BREAST CANCER PREDICTION USING SVM (ENSEMBLING TECHNIQUES)

In [1]:
```python
#Import Required Packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#Supress Warnings
from  warnings import filterwarnings
filterwarnings('ignore')

#Read the Data
df=pd.read_csv('cancer.csv')
df.head()
```

Out[1]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	p
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	

5 rows × 33 columns

In [2]:
```python
#For making the data ready for analysis we have to :
#1.check the dimensions and data types of dataframe
#2.study the summary statistics
#3.check for missing values
#4.find the correlation
```

In [3]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 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
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

In [4]:
```python
# Here we can observe that 'id' cannot be used for classification.
#Also unnamed:32 feature includes NAN values, so we donot need it.
```

```
#Diagnosis is our class label. So we are removing the 'id'and 'unnamed:32'features from the dataset.
```

In [5]:
```
data=df.drop(['id', 'Unnamed: 32'],axis=1)
data.head()
```

Out[5]:

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

5 rows × 31 columns

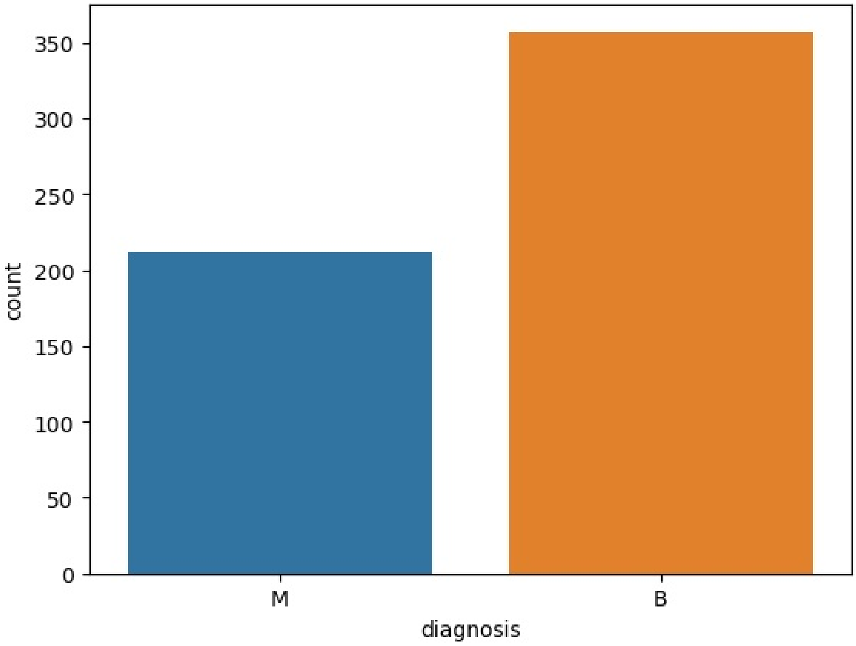
In [6]:
```
#Get the shape
print(data.shape)
```

```
(569, 31)
```

Here we can observe that the dataframe consists of 31 columns and 569 observations

In [7]:
```
#Checking the class distribution of the target variable
axis=sns.countplot(data['diagnosis'],label='Count Plot')
B,M=data['diagnosis'].value_counts()
print('Number of Benign:',B)
print('Number of Malignant:',M)
```

```
Number of Benign: 357
Number of Malignant: 212
```



Here the data is unbalanced

Statistical Summary

In [8]:
```
#We are going to check for the summary statistics of all the variables,
#for numeric variables we use describe() and for categorical variable we use describe(include='object')
```

In [9]:
```
# Dataframe with numerical features
data.describe().T
```

Out[9]:

	count	mean	std	min	25%	50%	75%	max
radius_mean	569.0	14.127292	3.524049	6.981000	11.700000	13.370000	15.780000	28.11000
texture_mean	569.0	19.289649	4.301036	9.710000	16.170000	18.840000	21.800000	39.28000
perimeter_mean	569.0	91.969033	24.298981	43.790000	75.170000	86.240000	104.100000	188.50000
area_mean	569.0	654.889104	351.914129	143.500000	420.300000	551.100000	782.700000	2501.00000
smoothness_mean	569.0	0.096360	0.014064	0.052630	0.086370	0.095870	0.105300	0.16340
compactness_mean	569.0	0.104341	0.052813	0.019380	0.064920	0.092630	0.130400	0.34540
concavity_mean	569.0	0.088799	0.079720	0.000000	0.029560	0.061540	0.130700	0.42680
concave points_mean	569.0	0.048919	0.038803	0.000000	0.020310	0.033500	0.074000	0.20120
symmetry_mean	569.0	0.181162	0.027414	0.106000	0.161900	0.179200	0.195700	0.30400
fractal_dimension_mean	569.0	0.062798	0.007060	0.049960	0.057700	0.061540	0.066120	0.09744
radius_se	569.0	0.405172	0.277313	0.111500	0.232400	0.324200	0.478900	2.87300
texture_se	569.0	1.216853	0.551648	0.360200	0.833900	1.108000	1.474000	4.88500
perimeter_se	569.0	2.866059	2.021855	0.757000	1.606000	2.287000	3.357000	21.98000
area_se	569.0	40.337079	45.491006	6.802000	17.850000	24.530000	45.190000	542.20000
smoothness_se	569.0	0.007041	0.003003	0.001713	0.005169	0.006380	0.008146	0.03113
compactness_se	569.0	0.025478	0.017908	0.002252	0.013080	0.020450	0.032450	0.13540
concavity_se	569.0	0.031894	0.030186	0.000000	0.015090	0.025890	0.042050	0.39600
concave points_se	569.0	0.011796	0.006170	0.000000	0.007638	0.010930	0.014710	0.05279
symmetry_se	569.0	0.020542	0.008266	0.007882	0.015160	0.018730	0.023480	0.07895
fractal_dimension_se	569.0	0.003795	0.002646	0.000895	0.002248	0.003187	0.004558	0.02984
radius_worst	569.0	16.269190	4.833242	7.930000	13.010000	14.970000	18.790000	36.04000
texture_worst	569.0	25.677223	6.146258	12.020000	21.080000	25.410000	29.720000	49.54000
perimeter_worst	569.0	107.261213	33.602542	50.410000	84.110000	97.660000	125.400000	251.20000
area_worst	569.0	880.583128	569.356993	185.200000	515.300000	686.500000	1084.000000	4254.00000
smoothness_worst	569.0	0.132369	0.022832	0.071170	0.116600	0.131300	0.146000	0.22260
compactness_worst	569.0	0.254265	0.157336	0.027290	0.147200	0.211900	0.339100	1.05800
concavity_worst	569.0	0.272188	0.208624	0.000000	0.114500	0.226700	0.382900	1.25200
concave points_worst	569.0	0.114606	0.065732	0.000000	0.064930	0.099930	0.161400	0.29100
symmetry_worst	569.0	0.290076	0.061867	0.156500	0.250400	0.282200	0.317900	0.66380
fractal_dimension_worst	569.0	0.083946	0.018061	0.055040	0.071460	0.080040	0.092080	0.20750

The above data illustrates the summary statistics of all the numeric values like mean, median, standars deviation

minimum and maximum values

In [10]:

```
#Dataframe with categorical features
data.describe(include='object').T
```

Out[10]:

	count	unique	top	freq
diagnosis	569	2	B	357

The above data illustrates the summary statistics of categorical variables ie. diagnosis (no. of levels in the variable),

top(majority level) and the count of majority level

Label Encoding for Target Varible

In [11]:

```
#Converting target categorical variable into numeric variable
#Replace 'M' with zero
data['diagnosis']=data['diagnosis'].replace('M',0)
#Replace 'B' with one
```

```
data['diagnosis']=data['diagnosis'].replace('B',1)
data.head()
```

Out[11]:

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

5 rows × 31 columns

Treating Missing Values

If the missing values are not treated properly, we may end up drawing an inaccurate inference about the data

To get the missing values in each column, we use inbuilt function isnull().sum()

In [12]:

```
#To get the count of missing values
missing_values=data.isnull().sum()
#print the count of missing values
print(missing_values)
```

```
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
```

There are no missing values present in the data

Visualization

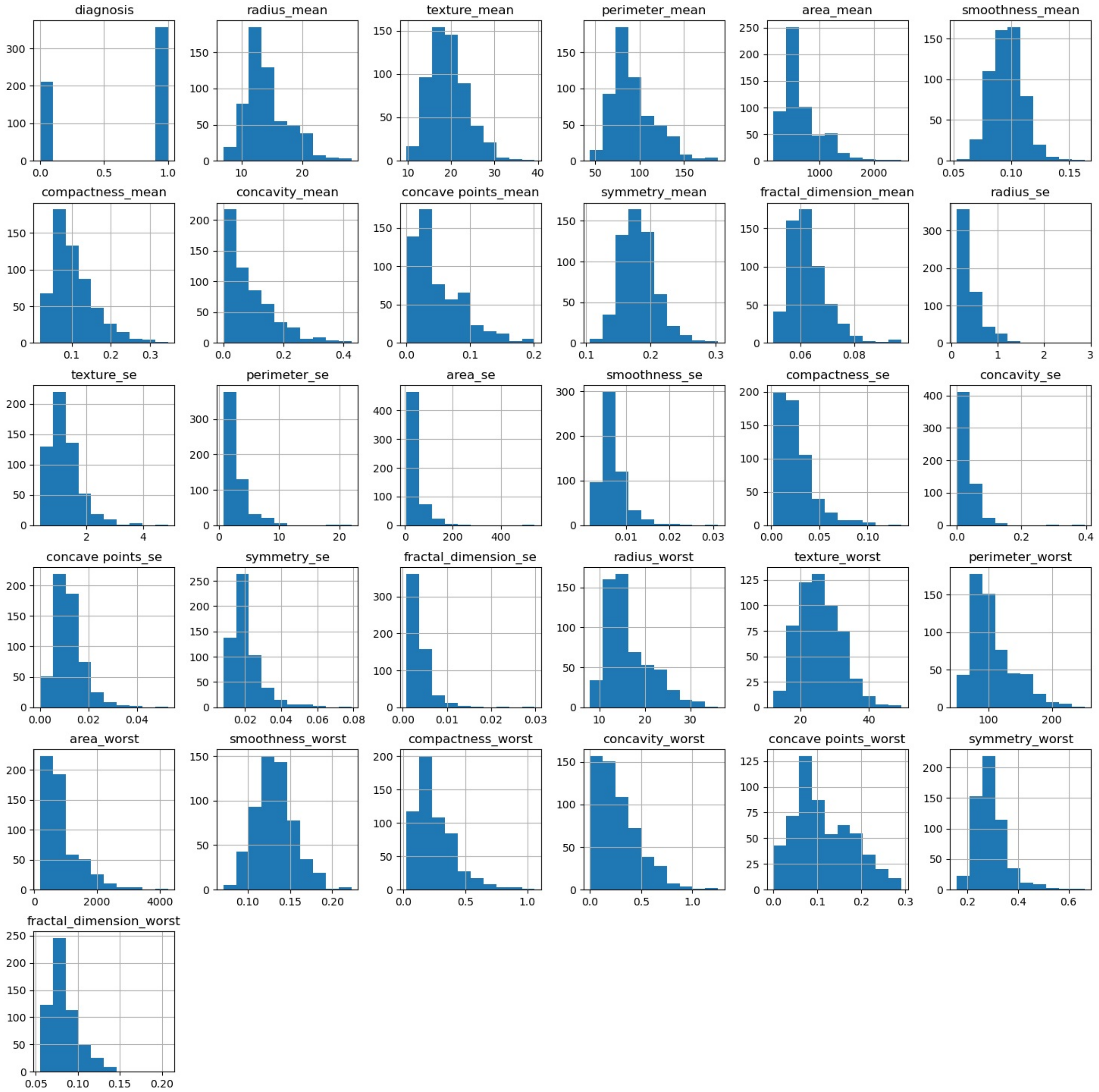
In [13]:

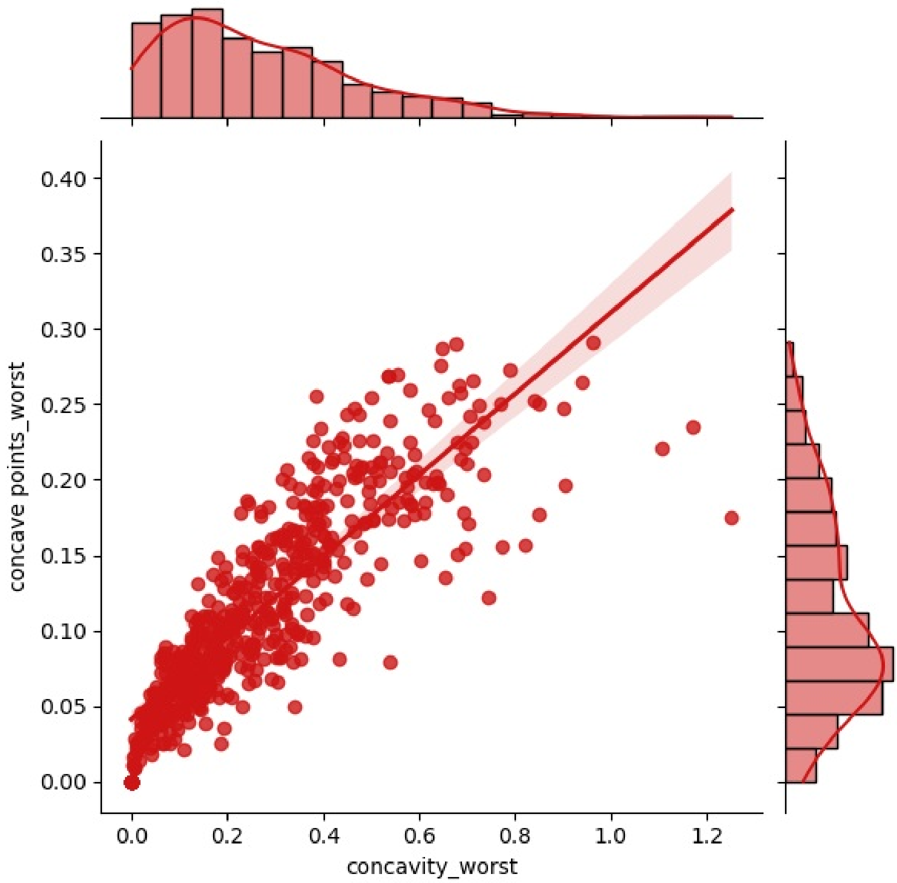
```
fig=data.hist(figsize=(18,18))
list(data.columns)
data.columns
x=data.drop(['diagnosis'],axis=1)
sns.jointplot(x.loc[:,'concavity_worst'],x.loc[:,'concave points_worst'],kind='reg',color='#ce1414')
x.head()
```

Out[13]:

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

5 rows × 30 columns





Call the correlation function which will return the correlation matrix of numeric variables

In [14]:

```
#Check correlation
data_num=data.drop('diagnosis',axis=1)
corr=data_num.corr()
corr
```

Out[14]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean
radius_mean	1.000000	0.323782	0.997855	0.987357	0.170581	0.506124	0.676764
texture_mean	0.323782	1.000000	0.329533	0.321086	-0.023389	0.236702	0.302418
perimeter_mean	0.997855	0.329533	1.000000	0.986507	0.207278	0.556936	0.716136
area_mean	0.987357	0.321086	0.986507	1.000000	0.177028	0.498502	0.685983
smoothness_mean	0.170581	-0.023389	0.207278	0.177028	1.000000	0.659123	0.521984
compactness_mean	0.506124	0.236702	0.556936	0.498502	0.659123	1.000000	0.883121
concavity_mean	0.676764	0.302418	0.716136	0.685983	0.521984	0.883121	1.000000
concave points_mean	0.822529	0.293464	0.850977	0.823269	0.553695	0.831135	0.921391
symmetry_mean	0.147741	0.071401	0.183027	0.151293	0.557775	0.602641	0.500667
fractal_dimension_mean	-0.311631	-0.076437	-0.261477	-0.283110	0.584792	0.565369	0.336783
radius_se	0.679090	0.275869	0.691765	0.732562	0.301467	0.497473	0.631925
texture_se	-0.097317	0.386358	-0.086761	-0.066280	0.068406	0.046205	0.076218
perimeter_se	0.674172	0.281673	0.693135	0.726628	0.296092	0.548905	0.660391
area_se	0.735864	0.259845	0.744983	0.800086	0.246552	0.455653	0.617427
smoothness_se	-0.222600	0.006614	-0.202694	-0.166777	0.332375	0.135299	0.098564
compactness_se	0.206000	0.191975	0.250744	0.212583	0.318943	0.738722	0.670279
concavity_se	0.194204	0.143293	0.228082	0.207660	0.248396	0.570517	0.691270
concave points_se	0.376169	0.163851	0.407217	0.372320	0.380676	0.642262	0.683260
symmetry_se	-0.104321	0.009127	-0.081629	-0.072497	0.200774	0.229977	0.178009
fractal_dimension_se	-0.042641	0.054458	-0.005523	-0.019887	0.283607	0.507318	0.449301
radius_worst	0.969539	0.352573	0.969476	0.962746	0.213120	0.535315	0.688236
texture_worst	0.297008	0.912045	0.303038	0.287489	0.036072	0.248133	0.299879
perimeter_worst	0.965137	0.358040	0.970387	0.959120	0.238853	0.590210	0.729565
area_worst	0.941082	0.343546	0.941550	0.959213	0.206718	0.509604	0.675987
smoothness_worst	0.119616	0.077503	0.150549	0.123523	0.805324	0.565541	0.448822
compactness_worst	0.413463	0.277830	0.455774	0.390410	0.472468	0.865809	0.754968
concavity_worst	0.526911	0.301025	0.563879	0.512606	0.434926	0.816275	0.884103
concave points_worst	0.744214	0.295316	0.771241	0.722017	0.503053	0.815573	0.861323
symmetry_worst	0.163953	0.105008	0.189115	0.143570	0.394309	0.510223	0.409464
fractal_dimension_worst	0.007066	0.119205	0.051019	0.003738	0.499316	0.687382	0.514930

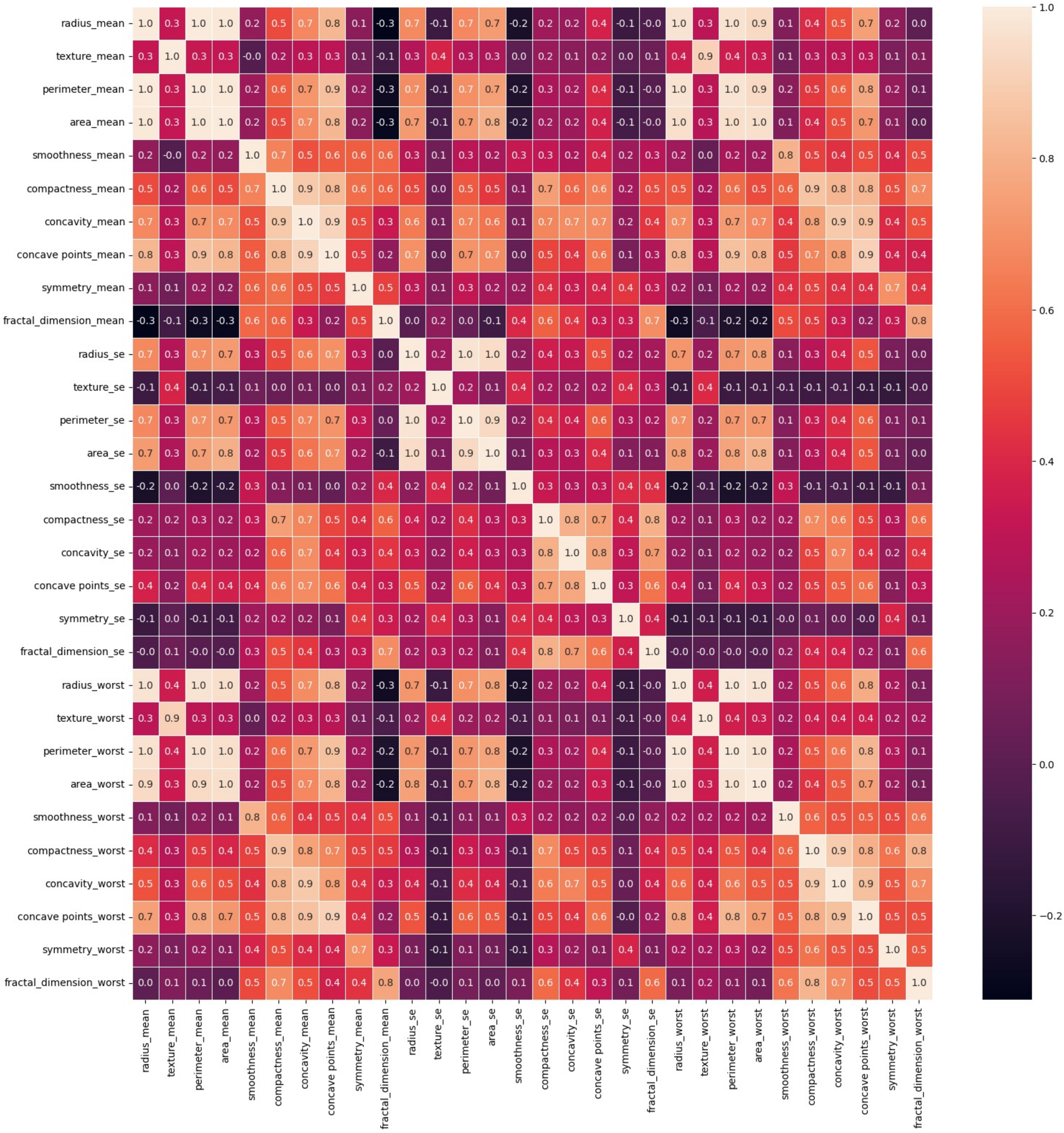
30 rows × 30 columns

In [15]:

```
#Correlation map
import matplotlib.pyplot as plt
f,ax=plt.subplots(figsize=(18,18))
sns.heatmap(data_num.corr(),annot=True, linewidth=.5,fmt='.1f',ax=ax)
```

Out[15]:

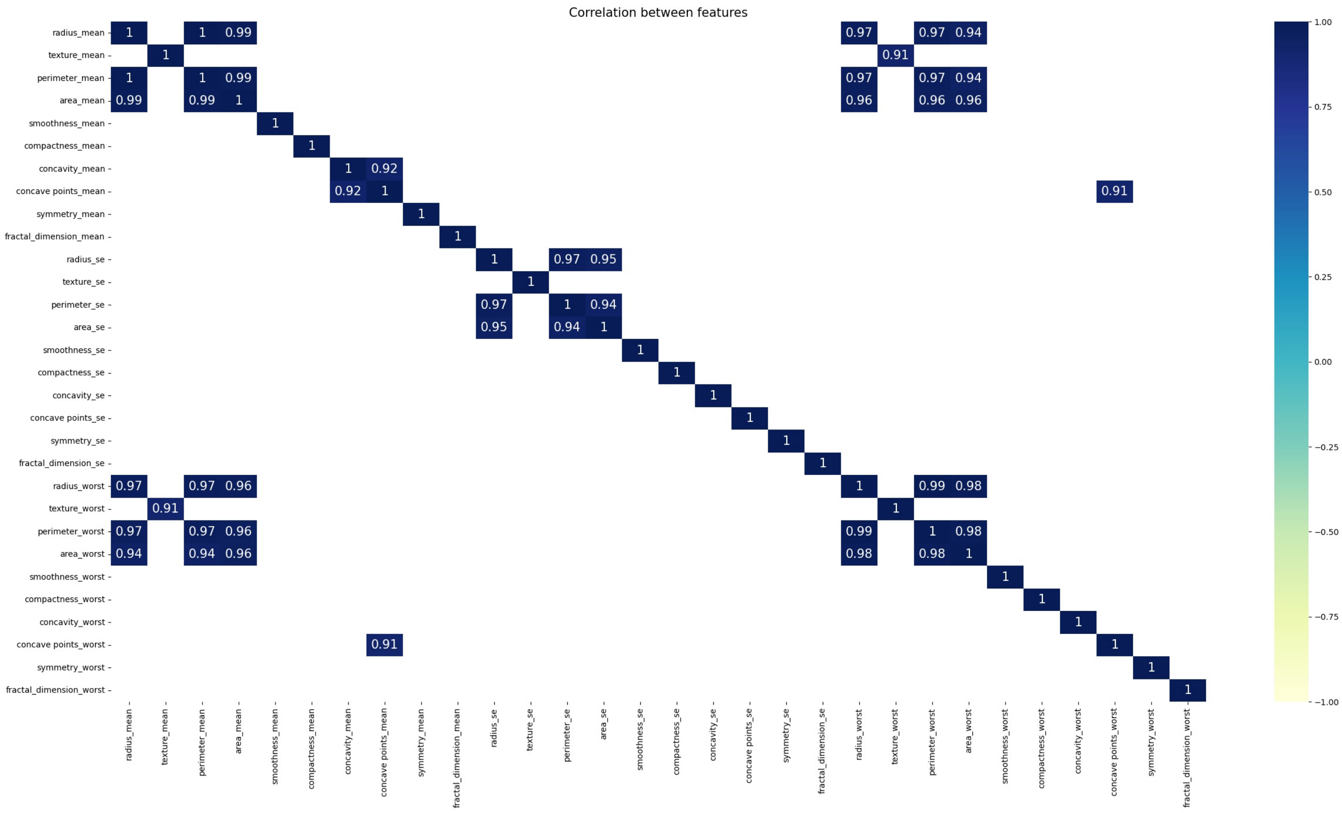
```
<AxesSubplot:>
```



In [16]:

```
import seaborn as sns
#Plotting correlation map
#set the figure size
```

```
plt.figure(figsize=(30,15))
#plotting the heat map
#corr:give the correlation matrix
#cmap:color code used for plotting
#vmax:gives maximum range of values for the chart
#vmin:gives minimum rnge of values for the chart
#annot:prints the correlation values in the chart
#annot-kws:sets the font size of the annotation
#set condition to get a strong correlation between the variables
sns.heatmap(corr[(corr>=0.9)|(corr<=-0.9)],
            cmap='YlGnBu',vmax=1.0,vmin=-1.0,
            annot=True,annot_kws={"size":15})

#set the title
#fontsize=30
plt.title('Correlation between features',fontsize=15)
#display the plot
plt.show()
```



In [17]:
```
drop_list=['perimeter_mean','radius_mean','compactness_mean','concave points_mean','radius_se','perimeter_se','
data_cleaned=data.drop(drop_list,axis=1)
data_cleaned.head()
```

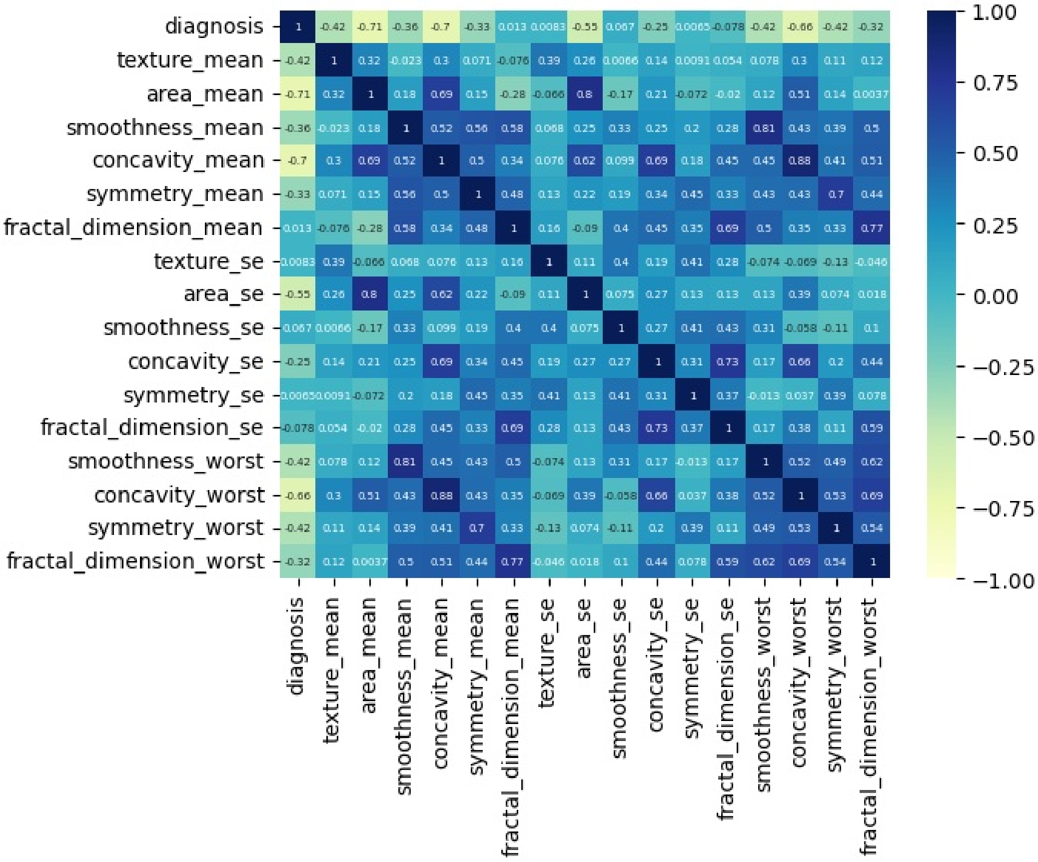
Out[17]:

	diagnosis	texture_mean	area_mean	smoothness_mean	concavity_mean	symmetry_mean	fractal_dimension_mean	texture_se	area_se	s
0	0	10.38	1001.0	0.11840	0.3001	0.2419	0.07871	0.9053	153.40	
1	0	17.77	1326.0	0.08474	0.0869	0.1812	0.05667	0.7339	74.08	
2	0	21.25	1203.0	0.10960	0.1974	0.2069	0.05999	0.7869	94.03	
3	0	20.38	386.1	0.14250	0.2414	0.2597	0.09744	1.1560	27.23	
4	0	14.34	1297.0	0.10030	0.1980	0.1809	0.05883	0.7813	94.44	

In [18]:
```
#These feature pairs are strongly positively correlated to each other, so we should not select these
#features together for training the model
```

In [19]:
```
sns.heatmap(data_cleaned.corr(),
           cmap='YlGnBu',vmax=1.0,vmin=-1.0,
           annot=True, annot_kws={'fontsize': 5})
```

Out[19]: <AxesSubplot:>

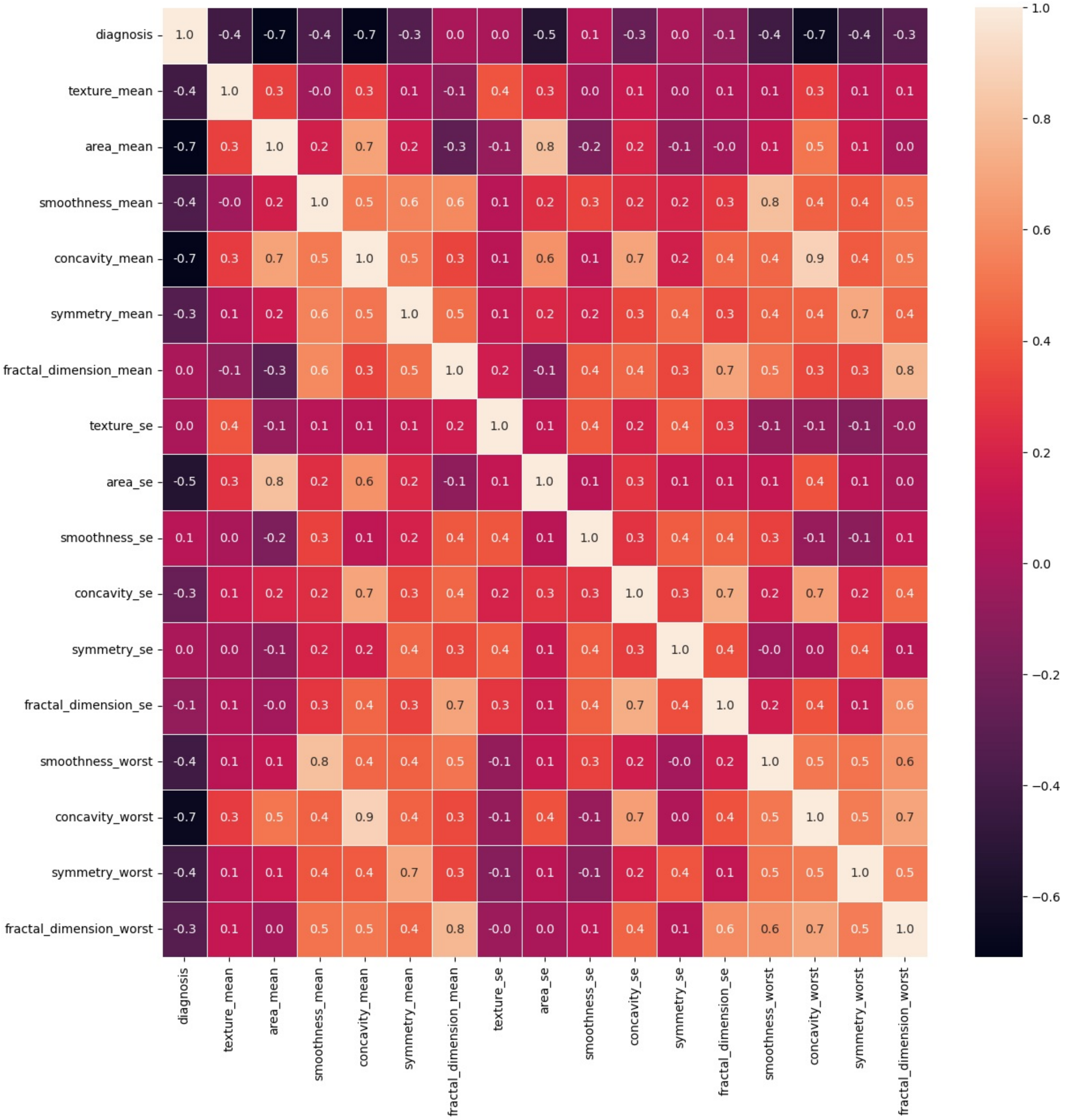


In [20]:
```
#print the name of columns
data_cleaned.columns
```

Out[20]:
```
Index(['diagnosis', 'texture_mean', 'area_mean', 'smoothness_mean',
       'concavity_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'texture_se', 'area_se', 'smoothness_se', 'concavity_se', 'symmetry_se',
       'fractal_dimension_se', 'smoothness_worst', 'concavity_worst',
       'symmetry_worst', 'fractal_dimension_worst'],
      dtype='object')
```

In [21]:
```
#correlation map
f,ax=plt.subplots(figsize=(14,14))
sns.heatmap(data_cleaned.corr(),annot=True,linewidth=.5,fmt='.1f',ax=ax)
```

Out[21]:
```
<AxesSubplot:>
```



In [22]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

x = data_cleaned.drop(['diagnosis'], axis=1)
```

```
y = data_cleaned['diagnosis']

select_feature = SelectKBest(chi2, k=5).fit(x, y)
x = select_feature.transform(x)
x
```

Out[22]:

```
array([[1.038e+01, 1.001e+03, 3.001e-01, 1.534e+02, 7.119e-01],
       [1.777e+01, 1.326e+03, 8.690e-02, 7.408e+01, 2.416e-01],
       [2.125e+01, 1.203e+03, 1.974e-01, 9.403e+01, 4.504e-01],
       ...,
       [2.808e+01, 8.581e+02, 9.251e-02, 4.855e+01, 3.403e-01],
       [2.933e+01, 1.265e+03, 3.514e-01, 8.622e+01, 9.387e-01],
       [2.454e+01, 1.810e+02, 0.000e+00, 1.915e+01, 0.000e+00]])
```

In [23]:

```
print(select_feature)
```

```
SelectKBest(k=5, score_func=<function chi2 at 0x00000198DDBF4040>)
```

chi2 test is used to select top 5 best features from the cleaned data so as to increase model performance

Ensemble Learning

We build an ensemble model using Bagging meta-estimator.

We start withbour dataset gradually proceeding with an analysis

in order to build an ensemble model using Bagging meta-estimator we do the folloing

Split the dataset

Build the model

Predict the values

Compute the accuracy measures

Tabulate the results

In [24]:

```
from sklearn.model_selection import train_test_split
#splitting the dataset into train and test
X_train,X_test,y_train,y_test=train_test_split(x,y,test_size=0.30,random_state=10)
#split the dataset
#print the shape of x_train
print('X_train',X_train.shape)
print('X_test',X_test.shape)
print('y_train',y_train.shape)
print('y_test',y_test.shape)
```

```
X_train (398, 5)
X_test (171, 5)
y_train (398,)
y_test (171,)
```

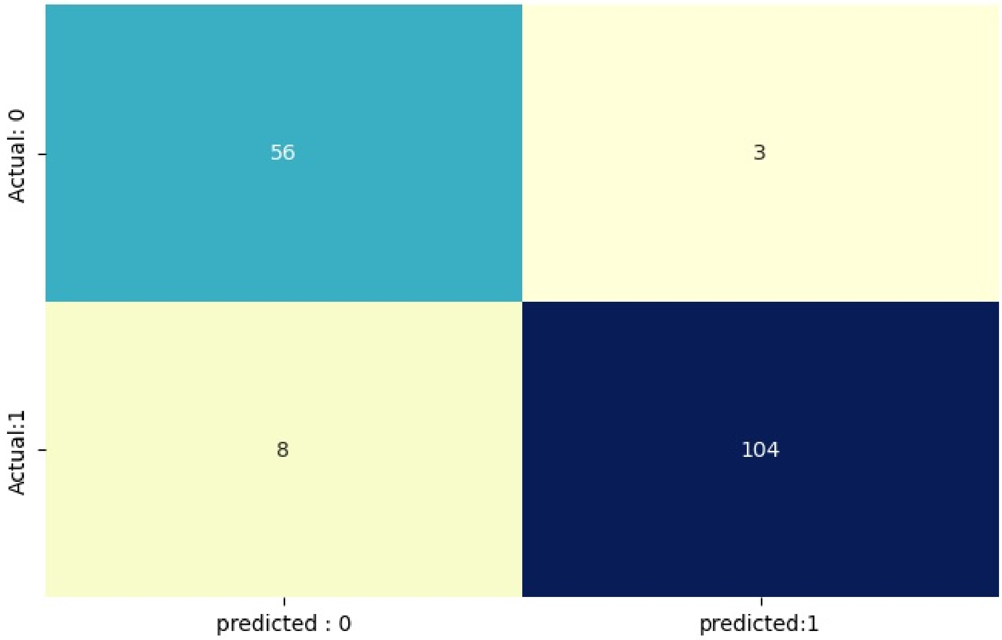
In [25]:

```
#Buid the model
from sklearn.ensemble import BaggingClassifier
from sklearn import tree
#building the model
meta_estimator=BaggingClassifier(tree.DecisionTreeClassifier(random_state=10))
#fit the model
meta_estimator.fit(X_train,y_train)
#predicting the values
y_pred=meta_estimator.predict(X_test)
```

Compute Accuracy Measures

Compute Accuracy Measures

In [26]:
```python
from sklearn.metrics import confusion_matrix

#Compute the confusion matrix
cm= confusion_matrix(y_test,y_pred)

#Label the confusion matrix
conf_matrix=pd.DataFrame(data=cm,columns=['predicted : 0', 'predicted:1'],index=['Actual: 0','Actual:1'])
#Set the size of the plot
plt.figure(figsize=(8,5))
#Plot a heat map
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap="YlGnBu", cbar=False)
plt.show()
```

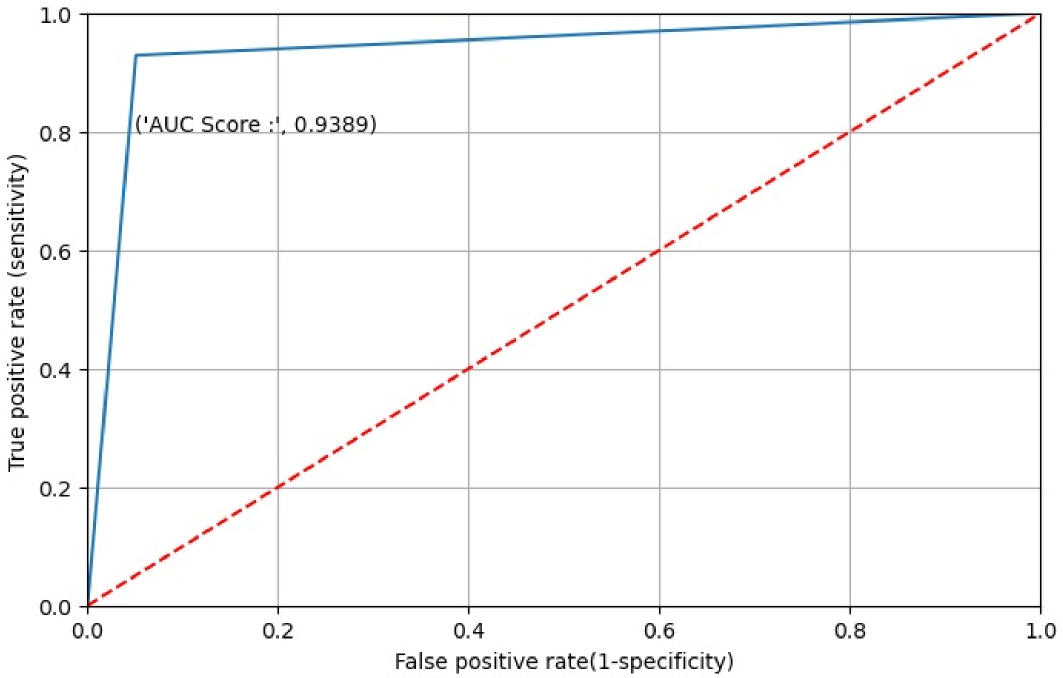


In [27]:
```python
#True Negatives are denoted by 'TN'
#Actual '0' values which are classified correctly

TN=cm[0,0]

#True positives are denoted by 'TP'
#Actual '1' values which are classified correctly

TP=cm[1,1]

#False negatives are denoted by 'FN'
#Actual '1' vaues which are classified wrongly as '0'

FN=cm[1,0]

#False positives are denoted by 'FP'
#Actual '0' values which are classified  wrongly as '1'

FP=cm[0,1]

from sklearn.metrics import classification_report
#Accuracy measures by classification_report()

result=classification_report(y_test,y_pred)

#print the result

print(result)
```

```
              precision    recall  f1-score   support

           0       0.88      0.95      0.91        59
           1       0.97      0.93      0.95       112

    accuracy                           0.94       171
   macro avg       0.92      0.94      0.93       171
weighted avg       0.94      0.94      0.94       171
```

In [28]:
```python
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

#set the figure size

plt.rcParams['figure.figsize']=(8,5)
fpr,tpr,thresholds=roc_curve(y_test,y_pred)
```

```
#plot the ROC curve
plt.plot(fpr,tpr)

#set limits for X and Y axes

plt.xlim([0.0,1.0])
plt.ylim([0.0,1.0])

#plot the straight line showing showing worst prediction for the model

plt.plot([0,1],[0,1],'r--')

#Add the AUC score
plt.text(x=0.05, y=0.8, s=('AUC Score :',round(roc_auc_score(y_test,y_pred),4)))

#Name the plot and both axes
plt.xlabel('False positive rate(1-specificity)')
plt.ylabel('True positive rate (sensitivity)')

#plot the grid
plt.grid(True)
```



In [29]:
```
from sklearn import metrics

#Create the result table for all accuracy scores
#Accuracy measures considered for model comparison are 'Model', 'AUC score','Precision score','Recall score','A

#Create a list of column names
cols=['Model','AUC score','Precision score','Recall score', 'Accuracy score','f1-score']

#Create an empty dataframe of the columns
result_tabulation=pd.DataFrame(columns=cols)

#Compiling the required information
Bagging_Meta_estimator=pd.Series({'Model':"Bagging_Meta_estimator",
                                  'AUC score':metrics.roc_auc_score(y_test,y_pred),
                                  'Precision score':metrics.precision_score(y_test,y_pred),
                                  'Recall score':metrics.recall_score(y_test,y_pred),
                                  'Accuracy score':metrics.accuracy_score(y_test,y_pred),
                                  'f1-score':metrics.f1_score(y_test,y_pred)
                                  })

#Appending our result table
result_tabulation=result_tabulation.append(Bagging_Meta_estimator,ignore_index=True)

#view the result table
result_tabulation
```

Out[29]:

	Model	AUC score	Precision score	Recall score	Accuracy score	f1-score
0	Bagging_Meta_estimator	0.938862	0.971963	0.928571	0.935673	0.949772

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js