MACHINE LEARNING ASSIGNMENT-2

(LAB MANUAL)

Name: Pranshu Aggarwal

Roll No: 40576802717

Section: CSE-3

1. PROBLEM STATEMENT

Q.2 Implementation of decision tree on a breast cancer dataset using sklearn in python.

Ans.2

2.ALGORITHM OF ML PROBLEM

Tree-Learning (TR, Target, Attr)
    TR: training examples
    Target: target attribute
    Attr: set of descriptive attributes
{
    Create a Root node for the tree.
    If TR have the same target attribute value t_i,
      Then Return the single-node tree, i.e. Root, with target attribute = t_i
    If Attr = empty (i.e. there is no descriptive attributes available),
      Then Return the single-node tree, i.e. Root, with most common value of Target in TR
    Otherwise
    {
        Select attribute A from Attr that best classify TR based on an entropy-based measure
        Set A the attribute for Root
        For each legal value of A, v_i, do
        {
            Add a branch below Root, corresponding to A = v_i
            Let TR_{vi} **be the subset of TR that have** A = v_i
            If TR_{vi} is empty,
              Then add a leaf node below the branch with target value = most common value of
                  Target in TR
              Else below the branch, add the subtree learned by
                  Tree-Learning(TR_{vi}, Target, Attr-{A})
        }
    }
    Return (Root)
}

3.Program Code Snippet

(i) Loading Dataset

In [3]:
```
import pandas as pd

df = pd.read_csv("cancer.csv - cancer.csv.csv")
df
```

Out[3]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	...
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	...
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	...
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	...
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	...
...	...	...	...	...	...	...	...	...	...	...	...
564	926424	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	...
565	926682	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	...
566	926954	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	...
567	927241	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	...
568	92751	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	...

569 rows × 32 columns

Viewing the Data:

One of the most used method for getting a quick overview of the DataFrame, is the head() method.

The head() method returns the headers and a specified number of rows, starting from the top.

In [4]:
```
df.head(10)
```

Out[4]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	ra
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	...	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	...	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	...	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	...	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	...	
5	843786	M	12.45	15.70	82.57	477.1	0.12780	0.17000	0.15780	0.08089	...	
6	844359	M	18.25	19.98	119.60	1040.0	0.09463	0.10900	0.11270	0.07400	...	
7	84458202	M	13.71	20.83	90.20	577.9	0.11890	0.16450	0.09366	0.05985	...	
8	844981	M	13.00	21.82	87.50	519.8	0.12730	0.19320	0.18590	0.09353	...	
9	84501001	M	12.46	24.04	83.97	475.9	0.11860	0.23960	0.22730	0.08543	...	

10 rows × 32 columns

In [ ]:
```
#to read the last end of data
df.tail()
```

Out[3]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	t
564	926424	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	...	
565	926682	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	...	
566	926954	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	...	
567	927241	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	...	
568	92751	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	...	

5 rows × 33 columns

Info About the Data:

The DataFrames object has a method called info(), that gives you more information about the data set.

In [5]:
```
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
```

In [6]:
```
df.shape
```
Out[6]:
```
(569, 32)
```

In [7]:
```
#print all the columns of dataset
df.columns.values
```
Out[7]:
```
array(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean',
       'concavity_mean', 'concave points_mean', 'symmetry_mean',
       'fractal_dimension_mean', 'radius_se', 'texture_se',
       'perimeter_se', 'area_se', 'smoothness_se', 'compactness_se',
       'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst'], dtype=object)
```

Finding Relationships:

A great aspect of the Pandas module is the corr() method.

The corr() method calculates the relationship between each column in your data set.

In [8]:
```
df.corr()
```
Out[8]:

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	con points_n
id	1.000000	0.074626	0.099770	0.073159	0.096893	-0.012968	0.000096	0.050080	0.04
radius_mean	0.074626	1.000000	0.323782	0.997855	0.987357	0.170581	0.506124	0.676764	0.82
texture_mean	0.099770	0.323782	1.000000	0.329533	0.321086	-0.023389	0.236702	0.302418	0.29
perimeter_mean	0.073159	0.997855	0.329533	1.000000	0.986507	0.207278	0.556936	0.716136	0.85
area_mean	0.096893	0.987357	0.321086	0.986507	1.000000	0.177028	0.498502	0.685983	0.82
smoothness_mean	-0.012968	0.170581	-0.023389	0.207278	0.177028	1.000000	0.659123	0.521984	0.55
compactness_mean	0.000096	0.506124	0.236702	0.556936	0.498502	0.659123	1.000000	0.883121	0.83
concavity_mean	0.050080	0.676764	0.302418	0.716136	0.685983	0.521984	0.883121	1.000000	0.92
concave points_mean	0.044158	0.822529	0.293464	0.850977	0.823269	0.553695	0.831135	0.921391	1.00
symmetry_mean	-0.022114	0.147741	0.071401	0.183027	0.151293	0.557775	0.602641	0.500667	0.46
fractal_dimension_mean	-0.052511	-0.311631	-0.076437	-0.261477	-0.283110	0.584792	0.565369	0.336783	0.16
radius_se	0.143048	0.679090	0.275869	0.691765	0.732562	0.301467	0.497473	0.631925	0.69
texture_se	-0.007526	-0.097317	0.386358	-0.086761	-0.066280	0.068406	0.046205	0.076218	0.02
perimeter_se	0.137331	0.674172	0.281673	0.693135	0.726628	0.296092	0.548905	0.660391	0.71
area_se	0.177742	0.735864	0.259845	0.744983	0.800086	0.246552	0.455653	0.617427	0.69
smoothness_se	0.096781	-0.222600	0.006614	-0.202694	-0.166777	0.332375	0.135299	0.098564	0.02
compactness_se	0.033961	0.206000	0.191975	0.250744	0.212583	0.318943	0.738722	0.670279	0.49
concavity_se	0.055239	0.194204	0.143293	0.228082	0.207660	0.248396	0.570517	0.691270	0.43
concave points_se	0.078768	0.376169	0.163851	0.407217	0.372320	0.380676	0.642262	0.683260	0.61
symmetry_se	-0.017306	-0.104321	0.009127	-0.081629	-0.072497	0.200774	0.229977	0.178009	0.09
fractal_dimension_se	0.025725	-0.042641	0.054458	-0.005523	-0.019887	0.283607	0.507318	0.449301	0.25
radius_worst	0.082405	0.969539	0.352573	0.969476	0.962746	0.213120	0.535315	0.688236	0.83
texture_worst	0.064720	0.297008	0.912045	0.303038	0.287489	0.036072	0.248133	0.299879	0.29
perimeter_worst	0.079986	0.965137	0.358040	0.970387	0.959120	0.238853	0.590210	0.729565	0.85
area_worst	0.107187	0.941082	0.343546	0.941550	0.959213	0.206718	0.509604	0.675987	0.80
smoothness_worst	0.010338	0.119616	0.077503	0.150549	0.123523	0.805324	0.565541	0.448822	0.45
compactness_worst	-0.002968	0.413463	0.277830	0.455774	0.390410	0.472468	0.865809	0.754968	0.66
concavity_worst	0.023203	0.526911	0.301025	0.563879	0.512606	0.434926	0.816275	0.884103	0.75
concave points_worst	0.035174	0.744214	0.295316	0.771241	0.722017	0.503053	0.815573	0.861323	0.91
symmetry_worst	-0.044224	0.163953	0.105008	0.189115	0.143570	0.394309	0.510223	0.409464	0.37
fractal_dimension_worst	-0.029866	0.007066	0.119205	0.051019	0.003738	0.499316	0.687382	0.514930	0.36

In [9]:
```
#check for the null value
df.isnull().sum()
```

Out[9]:
```
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
```

Result Explained:

The Result of the corr() method is a table with a lot of numbers that represents how well the relationship is between two columns.

The number varies from -1 to 1.

1 means that there is a 1 to 1 relationship (a perfect correlation), and for this data set, each time a value went up in the first column, the other one went up as well.

0.9 is also a good relationship, and if you increase one value, the other will probably increase as well.

-0.9 would be just as good relationship as 0.9, but if you increase one value, the other will probably go down.

0.2 means NOT a good relationship, meaning that if one value goes up does not mean that the other will.

(ii)Preprocessing/Cleaning of Dataset

Data Cleaning:

Data cleaning means fixing bad data in your data set.

Bad data could be:

-Empty cells

-Data in wrong format

-Wrong data

-Duplicates

Empty Cells

Empty cells can potentially give you a wrong result when you analyze data.

Remove Rows

One way to deal with empty cells is to remove rows that contain empty cells.

This is usually OK, since data sets can be very big, and removing a few rows will not have a big impact on the result.

In [10]:
```python
for i in df.columns:
    print(i)
    print(df[i].value_counts())
    print('----------------------*********-------------------------')
```

```
id
883263     1
906564     1
89122      1
9013579    1
868682     1
          ..
874158     1
914062     1
918192     1
872113     1
875878     1
Name: id, Length: 569, dtype: int64
----------------------*********-------------------------
diagnosis
B    357
M    212
Name: diagnosis, dtype: int64
----------------------*********-------------------------
```

In [11]:
```python
df['diagnosis'].value_counts()
```

Out[11]:
```
B    357
M    212
Name: diagnosis, dtype: int64
```

In [21]:
```python
df= df.drop(["id"], axis = 1,errors='ignore')
df
```

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	...	radiu
0	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	...	
1	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	...	
2	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	...	
3	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	...	
4	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	...	
...	...	...	...	...	...	...	...	...	...	...	...	
564	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	...	
565	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	...	
566	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	...	
567	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	...	

In [26]:
```python
df = df.drop(["Unnamed: 32"], axis = 1,errors='ignore')
df
```

Out[26]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_me
0	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.24
1	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.18
2	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.20
3	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.25
4	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.18
...	...	...	...	...	...	...	...	...	...	
564	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.17
565	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.17
566	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.15
567	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.23
568	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.15

569 rows × 31 columns

(iii) Visualization

Visualization

it is import to see that counts of different type of cancer

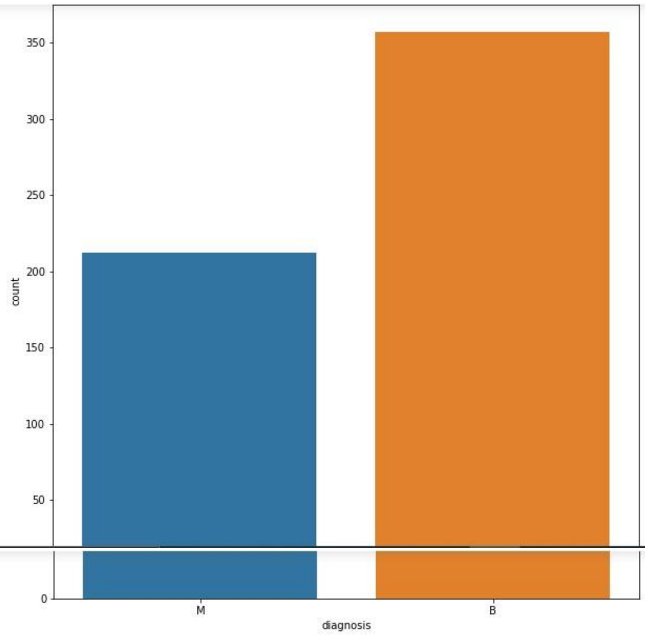
In [27]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

In [28]:
```
benign, malignant=df['diagnosis'].value_counts()
print("No of Benign cell", benign)
print("No of malignant cell", malignant)
```

```
No of Benign cell 357
No of malignant cell 212
```

In [29]:
```
plt.figure(figsize=(10,10))
sns.countplot(df['diagnosis'])
plt.show()
```

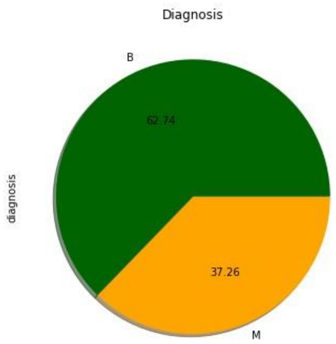
```
c:\users\dell\appdata\local\programs\python\python37\lib\site-packages\seaborn\_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```



In [30]:
```
print("% of Benign cell is ", benign*100/len(df))
print("% of Malignant cell is ", malignant*100/len(df))
```

```
% of Benign cell is  62.74165202108963
% of Malignant cell is  37.25834797891037
```

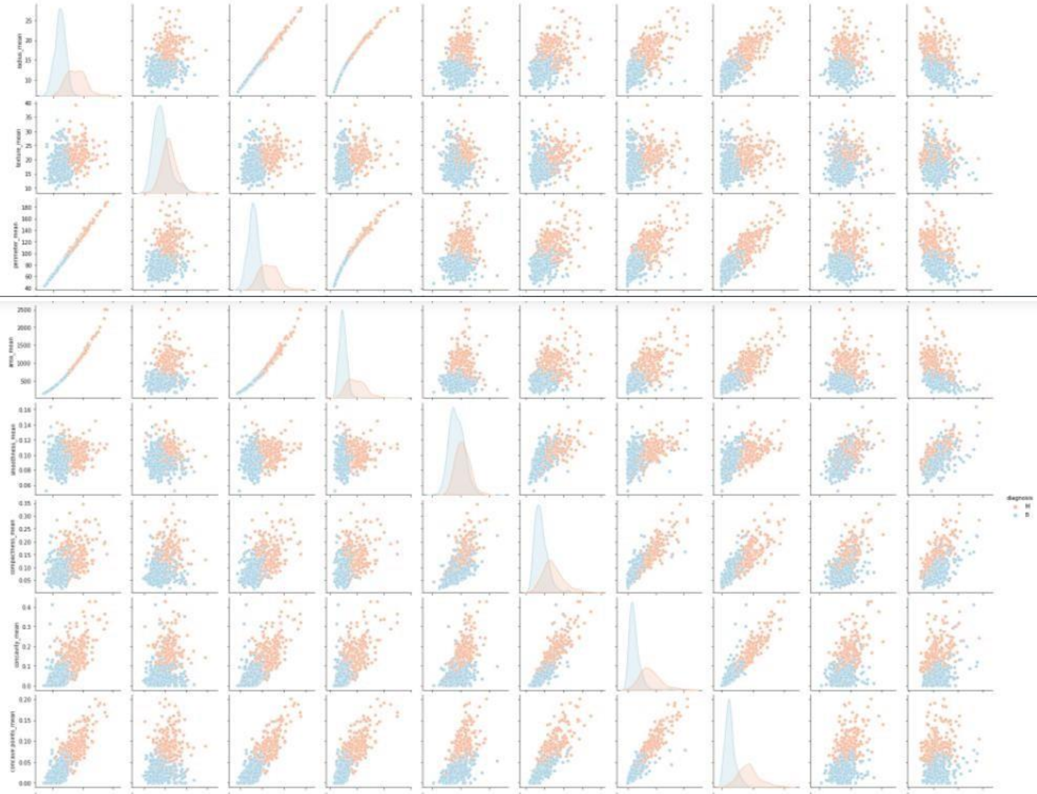
In [31]:
```
df.diagnosis.value_counts().plot(kind='pie',shadow=True,colors=('darkgreen','orange'),autopct='%.2f',figsize=(8,6))
plt.title('Diagnosis')
plt.show()
```



Pairplot helps to plot among the most useful feature

In [32]:
```
cols=['diagnosis','radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean','smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean']
plt.figure(figsize=(10,10))
sns.pairplot(data=df[cols],hue='diagnosis', palette='RdBu')
```

Out[32]:
```
<seaborn.axisgrid.PairGrid at 0x199a4697e48>

<Figure size 720x720 with 0 Axes>
```



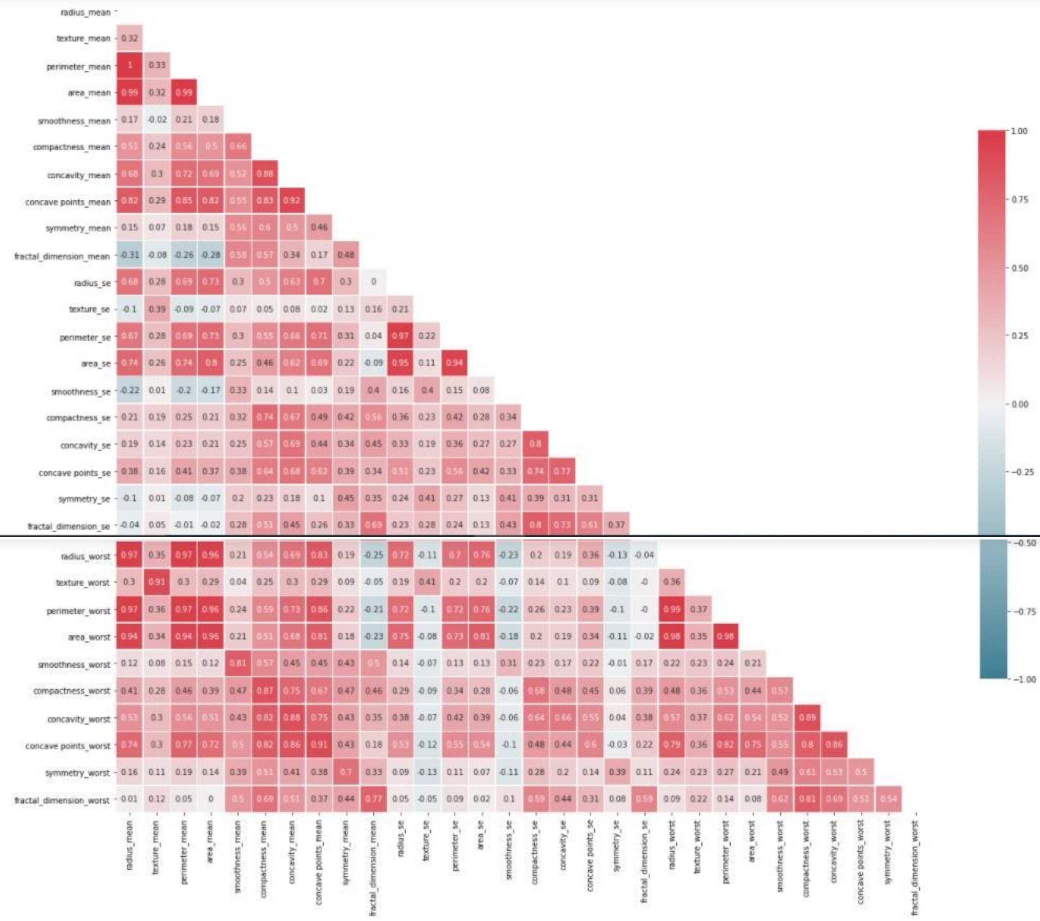
Heatmap:

To find the most correlated features

In [33]:
```
import numpy as np
```

In [34]:
```
#generate the corelation matrix
corr=df.corr().round(2)
#mask for the upper triangle
mask=np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)]
# Set figure size
f, ax = plt.subplots(figsize=(20, 20))

#define custom colormap
cmap=sns.diverging_palette(220,10, as_cmap=True)

#draw the heatmap
sns.heatmap(corr, mask=mask, cmap=cmap, vmin=-1, vmax=1, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5}, annot=True)

plt.tight_layout()
```


```
In [35]: # Generate and visualize the correlation matrix
         corr = df.corr().round(2)

         # Mask for the upper triangle
         mask = np.zeros_like(corr, dtype=np.bool)
         mask[np.triu_indices_from(mask)] = True

         # Set figure size
         f, ax = plt.subplots(figsize=(20, 20))

         # Define custom colormap
         cmap = sns.diverging_palette(220, 10, as_cmap=True)

         # Draw the heatmap
         sns.heatmap(corr, mask=mask, cmap=cmap, vmin=-1, vmax=1, center=0,
                     square=True, linewidths=.5, cbar_kws={"shrink": .5}, annot=True)

         plt.tight_layout()
```



In [36]:
```
M = df[df.diagnosis == "M"]
M.head()
```

Out[36]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean
0	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419
1	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812
2	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069
3	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597
4	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809

5 rows × 31 columns

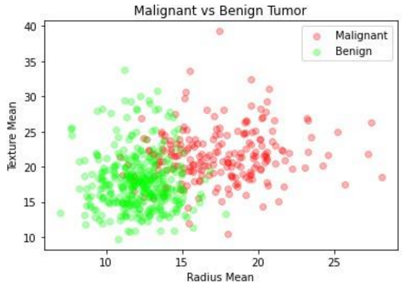
In [37]:
```
B = df[df.diagnosis == "B"]
B.head()
```

Out[37]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mea
19	B	13.540	14.36	87.46	566.3	0.09779	0.08129	0.06664	0.047810	0.188
20	B	13.080	15.71	85.63	520.0	0.10750	0.12700	0.04568	0.031100	0.196
21	B	9.504	12.44	60.34	273.9	0.10240	0.06492	0.02956	0.020760	0.181
37	B	13.030	18.42	82.61	523.8	0.08983	0.03766	0.02562	0.029230	0.146
46	B	8.196	16.84	51.71	201.9	0.08600	0.05943	0.01588	0.005917	0.176

5 rows × 31 columns

In [38]:
```
plt.title("Malignant vs Benign Tumor")
plt.xlabel("Radius Mean")
plt.ylabel("Texture Mean")
plt.scatter(M.radius_mean, M.texture_mean, color = "red", label = "Malignant", alpha = 0.3)
plt.scatter(B.radius_mean, B.texture_mean, color = "lime", label = "Benign", alpha = 0.3)
plt.legend()
plt.show()
```



(iii)ML ALGORITHM IMPLEMENTATION OF PREDICTION OR COMPARISON

Meaning Of Decision Tree Algorithm

Decision tree models where the target variable uses a discrete set of values are classified as Classification Trees.

In these trees, each node, or leaf, represent class labels while the branches represent conjunctions of features leading to class labels.

A decision tree where the target variable takes a continuous value, usually numbers, are called Regression Trees.

The two types are commonly referred to together at CART (Classification and Regression Tree).

image.png

Decision Tree with Sklearn

In [39]:
```
feature_cols = ['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean','smoothness_mean', 'compactness_mean', 'concavity_me
```

In [40]:
```
x = df[feature_cols]
y = df.diagnosis.values
```

In [41]:
```
x.head()
```

Out[41]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dir
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	

What is normalization?

Normalization is a technique often applied as part of data preparation for machine learning. The goal of normalization is to change the values of numeric columns in the dataset to use a common scale, without distorting differences in the ranges of values or losing information. Normalization is also required for some algorithms to model the data correctly.

MinMax:

The min-max normalizer linearly rescales every feature to the [0,1] interval.

Rescaling to the [0,1] interval is done by shifting the values of each feature so that the minimal value is 0, and then dividing by the new maximal value (which is the difference between the original maximal and minimal values).

The values in the column are transformed using the following formula:

normalization using the min-max function

image.png

In [42]:
```
# Normalization:
x = (x - np.min(x)) / (np.max(x) - np.min(x))
x
```

Out[42]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_
0	0.521037	0.022658	0.545989	0.363733	0.593753	0.792037	0.703140	0.731113	0.686364	
1	0.643144	0.272574	0.615783	0.501591	0.289880	0.181768	0.203608	0.348757	0.379798	
2	0.601496	0.390260	0.595743	0.449417	0.514309	0.431017	0.462512	0.635686	0.509596	
3	0.210090	0.360839	0.233501	0.102906	0.811321	0.811361	0.565604	0.522863	0.776263	
4	0.629893	0.156578	0.630986	0.489290	0.430351	0.347893	0.463918	0.518390	0.378283	
...	...	...	...	...	...	...	...	...	...	
564	0.690000	0.428813	0.678668	0.566490	0.526948	0.296055	0.571462	0.690358	0.336364	
565	0.622320	0.626987	0.604036	0.474019	0.407782	0.257714	0.337395	0.486630	0.349495	
566	0.455251	0.621238	0.445788	0.303118	0.288165	0.254340	0.216753	0.263519	0.267677	
567	0.644564	0.663510	0.665538	0.475716	0.588336	0.790197	0.823336	0.755467	0.675253	
568	0.036869	0.501522	0.028540	0.015907	0.000000	0.074351	0.000000	0.000000	0.266162	

569 rows × 10 columns

In [43]:
```
from sklearn.model_selection import train_test_split

#for checking testing results
from sklearn.metrics import classification_report, confusion_matrix

#for visualizing tree
from sklearn.tree import plot_tree

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 0)

print("Training split input- ", x_train.shape)
print("Testing split input- ", x_test.shape)
```

```
Training split input-  (455, 10)
Testing split input-  (114, 10)
```

In [44]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [45]:
```
dt = DecisionTreeClassifier()
```

In [46]:
```
dt.fit(x_train, y_train)
```

Out[46]:
```
DecisionTreeClassifier()
```

Testing

Precision — Also called Positive predictive value

The ratio of correct positive predictions to the total predicted positives.

Recall — Also called Sensitivity, Probability of Detection, True Positive Rate

The ratio of correct positive predictions to the total positives examples.

(iv)FINAL GRAPH ROC/AUC/CONFUSION MATRIX

Confusion matrix

confusion matrix usage to evaluate the quality of the output of a classifier. The diagonal elements represent the number of points for which the predicted label is equal to the true label, while off-diagonal elements are those that are mislabeled by the classifier. The higher the diagonal values of the confusion matrix the better, indicating many correct predictions.

image.png

image.png

Accuracy

Talking about accuracy, our favourite metric!

Accuracy is defined as the ratio of correctly predicted examples by the total examples.

image.png

image-2.png

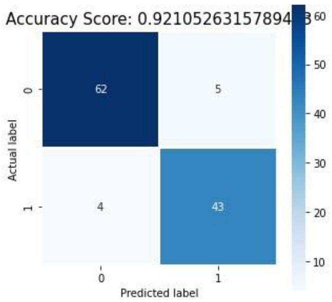
In [47]:
```
y_pred = dt.predict(x_test)
print("Classification report - \n", classification_report(y_test,y_pred))
```

```
Classification report -
               precision    recall  f1-score   support

           B       0.94      0.93      0.93        67
           M       0.90      0.91      0.91        47

    accuracy                           0.92       114
   macro avg       0.92      0.92      0.92       114
weighted avg       0.92      0.92      0.92       114
```

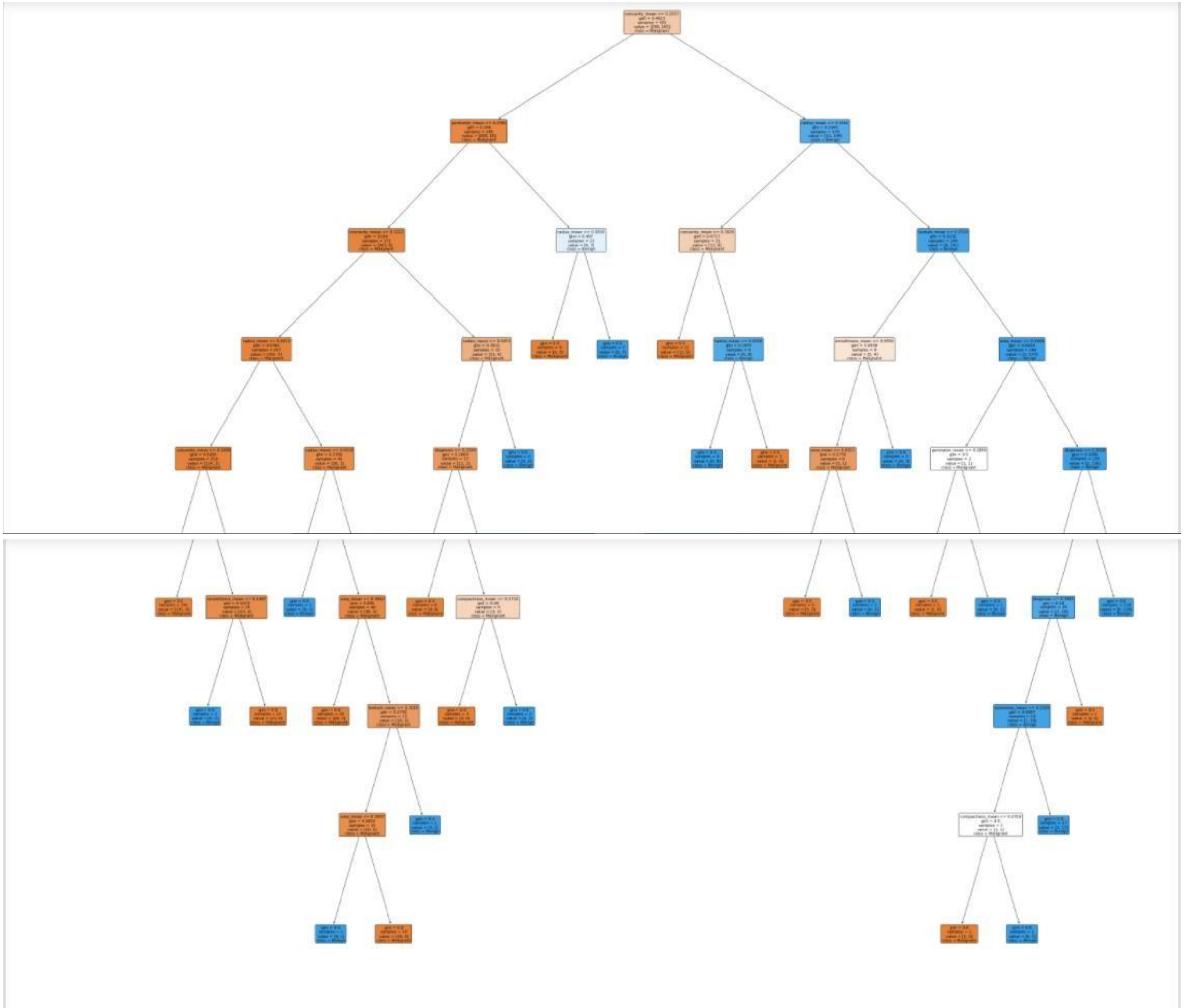
In [48]:
```
cm=confusion_matrix(y_test,y_pred)
cm
```

Out[48]:
```
array([[62,  5],
       [ 4, 43]], dtype=int64)
```

In [49]:
```
plt.figure(figsize=(5,5))

sns.heatmap(data=cm,linewidths=1.0, annot=True,square = True,  cmap = 'Blues')

plt.ylabel('Actual label')
plt.xlabel('Predicted label')

all_sample_title = 'Accuracy Score: {0}'.format(dt.score(x_test, y_test))
plt.title(all_sample_title, size = 15)

plt.savefig("D:/accu.png")
```



In [50]:
```
# Visualising the graph without the use of graphviz

plt.figure(figsize = (50,50))
dec_tree = plot_tree(decision_tree=dt, feature_names = df.columns, class_names =["Malignant", "Benign"] , filled = True , precisi

plt.savefig("D:/dt.png")
```



4.GITHUB LINK

https://github.com/pranshuag9/machine-learning-lab/blob/main/lab2/Lab1_DT_final.ipynb