Exercise 1.1

In [ ]:
```
import pandas as pd
```

1. Data Preparation

In [ ]:
```
# load the dataset and display the first 5 rows
df = pd.read_csv('Files_For_A2/cancer_data.csv')
df.head()
```

Out[ ]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	sr
0	842302	M	17.99	10.38	122.80	1001.0	
1	842517	M	20.57	17.77	132.90	1326.0	
2	84300903	M	19.69	21.25	130.00	1203.0	
3	84348301	M	11.42	20.38	77.58	386.1	
4	84358402	M	20.29	14.34	135.10	1297.0	

5 rows × 32 columns

We first need to check for missing values and convert non-numeric to numeric

In [ ]:
```
# display the number of missing values for each column

missing_values = df.isnull().sum()
missing_values
```

Out[ ]:
```
id                         0
diagnosis                  0
radius_mean                0
texture_mean               0
perimeter_mean             0
area_mean                  0
smoothness_mean            0
compactness_mean           0
concavity_mean             0
concave_points_mean        0
symmetry_mean              0
fractal_dimension_mean     0
radius_se                  0
texture_se                 0
perimeter_se               0
area_se                    0
smoothness_se              0
compactness_se             0
concavity_se               0
concave_points_se          0
symmetry_se                0
fractal_dimension_se       0
radius_worst               0
texture_worst              0
perimeter_worst            0
area_worst                 0
smoothness_worst           0
compactness_worst          0
concavity_worst            0
concave_points_worst       0
symmetry_worst             0
fractal_dimension_worst    0
dtype: int64
```

In [ ]:
```
# display key information about the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   id                       569 non-null    int64  
 1   diagnosis                569 non-null    object 
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave_points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave_points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave_points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
dtypes: float64(30), int64(1), object(1)
memory usage: 142.4+ KB
```

We can see that there is only 1 categorical attribute, so we need to convert in to a numeric attribute and save it for later use. We will also remove the 'id' attribute as it would skew our results.

In [ ]:
```
# drop the id column
df.drop('id', axis=1, inplace=True)
```

In [ ]:
```
# display the counts of the categorical data
df['diagnosis'].value_counts()
```

Out[ ]:
```
diagnosis
B    357
M    212
Name: count, dtype: int64
```

In [ ]:
```
# convert the categorical data to numerical data
from sklearn.preprocessing import LabelEncoder

# initialize LabelEncoder
labelencoder = LabelEncoder()

# convert the categorical data to numerical data and display the first 5 row
df['diagnosis'] = labelencoder.fit_transform(df['diagnosis'])
diagnosis = df['diagnosis']
df.drop('diagnosis', axis=1, inplace=True)
df.head()
```

Out[ ]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	comp
0	17.99	10.38	122.80	1001.0	0.11840	
1	20.57	17.77	132.90	1326.0	0.08474	
2	19.69	21.25	130.00	1203.0	0.10960	
3	11.42	20.38	77.58	386.1	0.14250	
4	20.29	14.34	135.10	1297.0	0.10030	

5 rows × 30 columns

Now we have:

- B == 0
- M == 1

Now we can scale the data using the z-score method

In [ ]:
```
normalized_df = (df - df.mean()) / df.std()
normalized_df
```

Out[ ]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	cc
0	1.096100	-2.071512	1.268817	0.983510	1.567087	
1	1.828212	-0.353322	1.684473	1.907030	-0.826235	
2	1.578499	0.455786	1.565126	1.557513	0.941382	
3	-0.768233	0.253509	-0.592166	-0.763792	3.280667	
4	1.748758	-1.150804	1.775011	1.824624	0.280125	
...	...	...	...	...	...	
564	2.109139	0.720838	2.058974	2.341795	1.040926	
565	1.703356	2.083301	1.614511	1.722326	0.102368	
566	0.701667	2.043775	0.672084	0.577445	-0.839745	
567	1.836725	2.334403	1.980781	1.733693	1.524426	
568	-1.806811	1.220718	-1.812793	-1.346604	-3.109349	

569 rows × 30 columns

We can see that the data is normalized by checking if the mean and standard deviation are 0, and 1 respectively

In [ ]:
```
normalized_df.std().mean(), round(normalized_df.mean().mean())
```

Out[ ]:
```
(1.0, 0)
```

2. PCA Application

Here we will use the sklearn PCA class to perform the PCA

In [ ]:
```
from sklearn.decomposition import PCA
num_components = 10
pca = PCA(n_components=num_components)
pca.fit(normalized_df)

principalComponents = pca.fit_transform(normalized_df)
pca_df = pd.DataFrame(data=principalComponents, columns=[f"PC{i+1}" for i in ir
pca_df
```

Out[ ]:

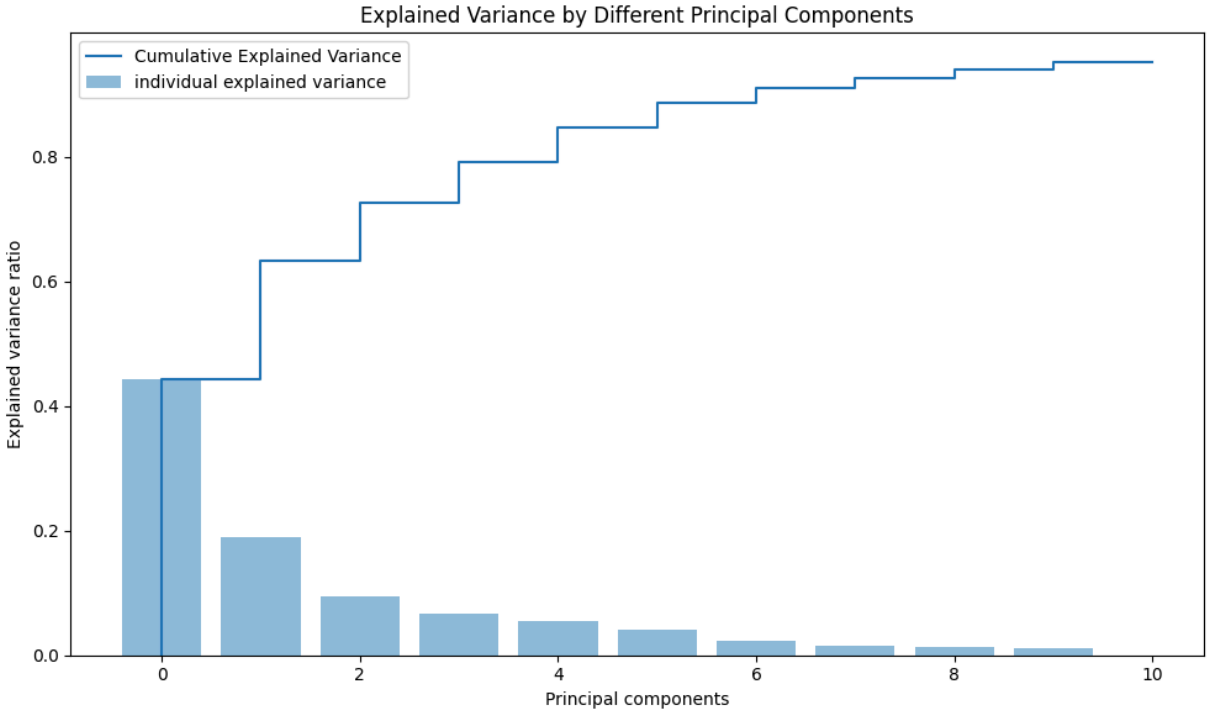
	PC1	PC2	PC3	PC4	PC5	PC6	PC7
0	9.184755	1.946870	-1.122179	3.630536	-1.194059	1.410184	2.157471
1	2.385703	-3.764859	-0.528827	1.117281	0.621228	0.028631	0.013347
2	5.728855	-1.074229	-0.551263	0.911281	-0.176930	0.540976	-0.667580
3	7.116691	10.266556	-3.229948	0.152413	-2.958275	3.050738	1.428653
4	3.931842	-1.946359	1.388545	2.938054	0.546267	-1.225416	-0.935389
...	...	...	...	...	...	...	...
564	6.433655	-3.573673	2.457324	1.176279	-0.074759	-2.373105	-0.595606
565	3.790048	-3.580897	2.086640	-2.503825	-0.510274	-0.246493	-0.715697
566	1.255075	-1.900624	0.562236	-2.087390	1.808400	-0.533977	-0.192589
567	10.365673	1.670540	-1.875379	-2.353960	-0.033712	0.567437	0.222885
568	-5.470430	-0.670047	1.489133	-2.297136	-0.184541	1.616415	1.697457

569 rows × 10 columns

3. Variance Analysis

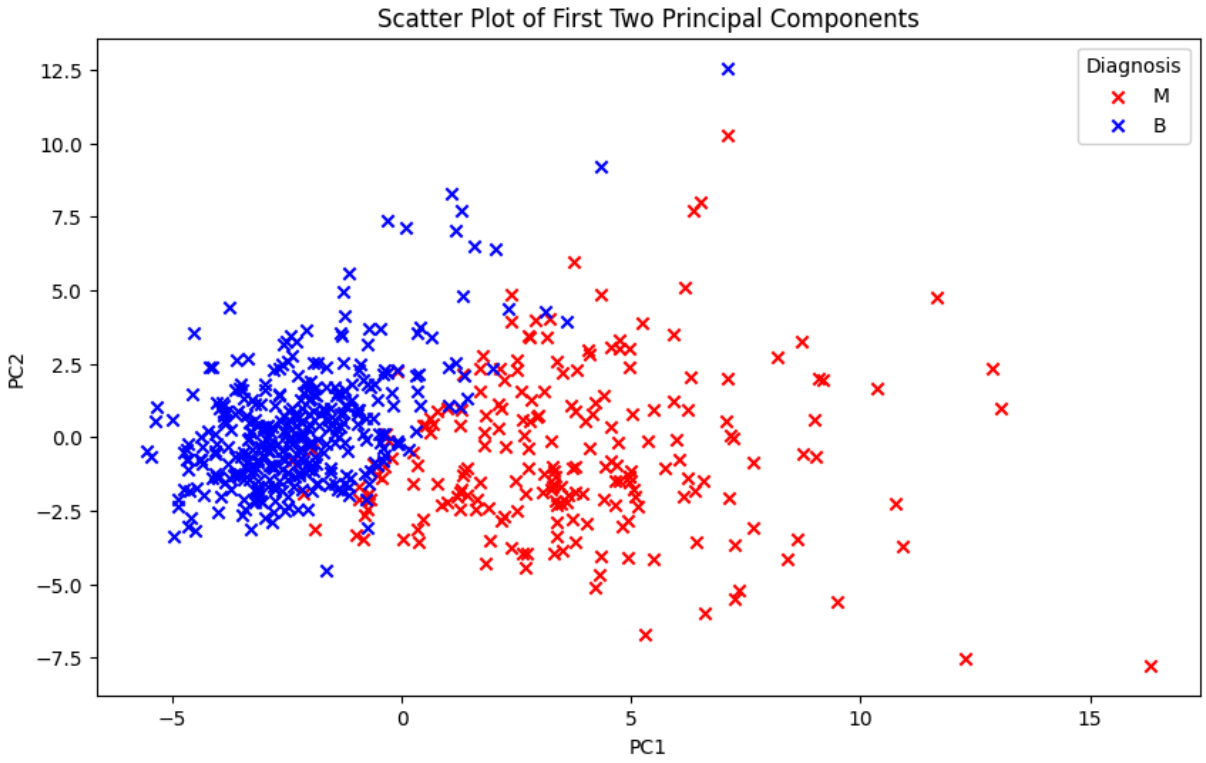
In [ ]:

```
import matplotlib.pyplot as plt
import numpy as np

explained_variance = pca.explained_variance_ratio_

cumulative_explained_variance = np.cumsum(explained_variance)

# plot the explained variance and the cumulative explained variance
plt.figure(figsize=(10, 6))
plt.title('Explained Variance by Different Principal Components')
plt.plot(range(len(explained_variance) + 1), [0] + list(cumulative_explained
plt.bar(range(len(explained_variance)), explained_variance, alpha=0.5, align
plt.ylabel('Explained variance ratio')
plt.xlabel('Principal components')
plt.legend(loc='best')
plt.tight_layout()
plt.show()
```

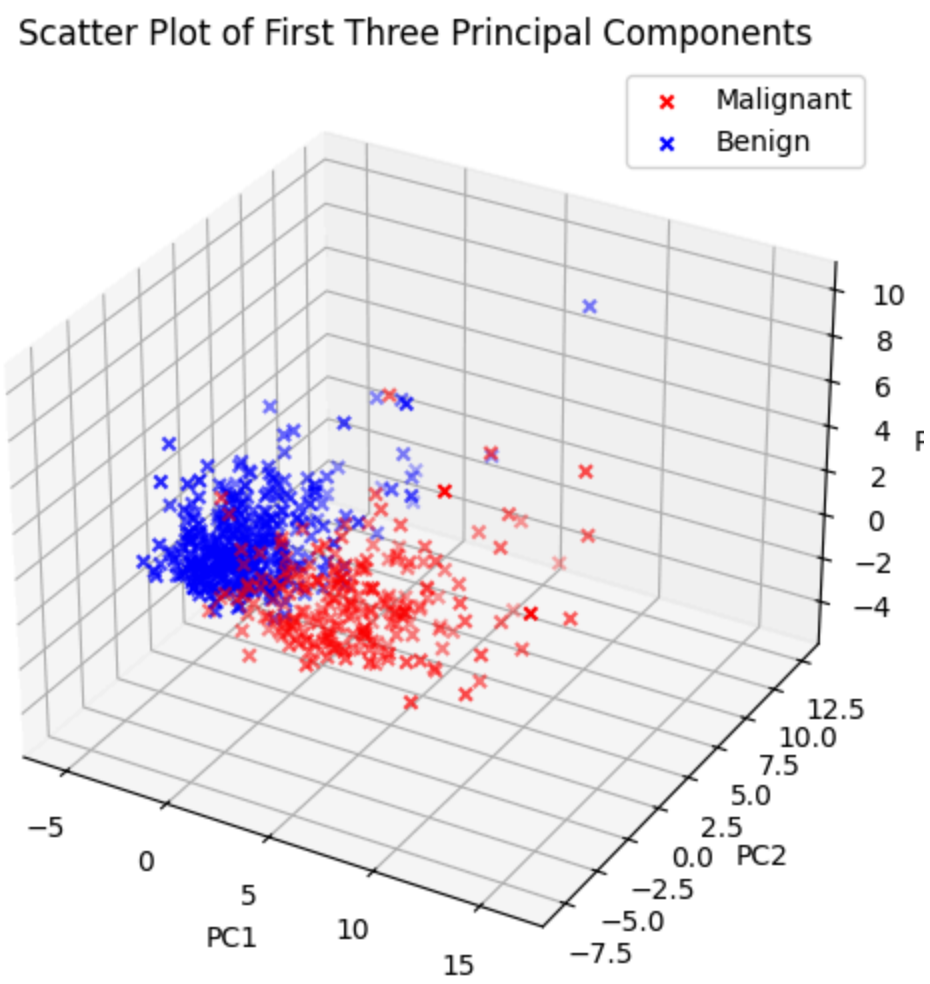


4. Visualization

In [ ]:
```
pca_df['diagnosis'] = diagnosis

# separate the data into two categories
category_M = pca_df[pca_df['diagnosis'] == 1]
category_B = pca_df[pca_df['diagnosis'] == 0]

# plot the first two principal components
plt.figure(figsize=(10, 6))

plt.scatter(category_M['PC1'], category_M['PC2'], c='red', label='M', marker
plt.scatter(category_B['PC1'], category_B['PC2'], c='blue', label='B', marke

# add title and labels
plt.title('Scatter Plot of First Two Principal Components')
plt.xlabel('PC1')
plt.ylabel('PC2')

# add legend
plt.legend(title='Diagnosis')

# display the plot
plt.show()
```



In [ ]:
```
# create a 3D scatter subplot
fig = plt.figure(figsize=(10, 6))
ax = fig.add_subplot(111, projection='3d')

# plot the first three principal components
ax.scatter(category_M['PC1'], category_M['PC2'], category_M['PC3'], c='red',
ax.scatter(category_B['PC1'], category_B['PC2'], category_B['PC3'], c='blue'

# add title and labels
ax.set_xlabel('PC1')
ax.set_ylabel('PC2')
ax.set_zlabel('PC3')

ax.set_title('Scatter Plot of First Three Principal Components')

ax.legend()

plt.show()
```



5. Interpretation

Based on the visualizations, it does appear that a predictive model could be developed to distinguish between malignant and benign tumors with a resonable degree of accuracy.

The 3D scatter plot shows a clear distintion between malignent and benign tumors, this suggests that the principle components have captured significant features which differenciate the 2 types of tumors. We can also observe that both tumors form distinct clusters, which indicates that there is a pattern a predictive model could learn from.

Overall, since there is a clear distinction in the data and we are using PCA which implies these components retain most of the variance in the dataset, we can say that a predictive model should perform well.