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

In [ ]:
```python
# 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 ir
pca_df
```

Out[ ]:

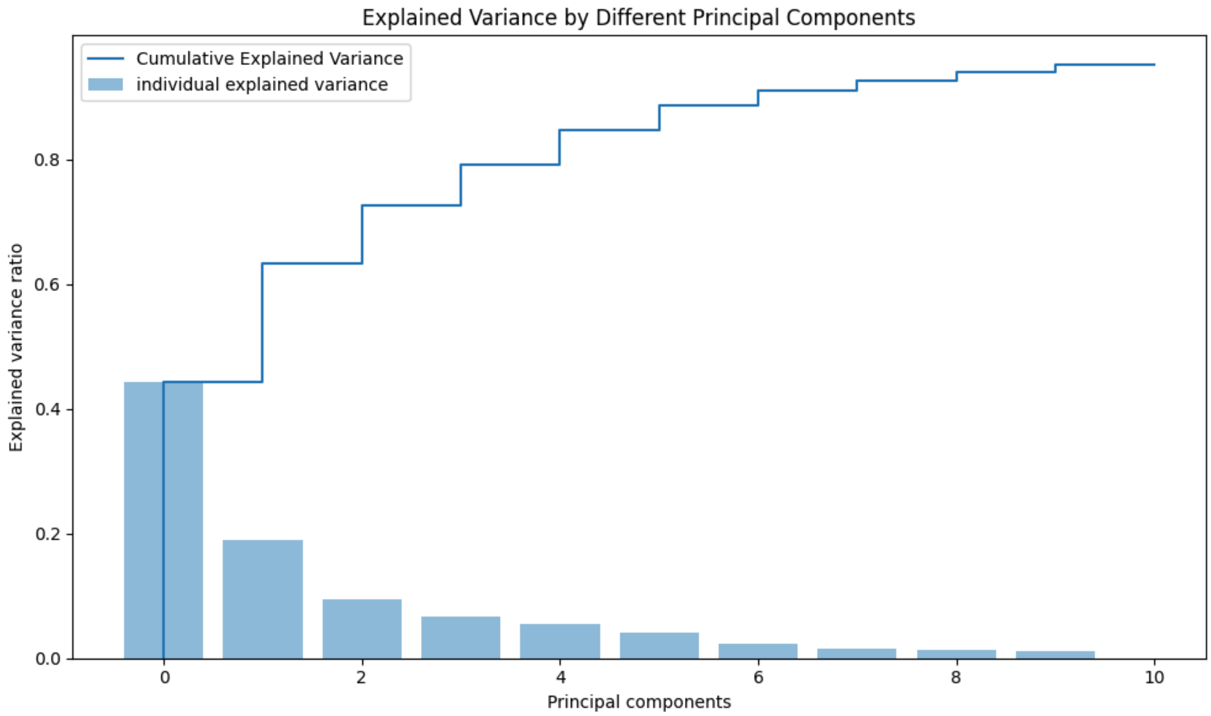
	PC1	PC2	PC3	PC4	PC5	PC6	PC7
0	9.184755	1.946870	-1.122179	3.630536	-1.194059	1.410184	2.157471
1	2.385703	-3.764859	-0.528827	1.117281	0.621228	0.028631	0.013347
2	5.728855	-1.074229	-0.551263	0.911281	-0.176930	0.540976	-0.667580
3	7.116691	10.266556	-3.229948	0.152413	-2.958275	3.050738	1.428653
4	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

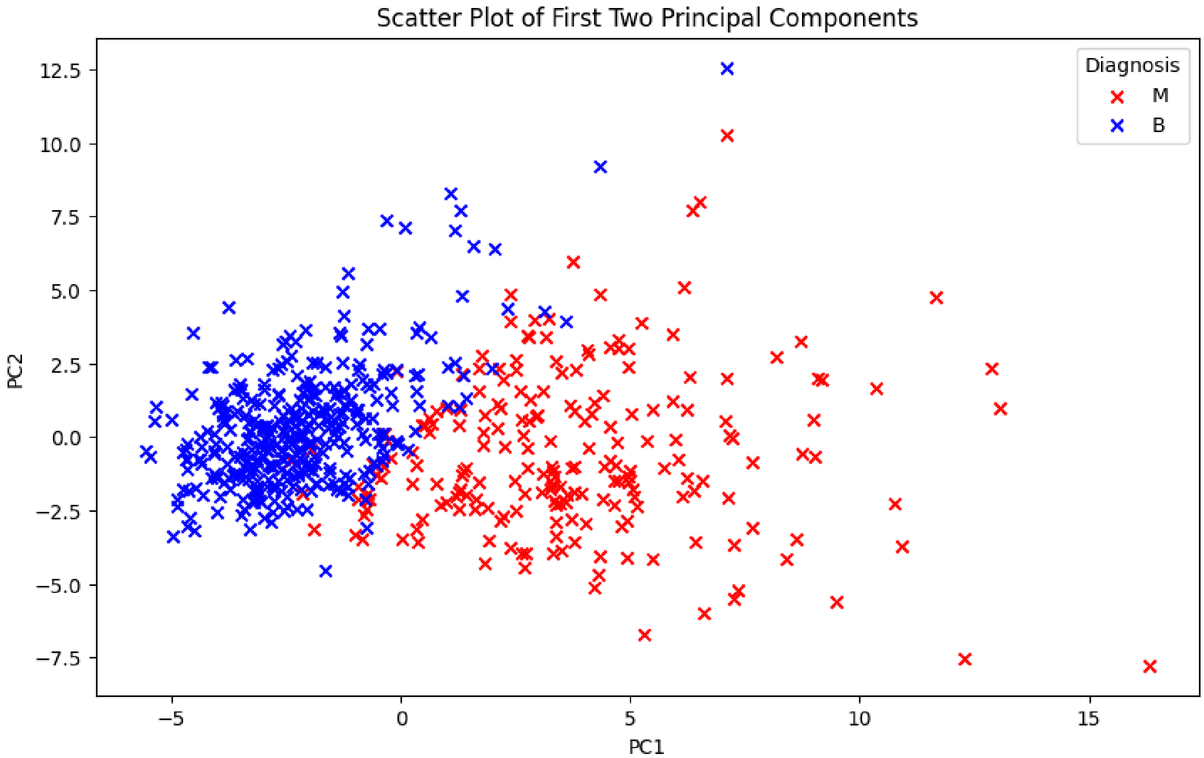
In [ ]:

```
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_explainec
plt.bar(range(len(explained_variance)), explained_variance, alpha=0.5, aligr
plt.ylabel('Explained variance ratio')
plt.xlabel('Principal components')
plt.legend(loc='best')
plt.tight_layout()
plt.show()
```

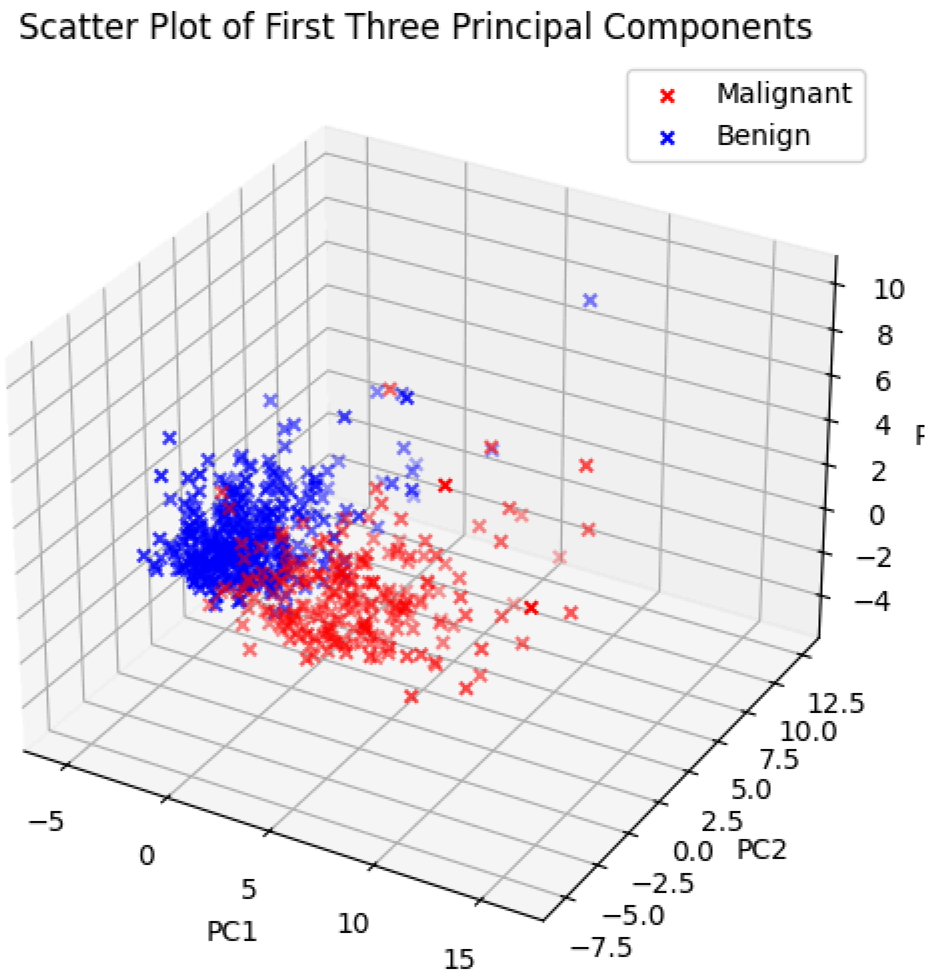


4. Visualization

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.

Exercise 1.2

1. Model Construction:

In [ ]:
```
import pandas as pd

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

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

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'])
df
```

Out[ ]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothnes
0	1	17.99	10.38	122.80	1001.0	
1	1	20.57	17.77	132.90	1326.0	
2	1	19.69	21.25	130.00	1203.0	
3	1	11.42	20.38	77.58	386.1	
4	1	20.29	14.34	135.10	1297.0	
...	...	...	...	...	...	
564	1	21.56	22.39	142.00	1479.0	
565	1	20.13	28.25	131.20	1261.0	
566	1	16.60	28.08	108.30	858.1	(
567	1	20.60	29.33	140.10	1265.0	
568	0	7.76	24.54	47.92	181.0	(

569 rows × 31 columns

In [ ]:

```
from sklearn.tree import DecisionTreeClassifier

# split the data into features and target
X = df.drop('diagnosis', axis=1)
y = df['diagnosis']

# initialize the DecisionTreeClassifier
classifier = DecisionTreeClassifier()
classifier.fit(X, y)
```

Out[ ]:

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [ ]:

```
feature_importances = pd.DataFrame(classifier.feature_importances_, index=cl
feature_importances['cumsum'] = feature_importances['importance'].cumsum()
feature_importances.head(10)
```

Out[ ]:

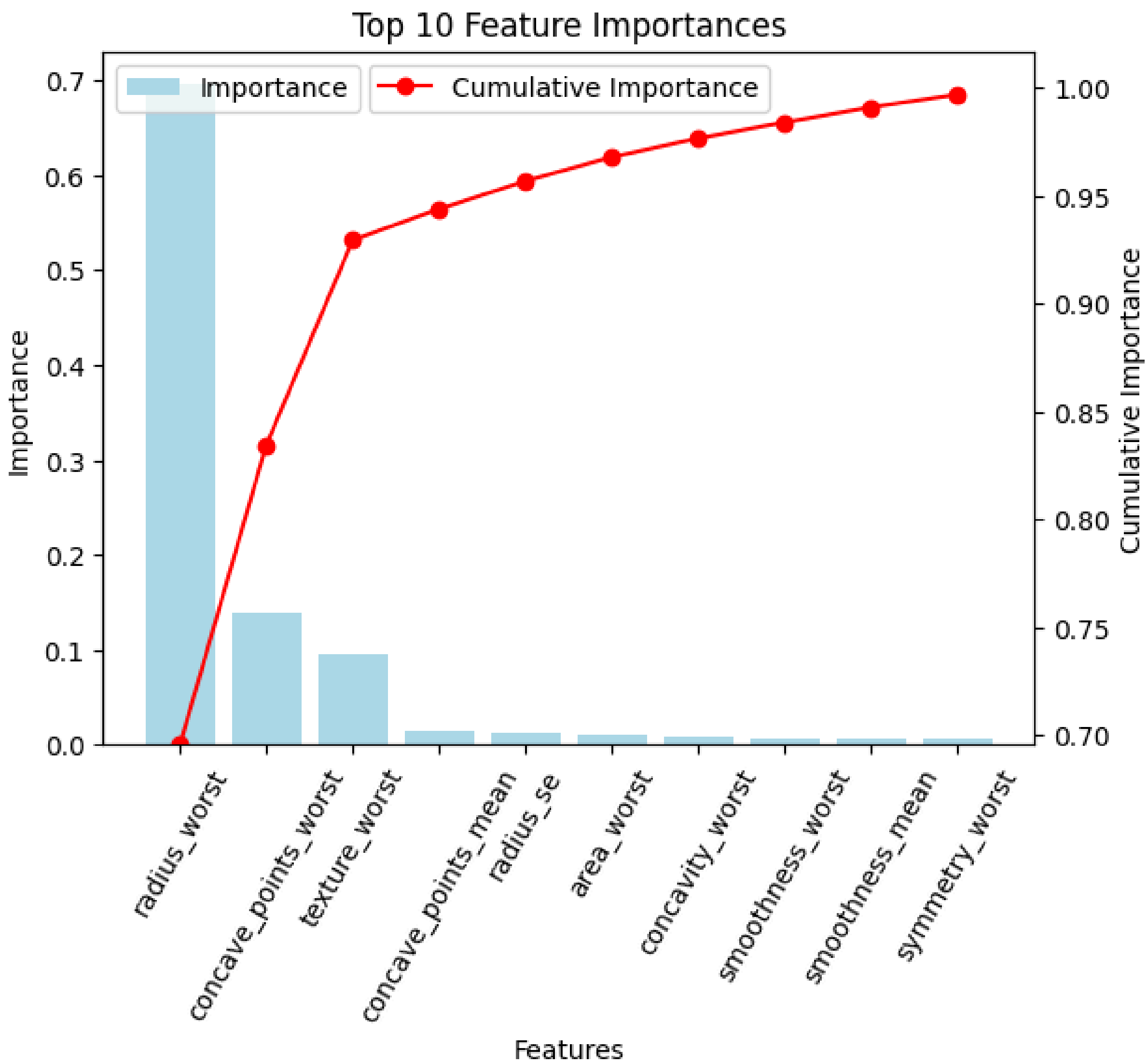
	importance	cumsum
radius_worst	0.695594	0.695594
concave_points_worst	0.138938	0.834532
texture_worst	0.095005	0.929537
concave_points_mean	0.014410	0.943947
radius_se	0.012955	0.956901
area_worst	0.011086	0.967987
concavity_worst	0.008727	0.976715
smoothness_worst	0.007388	0.984103
smoothness_mean	0.007017	0.991120
symmetry_worst	0.005831	0.996951

In [ ]:

```
import matplotlib.pyplot as plt

# Create a figure and axis objects
fig, ax1 = plt.subplots()

# Plot the bar chart on the first y-axis
ax1.bar(feature_importances.index[:10], feature_importances['importance'][:1

# Set the x-axis label
ax1.set_xlabel('Features')

# Set the first y-axis label
ax1.set_ylabel('Importance')

# Set the title
ax1.set_title('Top 10 Feature Importances')

ax1.set_xticklabels(feature_importances.index[:10], rotation=60)

# Create a second y-axis
ax2 = ax1.twinx()

# Plot the line graph on the second y-axis
ax2.plot(feature_importances.index[:10], feature_importances['cumsum'][:10],

# Set the second y-axis label
ax2.set_ylabel('Cumulative Importance')

# Set the limits of the second y-axis based on the minimum and maximum value
ax2.set_ylim(feature_importances['cumsum'][:10].min(), feature_importances['

# Add a legend
ax1.legend(loc='upper left')
ax2.legend(loc='upper center')
```

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

```
/var/folders/2v/mcgfxq4d2_n2639c1xhyd92w0000gn/T/ipykernel_35793/1564416163.
py:18: UserWarning: set_ticklabels() should only be used with a fixed number
of ticks, i.e. after set_ticks() or using a FixedLocator.
  ax1.set_xticklabels(feature_importances.index[:10], rotation=60)
```



4. Analysis

From the Top 10 Feature Importances graph it is clear that the radius_worst feature has the most significant influence of the models predictions. After that, the importance scores decrease significantly, with concave_points_worst and texture_worst being the next 2 most important features. The cumulative importance curve shows us that adding more features beyond the top 1 - 3 will provide diminishing returns in terms of the models performance.

To develop more accurate models using these insights, we can focus on the most important features and ensure they are accurate and well-preprocessed because errors in these feautures will have a greater impact on model performance. Also, since there is a steep dropoff in feature importance we can use dimensionality reduction techniques such as PCA without losing significant reliability.

Overall, it is clear that the insights from the Feature Importance graph suggest that there is a significant drop in the influence of features beyond 1-3. This would have a significant impact on the development of more accurate predictive models because we can allocate resources and focus on optimizing the most predictive features.

Assignment 2

Brady Mitchelmore - 202112249

Question 2.1:

`min_sup = 0.60`

`min_conf = 0.80`

TID	items_purchased
T1	{A, B, C, D, E, F}
T2	{G, B, C, D, E, F}
T3	{A, H, D, E}
T4	{A, I, J, D, F}
T5	{J, B, B, D, K, E}

Exercise 2.1.1:

Apriori:

First we scan the table for count of each candidate

Itemset	Support Count	Support
{A}	3	0.60
{B}	4	0.80
{C}	2	0.40
{D}	5	1.00
{E}	4	0.80
{F}	3	0.60
{G}	1	0.20
{H}	1	0.20
{I}	1	0.20
{J}	2	0.40

Next compare relative candidate support with the minimum support of 0.60. Here only 5 candidates in the first table satisfy the minimum support

Itemset	Support Count	Support
{A}	3	0.60
{B}	4	0.80
{D}	5	1.00
{E}	4	0.80
{F}	3	0.60

Itemset	Support Count	Support
{K}	1	0.20

Next we generate frequent 2-itemsets from the table above

Itemset	Support Count	Support
{A, B}	1	0.20
{A, D}	3	0.60
{A, E}	1	0.20
{A, F}	2	0.40
{B, D}	3	0.60
{B, E}	3	0.60
{B, F}	2	0.40
{D, E}	4	0.80
{D, F}	3	0.60
{E, F}	2	0.40

Next we compare relative candidate support with the minimum support of 0.60. Here only 5 candidates in the table above satisfy the minimum support

Itemset	Support Count	Support
{A, D}	3	0.60
{B, D}	3	0.60
{B, E}	3	0.60
{D, E}	4	0.80
{D, F}	3	0.60

Next we generate frequent 3-itemsets from the table above.

Itemset	Support Count	Support
{B, D, E}	3	0.60
{D, E, F}	2	0.40

Next we compare relative candidate support with the minimum support of 0.60. Here only 1 candidate in the table above satisfy the minimum support

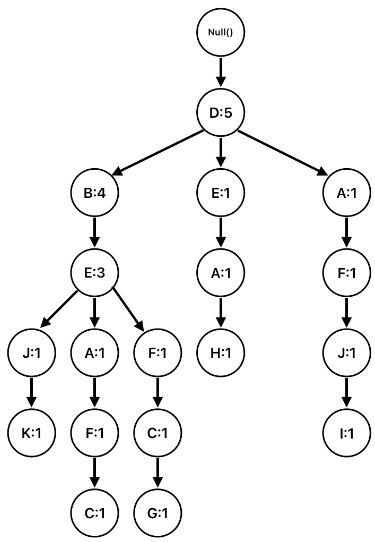
Itemset	Support Count	Support
{B, D, E}	3	0.60

No frequent 4-itemsets can be generated so we stop here with a frequent itemset of **{B, D, E}** which has a support count of 6

FPGrowth:

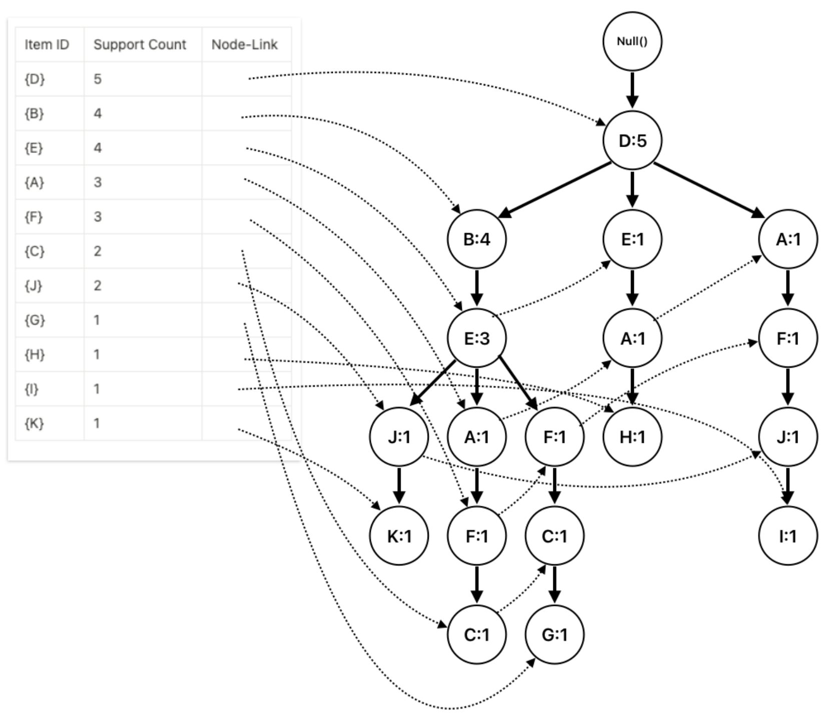
First we scan the table for count of each candidate sorted in descending order

Itemset	Support Count	Support
{D}	5	1.00
{B}	4	0.80
{E}	4	0.80
{A}	3	0.60
{F}	3	0.60



Itemset	Support Count	Support
{C}	2	0.40
{J}	2	0.40
{G}	1	0.20
{H}	1	0.20
{I}	1	0.20
{K}	1	0.20

Then we construct the node links between the itemsets and nodes



Mining the FP-tree by creating conditional pattern bases

Item	Conditional Pattern Base	Conditional FP-tree	Frequent Patterns Generated
{K}	{{D, B, E, J: 1}}	{}	{}
{I}	{{D, A, F, J: 1}}	{}	{}
{H}	{{D, E, A: 1}}	{}	{}
{G}	{{D, B, E, F, C: 1}}	{}	{}
{J}	{{D, B, E: 1}, {D, A, F: 1}}	{}	{}
{C}	{{D, B, E, A, F: 1}, {D, B, E, F: 1}}	{}	{}
{F}	{{D, B, E, A: 1}, {D, B, E: 1}, {D, A: 1}}	{D: 3}	{D, F: 3}
{A}	{{D, B, E: 1}, {D, E: 1}, {D: 1}}	{D: 3}	{D, A: 3}
{E}	{{D, B: 3}, {D: 1}}	{D: 4}	{D, E: 4}
{B}	{{D: 4}}	{D: 4}	{D, B: 4}
{D}	{{}}	{}	{}

Exercise 2.1.2:

From Exercise 2.1.1 we have the following frequent 3-itemset.

Itemset	Support Count	Support
{B, D, E}	3	0.60

With {B, D} = 3, and {B, D, E} = 3, we can calculate the confidence.

$$confidence = \{B, D, E\} / \{B, D\} = 3/3 = 1.00$$

Itemset	Support Count	Support	Confidence
{B, D, E}	3	0.60	1.00

This is the only strong association rule with support 0.60 and confidence 0.80 which matches the metarule {B, D} $\rightarrow$ {E}.

Frequent Itemset and Association Rules Mining using Apriori Algorithm

In this part, you will build a system which can help make recommendations using the Apriori algorithm.

To solve this assignment you will need to go though these pages:

- https://rasbt.github.io/mlxtend/user_guide/preprocessing/TransactionEncoder/
- https://rasbt.github.io/mlxtend/user_guide/frequent_patterns/association_rules/
- https://rasbt.github.io/mlxtend/user_guide/frequent_patterns/apriori/
- https://rasbt.github.io/mlxtend/user_guide/frequent_patterns/fpgrowth/

The `apply` function in `pandas` can prove very useful for this assignment. See https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.apply.html

Source: Online Retail. (2015). UCI Machine Learning Repository. https://doi.org/10.24432/C5BW33.

In [ ]:
```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
```

Load and Inspect Data

In [ ]:
```
invoices = pd.read_csv('apriori_data.csv')
invoices.head()
```

Out[ ]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0

Data Transformation

Drop everything except InvoiceNo and StockCode since we can use InvoiceNo for transaction id and StockCode for item name

In [ ]:
```
data = invoices[['InvoiceNo', 'StockCode']]
```

In [ ]:
```
data.head()
```

Out[ ]:

	InvoiceNo	StockCode
0	536365	85123A
1	536365	71053
2	536365	84406B
3	536365	84029G
4	536365	84029E

Group the data by InvoiceNo and create a list of StockCode for each invoice

In [ ]:
```
transactions = data.groupby(['InvoiceNo'])['StockCode'].apply(list).values.t
```

In [ ]:
```
transactions[0:4]
```

Out[ ]:
```
[['85123A', '71053', '84406B', '84029G', '84029E', '22752', '21730'],
 ['22633', '22632'],
 ['84879',
  '22745',
  '22748',
  '22749',
  '22310',
  '84969',
  '22623',
  '22622',
  '21754',
  '21755',
  '21777',
  '48187'],
 ['22960', '22913', '22912', '22914']]
```

Using TransactionEncoder, convert the transactions into a dataset where each row represents a transaction and each column represents an item. The values will be True or False depending on whether the item is present in that specific transaction.

In [ ]:
```
te = TransactionEncoder()
te_ary = te.fit(transactions).transform(transactions)
transactions_df = pd.DataFrame(te_ary, columns=te.columns_)
```

In [ ]:
```
transactions_df.head()
```

Out[ ]:

	10002	10080	10120	10123C	10123G	10124A	10124G	10125	10133	10134	.
0	False	False	False	False	False	False	False	False	False	False	.
1	False	False	False	False	False	False	False	False	False	False	.
2	False	False	False	False	False	False	False	False	False	False	.
3	False	False	False	False	False	False	False	False	False	False	.
4	False	False	False	False	False	False	False	False	False	False	.

5 rows × 4070 columns

Use Apriori to get the frequent itemsets and inspect the results

Use apriori to find the frequent_itemsets for `min_sup` = `1%`

In [ ]:
```
frequent_itemsets =  apriori(transactions_df, min_support=0.01, use_colnames
```

In [ ]:
```
frequent_itemsets.shape
```

Out[ ]: (1087, 2)

In [ ]:
```
frequent_itemsets.head()
```

Out[ ]:

	support	itemsets
0	0.020193	(15036)
1	0.012587	(15056BL)
2	0.017876	(15056N)
3	0.011236	(16237)
4	0.012510	(20675)

Add an additional column called `items_count` to the dataframe which represents the number of items in the itemset.

In [ ]:
```
frequent_itemsets['items_count'] = frequent_itemsets['itemsets'].apply(lambc
```

In [ ]:
```
frequent_itemsets.head()
```

Out[ ]:

	support	itemsets	items_count
0	0.020193	(15036)	1
1	0.012587	(15056BL)	1
2	0.017876	(15056N)	1
3	0.011236	(16237)	1
4	0.012510	(20675)	1

Display the various itemsets generated sorted (descending) by the items_count.

In [ ]:
```
frequent_itemsets.sort_values(by='items_count', ascending=False).head()
```

Out[ ]:

	support	itemsets	items_count
1086	0.011699	(22423, 22699, 22697, 22698)	4
1085	0.010386	(21931, 22386, 85099B, 22411)	4
1084	0.010077	(20719, 22355, 20723, 20724)	4
1032	0.012548	(20725, 22384, 20728)	3
1024	0.011042	(20725, 22384, 20726)	3

Show how many itemsets exist by items_count

In [ ]:
```
frequent_itemsets.groupby('items_count')['itemsets'].count()
```

Out[ ]:
```
items_count
1    598
2    404
3     82
4      3
Name: itemsets, dtype: int64
```

Generate association rules

Generate all association rules using the `lift` metric with a minimum value of 2

In [ ]:
```
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=2)
```

In [ ]:
```
rules.shape
```

Out[ ]:
```
(1338, 10)
```

In [ ]:
```
rules.head()
```

Out[ ]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	l
0	(20711)	(20712)	0.020541	0.033668	0.011158	0.543233	16.1350
1	(20712)	(20711)	0.033668	0.020541	0.011158	0.331422	16.1350
2	(21931)	(20711)	0.046371	0.020541	0.011506	0.248127	12.0798
3	(20711)	(21931)	0.020541	0.046371	0.011506	0.560150	12.0798
4	(20711)	(22386)	0.020541	0.047529	0.010888	0.530075	11.1526

In [ ]:
```
invoices.head()
```

Out[ ]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2010-12-01 08:26:00	2.55	17850.0
1	536365	71053	WHITE METAL LANTERN	6	2010-12-01 08:26:00	3.39	17850.0
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	2010-12-01 08:26:00	2.75	17850.0
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	2010-12-01 08:26:00	3.39	17850.0
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	2010-12-01 08:26:00	3.39	17850.0

Add the names of the items back in the data frame as save all rules in a csv file

In [ ]:
```
rules['consequents_description'] = rules['consequents'].apply(lambda x: [inv
rules['antecedents_description'] = rules['antecedents'].apply(lambda x: [inv
```

In [ ]:
```
rules.head()
```

Out[ ]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	
0	(20711)	(20712)	0.020541	0.033668	0.011158	0.543233	16.1350
1	(20712)	(20711)	0.033668	0.020541	0.011158	0.331422	16.1350
2	(21931)	(20711)	0.046371	0.020541	0.011506	0.248127	12.0798
3	(20711)	(21931)	0.020541	0.046371	0.011506	0.560150	12.0798
4	(20711)	(22386)	0.020541	0.047529	0.010888	0.530075	11.1526

In [ ]:
```
rules.shape
```

Out[ ]: (1338, 12)

In [ ]:
```
# I used the following line to create the rules_100.csv file which only give
# rules.sample(100).to_csv('rules_100.csv', index=False)

# You must submit the rules.csv file that contains all the 1338 rules by run
rules.to_csv('rules.csv', index=False)
```

In [ ]: