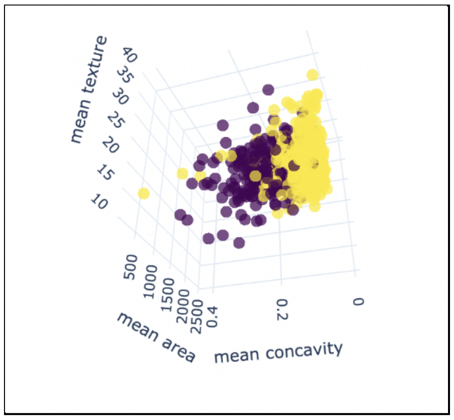
Visualisation before dimensionality reduction

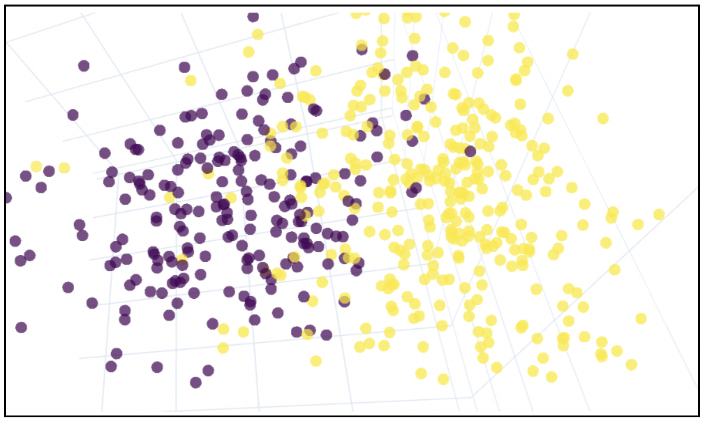
After a plotting using different possible combinations of features, the plots that I found the best separation b/w the classes are as follows:



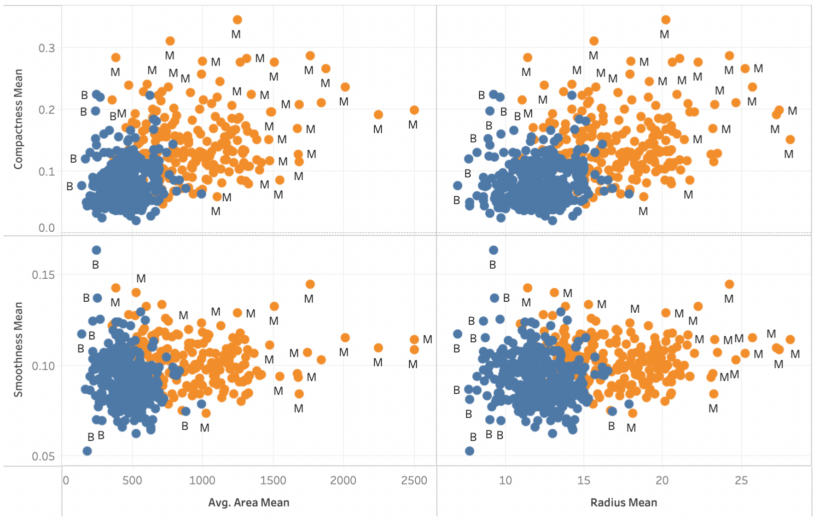


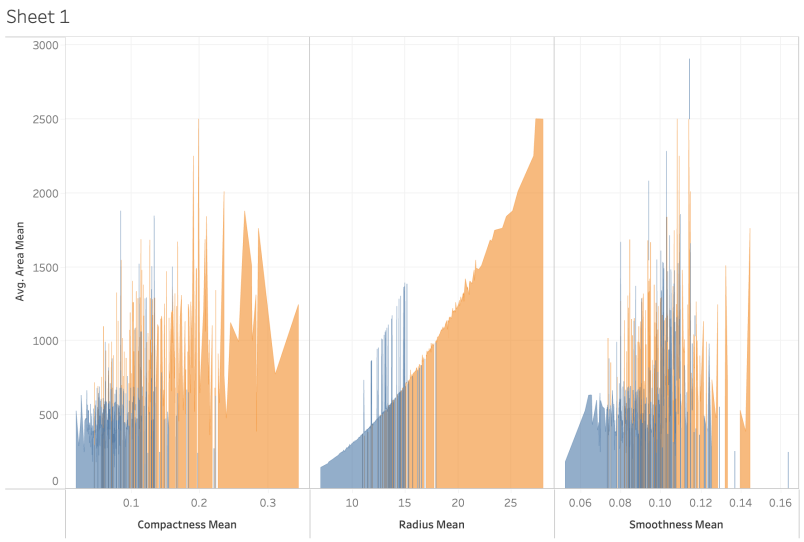
There seems to be a lot of mixup at the intersection of 2 classes in the 2d plots. Below are the results of 3d plots that best separate the classes:





Even though the 3d plot seemed to separate the classes, a zoomed in picture shows that there is no clear border that we could draw between the two classes.





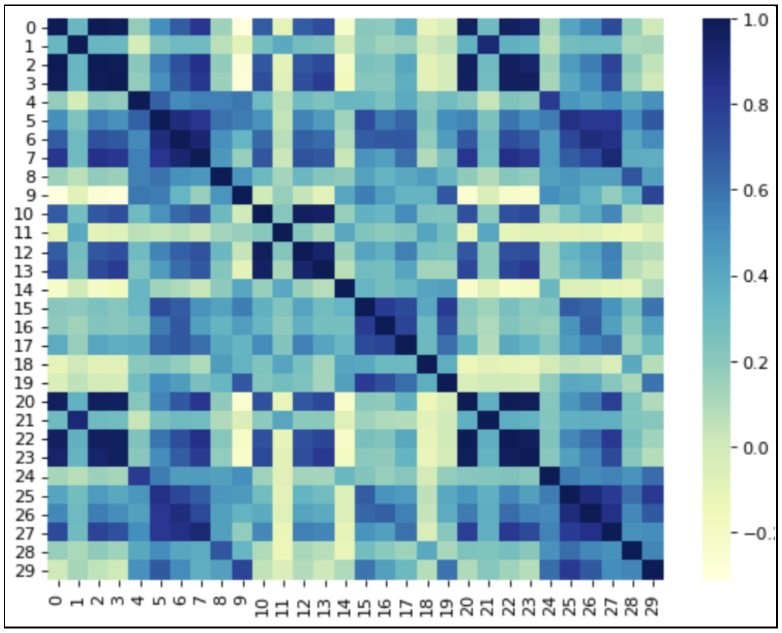
Even plotting using 3-4 features at a time doesn't give a good distinction between the 2 target classes.

This is because information for each data point is distributed among 30 features, but we have visualised only 2 or 3 features at time. **For the purpose of better visualisation, we have to reduce the dimensions of the input dataset.**

Dimensionality reduction can be done in 2 ways:

1. Principal component analysis (PCA)
2. Singular value decomposition (SVD)

Why PCA and SVD?
From the covariance matrix, we can see that there is a good amount of covariance between different features. This means that information is being duplicated. PCA and SVD reduces the initial number of dimensions to new dimensions which account for the maximum variance in the data.



Principal Component Analysis:

There are 5 steps :

1. Standardisation of the data
2. Computing covariance matrix
3. Calculating eigenvectors and eigenvalues
4. Computing Principal Components
5. Reducing Dimensions of Dataset

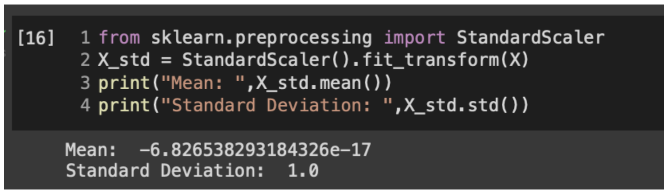
Singular Value Decomposition
There are 4 steps:

1. Standardisation of data
2. Calculating U, Sigma, and V matrices
3. Calculating the Variance and scree plot
4. Reducing dimensions of the Dataset

PRINCIPAL COMPONENT ANALYSIS

Standardisation of data

It is a common practice to normalise the data before feeding it to any machine learning algorithm.
To apply normalisation, I have imported the StandardScaler module from the sklearn library and then applied scaling by doing fit_transform on the feature data.
While applying StandardScaler, each feature of the data should be normally distributed such that it will scale the distribution to a **mean of zero** and a **standard deviation of one**.



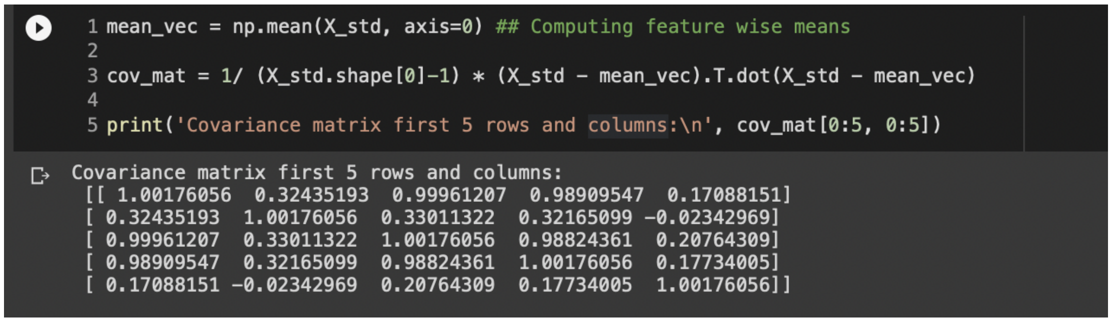
```
[16]
from sklearn.preprocessing import StandardScaler
X_std = StandardScaler().fit_transform(X)
print("Mean: ",X_std.mean())
print("Standard Deviation: ",X_std.std())
```

```
Mean:  -6.826538293184326e-17
Standard Deviation:  1.0
```

Computing the covariance matrix

Covariance is always measured between 2 dimensions. If we have a data set with more than 2 dimensions, there is more than one covariance measurement that can be calculated. For an n-dimensional data set, one can calculate $^{N}C_2$ different covariances.
Here there are 30 different features, so we will have to compute 435 different covariances.

$$covariance\ matrix\ =\ 1/(N\ -\ 1)\ \ *\ \Sigma\ (X_i - \overline{X})\ \ *\ \ (X\ -\ \overline{X})^T$$

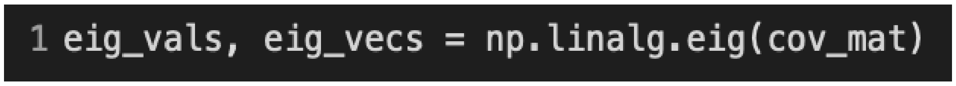


```
mean_vec = np.mean(X_std, axis=0) ## Computing feature wise means

cov_mat = 1/ (X_std.shape[0]-1) * (X_std - mean_vec).T.dot(X_std - mean_vec)

print('Covariance matrix first 5 rows and columns:\n', cov_mat[0:5, 0:5])
```

```
Covariance matrix first 5 rows and columns:
 [[ 1.00176056  0.32435193  0.99961207  0.98909547  0.17088151]
 [ 0.32435193  1.00176056  0.33011322  0.32165099 -0.02342969]
 [ 0.99961207  0.33011322  1.00176056  0.98824361  0.20764309]
 [ 0.98909547  0.32165099  0.98824361  1.00176056  0.17734005]
 [ 0.17088151 -0.02342969  0.20764309  0.17734005  1.00176056]]
```

Calculating the eigenvectors and eigenvalues

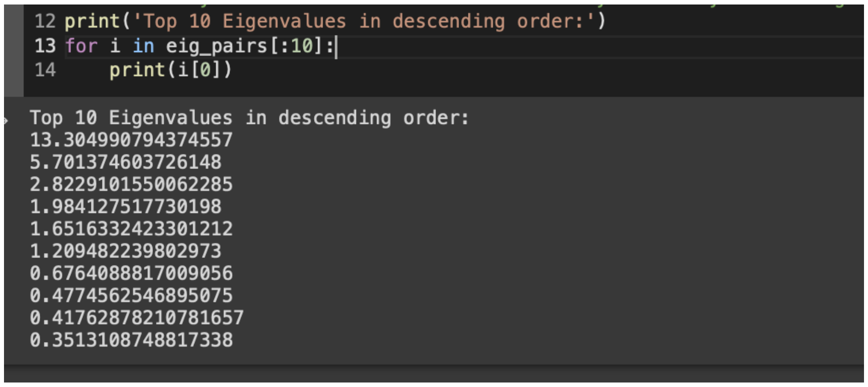
Since the covariance matrix is square, we can calculate the eigenvectors and eigenvalues for this matrix.



```
eig_vals, eig_vecs = np.linalg.eig(cov_mat)
```

Computing the Principal Components

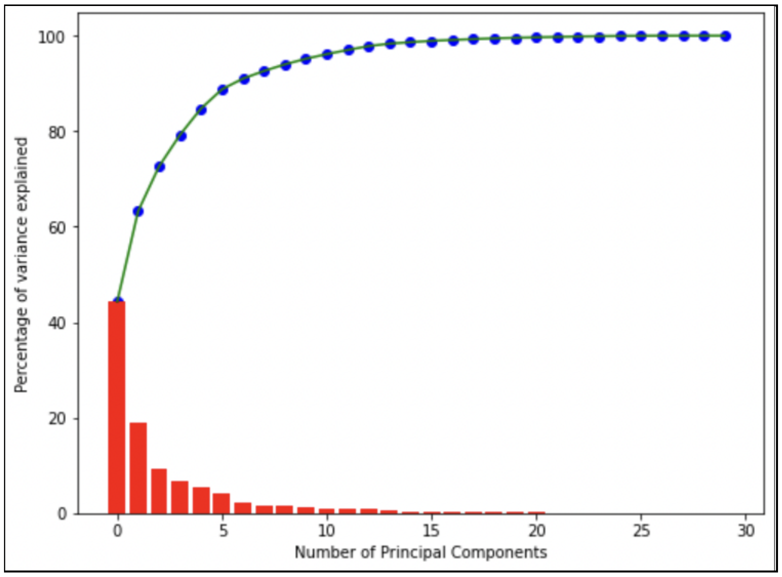
I made a list of (eigenvalue, eigenvector) tuples and sorted the (eigenvalue, eigenvector) tuples from high to low.



```
print('Top 10 Eigenvalues in descending order:')
for i in eig_pairs[:10]:
    print(i[0])
```

```
Top 10 Eigenvalues in descending order:
13.304990794374557
5.701374603726148
2.8229101550062285
1.984127517730198
1.6516332423301212
1.209482239802973
0.6764088817009056
0.4774562546895075
0.41762878210781657
0.3513108748817338
```

I have plotted the scree plot that explains the cumulative percentage of variance represented by x number of principal components.

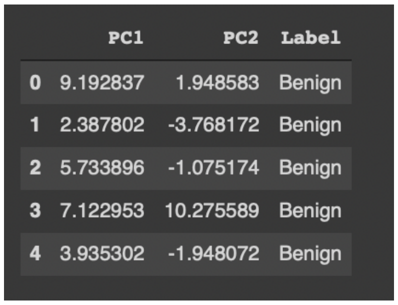


The first principal component alone explains more than 40% variance in the dataset. Considering all the 30 dimensions explains 100% of the variance in the dataset as expected.
The first 2 PCs explain nearly 65% variance, and the first 3 explain close to 75% variance.

I'll have reduced the dimensions of our dataset to the first 2 and 3 principal components

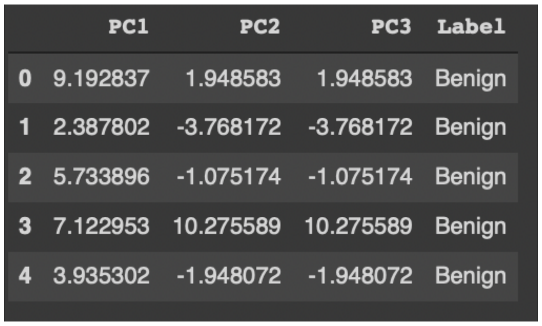
Reducing Dimensions of Dataset

Dataset with 2 features (obtained by choosing the first 2 principal components)



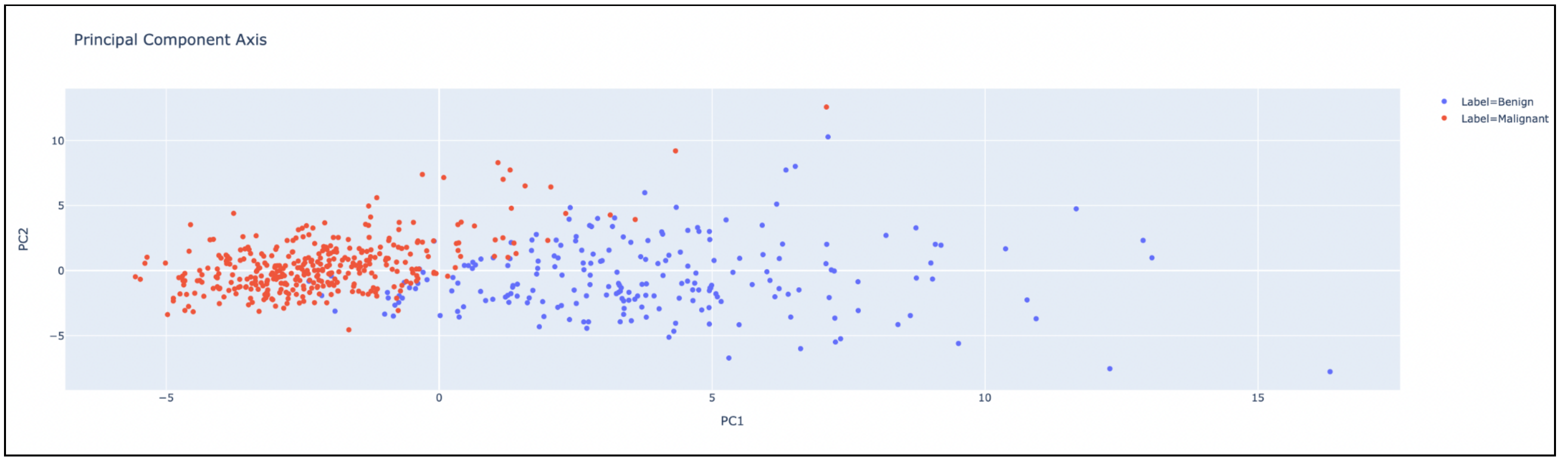
	PC1	PC2	Label
0	9.192837	1.948583	Benign
1	2.387802	-3.768172	Benign
2	5.733896	-1.075174	Benign
3	7.122953	10.275589	Benign
4	3.935302	-1.948072	Benign

Dataset with 3 features (obtained by choosing the first 3 principal components)

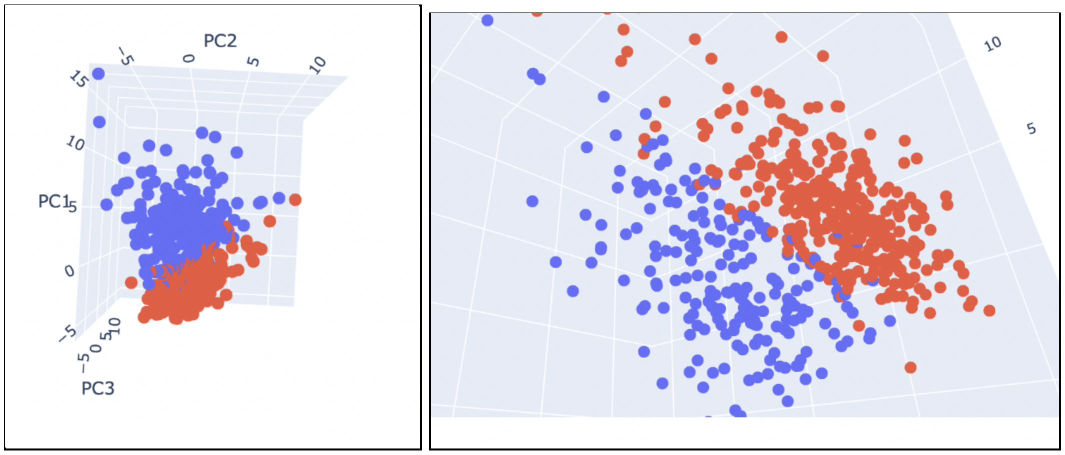


	PC1	PC2	PC3	Label
0	9.192837	1.948583	1.948583	Benign
1	2.387802	-3.768172	-3.768172	Benign
2	5.733896	-1.075174	-1.075174	Benign
3	7.122953	10.275589	10.275589	Benign
4	3.935302	-1.948072	-1.948072	Benign

Visualisation after reduction in dimensions



This 2d plot with two principal components helps us better distinguish between the two classes. **Using two primary components saved roughly 65% of the data variance.**

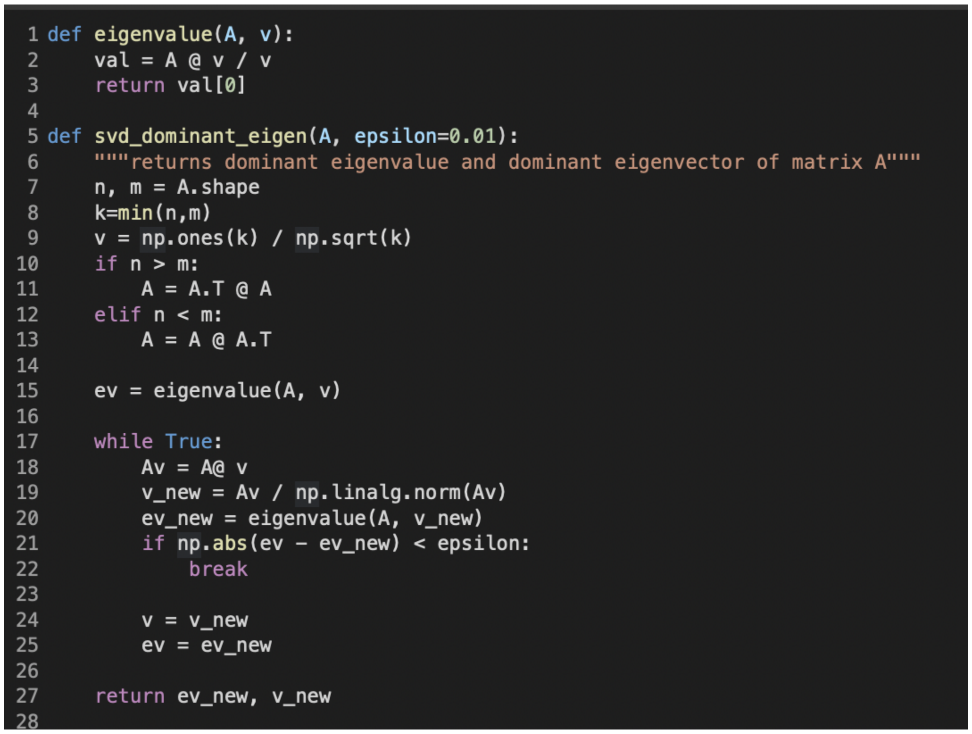


I plotted graphs using three main components. The division between the two classes is clearer than when we used two primary components. **Using 3 principal components saves nearly 75% of the variance of the data.**

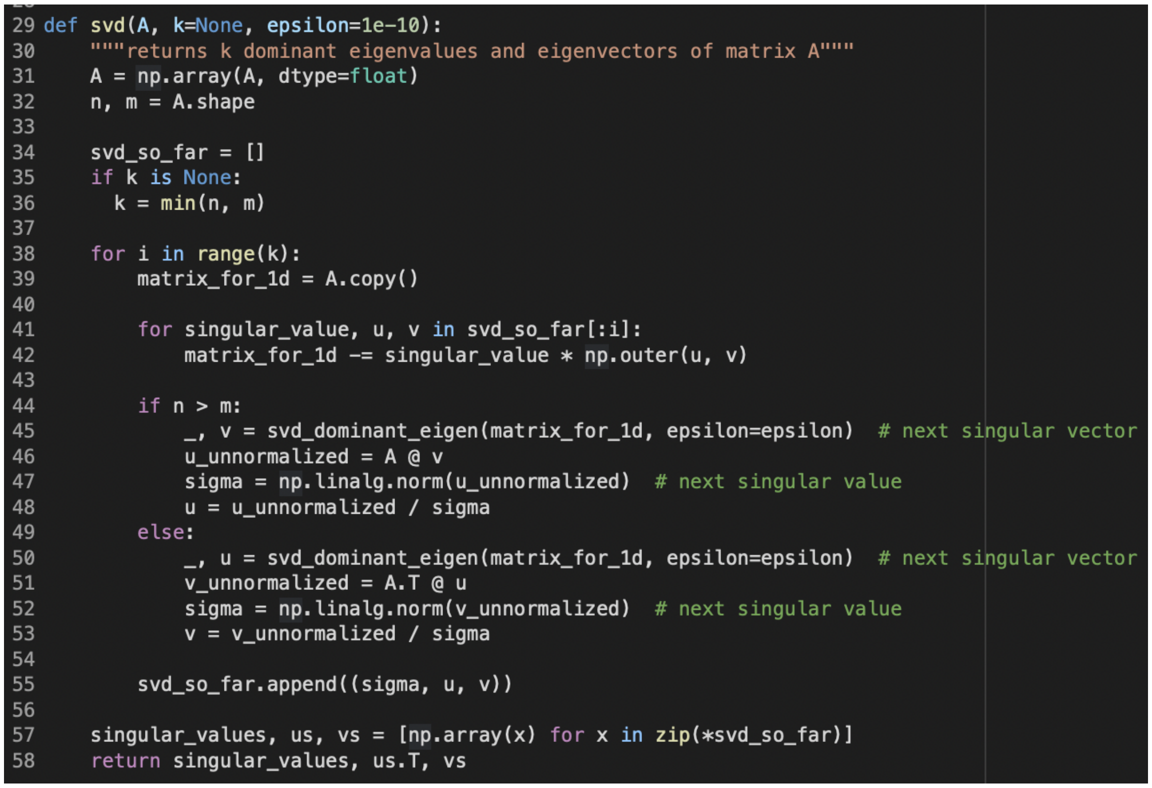
USING SINGULAR VALUE DECOMPOSITION

The first step of standardising the data is the same (using standardscalar form sklearn).

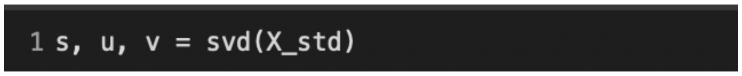
Calculating U, Sigma and V matrices



```python
def eigenvalue(A, v):
    val = A @ v / v
    return val[0]

def svd_dominant_eigen(A, epsilon=0.01):
    """returns dominant eigenvalue and dominant eigenvector of matrix A"""
    n, m = A.shape
    k=min(n,m)
    v = np.ones(k) / np.sqrt(k)
    if n > m:
        A = A.T @ A
    elif n < m:
        A = A @ A.T

    ev = eigenvalue(A, v)

    while True:
        Av = A@ v
        v_new = Av / np.linalg.norm(Av)
        ev_new = eigenvalue(A, v_new)
        if np.abs(ev - ev_new) < epsilon:
            break

        v = v_new
        ev = ev_new

    return ev_new, v_new

```



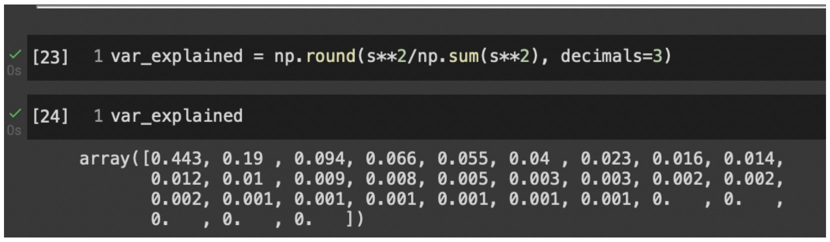
```python
def svd(A, k=None, epsilon=1e-10):
    """returns k dominant eigenvalues and eigenvectors of matrix A"""
    A = np.array(A, dtype=float)
    n, m = A.shape

    svd_so_far = []
    if k is None:
      k = min(n, m)

    for i in range(k):
        matrix_for_1d = A.copy()

        for singular_value, u, v in svd_so_far[:i]:
            matrix_for_1d -= singular_value * np.outer(u, v)

        if n > m:
            _, v = svd_dominant_eigen(matrix_for_1d, epsilon=epsilon)  # next singular vector
            u_unnormalized = A @ v
            sigma = np.linalg.norm(u_unnormalized)  # next singular value
            u = u_unnormalized / sigma
        else:
            _, u = svd_dominant_eigen(matrix_for_1d, epsilon=epsilon)  # next singular vector
            v_unnormalized = A.T @ u
            sigma = np.linalg.norm(v_unnormalized)  # next singular value
            v = v_unnormalized / sigma

        svd_so_far.append((sigma, u, v))

    singular_values, us, vs = [np.array(x) for x in zip(*svd_so_far)]
    return singular_values, us.T, vs
```



```python
s, u, v = svd(X_std)
```

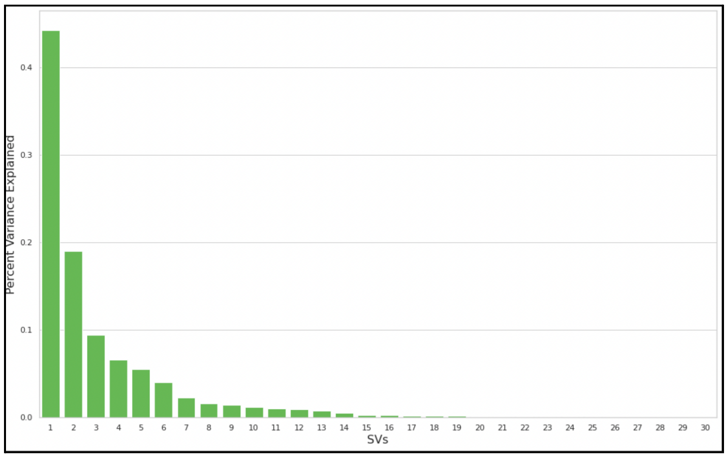
Calculating the Variance

$variance = sigma^2 / \Sigma\, sigma_i^2$



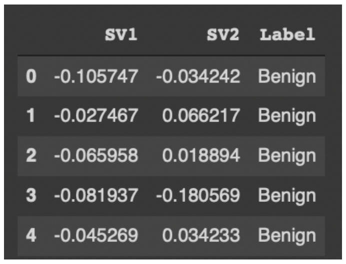
```
[23]  var_explained = np.round(s**2/np.sum(s**2), decimals=3)

[24]  var_explained

    array([0.443, 0.19 , 0.094, 0.066, 0.055, 0.04 , 0.023, 0.016, 0.014,
           0.012, 0.01 , 0.009, 0.008, 0.005, 0.003, 0.003, 0.002, 0.002,
           0.002, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.   , 0.   ,
           0.   , 0.   , 0.   ])
```

Scree plot that explains the cumulative percentage of variance represented by x number of dimensions.



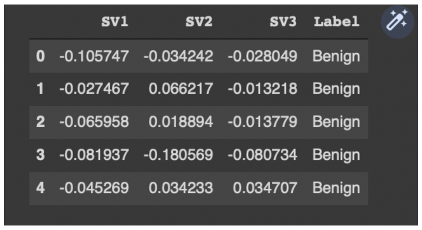
Using 2 dimensions we can preserve close to 63% of variance and using 3 dimensions we can preserve close to 73% of variance of data.

Dataset with 2 features (obtained by choosing the first 2 SVs)



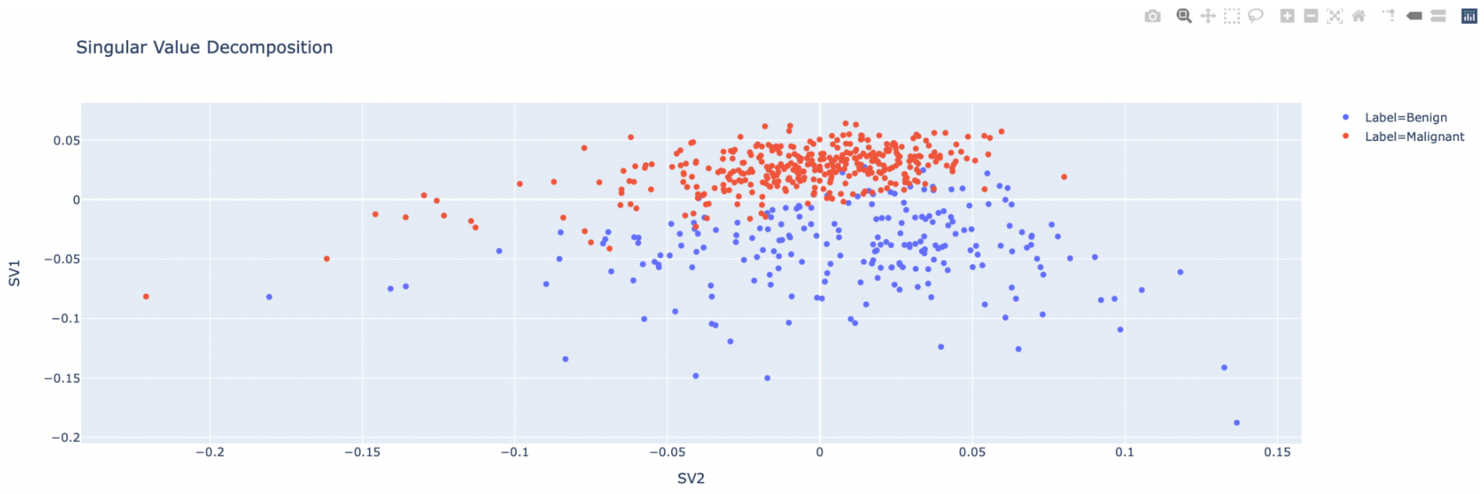
	SV1	SV2	Label
0	-0.105747	-0.034242	Benign
1	-0.027467	0.066217	Benign
2	-0.065958	0.018894	Benign
3	-0.081937	-0.180569	Benign
4	-0.045269	0.034233	Benign

Dataset with 3 features (obtained by choosing the first 3 SVs)

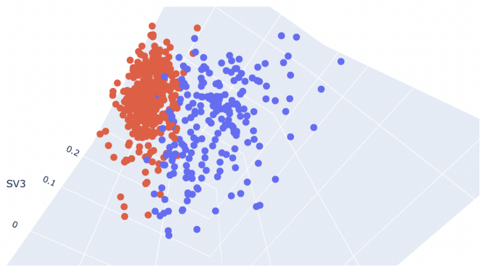


	SV1	SV2	SV3	Label
0	-0.105747	-0.034242	-0.028049	Benign
1	-0.027467	0.066217	-0.013218	Benign
2	-0.065958	0.018894	-0.013779	Benign
3	-0.081937	-0.180569	-0.080734	Benign
4	-0.045269	0.034233	0.034707	Benign

Visualisation after reduction in dimensions



This 2d plot with two principal components helps us better distinguish between the two classes.



3d plot using 3 SVs. We seem to get a better distinction between the 2 classes.

Conclusion

Despite the fact that SVD and PCA performed similarly in terms of variance preservation, I felt the distinction between different classes was more interpretable following PCA visualisation of the dataset. These dimensionality reduction techniques have helped in the following ways:

1. Reduced the size of dataset - reduction from 30 features to 2-4 features (depending on how much variance we want to preserve)
2. This further makes ML algorithms work more efficiently (reduction in training time due to less number of features)
3. Helps in visualising the dataset (we humans are limited to a maximum of three-dimensional visualisation)

Link to colab notebook: [191IT207-IT350-Lab1]