Expt No.: 1 **Name: Swaranjana Nayak**
Date: 12/02/2021 **Reg. No.: 19BCE0977**

EXPERIMENT 1

Aim:
To visualize Breast Cancer Wisconsin (Diagnostic) dataset with at least 16 different types of plots.

About the dataset:
Title: Breast Cancer Wisconsin (Diagnostic) dataset
Description:

Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image.
n the 3-dimensional space is that described in: [K. P. Bennett and O. L. Mangasarian: "Robust Linear Programming Discrimination of Two Linearly Inseparable Sets", Optimization Methods and Software 1, 1992, 23-34].

Repository: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29

Attribute Information:

1) ID number
2) Diagnosis (M = malignant, B = benign)
3-32)

Ten real-valued features are computed for each cell nucleus:

a) radius (mean of distances from center to points on the perimeter)
b) texture (standard deviation of gray-scale values)
c) perimeter
d) area
e) smoothness (local variation in radius lengths)
f) compactness (perimeter^2 / area - 1.0)
g) concavity (severity of concave portions of the contour)
h) concave points (number of concave portions of the contour)
i) symmetry
j) fractal dimension ("coastline approximation" - 1)

The mean, standard error and "worst" or largest (mean of the three largest values) of these features were computed for each image, resulting in 30 features. For instance, field 3 is Mean Radius, field 13 is Radius SE, field 23 is Worst Radius.

All feature values are recoded with four significant digits.

Missing attribute values: none

Class distribution: 357 benign, 212 malignant

Data Visualization:

1. **Importing Libraries**
   Code

```
# importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import copy
%matplotlib inline
```

2. **Reading the csv file into a dataframe**

Code

```
df = pd.read_csv('data.csv')
df.head()
```

Output:

5 rows x 32 columns

So there are 32 columns in the dataset. As explained in the description of the dataset.

3. **Getting to know the dataset**

Code

```
df.info()
```

Output

```
<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
```

Interpretation:

There are 569 rows in the dataset, and none of them have Null values. So we don't need to do any processing for null values. We can see that except *diagnosis* all columns are numeric. *diagnosis* is a categorical column.

Code:

```
column = df.columns
column
```

What the code does:

In Jupyter Notebook, the code will show all the column names. We will use this column array over and over again in the rest of our code to group them and plot different kinds of plots.

Output:

```
Index(['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')
```

Code:

```
len(df['id'].unique()) == len(df.index)
# which means that all id values are unique we may ignore that column
```

Output:

```
True
```

Interpretation:
All values of the id column are unique. So the id column is the primary key of the table. We may ignore the attribute for plots since it won't help in analysis. It will help only in row identification.

Code:

```
df['diagnosis'].unique()
# M = malignant, B = benign
```

Output:

```
array(['M', 'B'], dtype=object)
```

Interpretation:
There are only two categories in the categorical attribute, M for Malignant Tumor diagnosis and B for Benign Tumor Diagnosis.

4. Visualizations

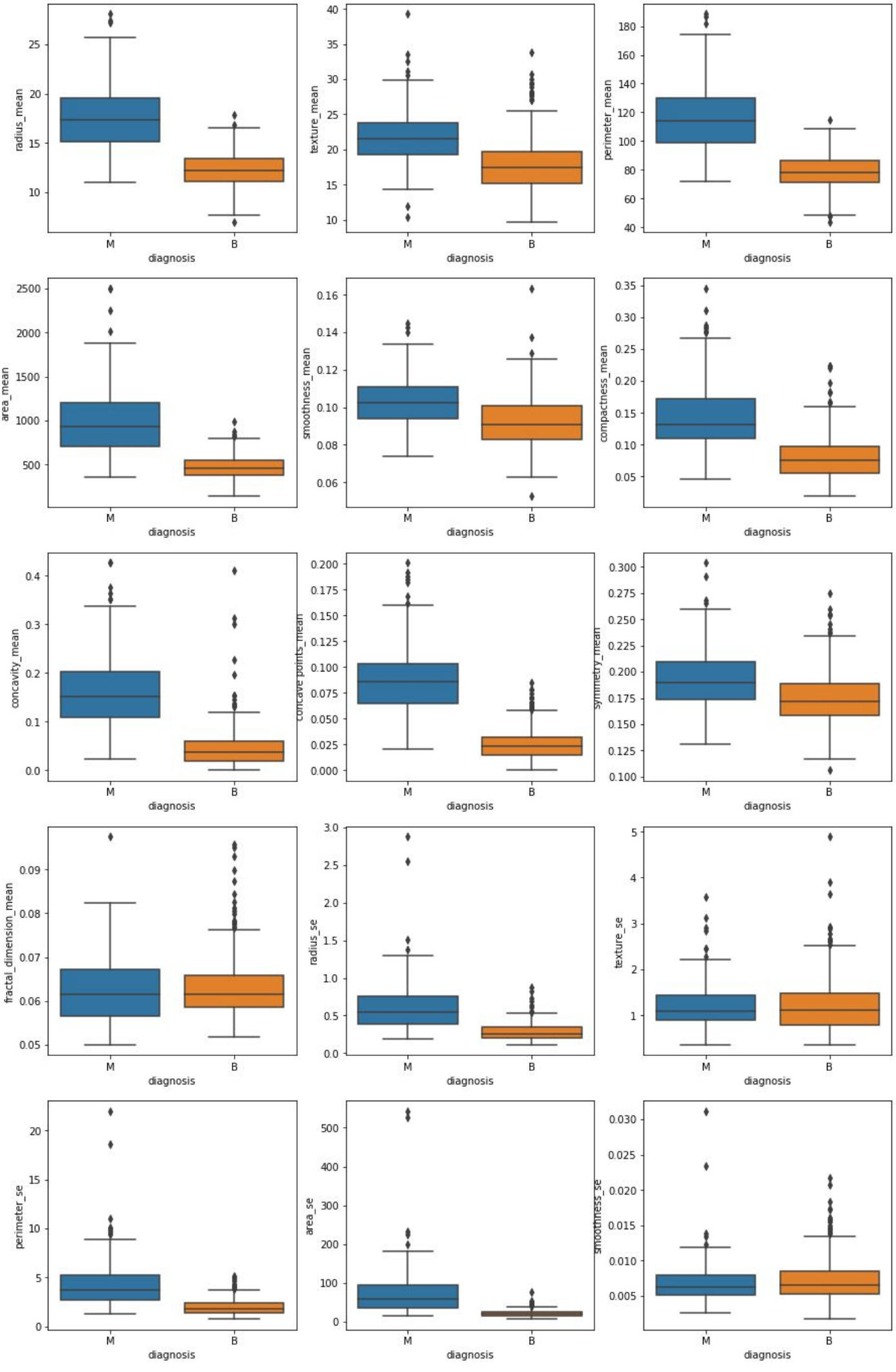
a. Box Plot

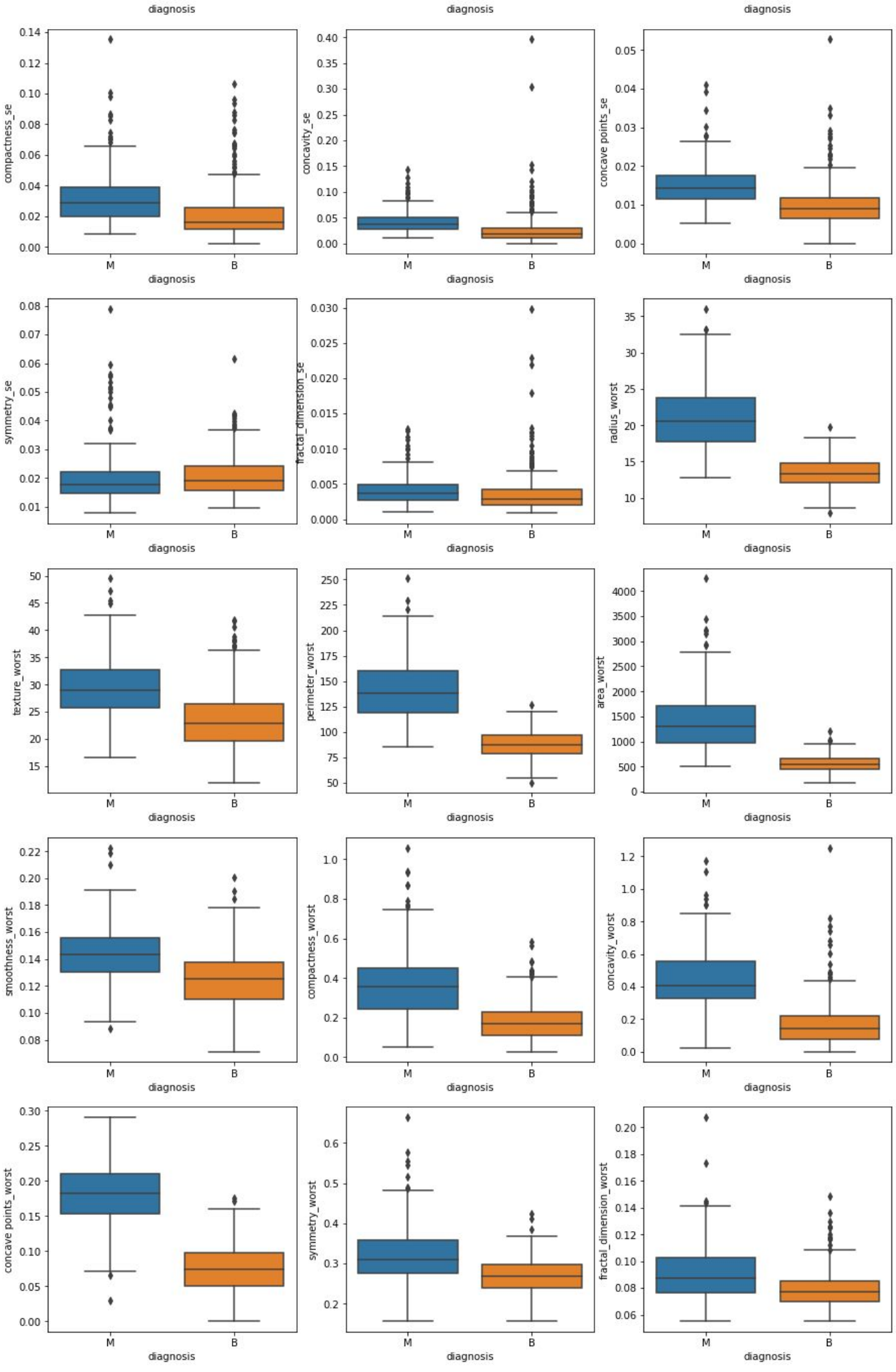
Justification:
Here, a boxplot of every numerical attribute is drawn against the categorical attribute *diagnosis.* A boxplot is a standardized way of displaying the distribution of data based on a five number summary ("minimum", first quartile (Q1), median, third quartile (Q3), and "maximum"). It can tell you about your outliers and what their values are. It can also tell you if your data is symmetrical, how tightly your data is grouped, and if and how your data is skewed. It was plotted for all 30 attributes since we need to know how they all are distributed categorized by Malignant and Benign values. We need to have information on the variability or the dispersion of the data. We can also see where the outliers in the data are.

Code:

```
fig, axes = plt.subplots(10, 3, figsize=(15,50))
c = np.reshape(column[2:], (10, 3))

for i in range(10):
    for j in range(3):
        sns.boxplot(df[column[1]], df[c[i][j]], ax = axes[i, j])

fig.savefig('boxplot.png', bbox_inches = 'tight')
```

Output:


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
diagnosis
M
B


diagnosis
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
M
B

b. Bar Chart

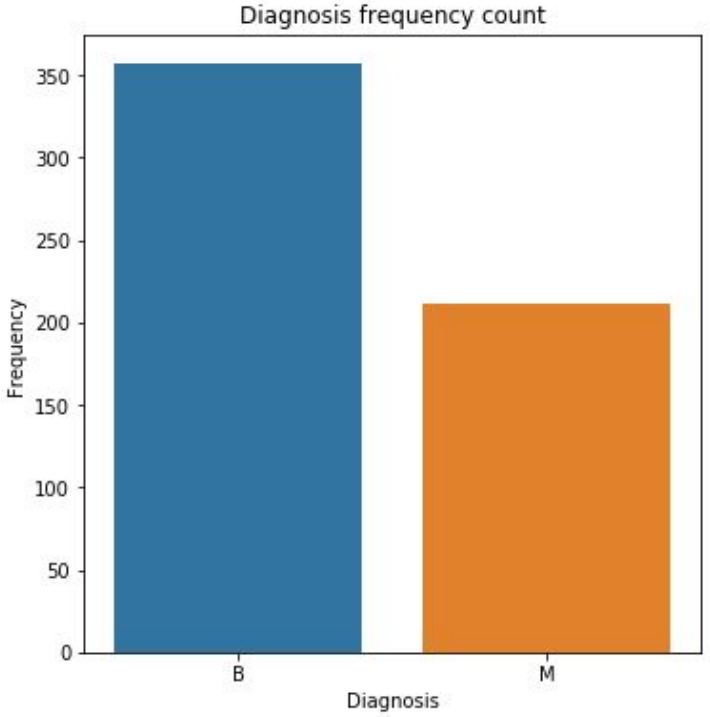
Justification:

In the following visualization, diagnosis is plotted against frequency. As in how many instances are diagnosed as Benign and Malignant. The primary variable of a bar chart is its categorical variable. A categorical variable takes discrete values, which can be thought of as labels. The primary categorical variable is diagnosis here. The secondary numerical variable is Frequency.

Code:

```
cat = cat_df['diagnosis'].value_counts()
fig = plt.figure(figsize = (9, 9))
ax = fig.add_axes([0, 0, 0.5, 0.5])
x = cat.index
y = cat.values
sns.barplot(x, y, ax = ax)
ax.set_title("Diagnosis frequency count")
ax.set_xlabel('Diagnosis')
ax.set_ylabel('Frequency')
fig.savefig('diagnosis_barplot.png', bbox_inches = 'tight')
```

Output:



c. Pie Chart

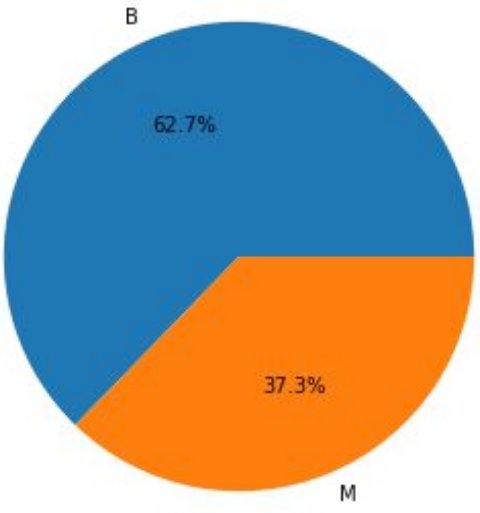
Justification:

A pie chart is used when we are trying to work out the composition of something. Here, by working out the composition of the diagnosis feature, we are working out the composition of all instances in the dataset.

Code:

```
labels = cat.index.tolist()
sizes = cat.values.tolist()

fig = plt.figure()
ax = fig.add_axes([0, 0, 1, 1])
ax.pie(sizes, labels=labels, autopct='%1.1f%%')
fig.savefig('diagnosis_piechart.png')
```

Output:



d. Pair Plot

Justification:

A pairs plot allows us to see both distribution of single variables and relationships between two variables. Pair plots are a great method to identify trends for follow-up analysis.
Here the attributes have been first grouped according to the metric they represent i.e radius, texture, perimeter etc. This plot helps us visualize how those attributes are correlated to each other and if there's any visual relationship between the mean, standard error and worst metrics.
We also have distributions of those single attributes.

Code for grouping:

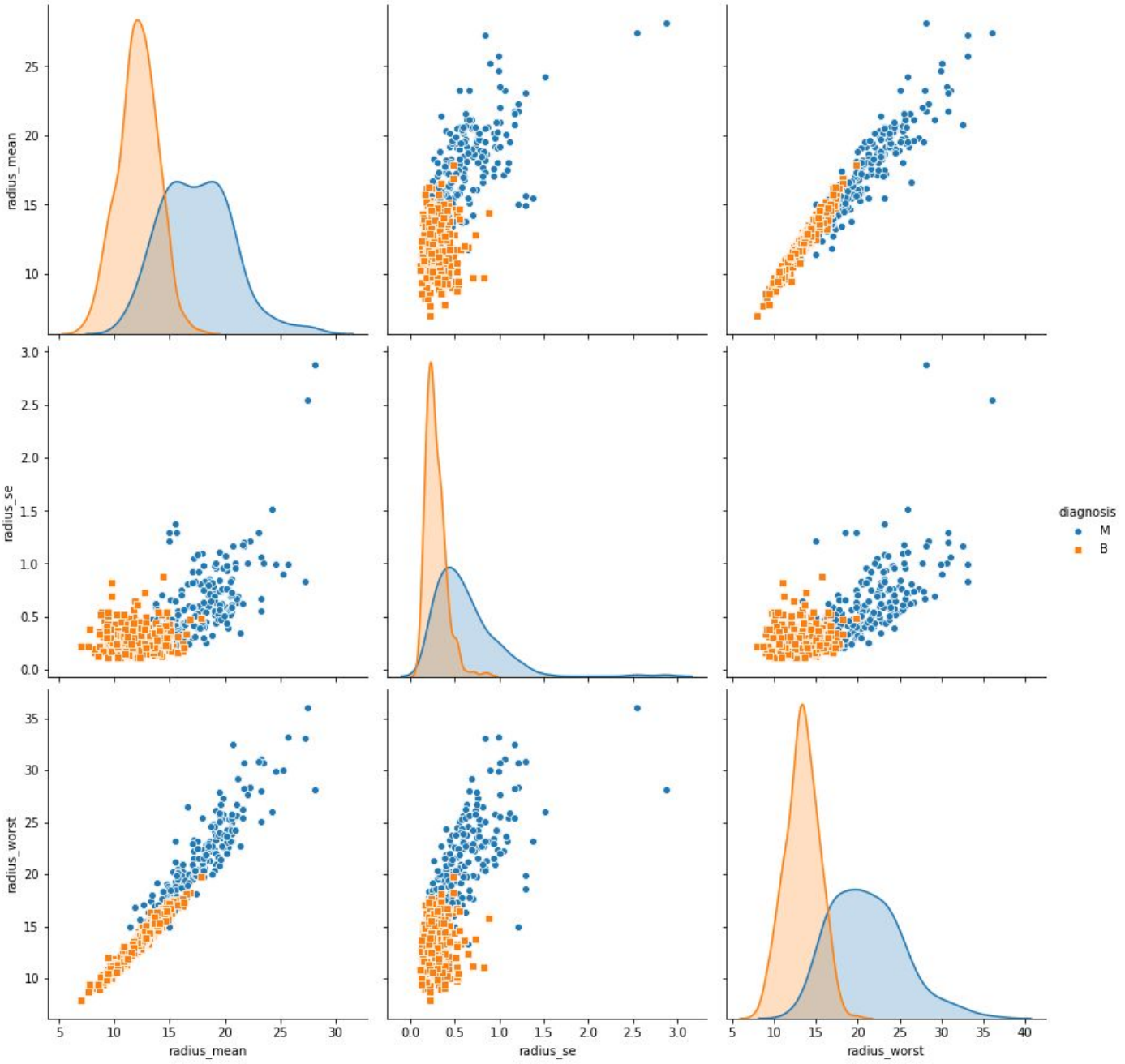
```
groups = []

for i in range (2, 12):
    l = []
    for j in range(3):
        l.append(column[i + j*10])
    l.append(column[1])
    groups.append(l)
```

i. Radius Attributes

Code:

```
radius = df[groups[0]]
fig = plt.figure()
fig = sns.pairplot(radius, hue='diagnosis', markers=["o",
"s"],height=4)
fig.savefig('radius_pairplot.png', bbox_inches = 'tight')
```

Output:

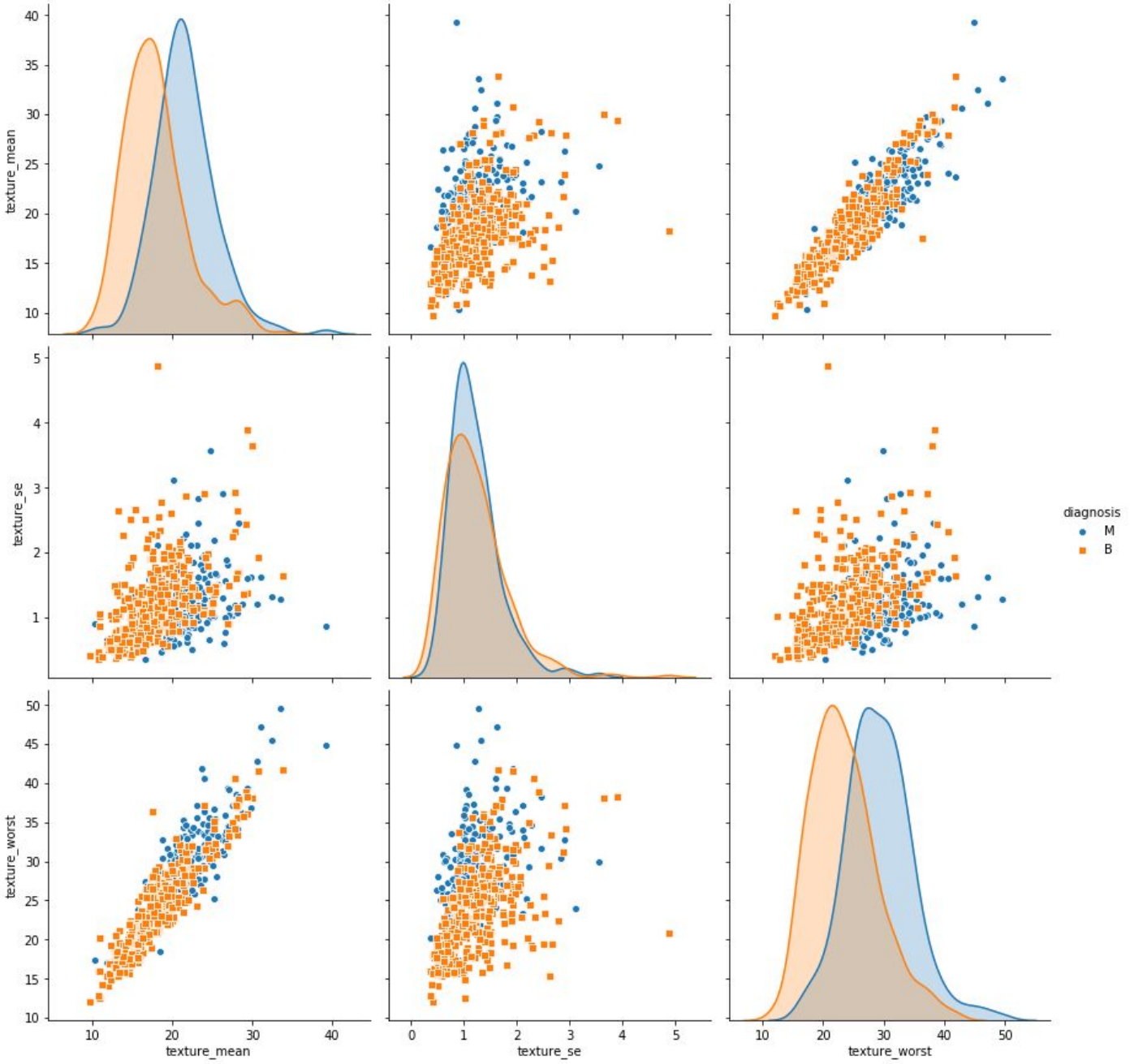


ii. Texture Attributes

Code:

```
texture = df[groups[1]]
fig = plt.figure()
fig = sns.pairplot(texture, hue='diagnosis', markers=["o",
"s"],height=4)
fig.savefig('texture_pairplot.png', bbox_inches = 'tight')
```

Output:

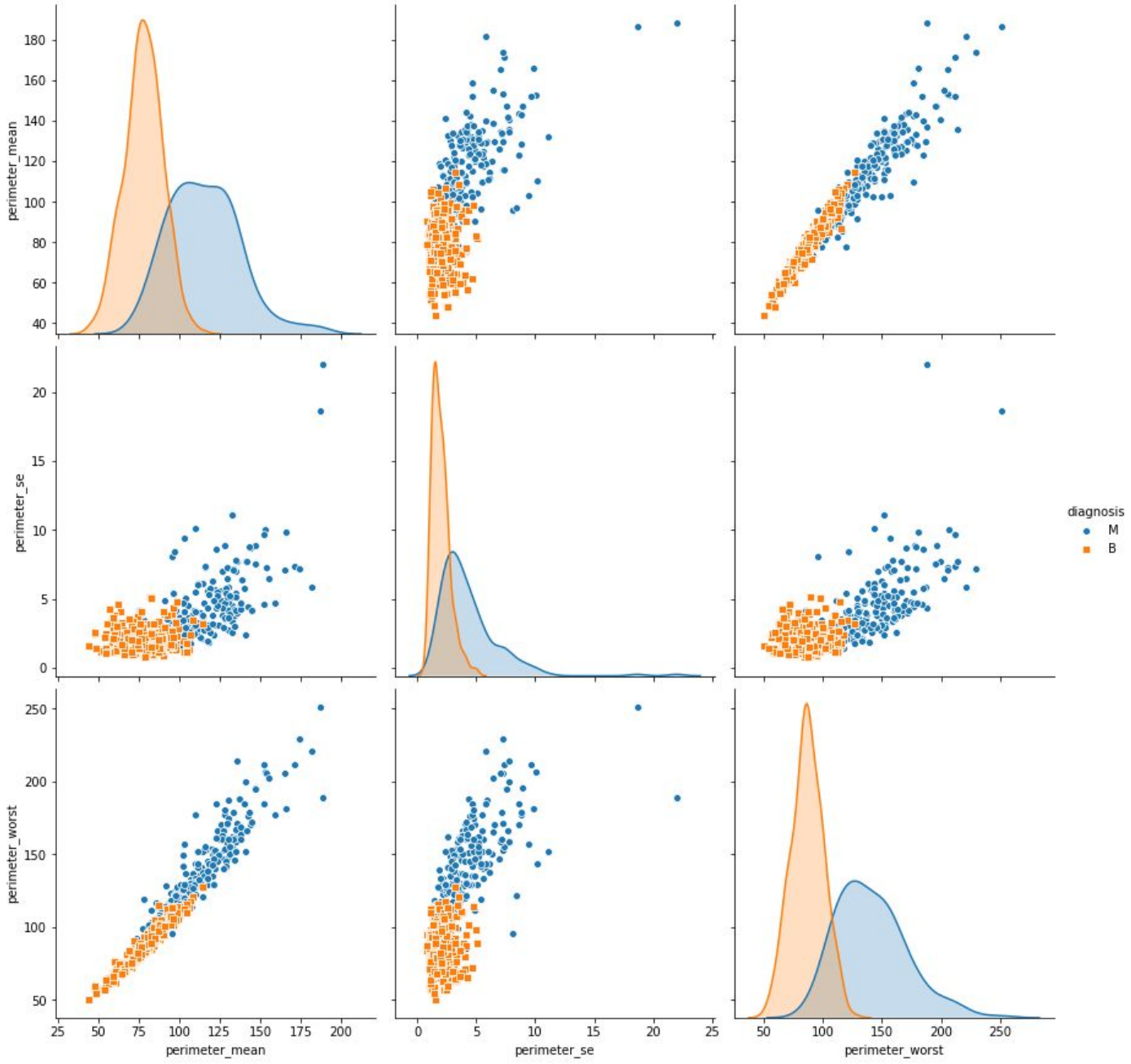


iii. Perimeter Attributes

Code:

```
perimeter = df[groups[2]]
fig = plt.figure()
fig = sns.pairplot(perimeter, hue='diagnosis',
markers=["o", "s"],height=4)
fig.savefig('perimeter_pairplot.png', bbox_inches =
'tight')
```

Output:

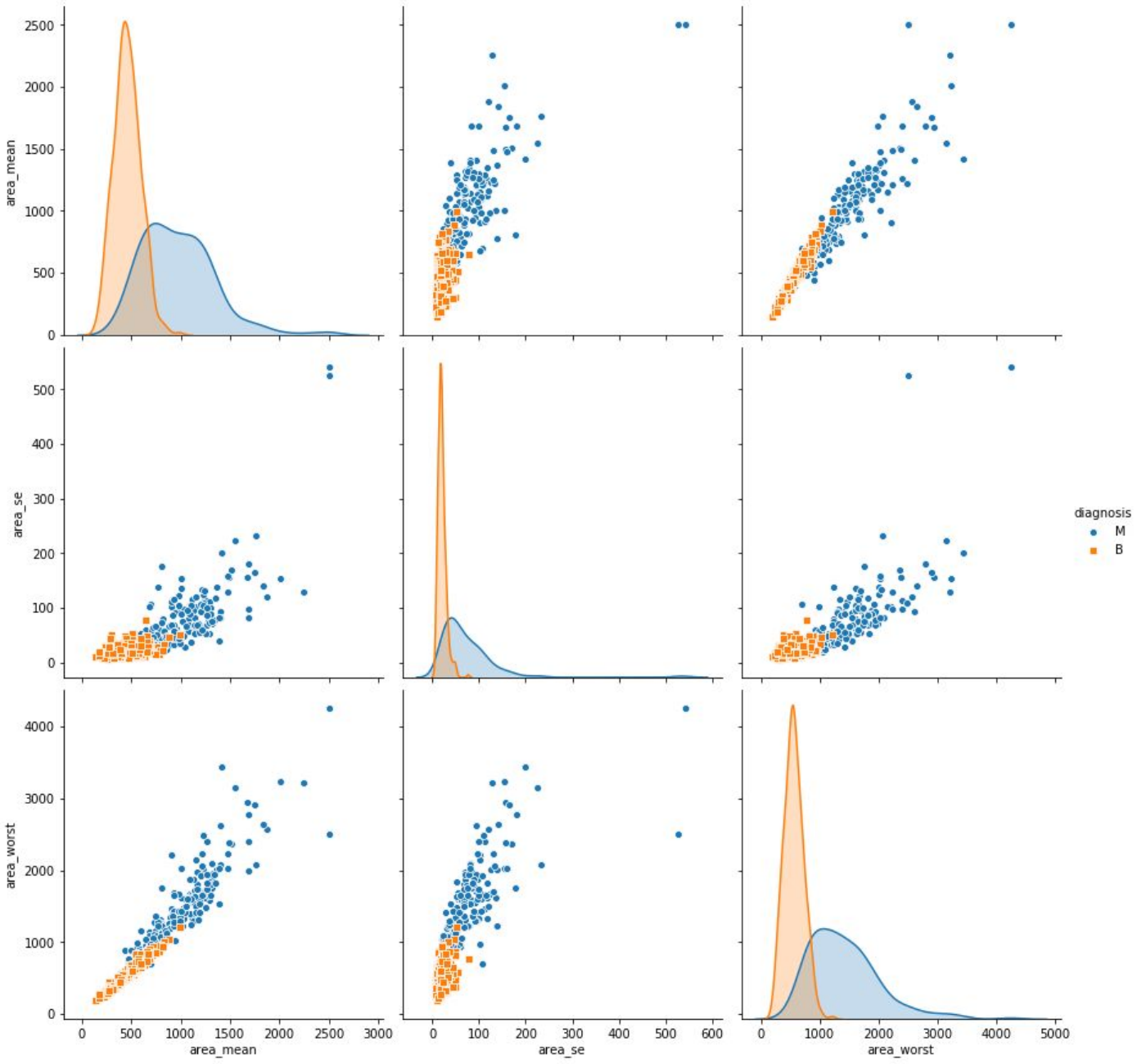


iv. <u>Area Attributes</u>

Code:

```
area = df[groups[3]]
fig = plt.figure()
fig = sns.pairplot(area, hue='diagnosis', markers=["o",
"s"],height=4)
fig.savefig('area_pairplot.png', bbox_inches = 'tight')
```

Output:

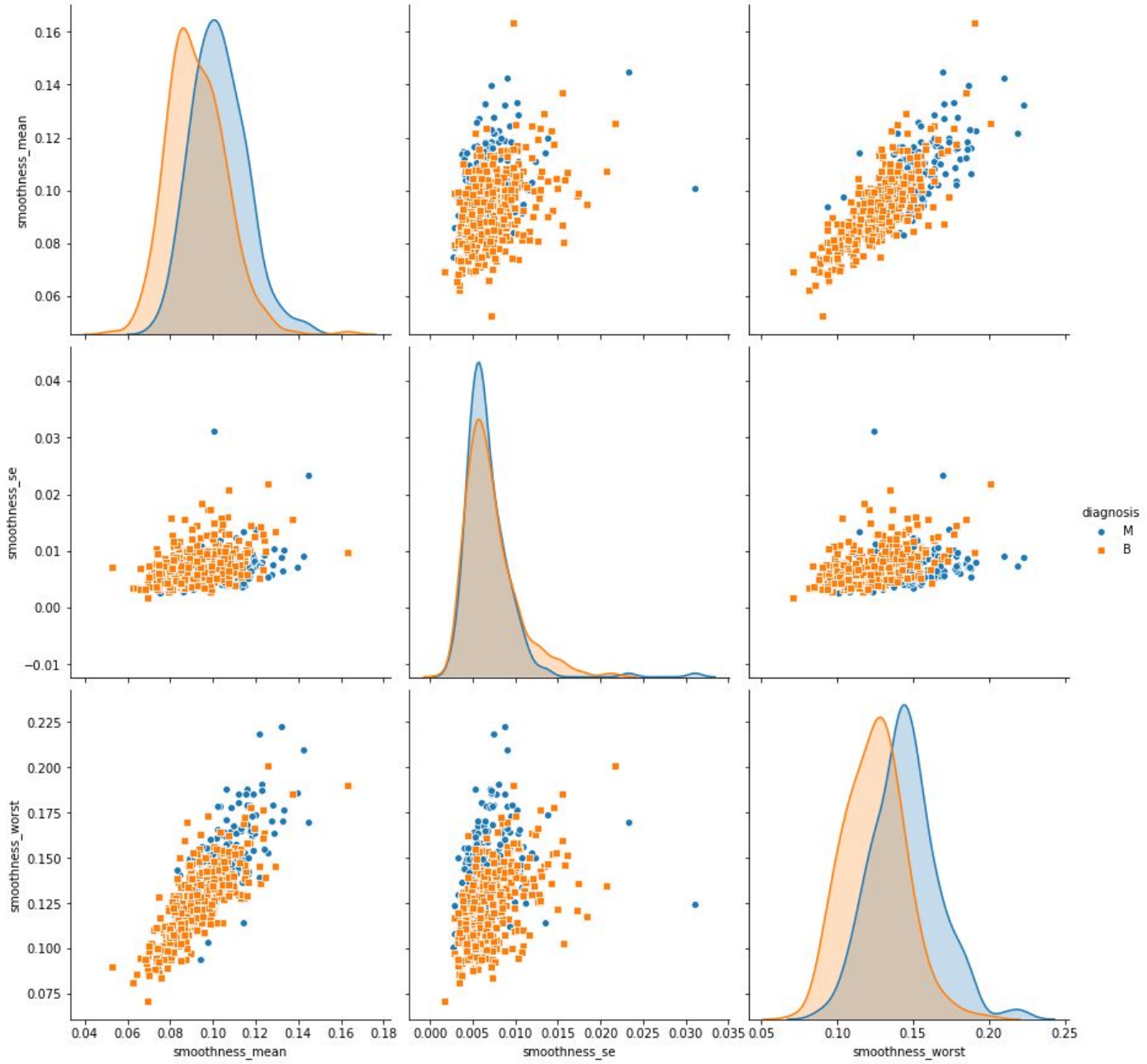


v. **Smoothness Attributes**

Code:

```
smoothness = df[groups[4]]
fig = plt.figure()
fig = sns.pairplot(smoothness, hue='diagnosis',
markers=["o", "s"],height=4)
fig.savefig('smoothness_pairplot.png', bbox_inches =
'tight')
```

Output:

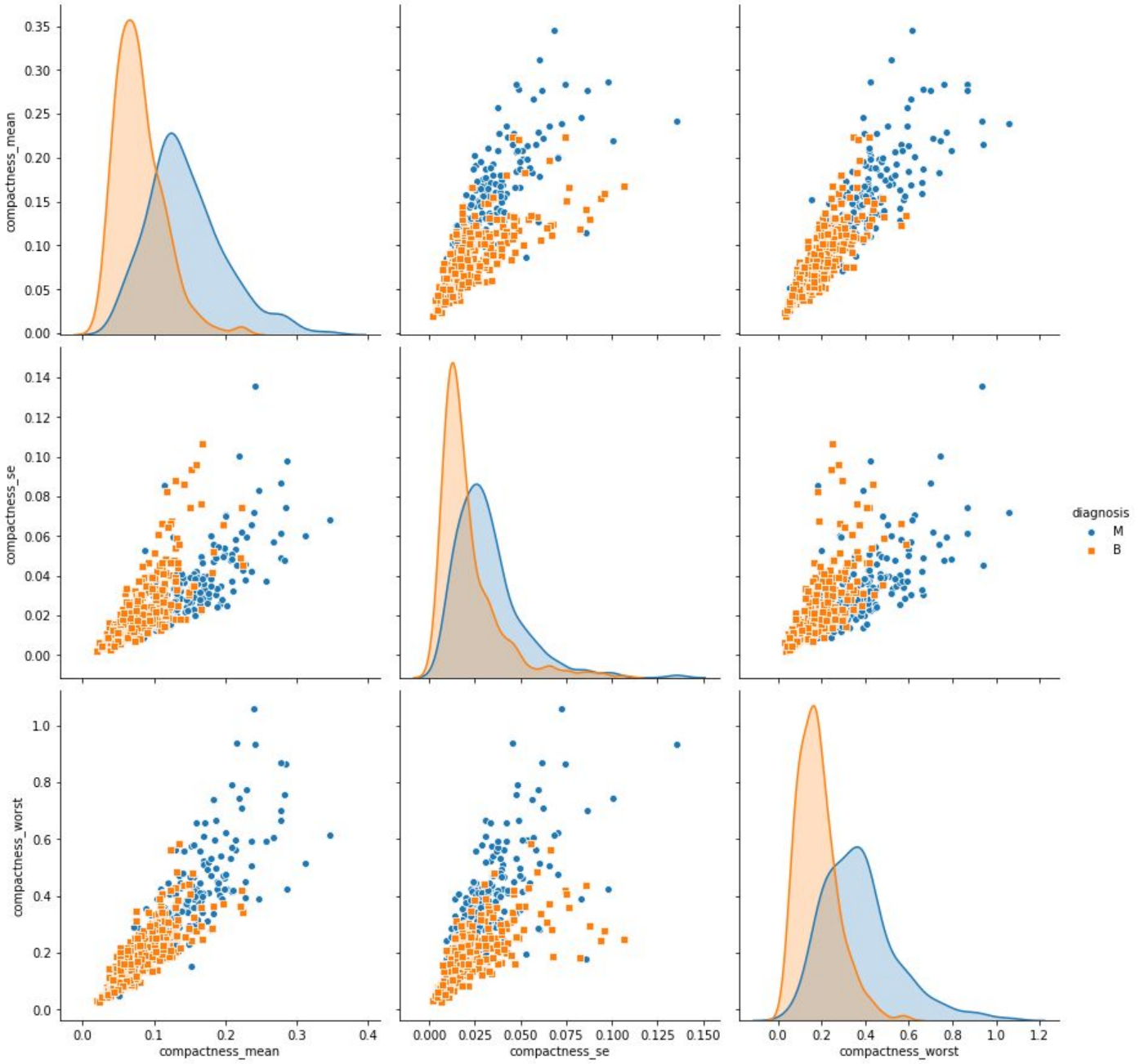


vi. **<u>Compactness Attributes</u>**

Code:

```
compactness = df[groups[5]]
fig = plt.figure()
fig = sns.pairplot(compactness, hue='diagnosis',
markers=["o", "s"],height=4)
fig.savefig('compactness_pairplot.png', bbox_inches =
'tight')
```

Output:

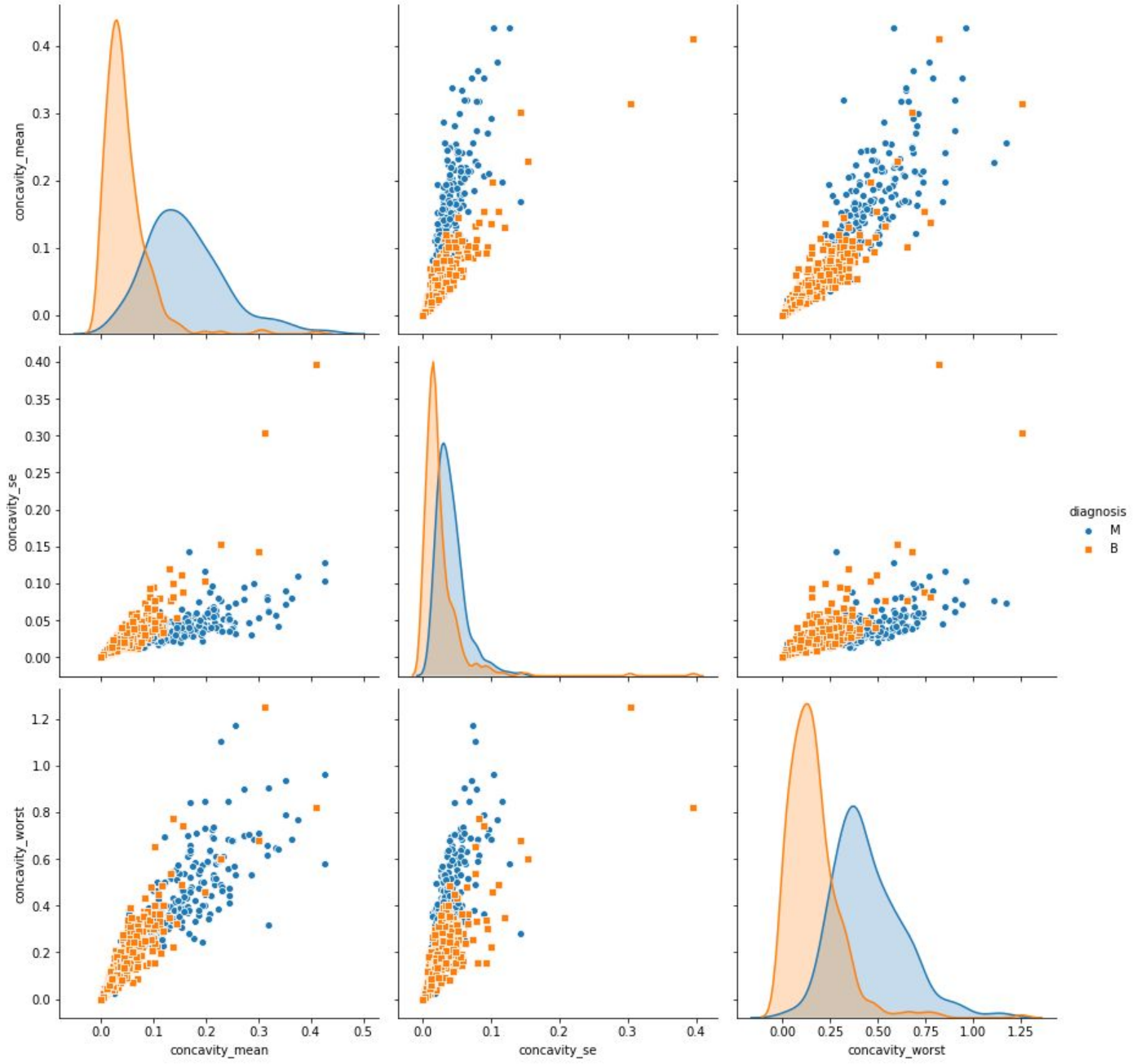


vii. Concavity Attributes

Code:

```
concavity = df[groups[6]]
fig = plt.figure()
fig = sns.pairplot(concavity, hue='diagnosis',
markers=["o", "s"],height=4)
fig.savefig('concavity_pairplot.png', bbox_inches =
'tight')
```

Output:

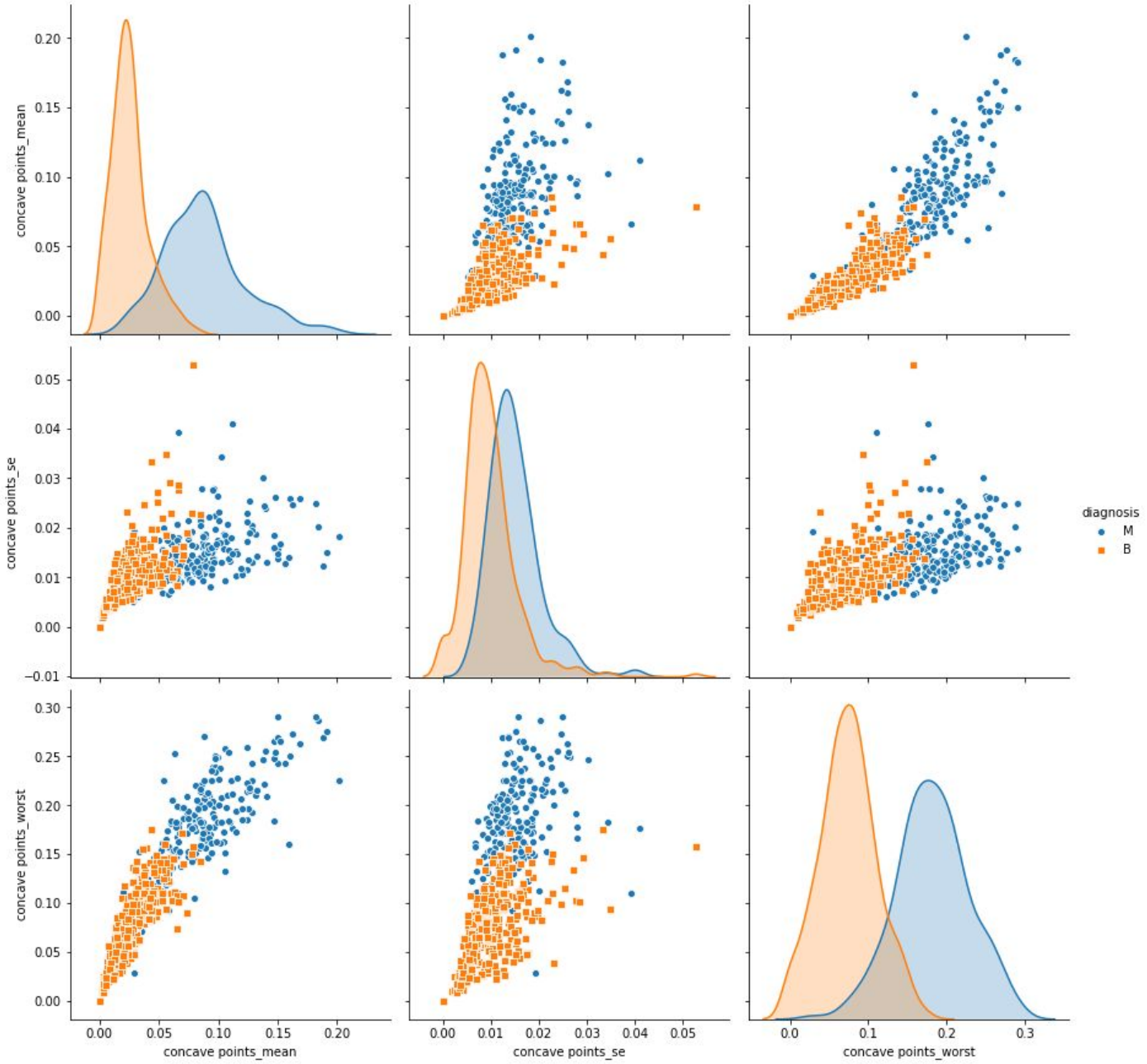


viii. <u>Concave points attributes</u>

Code:

```
concave_points = df[groups[7]]
fig = plt.figure()
fig = sns.pairplot(concave_points, hue='diagnosis',
markers=["o", "s"],height=4)
fig.savefig('concave_points_pairplot.png', bbox_inches =
'tight')
```

Output:

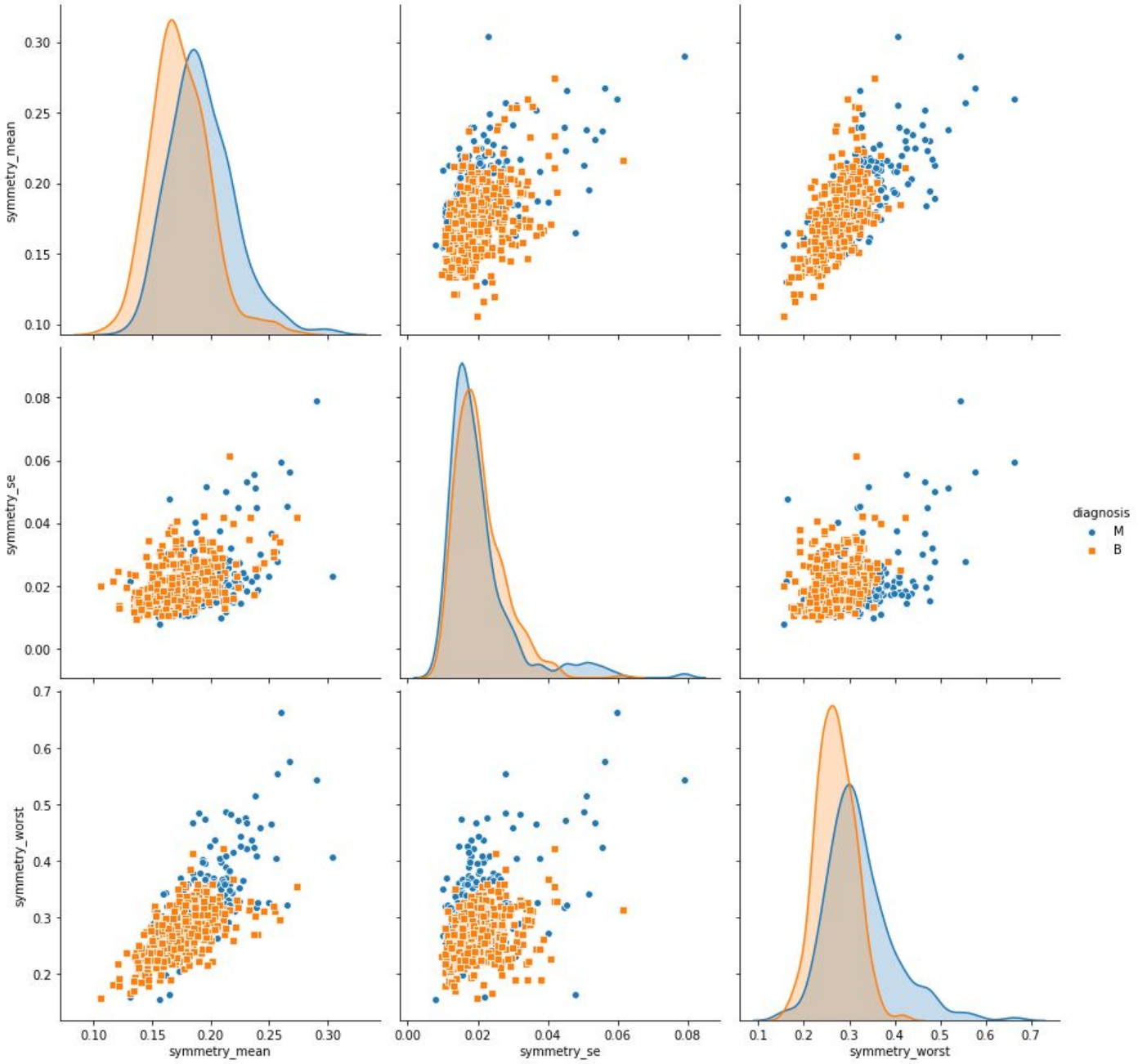


ix. Symmetry Attributes

Code:

```
symmetry = df[groups[8]]
fig = plt.figure()
fig = sns.pairplot(symmetry, hue='diagnosis',
markers=["o", "s"],height=4)
fig.savefig('symmetry_pairplot.png', bbox_inches =
'tight')
```

Output:

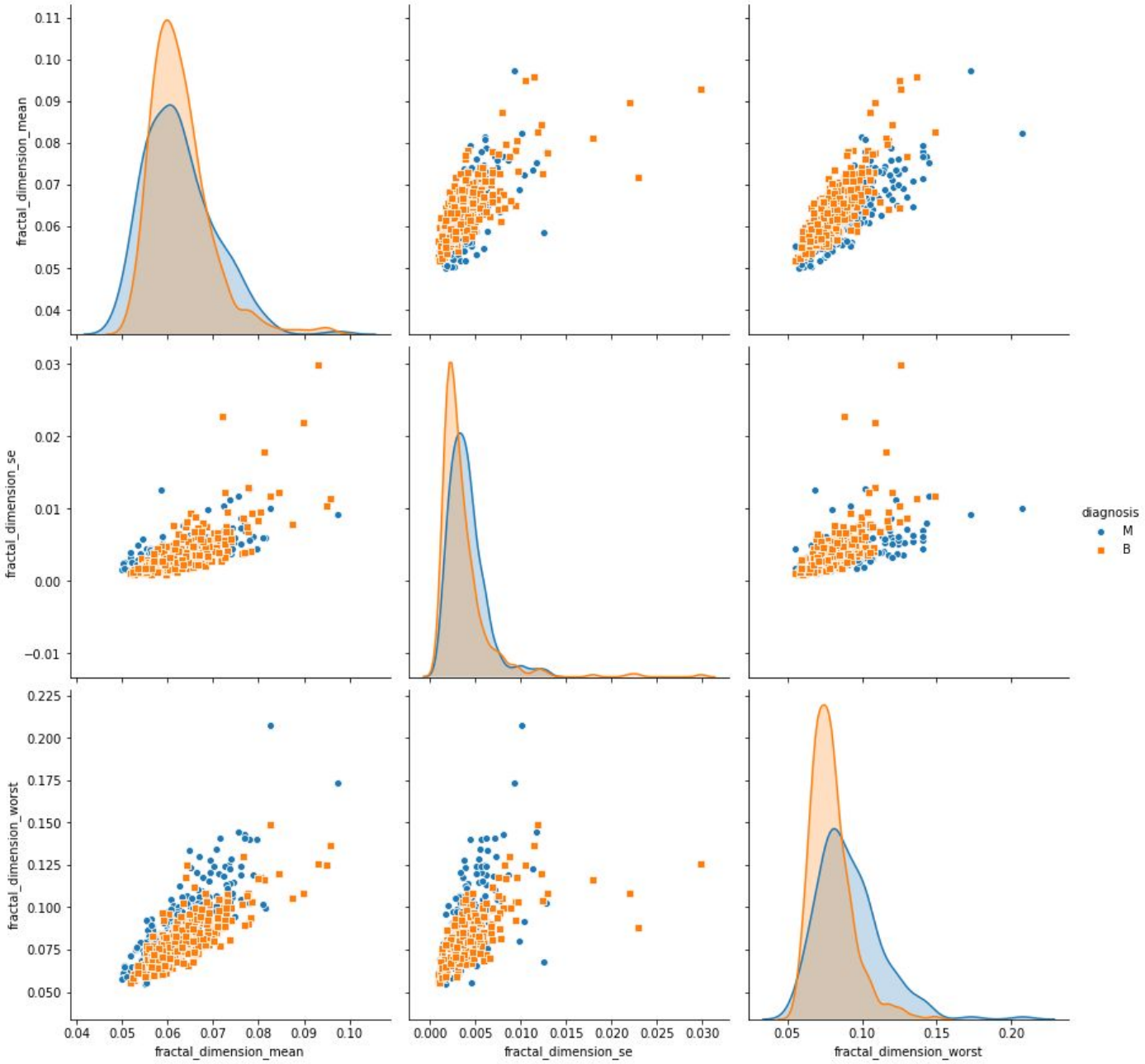


x. Fractal dimension Attributes

Code:

```
fractal_dimension = df[groups[9]]
fig = plt.figure()
fig = sns.pairplot(fractal_dimension, hue='diagnosis',
markers=["o", "s"],height=4)
fig.savefig('fractal_dimension_pairplot.png', bbox_inches
= 'tight')
```

Output:



e. Distplot

Justification:
Distplot combines histogram and Kernel Density Estimation plots. It's very useful to see the bins as well as the KDE curve. The distplot is evaluated for all the attributes in the dataset, after replacing the categorical values with binary encoding. Benign 'B' is mapped to 0 since it's the absence of cancerous tumor and Malignant 'M' is mapped to 1 since it is the presence of cancerous tissue. After that, the 30 plots are plotted in groups of 10 - mean, standard error and worst metrics, 10 each. We get an insight into how these attributes are distributed

Code to Binary Encode the categorical feature:

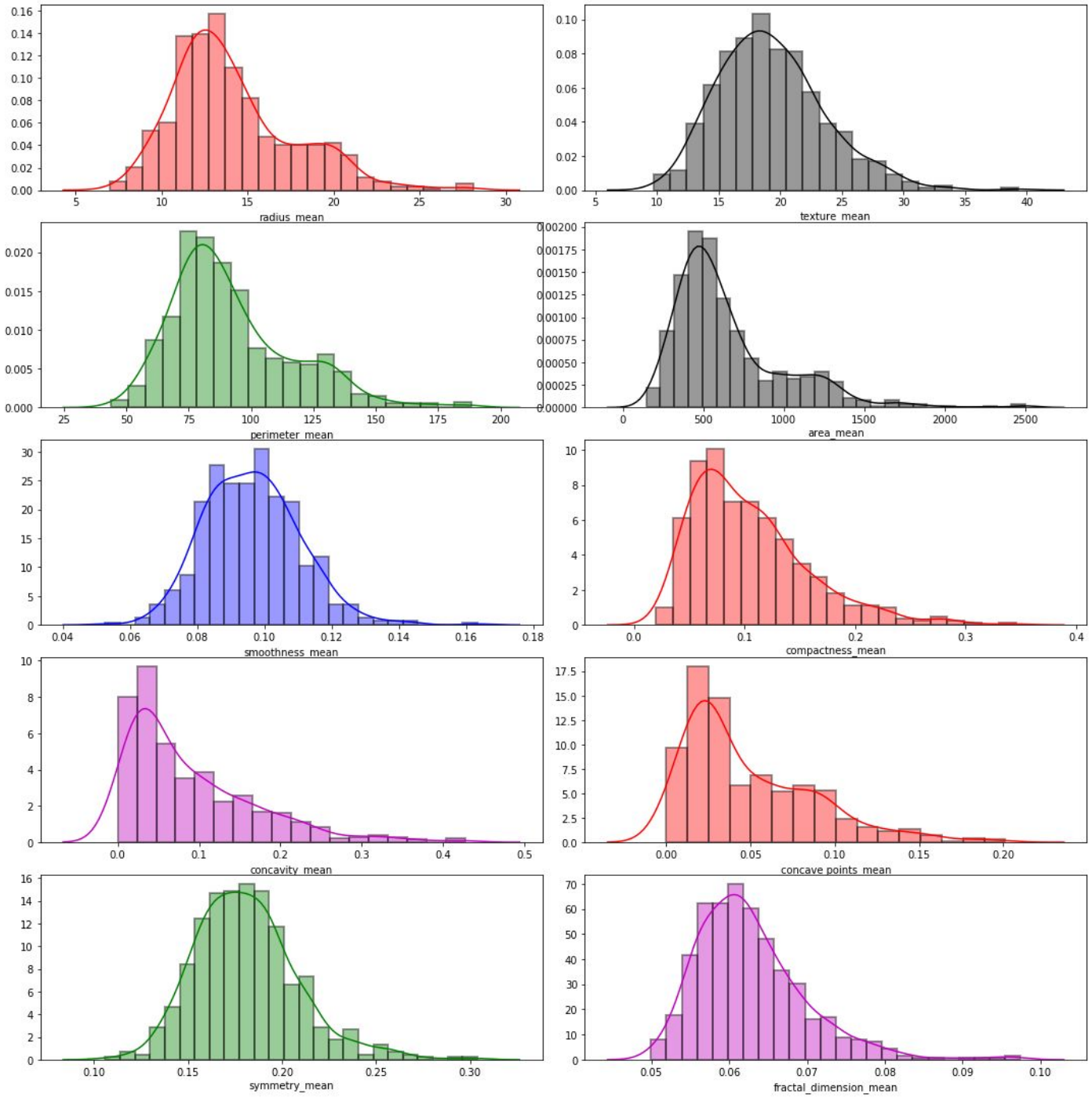
```
df['diagnosis'].replace({'B' : 0, 'M' : 1}, inplace = True)
```

i. For Mean Attributes

Code:

```
colors = ['b', 'g', 'r', 'm', 'k']
fig, axes = plt.subplots(nrows = 5, ncols = 2, figsize = (15, 15))
plt.tight_layout()
data1 = np.reshape(column[2:12], (5, 2))

for i in range(5):
    for j in range(2):
                    sns.distplot(df[data1[i][j]],  ax  =  axes[i][j],
hist_kws=dict(edgecolor=      'k',      linewidth=2),      color      =
np.random.choice(colors))

fig.savefig('distplot1.png', bbox_inches = 'tight')
```

Output:

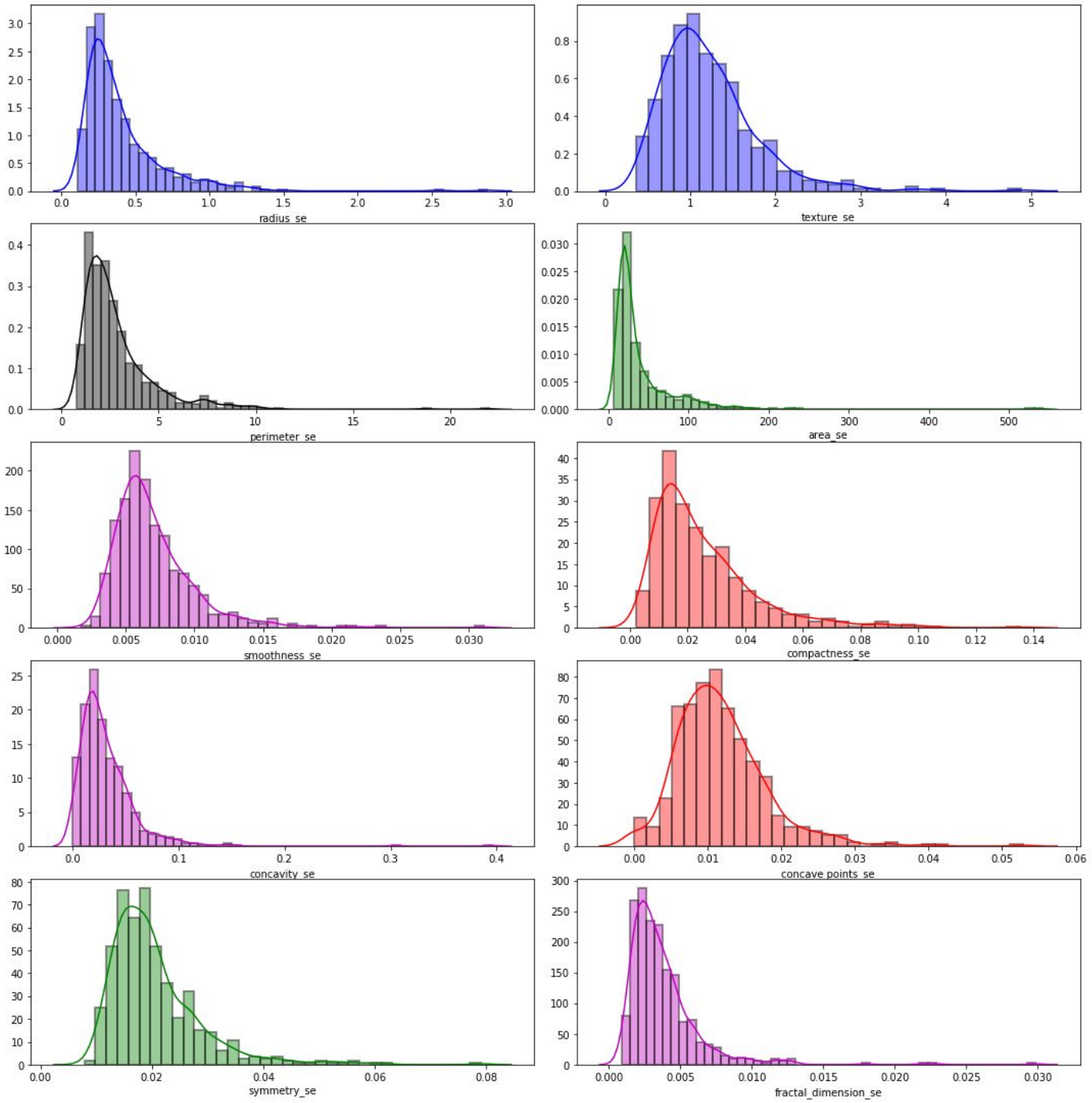


ii. For Standard Error Attributes

Code:

```
colors = ['b', 'g', 'r', 'm', 'k']
fig, axes = plt.subplots(nrows = 5, ncols = 2, figsize = (15, 15))
plt.tight_layout()
data2 = np.reshape(column[12:22], (5, 2))

for i in range(5):
    for j in range(2):
                    sns.distplot(df[data2[i][j]],  ax  =  axes[i][j],
hist_kws=dict(edgecolor=      'k',      linewidth=2),      color      =
np.random.choice(colors))

fig.savefig('distplot2.png', bbox_inches = 'tight')
```

Output:

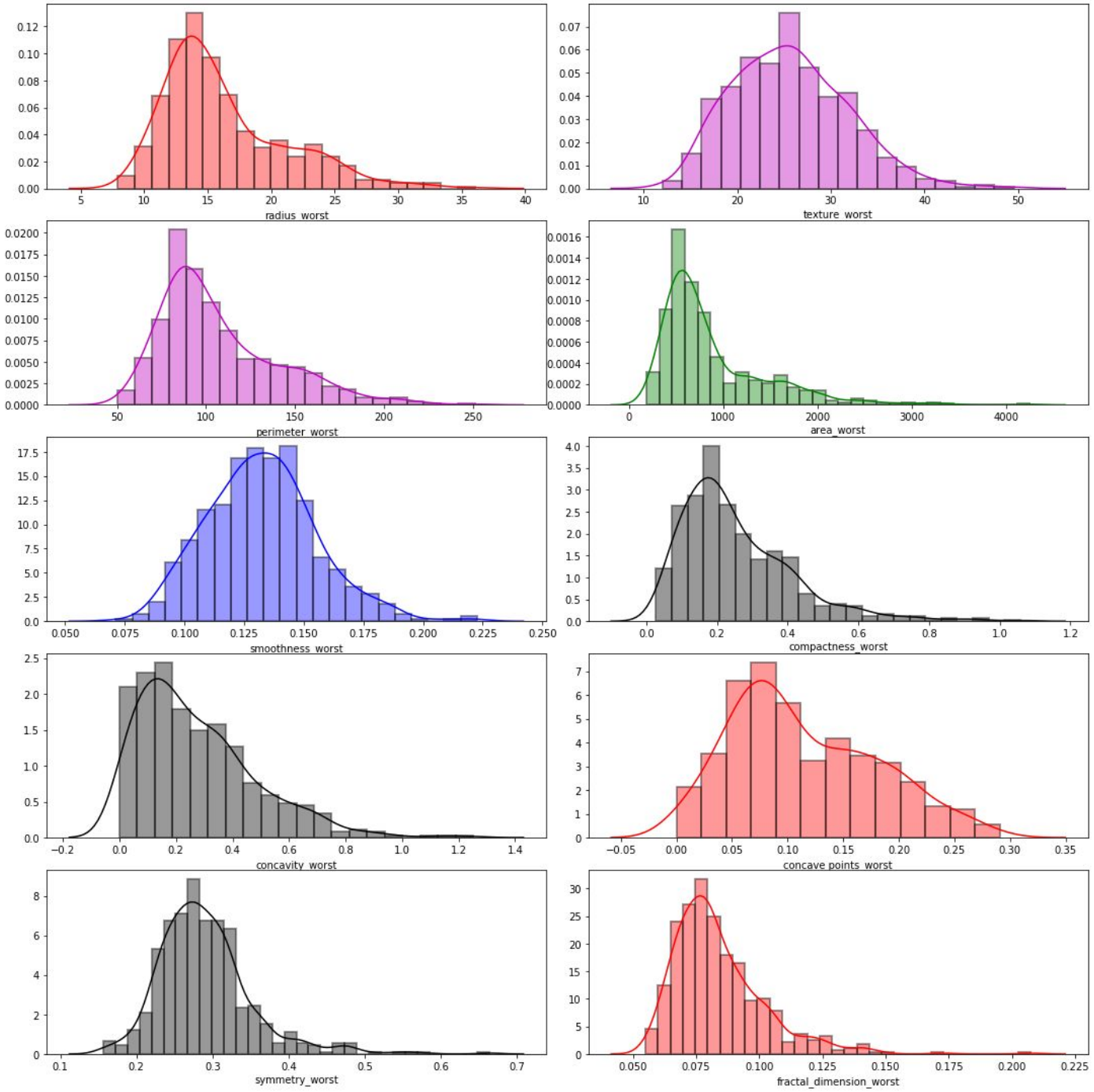


iii. For Worst Attributes

Code:

```
colors = ['b', 'g', 'r', 'm', 'k']
fig, axes = plt.subplots(nrows = 5, ncols = 2, figsize = (15, 15))
plt.tight_layout()
data3 = np.reshape(column[22:32], (5, 2))

for i in range(5):
    for j in range(2):
                    sns.distplot(df[data3[i][j]],  ax  =  axes[i][j],
hist_kws=dict(edgecolor=      'k',      linewidth=2),      color      =
np.random.choice(colors))
fig.savefig('distplot3.png', bbox_inches = 'tight')
```

Output:



f. Heatmap

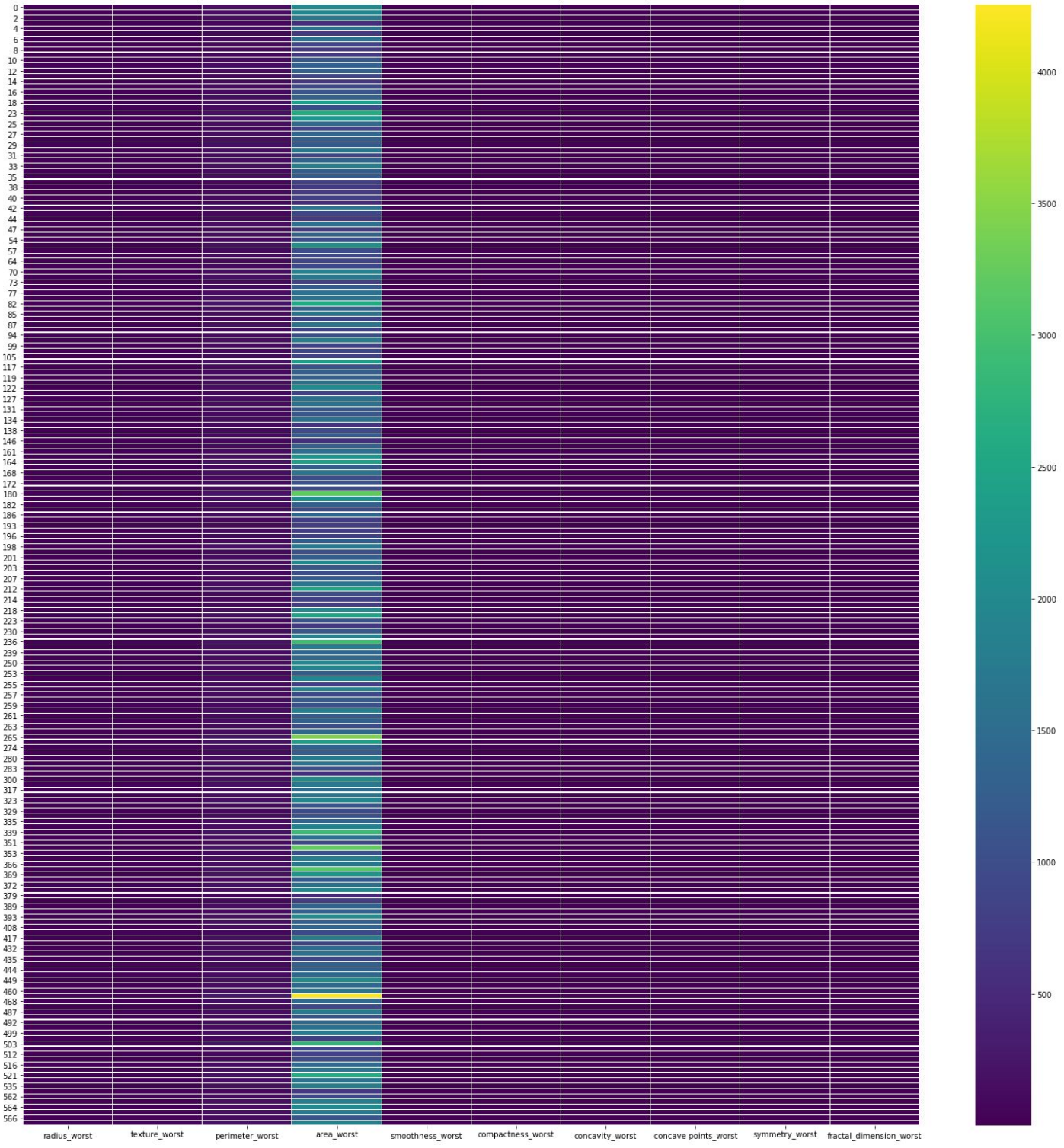
Justification:

Heatmaps are used to show relationships between two variables, one plotted on each axis. By observing how cell colors change across each axis, you can observe if there are any patterns in value for one or both variables. Here, we are observing what are the worst values of the observed sample that correspond to a Malignant tumor.

Code:

```
fig = plt.figure(figsize = (20, 20))
ax = fig.add_axes([0, 0, 1, 1])
sns.heatmap(df1[df1['diagnosis'] == 1][column[22:]], ax = ax, cmap =
'viridis', linewidths = .1)
fig.savefig('heatmap.png', bbox_inches = 'tight')
```

Output:



g. Correlation Plot

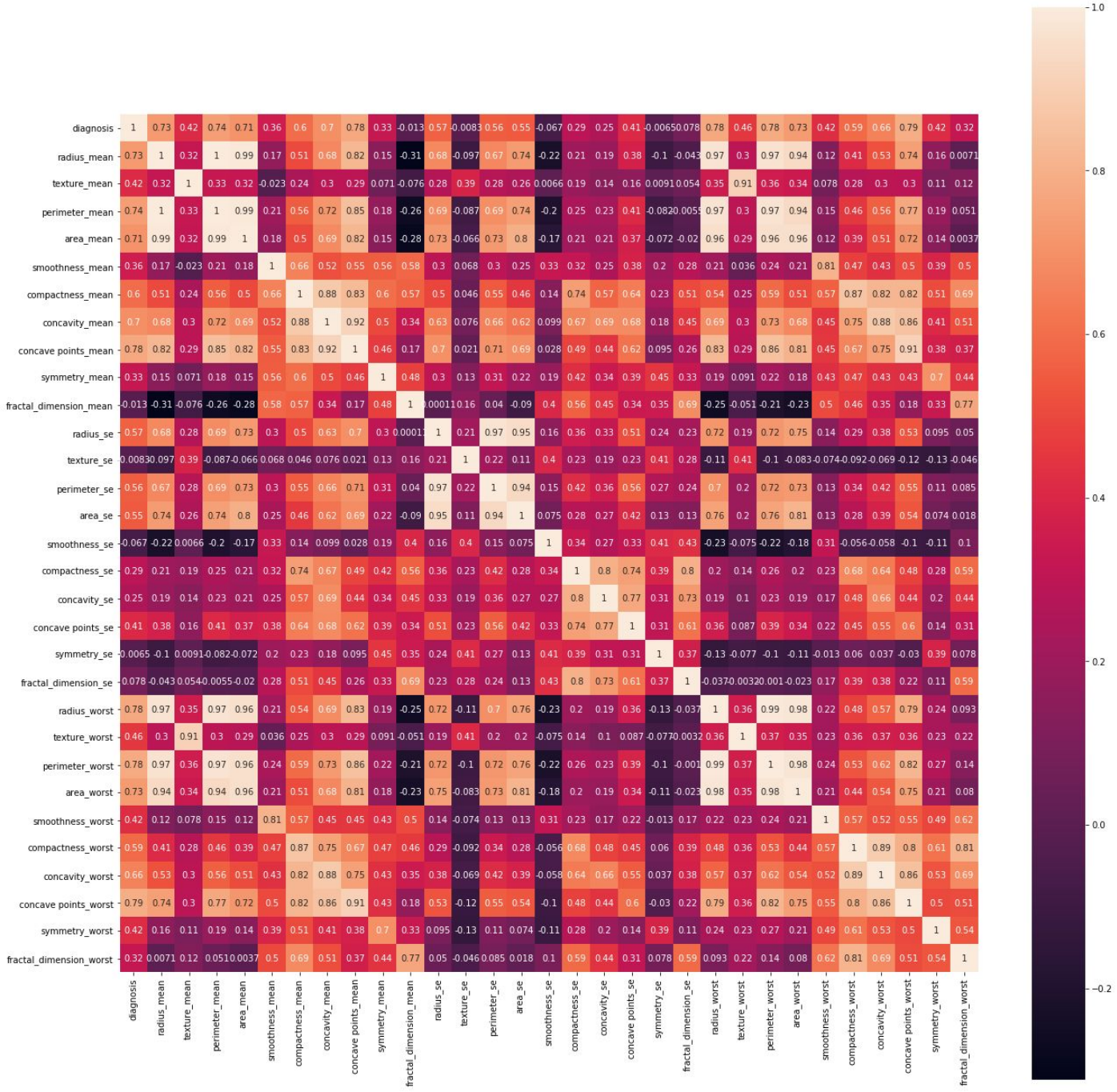
Justification:

The following is a correlation plot. First the correlation matrix has to be evaluated. If n is the number of attributes in the dataset, then the correlation matrix is nxn where a[i][j] is the correlation coefficient of ith and jth attributes. Finally this matrix is visualized as a heatmap. It helps us visualize how strongly or weakly all the attributes are correlated with each other at a glance.

Code:

```
correlation = df1.corr()
# correlation matrix of the dataset
fig = plt.figure(figsize = (17, 17))
ax = fig.add_axes([0, 0, 1, 1])
sns.heatmap(correlation, ax = ax, square = True, annot = True, linecolor = 'w')
fig.savefig('correlation_heatmap.png', bbox_inches = 'tight')
```

Output:



h. Scatter Plot

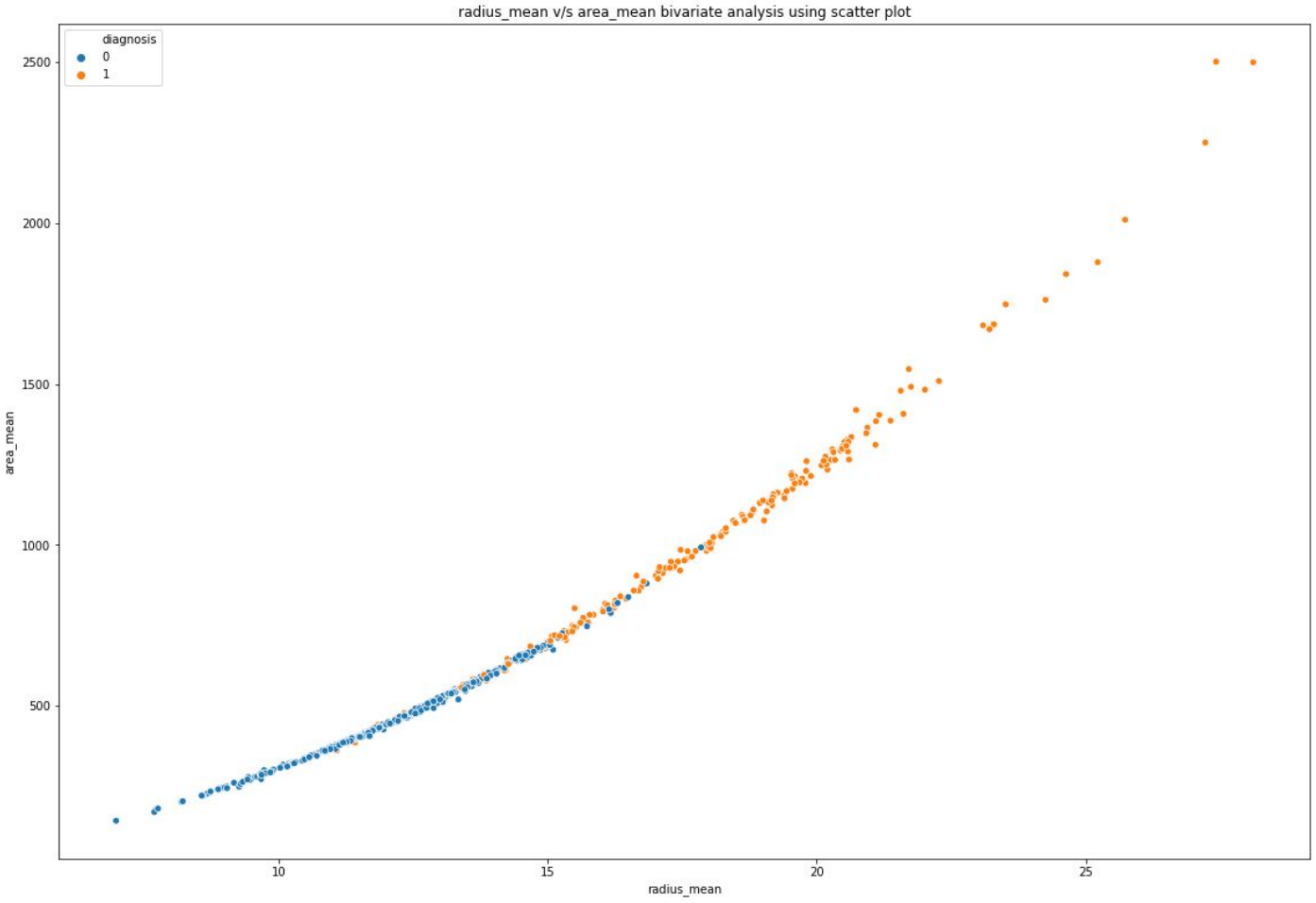
Justification:

Here, a scatter plot is used to perform bivariate analysis of radius_mean versus area_mean. Since from previous visualizations we can see that the nucleus swab image is more on the concave side, we can expect the area to be very close to radius' square. And we can see from the scatter plot that it is a parabolic curve. We can also see that Benign tumors have less radius and area than Malignant tumour nuclei images.

Code:

```
fig = plt.figure(figsize = (15, 10))
ax = fig.add_axes([0, 0, 1, 1])
sns.scatterplot(x = 'radius_mean', y = 'area_mean', data = df1,
hue = 'diagnosis', ax = ax)
ax.set_title('radius_mean v/s area_mean bivariate analysis using
scatter plot')
fig.savefig('scatterplot.png', bbox_inches = 'tight')
```

Output:



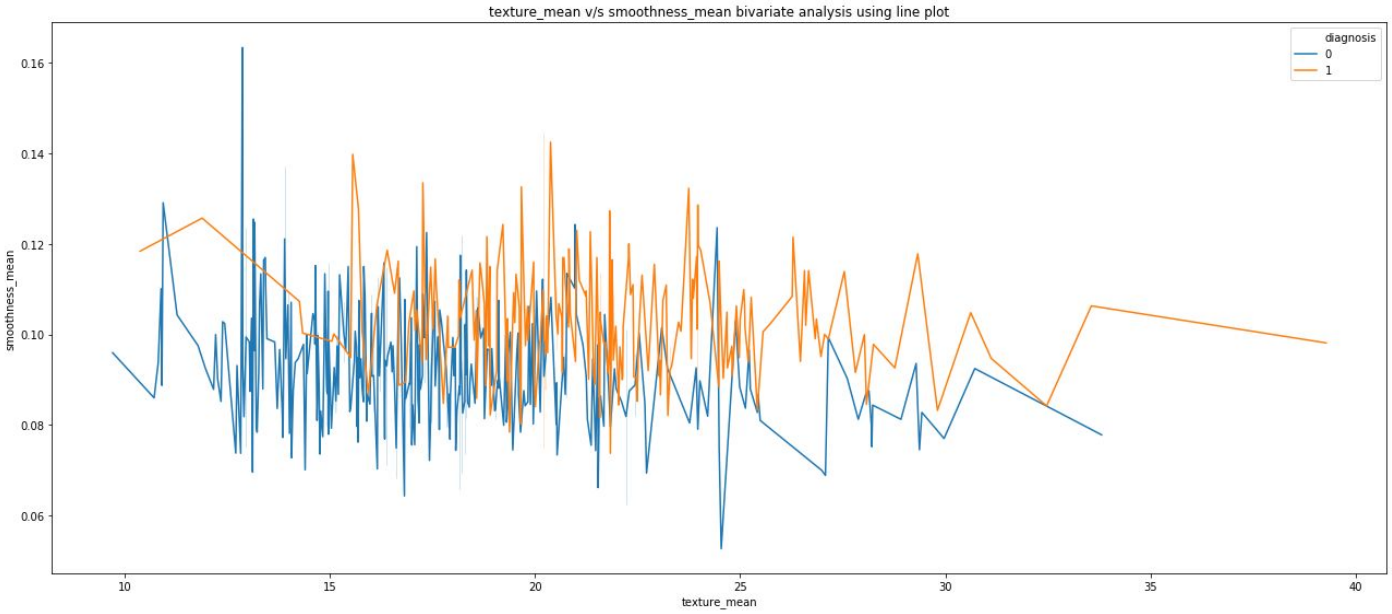
i. Line Plot

Justification:
A line plot is used in the following visualization to perform bivariate analysis for texture_mean versus smoothness_mean. i.e. how texture varies as compared to smoothness. The smoother the surface, the lower the texture_mean value.

Code:

```
fig = plt.figure(figsize = (17, 7))
ax = fig.add_axes([0, 0, 1, 1])
sns.lineplot(x = 'texture_mean', y = 'smoothness_mean', data = df1, hue = 'diagnosis', ax = ax)
ax.set_title('texture_mean v/s smoothness_mean bivariate analysis using line plot')
fig.savefig('lineplot.png', bbox_inches = 'tight')
```

Output:



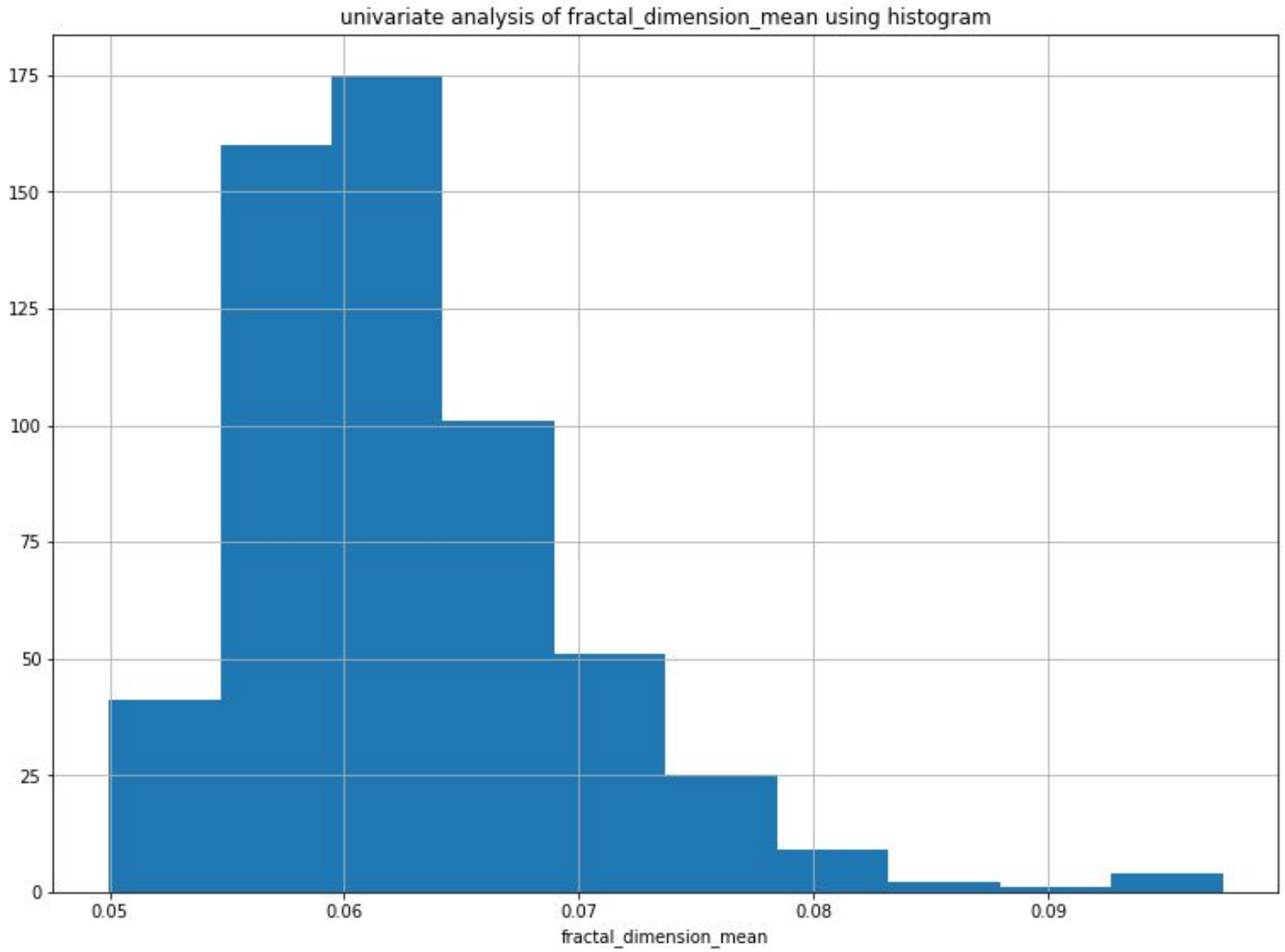
j. Histogram

Justification:
A histogram is plotted for univariate analysis of fractal_dimension_mean attribute. We can see that most of the values lie between 0.05 and 0.07.

Code:

```
fig = plt.figure(figsize = (10, 7))
ax = fig.add_axes([0, 0, 1, 1])
df['fractal_dimension_mean'].hist(ax = ax)
ax.set_xlabel('fractal_dimension_mean')
ax.set_title('univariate analysis of fractal_dimension_mean using histogram')
fig.savefig('histplot.png', bbox_inches = 'tight')
```

Output:



k. Kernel Density Estimation Plot

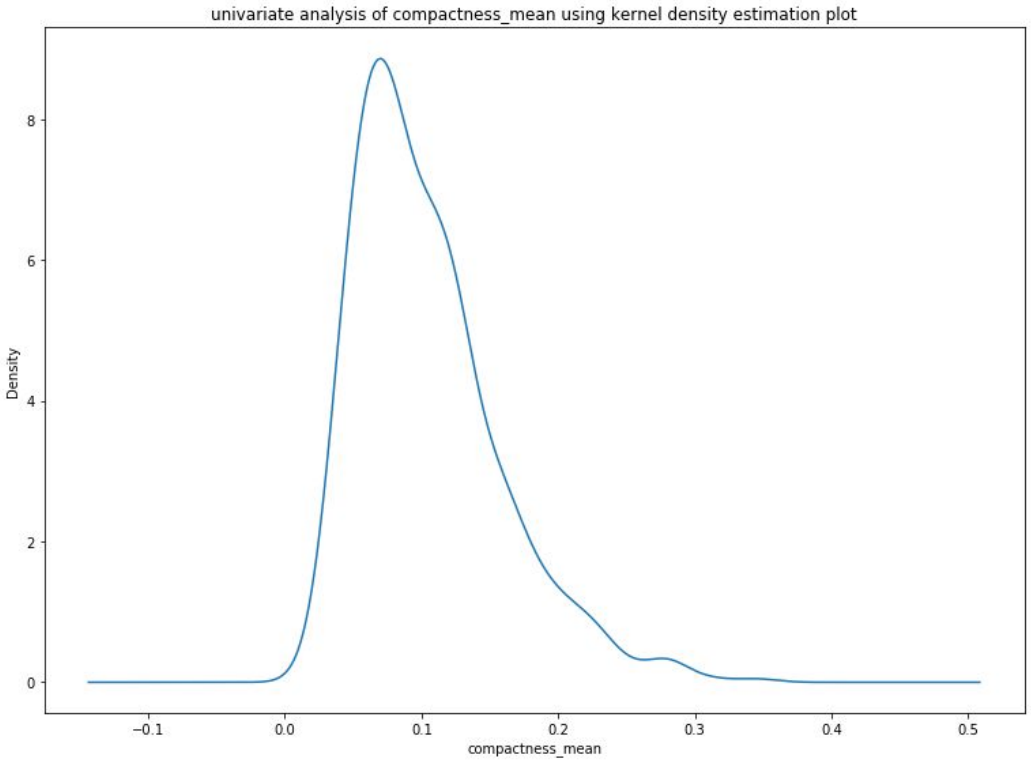
Justification:

One of the issues with using a histogram as a density estimator is that the choice of bin size and location can lead to representations that have qualitatively different features. KDE plot is a smoothed-out plot, with a Gaussian distribution contributed at the location of each input point, gives a much more accurate idea of the shape of the data distribution, and one which has much less variance. KDE plot can be useful if you want to visualize just the "shape" of some data, as a kind of continuous replacement for the discrete histogram. It can also be used to generate points that look like they came from a certain dataset - this behavior can power simple simulations, where simulated objects are modeled off of real data. Here, the KDE of compactness_mean is plotted to perform univariate analysis on the same.

Code:

```
fig = plt.figure(figsize = (10, 7))
ax = fig.add_axes([0, 0, 1, 1])
df['compactness_mean'].plot.kde(ax = ax)
ax.set_xlabel('compactness_mean')
ax.set_title('univariate analysis of compactness_mean using kernel density
estimation plot')
fig.savefig('kdeplot.png', bbox_inches = 'tight')
```

Output:



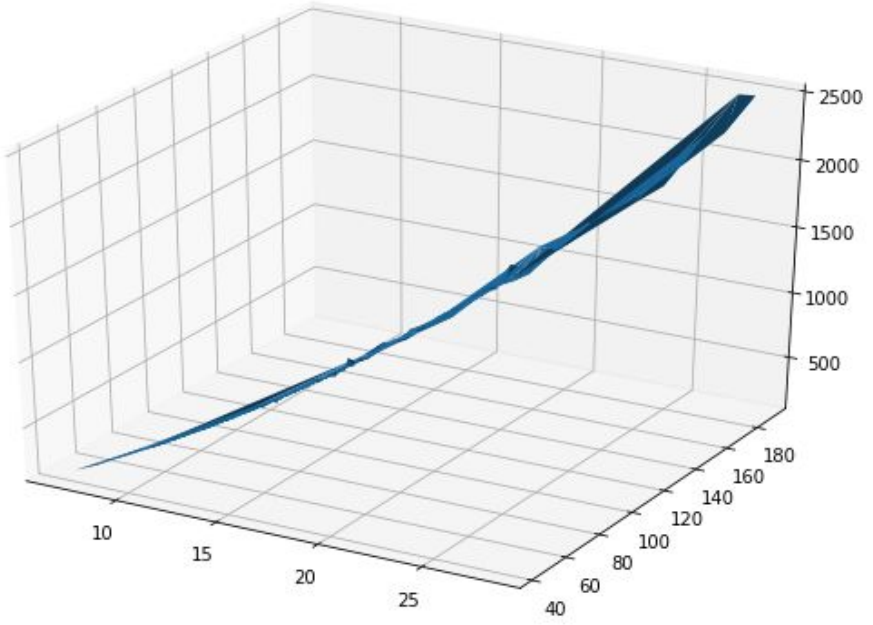
l. Surface Plot

Justification:
The following plot is a 3D visualization of radius_mean, perimeter_mean, and area_mean. Even from the x, y, z ticks alone, we can see that perimeter is the order of 6*radius and area is the order of radius squared.

Code:

```
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(figsize = (10, 7))
axes = fig.gca(projection ='3d')
axes.plot_trisurf(df1['radius_mean',df1['perimeter_mean'],df1['area_mean'])
fig.savefig('surfaceplot.png', bbox_inches = 'tight')
```

Output:



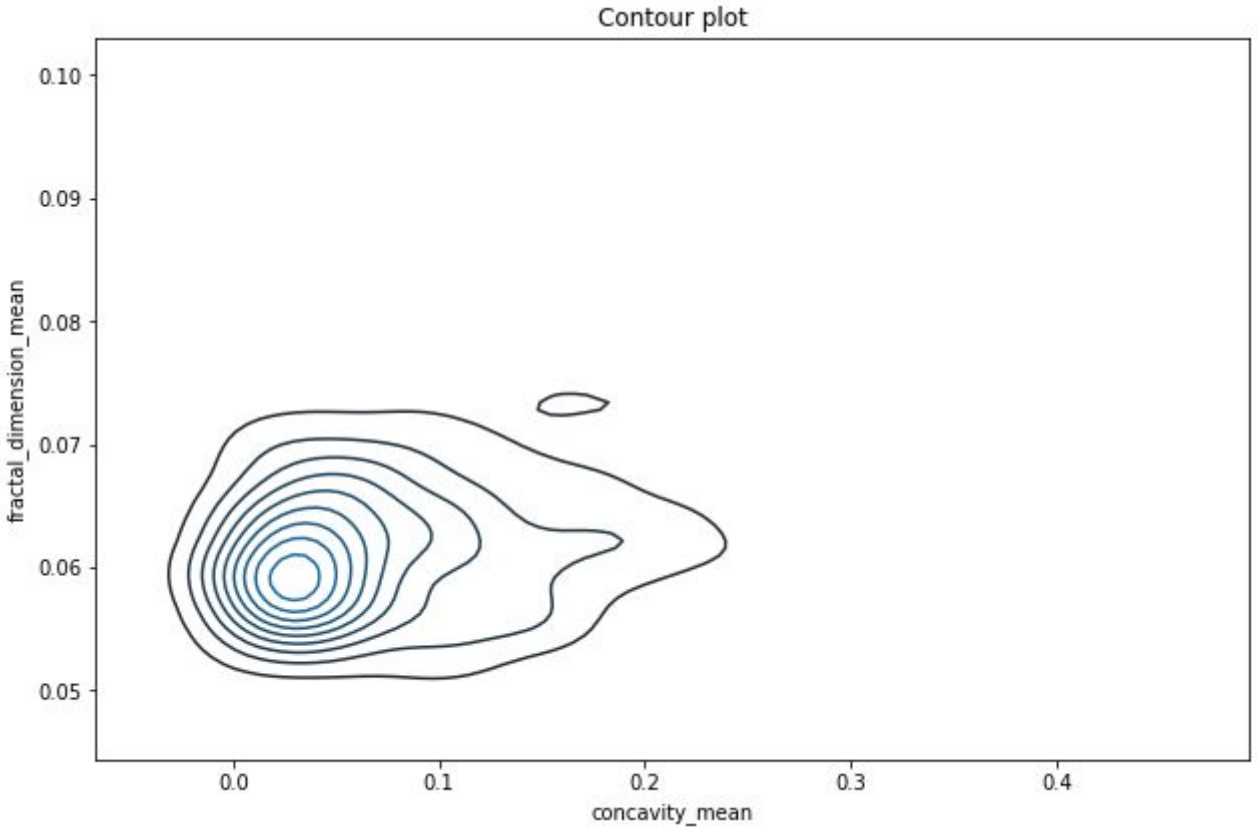
m. Contour Plot

Justification:
Contour plot is yet another visualization of how the distributions of concavity_mean and fractal_dimension_mean are varying with each other. We can easily identify the high density and outlier regions.

Code:

```
fig = plt.figure(figsize = (8, 5))
ax = fig.add_axes([0, 0, 1, 1])
sns.kdeplot(df['concavity_mean'],
df['fractal_dimension_mean'], ax = ax)
ax.set_title('Contour plot')
fig.savefig('contourplot.png', bbox_inches = 'tight')
```

Output:



n. Violin Plot

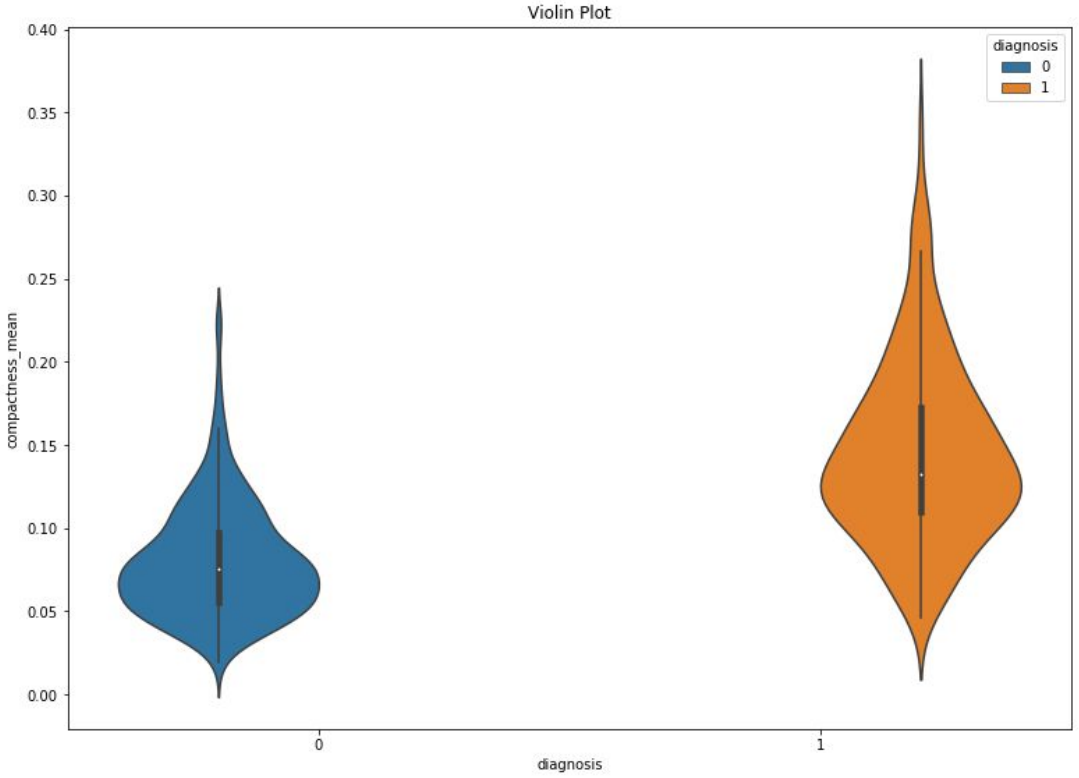
Justification:
A Violin Plot is used to visualise the distribution of the data and its probability density. This chart is a combination of a Box Plot and a Density Plot that is rotated and placed on each side, to show the distribution shape of the data. The following plot is a visual representation of how compactness_mean is distributed over the two categories.

Code:

```
fig = plt.figure(figsize = (10, 7))
axes = fig.add_axes([0, 0, 1, 1])
sns.violinplot(x='diagnosis', y='compactness_mean', data=df, hue='diagnosis',
ax = axes)
axes.set_title('Violin Plot')
```

```
fig.savefig('violinplot.png', bbox_inches = 'tight')
```

Output:



o. Facet Grid

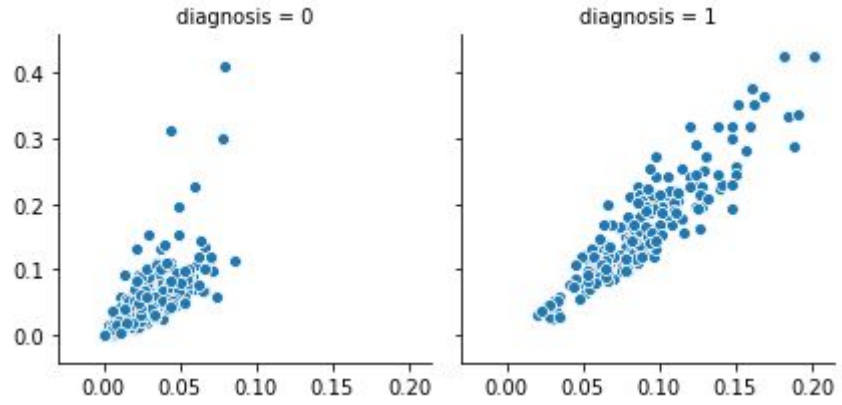
Justification:

Facet Grid forms a matrix of panels defined by row and column faceting variables. It is most useful when you have two discrete variables, and all combinations of the variables exist in the data. Here we have one discrete variable (categorical) that is *diagnosis* and the plot visualizes how the means of concave points and concavity of the sample are distributed against each other when grouped by the diagnosis.

Code:

```
g = sns.FacetGrid(df1, col = 'diagnosis')
g.map_dataframe(sns.scatterplot, x = 'concave points_mean', y = 'concavity_mean')
g.savefig('FacetGridplot.png')
```

Output:



p. Swarm Plot

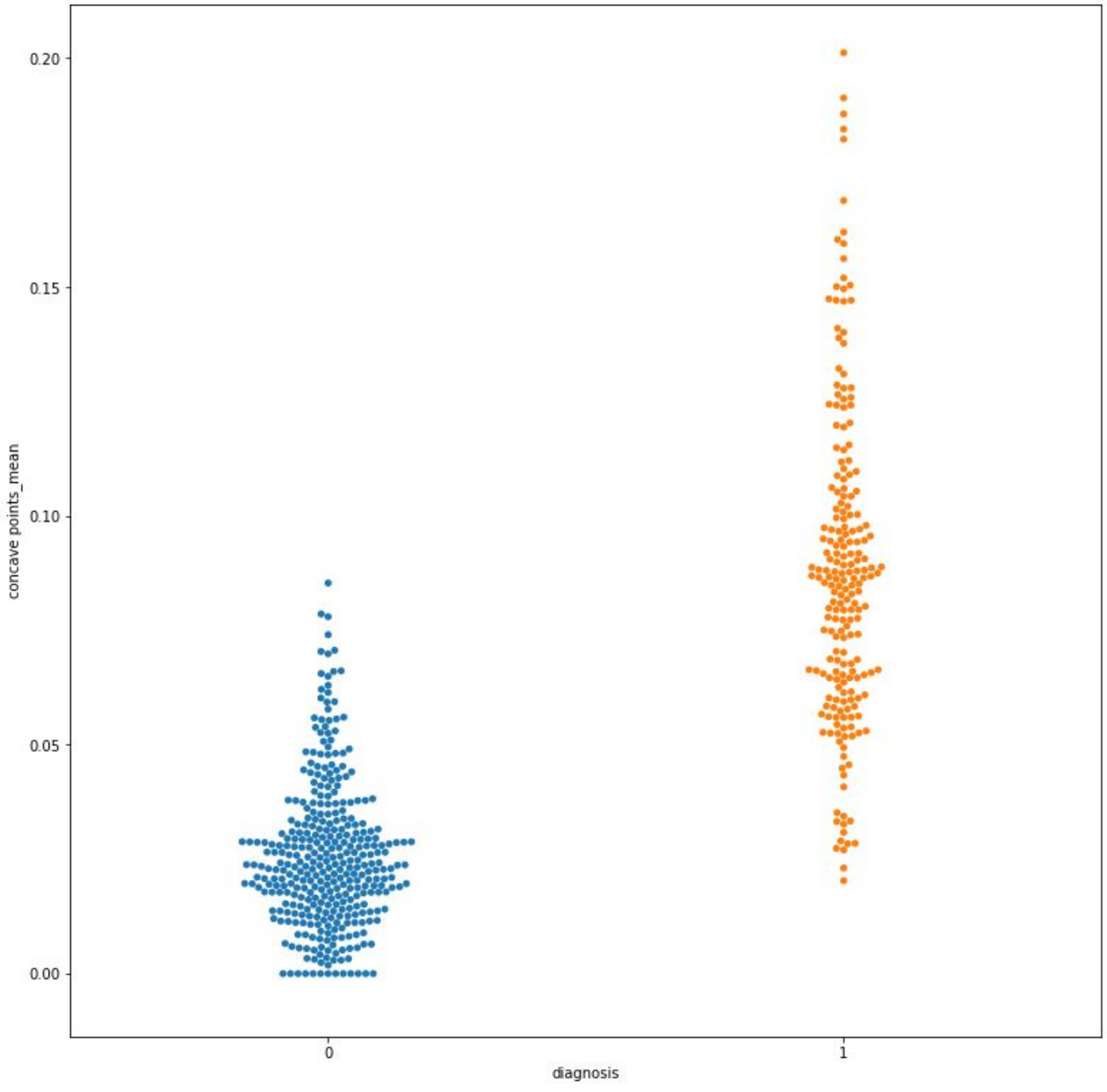
Justification:

Swarm plots plot every point in the dataset and these points are non overlapping. It makes it easy to see all the outliers. And when we use it with a categorical attribute, we can easily see the distribution. Here, concave points mean of the sample is visualized grouped by the categories.

Code:

```
fig = plt.figure(figsize = (10, 10))
axes = fig.add_axes([0, 0, 1, 1])
sns.swarmplot(x = df['diagnosis'], y = df['concave points_mean'], ax = axes)
fig.savefig('swarmplot.png', bbox_inches = 'tight')
```

Output:



q. Scatter Matrix

Justification:

A scatter matrix consists of several pairwise scatter plots of variables presented in a matrix format. It can be used to determine whether the variables are correlated and whether the correlation is positive or negative. Here, a scatter matrix of all attributes under the mean metric is visualized.

Code:

```
phi = pd.plotting.scatter_matrix(df1[column[1:12]], figsize = (25, 25))
```

Output: