Breast Cancer Prediction Using Python

Tenzin Lhundup

RollNo:21118104

Abstract

In this document, I will summarize the process of predicting breast cancer using Python. The dataset used for this prediction is a breast cancer dataset, and I will explore both Weak Learner and Strong Learner classification models.

1 Data Preparation

I start by importing the necessary libraries and reading the breast cancer dataset from a CSV file.

```
import numpy
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

# Reading data from the file
df = pd.read_csv("data.csv")
```

I then check the basic information about the dataset using `df.info()` and find that there is an empty column named "Unnamed: 32." I remove this column to clean the data.

```
# Remove the column with null values
df = df.dropna(axis=1)
```

2 Data Exploration

I explore the dataset by checking the count of malignant (M) and benign (B) cells and visualizing it using a count plot.

```
# Get the count of malignant (M) and benign (B) cells
df['diagnosis'].value_counts()

# Visualize the count
sns.countplot(df['diagnosis'], label="count")
```

3 Data Preprocessing

I perform label encoding to convert the values of "M" and "B" into 1 and 0, respectively, in the "diagnosis" column.

```
# Label encoding
from sklearn.preprocessing import LabelEncoder
labelencoder_Y = LabelEncoder()
df.iloc[:, 1] = labelencoder_Y.fit_transform(df.iloc[:, 1].values)
```

4 Data Visualization and Correlation

I visualize the data by creating a pair plot of the first five columns and calculate the correlation matrix to understand the relationships between features.

```
# Data visualization with a pair plot
sns.pairplot(df.iloc[:, 1:5], hue="diagnosis")

# Calculate the correlation matrix
correlation_matrix = df.iloc[:, 1:31].corr()
```

5 Data Splitting and Feature Scaling

I split the dataset into independent (X) and dependent (Y) datasets and further divide the data into training and test sets. Feature scaling is applied to standardize the data.

```
# Split the dataset into independent (X) and dependent (Y) datasets
X = df.iloc[:, 2:31].values
Y = df.iloc[:, 1].values

# Split the data into training and test datasets
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.20, random_state=0)

# Feature scaling
from sklearn.preprocessing import StandardScaler
X_train = StandardScaler().fit_transform(X_train)
X_test = StandardScaler().fit_transform(X_test)
```

6 Weak Learner and Strong Learner Models

In this step, I build both Weak Learner and Strong Learner models for breast cancer prediction.

I first define the Weak Learner, which is a Decision Tree with a maximum depth of 1.

```
# Instantiate the Weak Learner (Decision Tree)
weak_learner = DecisionTreeClassifier(max_depth=1)
```

Then, I define the Strong Learners: Random Forest, AdaBoost, and Gradient Boosting.

```
# Instantiate the Strong Learners (Random Forest, AdaBoost, GradientBoost)
random_forest = RandomForestClassifier(n_estimators=100, random_state=0)
adaboost = AdaBoostClassifier(base_estimator=weak_learner, n_estimators=100, random_state=0)
gradientboost = GradientBoostingClassifier(n_estimators=100, random_state=0)

# Fit the Strong Learners
random_forest.fit(X_train, Y_train)
adaboost.fit(X_train, Y_train)
gradientboost.fit(X_train, Y_train)
```

7 Model Evaluation

I evaluate the accuracy of the Strong Learner models on the test data.

```
# Get the accuracy of the Strong Learners on the test data
print("Random Forest Accuracy:", random_forest.score(X_test, Y_test))
print("AdaBoost Accuracy:", adaboost.score(X_test, Y_test))
print("Gradient Boost Accuracy:", gradientboost.score(X_test, Y_test))
```

8 Conclusion

In this document, I have explored the process of predicting breast cancer using Python. I started by preparing the data, exploring it, and preprocessing it. I visualized the data and calculated the correlation matrix. I then split the data into training and test sets, performed feature scaling, and built Weak Learner and Strong Learner models for breast cancer prediction.

The accuracy of the Strong Learner models on the test data was evaluated, and the results were provided.

Breast cancer prediction is a critical task in the field of medical science, and machine learning models can assist in diagnosing patients more accurately. These models can be further fine-tuned to enhance their predictive capabilities.

Title:Breast Cancer data classification using Weak Learner and Strong Learner classification .

RollNumber:21118104

Assignment 1

Branch:IT

Semester:5th

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

In [3]:
```
df=pd.read_csv("C:\\Users\\tenzin Lhum\\OneDrive\\Desktop\\breast-cancer.csv")
```

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

Out[4]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mea
0	842302	M	17.99	10.38	122.80	1001.0	0.1184
1	842517	M	20.57	17.77	132.90	1326.0	0.0847
2	84300903	M	19.69	21.25	130.00	1203.0	0.1096
3	84348301	M	11.42	20.38	77.58	386.1	0.1425
4	84358402	M	20.29	14.34	135.10	1297.0	0.1003
5	843786	M	12.45	15.70	82.57	477.1	0.1278
6	844359	M	18.25	19.98	119.60	1040.0	0.0946
7	84458202	M	13.71	20.83	90.20	577.9	0.1189
8	844981	M	13.00	21.82	87.50	519.8	0.1273
9	84501001	M	12.46	24.04	83.97	475.9	0.1186

10 rows × 32 columns

In [6]:
```
df.shape
```

Out[6]:
```
(569, 32)
```

In [7]:
```
df.columns
```

Out[7]:
```
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')
```

In [9]:
```
#To check if their is any null values or not
df.isna().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
```

In [11]:
```
df.info(show_counts=True,memory_usage=True)
```

```
<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 [12]:
```
df.describe()
```

Out[12]:

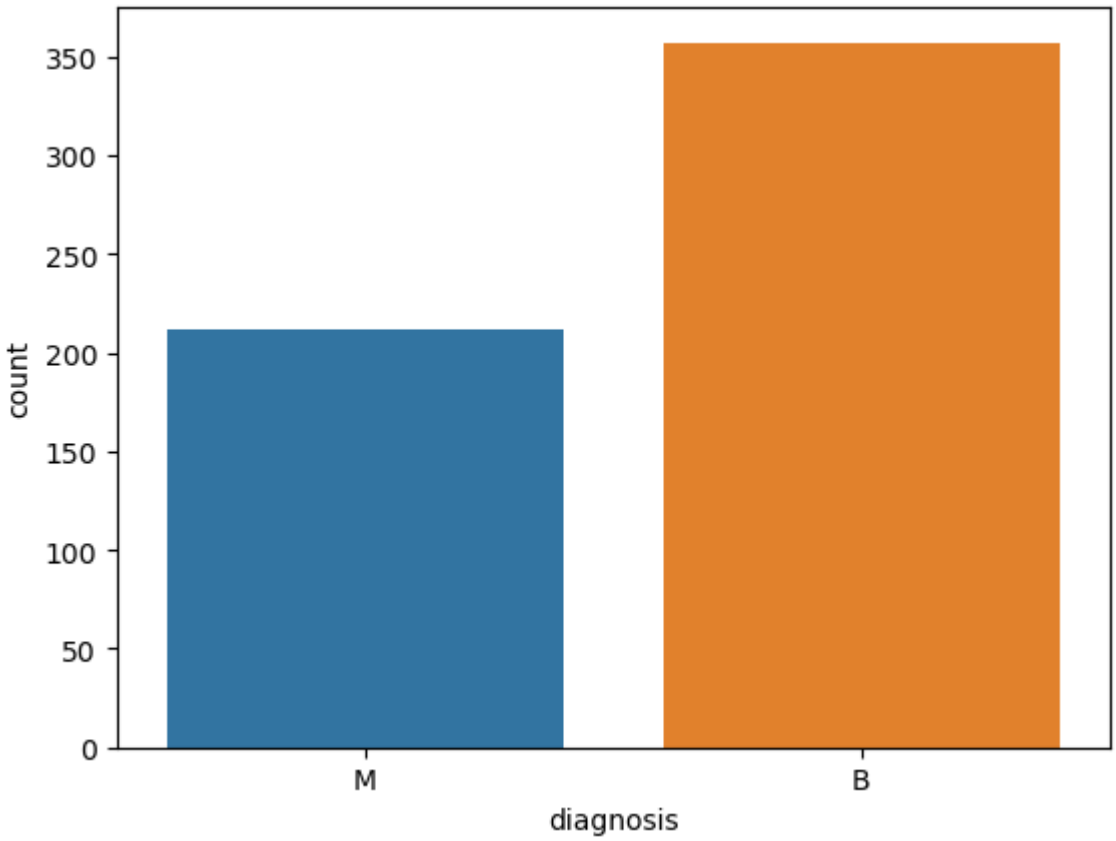
	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.000000
mean	3.037183e+07	14.127292	19.289649	91.969033	654.889104	0.096360
std	1.250206e+08	3.524049	4.301036	24.298981	351.914129	0.014064
min	8.670000e+03	6.981000	9.710000	43.790000	143.500000	0.052630
25%	8.692180e+05	11.700000	16.170000	75.170000	420.300000	0.086370
50%	9.060240e+05	13.370000	18.840000	86.240000	551.100000	0.095870
75%	8.813129e+06	15.780000	21.800000	104.100000	782.700000	0.105300
max	9.113205e+08	28.110000	39.280000	188.500000	2501.000000	0.163400

8 rows × 31 columns

In [13]:
```
df["diagnosis"].value_counts()
```

Out[13]:
```
B    357
M    212
Name: diagnosis, dtype: int64
```

In [16]:
```
sns.countplot(data=df, x="diagnosis")
plt.show()
```



In [18]:
```
# 'M' is encoded as 1
# 'B' is encoded as 0
from sklearn.preprocessing import LabelEncoder
LabelEncoder_x=LabelEncoder()
df["diagnosis"]= LabelEncoder_x.fit_transform(df['diagnosis'].values)
```

In [19]:
```
df.head(10)
```

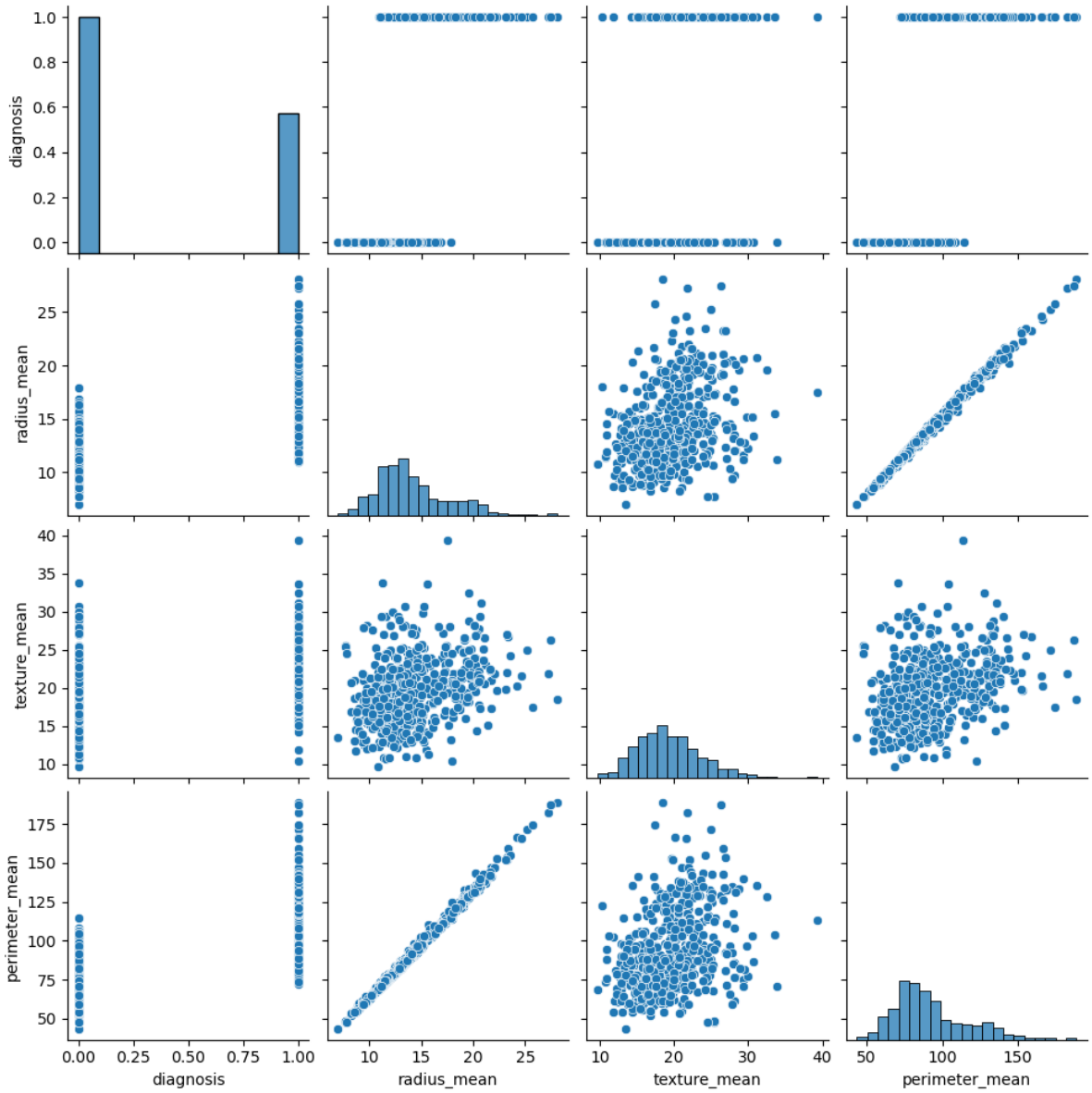
Out[19]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mea
0	842302	1	17.99	10.38	122.80	1001.0	0.1184
1	842517	1	20.57	17.77	132.90	1326.0	0.0847
2	84300903	1	19.69	21.25	130.00	1203.0	0.1096
3	84348301	1	11.42	20.38	77.58	386.1	0.1425
4	84358402	1	20.29	14.34	135.10	1297.0	0.1003
5	843786	1	12.45	15.70	82.57	477.1	0.1278
6	844359	1	18.25	19.98	119.60	1040.0	0.0946
7	84458202	1	13.71	20.83	90.20	577.9	0.1189
8	844981	1	13.00	21.82	87.50	519.8	0.1273
9	84501001	1	12.46	24.04	83.97	475.9	0.1186

10 rows × 32 columns

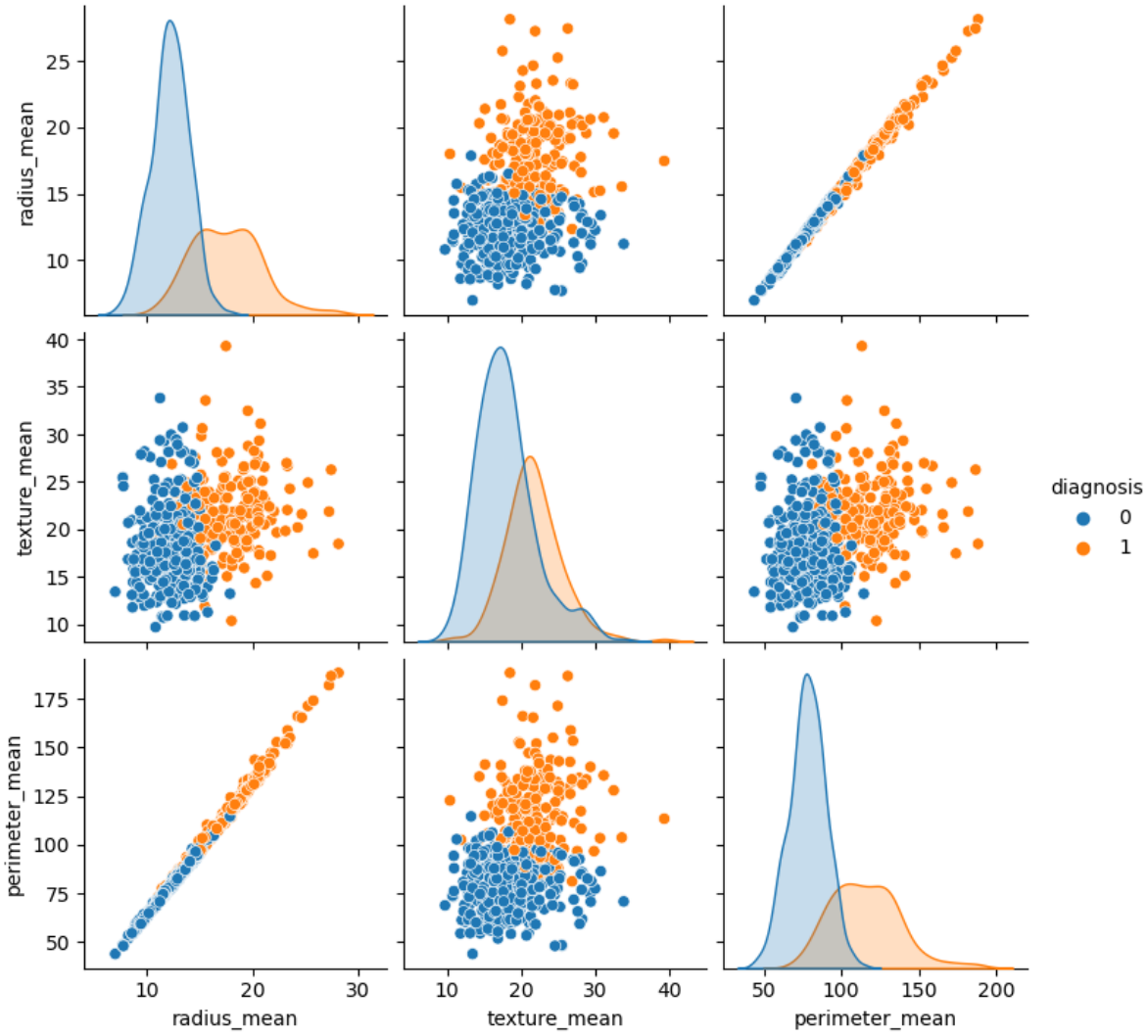
In [20]:
```
#iloc indicate the location of a column with the start of the 0 base index ifdf.ilo
sns.pairplot(df.iloc[:, 1:5])
```

Out[20]:
```
<seaborn.axisgrid.PairGrid at 0x19e214da990>
```



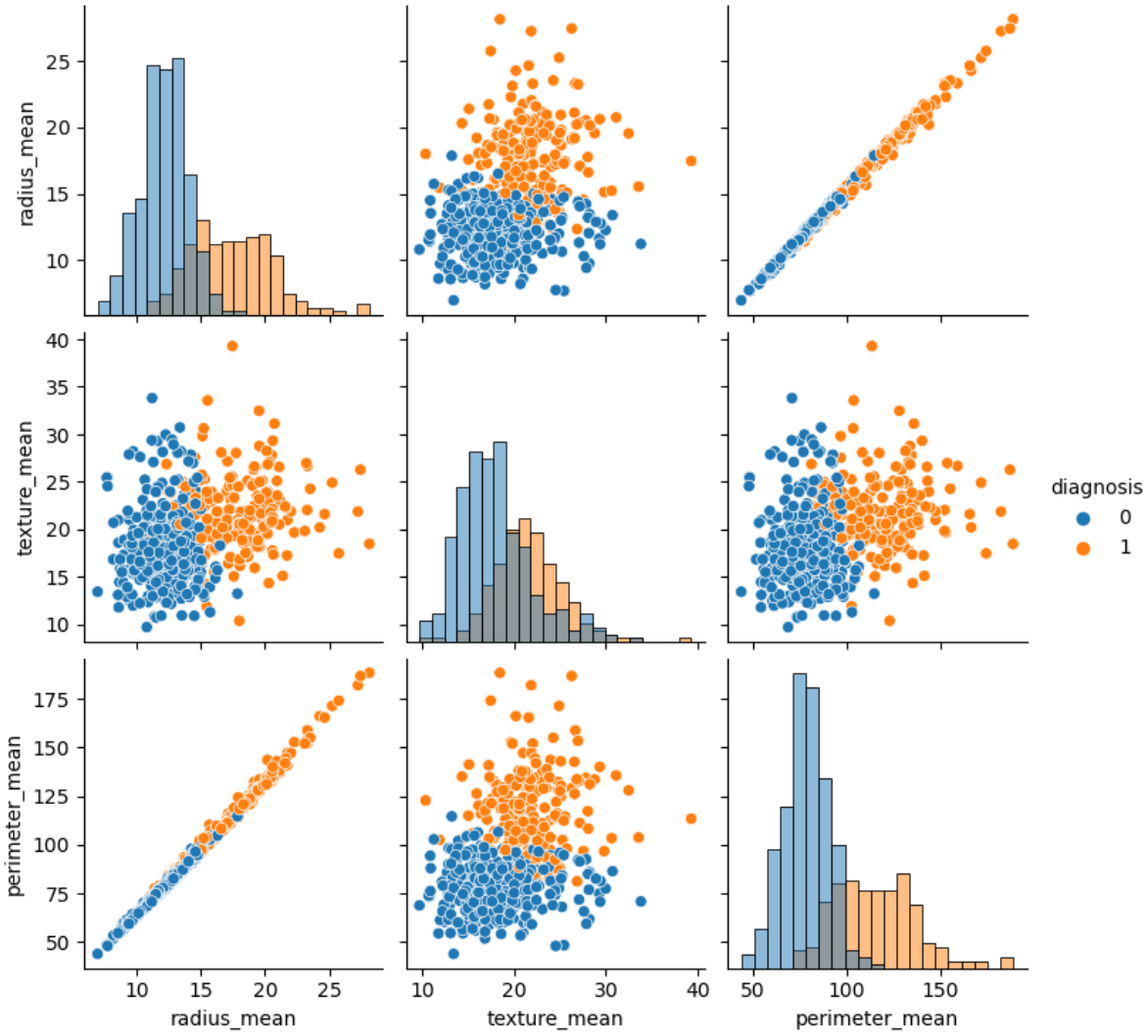
In [21]:
```
sns.pairplot(df.iloc[:, 1:5], hue="diagnosis")
```

Out[21]:
```
<seaborn.axisgrid.PairGrid at 0x19e2502b210>
```



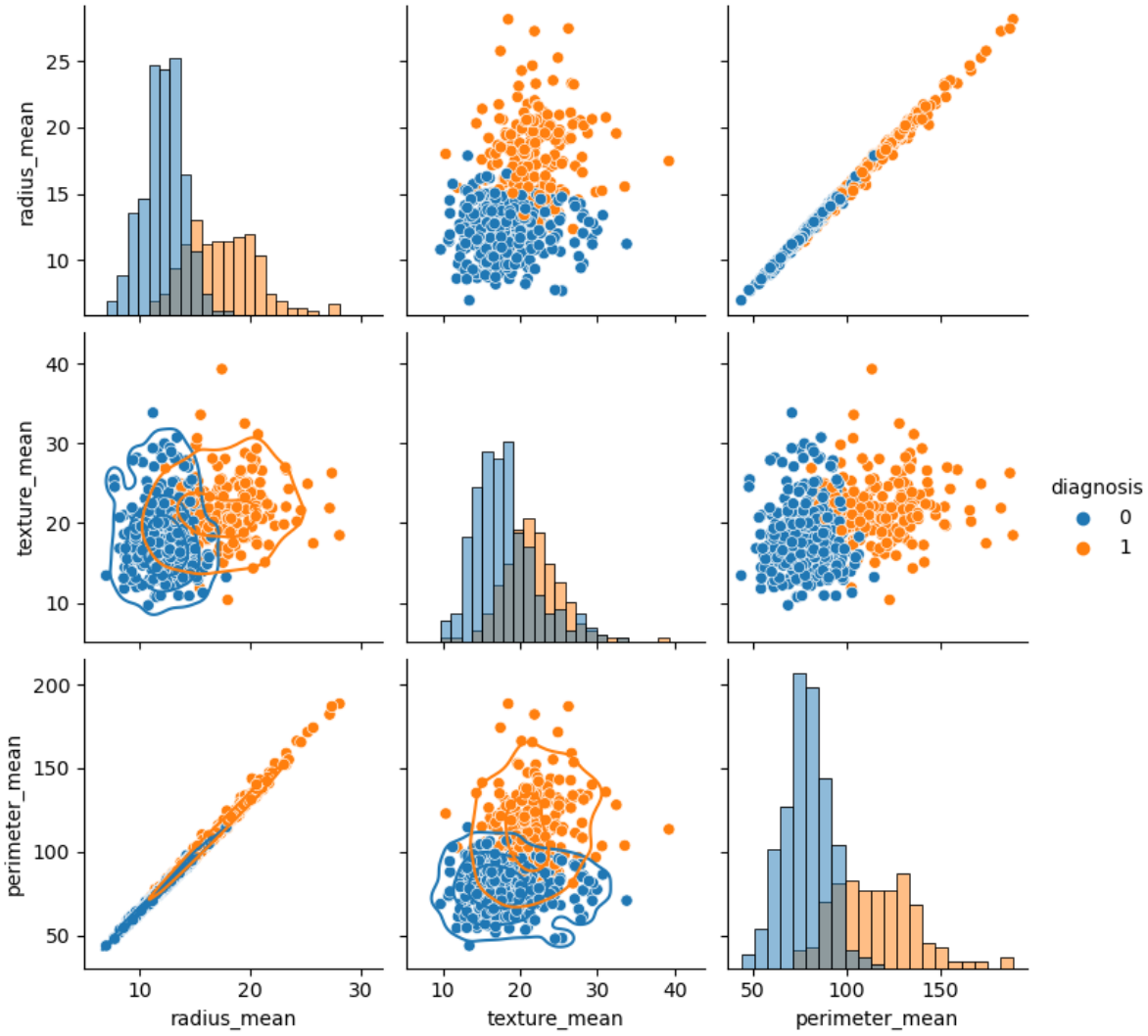
In [22]:
```
sns.pairplot(df.iloc[:, 1:5], hue="diagnosis",diag_kind="hist")
```

Out[22]:
```
<seaborn.axisgrid.PairGrid at 0x19e25213dd0>
```



In [23]:
```
g=sns.pairplot(df.iloc[:, 1:5], hue="diagnosis",diag_kind="hist")
g.map_lower(sns.kdeplot, levels=4, color=".2")
```

Out[23]:
```
<seaborn.axisgrid.PairGrid at 0x19e2768bdd0>
```



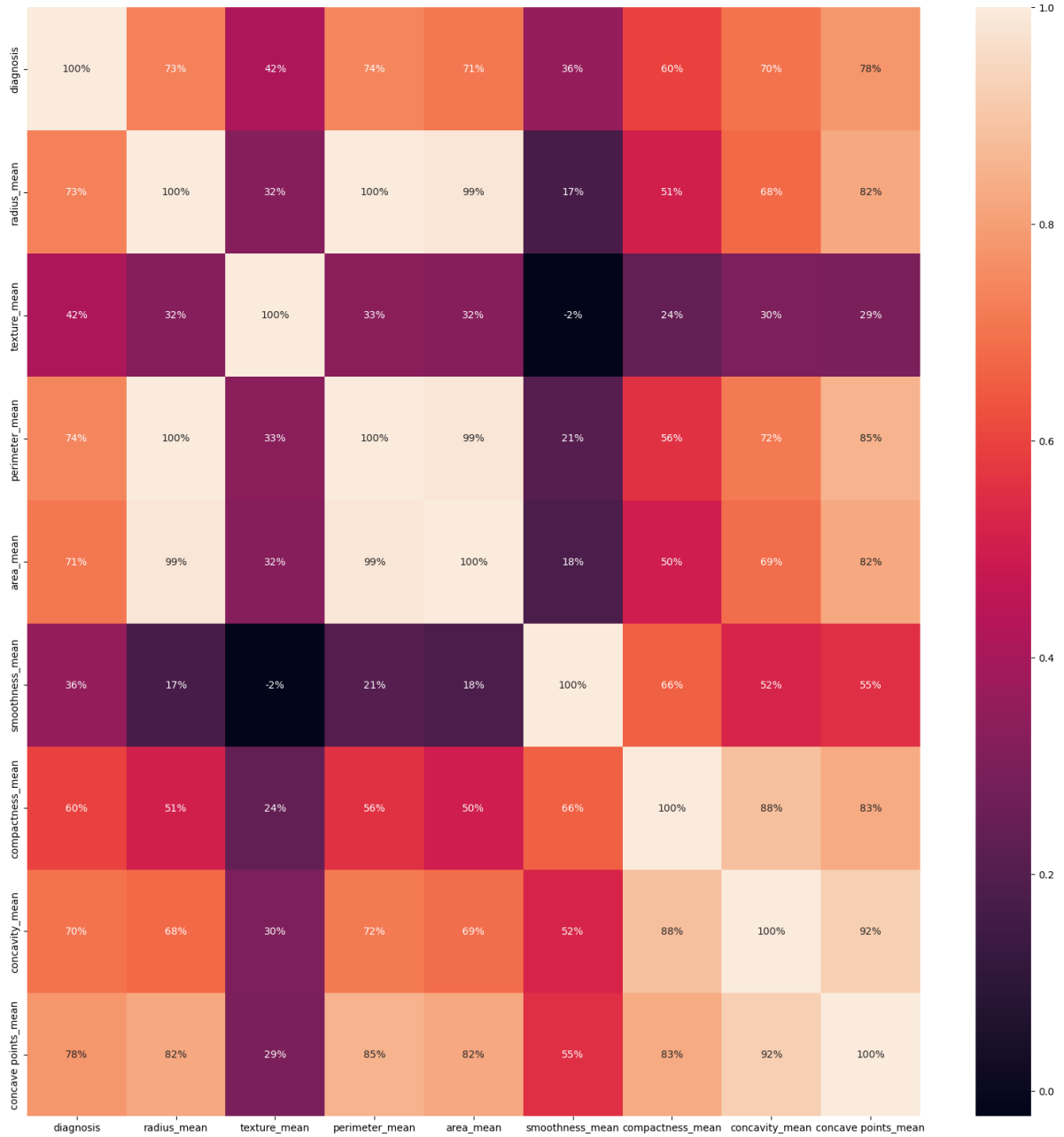
In [25]:
```
df.iloc[:,1:10,].corr()
```

Out[25]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothne
diagnosis	1.000000	0.730029	0.415185	0.742636	0.708984	
radius_mean	0.730029	1.000000	0.323782	0.997855	0.987357	
texture_mean	0.415185	0.323782	1.000000	0.329533	0.321086	-(
perimeter_mean	0.742636	0.997855	0.329533	1.000000	0.986507	
area_mean	0.708984	0.987357	0.321086	0.986507	1.000000	
smoothness_mean	0.358560	0.170581	-0.023389	0.207278	0.177028	
compactness_mean	0.596534	0.506124	0.236702	0.556936	0.498502	
concavity_mean	0.696360	0.676764	0.302418	0.716136	0.685983	
concave points_mean	0.776614	0.822529	0.293464	0.850977	0.823269	

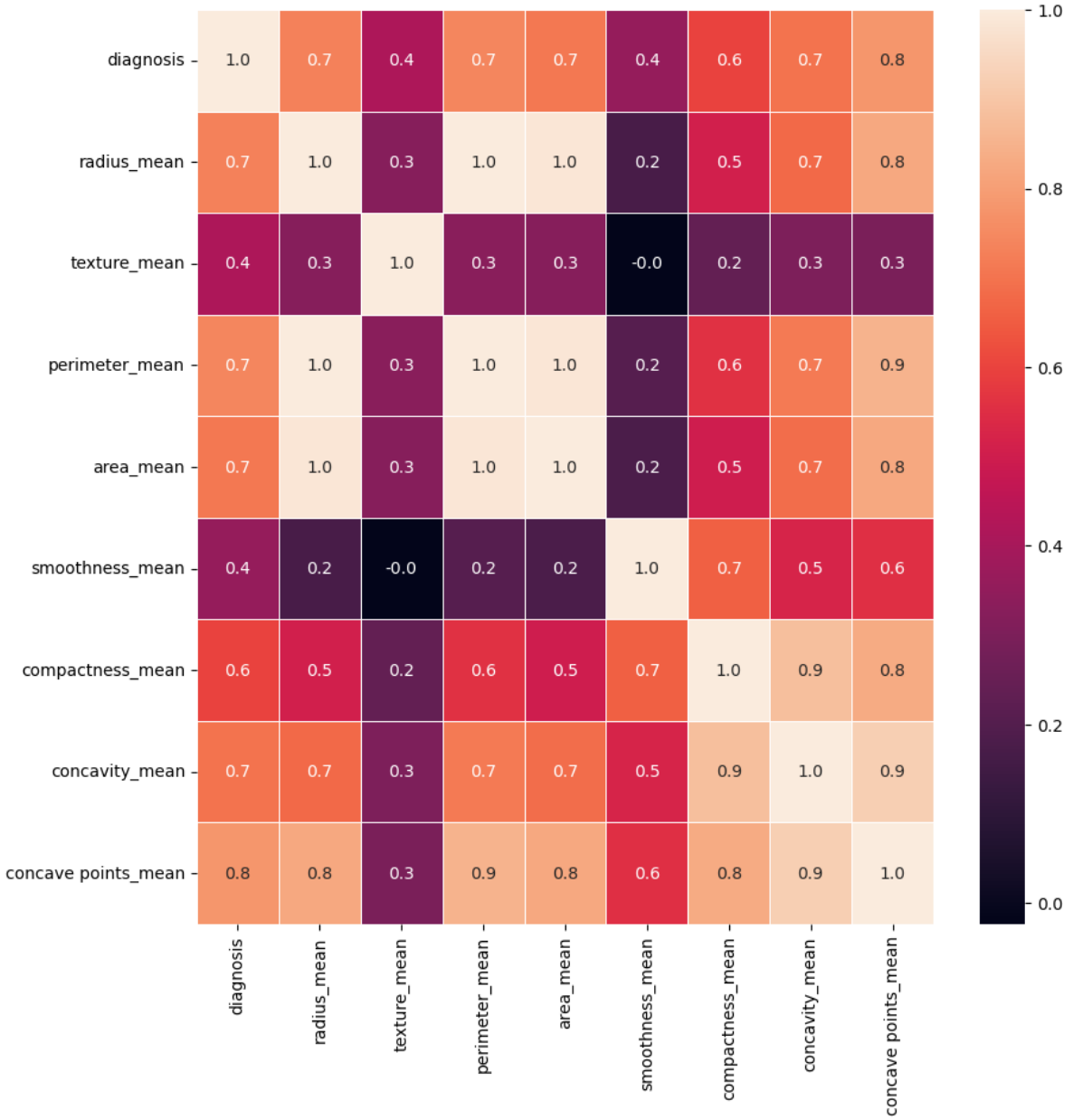
In [26]:
```
plt.figure(figsize=(20,20))
sns.heatmap(df.iloc[:,1:10,].corr(),annot=True,fmt=".0%")
```

Out[26]:
```
<Axes: >
```



In [30]:
```
plt.figure(figsize=(10,10))
sns.heatmap(df.iloc[:,1:10,].corr(),annot=True,fmt=".1f",linewidth=.5)
```

Out[30]:
```
<Axes: >
```



In [31]:
```
# split the dataset into dependent(X) and Independent(Y) datasets
X=df.iloc[:,2:31].values
Y=df.iloc[:,1].values
```

In [32]:
```
# spliting the data into trainning and test dateset
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.20,random_state=0)
```

In [33]:
```
# feature scaling
from sklearn.preprocessing import StandardScaler
X_train=StandardScaler().fit_transform(X_train)
X_test=StandardScaler().fit_transform(X_test)
```

In [34]:
```
# models/ Algorithms

def models(X_train,Y_train):
        #Logistic regression
        from sklearn.linear_model import LogisticRegression
        log=LogisticRegression(random_state=0)
        log.fit(X_train,Y_train)


        #Decision Tree
```

```
        from sklearn.tree import DecisionTreeClassifier
        tree=DecisionTreeClassifier(random_state=0,criterion="entropy")
        tree.fit(X_train,Y_train)

        #Random Forest
        from sklearn.ensemble import RandomForestClassifier
        forest=RandomForestClassifier(random_state=0,criterion="entropy",n_estimato
        forest.fit(X_train,Y_train)

        print('[0]logistic regression accuracy:',log.score(X_train,Y_train))
        print('[1]Decision tree accuracy:',tree.score(X_train,Y_train))
        print('[2]Random forest accuracy:',forest.score(X_train,Y_train))

        return log,tree,forest
```

In [35]:
```
model=models(X_train,Y_train)
```

```
[0]logistic regression accuracy: 0.9912087912087912
[1]Decision tree accuracy: 1.0
[2]Random forest accuracy: 0.9978021978021978
```

In [36]:
```
# testing the models/result

from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

for i in range(len(model)):
    print("Model",i)
    print(classification_report(Y_test,model[i].predict(X_test)))
    print('Accuracy : ',accuracy_score(Y_test,model[i].predict(X_test)))
```

```
Model 0
              precision    recall  f1-score   support

           0       0.96      0.99      0.97        67
           1       0.98      0.94      0.96        47

    accuracy                           0.96       114
   macro avg       0.97      0.96      0.96       114
weighted avg       0.97      0.96      0.96       114

Accuracy :  0.9649122807017544
Model 1
              precision    recall  f1-score   support

           0       0.94      0.96      0.95        67
           1       0.93      0.91      0.92        47

    accuracy                           0.94       114
   macro avg       0.94      0.94      0.94       114
weighted avg       0.94      0.94      0.94       114

Accuracy :  0.9385964912280702
Model 2
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        67
           1       1.00      0.94      0.97        47

    accuracy                           0.97       114
   macro avg       0.98      0.97      0.97       114
weighted avg       0.97      0.97      0.97       114

Accuracy :  0.9736842105263158
```

In [37]:
```
# prediction of random-forest
pred=model[2].predict(X_test)
print('Predicted values:')
print(pred)
print('Actual values:')
print(Y_test)
```

```
Predicted values:
[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 0 0 1 0 0 1 0 1 0 1 0 1 0 1 0
 1 0 1 0 0 0 0 0 1 0 0 0 1 1 1 1 0 0 0 0 0 0 1 1 1 0 0 1 0 1 1 1 0 0 1 0 0
 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]
Actual values:
[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 0 0 1 0 0 1 0 1 0 1 0 1 0 1 0
 1 0 1 1 0 1 0 0 1 0 0 0 1 1 1 1 0 0 0 0 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]
```

In [38]:
```
from joblib import dump
dump(model[2],"Cancer_prediction.joblib")
```

Out[38]:
```
['Cancer_prediction.joblib']
```

In [ ]:
```

```