INTRODUCTION

Breast cancer is a type of cancer that starts in the cells of the breast. It can occur in both men and women, but it's more common in women(Mayo Clinic,2022). In this study, we will utilise unsupervised Machine Learning techniques for clustering, employing KMeans Clustering, DBScan, and Agglomerative Clustering algorithms. The supervised machine learning classification will involve the utilization of four distinct algorithms: Decision Tree, Random Forest, Logistic Regression, and Support Vector Classification (SVC).

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

Part 1 - Exploratory Data Analysis

Viewing data set

In [2]:
```
cancer = pd.read_csv("MS4S16_Dataset.csv")

#first observations
cancer.head()
```

Out[2]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean
0	842302.0	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001
1	842517.0	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869
2	84300903.0	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974
3	84348301.0	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414
4	84358402.0	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980

5 rows × 32 columns

In [3]:
```
#buttom five observations
cancer.tail()
```

Out[3]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean
566	926682.0	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400
567	926954.0	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251
568	927241.0	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140
569	92751.0	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000
570	92751.0	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000

5 rows × 32 columns

Summary Statistics

In [4]:
```
cancer.describe()
```

Out[4]:

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	poi
count	5.680000e+02	566.000000	565.000000	567.000000	566.000000	568.000000	567.000000	567.000000	56
mean	3.011402e+07	14.103267	-241.973664	91.949048	654.942403	0.096312	0.104333	0.088712	
std	1.250894e+08	3.517424	445.216862	24.358029	352.555899	0.014178	0.052878	0.079739	5
min	8.670000e+03	6.981000	-999.000000	43.790000	143.500000	0.052630	0.019380	0.000000	-9
25%	8.690778e+05	11.692500	-999.000000	75.190000	420.300000	0.086290	0.064710	0.029520	
50%	9.060010e+05	13.320000	17.000000	86.240000	548.750000	0.095895	0.092630	0.061540	
75%	8.812852e+06	15.780000	21.010000	104.200000	787.050000	0.105325	0.130400	0.130000	
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	

8 rows × 31 columns

Viewing columns names from the Non-null count below, it can be observed that there are some missing values in the dataset

In [5]:
```
cancer.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 571 entries, 0 to 570
Data columns (total 32 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   id                       568 non-null    float64
 1   diagnosis                568 non-null    object 
 2   radius_mean              566 non-null    float64
 3   texture_mean             565 non-null    float64
 4   perimeter_mean           567 non-null    float64
 5   area_mean                566 non-null    float64
 6   smoothness_mean          568 non-null    float64
 7   compactness_mean         567 non-null    float64
 8   concavity_mean           567 non-null    float64
 9   concave points_mean      563 non-null    float64
 10  symmetry_mean            568 non-null    float64
 11  fractal_dimension_mean   567 non-null    float64
 12  radius_se                565 non-null    float64
 13  texture_se               563 non-null    float64
 14  perimeter_se             568 non-null    float64
 15  area_se                  565 non-null    float64
 16  smoothness_se            565 non-null    float64
 17  compactness_se           564 non-null    float64
 18  concavity_se             563 non-null    float64
 19  concave points_se        562 non-null    float64
 20  symmetry_se              563 non-null    float64
 21  fractal_dimension_se     564 non-null    float64
 22  radius_worst             558 non-null    float64
 23  texture_worst            550 non-null    float64
 24  perimeter_worst          565 non-null    float64
 25  area_worst               567 non-null    float64
 26  smoothness_worst         562 non-null    float64
 27  compactness_worst        567 non-null    float64
 28  concavity_worst          568 non-null    float64
 29  concave points_worst     565 non-null    float64
 30  symmetry_worst           567 non-null    float64
 31  fractal_dimension_worst  558 non-null    float64
dtypes: float64(31), object(1)
memory usage: 142.9+ KB
```

Viewing total number of missing values per column and in the whole dataset

In [6]:
```
print(np.sum(cancer.isna().sum()))
cancer.isna().sum()
```

```
203
```

Out[6]:
```
id                         3
diagnosis                  3
radius_mean                5
texture_mean               6
perimeter_mean             4
area_mean                  5
smoothness_mean            3
compactness_mean           4
concavity_mean             4
concave points_mean        8
symmetry_mean              3
fractal_dimension_mean     4
radius_se                  6
texture_se                 8
perimeter_se               3
area_se                    6
smoothness_se              6
compactness_se             7
concavity_se               8
concave points_se          9
symmetry_se                8
fractal_dimension_se       7
radius_worst              13
texture_worst             21
perimeter_worst            6
area_worst                 4
smoothness_worst           9
compactness_worst          4
concavity_worst            3
concave points_worst       6
symmetry_worst             4
fractal_dimension_worst    13
dtype: int64
```

In [7]:
```
#replacing invalid values
cancer.replace(['?','*','-',-999,999],np.nan,inplace=True)
```

Viewing total number of missing values per column and in the whole dataset after replacing invalid values

In [8]:
```
print(np.sum(cancer.isna().sum()))
cancer.isna().sum()
```

Out[8]:
```
358
id                          3
diagnosis                   3
radius_mean                 5
texture_mean              151
perimeter_mean              4
area_mean                   5
smoothness_mean             3
compactness_mean            4
concavity_mean              4
concave points_mean        10
symmetry_mean               3
fractal_dimension_mean     12
radius_se                   6
texture_se                  8
perimeter_se                3
area_se                     6
smoothness_se               6
compactness_se              7
concavity_se                8
concave points_se           9
symmetry_se                 8
fractal_dimension_se        7
radius_worst               13
texture_worst              21
perimeter_worst             6
area_worst                  4
smoothness_worst            9
compactness_worst           4
concavity_worst             3
concave points_worst        6
symmetry_worst              4
fractal_dimension_worst    13
dtype: int64
```

Removing duplicates from dataset

In [9]:
```
#removing duplicates
cancer.drop_duplicates(inplace=True)
```

In [10]:
```
print(np.sum(cancer.isna().sum()))
cancer.isna().sum()
```

Out[10]:
```
293
id                          1
diagnosis                   1
radius_mean                 3
texture_mean              148
perimeter_mean              2
area_mean                   3
smoothness_mean             1
compactness_mean            2
concavity_mean              2
concave points_mean         8
symmetry_mean               1
fractal_dimension_mean     10
radius_se                   4
texture_se                  6
perimeter_se                1
area_se                     4
smoothness_se               4
compactness_se              5
concavity_se                6
concave points_se           7
symmetry_se                 6
fractal_dimension_se        5
radius_worst               11
texture_worst              19
perimeter_worst             4
area_worst                  2
smoothness_worst            7
compactness_worst           2
concavity_worst             1
concave points_worst        4
symmetry_worst              2
fractal_dimension_worst    11
dtype: int64
```

Removing all rows with NaN

In [11]:
```
cancer.dropna(how='all',inplace=True)
```

Dropping ID column

In [12]:
```
cancer.drop('id',axis=1,inplace=True)
cancer.head()
```

Out[12]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean
0	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710
1	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017
2	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790
3	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520
4	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430

5 rows × 31 columns

Calculating with the median score of each column

In [13]:
```
medians = cancer.iloc[:,1:].median()
medians
```

Out[13]:
```
radius_mean                 13.320000
texture_mean                18.910000
perimeter_mean              86.240000
area_mean                  548.750000
smoothness_mean              0.095895
compactness_mean             0.092630
concavity_mean               0.061540
concave points_mean          0.033410
symmetry_mean                0.179200
fractal_dimension_mean       0.061550
radius_se                    0.323700
texture_se                   1.127000
perimeter_se                 2.282500
area_se                     24.440000
smoothness_se                0.006356
compactness_se               0.020435
concavity_se                 0.025860
concave points_se            0.010920
symmetry_se                  0.018730
fractal_dimension_se         0.003217
radius_worst                14.965000
texture_worst               25.445000
perimeter_worst             97.820000
area_worst                 686.600000
smoothness_worst             0.131400
compactness_worst            0.214100
concavity_worst              0.227450
concave points_worst         0.099930
symmetry_worst               0.282300
fractal_dimension_worst      0.080015
dtype: float64
```

Replacing the missing values with the calculated median for each column

In [14]:
```
cancer.fillna(medians,inplace=True)
cancer.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 566 entries, 0 to 569
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   diagnosis                566 non-null    object 
 1   radius_mean              566 non-null    float64
 2   texture_mean             566 non-null    float64
 3   perimeter_mean           566 non-null    float64
 4   area_mean                566 non-null    float64
 5   smoothness_mean          566 non-null    float64
 6   compactness_mean         566 non-null    float64
 7   concavity_mean           566 non-null    float64
 8   concave points_mean      566 non-null    float64
 9   symmetry_mean            566 non-null    float64
 10  fractal_dimension_mean   566 non-null    float64
 11  radius_se                566 non-null    float64
 12  texture_se               566 non-null    float64
 13  perimeter_se             566 non-null    float64
 14  area_se                  566 non-null    float64
 15  smoothness_se            566 non-null    float64
 16  compactness_se           566 non-null    float64
 17  concavity_se             566 non-null    float64
 18  concave points_se        566 non-null    float64
 19  symmetry_se              566 non-null    float64
 20  fractal_dimension_se     566 non-null    float64
 21  radius_worst             566 non-null    float64
 22  texture_worst            566 non-null    float64
 23  perimeter_worst          566 non-null    float64
 24  area_worst               566 non-null    float64
 25  smoothness_worst         566 non-null    float64
 26  compactness_worst        566 non-null    float64
 27  concavity_worst          566 non-null    float64
 28  concave points_worst     566 non-null    float64
 29  symmetry_worst           566 non-null    float64
 30  fractal_dimension_worst  566 non-null    float64
dtypes: float64(30), object(1)
memory usage: 141.5+ KB
```

Comfirming no null values in the dataset

In [15]:
```
print(np.sum(cancer.isna().sum()))
cancer.isna().sum()
```

Out[15]:
```
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
```

Finding the correlation matrixes in the dataset

In [16]:
```
cancer_corr = cancer.corr(numeric_only=True,method='spearman')
cancer_corr
```

Out[16]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	p
radius_mean	1.000000	0.297752	0.994021	0.994437	0.143170	0.491618	0.640195	
texture_mean	0.297752	1.000000	0.313676	0.309660	0.015267	0.241098	0.290223	
perimeter_mean	0.994021	0.313676	1.000000	0.993397	0.179369	0.539625	0.678123	
area_mean	0.994437	0.309660	0.993397	1.000000	0.134206	0.486096	0.639134	
smoothness_mean	0.143170	0.015267	0.179369	0.134206	1.000000	0.678335	0.518251	
compactness_mean	0.491618	0.241098	0.539625	0.486096	0.678335	1.000000	0.896522	
concavity_mean	0.640195	0.290223	0.678123	0.639134	0.518251	0.896522	1.000000	
concave points_mean	0.752496	0.258965	0.782209	0.749111	0.558306	0.838432	0.920041	
symmetry_mean	0.117408	0.091654	0.148939	0.116973	0.543026	0.554474	0.449128	
fractal_dimension_mean	-0.354226	-0.048577	-0.310838	-0.360018	0.574098	0.484410	0.249655	
radius_se	0.548502	0.306454	0.561951	0.556028	0.336899	0.504206	0.572655	
texture_se	-0.156886	0.356127	-0.141598	-0.148301	0.097156	0.049083	0.046353	
perimeter_se	0.561593	0.327813	0.582747	0.569360	0.332595	0.585092	0.646947	
area_se	0.728761	0.335644	0.737321	0.735573	0.305806	0.546692	0.648184	
smoothness_se	-0.326394	0.015482	-0.304673	-0.323744	0.338265	0.126316	0.073204	
compactness_se	0.261605	0.206116	0.306577	0.260143	0.400773	0.814902	0.756638	
concavity_se	0.356540	0.232526	0.395712	0.355354	0.358012	0.772144	0.855275	
concave points_se	0.402200	0.182996	0.435061	0.396671	0.440758	0.732236	0.770484	
symmetry_se	-0.242347	-0.000022	-0.225446	-0.236608	0.147781	0.100463	0.021937	
fractal_dimension_se	-0.025100	0.094800	0.016721	-0.023781	0.406694	0.606357	0.501761	
radius_worst	0.965258	0.324762	0.969411	0.966582	0.198727	0.537745	0.672741	
texture_worst	0.292402	0.773640	0.309008	0.304225	0.071203	0.251576	0.325569	
perimeter_worst	0.951554	0.341942	0.965717	0.952519	0.215805	0.574109	0.703884	
area_worst	0.968924	0.318818	0.972037	0.970814	0.192656	0.526893	0.670641	
smoothness_worst	0.122112	0.099876	0.153629	0.118917	0.788058	0.573317	0.488535	
compactness_worst	0.487548	0.253018	0.531370	0.484210	0.479916	0.900970	0.849451	
concavity_worst	0.591351	0.290195	0.628882	0.591600	0.427905	0.837712	0.938807	
concave points_worst	0.719456	0.278454	0.751366	0.718347	0.493277	0.821977	0.905343	
symmetry_worst	0.173842	0.124999	0.197974	0.172718	0.394624	0.454450	0.385332	
fractal_dimension_worst	0.040004	0.094786	0.083553	0.035975	0.513224	0.688167	0.541613	

30 rows × 30 columns

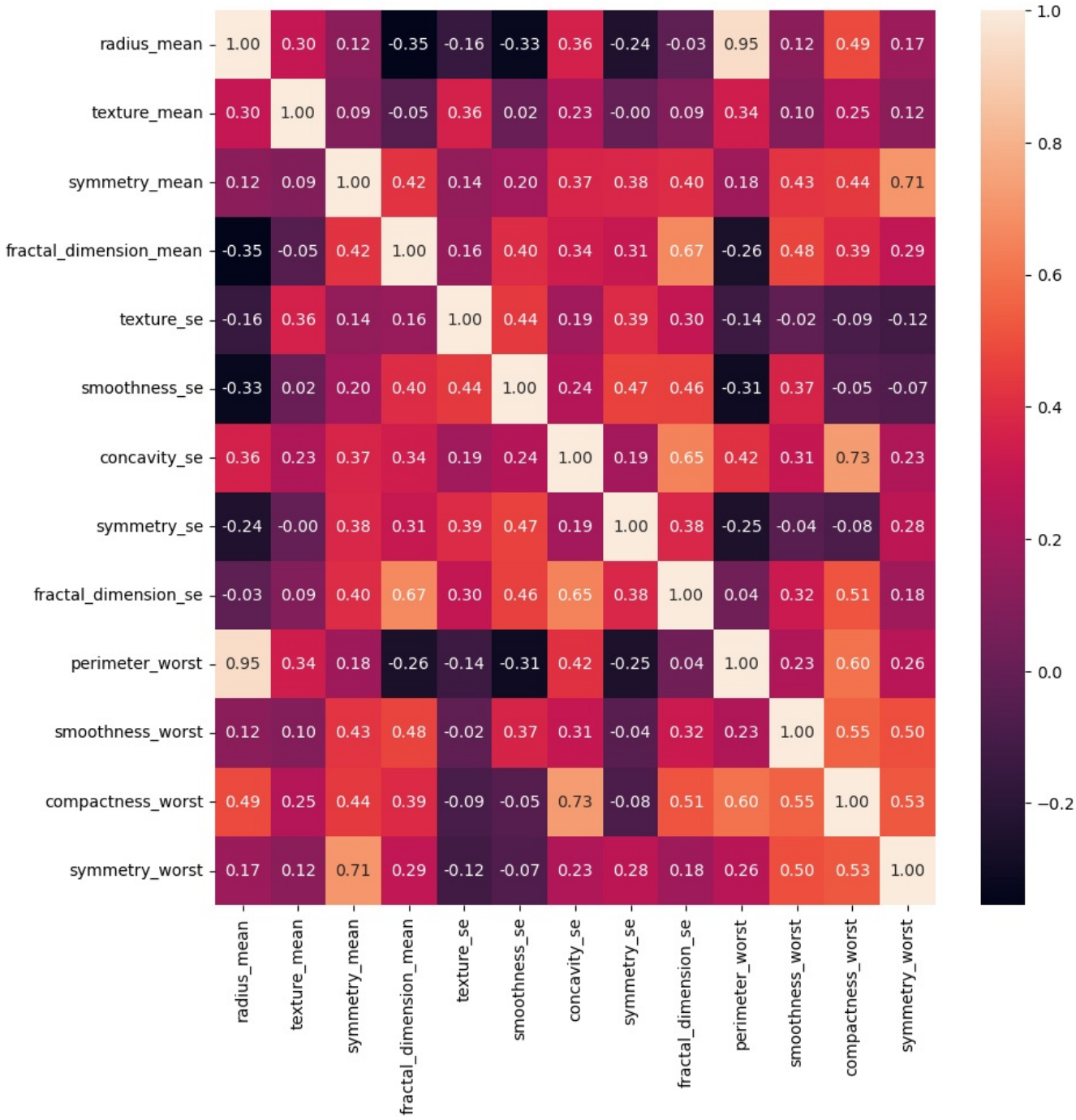
Dropping highly correlated features from the dataset with plotted heatmap showing features which are not highly correlated

In [17]:
```
cancer_corr.drop(['perimeter_mean','concave points_mean','area_mean','concavity_mean','radius_worst','compactne
'fractal_dimension_worst','radius_se','perimeter_se','area_se','concavity_worst','concave points_worst',
'compactness_se','concave points_se','texture_worst','area_worst'],inplace=True,axis=1)

cancer_corr.drop(['perimeter_mean','concave points_mean','area_mean','concavity_mean','radius_worst','compactne
'fractal_dimension_worst','radius_se','perimeter_se','area_se','concavity_worst','concave points_worst',
'compactness_se','concave points_se','texture_worst','area_worst'],inplace=True,axis=0)
```

In [18]:
```
plt.figure(figsize=(10,10))
sns.heatmap(cancer_corr,annot=True,fmt='.2f')
```

Out[18]:
```
<Axes: >
```



Using features from correlation to get the actual columns for the analysis and viewing the refinded dataset

In [19]:
```
#using features from correlation to get columns for the analysis
list_of_cols = list(cancer_corr.columns)
list_of_cols.append('diagnosis')
```

In [20]:
```
cancer = cancer[list_of_cols]
cancer
```

Out[20]:

	radius_mean	texture_mean	symmetry_mean	fractal_dimension_mean	texture_se	smoothness_se	concavity_se	symmetry_se	fractal_di
0	17.99	10.38	0.2419	0.07871	0.9053	0.006399	0.05373	0.03003	
1	20.57	17.77	0.1812	0.05667	0.7339	0.005225	0.01860	0.01389	
2	19.69	21.25	0.2069	0.05999	0.7869	0.006150	0.03832	0.02250	
3	11.42	20.38	0.2597	0.09744	1.1560	0.009110	0.05661	0.05963	
4	20.29	14.34	0.1809	0.05883	0.7813	0.011490	0.05688	0.01756	
...	...	...	...	...	...	...	...	...	
565	21.56	22.39	0.1726	0.05623	1.2560	0.010300	0.05198	0.01114	
566	20.13	28.25	0.1752	0.05533	2.4630	0.005769	0.03950	0.01898	
567	16.60	28.08	0.1590	0.05648	1.0750	0.005903	0.04730	0.01318	
568	20.60	29.33	0.2397	0.06155	1.5950	0.006522	0.07117	0.02324	
569	7.76	24.54	0.1587	0.05884	1.4280	0.007189	0.00000	0.02676	

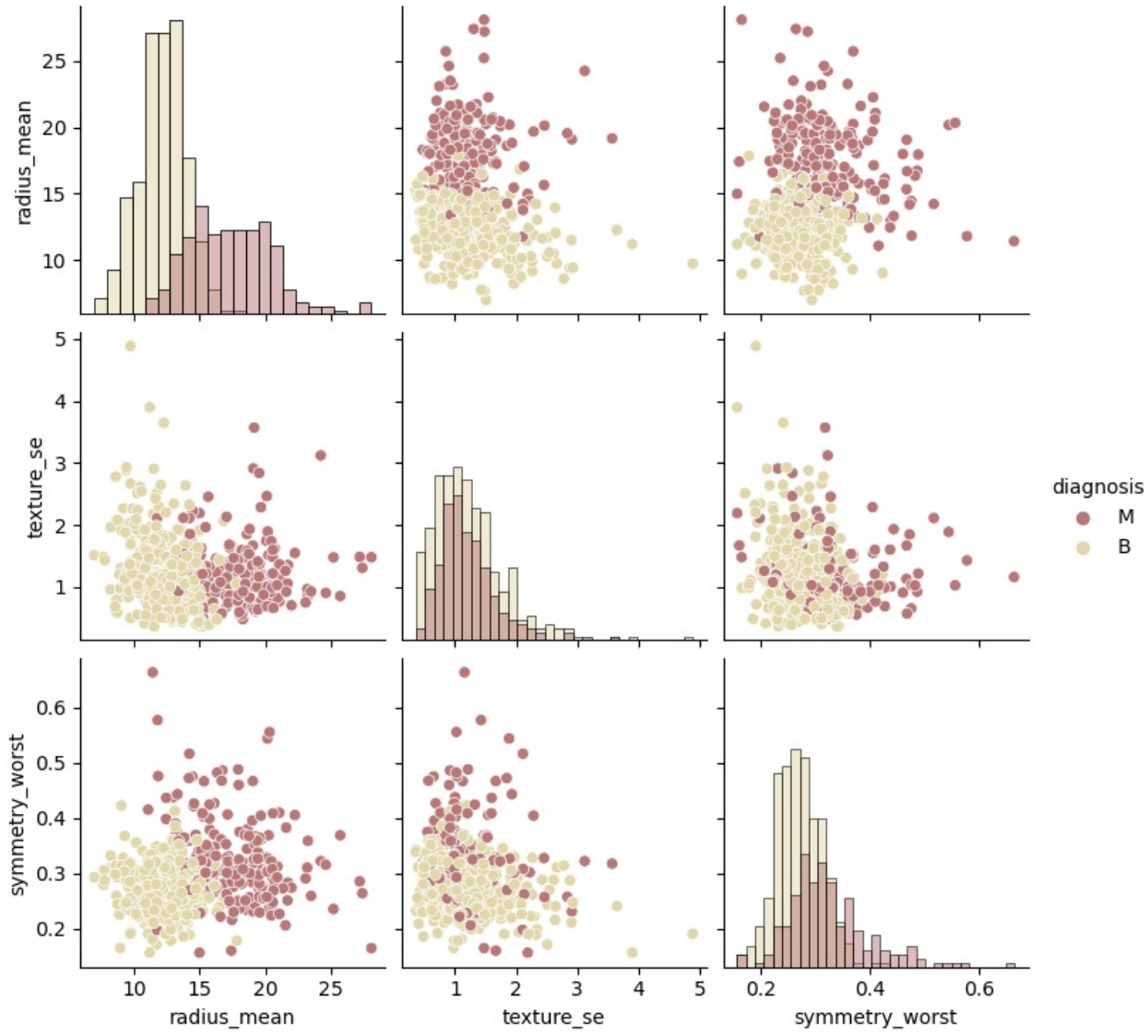
566 rows × 14 columns

Pair plots of some features

In [87]:
```
sns.pairplot(cancer, x_vars=['radius_mean','texture_se','symmetry_worst'],
             y_vars=['radius_mean','texture_se','symmetry_worst'],hue='diagnosis',diag_kind='hist',
            palette='pink')
```

```
/Users/jay/anaconda3/lib/python3.11/site-packages/seaborn/axisgrid.py:118: UserWarning: The figure layout has c
hanged to tight
  self._figure.tight_layout(*args, **kwargs)
```

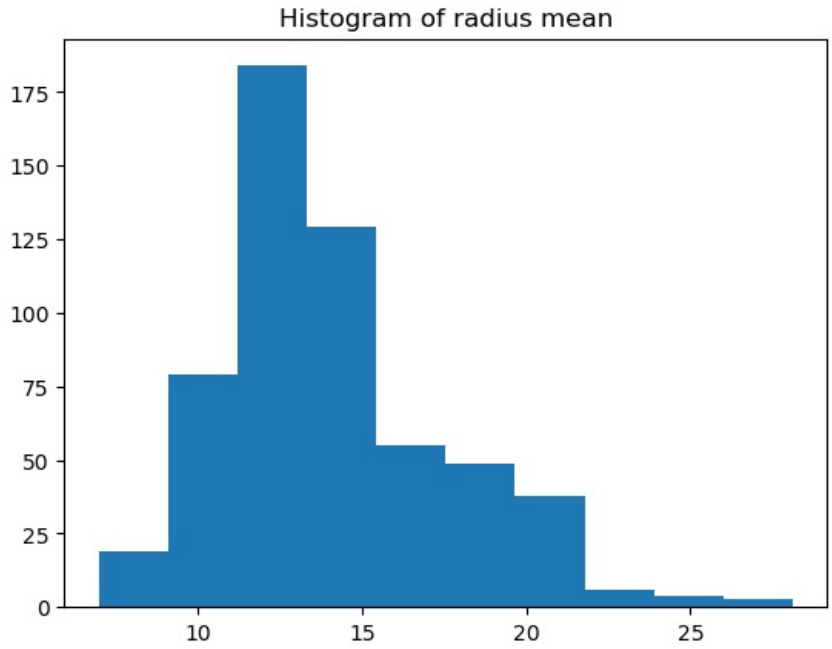
Out[87]:
```
<seaborn.axisgrid.PairGrid at 0x16b9bcfd0>
```



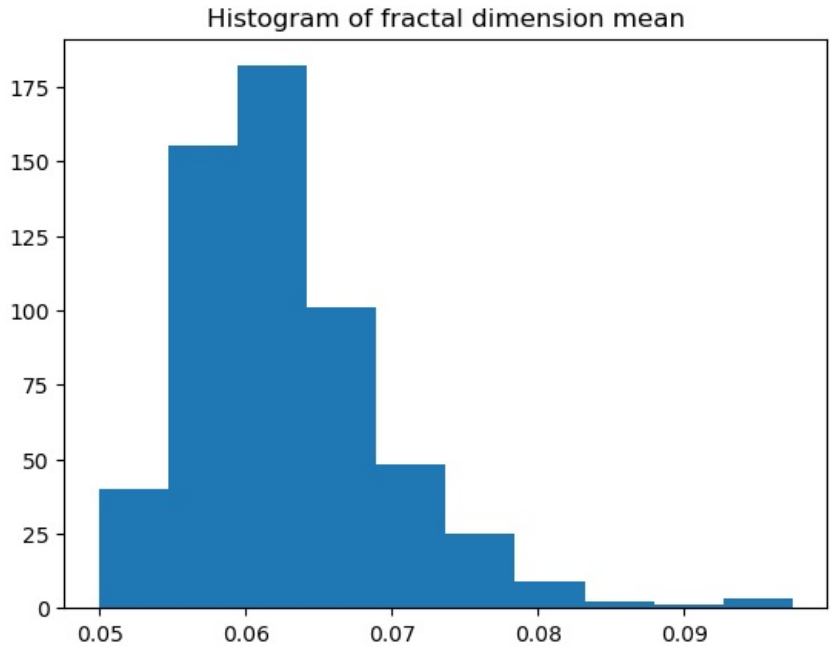
Histograms of some features in the refined dataset

In [22]:
```
plt.hist(x=cancer['radius_mean'])
plt.title('Histogram of radius mean')
```

```
plt.show()
```



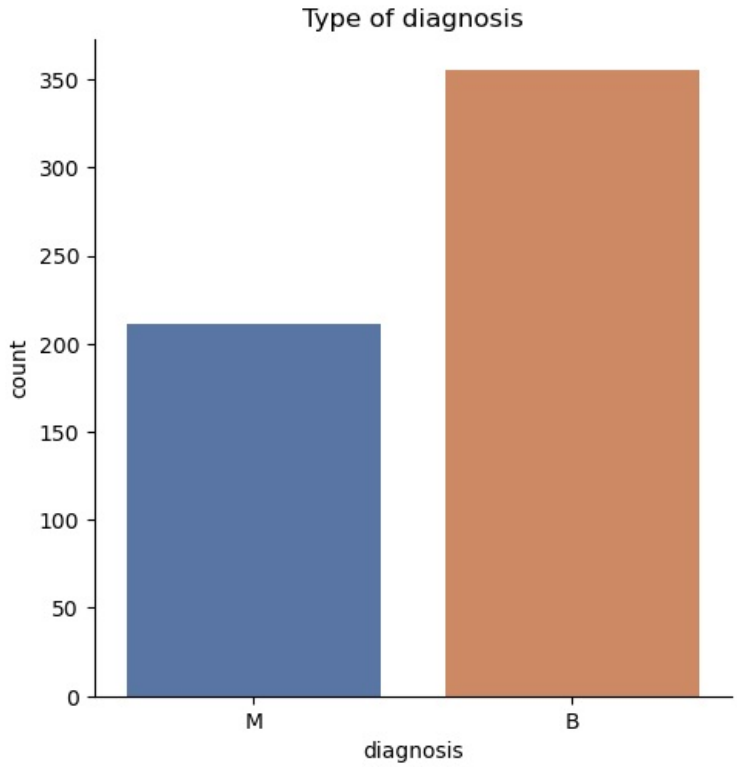
In [23]:
```
plt.hist(x=cancer['fractal_dimension_mean'])
plt.title('Histogram of fractal dimension mean')
plt.show()
```



Displaying Count Plot of type of diagnosis instances

In [24]:
```
g = sns.catplot(kind='count',data=cancer,x='diagnosis',palette='deep')
plt.title('Type of diagnosis')
plt.show()
```

```
/Users/jay/anaconda3/lib/python3.11/site-packages/seaborn/axisgrid.py:118: UserWarning: The figure layout has c
hanged to tight
  self._figure.tight_layout(*args, **kwargs)
```



Feature Scaling of refined dataset, using standard scaler because it is more tolerant to outliers...etc

In [25]:
```
#using Standard Scaler
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

In [26]:
```
#selecting input features from the dataset
X = cancer.drop('diagnosis',axis=1)
```

In [27]:
```
X
```

Out[27]:

	radius_mean	texture_mean	symmetry_mean	fractal_dimension_mean	texture_se	smoothness_se	concavity_se	symmetry_se	fractal_di
0	17.99	10.38	0.2419	0.07871	0.9053	0.006399	0.05373	0.03003	
1	20.57	17.77	0.1812	0.05667	0.7339	0.005225	0.01860	0.01389	
2	19.69	21.25	0.2069	0.05999	0.7869	0.006150	0.03832	0.02250	
3	11.42	20.38	0.2597	0.09744	1.1560	0.009110	0.05661	0.05963	
4	20.29	14.34	0.1809	0.05883	0.7813	0.011490	0.05688	0.01756	
...	...	...	...	...	...	...	...	...	
565	21.56	22.39	0.1726	0.05623	1.2560	0.010300	0.05198	0.01114	
566	20.13	28.25	0.1752	0.05533	2.4630	0.005769	0.03950	0.01898	
567	16.60	28.08	0.1590	0.05648	1.0750	0.005903	0.04730	0.01318	
568	20.60	29.33	0.2397	0.06155	1.5950	0.006522	0.07117	0.02324	
569	7.76	24.54	0.1587	0.05884	1.4280	0.007189	0.00000	0.02676	

566 rows × 13 columns

In [28]:
```
X_scaled = scaler.fit_transform(X)
```

```
X_scaled
```

Out[28]:
```
array([[ 1.10985104, -2.28386284,  0.47342389, ...,  1.30719176,
         2.61069355,  2.74424311],
       [ 1.84685777, -0.38092512, -0.05394972, ..., -0.38491285,
        -0.43278748, -0.24651037],
       [ 1.59547563,  0.51518087,  0.16933697, ...,  0.52283077,
         1.07878692,  1.14788199],
       ...,
       [ 0.71278152,  2.27391763, -0.24682771, ..., -0.82115856,
         0.34746193, -1.10608944],
       [ 1.85542762,  2.59579478,  0.45430986, ...,  1.43057438,
         3.89734263,  1.91374755],
       [-1.81246635,  1.36236154, -0.24943417, ..., -1.87608003,
        -1.20897045, -0.05100461]])
```

PART 2 - CLUSTER ANALYSIS - Unsupervised Machine Learning

In [29]:
```
#unsupervised learning
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
```

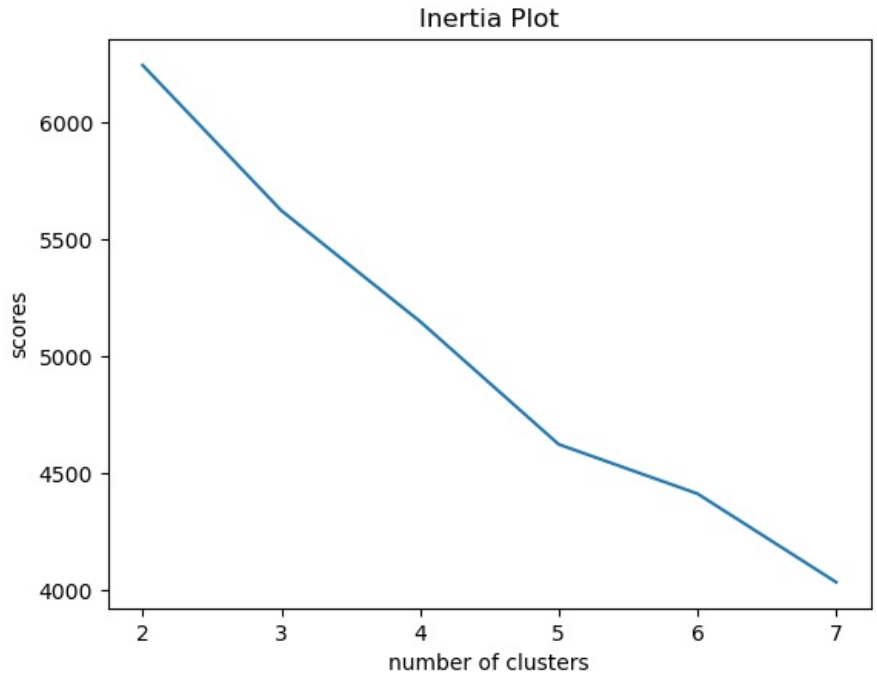
Finding Inertia to help with cluster identification

In [30]:
```
inertia = []

for i in range(2,8):
    kmeans = KMeans(n_clusters=i,n_init='auto')
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)
```

Plot showing Inertia Values for clusters

In [31]:
```
#inertia values
sns.lineplot(y=inertia,x=range(2,8))
#plt.scatter(x=range(2,8),y=inertia)
plt.title('Inertia Plot')
plt.xlabel('number of clusters')
plt.ylabel('scores')
plt.show()
```



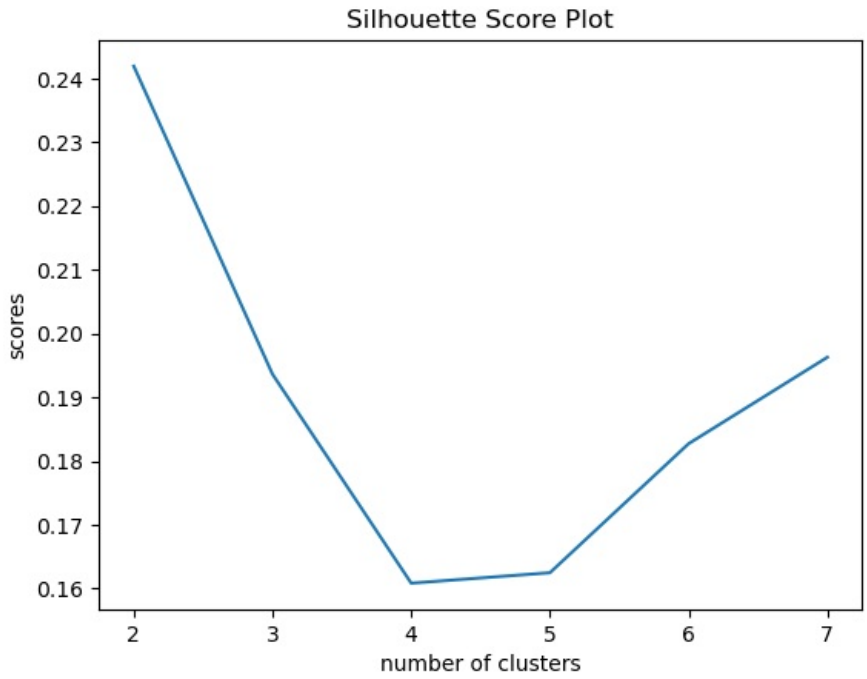
Finding the silhouette score to determine the number of clusters

In [32]:
```
sil_score = []
for i in range(2,8):
    kmeans = KMeans(n_clusters=i,n_init='auto')
    kmeans.fit(X_scaled)
    sil_score.append(silhouette_score(X_scaled,kmeans.labels_))
```

In [33]:
```
sil_score
```

Out[33]:
```
[0.2418984985567839,
 0.19362258501850732,
 0.1608609357669621,
 0.16249656851872757,
 0.18273100392404268,
 0.19627323845947534]
```

In [34]:
```
sns.lineplot(y=sil_score,x=range(2,8))
plt.title('Silhouette Score Plot')
plt.xlabel('number of clusters')
plt.ylabel('scores')
plt.show()
```



From the above silhouette score the highest number of clusters to consider is 2.

Fitting the scaled data using the selected number of clusters

In [35]:
```
kmeans = KMeans(n_clusters=2,n_init='auto')
```

In [36]:
```
kmeans.fit(X_scaled)
```

Out[36]:
```
KMeans(n_clusters=2, n_init='auto')
```

Predicted clusters from Kmeans

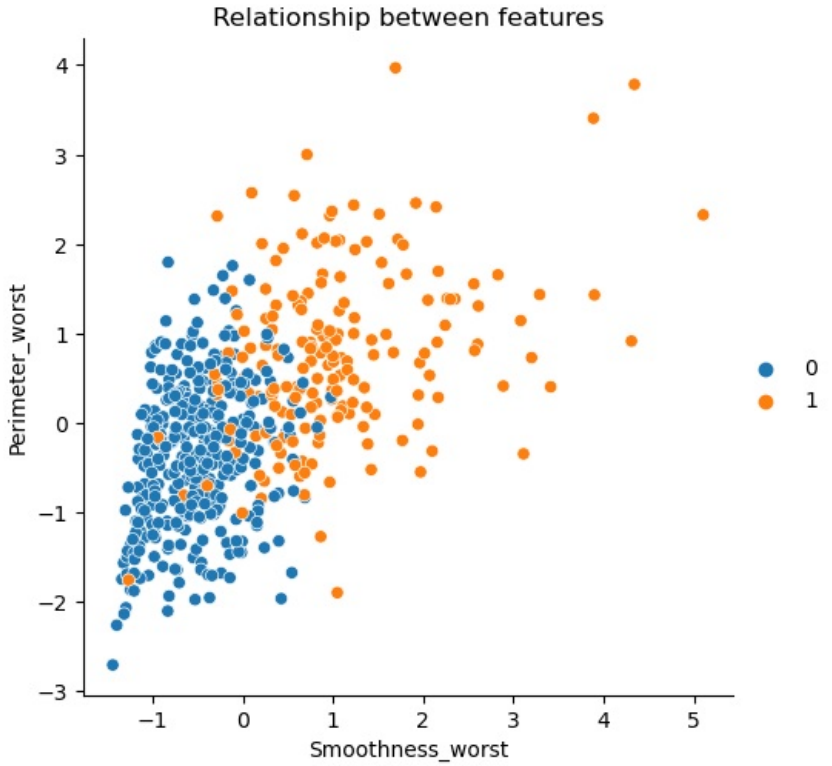
In [37]:
```
kmeans.labels_
```

Out[37]:
```
array([1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0,
       1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1,
       1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1,
       0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0,
       0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0,
       1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
       1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0,
       1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
       1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0,
       0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0,
       0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0,
       1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0], dtype=int32)
```

Plotting the Clusters

In [38]:
```
#plotting clusters
sns.relplot(x = X_scaled[:,11],y =X_scaled[:,10],kind='scatter',hue=kmeans.labels_)
plt.title('Relationship between features')
plt.xlabel('Smoothness_worst')
plt.ylabel('Perimeter_worst')
plt.show()
```

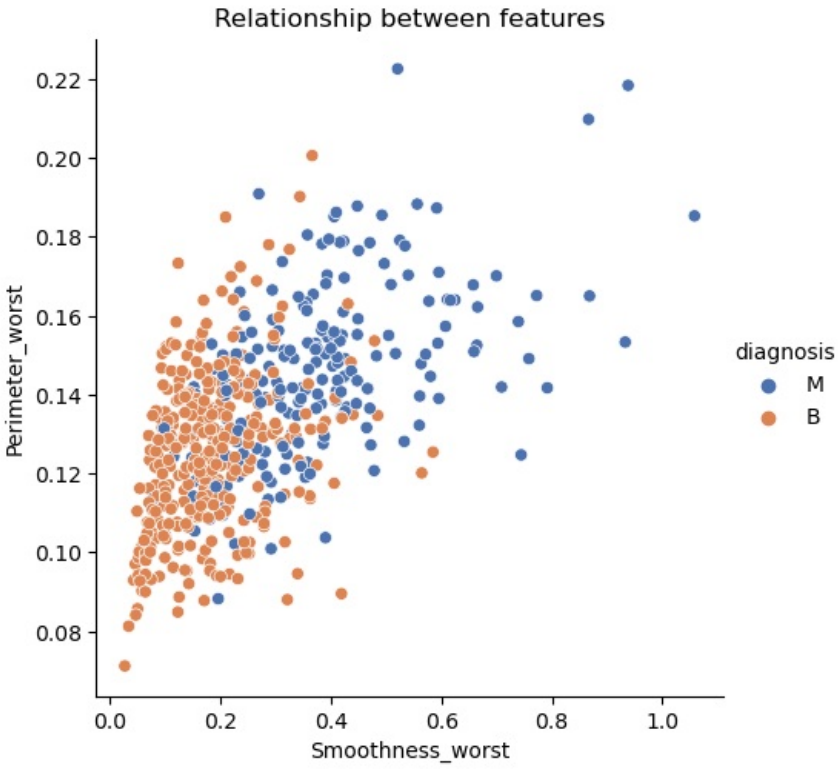
```
/Users/jay/anaconda3/lib/python3.11/site-packages/seaborn/axisgrid.py:118: UserWarning: The figure layout has c
hanged to tight
  self._figure.tight_layout(*args, **kwargs)
```



The above scatter plot displays two features being seperated by the Kmeans label confirming the two clusters

In [39]:
```
sns.relplot(data = cancer,x = cancer.iloc[:,11],
            y =cancer.iloc[:,10],kind='scatter',hue='diagnosis',palette='deep')
plt.title('Relationship between features')
plt.xlabel('Smoothness_worst')
plt.ylabel('Perimeter_worst')
plt.show()
```

```
/Users/jay/anaconda3/lib/python3.11/site-packages/seaborn/axisgrid.py:118: UserWarning: The figure layout has c
hanged to tight
  self._figure.tight_layout(*args, **kwargs)
```



The above scatter plot displays features being seperated by the diagnosis feature of the dataset

In [40]:
```
#Agglomerative clsutering
from sklearn.cluster import AgglomerativeClustering
agglomerative = AgglomerativeClustering()
```

In [41]:
```
agglomerative.fit(X_scaled)
```

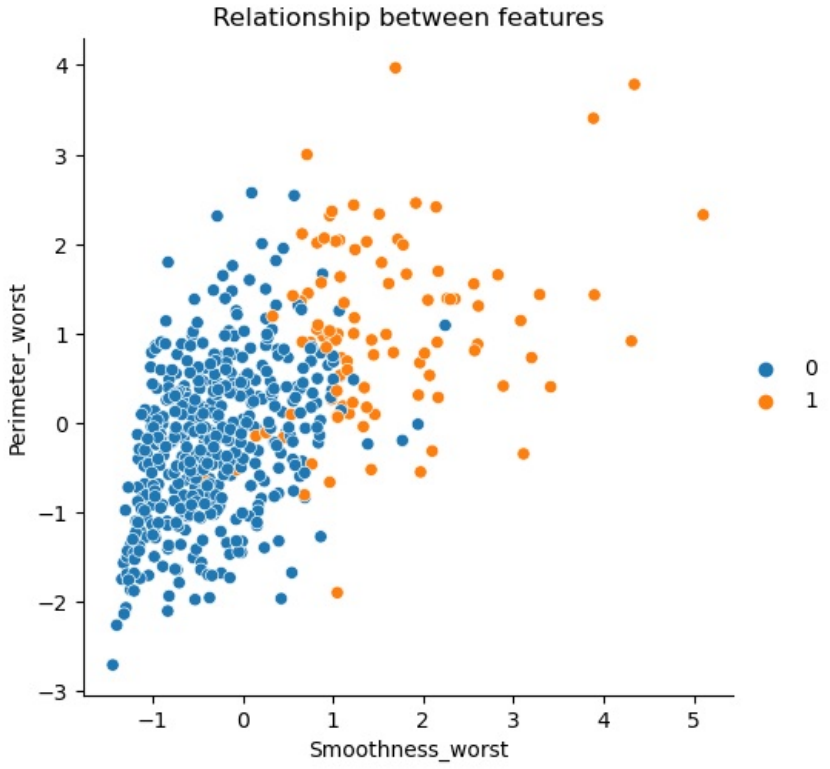
Out[41]:
```
▾ AgglomerativeClustering
AgglomerativeClustering()
```

In [42]:
```
agglomerative.labels_
```

Out[42]:
```
array([1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0,
       1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1,
       0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0,
       0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0,
       0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0])
```

In [43]:
```
sns.relplot(x = X_scaled[:,11],y =X_scaled[:,10],kind='scatter',hue=agglomerative.labels_)
plt.title('Relationship between features')
plt.xlabel('Smoothness_worst')
plt.ylabel('Perimeter_worst')
plt.show()
```

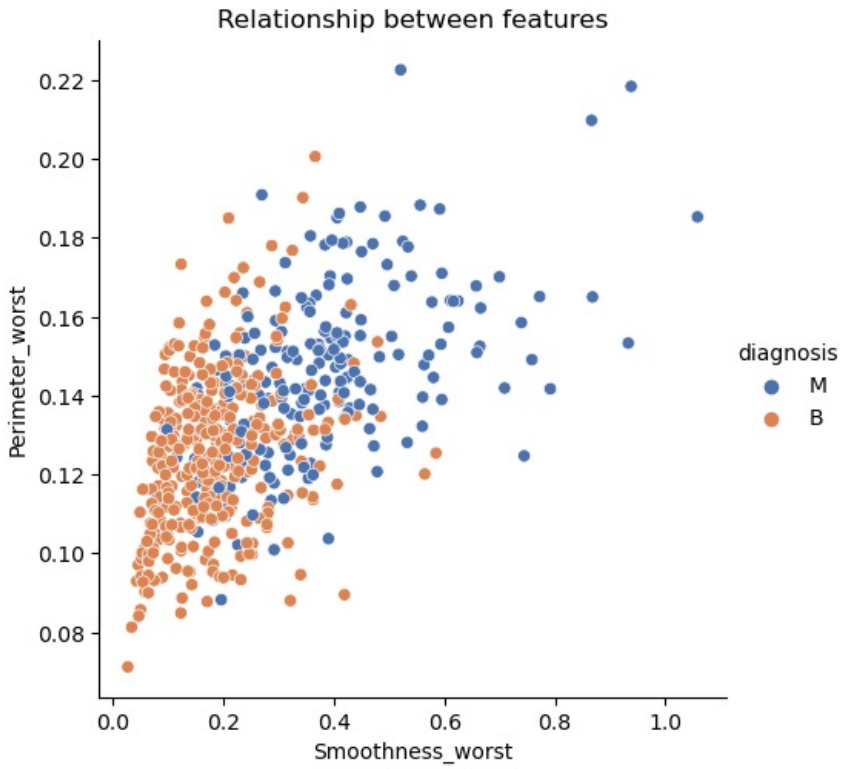
```
/Users/jay/anaconda3/lib/python3.11/site-packages/seaborn/axisgrid.py:118: UserWarning: The figure layout has c
hanged to tight
  self._figure.tight_layout(*args, **kwargs)
```



The above scatter plot displays two features being seperated by the agglomerative label confirming the two clusters

In [44]:
```
sns.relplot(data = cancer,x = cancer.iloc[:,11],
            y =cancer.iloc[:,10],kind='scatter',hue='diagnosis',palette='deep')
plt.title('Relationship between features')
plt.xlabel('Smoothness_worst')
plt.ylabel('Perimeter_worst')
plt.show()
```

```
/Users/jay/anaconda3/lib/python3.11/site-packages/seaborn/axisgrid.py:118: UserWarning: The figure layout has c
hanged to tight
  self._figure.tight_layout(*args, **kwargs)
```



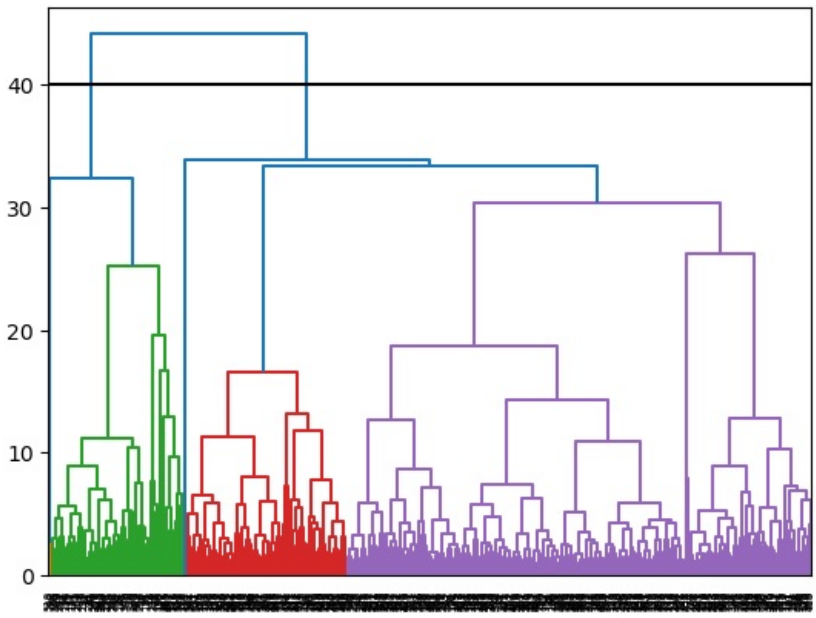
The above scatter plot displays features being seperated by the diagnosis feature of the dataset

Dendrograms

visual representation confirming the number of clusters chosen

In [45]:
```
#Dendrograms
from scipy.cluster.hierarchy import dendrogram,linkage
```

In [46]:
```
dendro_plot = linkage(X_scaled,'ward')
dendrogram(dendro_plot)
plt.axhline(40, c='black')
plt.show()
```



The above dendograms confirms the decision to take 2 clusters

DBSCAN

Finding clusters using DBSCAN

In [47]:
```
from sklearn.cluster import DBSCAN
```

In [48]:
```
dbscan = DBSCAN(eps=2)
```

In [49]:
```
dbscan.fit(X_scaled)
```

Out[49]:
```
DBSCAN(eps=2)
```

In [50]:
```
dbscan.labels_
```

Out[50]:
```
array([-1,  0,  0, -1,  0,  0,  0,  0,  0, -1,  0,  0, -1,  0,  0,  0,  0,
        0,  0,  0,  0,  0, -1,  0,  0, -1,  0,  0,  0,  0,  0,  0,  0,  0,
        0, -1,  0,  0, -1,  0,  0,  0, -1,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0, -1,  0,  0, -1,  0,  0,  0,  0,
       -1,  0,  0, -1,  0,  0,  0,  0, -1,  0, -1,  0,  0,  0, -1, -1,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, -1,
        0,  0,  0,  0,  0,  0, -1,  0,  0,  0, -1,  0,  0,  0, -1,  0,  0,
       -1,  0,  0, -1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0, -1,  0,  0,  0,  0,  0,  0,  0, -1,  0,  0,  0,  0, -1, -1,
        0,  0,  0,  0,  0,  0,  0,  0, -1,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0, -1,  0,  0,  0, -1, -1,  0,  0,  0,  0,  0,
        0,  0,  0, -1,  0, -1,  0,  0,  0, -1,  0,  0,  0,  0,  0, -1, -1,
        0,  0,  0,  0,  0,  0,  0,  0, -1, -1, -1,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0, -1,  0,  0, -1, -1,  0, -1,  0,  0, -1,  0,  0,  0,  0,  0,  0,
       -1, -1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, -1,  0, -1,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0, -1,  0,  0,  0,  0, -1,  0,
        0,  0,  0,  0,  0, -1,  0,  0,  0, -1,  0,  0,  0,  0, -1,  0,  0,
        0,  0,  0,  0,  0,  0, -1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
       -1,  0, -1,  0,  0,  0,  0,  0, -1,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0, -1,  0,  0,  0, -1,  0,  0,  0,  0, -1,
        0,  0, -1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0, -1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0, -1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, -1, -1,  0,  0,  0, -1,
        0,  0,  0,  0,  0,  0, -1, -1,  0,  0,  0, -1, -1,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0, -1,  0,  0,  0, -1,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0, -1, -1,  0, -1,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0, -1,  0,  0,  0,  0,  0,  0,  0,  0, -1,  0,
        0,  0,  0,  0,  0,  0,  0,  0, -1, -1, -1,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0, -1,  0, -1,  0, -1,  0, -1,  0, -1,
        0, -1,  0, -1,  0])
```

The above DBSCAN shows a single cluster labeled 0 and -1 indicating noise.

Part 3 - Supervised Machine Learning - Classification of 'M' and 'B'

In [51]:
```
#Training data for classification
from sklearn.model_selection import train_test_split
```

Replacing diagnosis 'M' with 1 and 'B' with 0

In [52]:
```
cancer['class'] = cancer['diagnosis'].apply(lambda x: 1 if x=='M' else 0)
```

In [53]:
```
X= cancer.drop(['diagnosis','class'],axis=1) #Data
y = cancer['class'] #Label data
X = np.array(X).reshape(len(X),len(X.columns))
y = np.array(y).reshape(len(y),)
```

Spiltting the Dataset

In [54]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2)
```

Feature Scaling of Dataset

In [55]:
```
#using Standard Scaler
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

In [56]:
```
X_train_set = scaler.fit_transform(X_train)
X_test_set = scaler.transform(X_test)
```

Prefered Scores being used

In [57]:
```
from sklearn.metrics import accuracy_score, recall_score,precision_score, confusion_matrix
```

Decision Tree

In [58]:
```
#Decision trees

from sklearn.tree import DecisionTreeClassifier
dec_tree = DecisionTreeClassifier()
```

Fitting the X-values and the Y_train set

In [59]:
```
dec_tree.fit(X_train,y_train)
```

Out[59]:
```
DecisionTreeClassifier
DecisionTreeClassifier()
```

Predicting the X_train values for model evaluation

In [60]:
```
y_predictions = dec_tree.predict(X_train)
```

In [61]:
```
print(f"Accuracy: {accuracy_score(y_train,y_predictions)}")
print(f"Recall: {recall_score(y_train,y_predictions)}")
print(f"Precision: {precision_score(y_train,y_predictions)}")
print(f"Confusion matrix:\n {confusion_matrix(y_train,y_predictions)}")
```

```
Accuracy: 1.0
Recall: 1.0
Precision: 1.0
Confusion matrix:
 [[282   0]
 [  0 170]]
```

Logistic Regression

In [62]:
```
#logistic regression
from sklearn.linear_model import LogisticRegression
logistic = LogisticRegression(max_iter=10000)
```

In [63]:
```
logistic.fit(X_train_set,y_train)
```

Out[63]:
```
LogisticRegression
LogisticRegression(max_iter=10000)
```

In [64]:
```
y_predictions = logistic.predict(X_train_set)
```

In [65]:
```
print(f"Accuracy: {accuracy_score(y_train,y_predictions)}")
print(f"Recall: {recall_score(y_train,y_predictions)}")
print(f"Precision: {precision_score(y_train,y_predictions)}")
print(f"Confusion matrix:\n {confusion_matrix(y_train,y_predictions)}")
```

```
Accuracy: 0.9601769911504425
Recall: 0.9470588235294117
Precision: 0.9470588235294117
Confusion matrix:
 [[273   9]
 [  9 161]]
```

SVC

In [66]:
```
#svc
from sklearn.svm import SVC
svc = SVC()
```

In [67]:
```
svc.fit(X_train_set,y_train)
```

Out[67]:
```
SVC
SVC()
```

In [68]:
```
y_predictions = svc.predict(X_train_set)
```

In [69]:
```
print(f"Accuracy: {accuracy_score(y_train,y_predictions)}")
print(f"Recall: {recall_score(y_train,y_predictions)}")
print(f"Precision: {precision_score(y_train,y_predictions)}")
print(f"Confusion matrix:\n {confusion_matrix(y_train,y_predictions)}")
```

```
Accuracy: 0.9668141592920354
Recall: 0.9529411764705882
Precision: 0.9585798816568047
Confusion matrix:
 [[275   7]
 [  8 162]]
```

Random Forest

In [70]:
```
from sklearn.ensemble import RandomForestClassifier
forest = RandomForestClassifier()
```

In [71]:
```
forest.fit(X_train,y_train)
```

Out[71]:
```
RandomForestClassifier
RandomForestClassifier()
```

In [72]:
```
y_predictions = forest.predict(X_train)
```

In [73]:
```
print(f"Accuracy: {accuracy_score(y_train,y_predictions)}")
print(f"Recall: {recall_score(y_train,y_predictions)}")
print(f"Precision: {precision_score(y_train,y_predictions)}")
print(f"Confusion matrix:\n {confusion_matrix(y_train,y_predictions)}")
plt.show()
```

```
Accuracy: 1.0
Recall: 1.0
Precision: 1.0
Confusion matrix:
 [[282   0]
 [  0 170]]
```

An accuracy of 1.0 (100%) obtained for Decision Tree and Random forest over fits the Train dataset.

Cross Validation (CV) using the recall as scoring metric

In [74]:
```
from sklearn.model_selection import cross_val_score
```

For Decision Tree

In [75]:
```
scores = cross_val_score(dec_tree,X_train,y_train,
                         scoring='recall',
                         cv=10)
np.mean(scores)
```

Out[75]:
```
0.9058823529411765
```

For Logistic Regression

In [76]:
```
scores = cross_val_score(logistic,X_train_set,y_train,
                         scoring='recall',
                         cv=10)
np.mean(scores)
```

Out[76]:
```
0.9235294117647059
```

For SVC

In [77]:
```
scores = cross_val_score(svc,X_train_set,y_train,
                         scoring='recall',
                         cv=10)
np.mean(scores)
```

Out[77]:
```
0.9235294117647058
```

For Random Forest

In [78]:
```
scores = cross_val_score(forest,X_train,y_train,
                         scoring='recall',
                         cv=10)
np.mean(scores)
```

Out[78]:
```
0.9235294117647059
```

The above mean recall scores from the cross validation metric shows the logistic regression having a score of 0.9235 (92.35%), which performs better than the other models tested. Also, the mean recall scores obtained from Decision Tree and Random Forest proves that the model was over fitting the Train dataset

Best Model For Prediction (Logistic Regression)

In [79]:
```
final_prediction = logistic.predict(X_test_set)
```

In [80]:
```
print(f"Accuracy: {accuracy_score(y_test,final_prediction)}")
print(f"Recall: {recall_score(y_test,final_prediction)}")
print(f"Precision: {precision_score(y_test,final_prediction)}")
print(f"Confusion matrix:\n {confusion_matrix(y_test,final_prediction)}")
plt.show()
```

```
Accuracy: 0.9736842105263158
Recall: 0.9512195121951219
Precision: 0.975
Confusion matrix:
 [[72  1]
 [ 2 39]]
```

After running the final prediction model, the best recall score observed was 0.9512(95.12%) with accuracy of 0.97367(97.37%) and a precision of 0.975(97.5%)

CONCLUSION

The study aimed to understand nuclei cells and distinguish between malignant cancer cells and benign cells. It initially involved analyzing 32 attributes, with 30 representing cell characteristics. Following identification and mitigation of multicollinearity among variables, the study focused on 10 features. Exploratory Data Analysis revealed 211 malignant cells and 355 benign cells. Standardization was performed to address variations in data scales.

Unsupervised Machine Learning techniques were employed to uncover patterns and insights. Cluster analysis algorithms, including KMeans, DBScans, Agglomerative, and Dendrograms, were utilised to identify cell clusters, supported by visual representations. The Silhouette scores metric under KMeans, along with analysis from the Dendrogram, confirmed the presence of two clusters, corresponding to the diagnosis of cancer cells as either Malignant (M) or Benign (B). Among the models employed, KMeans emerged as the most effective for cluster analysis, providing accurate predictions for cells categorised as Malignant or Benign.

Furthermore, in the supervised Machine Learning phase, four classification algorithms (Logistic Regression, SVC, Decision Tree Classifier, and Random Forest) were utilized. The models underwent comprehensive evaluation based on average recall scores following 10-fold cross-validation. Logistic Regression demonstrated the highest average recall score of 92.35%, while Decision Tree performed least favorably with a score of 90.5%. Logistic Regression was deemed the most suitable model for final dataset testing based on these scores.

The study successfully distinguished nuclei groups as Malignant or Benign based on cell characteristics.

However, potential limitations encountered during the analysis included some models over fitting the train set data due to imbalance of the number of Malignant and Benign observation.

ETHICAL IMPLICATION ON THE STUDY

For this study, some of the ethical implication might include ensuring adequate informed consent from patients for the use of their biological samples and clinical data, especially concerning the potential risks and benefits of participation. Patient privacy also presents an ethical consideration, necessitating the safeguarding of patient confidentiality through anonymization of data and secure storage practices to prevent unauthorized access.

REFERENCES

Mayo Clinic(2022) *Breast Cancer* Available at: https://www.mayoclinic.org/diseases-conditions/breast-cancer/symptoms-causes/syc-20352470#:~:text=Breast%20cancer%20is%20cancer%20that,far%20more%20common%20in%20women.(Accessed: 25 January 2024)

Processing math: 100%