In [1]:
```
# Importing libraries
import pandas as pd
from IPython.display import Markdown, display
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns; sns.set(style="ticks", color_codes=True)

def printmd(string):
    display(Markdown(string))
```

In [2]:
```
data = pd.read_csv('data.csv')
```

1. Preliminary Data Analysis

In [3]:
```
# Setting all the categorical columns to type category
for col in set(data.columns) - set(data.describe().columns):
    data[col] = data[col].astype('category')

printmd('## 1.1. Columns and their types')
print(data.info())
```

1.1. Columns and their types

```
<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    category
 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: category(1), float64(30), int64(1)
memory usage: 138.6 KB
None
```

In [4]:

```python
# Top 5 records
printmd('## 1.2. Data')
data.head()
```

1.2. Data

Out[4]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_me
0	842302	M	17.99	10.38	122.80	1001.0	0.118
1	842517	M	20.57	17.77	132.90	1326.0	0.084
2	84300903	M	19.69	21.25	130.00	1203.0	0.109
3	84348301	M	11.42	20.38	77.58	386.1	0.142
4	84358402	M	20.29	14.34	135.10	1297.0	0.100

5 rows × 32 columns

In [5]:
```
printmd('## 1.3. Summary Statistics')

data.describe()
```

1.3. Summary Statistics

Out[5]:

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mea
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.00000
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0.09636
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	0.01406
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	0.05263
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	0.08637
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	0.09587
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	0.10530
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	0.16340

8 rows × 31 columns

In [6]:
```
printmd('## 1.4. Missing values')

data.isnull().sum()
```

1.4. Missing values

Out[6]:
```
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 [7]:
```
printmd('## 1.5. Clean Data')

# Cleaning and modifying the data - droping 'id' cloumn/attribitues
data.drop('id',axis=1, inplace=True)
data.head()
```

1.5. Clean Data

Out[7]:

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

5 rows × 31 columns

In [8]:
```
printmd('## 1.6. Correlation Matrix')

display(data.corr())

printmd('We see that none of the columns are highly correlated.')
```

1.6. Correlation Matrix

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_me
radius_mean	1.000000	0.323782	0.997855	0.987357	0.1705
texture_mean	0.323782	1.000000	0.329533	0.321086	-0.0233
perimeter_mean	0.997855	0.329533	1.000000	0.986507	0.2072
area_mean	0.987357	0.321086	0.986507	1.000000	0.1770
smoothness_mean	0.170581	-0.023389	0.207278	0.177028	1.0000
compactness_mean	0.506124	0.236702	0.556936	0.498502	0.6591
concavity_mean	0.676764	0.302418	0.716136	0.685983	0.5219
concave points_mean	0.822529	0.293464	0.850977	0.823269	0.5536
symmetry_mean	0.147741	0.071401	0.183027	0.151293	0.5577
fractal_dimension_mean	-0.311631	-0.076437	-0.261477	-0.283110	0.5847
radius_se	0.679090	0.275869	0.691765	0.732562	0.3014
texture_se	-0.097317	0.386358	-0.086761	-0.066280	0.0684
perimeter_se	0.674172	0.281673	0.693135	0.726628	0.2960
area_se	0.735864	0.259845	0.744983	0.800086	0.2465
smoothness_se	-0.222600	0.006614	-0.202694	-0.166777	0.3323
compactness_se	0.206000	0.191975	0.250744	0.212583	0.3189
concavity_se	0.194204	0.143293	0.228082	0.207660	0.2483
concave points_se	0.376169	0.163851	0.407217	0.372320	0.3806
symmetry_se	-0.104321	0.009127	-0.081629	-0.072497	0.2007
fractal_dimension_se	-0.042641	0.054458	-0.005523	-0.019887	0.2836
radius_worst	0.969539	0.352573	0.969476	0.962746	0.2131
texture_worst	0.297008	0.912045	0.303038	0.287489	0.0360
perimeter_worst	0.965137	0.358040	0.970387	0.959120	0.2388
area_worst	0.941082	0.343546	0.941550	0.959213	0.2067
smoothness_worst	0.119616	0.077503	0.150549	0.123523	0.8053
compactness_worst	0.413463	0.277830	0.455774	0.390410	0.4724
concavity_worst	0.526911	0.301025	0.563879	0.512606	0.4349
concave points_worst	0.744214	0.295316	0.771241	0.722017	0.5030
symmetry_worst	0.163953	0.105008	0.189115	0.143570	0.3943
fractal_dimension_worst	0.007066	0.119205	0.051019	0.003738	0.4993

30 rows × 30 columns

We see that none of the columns are highly correlated.

In [9]:
```
printmd('## 1.7. Mapping the Categorical Attr.')

# Mapping Benign to 0 and Malignant to 1
data['diagnosis'] = data['diagnosis'].map({'M':1,'B':0})
#Check the data stats
data.describe()
```

1.7. Mapping the Categorical Attr.

Out[9]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.

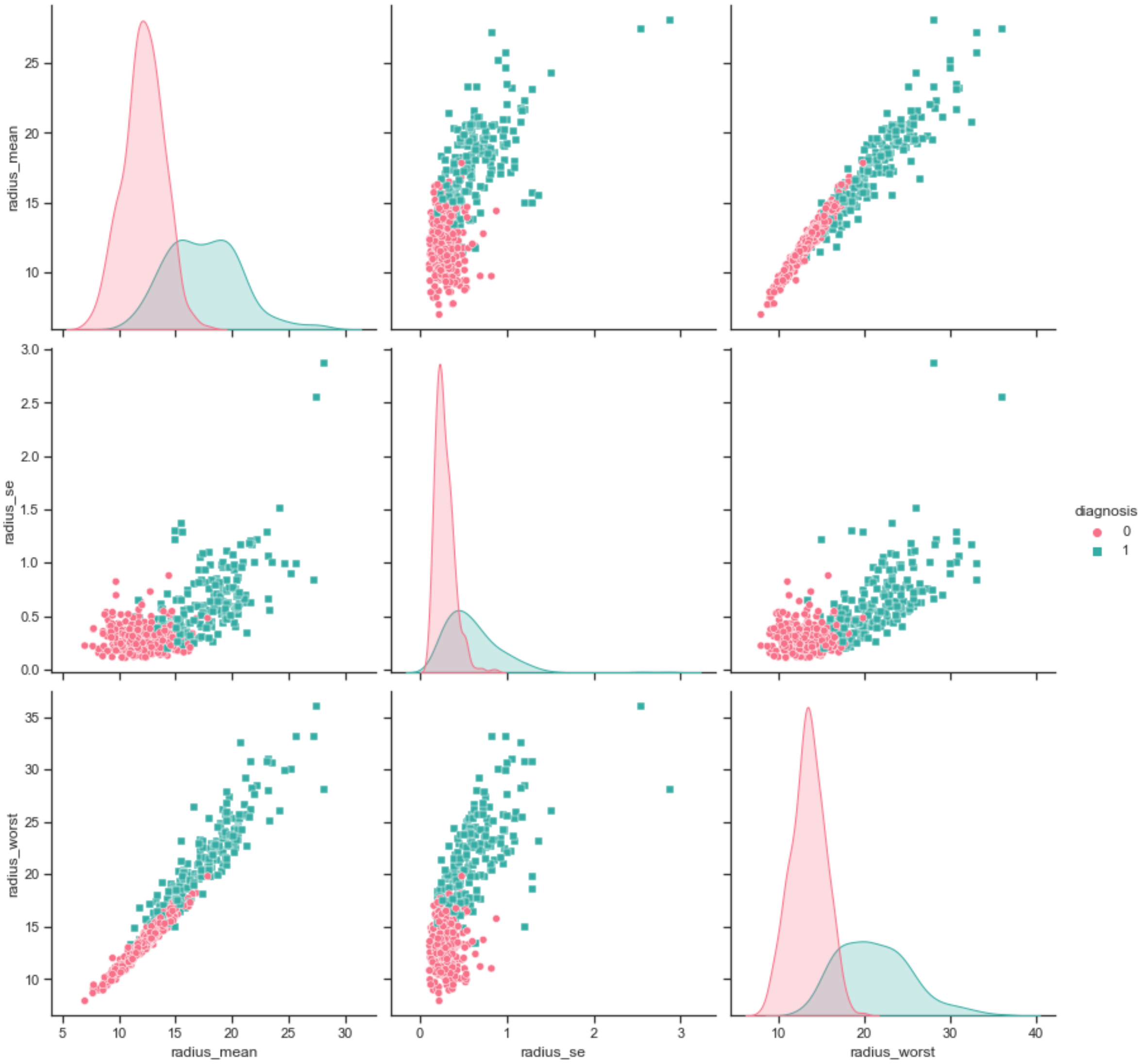
8 rows × 30 columns

2. Exploratory Analysis

In [10]:
```
radius = data[['radius_mean','radius_se','radius_worst','diagnosis']]

# M : 1, B: 0
sns.pairplot(radius, hue='diagnosis',palette="husl", markers=["o", "s"],size=4)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\axisgrid.py:2076: UserWarning:
The `size` parameter has been renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
```

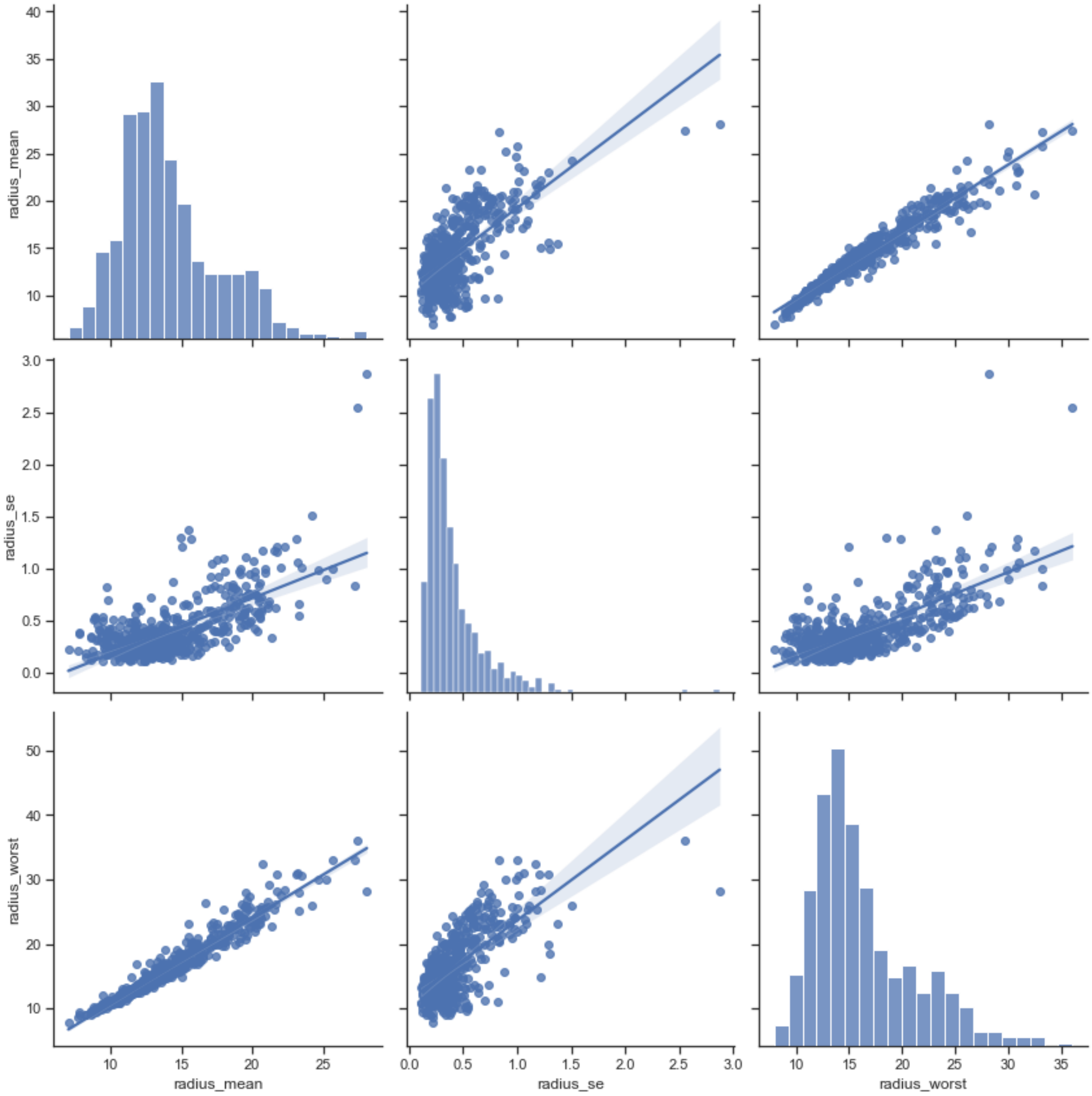
Out[10]:
```
<seaborn.axisgrid.PairGrid at 0x17d678dc5e0>
```



In [11]:
```
sns.pairplot(radius,kind="reg",size=4)
```

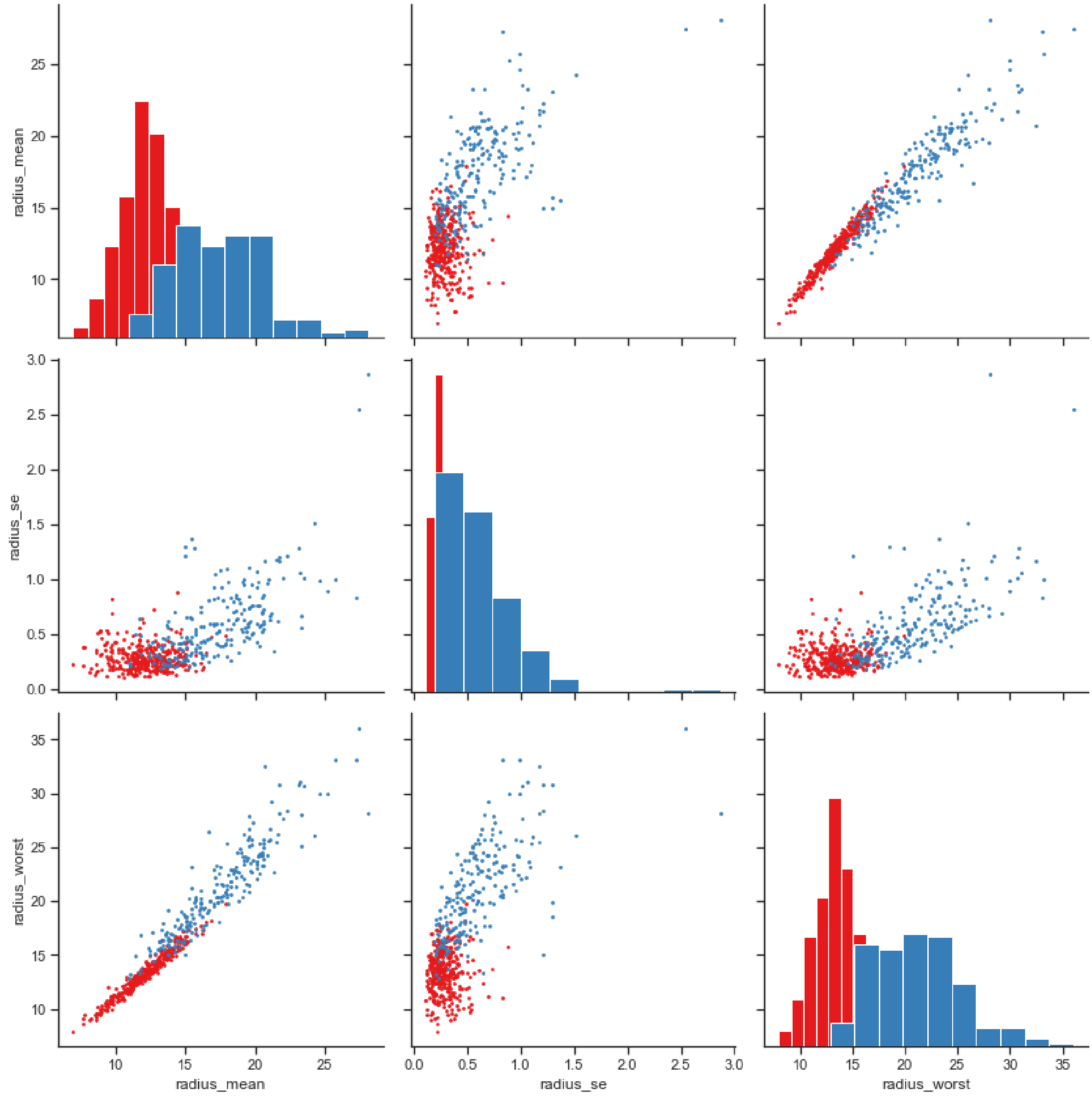
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\axisgrid.py:2076: UserWarning:
The `size` parameter has been renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
```

Out[11]:
```
<seaborn.axisgrid.PairGrid at 0x17d67918610>
```



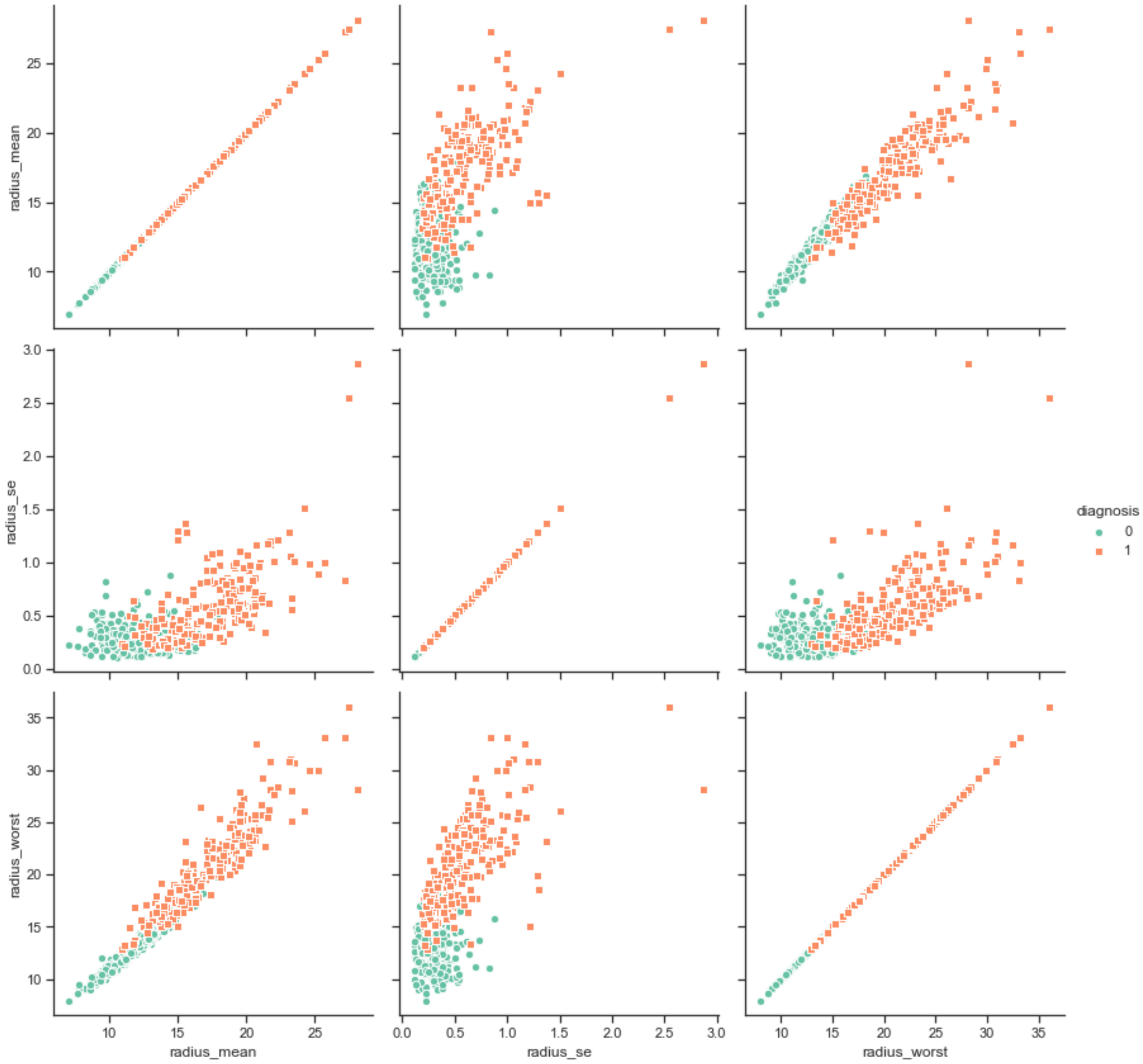
In [12]:
```
g = sns.PairGrid(radius,hue='diagnosis', palette="Set1",size=4)
g = g.map_diag(plt.hist)
g = g.map_offdiag(plt.scatter, s = 3)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\axisgrid.py:1209: UserWarning:
The `size` parameter has been renamed to `height`; please update your code.
  warnings.warn(UserWarning(msg))
```



In [13]:
```
g = sns.PairGrid(radius, hue="diagnosis", palette="Set2",size=4,hue_kws={"marker":
g = g.map(plt.scatter, linewidths=1, edgecolor="w", s=40)
g = g.add_legend()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\axisgrid.py:1209: UserWarning:
The `size` parameter has been renamed to `height`; please update your code.
  warnings.warn(UserWarning(msg))
```



In [14]:
```
printmd('## 2.1. Scaling the Dataset')
from sklearn import preprocessing
# Scaling the dataset
datas = pd.DataFrame(preprocessing.scale(data.iloc[:,1:32]))
datas.columns = list(data.iloc[:,1:32].columns)
datas['diagnosis'] = data['diagnosis']
datas.head()
```

2.1. Scaling the Dataset

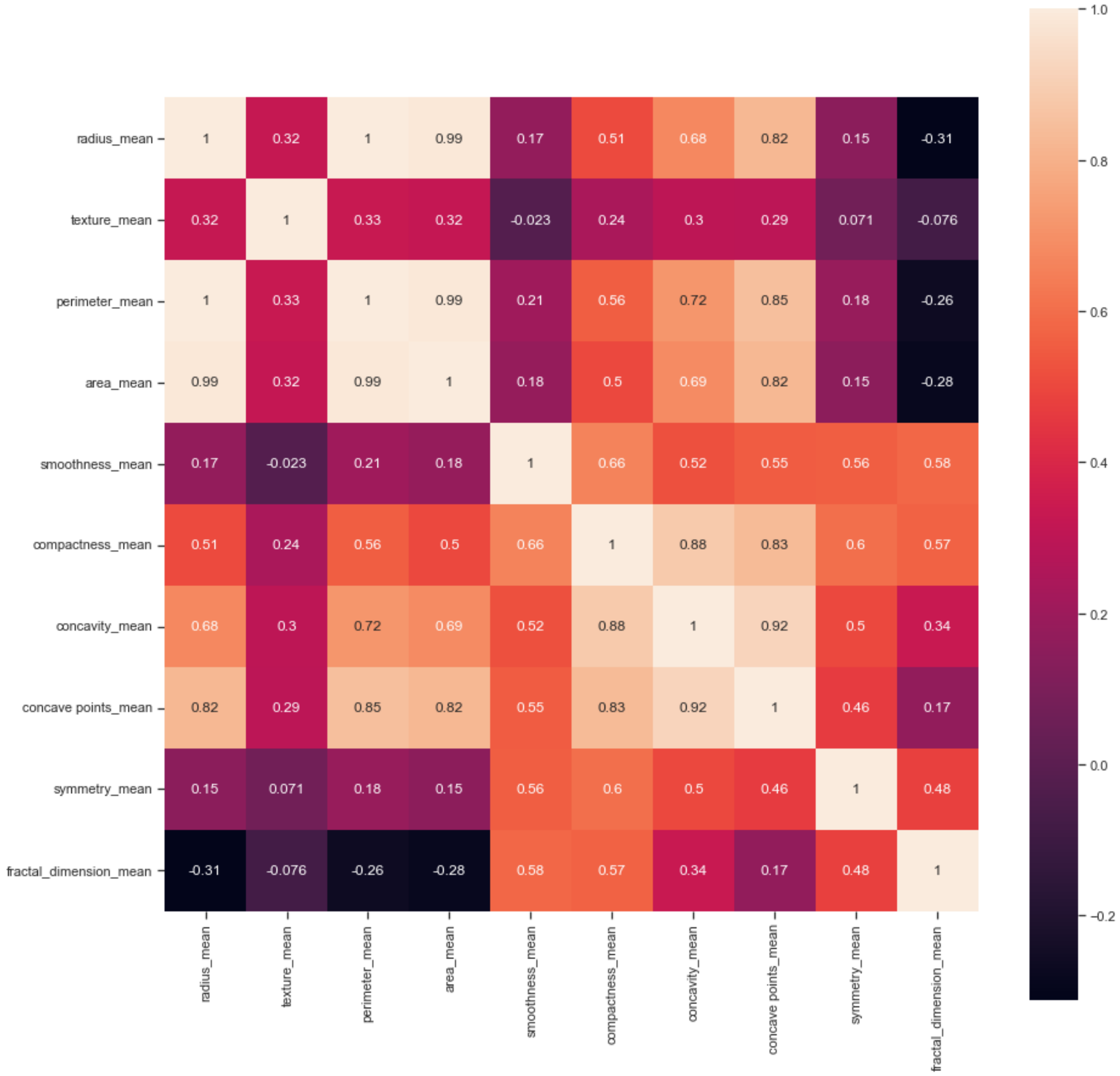
Out[14]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mea
0	1.097064	-2.073335	1.269934	0.984375	1.568466	3.28351
1	1.829821	-0.353632	1.685955	1.908708	-0.826962	-0.48707
2	1.579888	0.456187	1.566503	1.558884	0.942210	1.05292
3	-0.768909	0.253732	-0.592687	-0.764464	3.283553	3.40290
4	1.750297	-1.151816	1.776573	1.826229	0.280372	0.53934

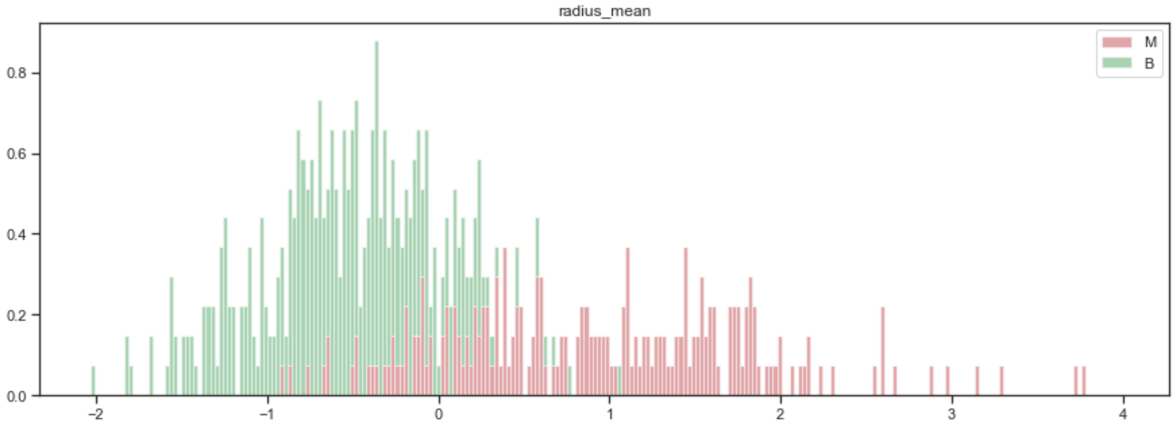
5 rows × 31 columns

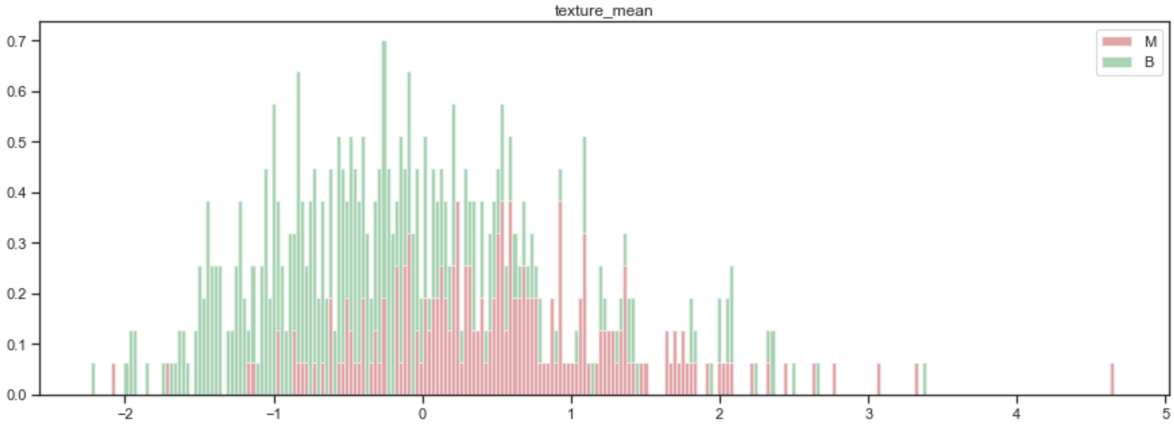
In [15]:
```
#draw a heatmap between mean features and diagnosis
features_mean = ['radius_mean','texture_mean','perimeter_mean','area_mean','smoothn
```

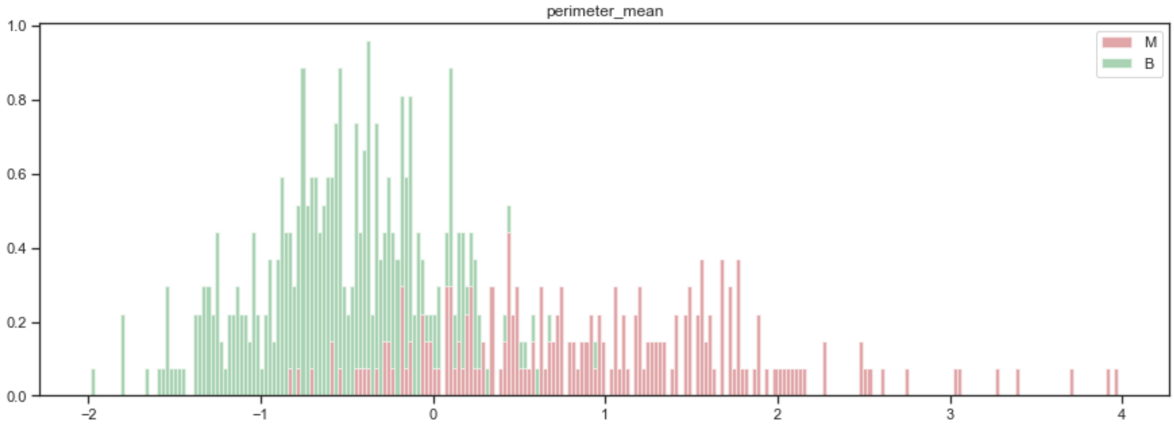
```
plt.figure(figsize=(15,15))
heat = sns.heatmap(datas[features_mean].corr(), vmax=1, square=True, annot=True)
```

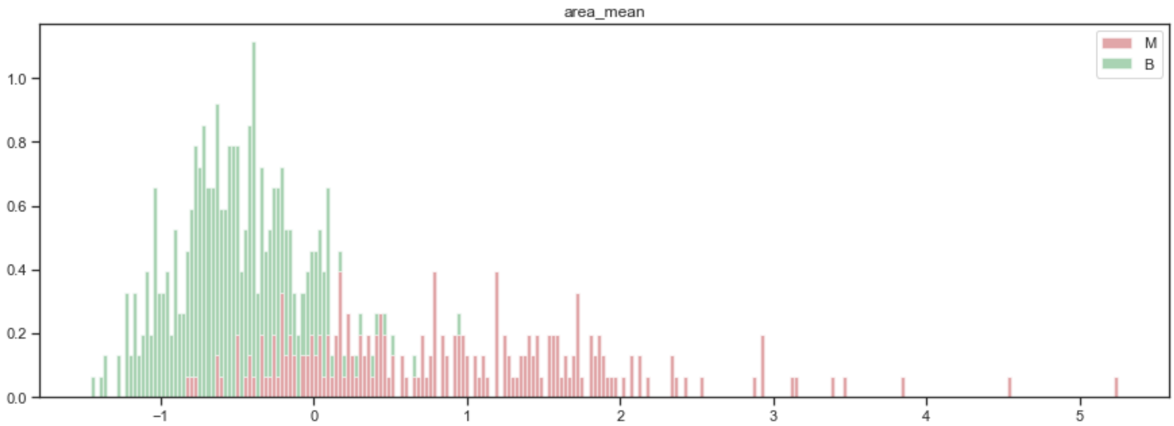


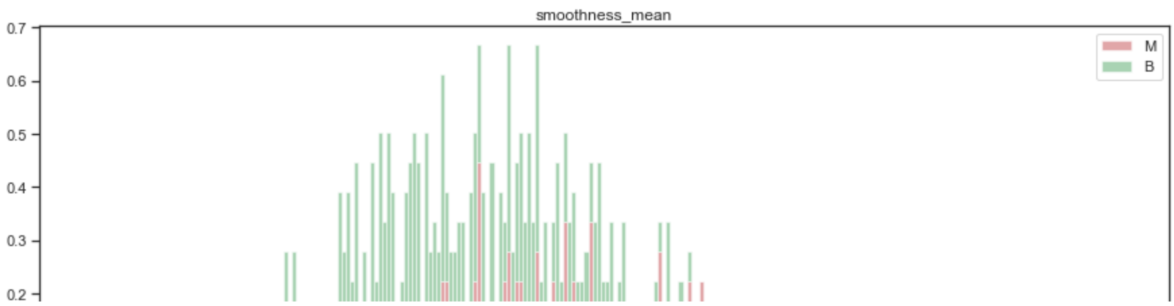
In [16]:
```
import numpy as np
# Splitting the dataset into malignant and benign.
dataMalignant=datas[datas['diagnosis'] ==1]
dataBenign=datas[datas['diagnosis'] ==0]

#Plotting these features as a histogram
fig, axes = plt.subplots(nrows=10, ncols=1, figsize=(15,60))
for idx,ax in enumerate(axes):
    ax.figure
    binwidth= (max(datas[features_mean[idx]]) - min(datas[features_mean[idx]]))/25(
    ax.hist([dataMalignant[features_mean[idx]],dataBenign[features_mean[idx]]], bi
    ax.legend(loc='upper right')
    ax.set_title(features_mean[idx])
plt.show()
```

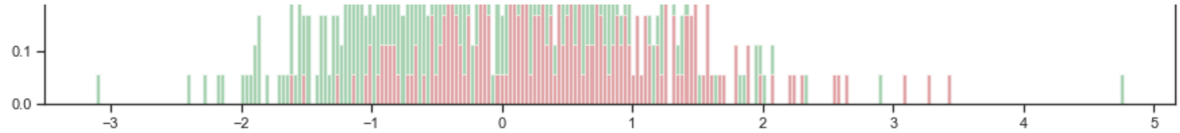

radius_mean
M
B
0.0
0.2
0.4
0.6
0.8
−2
−1
0
1
2
3
4


texture_mean
M
B
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
−2
−1
0
1
2
3
4
5

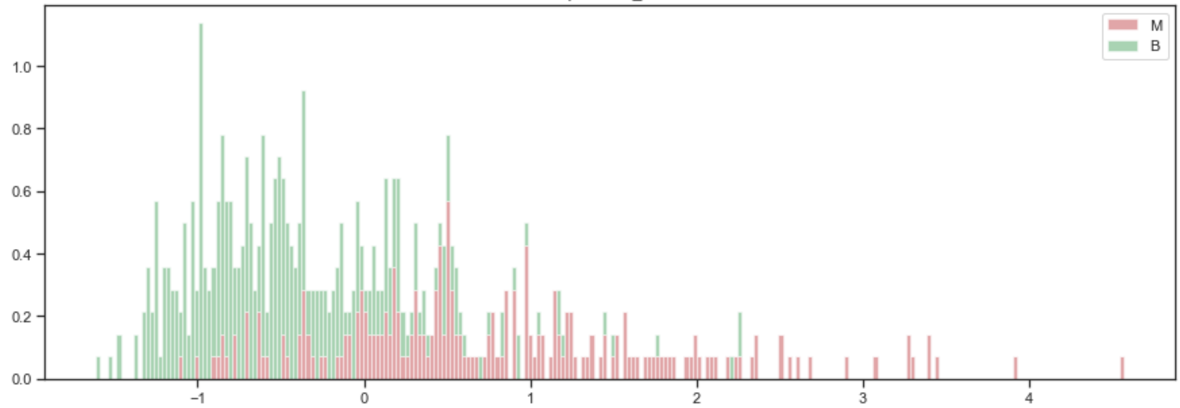

perimeter_mean
M
B
0.0
0.2
0.4
0.6
0.8
1.0
−2
−1
0
1
2
3
4


area_mean
M
B
0.0
0.2
0.4
0.6
0.8
1.0
−1
0
1
2
3
4
5

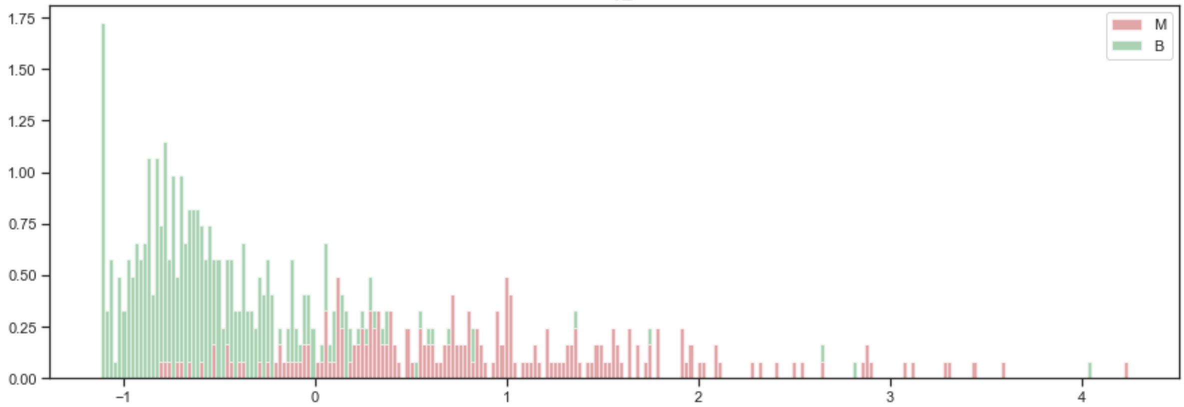

smoothness_mean
M
B
0.2
0.3
0.4
0.5
0.6
0.7



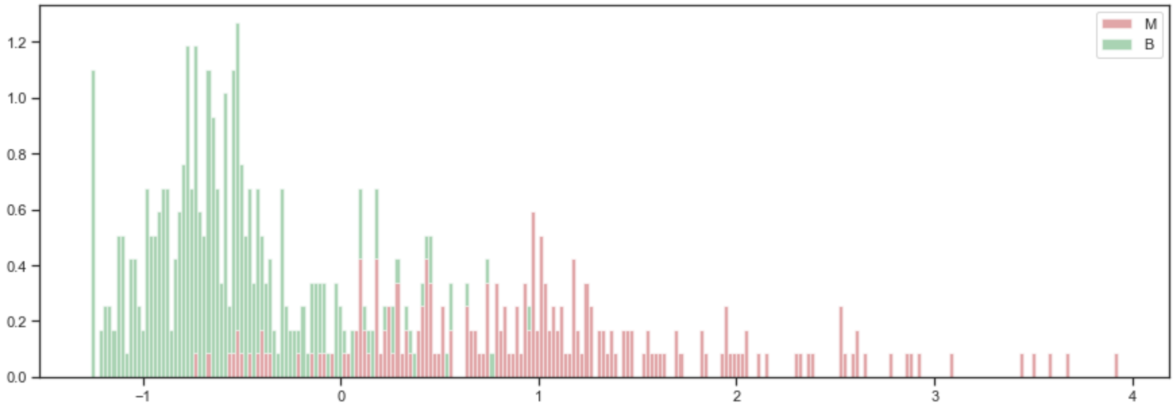
compactness_mean



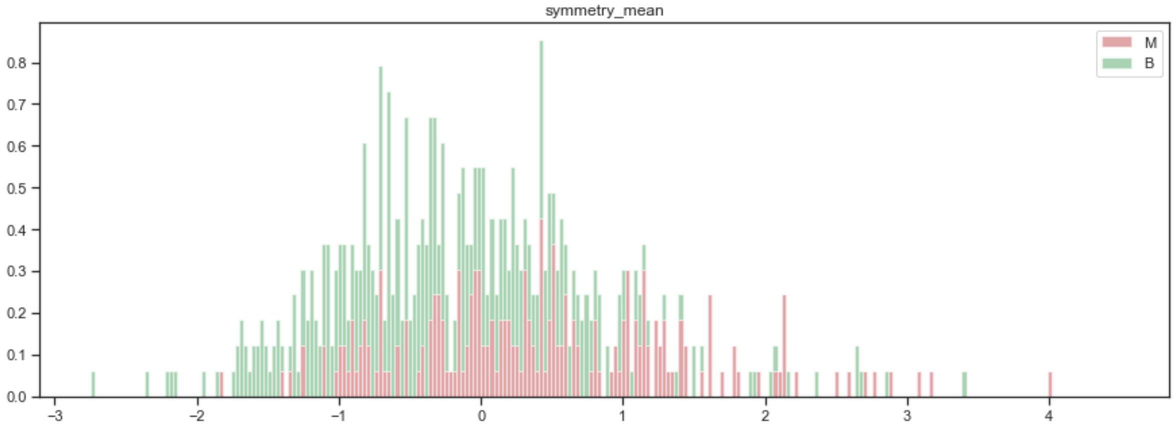
concavity_mean



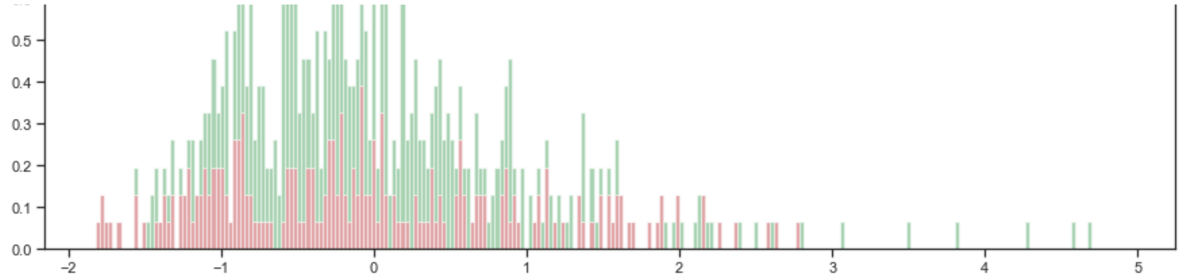
concave points_mean



symmetry_mean



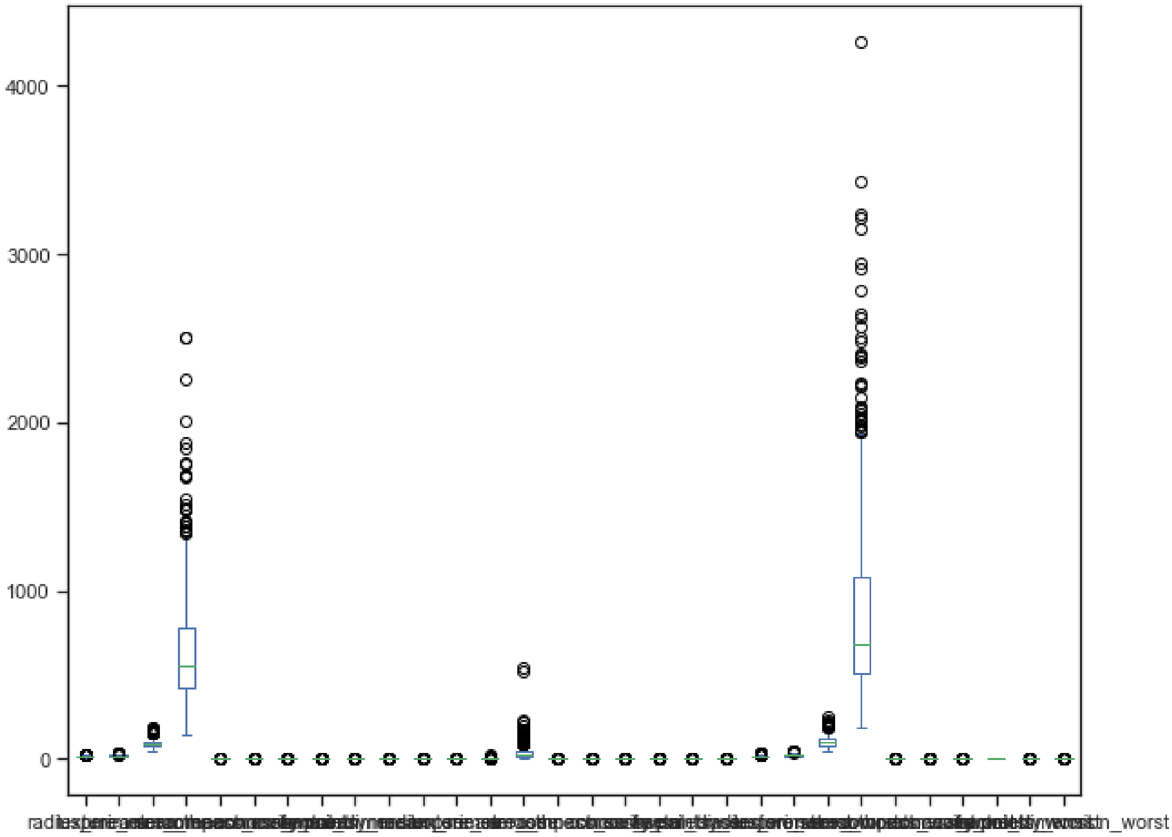
fractal_dimension_mean



In [17]:
```
printmd('## Box plot')
data.select_dtypes(exclude = 'category').plot(kind = 'box', figsize = (10,8))
```

Box plot

Out[17]: <AxesSubplot:>



4. Model Development & Classification

4.1. Data Preparation

In [18]:
```
data
```

Out[18]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	com
0	1	17.99	10.38	122.80	1001.0	0.11840	
1	1	20.57	17.77	132.90	1326.0	0.08474	
2	1	19.69	21.25	130.00	1203.0	0.10960	
3	1	11.42	20.38	77.58	386.1	0.14250	
4	1	20.29	14.34	135.10	1297.0	0.10030	
...	...	...	...	...	...	...	
564	1	21.56	22.39	142.00	1479.0	0.11100	
565	1	20.13	28.25	131.20	1261.0	0.09780	
566	1	16.60	28.08	108.30	858.1	0.08455	
567	1	20.60	29.33	140.10	1265.0	0.11780	
568	0	7.76	24.54	47.92	181.0	0.05263	

569 rows × 31 columns

In [19]:

```
X = data.iloc[:, 1:].values
y = data.iloc[:, 0].values

from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y)
```

In [20]:

```
X
```

Out[20]:

```
array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01,
        1.189e-01],
       [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01,
        8.902e-02],
       [1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.613e-01,
        8.758e-02],
       ...,
       [1.660e+01, 2.808e+01, 1.083e+02, ..., 1.418e-01, 2.218e-01,
        7.820e-02],
       [2.060e+01, 2.933e+01, 1.401e+02, ..., 2.650e-01, 4.087e-01,
        1.240e-01],
       [7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 2.871e-01,
        7.039e-02]])
```

In [21]:

```
y
```

Out[21]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1,
       0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1,
       0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1,
       1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0,
       0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1,
       1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1,
       1, 0, 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, 1, 1, 0, 1, 1, 1, 0, 1, 1,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 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, 1, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0,
       0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 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, 1, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 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, 1, 1, 1, 1, 1, 1, 0],
      dtype=int64)
```

In [22]:
```
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype   
---  ------                   --------------  -----   
 0   diagnosis                569 non-null    category
 1   radius_mean              569 non-null    float64 
 2   texture_mean             569 non-null    float64 
 3   perimeter_mean           569 non-null    float64 
 4   area_mean                569 non-null    float64 
 5   smoothness_mean          569 non-null    float64 
 6   compactness_mean         569 non-null    float64 
 7   concavity_mean           569 non-null    float64 
 8   concave points_mean      569 non-null    float64 
 9   symmetry_mean            569 non-null    float64 
 10  fractal_dimension_mean   569 non-null    float64 
 11  radius_se                569 non-null    float64 
 12  texture_se               569 non-null    float64 
 13  perimeter_se             569 non-null    float64 
 14  area_se                  569 non-null    float64 
 15  smoothness_se            569 non-null    float64 
 16  compactness_se           569 non-null    float64 
 17  concavity_se             569 non-null    float64 
 18  concave points_se        569 non-null    float64 
 19  symmetry_se              569 non-null    float64 
 20  fractal_dimension_se     569 non-null    float64 
 21  radius_worst             569 non-null    float64 
 22  texture_worst            569 non-null    float64 
 23  perimeter_worst          569 non-null    float64 
 24  area_worst               569 non-null    float64 
 25  smoothness_worst         569 non-null    float64 
 26  compactness_worst        569 non-null    float64 
 27  concavity_worst          569 non-null    float64 
 28  concave points_worst     569 non-null    float64 
 29  symmetry_worst           569 non-null    float64 
 30  fractal_dimension_worst  569 non-null    float64 
dtypes: category(1), float64(30)
memory usage: 134.2 KB
None
```

In [23]:
```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, random_
```

In [24]:
```
#Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

4.2. Model Development

In [25]:
```
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout
```

In [26]:
```
# Initialising the ANN
classifier = Sequential()
```

In [27]:
```
#first hidden layer
```

```
classifier.add(Dense(units=9,kernel_initializer='he_uniform',activation='relu',inpu
```

In [28]:
```
classifier.add(Dropout(rate=0.1))
```

In [29]:
```
#second hidden layer
classifier.add(Dense(units=9,kernel_initializer='he_uniform',activation='relu'))
```

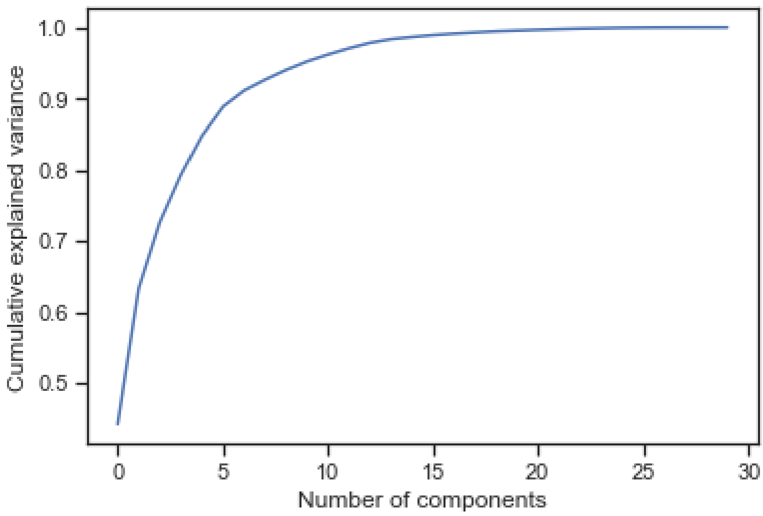
In [30]:
```
# last layer or output layer
classifier.add(Dense(units=1,kernel_initializer='glorot_uniform',activation='sigmoi
```

In [31]:
```
#taking summary of layers
classifier.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense (Dense)               (None, 9)                 279       
                                                                 
 dropout (Dropout)           (None, 9)                 0         
                                                                 
 dense_1 (Dense)             (None, 9)                 90        
                                                                 
 dense_2 (Dense)             (None, 1)                 10        
                                                                 
=================================================================
Total params: 379
Trainable params: 379
Non-trainable params: 0
_________________________________________________________________
```

In [32]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=30)
pca.fit(X_train)

plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Number of components')
plt.ylabel('Cumulative explained variance')
```

Out[32]:
```
Text(0, 0.5, 'Cumulative explained variance')
```



In [33]:
```
classifier.compile(optimizer='adam', loss='binary_crossentropy', metrics='accuracy
```

In [34]:
```
X_train
```

Out[34]:
```
array([[-0.52787029,  2.49821982, -0.59939466, ..., -1.74713139,
        -0.79044533, -0.91054389],
       [-0.55333608,  0.29431013, -0.60759343, ..., -0.62275667,
        -0.33646358, -0.83551633],
       [ 2.15452653,  0.40392257,  2.26525805, ...,  1.03846122,
        -0.11504791,  0.26488788],
       ...,
       [-1.3297598 , -0.21876938, -1.32088704, ..., -0.98271999,
        -0.718764  , -0.13637062],
       [-1.24940108, -0.24209117, -1.2835826 , ..., -1.74713139,
        -1.58690456, -1.01280367],
       [-0.74291476,  1.08958336, -0.71827692, ..., -0.2865488 ,
        -1.26354211,  0.19486216]])
```

In [35]:
```
len(X_train)
```

Out[35]:
```
512
```

In [36]:
```
y_train
```

Out[36]:
```
array([0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0,
       0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0,
       1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
       1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1,
       0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1,
       0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0,
       0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0,
       0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1,
       1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1,
       0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0,
       0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1,
       0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1,
       1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0,
       1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
       1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0,
       0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0,
       1, 1, 1, 0, 0, 0], dtype=int64)
```

In [37]:
```
len(y_train)
```

Out[37]:
```
512
```

In [39]:
```
# Fitting the ANN to the Training set
history = classifier.fit(X_train, y_train, batch_size=32, epochs=50, validation_sp
# Long scroll ahead but worth
# The batch size and number of epochs have been set using trial and error. Still lo
```

```
Epoch 1/50
13/13 [==============================] - 1s 13ms/step - loss: 0.6886 - accuracy:
0.6381 - val_loss: 0.6246 - val_accuracy: 0.6019
Epoch 2/50
13/13 [==============================] - 0s 3ms/step - loss: 0.6014 - accuracy: 0.
6699 - val_loss: 0.5362 - val_accuracy: 0.6408
Epoch 3/50
13/13 [==============================] - 0s 3ms/step - loss: 0.5387 - accuracy: 0.
7066 - val_loss: 0.4654 - val_accuracy: 0.7184
Epoch 4/50
13/13 [==============================] - 0s 3ms/step - loss: 0.4715 - accuracy: 0.
7677 - val_loss: 0.4013 - val_accuracy: 0.7961
Epoch 5/50
13/13 [==============================] - 0s 3ms/step - loss: 0.4360 - accuracy: 0.
8117 - val_loss: 0.3484 - val_accuracy: 0.8932
Epoch 6/50
13/13 [==============================] - 0s 3ms/step - loss: 0.3941 - accuracy: 0.
8337 - val_loss: 0.3057 - val_accuracy: 0.9126
Epoch 7/50
13/13 [==============================] - 0s 3ms/step - loss: 0.3653 - accuracy: 0.
8460 - val_loss: 0.2712 - val_accuracy: 0.9223
Epoch 8/50
13/13 [==============================] - 0s 3ms/step - loss: 0.3191 - accuracy: 0.
8729 - val_loss: 0.2433 - val_accuracy: 0.9709
Epoch 9/50
13/13 [==============================] - 0s 3ms/step - loss: 0.2982 - accuracy: 0.
8924 - val_loss: 0.2178 - val_accuracy: 0.9806
Epoch 10/50
13/13 [==============================] - 0s 3ms/step - loss: 0.2650 - accuracy: 0.
9169 - val_loss: 0.1950 - val_accuracy: 0.9806
Epoch 11/50
13/13 [==============================] - 0s 3ms/step - loss: 0.2429 - accuracy: 0.
9315 - val_loss: 0.1735 - val_accuracy: 0.9903
Epoch 12/50
13/13 [==============================] - 0s 3ms/step - loss: 0.2273 - accuracy: 0.
9413 - val_loss: 0.1553 - val_accuracy: 0.9903
Epoch 13/50
13/13 [==============================] - 0s 3ms/step - loss: 0.2134 - accuracy: 0.
9413 - val_loss: 0.1401 - val_accuracy: 0.9903
Epoch 14/50
13/13 [==============================] - 0s 3ms/step - loss: 0.1927 - accuracy: 0.
9462 - val_loss: 0.1281 - val_accuracy: 0.9903
Epoch 15/50
13/13 [==============================] - 0s 3ms/step - loss: 0.1796 - accuracy: 0.
9560 - val_loss: 0.1181 - val_accuracy: 0.9903
Epoch 16/50
13/13 [==============================] - 0s 3ms/step - loss: 0.1689 - accuracy: 0.
9511 - val_loss: 0.1092 - val_accuracy: 0.9903
Epoch 17/50
13/13 [==============================] - 0s 3ms/step - loss: 0.1534 - accuracy: 0.
9584 - val_loss: 0.1013 - val_accuracy: 0.9903
Epoch 18/50
13/13 [==============================] - 0s 3ms/step - loss: 0.1547 - accuracy: 0.
9560 - val_loss: 0.0955 - val_accuracy: 0.9903
Epoch 19/50
13/13 [==============================] - 0s 3ms/step - loss: 0.1536 - accuracy: 0.
9511 - val_loss: 0.0909 - val_accuracy: 0.9903
Epoch 20/50
13/13 [==============================] - 0s 3ms/step - loss: 0.1322 - accuracy: 0.
9584 - val_loss: 0.0861 - val_accuracy: 0.9903
Epoch 21/50
13/13 [==============================] - 0s 3ms/step - loss: 0.1269 - accuracy: 0.
9584 - val_loss: 0.0817 - val_accuracy: 0.9903
Epoch 22/50
```

```
13/13 [==============================] - 0s 3ms/step - loss: 0.1289 - accuracy: 0.
9584 - val_loss: 0.0781 - val_accuracy: 0.9903
Epoch 23/50
13/13 [==============================] - 0s 3ms/step - loss: 0.1154 - accuracy: 0.
9658 - val_loss: 0.0747 - val_accuracy: 0.9903
Epoch 24/50
13/13 [==============================] - 0s 3ms/step - loss: 0.1163 - accuracy: 0.
9658 - val_loss: 0.0720 - val_accuracy: 0.9903
Epoch 25/50
13/13 [==============================] - 0s 3ms/step - loss: 0.1081 - accuracy: 0.
9707 - val_loss: 0.0692 - val_accuracy: 0.9903
Epoch 26/50
13/13 [==============================] - 0s 3ms/step - loss: 0.1137 - accuracy: 0.
9658 - val_loss: 0.0672 - val_accuracy: 0.9903
Epoch 27/50
13/13 [==============================] - 0s 3ms/step - loss: 0.1096 - accuracy: 0.
9658 - val_loss: 0.0645 - val_accuracy: 0.9903
Epoch 28/50
13/13 [==============================] - 0s 3ms/step - loss: 0.1037 - accuracy: 0.
9658 - val_loss: 0.0623 - val_accuracy: 0.9903
Epoch 29/50
13/13 [==============================] - 0s 3ms/step - loss: 0.0974 - accuracy: 0.
9731 - val_loss: 0.0602 - val_accuracy: 0.9903
Epoch 30/50
13/13 [==============================] - 0s 3ms/step - loss: 0.1022 - accuracy: 0.
9731 - val_loss: 0.0586 - val_accuracy: 0.9903
Epoch 31/50
13/13 [==============================] - 0s 3ms/step - loss: 0.0977 - accuracy: 0.
9731 - val_loss: 0.0567 - val_accuracy: 0.9903
Epoch 32/50
13/13 [==============================] - 0s 3ms/step - loss: 0.0922 - accuracy: 0.
9731 - val_loss: 0.0546 - val_accuracy: 0.9903
Epoch 33/50
13/13 [==============================] - 0s 3ms/step - loss: 0.0880 - accuracy: 0.
9756 - val_loss: 0.0528 - val_accuracy: 0.9903
Epoch 34/50
13/13 [==============================] - 0s 3ms/step - loss: 0.0901 - accuracy: 0.
9756 - val_loss: 0.0514 - val_accuracy: 0.9903
Epoch 35/50
13/13 [==============================] - 0s 3ms/step - loss: 0.0872 - accuracy: 0.
9756 - val_loss: 0.0502 - val_accuracy: 0.9903
Epoch 36/50
13/13 [==============================] - 0s 3ms/step - loss: 0.0844 - accuracy: 0.
9780 - val_loss: 0.0487 - val_accuracy: 0.9903
Epoch 37/50
13/13 [==============================] - 0s 3ms/step - loss: 0.0832 - accuracy: 0.
9756 - val_loss: 0.0465 - val_accuracy: 0.9903
Epoch 38/50
13/13 [==============================] - 0s 3ms/step - loss: 0.0913 - accuracy: 0.
9731 - val_loss: 0.0454 - val_accuracy: 0.9903
Epoch 39/50
13/13 [==============================] - 0s 3ms/step - loss: 0.0896 - accuracy: 0.
9780 - val_loss: 0.0443 - val_accuracy: 0.9903
Epoch 40/50
13/13 [==============================] - 0s 3ms/step - loss: 0.0856 - accuracy: 0.
9731 - val_loss: 0.0431 - val_accuracy: 0.9903
Epoch 41/50
13/13 [==============================] - 0s 3ms/step - loss: 0.0850 - accuracy: 0.
9780 - val_loss: 0.0419 - val_accuracy: 0.9903
Epoch 42/50
13/13 [==============================] - 0s 3ms/step - loss: 0.0827 - accuracy: 0.
9756 - val_loss: 0.0410 - val_accuracy: 0.9903
Epoch 43/50
13/13 [==============================] - 0s 3ms/step - loss: 0.0752 - accuracy: 0.
```

```
9804 - val_loss: 0.0397 - val_accuracy: 0.9903
Epoch 44/50
13/13 [==============================] - 0s 3ms/step - loss: 0.0843 - accuracy: 0.
9804 - val_loss: 0.0389 - val_accuracy: 0.9903
Epoch 45/50
13/13 [==============================] - 0s 3ms/step - loss: 0.0772 - accuracy: 0.
9780 - val_loss: 0.0375 - val_accuracy: 0.9903
Epoch 46/50
13/13 [==============================] - 0s 3ms/step - loss: 0.0761 - accuracy: 0.
9780 - val_loss: 0.0364 - val_accuracy: 1.0000
Epoch 47/50
13/13 [==============================] - 0s 3ms/step - loss: 0.0766 - accuracy: 0.
9756 - val_loss: 0.0350 - val_accuracy: 1.0000
Epoch 48/50
13/13 [==============================] - 0s 5ms/step - loss: 0.0711 - accuracy: 0.
9829 - val_loss: 0.0342 - val_accuracy: 1.0000
Epoch 49/50
13/13 [==============================] - 0s 3ms/step - loss: 0.0671 - accuracy: 0.
9780 - val_loss: 0.0331 - val_accuracy: 1.0000
Epoch 50/50
13/13 [==============================] - 0s 3ms/step - loss: 0.0715 - accuracy: 0.
9804 - val_loss: 0.0322 - val_accuracy: 1.0000
```

In [40]:

```
# Predicting the Test set results
y_pred = classifier.predict(X_test)
y_pred = (y_pred > 0.5)
```

In [41]:

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

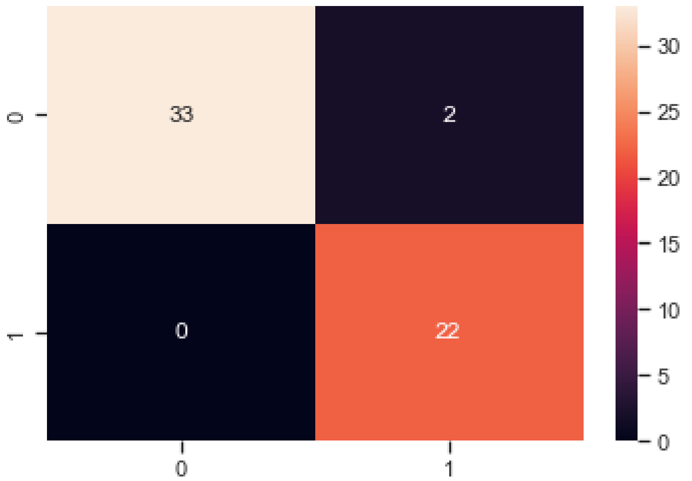
In [42]:

```
print("Our accuracy is {}%".format(((cm[0][0] + cm[1][1])/57)*100))
```

```
Our accuracy is 96.49122807017544%
```

In [43]:

```
sns.heatmap(cm,annot=True)
plt.savefig('h.png')
```

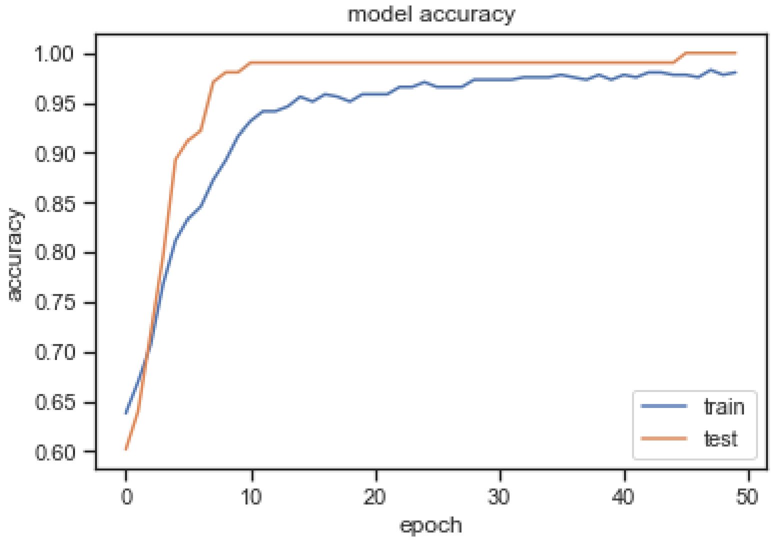


In [ ]:

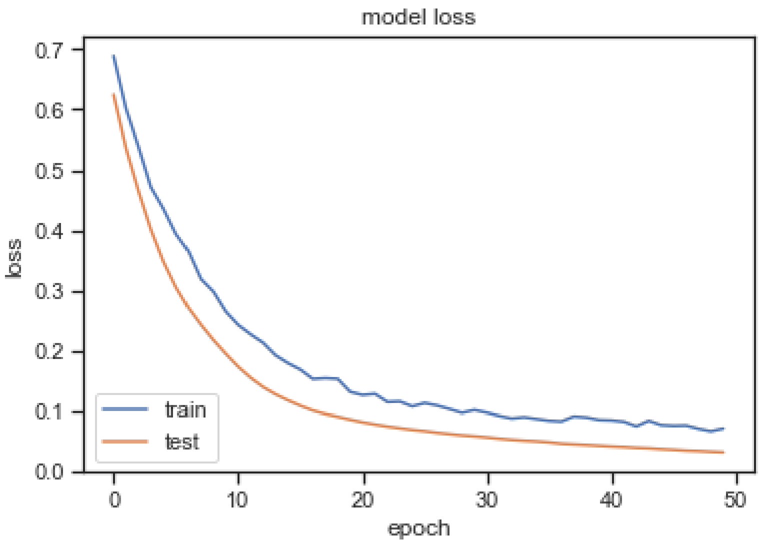
In [44]:

```
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
```

```
plt.legend(['train', 'test'], loc='lower right')
plt.show()
```



In [45]:
```
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='lower left')
plt.show()
```



In [46]:
```
classifier.evaluate(X_test, y_test, verbose=2)
```

```
2/2 - 0s - loss: 0.0750 - accuracy: 0.9649 - 14ms/epoch - 7ms/step
```

Out[46]:
```
[0.07502038776874542, 0.9649122953414917]
```

In [47]:
```
y_pred = classifier.predict(X_test, verbose=0)
y_pred
```

Out[47]:
```
array([[9.8528194e-01],
       [4.2927146e-02],
       [6.1172247e-04],
       [6.7895651e-03],
       [8.1724393e-05],
       [2.8225183e-03],
       [7.2002498e-05],
       [5.4946542e-04],
       [1.9304873e-05],
       [7.8006798e-07],
       [3.0570233e-01],
       [1.0753912e-01],
       [7.3306014e-06],
       [6.5287519e-01],
       [8.2655454e-01],
       [9.9573505e-01],
       [2.2798777e-04],
       [9.9981570e-01],
       [9.9914670e-01],
       [9.9999940e-01],
       [9.9596143e-01],
       [9.9139106e-01],
       [4.4696629e-03],
       [7.5700879e-04],
       [9.9949783e-01],
       [1.9466877e-04],
       [8.5376541e-06],
       [9.9427080e-01],
       [7.2956085e-04],
       [9.9994797e-01],
       [1.0338121e-05],
       [9.9951959e-01],
       [9.8506540e-02],
       [9.7987396e-01],
       [3.8287490e-07],
       [9.6691626e-01],
       [9.4931126e-03],
       [9.9814558e-01],
       [4.9524903e-03],
       [9.8646605e-01],
       [7.5975442e-01],
       [1.3745483e-05],
       [7.3339814e-01],
       [2.0404559e-05],
       [5.3348005e-02],
       [9.9999940e-01],
       [1.4404962e-07],
       [3.0549169e-02],
       [4.0635467e-04],
       [9.9941981e-01],
       [9.9986076e-01],
       [9.7790194e-01],
       [9.9879289e-01],
       [6.6787004e-04],
       [1.5106201e-03],
       [1.1739471e-04],
       [1.3407469e-03]], dtype=float32)
```

In [49]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred.round(), labels=[0,1]))
```

```
              precision    recall  f1-score   support

           0       1.00      0.94      0.97        35
           1       0.92      1.00      0.96        22

    accuracy                           0.96        57
   macro avg       0.96      0.97      0.96        57
weighted avg       0.97      0.96      0.97        57
```

In [ ]: