Predicting Breast Cancer in a patient

Problem statement: The objective of the dataset is to predict whether the cancer is benign or malignant

Data definition: The dataset consists of several predictor variables and one target variable, Diagnosis. The target variable has values 'Benign' and 'Malignant', where 'Benign' means that the cells are not harmful or there is no cancer and 'Malignant' means that the patient has cancer and the cells have a harmful effect

1.Import packages

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
#Suppress warnings
from warnings import filterwarnings
filterwarnings('ignore')
```

2.Read Data

In [3]:
```
#read the data
data=pd.read_csv("cancer.csv")

#print first five rows of data
data.head()
```

Out[3]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	texture_worst	perimeter_worst	area_worst
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	17.33	184.60	2019.0
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	23.41	158.80	1956.0
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	25.53	152.50	1709.0
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	26.50	98.87	567.7
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	16.67	152.20	1575.0

5 rows × 33 columns

3.Understand and prepare the Data

The process of data preparartion entails cleansing, structuring and integrating data to make it ready for analysis.

3.1 Datatypes and Dimensions

In [4]:
```
data.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
```

From the above dataset, we see that 1.There is an id that cannot be used for classification 2.Unnamed:32 feature includes NaN so we can drop both the columns from the dataset

In [5]:
```
data1=data.drop(['id','Unnamed: 32'],axis=1)
data1.head()
```

Out[5]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	...	radius_worst	texture_worst
0	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	...	25.38	17.33
1	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	...	24.99	23.41
2	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	...	23.57	25.53
3	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	...	14.91	26.50
4	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	...	22.54	16.67

5 rows × 31 columns

In [7]:
```
#shape of dataset
data1.shape
```

Out[7]:
```
(569, 31)
```

In [8]:
```
#plot the target
diag=sns.countplot(data1['diagnosis'],label="count")
B,M=data1["diagnosis"].value_counts()
print("Number of B: ",B)
print("Number of M: ",M)
```

```
Number of B:  357
Number of M:  212
```

3.2 Statistical Summary

In dataset there are numerical varibales. describe() gives the statistical summary of numerical variables.

In [9]:
```
data1.describe()
```

Out[9]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dimension_mean	...	radius_worst
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	...	569.000000
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	0.181162	0.062798	...	16.269190
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.027414	0.007060	...	4.833242
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.106000	0.049960	...	7.930000
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0.161900	0.057700	...	13.010000
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0.179200	0.061540	...	14.970000
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	0.195700	0.066120	...	18.790000
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	0.304000	0.097440	...	36.040000

8 rows × 30 columns

3.3 Label encoding for Target variable

Converting target variable into numeric

In [10]:
```
#replace M with zero and B with one
data1['diagnosis']=data1['diagnosis'].replace('M',0)
data1['diagnosis']=data1['diagnosis'].replace('B',1)
data1.head()
```

Out[10]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	...	radius_worst	texture_worst
0	0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	...	25.38	17.33
1	0	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	...	24.99	23.41
2	0	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	...	23.57	25.53
3	0	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	...	14.91	26.50
4	0	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	...	22.54	16.67

5 rows × 31 columns

3.4 Missing Data treatment

First we have to find whether there is any null values in our gven data set. If there is null values we have to handle it properly , otherwise it effects our prediction

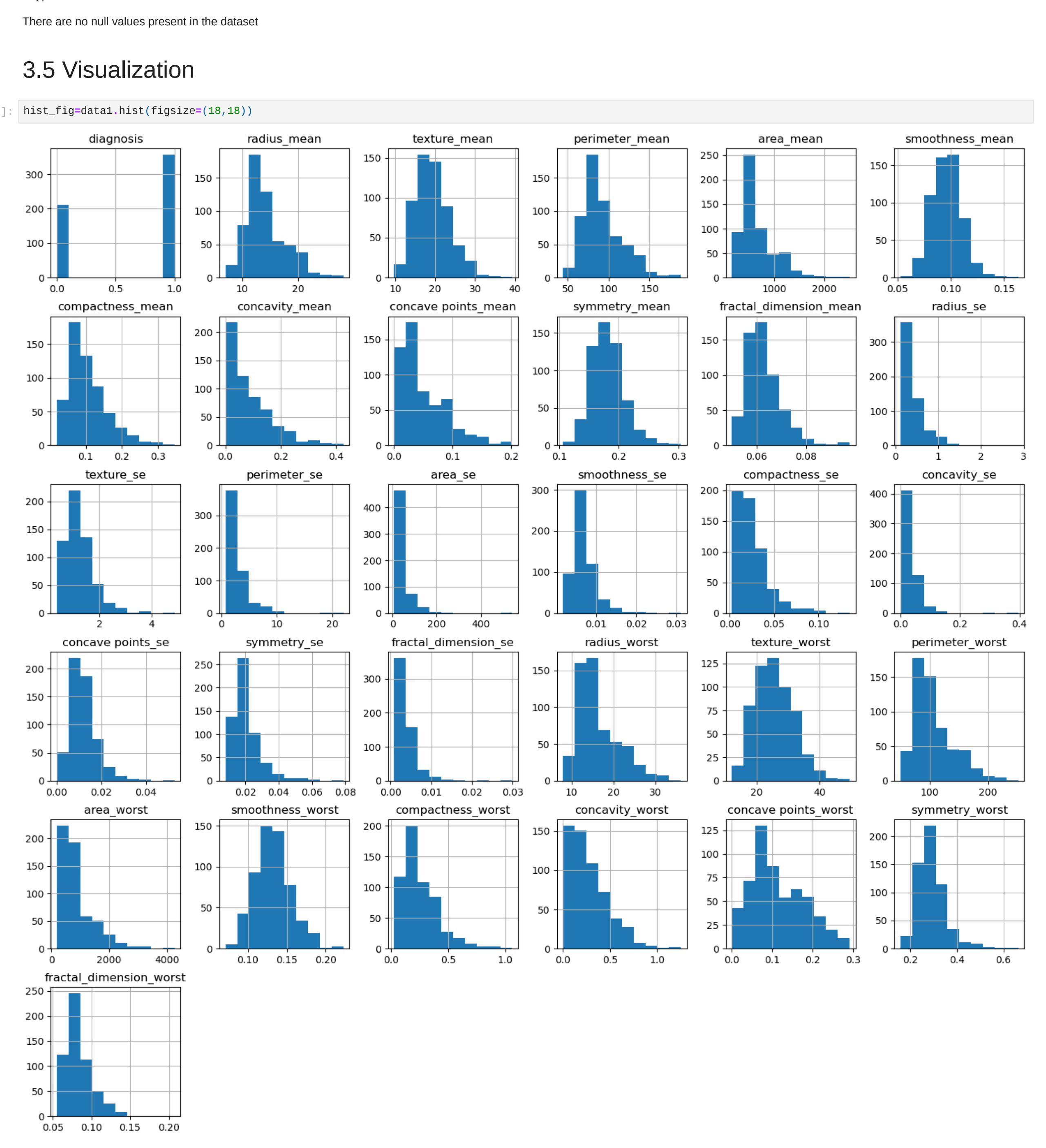
In [11]:
```
#get the count of missing values
data1.isnull().sum()
```

Out[11]:
```
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 null values present in the dataset

3.5 Visualization

In [12]:
```
hist_fig=data1.hist(figsize=(18,18))
```



Correlation

In [13]:
```
target=data1.drop('diagnosis',axis=1)
corr=target.corr()
corr
```

Out[13]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dimension_mean	...
radius_mean	1.000000	0.323782	0.997855	0.987357	0.170581	0.506124	0.676764	0.822529	0.147741	-0.311631	...
texture_mean	0.323782	1.000000	0.329533	0.321086	-0.023389	0.236702	0.302418	0.293464	0.071401	-0.076437	...
perimeter_mean	0.997855	0.329533	1.000000	0.986507	0.207278	0.556936	0.716136	0.850977	0.183027	-0.261477	...
area_mean	0.987357	0.321086	0.986507	1.000000	0.177028	0.498502	0.685983	0.823269	0.151293	-0.283110	...
smoothness_mean	0.170581	-0.023389	0.207278	0.177028	1.000000	0.659123	0.521984	0.553695	0.557775	0.584792	...
compactness_mean	0.506124	0.236702	0.556936	0.498502	0.659123	1.000000	0.883121	0.831135	0.602641	0.565369	...
concavity_mean	0.676764	0.302418	0.716136	0.685983	0.521984	0.883121	1.000000	0.921391	0.500667	0.336783	...
concave points_mean	0.822529	0.293464	0.850977	0.823269	0.553695	0.831135	0.921391	1.000000	0.462497	0.166917	...
symmetry_mean	0.147741	0.071401	0.183027	0.151293	0.557775	0.602641	0.500667	0.462497	1.000000	0.479921	...
fractal_dimension_mean	-0.311631	-0.076437	-0.261477	-0.283110	0.584792	0.565369	0.336783	0.166917	0.479921	1.000000	...
radius_se	0.679090	0.275869	0.691765	0.732562	0.301467	0.497473	0.631925	0.698050	0.303379	0.000111	...
texture_se	-0.097317	0.386358	-0.086761	-0.066280	0.068406	0.046205	0.076218	0.021480	0.128053	0.164174	...
perimeter_se	0.674172	0.281673	0.693135	0.726628	0.296092	0.548905	0.660391	0.710650	0.313893	0.039830	...
area_se	0.735864	0.259845	0.744983	0.800086	0.246552	0.455653	0.617427	0.690299	0.223970	-0.090170	...
smoothness_se	-0.222600	0.006614	-0.202694	-0.166777	0.332375	0.135299	0.098564	0.027653	0.187321	0.401964	...
compactness_se	0.206000	0.191975	0.250744	0.212583	0.318943	0.738722	0.670279	0.490424	0.421659	0.559837	...
concavity_se	0.194204	0.143293	0.228082	0.207660	0.248396	0.570517	0.691270	0.439167	0.342627	0.446630	...
concave points_se	0.376169	0.163851	0.407217	0.372320	0.380676	0.642262	0.683260	0.615634	0.393298	0.341198	...
symmetry_se	-0.104321	0.009127	-0.081629	-0.072497	0.200774	0.229977	0.178009	0.095351	0.449137	0.345007	...
fractal_dimension_se	-0.042641	0.054458	-0.005523	-0.019887	0.283607	0.507318	0.449301	0.257584	0.331786	0.688132	...
radius_worst	0.969539	0.352573	0.969476	0.962746	0.213120	0.535315	0.688236	0.830318	0.185728	-0.253691	...
texture_worst	0.297008	0.912045	0.303038	0.287489	0.036072	0.248133	0.299879	0.292752	0.090651	-0.051269	...
perimeter_worst	0.965137	0.358040	0.970387	0.959120	0.238853	0.590210	0.729565	0.855923	0.219169	-0.205151	...
area_worst	0.941082	0.343546	0.941550	0.959213	0.206718	0.509604	0.675987	0.809630	0.177193	-0.231854	...
smoothness_worst	0.119616	0.077503	0.150549	0.123523	0.805324	0.565541	0.448822	0.452753	0.426675	0.504942	...
compactness_worst	0.413463	0.277830	0.455774	0.390410	0.472468	0.865809	0.754968	0.667454	0.473200	0.458798	...
concavity_worst	0.526911	0.301025	0.563879	0.512606	0.434926	0.816275	0.884103	0.752399	0.433721	0.346234	...
concave points_worst	0.744214	0.295316	0.771241	0.722017	0.503053	0.815573	0.861323	0.910155	0.430297	0.175325	...
symmetry_worst	0.163953	0.105008	0.189115	0.143570	0.394309	0.510223	0.409464	0.375744	0.699826	0.334019	...
fractal_dimension_worst	0.007066	0.119205	0.051019	0.003738	0.499316	0.687382	0.514930	0.368661	0.438413	0.767297	...

30 rows × 30 columns

In [23]:
```
#correlation map
sns.heatmap(target.corr(), cmap='coolwarm')
plt.show()
```

In [24]:
```
drop_list=['perimeter_mean','radius_mean', 'compactness_mean','concave points_mean','radius_se','perimeter_se','radius_worst','perimeter_worst','compactness_wo
data_cleaned=data1.drop(drop_list,axis=1)
data_cleaned.head()
```

Out[24]:

	diagnosis	texture_mean	area_mean	smoothness_mean	concavity_mean	symmetry_mean	fractal_dimension_mean	texture_se	area_se	smoothness_se	concavity_se	symmetry_se
0	0	10.38	1001.0	0.11840	0.3001	0.2419	0.07871	0.9053	153.40	0.006399	0.05373	0.03003
1	0	17.77	1326.0	0.08474	0.0869	0.1812	0.05667	0.7339	74.08	0.005225	0.01860	0.01389
2	0	21.25	1203.0	0.10960	0.1974	0.2069	0.05999	0.7869	94.03	0.006150	0.03832	0.02250
3	0	20.38	386.1	0.14250	0.2414	0.2597	0.09744	1.1560	27.23	0.009110	0.05661	0.05963
4	0	14.34	1297.0	0.10030	0.1980	0.1809	0.05883	0.7813	94.44	0.011490	0.05688	0.01756

In [27]:
```
#correlation plot
plt.figure(figsize=(30, 15))
sns.heatmap(data_cleaned.corr(),cmap='Blues_r', vmax=1.0, vmin=-1.0, annot=True, annot_kws={"size":15})
plt.title("correlation map",fontsize=15)
plt.show()
```

3.8 Feature Selection

In [30]:
```
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[30]:
```
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]])
```

Note: Chi-squared test to select top 5 best features from the cleaned data so as to increase model perfomance

4.Ensemble Learning Techniques

4.1 Bagging meta estimator

Build an ensemble model using bagging meta estiamtor 1.Split the data set 2.Build the model 3.Predict the output 4.Compute accuracy 5.Tabulate the result

1.Split the data set

In [31]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.30,random_state=10)
print("x_train",x_train.shape)
print("y_train",y_train.shape)
print("x_test",x_test.shape)
print("y_test",y_test.shape)
```

```
x_train (398, 5)
y_train (398,)
x_test (171, 5)
y_test (171,)
```

2.Build the model

In [34]:
```
from sklearn.ensemble import BaggingClassifier
from sklearn import tree
model_meta = BaggingClassifier(tree.DecisionTreeClassifier(random_state=10))
model_meta.fit(x_train, y_train)
```

Out[34]:
```
BaggingClassifier(base_estimator=DecisionTreeClassifier(random_state=10))
```

3.predict the values

In [35]:
```
y_pred=model_meta.predict(x_test)
```

4.Compute accuracy

In [37]:
```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred)
conf_matrix=pd.DataFrame(data=cm,columns=['Predicted:0','Predicted:1'],index=['Actual:0','Actual:1'])
plt.figure(figsize=(8,5))
sns.heatmap(conf_matrix,annot=True,fmt='d',cmap='BrBG',cbar=False)
plt.show()
```

In [38]:
```
from sklearn.metrics import classification_report
result=classification_report(y_test,y_pred)
print(result)
```

```
              precision    recall  f1-score   support

           0       0.91      0.88      0.90        59
           1       0.94      0.96      0.95       112

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

5.Tabulate the result

In [41]:
```
from sklearn import metrics
cols=['Model','AUC score','Precision Score','Recall Score','Accuracy Score','f1-score']
result_tab=pd.DataFrame(columns=cols)
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)})
result_tab=result_tab.append(Bagging_meta_estimator,ignore_index=True)
result_tab
```

Out[41]:

	Model	AUC score	Precision Score	Recall Score	Accuracy Score	f1-score	f1-Score
0	Bagging Meta-estimator	0.918357	0.938596	0.955357	0.929825	NaN	0.946903