```
Breast Cancer Classification
```

import libraries

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

load the data

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

	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 rows × 33 columns

Mapping Diagnosis variable which is our Target variable to 0,1 : 1 for Malignant 0: Benign

```
df_cancer.loc[:,'diagnosis'] = df_cancer.diagnosis.map({'M':1, 'B':0})
```

```
<ipython-input-22-1f716a526d65>:1: DeprecationWarning: In a future version, `df.iloc[:, i] = newvals` will attempt to set the value
  df_cancer.loc[:,'diagnosis'] = df_cancer.diagnosis.map({'M':1, 'B':0})
```

```
df_cancer.head()
```

	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 rows × 33 columns

Just looking at the tail or last 5 entries to know the distribution of data

```
df_cancer.tail()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	poi
564	926424	1	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	
565	926682	1	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	
566	926954	1	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	
567	927241	1	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	
568	92751	0	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	

5 rows × 33 columns

```
df_cancer.shape
```

```
(569, 33)
```

Checking if there are any nulls in any column

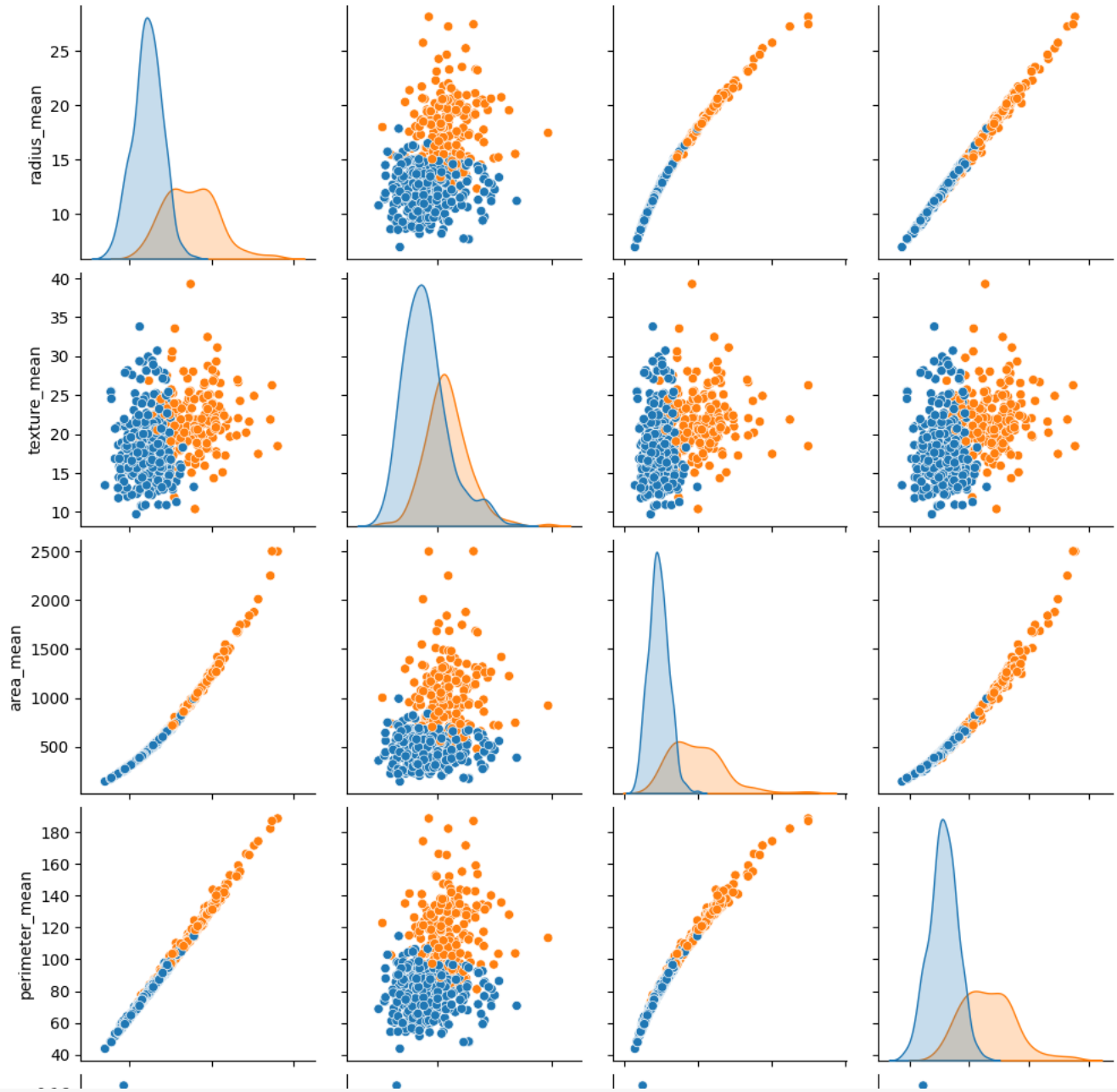
```
df_cancer.isnull().sum()
```

```
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
Unnamed: 32                569
dtype: int64
```

As we have seen there are no null entries except Unnamed:32 so we will delete it later before training. Looking at the Visualization of important features in relation to target variable diagnosis to see on which features it is more related

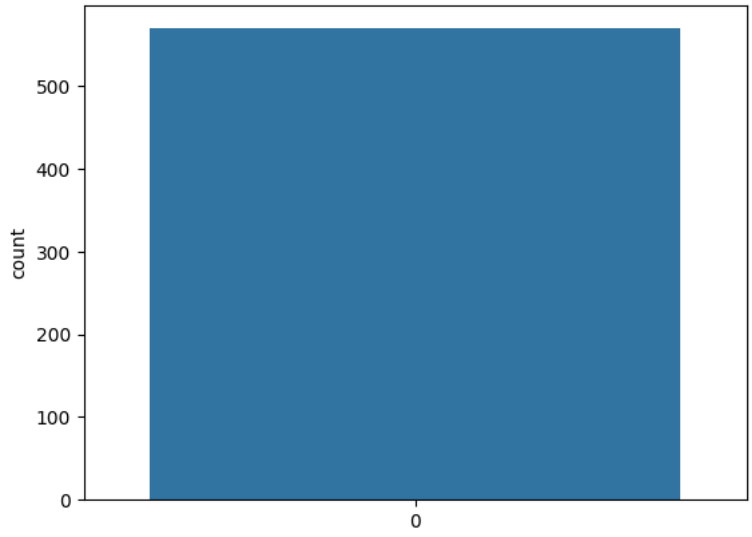
```
sns.pairplot(df_cancer, hue = 'diagnosis', vars = ['radius_mean', 'texture_mean', 'area_mean', 'perimeter_mean', 'smoothness_mean'] )
```

```
<seaborn.axisgrid.PairGrid at 0x7f0ba90bfbb0>
```



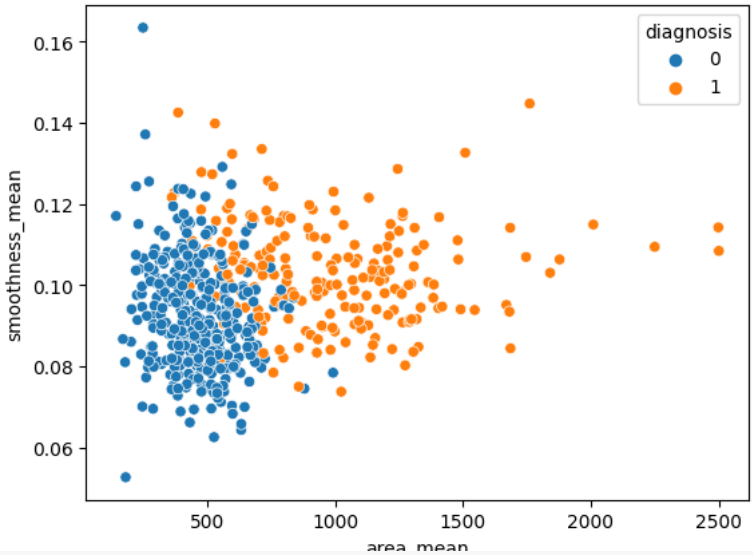

```
sns.countplot(df_cancer['diagnosis'], label = "Count")
```

```
<Axes: ylabel='count'>
```



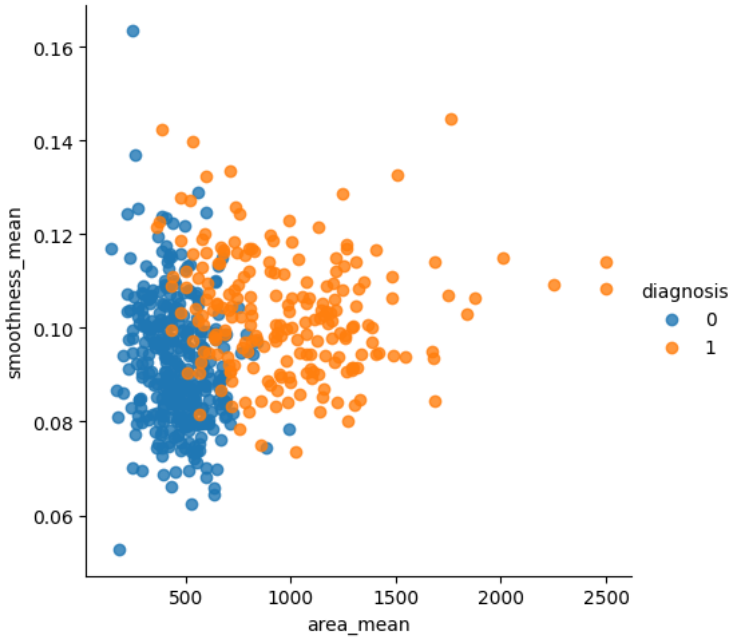

```
sns.scatterplot(x = 'area_mean', y = 'smoothness_mean', hue = 'diagnosis', data = df_cancer)
```

```
<Axes: xlabel='area_mean', ylabel='smoothness_mean'>
```



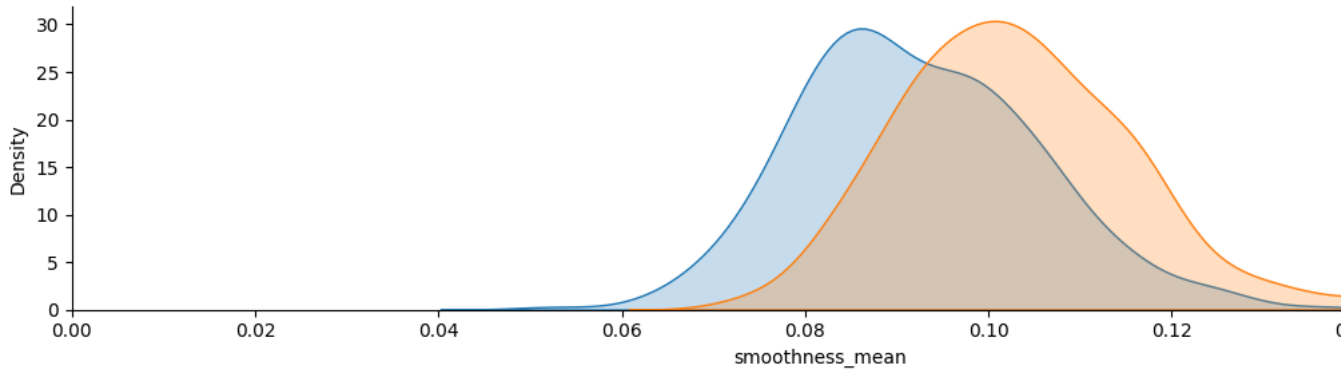

```
sns.lmplot(x=('area_mean'),y=('smoothness_mean'), hue ='diagnosis', data = df_cancer , fit_reg=False)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f0ba36c9b20>
```



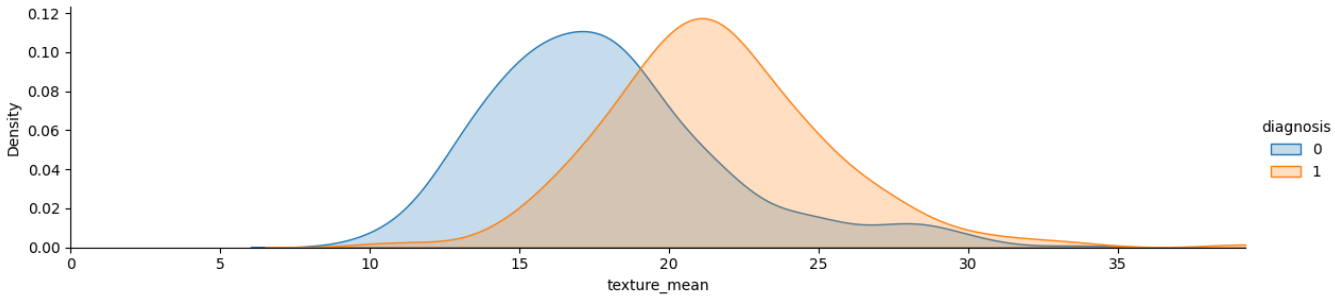

```
fig = sns.FacetGrid(df_cancer, hue="diagnosis",aspect=4)

# Next use map to plot all the possible kdeplots for the 'Age' column by the hue choice
fig.map(sns.kdeplot,'smoothness_mean',shade= True)

# Set the x max limit by the oldest passenger
oldest = df_cancer['smoothness_mean'].max()

#Since we know no one can be negative years old set the x lower limit at 0
fig.set(xlim=(0,oldest))

#Finally add a legend
fig.add_legend()
```

```
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:848: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  func(*plot_args, **plot_kwargs)
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:848: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  func(*plot_args, **plot_kwargs)
<seaborn.axisgrid.FacetGrid at 0x7f0ba363d7c0>
```




```
fig = sns.FacetGrid(df_cancer, hue="diagnosis",aspect=4)

# Next use map to plot all the possible kdeplots for the 'Age' column by the hue choice
fig.map(sns.kdeplot,'texture_mean',shade= True)

# Set the x max limit by the oldest passenger
oldest = df_cancer['texture_mean'].max()

#Since we know no one can be negative years old set the x lower limit at 0
fig.set(xlim=(0,oldest))

#Finally add a legend
fig.add_legend()
```

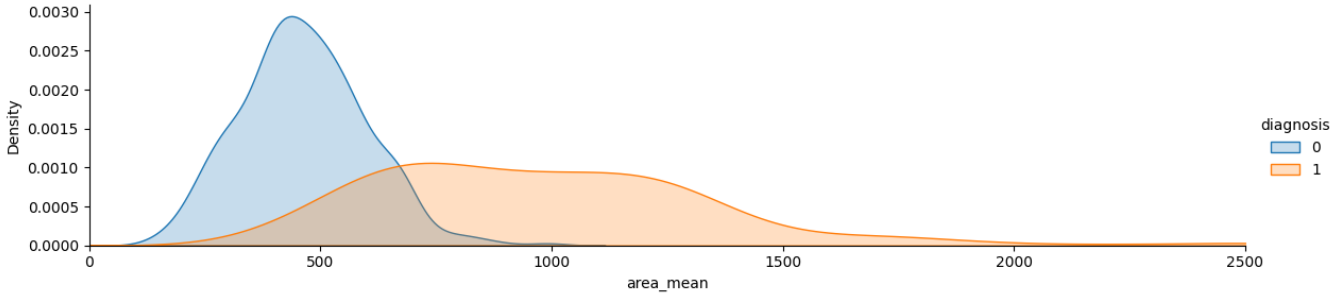
```
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:848: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  func(*plot_args, **plot_kwargs)
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:848: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  func(*plot_args, **plot_kwargs)
<seaborn.axisgrid.FacetGrid at 0x7f0ba358a9a0>
```




```
fig = sns.FacetGrid(df_cancer, hue="diagnosis",aspect=4)

# Next use map to plot all the possible kdeplots for the 'Age' column by the hue choice
fig.map(sns.kdeplot,'area_mean',shade= True)

# Set the x max limit by the oldest passenger
oldest = df_cancer['area_mean'].max()
```

```
#Since we know no one can be negative years old set the x lower limit at 0
fig.set(xlim=(0,oldest))

#Finally add a legend
fig.add_legend()
```

```
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:848: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  func(*plot_args, **plot_kwargs)
/usr/local/lib/python3.9/dist-packages/seaborn/axisgrid.py:848: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  func(*plot_args, **plot_kwargs)
<seaborn.axisgrid.FacetGrid at 0x7f0ba356c790>
```

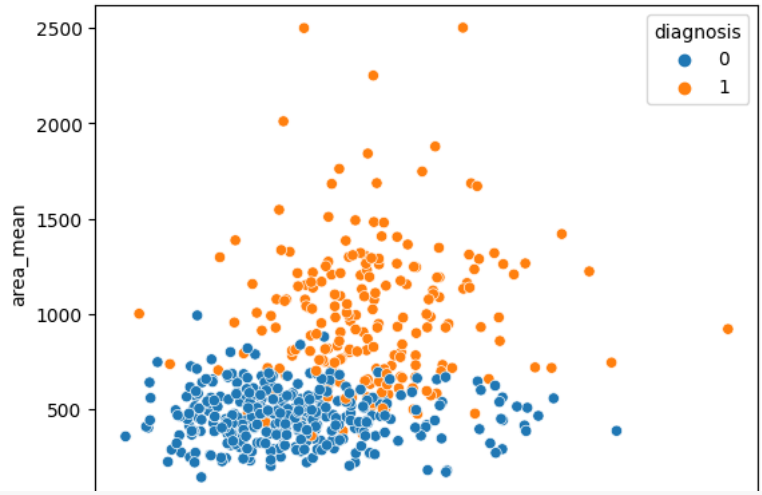



```
pip install -U seaborn
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: seaborn in /usr/local/lib/python3.9/dist-packages (0.12.2)
Requirement already satisfied: matplotlib!=3.6.1,>=3.1 in /usr/local/lib/python3.9/dist-packages (from seaborn) (3.7.1)
Requirement already satisfied: pandas>=0.25 in /usr/local/lib/python3.9/dist-packages (from seaborn) (1.5.3)
Requirement already satisfied: numpy!=1.24.0,>=1.17 in /usr/local/lib/python3.9/dist-packages (from seaborn) (1.22.4)
Requirement already satisfied: importlib-resources>=3.2.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib!=3.6.1,>=3.1->
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (8.4
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.9/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (0.11
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.9/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.9/dist-packages (from matplotlib!=3.6.1,>=3.1->seabor
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.9/dist-packages (from matplotlib!=3.6.1,>=3.1->seaborn) (2
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.9/dist-packages (from pandas>=0.25->seaborn) (2022.7.1)
Requirement already satisfied: zipp>=3.1.0 in /usr/local/lib/python3.9/dist-packages (from importlib-resources>=3.2.0->matplotlib!=
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.9/dist-packages (from python-dateutil>=2.7->matplotlib!=3.6.1,>=3
```

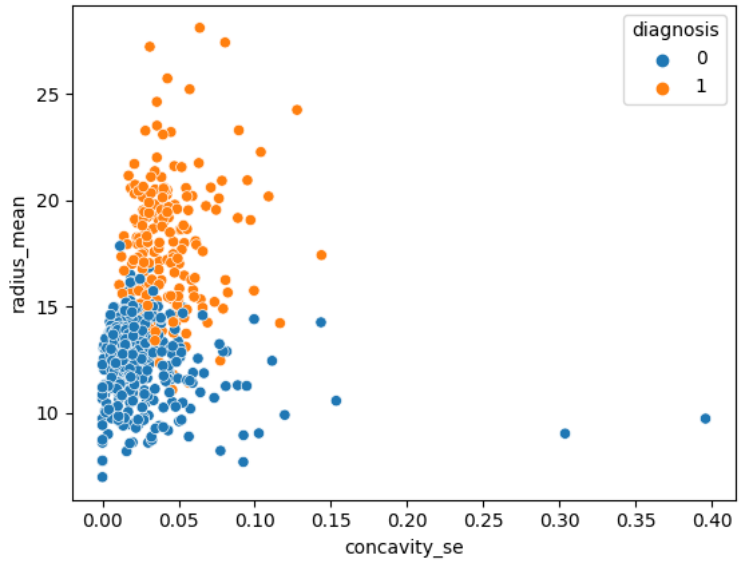
```
sns.scatterplot(x=('texture_mean'),y=('area_mean'),hue='diagnosis',data=df_cancer)
```

```
<Axes: xlabel='texture_mean', ylabel='area_mean'>
```



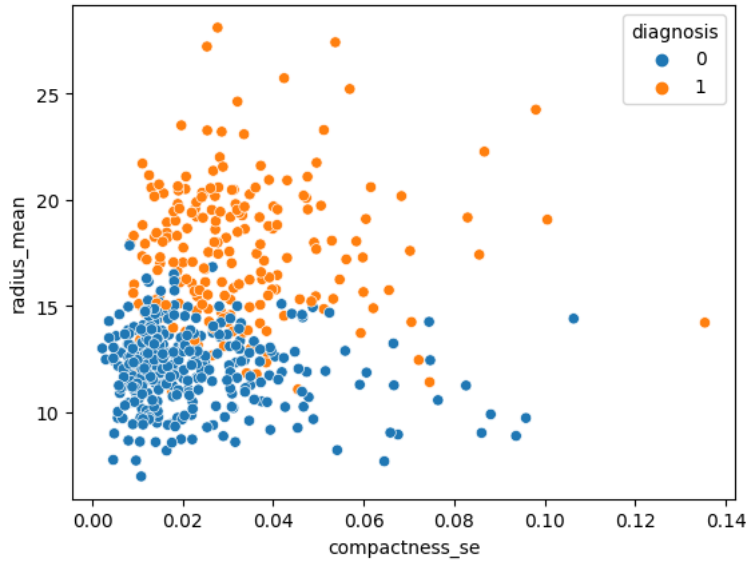

```
sns.scatterplot(x=('concavity_se'),y=( 'radius_mean'), hue ='diagnosis', data = df_cancer)
```

```
<Axes: xlabel='concavity_se', ylabel='radius_mean'>
```




```
sns.scatterplot(x=('compactness_se'), y=('radius_mean'), hue ='diagnosis', data = df_cancer)
```

```
<Axes: xlabel='compactness_se', ylabel='radius_mean'>
```

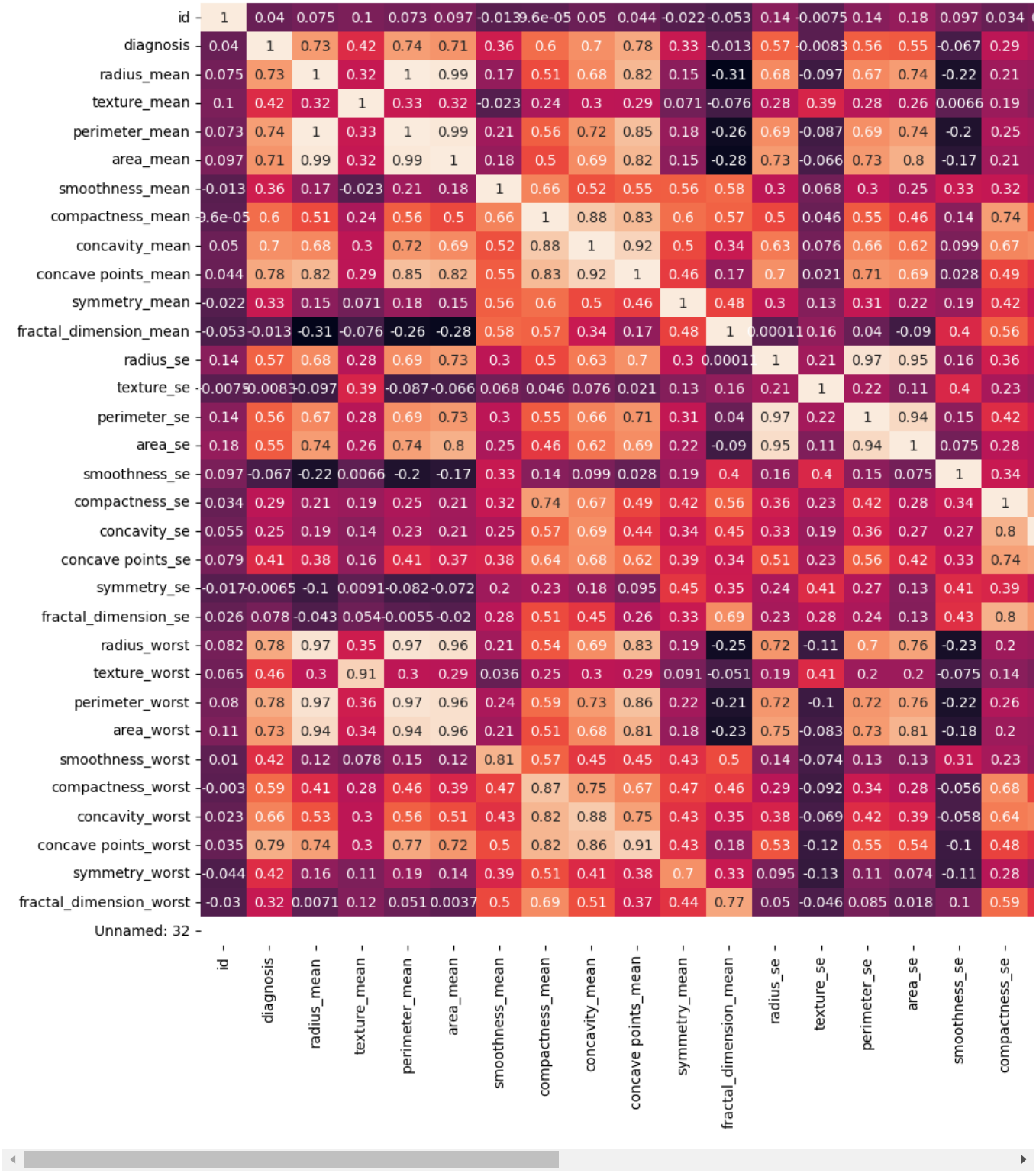


Spearman rank order correlation Pearson's Rank Correlation.

Double-click (or enter) to edit

```
plt.figure(figsize=(24,12))
sns.heatmap(df_cancer.corr(), annot=True)
```

```
<Axes: >
```




```
unwantedcolumnlist=["diagnosis","Unnamed: 32","id"]
```

```
X = df_cancer.drop(unwantedcolumnlist,axis=1)
```

```
y = df_cancer['diagnosis']
```

Now, we will split training and testing dataset using sklearn to X_train, X_test,y_train,y_test

```
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=5)
```

```
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix

svc_model = SVC()
svc_model.fit(X_train, y_train)
```

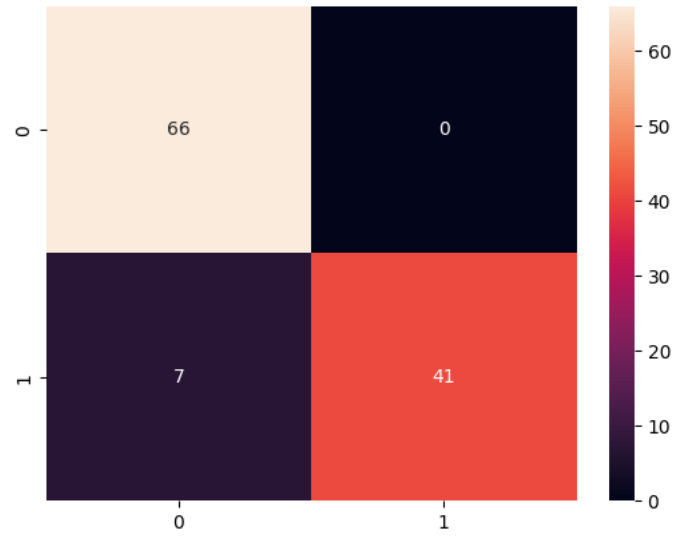
```
SVC()
```

```
y_predict = svc_model.predict(X_test)
cm = confusion_matrix(y_test, y_predict)
cm
```

```
array([[66,  0],
       [ 7, 41]])
```

```
sns.heatmap(cm, annot=True)
```

```
<Axes: >
```




```
print(classification_report(y_test, y_predict))
```

```
              precision    recall  f1-score   support

           0       0.90      1.00      0.95        66
           1       1.00      0.85      0.92        48

    accuracy                           0.94       114
   macro avg       0.95      0.93      0.94       114
weighted avg       0.94      0.94      0.94       114
```

```
min_train = X_train.min()
min_train
```

```
radius_mean                 6.981000
texture_mean                9.710000
perimeter_mean             43.790000
area_mean                 143.500000
smoothness_mean             0.052630
compactness_mean            0.019380
concavity_mean              0.000000
concave points_mean         0.000000
symmetry_mean               0.106000
fractal_dimension_mean      0.049960
radius_se                   0.111500
texture_se                  0.362100
```

```
perimeter_se                0.757000
area_se                     6.802000
smoothness_se               0.001713
compactness_se              0.002252
concavity_se                0.000000
concave points_se           0.000000
symmetry_se                 0.007882
fractal_dimension_se        0.000950
radius_worst                7.930000
texture_worst              12.020000
perimeter_worst            50.410000
area_worst                185.200000
smoothness_worst            0.071170
compactness_worst           0.027290
concavity_worst             0.000000
concave points_worst        0.000000
symmetry_worst              0.156500
fractal_dimension_worst     0.055040
dtype: float64
```

```
range_train = (X_train - min_train).max()
range_train
```

```
radius_mean                21.129000
texture_mean               29.570000
perimeter_mean            144.710000
area_mean                2355.500000
smoothness_mean             0.110770
compactness_mean            0.326020
concavity_mean              0.426800
concave points_mean         0.201200
symmetry_mean               0.198000
fractal_dimension_mean      0.045790
radius_se                   2.761500
texture_se                  4.522900
perimeter_se               21.223000
area_se                   518.798000
smoothness_se               0.029417
compactness_se              0.133148
concavity_se                0.396000
concave points_se           0.052790
symmetry_se                 0.071068
fractal_dimension_se        0.028890
radius_worst               25.190000
texture_worst              37.520000
perimeter_worst           170.390000
area_worst               3246.800000
smoothness_worst            0.129430
compactness_worst           1.030710
concavity_worst             1.105000
concave points_worst        0.291000
symmetry_worst              0.420900
fractal_dimension_worst     0.152460
dtype: float64
```

```
X_train_scaled = (X_train - min_train)/range_train
```

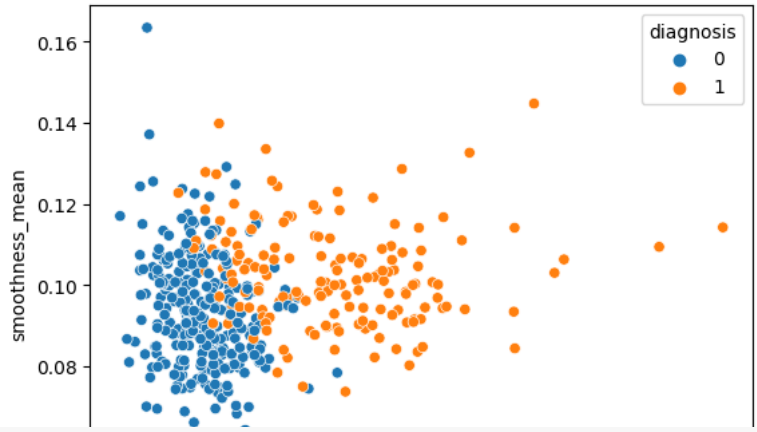
```
X_train_scaled.head()
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean
306	0.294335	0.206628	0.278350	0.167183	0.293220	0.101620
410	0.207251	0.265810	0.198328	0.108809	0.324546	0.103521
197	0.525297	0.410213	0.508673	0.373806	0.190304	0.205632
376	0.169861	0.355428	0.182157	0.082700	0.343956	0.449727
244	0.587770	0.466351	0.589524	0.429421	0.452018	0.418441

5 rows × 30 columns

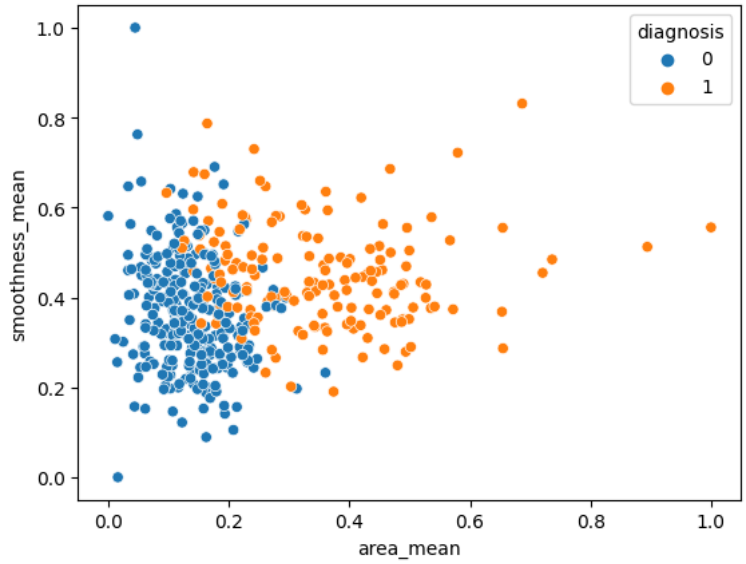
```
sns.scatterplot(x = X_train['area_mean'], y = X_train['smoothness_mean'], hue = y_train)
```

```
<Axes: xlabel='area_mean', ylabel='smoothness_mean'>
```




```
sns.scatterplot(x = X_train_scaled['area_mean'], y = X_train_scaled['smoothness_mean'], hue = y_train)
```

```
<Axes: xlabel='area_mean', ylabel='smoothness_mean'>
```



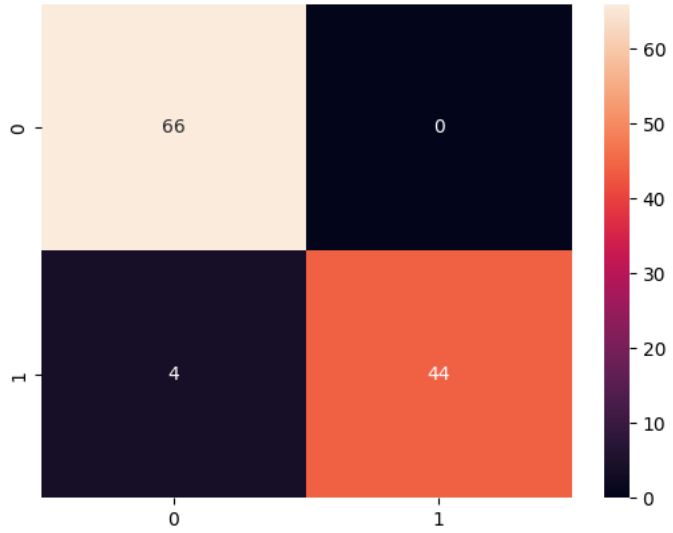

```
min_test = X_test.min()
range_test = (X_test - min_test).max()
X_test_scaled = (X_test - min_test)/range_test
```

```
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix

svc_model = SVC()
svc_model.fit(X_train_scaled, y_train)
```

▾ SVC
SVC()

```
y_predict = svc_model.predict(X_test_scaled)
cm = confusion_matrix(y_test, y_predict)

sns.heatmap(cm,annot=True,fmt="d")
```

```
<Axes: >
```




```
print(classification_report(y_test,y_predict))
```

```
                  precision    recall  f1-score   support

               0       0.94      1.00      0.97        66
               1       1.00      0.92      0.96        48

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

```
param_grid = {'C': [0.1, 1, 10, 100], 'gamma': [1, 0.1, 0.01, 0.001], 'kernel': ['rbf']}
```

```
from sklearn.model_selection import GridSearchCV
```

```
grid = GridSearchCV(SVC(),param_grid,refit=True,verbose=4)
```

```
grid.fit(X_train_scaled,y_train)
```

```
Fitting 5 folds for each of 16 candidates, totalling 80 fits
[CV 1/5] END ........C=0.1, gamma=1, kernel=rbf;, score=1.000 total time=   0.0s
[CV 2/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.945 total time=   0.0s
[CV 3/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.912 total time=   0.0s
[CV 4/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.956 total time=   0.0s
[CV 5/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.934 total time=   0.0s
[CV 1/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.945 total time=   0.0s
[CV 2/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.901 total time=   0.0s
[CV 3/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.890 total time=   0.0s
[CV 4/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.923 total time=   0.0s
[CV 5/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.868 total time=   0.0s
[CV 1/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.648 total time=   0.0s
[CV 2/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.637 total time=   0.0s
[CV 3/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.637 total time=   0.0s
[CV 4/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.637 total time=   0.0s
[CV 5/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.637 total time=   0.0s
[CV 1/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.648 total time=   0.0s
[CV 2/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.637 total time=   0.0s
[CV 3/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.637 total time=   0.0s
[CV 4/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.637 total time=   0.0s
[CV 5/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.637 total time=   0.0s
[CV 1/5] END ..........C=1, gamma=1, kernel=rbf;, score=1.000 total time=   0.0s
[CV 2/5] END ..........C=1, gamma=1, kernel=rbf;, score=0.956 total time=   0.0s
[CV 3/5] END ..........C=1, gamma=1, kernel=rbf;, score=0.967 total time=   0.0s
[CV 4/5] END ..........C=1, gamma=1, kernel=rbf;, score=1.000 total time=   0.0s
[CV 5/5] END ..........C=1, gamma=1, kernel=rbf;, score=0.967 total time=   0.0s
[CV 1/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.989 total time=   0.0s
[CV 2/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.945 total time=   0.0s
[CV 3/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.923 total time=   0.0s
[CV 4/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.967 total time=   0.0s
[CV 5/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.934 total time=   0.0s
[CV 1/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.945 total time=   0.0s
[CV 2/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.901 total time=   0.0s
[CV 3/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.879 total time=   0.0s
[CV 4/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.923 total time=   0.0s
[CV 5/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.868 total time=   0.0s
[CV 1/5] END ......C=1, gamma=0.001, kernel=rbf;, score=0.648 total time=   0.0s
[CV 2/5] END ......C=1, gamma=0.001, kernel=rbf;, score=0.637 total time=   0.0s
[CV 3/5] END ......C=1, gamma=0.001, kernel=rbf;, score=0.637 total time=   0.0s
[CV 4/5] END ......C=1, gamma=0.001, kernel=rbf;, score=0.637 total time=   0.0s
[CV 5/5] END ......C=1, gamma=0.001, kernel=rbf;, score=0.637 total time=   0.0s
[CV 1/5] END .........C=10, gamma=1, kernel=rbf;, score=1.000 total time=   0.0s
[CV 2/5] END .........C=10, gamma=1, kernel=rbf;, score=0.967 total time=   0.0s
[CV 3/5] END .........C=10, gamma=1, kernel=rbf;, score=0.956 total time=   0.0s
[CV 4/5] END .........C=10, gamma=1, kernel=rbf;, score=1.000 total time=   0.0s
[CV 5/5] END .........C=10, gamma=1, kernel=rbf;, score=0.956 total time=   0.0s
[CV 1/5] END .......C=10, gamma=0.1, kernel=rbf;, score=1.000 total time=   0.0s
[CV 2/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.967 total time=   0.0s
[CV 3/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.967 total time=   0.0s
[CV 4/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.989 total time=   0.0s
[CV 5/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.945 total time=   0.0s
[CV 1/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.989 total time=   0.0s
[CV 2/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.945 total time=   0.0s
[CV 3/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.923 total time=   0.0s
[CV 4/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.967 total time=   0.0s
```

```
grid.best_params_
```

```
{'C': 1, 'gamma': 1, 'kernel': 'rbf'}
```

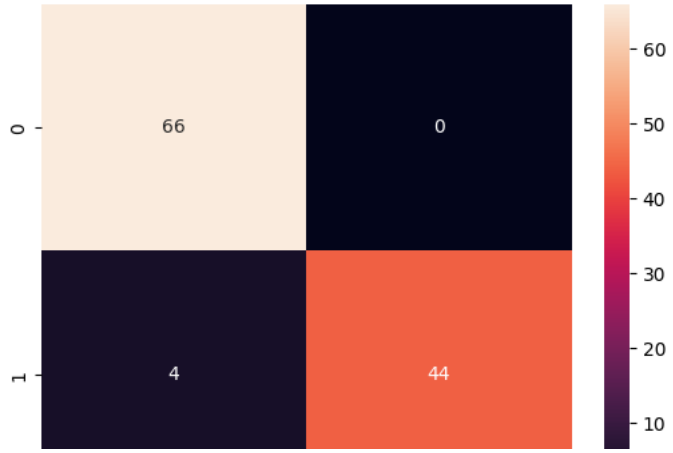
```
grid.best_estimator_
```

```
SVC(C=1, gamma=1)
```

```
grid_predictions = grid.predict(X_test_scaled)
```

```
cm = confusion_matrix(y_test, grid_predictions)
sns.heatmap(cm,annot=True,fmt="d")
```

```
<Axes: >
```



result

0 1

```
print(classification_report(y_test,grid_predictions))
```

```
              precision    recall  f1-score   support

           0       0.94      1.00      0.97        66
           1       1.00      0.92      0.96        48

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

✓ 0s completed at 4:40 PM