In [32]:
```
# Importing Libraries
# for reading and manipulating datasets
import pandas as pd
# to perform a wide variety of mathematical operations on arrays
import numpy as np
# for visualizations
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

In [33]:
```
# Reading and opening the datasets
df=pd.read_csv("C:\\Users\\santa\\Downloads\\Breast_Cancer (1).csv")
df.head()
```

Out[33]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_
0	842302	M	17.99	10.38	122.80	1001.0	0.
1	842517	M	20.57	17.77	132.90	1326.0	0.
2	84300903	M	19.69	21.25	130.00	1203.0	0.
3	84348301	M	11.42	20.38	77.58	386.1	0.
4	84358402	M	20.29	14.34	135.10	1297.0	0.

5 rows × 33 columns

In [34]:
```
# Removing unnecessary columns
df.drop(["id","Unnamed: 32"],axis=1,inplace=True)
```

In [35]:
```
df.columns
```

Out[35]:
```
Index(['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 [36]:
```
# setting the featured variables
x=df.drop("diagnosis",axis=1)
x
```

Out[36]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness
0	17.99	10.38	122.80	1001.0	0.11840	0
1	20.57	17.77	132.90	1326.0	0.08474	0
2	19.69	21.25	130.00	1203.0	0.10960	0
3	11.42	20.38	77.58	386.1	0.14250	0
4	20.29	14.34	135.10	1297.0	0.10030	0
...	...	...	...	...	...	
564	21.56	22.39	142.00	1479.0	0.11100	0
565	20.13	28.25	131.20	1261.0	0.09780	0
566	16.60	28.08	108.30	858.1	0.08455	0
567	20.60	29.33	140.10	1265.0	0.11780	0
568	7.76	24.54	47.92	181.0	0.05263	0

569 rows × 30 columns

In [37]:
```
# exploring the datasets
x.describe()
```

Out[37]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactn
count	569.000000	569.000000	569.000000	569.000000	569.000000	5
mean	14.127292	19.289649	91.969033	654.889104	0.096360	
std	3.524049	4.301036	24.298981	351.914129	0.014064	
min	6.981000	9.710000	43.790000	143.500000	0.052630	
25%	11.700000	16.170000	75.170000	420.300000	0.086370	
50%	13.370000	18.840000	86.240000	551.100000	0.095870	
75%	15.780000	21.800000	104.100000	782.700000	0.105300	
max	28.110000	39.280000	188.500000	2501.000000	0.163400	

8 rows × 30 columns

In [38]:
```
# to get the details of the null values
pd.isnull(df).sum()
```

Out[38]:
```
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 [39]:
```
# checking correlations
corr=x.corr()
corr
```

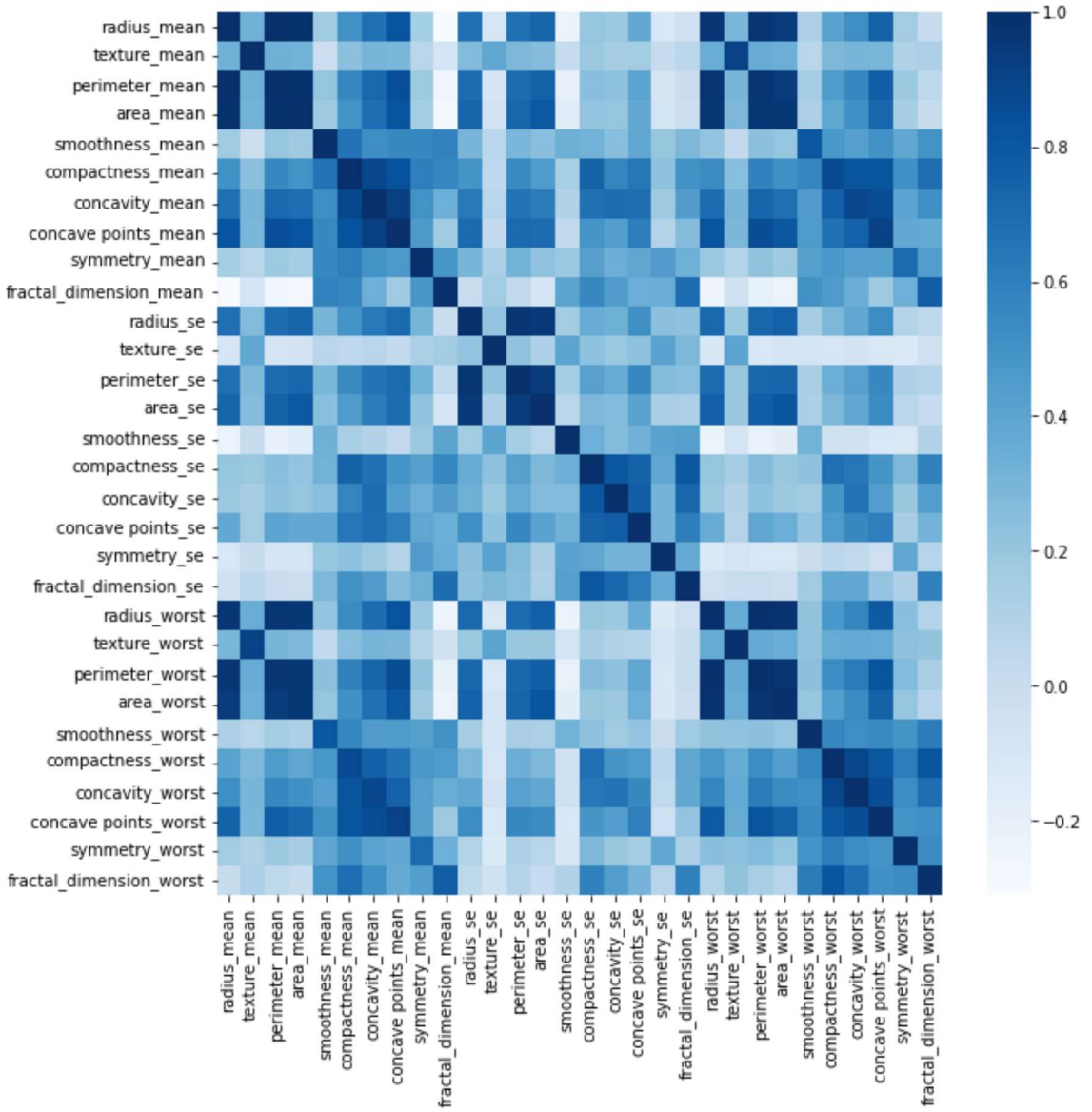
Out[39]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_
radius_mean	1.000000	0.323782	0.997855	0.987357	0.1
texture_mean	0.323782	1.000000	0.329533	0.321086	-0.0
perimeter_mean	0.997855	0.329533	1.000000	0.986507	0.2
area_mean	0.987357	0.321086	0.986507	1.000000	0.1
smoothness_mean	0.170581	-0.023389	0.207278	0.177028	1.0
compactness_mean	0.506124	0.236702	0.556936	0.498502	0.6
concavity_mean	0.676764	0.302418	0.716136	0.685983	0.5
concave points_mean	0.822529	0.293464	0.850977	0.823269	0.5
symmetry_mean	0.147741	0.071401	0.183027	0.151293	0.5
fractal_dimension_mean	-0.311631	-0.076437	-0.261477	-0.283110	0.5
radius_se	0.679090	0.275869	0.691765	0.732562	0.3
texture_se	-0.097317	0.386358	-0.086761	-0.066280	0.0
perimeter_se	0.674172	0.281673	0.693135	0.726628	0.2
area_se	0.735864	0.259845	0.744983	0.800086	0.2
smoothness_se	-0.222600	0.006614	-0.202694	-0.166777	0.3
compactness_se	0.206000	0.191975	0.250744	0.212583	0.3
concavity_se	0.194204	0.143293	0.228082	0.207660	0.2
concave points_se	0.376169	0.163851	0.407217	0.372320	0.3
symmetry_se	-0.104321	0.009127	-0.081629	-0.072497	0.2
fractal_dimension_se	-0.042641	0.054458	-0.005523	-0.019887	0.2
radius_worst	0.969539	0.352573	0.969476	0.962746	0.2
texture_worst	0.297008	0.912045	0.303038	0.287489	0.0
perimeter_worst	0.965137	0.358040	0.970387	0.959120	0.2
area_worst	0.941082	0.343546	0.941550	0.959213	0.2
smoothness_worst	0.119616	0.077503	0.150549	0.123523	0.8
compactness_worst	0.413463	0.277830	0.455774	0.390410	0.4
concavity_worst	0.526911	0.301025	0.563879	0.512606	0.4
concave points_worst	0.744214	0.295316	0.771241	0.722017	0.5
symmetry_worst	0.163953	0.105008	0.189115	0.143570	0.3
fractal_dimension_worst	0.007066	0.119205	0.051019	0.003738	0.4

30 rows × 30 columns

In [82]:
```
# visualizing the heatmap of correlation
plt.figure(figsize=(10,10))
sns.heatmap(corr,cmap='Blues')
```

Out[82]: <AxesSubplot: >



conducting the PCA

In [41]:
```
#Standardization
x= (x-x.mean())/x.std()
x.head()
```

Out[41]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_m
0	1.096100	-2.071512	1.268817	0.983510	1.567087	3.280
1	1.828212	-0.353322	1.684473	1.907030	-0.826235	-0.486
2	1.578499	0.455786	1.565126	1.557513	0.941382	1.052
3	-0.768233	0.253509	-0.592166	-0.763792	3.280667	3.399
4	1.748758	-1.150804	1.775011	1.824624	0.280125	0.538

5 rows × 30 columns

In [42]:
```
# finding co_varience matrix
cov_mx= np.cov(x.T)
cov_mx
```

In [43]:

```
# Finding eigen_pairs
from numpy.linalg import eig
eig_val, eig_vec=eig(cov_mx)

print("Eigenvector : ", eig_vec)
print("\nEigenvalues : ", eig_val)
```

```
  -1.04991974e-01  9.23439434e-02]]

Eigenvalues :  [1.32816077e+01 5.69135461e+00 2.81794898e+00 1.98064047e+00
 1.64873055e+00 1.20735661e+00 6.75220114e-01 4.76617140e-01
 4.16894812e-01 3.50693457e-01 2.93915696e-01 2.61161370e-01
 2.41357496e-01 1.57009724e-01 9.41349650e-02 7.98628010e-02
 5.93990378e-02 5.26187835e-02 4.94775918e-02 1.33044823e-04
 7.48803097e-04 1.58933787e-03 6.90046388e-03 8.17763986e-03
 1.54812714e-02 1.80550070e-02 2.43408378e-02 2.74394025e-02
 3.11594025e-02 2.99728939e-02]
```

In [44]:

```
eig_val, eig_vec=eig(cov_mx)

print("Eigenvector : ", eig_vec)
print("\nEigenvalues : ", eig_val)
```

```
  -1.04991974e-01  9.23439434e-02]]

Eigenvalues :  [1.32816077e+01 5.69135461e+00 2.81794898e+00 1.98064047e+00
 1.64873055e+00 1.20735661e+00 6.75220114e-01 4.76617140e-01
 4.16894812e-01 3.50693457e-01 2.93915696e-01 2.61161370e-01
 2.41357496e-01 1.57009724e-01 9.41349650e-02 7.98628010e-02
 5.93990378e-02 5.26187835e-02 4.94775918e-02 1.33044823e-04
 7.48803097e-04 1.58933787e-03 6.90046388e-03 8.17763986e-03
 1.54812714e-02 1.80550070e-02 2.43408378e-02 2.74394025e-02
 3.11594025e-02 2.99728939e-02]
```

In [45]:

```
#eigen value and eigen vector pair as tuples
eig_pairs= [(np.abs(eig_val[i]), eig_vec[:,i]) for i in range (len(eig_val))]
eig_pairs
```

```
        -0.39662323,  0.09697732,  0.1864516 ,  0.02458369,  0.20722186,
         0.17493043, -0.05698648, -0.07292764, -0.13185041, -0.0312107 ,
        -0.17316455, -0.01593998,  0.12954655,  0.01951493,  0.0841712 ,
        -0.07070972,  0.11818972, -0.11803403,  0.03828995,  0.04796476,
         0.62438494, -0.11577034, -0.26319634, -0.04529962, -0.28013348])),
(0.018055007000150048,
 array([-0.18257944,  0.09878679, -0.11664888,  0.06984834,  0.06869742,
        -0.10413552,  0.04474106,  0.0840277 ,  0.01933947, -0.13326055,
        -0.55870157,  0.0242673 ,  0.51675039, -0.02246072,  0.01563119,
        -0.12177779,  0.18820504, -0.10966898,  0.0032262 ,  0.07519442,
        -0.15683037, -0.1184846 ,  0.23711317,  0.14406303, -0.01099014,
         0.18674995, -0.28885257,  0.10734024, -0.01438181,  0.03782545])),
(0.02434083776697319,
 array([ 9.85526942e-02,  5.54997454e-04,  4.02447050e-02, -7.77727342e-03,
         2.06657211e-02, -5.23603957e-02, -3.24870378e-01,  5.14087968e-02,
         5.12005770e-02,  8.46898562e-02,  2.64125317e-01,  8.73880467e-04,
        -9.00742110e-02, -9.82150746e-02,  5.98177179e-02, -9.10387102e-03,
         3.87542329e-01, -3.51755074e-01,  4.23628949e-02, -8.57810992e-02,
         5.56767923e-02,  8.92289971e-03, -6.33448296e-02, -1.90889625e-01,
        -9.36901494e-02,  1.47920925e-01, -2.86433135e-01,  5.67527797e-01,
        -1.21343451e-01, -7.62533821e-03])),
(0.027439402531630255,
 array([-0.0729289 , -0.09480063, -0.07516048, -0.09756578, -0.06382295,
         0.09807756,  0.185212  ,  0.31185243,  0.01840673, -0.28786888,
         0.15027468, -0.04845693, -0.1593528 , -0.06423262, -0.0505449 ,
         0.04528769,  0.20521269,  0.07254538,  0.08465443, -0.24470508,
         0.09629821,  0.11111202, -0.01722163,  0.09695982,  0.06825409,
        -0.02967641, -0.46042619, -0.29984056, -0.09714484,  0.46947115])),
(0.031159402450160738,
 array([-0.04969866, -0.24413499, -0.01766501, -0.09014376,  0.01710096,
         0.48868633, -0.03338709, -0.23540761,  0.02606916, -0.17563722,
        -0.0908005 , -0.07165999, -0.17725063,  0.27420115,  0.09006148,
        -0.46109822,  0.06694617,  0.06886829,  0.10738529,  0.2223453 ,
        -0.00562691,  0.3005998 ,  0.01100386,  0.06004739, -0.1297239 ,
         0.22928059, -0.04648279,  0.03302234, -0.11675924, -0.10499197])),
(0.029972893911007485,
 array([ 0.06857001, -0.44836947,  0.06976904,  0.01844328,  0.11949175,
        -0.1926214 , -0.00557175,  0.00942382,  0.08693848,  0.07627184,
        -0.08638677, -0.21707197,  0.30495016, -0.19258779,  0.07209873,
         0.14038657, -0.06304793, -0.03437532,  0.09769953, -0.06284328,
        -0.0072939 ,  0.59444014,  0.0920236 , -0.14679013, -0.16484924,
        -0.18137487,  0.13210059, -0.00088608, -0.16270855,  0.09234394]))]
```

In [46]:

```
#sorting tuples from high to low
eig_pairs.sort(key = lambda x: x[0], reverse= True)
print("Eigenvalues in descending order: ")
for i in eig_pairs:
    print(i[0])
```

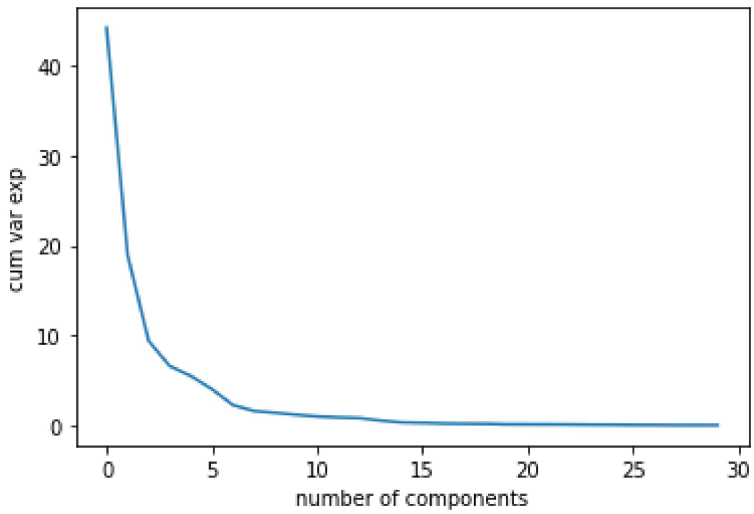
```
Eigenvalues in descending order:
13.281607682257924
5.6913546132099295
2.817948977229412
1.9806404746410475
1.6487305477038783
1.2073566119650032
0.6752201138947512
0.4766171400063975
0.4168948123677332
0.3506934568239444
0.2939156962794053
0.2611613702213645
0.24135749615901977
0.1570097236477906
0.0941349650288221
0.0798628009545697
0.05939903775972825
0.052618783506790785
0.049477591776754745
0.031159402450160738
0.029972893911007485
0.027439402531630255
0.02434083776697319
0.018055007000150048
0.015481271374955403
0.0081776398643251
0.006900463875178697
0.0015893378711427711
0.0007488030974062759
0.0001330448228208199
```

In [47]:
```
tot= sum(eig_val)
var_exp= [(i/tot)*100 for i in sorted(eig_val, reverse=True)]
cum_var_exp= np.cumsum(var_exp)
print("Variance explained by each component is \n", var_exp)
print("-----------------------------------------------------------------------")
print("cumulative variance explained as we proceed \n", cum_var_exp)
```

```
Variance explained by each component is
 [44.27202560752639, 18.971182044033085, 9.393163257431368, 6.602134915470155,
5.495768492346258, 4.024522039883341, 2.2507337129825027, 1.588723800021324, 1.
3896493745591099, 1.1689781894131475, 0.979718987598017, 0.8705379007378811, 0.
804524987196732, 0.523365745492635, 0.3137832167627401, 0.26620933651523215, 0.
19799679253242738, 0.17539594502263584, 0.16492530592251572, 0.1038646748338690
6, 0.09990964637002489, 0.09146467510543413, 0.08113612588991058, 0.06018335666
716679, 0.051604237916517984, 0.027258799547750318, 0.023001546250595643, 0.005
297792903809234, 0.002496010324687585, 0.00044348274273606604]
-----------------------------------------------------------------------
cumulative variance explained as we proceed
 [ 44.27202561  63.24320765  72.63637091  79.23850582  84.73427432
  88.75879636  91.00953007  92.59825387  93.98790324  95.15688143
  96.13660042  97.00713832  97.81166331  98.33502905  98.64881227
  98.91502161  99.1130184   99.28841435  99.45333965  99.55720433
  99.65711397  99.74857865  99.82971477  99.88989813  99.94150237
  99.96876117  99.99176271  99.99706051  99.99955652 100.        ]
```

In [84]:
```
#SCREE plot
plt.plot(var_exp)
plt.xlabel("number of components")
```

```
plt.ylabel("cum var exp")
plt.show()
```



since first three eigen values explain almost 73% of total variation , we can use three PC's

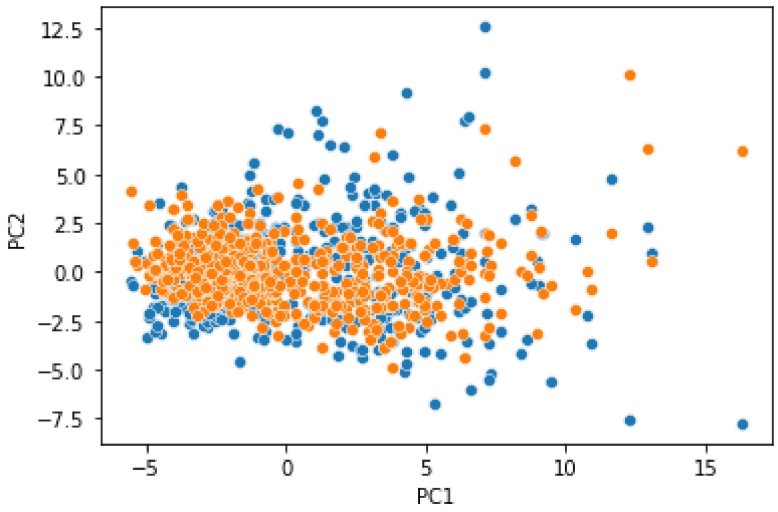
In [50]:
```
# create a new data frame of principal components
df2= pd.DataFrame()
```

In [51]:
```
p1=x.dot(eig_vec.T[0])
p2=x.dot(eig_vec.T[1])
p3=x.dot(eig_vec.T[2])
```

In [75]:
```
df2['PC1']= p1
df2['PC2']= p2
df2['PC3']= p3
```

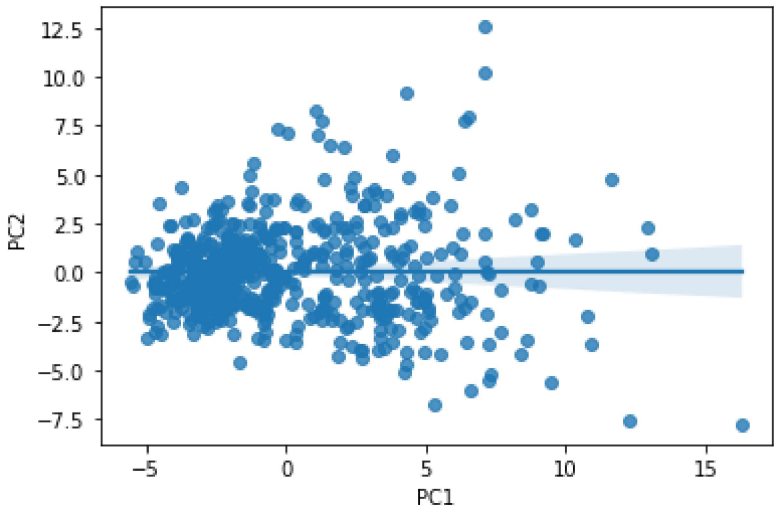
In [85]:
```
sns.scatterplot(x="PC1",y="PC2",data=df2)
sns.scatterplot(x="PC1",y="PC3",data=df2)
```

Out[85]:
```
<AxesSubplot: xlabel='PC1', ylabel='PC2'>
```



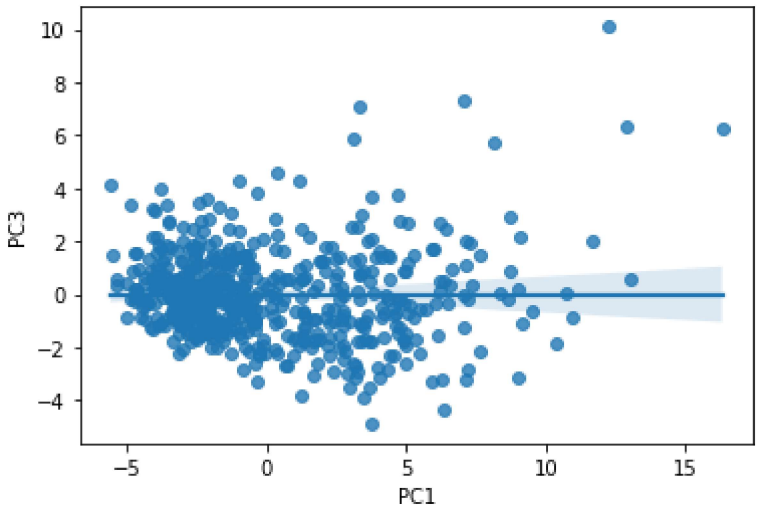
In [92]:
```
# correlation between principal components
sns.regplot(x='PC1',y='PC2',data=df2)
```

Out[92]:
```
<AxesSubplot: xlabel='PC1', ylabel='PC2'>
```



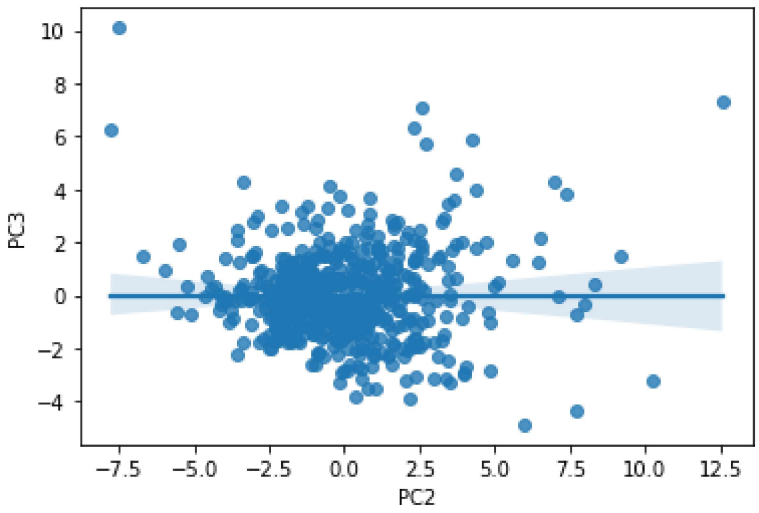
In [94]:
```
sns.regplot(x='PC1',y='PC3',data=df2)
```

Out[94]: <AxesSubplot: xlabel='PC1', ylabel='PC3'>



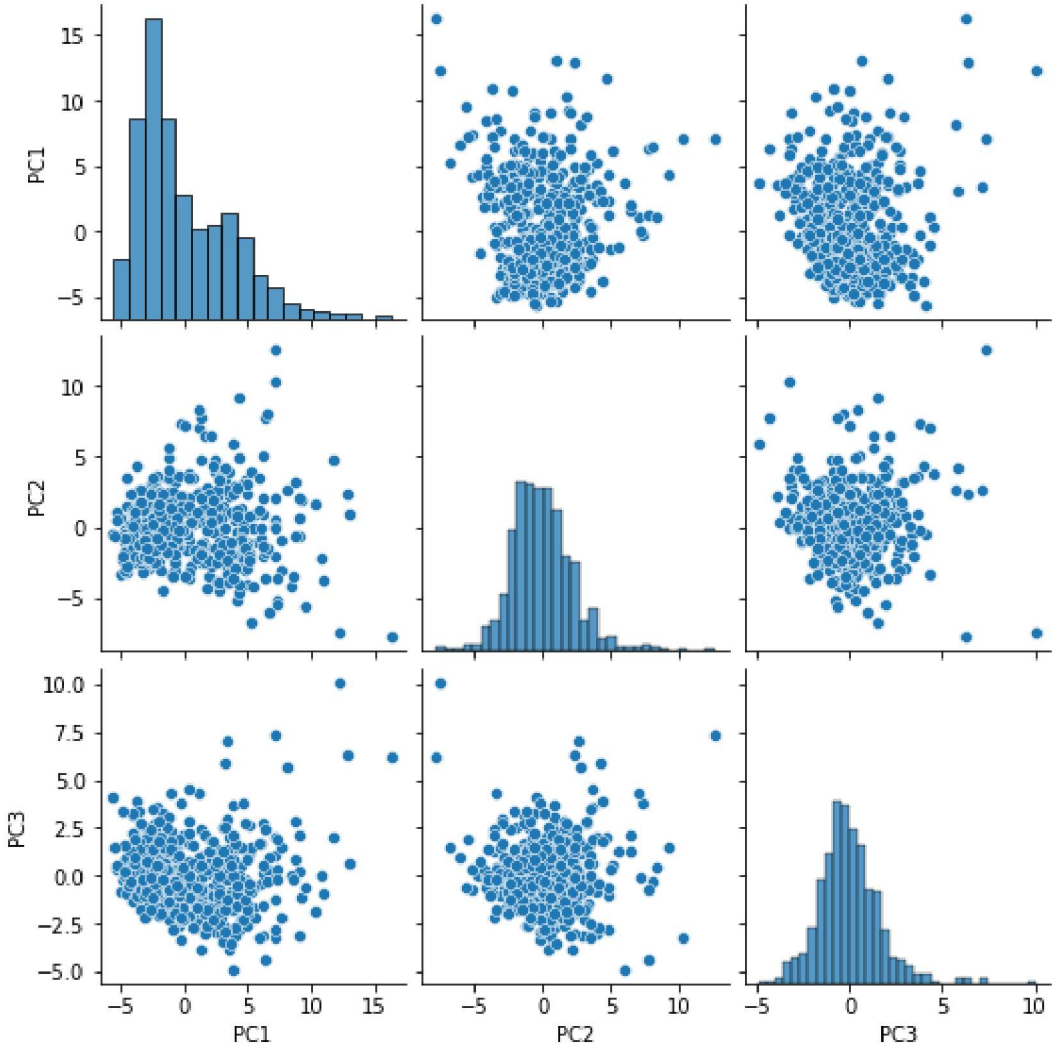
In [98]:
```
sns.regplot(x='PC2',y='PC3',data=df2)
```

Out[98]: <AxesSubplot: xlabel='PC2', ylabel='PC3'>



In [100…
```
sns.pairplot(data=df2)
```

Out[100]: <seaborn.axisgrid.PairGrid at 0x15301170580>



In [ ]:

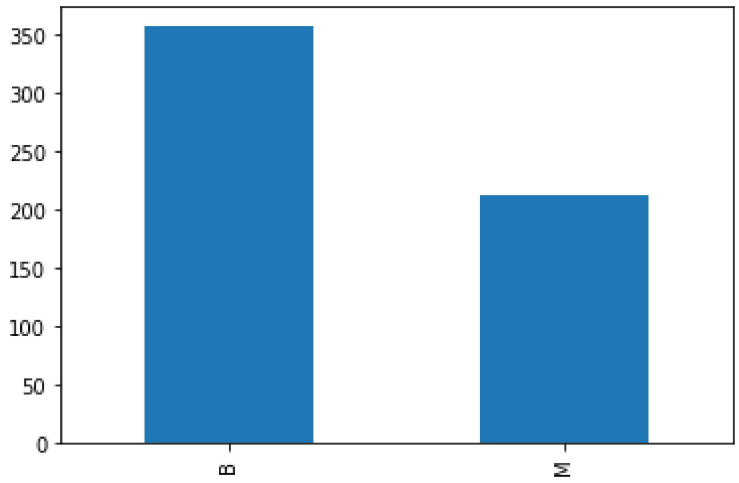
classification

In [53]:
```
# defining the target variable
y= df['diagnosis']
y.head()
```

Out[53]:
```
0    M
1    M
2    M
3    M
4    M
Name: diagnosis, dtype: object
```

In [54]:
```
df['diagnosis'].value_counts().plot.bar()
```

Out[54]:
```
<AxesSubplot: >
```



In [55]:
```
# replace M and B by 0 & 1
df['diagnosis']=df['diagnosis'].map({'M':1,'B':0})
```

In [56]:
```
df2['Y']=y
```

In [59]:
```
df2.head()
```

Out[59]:

	PC1	PC2	PC3	Y
0	9.184755	1.946870	-1.122179	M
1	2.385703	-3.764859	-0.528827	M
2	5.728855	-1.074229	-0.551263	M
3	7.116691	10.266556	-3.229948	M
4	3.931842	-1.946359	1.388545	M

In [60]:
```
x=df2.drop('Y',axis=1)
```

Training Data

In [58]:
```
from sklearn.model_selection import train_test_split
```

In [61]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)
```

In [62]:
```
# Applying Classifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
```

In [63]:
```
rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
```

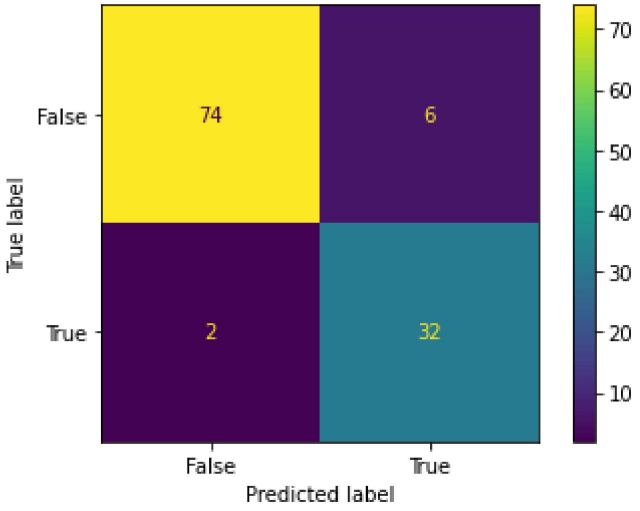
Out[63]:
```
▾ RandomForestClassifier

RandomForestClassifier()
```

In [67]:
```
y_pred=rfc.predict(x_test)
y_pred
```

Out[67]:
```
array(['M', 'B', 'B', 'B', 'M', 'M', 'B', 'B', 'M', 'B', 'B', 'B', 'B',
       'B', 'M', 'B', 'B', 'B', 'M', 'B', 'M', 'M', 'M', 'B', 'B', 'B',
       'M', 'B', 'M', 'B', 'B', 'M', 'M', 'B', 'B', 'M', 'M', 'B', 'B',
       'M', 'B', 'B', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'M', 'M', 'M', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', 'B', 'M', 'B', 'M', 'M', 'B', 'M', 'B', 'M', 'B', 'M', 'B',
       'M', 'B', 'B', 'B', 'B', 'B', 'B', 'M', 'B', 'B', 'B', 'M', 'M',
       'B', 'M', 'B', 'B', 'B', 'B', 'B', 'M', 'B', 'B', 'B', 'B', 'B',
       'M', 'B', 'M', 'B', 'B', 'M', 'M', 'B', 'B', 'M'], dtype=object)
```

In [74]:
```
#confusion matrix
from sklearn import metrics
confusion_matrix=metrics.confusion_matrix(y_test,y_pred)
con_display=metrics.ConfusionMatrixDisplay(confusion_matrix=confusion_matrix,dis
con_display.plot()
```

Out[74]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x15378a032e0
>
```



In [66]:
```
# checking the accuracy of the model
print(accuracy_score(y_test,y_pred))
```

```
0.9298245614035088
```

In [ ]: