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()
```

...

In [34]:

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

In [35]:

```
df.columns
```

...

In [36]:

```
# setting the featured variables
x=df.drop("diagnosis",axis=1)
x
```

...

In [37]:

```
# exploring the datasets
x.describe()
```

...

In [38]:

```
# to get the details of the null values
pd.isnull(df).sum()
```

...

In [39]:

```
# checking correlations
corr=x.corr()
corr
```

Out[39]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothnes:
radius_mean	1.000000	0.323782	0.997855	0.987357	0
texture_mean	0.323782	1.000000	0.329533	0.321086	-0
perimeter_mean	0.997855	0.329533	1.000000	0.986507	0
area_mean	0.987357	0.321086	0.986507	1.000000	0
smoothness_mean	0.170581	-0.023389	0.207278	0.177028	1
compactness_mean	0.506124	0.236702	0.556936	0.498502	0
concavity_mean	0.676764	0.302418	0.716136	0.685983	0
concave points_mean	0.822529	0.293464	0.850977	0.823269	0
symmetry_mean	0.147741	0.071401	0.183027	0.151293	0
fractal_dimension_mean	-0.311631	-0.076437	-0.261477	-0.283110	0
radius_se	0.679090	0.275869	0.691765	0.732562	0
texture_se	-0.097317	0.386358	-0.086761	-0.066280	0
perimeter_se	0.674172	0.281673	0.693135	0.726628	0
area_se	0.735864	0.259845	0.744983	0.800086	0
smoothness_se	-0.222600	0.006614	-0.202694	-0.166777	0
compactness_se	0.206000	0.191975	0.250744	0.212583	0
concavity_se	0.194204	0.143293	0.228082	0.207660	0
concave points_se	0.376169	0.163851	0.407217	0.372320	0
symmetry_se	-0.104321	0.009127	-0.081629	-0.072497	0
fractal_dimension_se	-0.042641	0.054458	-0.005523	-0.019887	0
radius_worst	0.969539	0.352573	0.969476	0.962746	0
texture_worst	0.297008	0.912045	0.303038	0.287489	0
perimeter_worst	0.965137	0.358040	0.970387	0.959120	0
area_worst	0.941082	0.343546	0.941550	0.959213	0
smoothness_worst	0.119616	0.077503	0.150549	0.123523	0
compactness_worst	0.413463	0.277830	0.455774	0.390410	0
concavity_worst	0.526911	0.301025	0.563879	0.512606	0
concave points_worst	0.744214	0.295316	0.771241	0.722017	0
symmetry_worst	0.163953	0.105008	0.189115	0.143570	0
fractal_dimension_worst	0.007066	0.119205	0.051019	0.003738	0

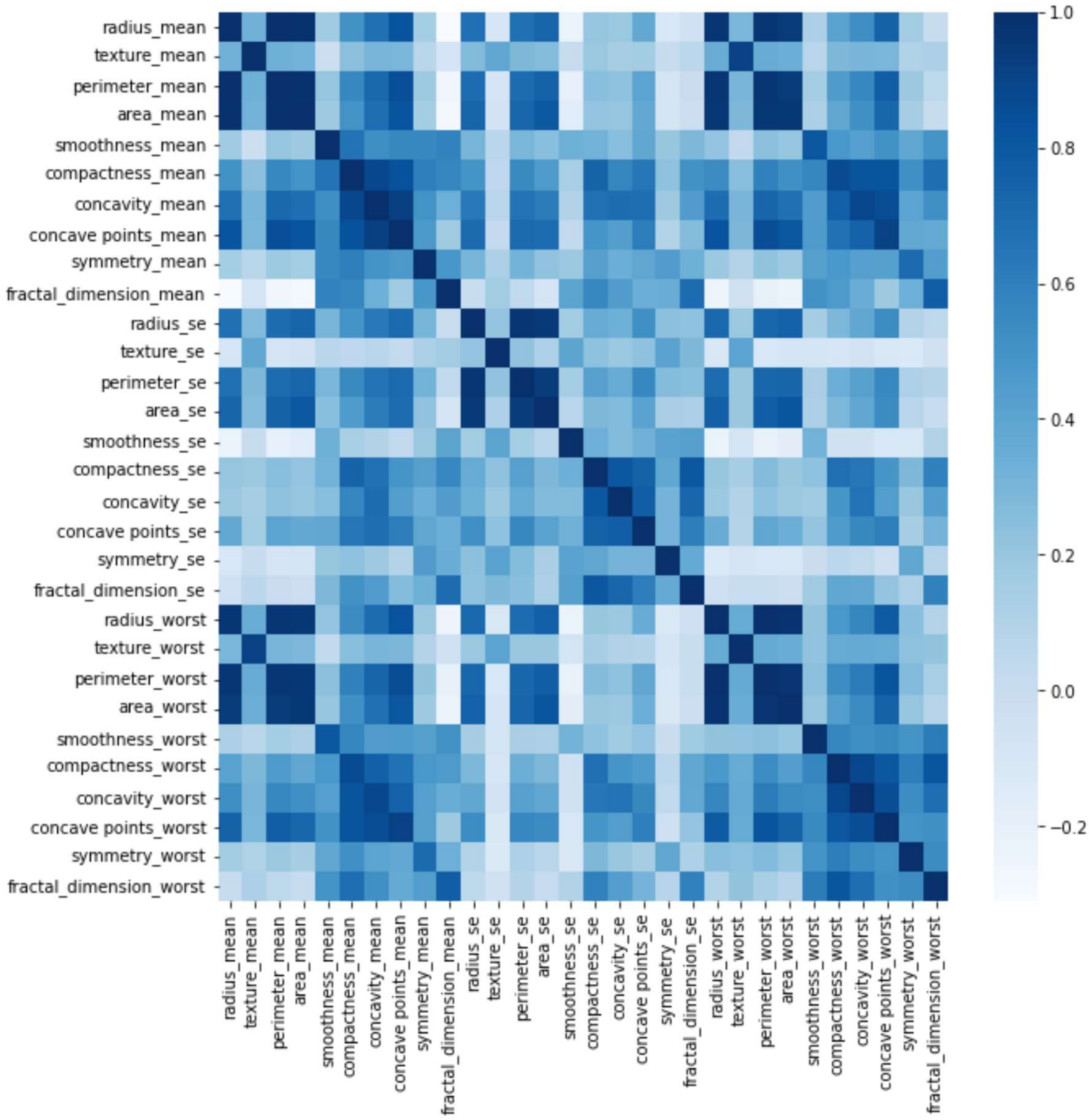
30 rows × 30 columns

In [82]:

```python
# 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_
0	1.096100	-2.071512	1.268817	0.983510	1.567087	3.2
1	1.828212	-0.353322	1.684473	1.907030	-0.826235	-0.4
2	1.578499	0.455786	1.565126	1.557513	0.941382	1.0
3	-0.768233	0.253509	-0.592166	-0.763792	3.280667	3.3
4	1.748758	-1.150804	1.775011	1.824624	0.280125	0.5

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)
```

...

In [44]:

```
eig_val, eig_vec=eig(cov_mx)

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

...

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
```

...

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])
```

...

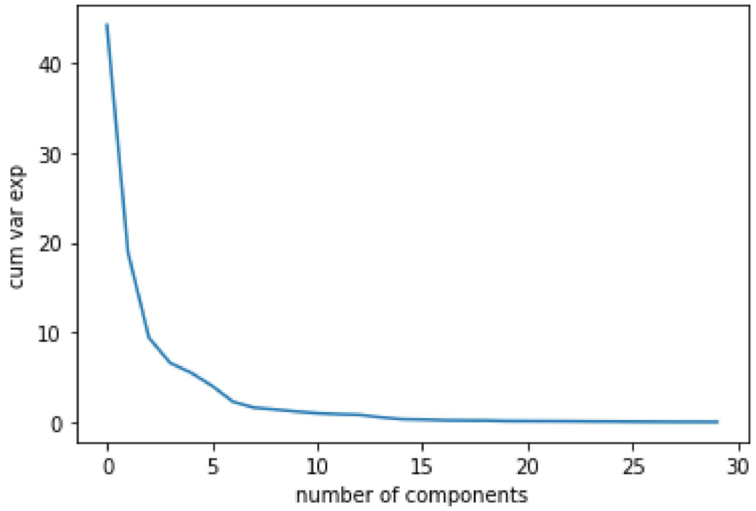
In [47]:

```
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, 6.602134915470
155, 5.495768492346258, 4.024522039883341, 2.2507337129825027, 1.588723800
021324, 1.3896493745591099, 1.1689781894131475, 0.979718987598017, 0.87053
79007378811, 0.804524987196732, 0.523365745492635, 0.3137832167627401, 0.2
6620933651523215, 0.19799679253242738, 0.17539594502263584, 0.164925305922
51572, 0.10386467483386906, 0.09990964637002489, 0.09146467510543413, 0.08
113612588991058, 0.06018335666716679, 0.051604237916517984, 0.027258799547
750318, 0.023001546250595643, 0.005297792903809234, 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]:

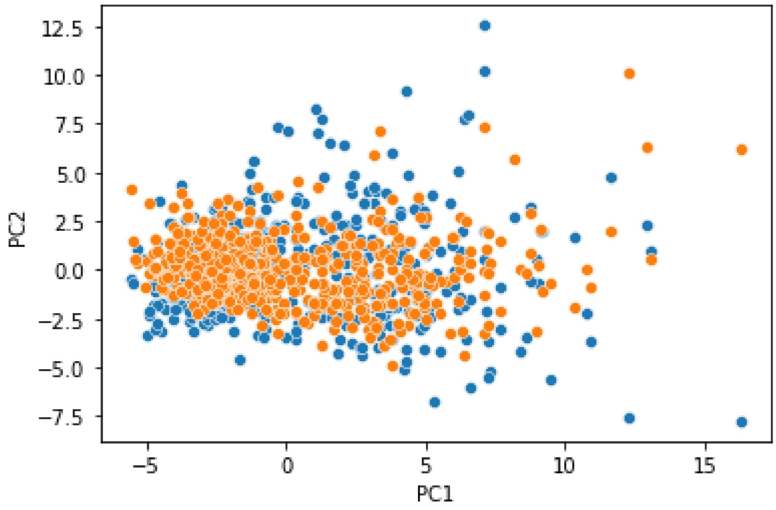
```
df2['PC1']= p1
df2['PC2']= p2
df2['PC3']= p3
```

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'>
```

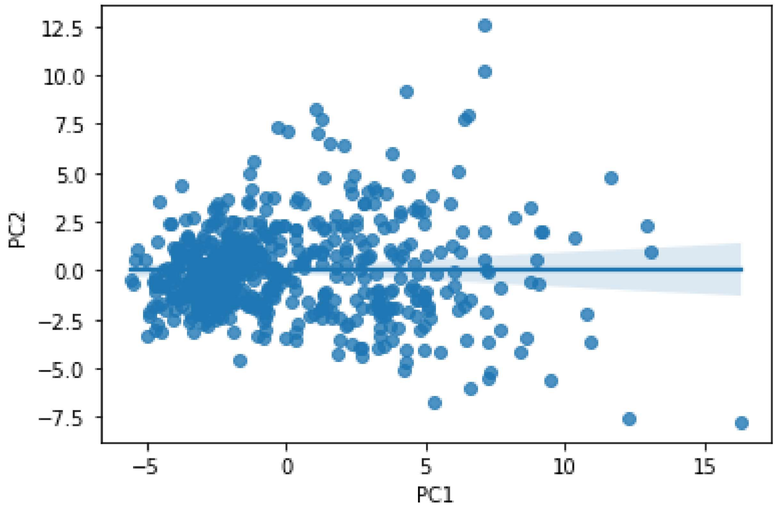


In [92]:

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

Out[92]:

```
<AxesSubplot: xlabel='PC1', ylabel='PC2'>
```

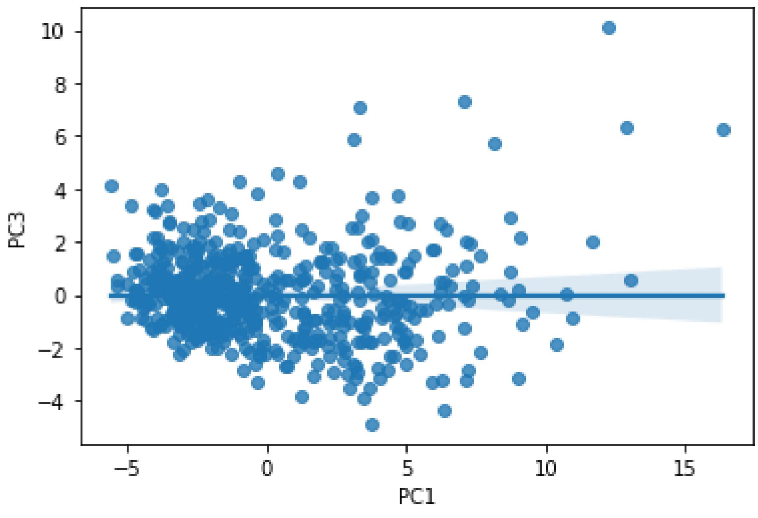


In [94]:

```
sns.regplot(x='PC1',y='PC3',data=df2)
```

Out[94]:

```
<AxesSubplot: xlabel='PC1', ylabel='PC3'>
```

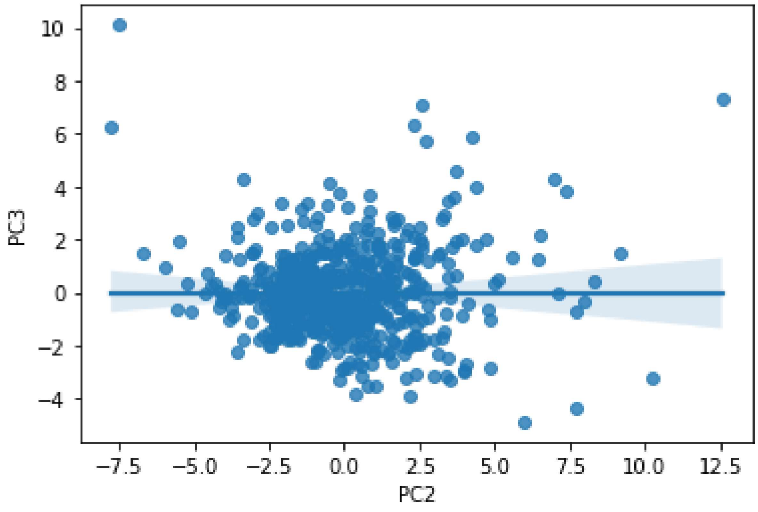


In [98]:

```
sns.regplot(x='PC2',y='PC3',data=df2)
```

Out[98]:

```
<AxesSubplot: xlabel='PC2', ylabel='PC3'>
```

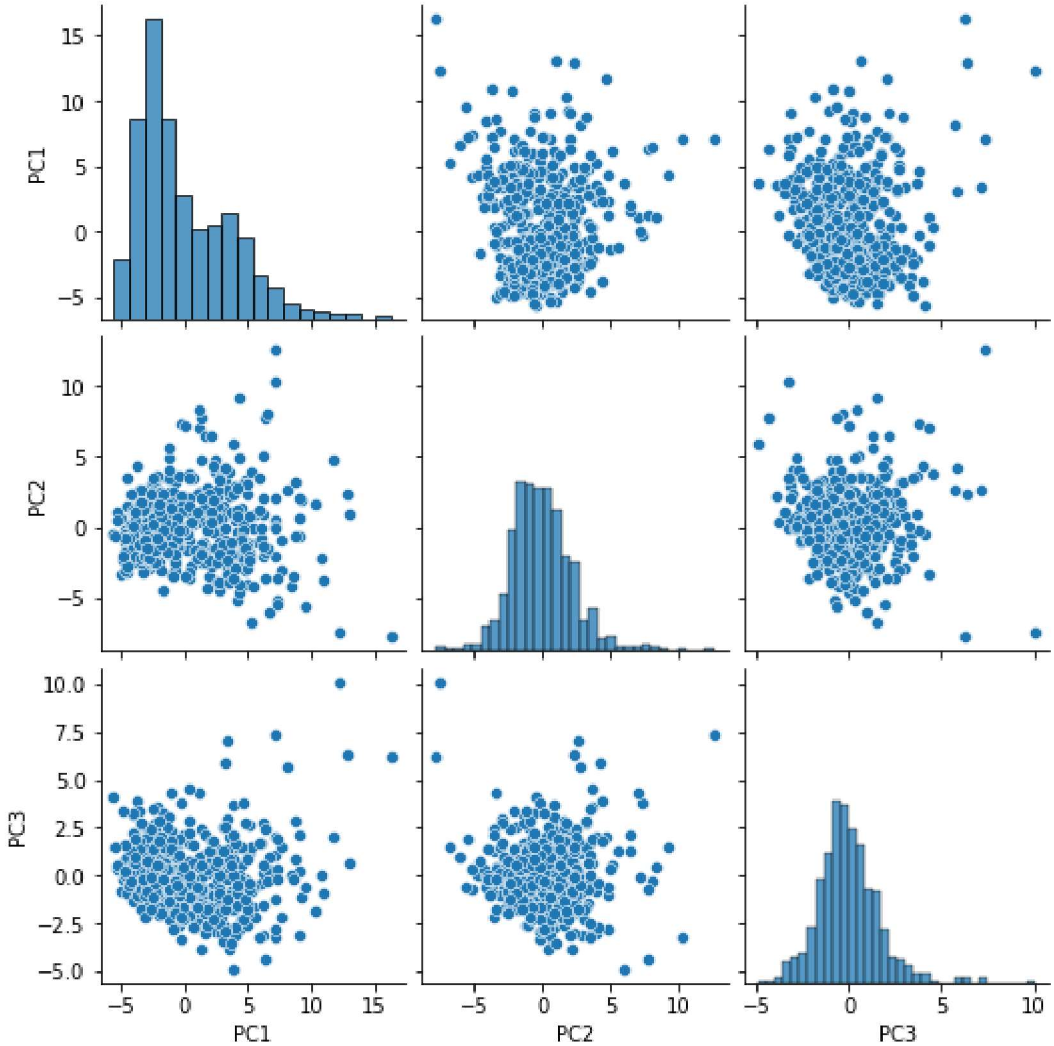


In [100]:

```
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]:

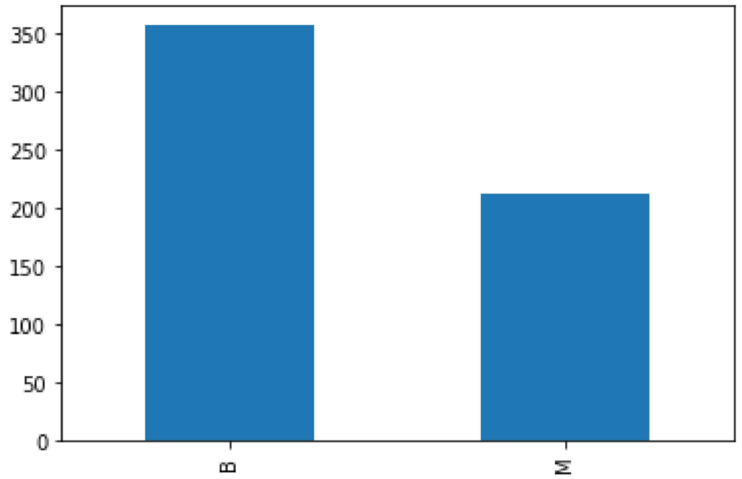
```
0    M
1    M
2    M
3    M
4    M
Name: diagnosis, dtype: object
```

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()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [67]:

```
y_pred=rfc.predict(x_test)
y_pred
```

Out[67]:

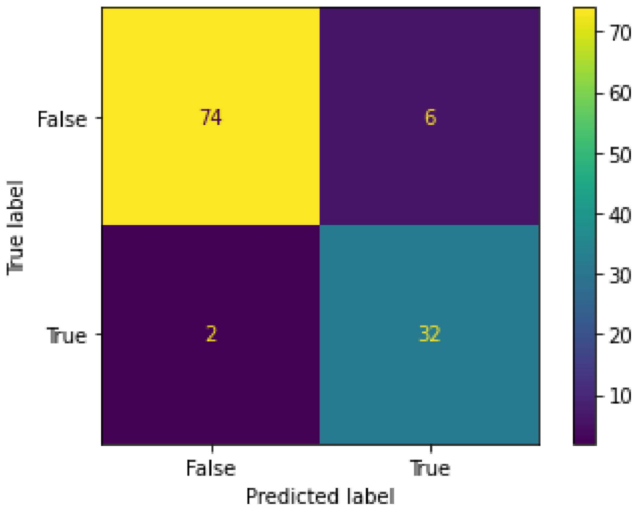
```
array(['M', 'B', 'B', 'B', 'M', 'M', 'B', 'B', 'M', 'B', 'B', 'B', 'B',
       'B', 'M', 'B', 'B', 'B', 'M', 'B', 'M', 'M', 'M', 'B', 'B', 'B',
       'M', 'B', 'M', 'B', 'B', 'M', 'M', 'B', 'B', 'M', 'M', 'B', 'B',
       'M', 'B', 'B', 'B', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
       'B', '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,display_labe
con_display.plot()
```

Out[74]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x15378a
032e0>
```



In [66]:

```
# checking the accuracy of the model
print(accuracy_score(y_test,y_pred))
```

```
0.9298245614035088
```

In [ ]: