In [119]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn import  metrics
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import KFold, LeaveOneOut, ShuffleSplit, StratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.naive_bayes import BernoulliNB
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import mean_squared_error, mean_absolute_error
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix,classification_report
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import DecisionTreeRegressor
from sklearn import tree
import warnings
warnings.filterwarnings('ignore') #ignoring warmings
```

In [2]:

```python
data = pd.read_csv("data.csv")
data
```

Out[2]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothnes
0	842302	M	17.99	10.38	122.80	1001.0	
1	842517	M	20.57	17.77	132.90	1326.0	
2	84300903	M	19.69	21.25	130.00	1203.0	
3	84348301	M	11.42	20.38	77.58	386.1	
4	84358402	M	20.29	14.34	135.10	1297.0	
...	...	...	...	...	...	...	
564	926424	M	21.56	22.39	142.00	1479.0	
565	926682	M	20.13	28.25	131.20	1261.0	
566	926954	M	16.60	28.08	108.30	858.1	
567	927241	M	20.60	29.33	140.10	1265.0	
568	92751	B	7.76	24.54	47.92	181.0	

569 rows × 33 columns

Data Cleaning

In [3]:

```python
data.head()
```

Out[3]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_
0	842302	M	17.99	10.38	122.80	1001.0	0.1
1	842517	M	20.57	17.77	132.90	1326.0	0.0
2	84300903	M	19.69	21.25	130.00	1203.0	0.1
3	84348301	M	11.42	20.38	77.58	386.1	0.1
4	84358402	M	20.29	14.34	135.10	1297.0	0.1

5 rows × 33 columns

In [4]:

```
data.head(10)
```

Out[4]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_
0	842302	M	17.99	10.38	122.80	1001.0	0.1
1	842517	M	20.57	17.77	132.90	1326.0	0.0
2	84300903	M	19.69	21.25	130.00	1203.0	0.1
3	84348301	M	11.42	20.38	77.58	386.1	0.1
4	84358402	M	20.29	14.34	135.10	1297.0	0.1
5	843786	M	12.45	15.70	82.57	477.1	0.1
6	844359	M	18.25	19.98	119.60	1040.0	0.0
7	84458202	M	13.71	20.83	90.20	577.9	0.1
8	844981	M	13.00	21.82	87.50	519.8	0.1
9	84501001	M	12.46	24.04	83.97	475.9	0.1

10 rows × 33 columns

In [5]:

```
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
```

In [6]:

```
data.shape
```

Out[6]:

```
(569, 33)
```

In [7]:

```
data.isnull()
```

Out[7]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_m
0	False	False	False	False	False	False	Fa
1	False	False	False	False	False	False	Fa
2	False	False	False	False	False	False	Fa
3	False	False	False	False	False	False	Fa
4	False	False	False	False	False	False	Fa
...	...	...	...	...	...	...	
564	False	False	False	False	False	False	Fa
565	False	False	False	False	False	False	Fa
566	False	False	False	False	False	False	Fa
567	False	False	False	False	False	False	Fa
568	False	False	False	False	False	False	Fa

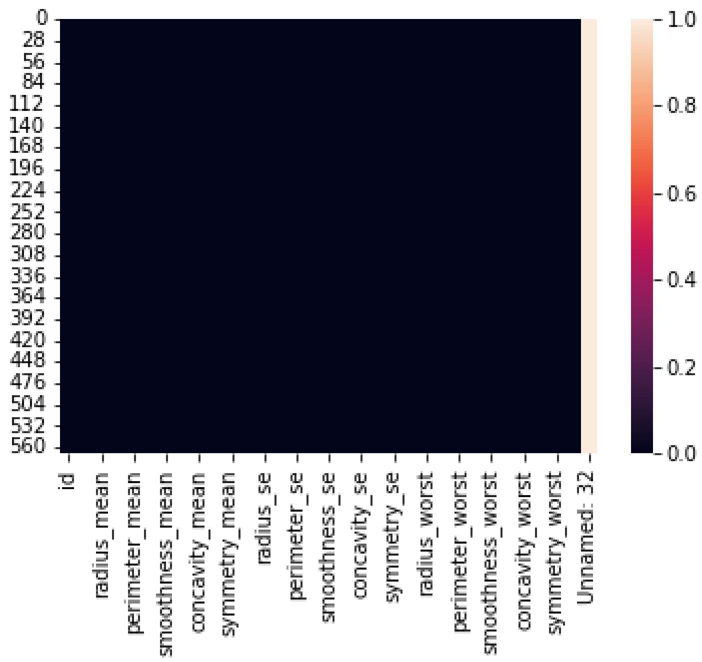
569 rows × 33 columns

In [8]:

```
sns.heatmap(data.isnull())
```

Out[8]:

<AxesSubplot:>

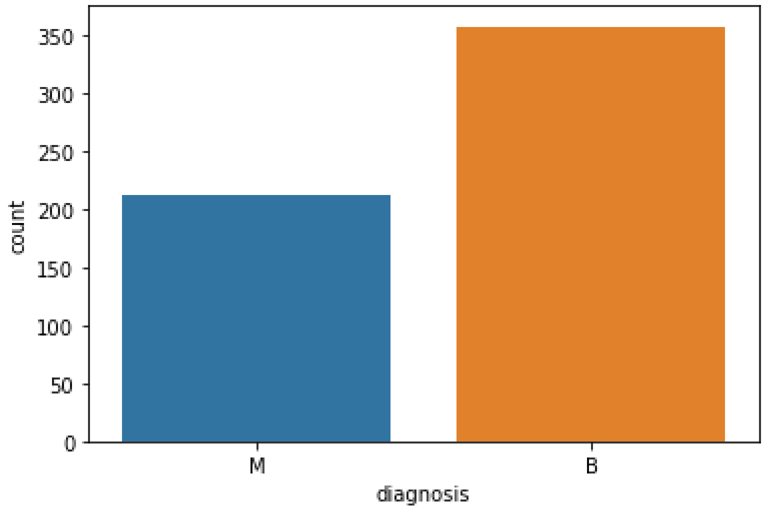


In [9]:

```
sns.countplot(x='diagnosis', data=data)
```

Out[9]:

```
&lt;AxesSubplot:xlabel='diagnosis', ylabel='count'&gt;
```



In [10]:

```
data.diagnosis.value_counts()
```

Out[10]:

```
B    357
M    212
Name: diagnosis, dtype: int64
```

In [11]:

```
data.dtypes
```

Out[11]:

```
id                           int64
diagnosis                   object
radius_mean                float64
texture_mean               float64
perimeter_mean             float64
area_mean                  float64
smoothness_mean            float64
compactness_mean           float64
concavity_mean             float64
concave points_mean        float64
symmetry_mean              float64
fractal_dimension_mean     float64
radius_se                  float64
texture_se                 float64
perimeter_se               float64
area_se                    float64
smoothness_se              float64
compactness_se             float64
concavity_se               float64
concave points_se          float64
symmetry_se                float64
fractal_dimension_se       float64
radius_worst               float64
texture_worst              float64
perimeter_worst            float64
area_worst                 float64
smoothness_worst           float64
compactness_worst          float64
concavity_worst            float64
concave points_worst       float64
symmetry_worst             float64
fractal_dimension_worst    float64
Unnamed: 32                float64
dtype: object
```

In [12]:

```
data['diagnosis'].unique()
```

Out[12]:

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

In [13]:

```
channel_map = {'M':0, 'B':1}
```

In [14]:

```
data['diagnosis'] = data['diagnosis'].map(channel_map)
```

In [15]:

```
data
```

Out[15]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothnes
0	842302	0	17.99	10.38	122.80	1001.0	
1	842517	0	20.57	17.77	132.90	1326.0	
2	84300903	0	19.69	21.25	130.00	1203.0	
3	84348301	0	11.42	20.38	77.58	386.1	
4	84358402	0	20.29	14.34	135.10	1297.0	
...	...	...	...	...	...	...	
564	926424	0	21.56	22.39	142.00	1479.0	
565	926682	0	20.13	28.25	131.20	1261.0	
566	926954	0	16.60	28.08	108.30	858.1	
567	927241	0	20.60	29.33	140.10	1265.0	
568	92751	1	7.76	24.54	47.92	181.0	

569 rows × 33 columns

In [16]:

```
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    int64  
 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(2)
memory usage: 146.8 KB
```

Data Visualization

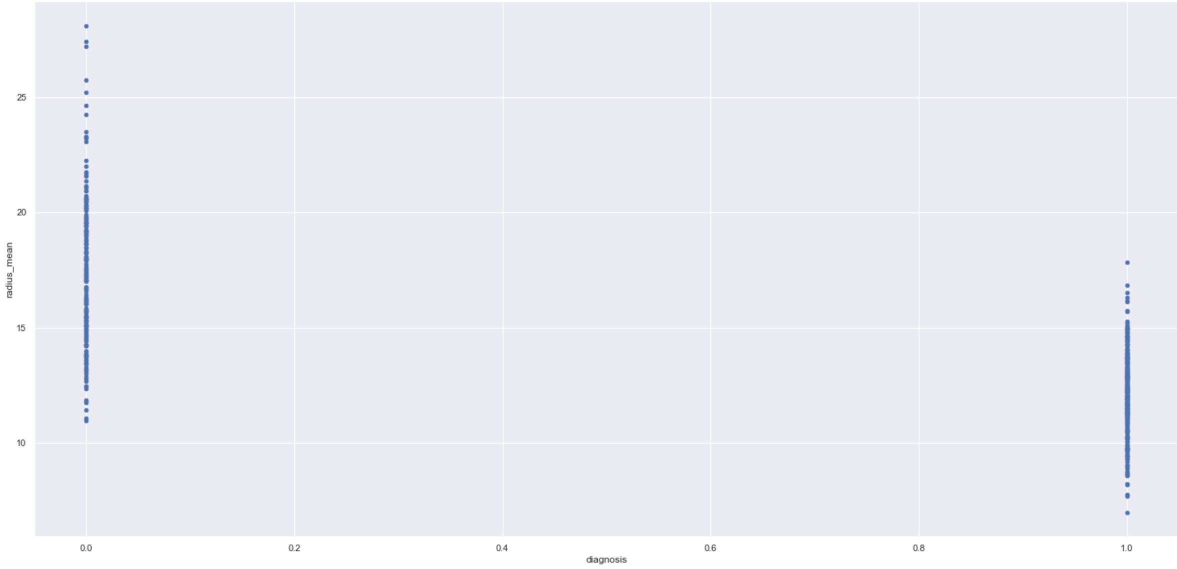
In [117]:

```
#plot the histograms for each feature:
data.hist(figsize = (30,30), color = 'pink')
plt.show()
```

In [121]:

```
#scatterplot
data.plot.scatter(x='diagnosis',y='radius_mean');
```

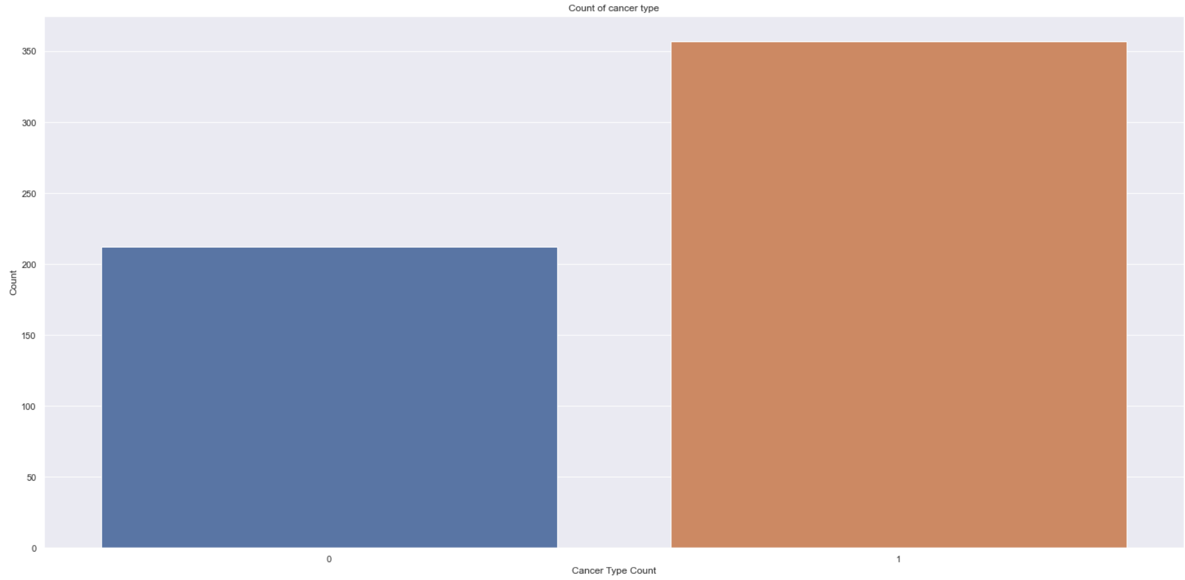
```
*c* argument looks like a single numeric RGB or RGBA sequence, which should
be avoided as value-mapping will have precedence in case its length matches
with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D arr
ay with a single row if you intend to specify the same RGB or RGBA value for
all points.
```



In [125]:

```
# Analyzing the target variable

plt.title('Count of cancer type')
sns.countplot(data['diagnosis'])
plt.xlabel('Cancer Type Count')
plt.ylabel('Count')
plt.show()
```



In [127]:

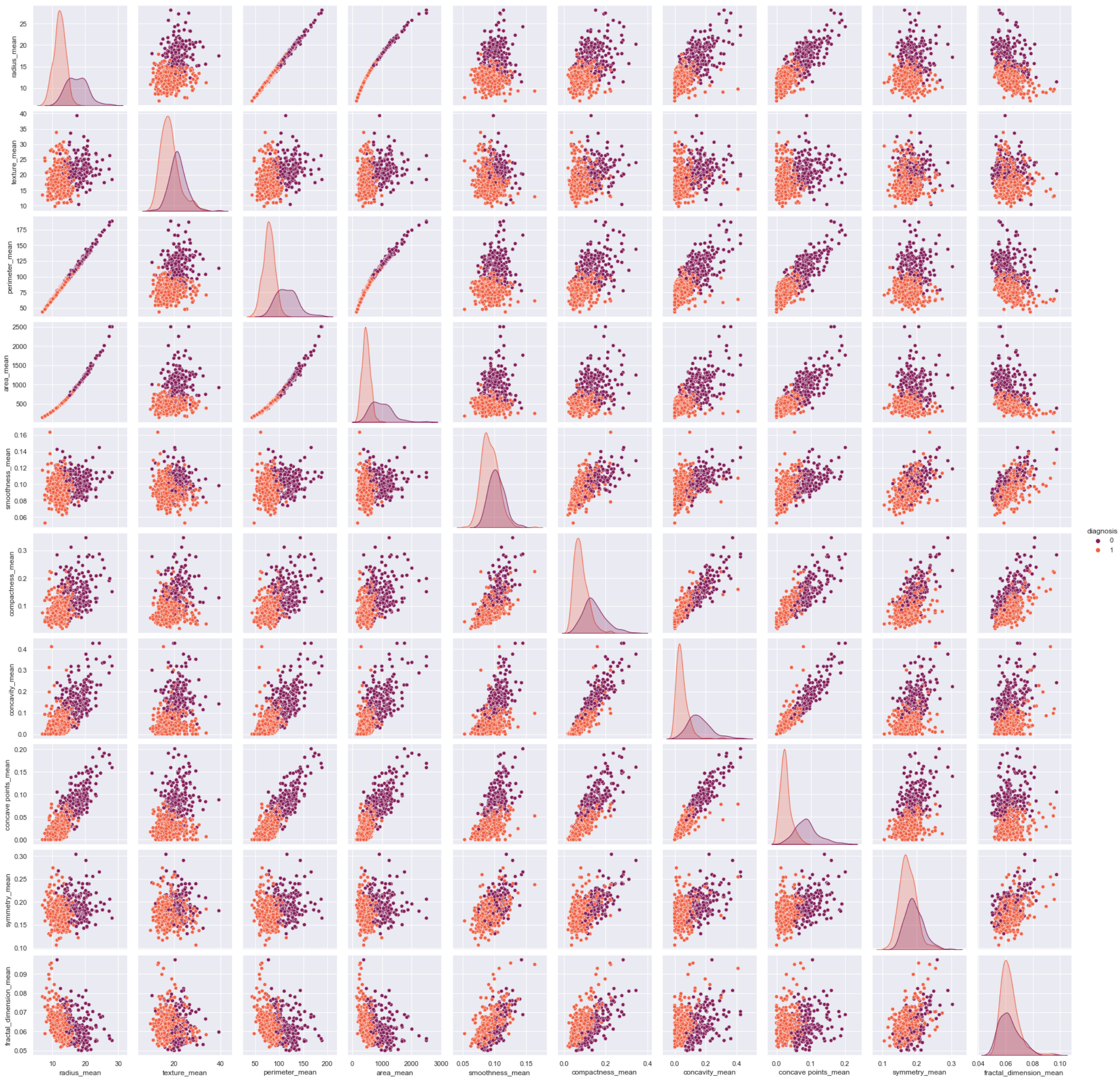
```
#generate a scatter plot with the following columns:

columns = ['diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean', 'smoo
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean']

sns.pairplot(data=data[columns], hue="diagnosis", palette='rocket')
```

Out[127]:

```
<seaborn.axisgrid.PairGrid at 0x19ebc9ac190>
```



In [122]:

```
cor = data.corr()
cor
```

Out[122]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	are
id	1.000000	-0.039769	0.074626	0.099770	0.073159	0
diagnosis	-0.039769	1.000000	-0.730029	-0.415185	-0.742636	-0
radius_mean	0.074626	-0.730029	1.000000	0.323782	0.997855	0
texture_mean	0.099770	-0.415185	0.323782	1.000000	0.329533	0
perimeter_mean	0.073159	-0.742636	0.997855	0.329533	1.000000	0
area_mean	0.096893	-0.708984	0.987357	0.321086	0.986507	1
smoothness_mean	-0.012968	-0.358560	0.170581	-0.023389	0.207278	0
compactness_mean	0.000096	-0.596534	0.506124	0.236702	0.556936	0
concavity_mean	0.050080	-0.696360	0.676764	0.302418	0.716136	0
concave points_mean	0.044158	-0.776614	0.822529	0.293464	0.850977	0
symmetry_mean	-0.022114	-0.330499	0.147741	0.071401	0.183027	0
fractal_dimension_mean	-0.052511	0.012838	-0.311631	-0.076437	-0.261477	-0
radius_se	0.143048	-0.567134	0.679090	0.275869	0.691765	0
texture_se	-0.007526	0.008303	-0.097317	0.386358	-0.086761	-0
perimeter_se	0.137331	-0.556141	0.674172	0.281673	0.693135	0
area_se	0.177742	-0.548236	0.735864	0.259845	0.744983	0
smoothness_se	0.096781	0.067016	-0.222600	0.006614	-0.202694	-0
compactness_se	0.033961	-0.292999	0.206000	0.191975	0.250744	0
concavity_se	0.055239	-0.253730	0.194204	0.143293	0.228082	0
concave points_se	0.078768	-0.408042	0.376169	0.163851	0.407217	0
symmetry_se	-0.017306	0.006522	-0.104321	0.009127	-0.081629	-0
fractal_dimension_se	0.025725	-0.077972	-0.042641	0.054458	-0.005523	-0
radius_worst	0.082405	-0.776454	0.969539	0.352573	0.969476	0
texture_worst	0.064720	-0.456903	0.297008	0.912045	0.303038	0
perimeter_worst	0.079986	-0.782914	0.965137	0.358040	0.970387	0
area_worst	0.107187	-0.733825	0.941082	0.343546	0.941550	0
smoothness_worst	0.010338	-0.421465	0.119616	0.077503	0.150549	0
compactness_worst	-0.002968	-0.590998	0.413463	0.277830	0.455774	0
concavity_worst	0.023203	-0.659610	0.526911	0.301025	0.563879	0
concave points_worst	0.035174	-0.793566	0.744214	0.295316	0.771241	0
symmetry_worst	-0.044224	-0.416294	0.163953	0.105008	0.189115	0
fractal_dimension_worst	-0.029866	-0.323872	0.007066	0.119205	0.051019	0
Unnamed: 32	NaN	NaN	NaN	NaN	NaN	

33 rows × 33 columns

In [124]:

```
cor.shape
```

Out[124]:

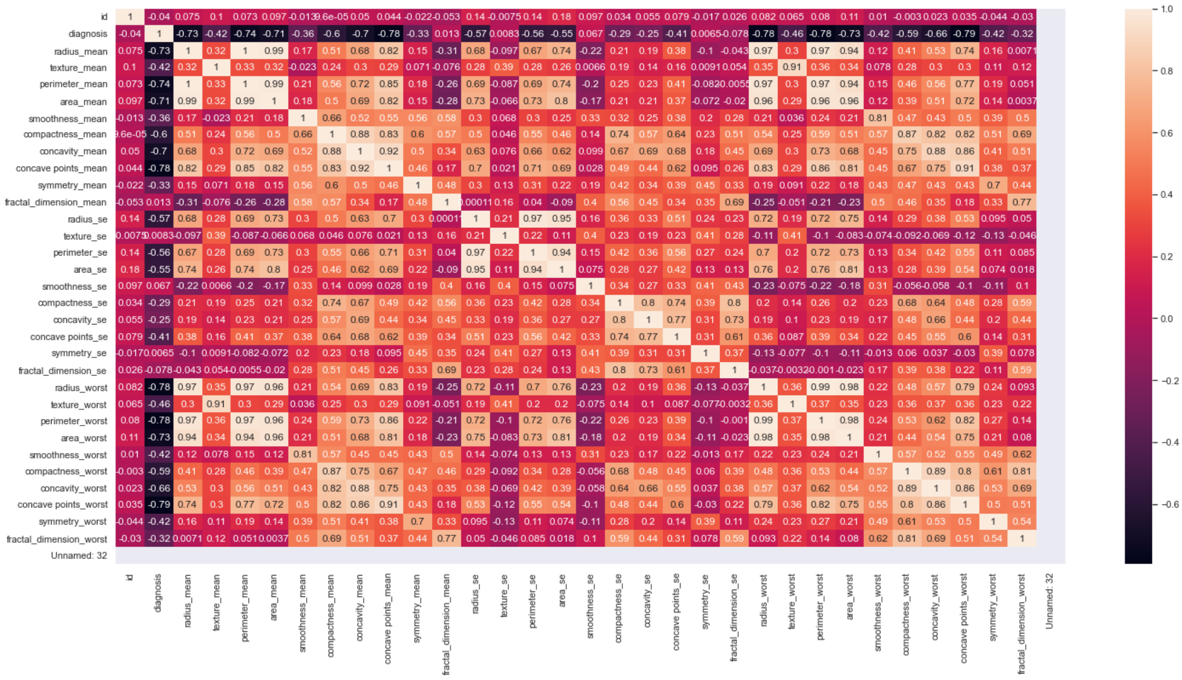
```
(33, 33)
```

In [19]:

```
sns.set(rc = {'figure.figsize':(25,12)})
sns.heatmap(cor,annot=True)
```

Out[19]:

```
<AxesSubplot:>
```



From above correlation we can see that 'concave points_worst' has the highest absolute correlation with diagnosis

In [20]:

```
srx = data['concave points_worst']
srx.shape
```

Out[20]:

```
(569,)
```

In [21]:

```
srxx = srx.values.reshape(-1,1)
srxx.shape
```

Out[21]:

```
(569, 1)
```

We will use 'concave points_worst' as feature for simple LR

In [22]:

```
y = data['diagnosis']
y.shape
```

Out[22]:

```
(569,)
```

In [23]:

```
ydt = y.values.reshape(-1,1)
ydt.shape
```

Out[23]:

```
(569, 1)
```

In [24]:

```
xtr, xts, ytr, yts = train_test_split(srxx,ydt, test_size = 0.1, random_state = 0)
print("Size of training set:", xtr.shape)
print("Size of testing set:", xts.shape)
```

```
Size of training set: (512, 1)
Size of testing set: (57, 1)
```

Linear Regression

In [25]:

```
LR=LinearRegression()
```

In [26]:

```
LR.fit(xtr,ytr)
```

Out[26]:

```
LinearRegression()
```

In [27]:

```
y_pred = LR.predict(xts)
```

In [28]:

```
acc = mean_squared_error(yts, y_pred)
acc
```

Out[28]:

```
0.09377714853706122
```

In [29]:

```
LR.score(xts,yts)
```

Out[29]:

```
0.604309148575439
```

In [30]:

```
df=pd.DataFrame({'Actual': yts.flatten(),'Predicted': y_pred.flatten()})
df
```

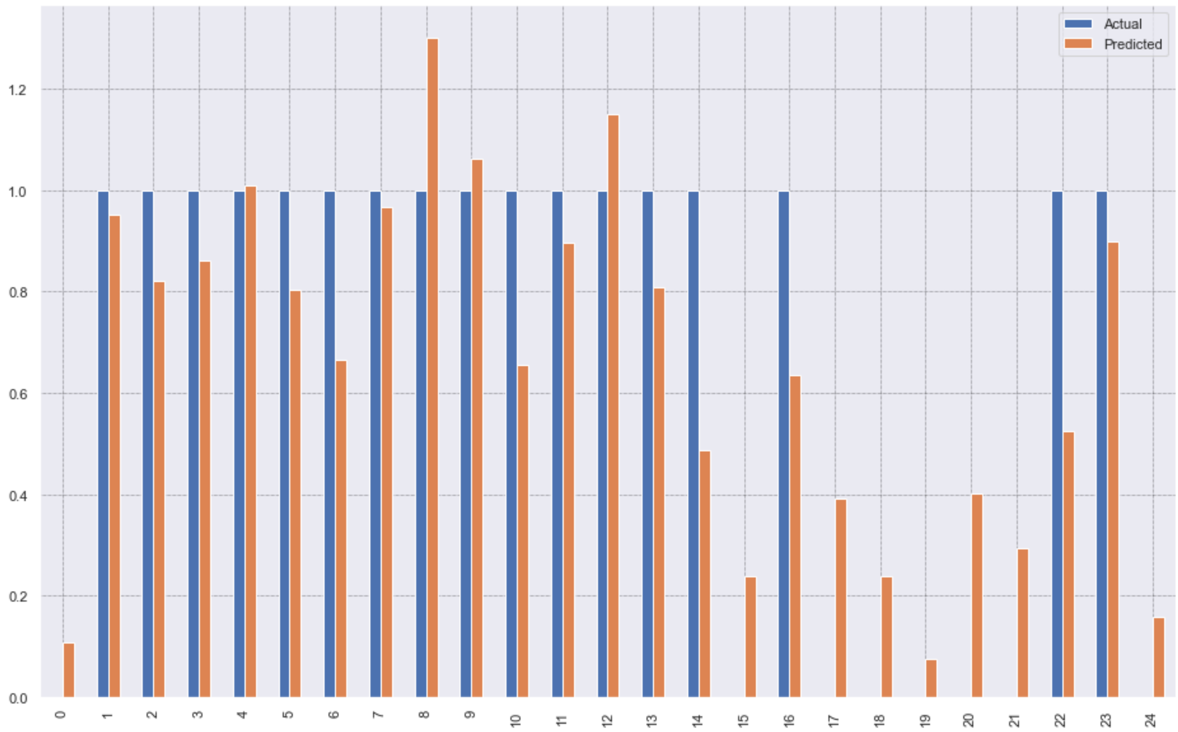
Out[30]:

	Actual	Predicted
0	0	0.107895
1	1	0.951039
2	1	0.822518
3	1	0.862394
4	1	1.008702
5	1	0.803569
6	1	0.665341
7	1	0.966385
8	1	1.300096
9	1	1.063284
10	1	0.656040
11	1	0.896341
12	1	1.151057
13	1	0.809730
14	1	0.488051
15	0	0.238102
16	1	0.634533
17	0	0.391559
18	0	0.239264
19	0	0.076506
20	0	0.401441
21	0	0.293323
22	1	0.524090
23	1	0.898259
24	0	0.157304
25	1	0.900410
26	1	1.070317
27	0	0.016635
28	1	0.893958
29	0	0.109639
30	1	1.068166
31	0	0.218920
32	1	0.613607
33	0	0.537460

	Actual	Predicted
34	1	1.300096
35	0	0.302624
36	1	0.737536
37	0	0.592100
38	1	0.836760
39	0	0.413066
40	0	0.917906
41	1	1.023292
42	0	0.720272
43	1	1.161694
44	1	0.511302
45	0	0.011985
46	1	1.300096
47	1	0.837225
48	1	1.049798
49	0	0.304949
50	0	0.272978
51	0	0.296229
52	0	0.498514
53	1	0.818333
54	1	0.856872
55	1	0.919476
56	1	1.179016

In [31]:

```
df1=df.head(25)
df1.plot(kind='bar',figsize=(16,10))
plt.grid(which='major',linestyle='-',linewidth='0.5',color='green')
plt.grid(which='major',linestyle=':',linewidth='0.5',color='black')
plt.show()
```



In [32]:

```
print('Mean Absolute Error:',mean_absolute_error(yts,y_pred))
print('Mean Squared Error:',mean_squared_error(yts,y_pred))
print('Root Mean Squared Error:',np.sqrt(mean_squared_error(yts,y_pred)))
```

```
Mean Absolute Error: 0.24369328283525543
Mean Squared Error: 0.09377714853706122
Root Mean Squared Error: 0.3062305480141738
```

Logistic Regression

In [33]:

```
LR=LogisticRegression()
```

In [34]:

```
LR.fit(xtr,ytr)
```

```
C:\Users\T2910\anaconda3\lib\site-packages\sklearn\utils\validation.py:993:
DataConversionWarning: A column-vector y was passed when a 1d array was expe
cted. Please change the shape of y to (n_samples, ), for example using ravel
().
  y = column_or_1d(y, warn=True)
```

Out[34]:

```
LogisticRegression()
```

In [35]:

```
y_pred = LR.predict(xts)
```

In [36]:

```
acc = mean_squared_error(yts, y_pred)
acc
```

Out[36]:

```
0.22807017543859648
```

In [37]:

```
LR.score(xts,yts)
```

Out[37]:

```
0.7719298245614035
```

In [38]:

```
df=pd.DataFrame({'Actual': yts.flatten(),'Predicted': y_pred.flatten()})
df
```

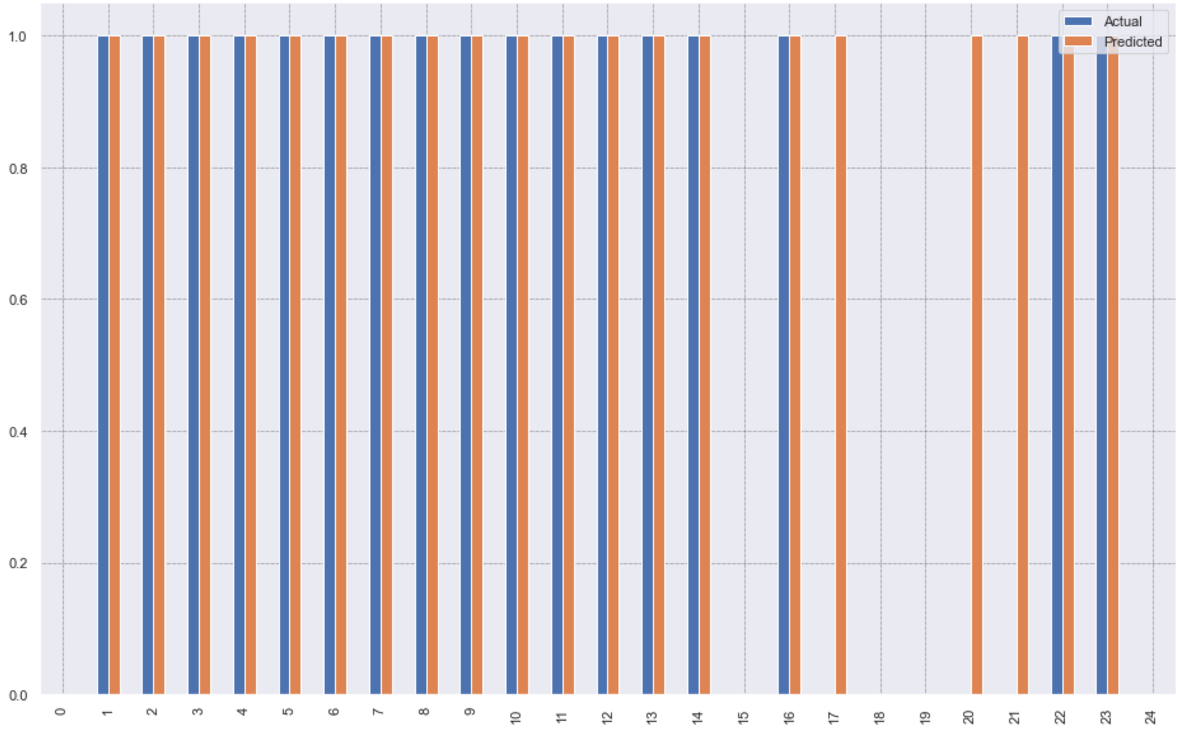
Out[38]:

	Actual	Predicted
0	0	0
1	1	1
2	1	1
3	1	1
4	1	1
5	1	1
6	1	1
7	1	1
8	1	1
9	1	1
10	1	1
11	1	1
12	1	1
13	1	1
14	1	1
15	0	0
16	1	1
17	0	1
18	0	0
19	0	0
20	0	1
21	0	1
22	1	1
23	1	1
24	0	0
25	1	1
26	1	1
27	0	0
28	1	1
29	0	0
30	1	1
31	0	0
32	1	1
33	0	1

	Actual	Predicted
34	1	1
35	0	1
36	1	1
37	0	1
38	1	1
39	0	1
40	0	1
41	1	1
42	0	1
43	1	1
44	1	1
45	0	0
46	1	1
47	1	1
48	1	1
49	0	1
50	0	1
51	0	1
52	0	1
53	1	1
54	1	1
55	1	1
56	1	1

In [39]:

```
df1=df.head(25)
df1.plot(kind='bar',figsize=(16,10))
plt.grid(which='major',linestyle='-',linewidth='0.5',color='green')
plt.grid(which='major',linestyle=':',linewidth='0.5',color='black')
plt.show()
```



In [40]:

```
print('Mean Absolute Error:',mean_absolute_error(yts,y_pred))
print('Mean Squared Error:',mean_squared_error(yts,y_pred))
print('Root Mean Squared Error:',np.sqrt(mean_squared_error(yts,y_pred)))
```

```
Mean Absolute Error: 0.22807017543859648
Mean Squared Error: 0.22807017543859648
Root Mean Squared Error: 0.4775669329409193
```

Multiclass Linear Regression

In [41]:

```
myx = data.drop(['id','Unnamed: 32','diagnosis'],axis=1)
```

In [42]:

```
myx
```

Out[42]:

	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 [43]:

```
myy = data['diagnosis']
```

In [44]:

```
myy
```

Out[44]:

```
0      0
1      0
2      0
3      0
4      0
      ..
564    0
565    0
566    0
567    0
568    1
Name: diagnosis, Length: 569, dtype: int64
```

In [45]:

```
myx.shape
```

Out[45]:

```
(569, 30)
```

In [46]:

```
myy.shape
```

Out[46]:

```
(569,)
```

In [47]:

```
myyy = myy.values.reshape(-1,1)
myyy
```

Out[47]:

```
array([[0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
```

In [48]:

```
myyy.shape
```

Out[48]:

```
(569, 1)
```

In [49]:

```
xtr, xts, ytr, yts = train_test_split(myx,myyy, test_size = 0.3, random_state = 0)
print("Size of training set:", xtr.shape)
print("Size of training set:", xts.shape)
```

```
Size of training set: (398, 30)
Size of training set: (171, 30)
```

In [50]:

```
LR=LinearRegression()
```

In [51]:

```
LR.fit(xtr,ytr)
```

Out[51]:

```
LinearRegression()
```

In [52]:

```
y_pred = LR.predict(xts)
```

In [53]:

```
acc = mean_squared_error(yts, y_pred)
acc
```

Out[53]:

```
0.062384962436604595
```

In [54]:

```
LR.score(xts, yts)
```

Out[54]:

```
0.7318931971474494
```

In [55]:

```
df=pd.DataFrame({'Actual': yts.flatten(),'Predicted': y_pred.flatten()})
df
```

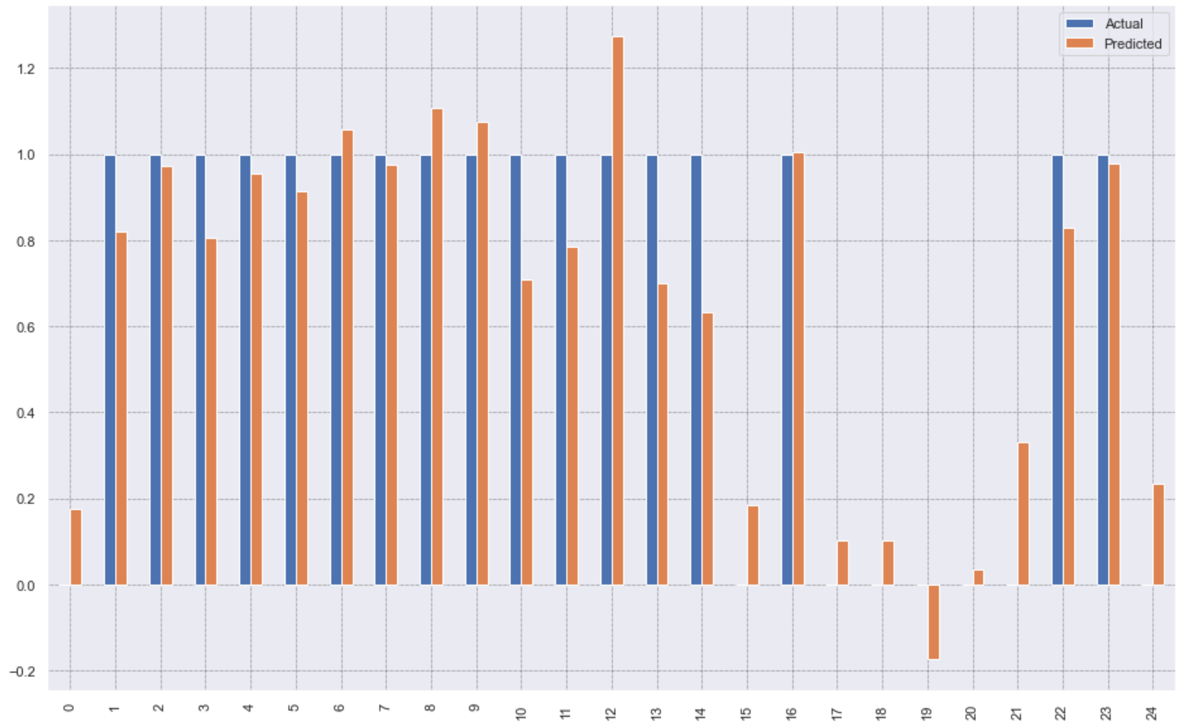
Out[55]:

	Actual	Predicted
0	0	0.177074
1	1	0.819875
2	1	0.973552
3	1	0.806176
4	1	0.956128
...	...	...
166	0	0.347625
167	0	0.188163
168	1	0.704667
169	1	0.902653
170	1	1.126109

171 rows × 2 columns

In [56]:

```
df1=df.head(25)
df1.plot(kind='bar',figsize=(16,10))
plt.grid(which='major',linestyle='-',linewidth='0.5',color='green')
plt.grid(which='major',linestyle=':',linewidth='0.5',color='black')
plt.show()
```



In [57]:

```
print('Mean Absolute Error:',mean_absolute_error(yts,y_pred))
print('Mean Squared Error:',mean_squared_error(yts,y_pred))
print('Root Mean Squared Error:',np.sqrt(mean_squared_error(yts,y_pred)))
```

```
Mean Absolute Error: 0.1960125785952852
Mean Squared Error: 0.062384962436604595
Root Mean Squared Error: 0.24976981890653763
```

Multiclass Logistic Regression

In [58]:

```
LR=LogisticRegression()
```

In [59]:

```
LR.fit(xtr,ytr)
```

```
C:\Users\T2910\anaconda3\lib\site-packages\sklearn\utils\validation.py:993:
DataConversionWarning: A column-vector y was passed when a 1d array was expe
cted. Please change the shape of y to (n_samples, ), for example using ravel
().
  y = column_or_1d(y, warn=True)
C:\Users\T2910\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.p
y:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scik
it-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regre
ssion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression)
  n_iter_i = _check_optimize_result(
```

Out[59]:

```
LogisticRegression()
```

In [60]:

```
y_pred = LR.predict(xts)
```

In [61]:

```
acc = mean_squared_error(yts, y_pred)
acc
```

Out[61]:

```
0.04678362573099415
```

In [62]:

```
LR.score(xts, yts)
```

Out[62]:

```
0.9532163742690059
```

In [63]:

```
df=pd.DataFrame({'Actual': yts.flatten(),'Predicted': y_pred.flatten()})
df
```

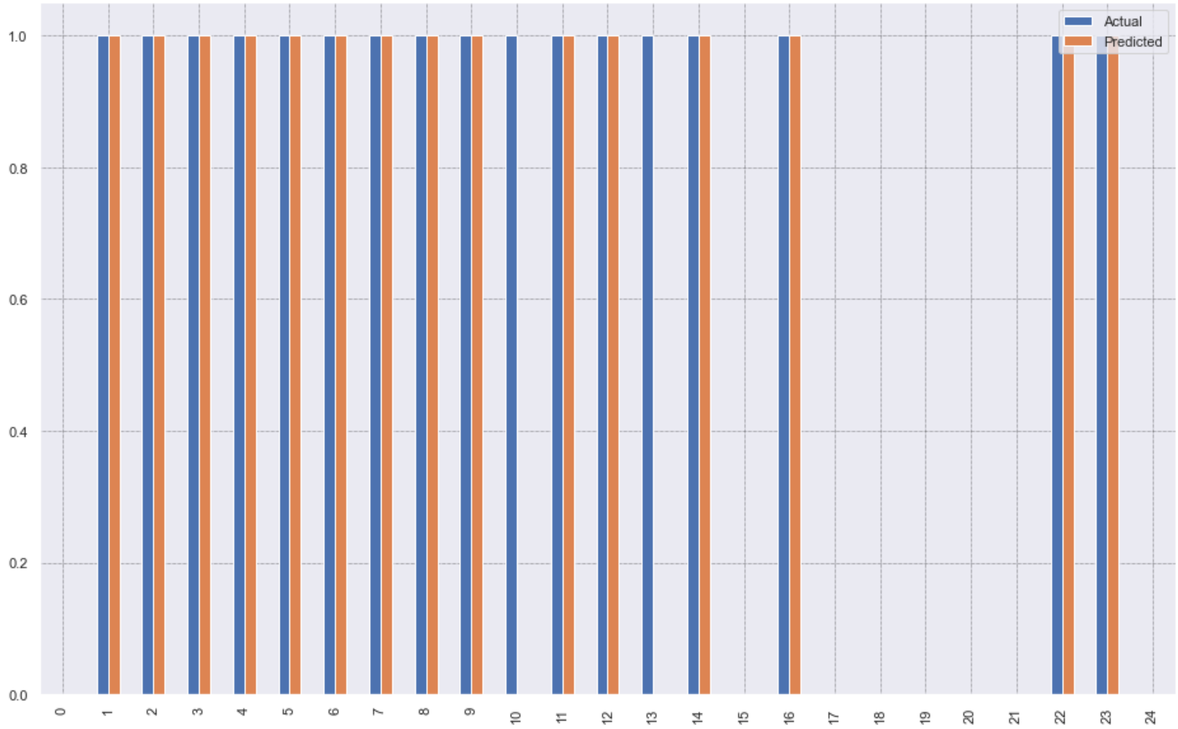
Out[63]:

	Actual	Predicted
0	0	0
1	1	1
2	1	1
3	1	1
4	1	1
...	...	...
166	0	0
167	0	0
168	1	1
169	1	1
170	1	1

171 rows × 2 columns

In [64]:

```
df1=df.head(25)
df1.plot(kind='bar',figsize=(16,10))
plt.grid(which='major',linestyle='-',linewidth='0.5',color='green')
plt.grid(which='major',linestyle=':',linewidth='0.5',color='black')
plt.show()
```



In [65]:

```
print('Mean Absolute Error:',mean_absolute_error(yts,y_pred))
print('Mean Squared Error:',mean_squared_error(yts,y_pred))
print('Root Mean Squared Error:',np.sqrt(mean_squared_error(yts,y_pred)))
```

```
Mean Absolute Error: 0.04678362573099415
Mean Squared Error: 0.04678362573099415
Root Mean Squared Error: 0.21629522817435004
```

Naive-Bayes

In [66]:

```
nb = BernoulliNB()
gnb = GaussianNB()
mnb = MultinomialNB()
```

In [67]:

```
nb.fit(xtr,ytr)
gnb.fit(xtr,ytr)
mnb.fit(xtr,ytr)
```

```
C:\Users\T2910\anaconda3\lib\site-packages\sklearn\utils\validation.py:993:
DataConversionWarning: A column-vector y was passed when a 1d array was expe
cted. Please change the shape of y to (n_samples, ), for example using ravel
().
  y = column_or_1d(y, warn=True)
C:\Users\T2910\anaconda3\lib\site-packages\sklearn\utils\validation.py:993:
DataConversionWarning: A column-vector y was passed when a 1d array was expe
cted. Please change the shape of y to (n_samples, ), for example using ravel
().
  y = column_or_1d(y, warn=True)
C:\Users\T2910\anaconda3\lib\site-packages\sklearn\utils\validation.py:993:
DataConversionWarning: A column-vector y was passed when a 1d array was expe
cted. Please change the shape of y to (n_samples, ), for example using ravel
().
  y = column_or_1d(y, warn=True)
```

Out[67]:

```
MultinomialNB()
```

BernoulliNB

In [68]:

```
ypred = nb.predict(xts)
```

In [69]:

```
accuracy_score(yts,ypred)
```

Out[69]:

```
0.631578947368421
```

In [70]:

```
print(classification_report(yts,ypred))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        63
           1       0.63      1.00      0.77       108

    accuracy                           0.63       171
   macro avg       0.32      0.50      0.39       171
weighted avg       0.40      0.63      0.49       171
```

```
C:\Users\T2910\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\T2910\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\T2910\anaconda3\lib\site-packages\sklearn\metrics\_classification.p
y:1318: UndefinedMetricWarning: Precision and F-score are ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` para
meter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [71]:

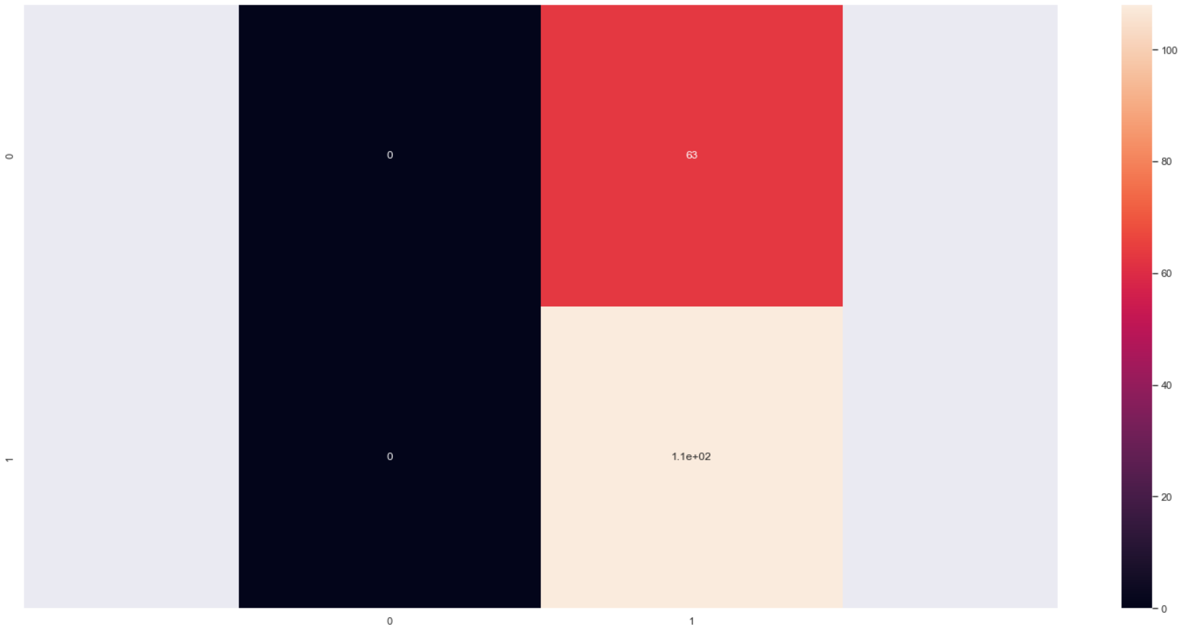
```
cf = confusion_matrix(yts,ypred)
cf
```

Out[71]:

```
array([[  0,  63],
       [  0, 108]], dtype=int64)
```

In [72]:

```
sns.heatmap (cf,annot=True)
plt.axis('equal')
plt.show()
```



In [73]:

```
nb.score(xts,yts)
```

Out[73]:

```
0.631578947368421
```

In [74]:

```
df=pd.DataFrame({'Actual': yts.flatten(),'Predicted': y_pred.flatten()})
df
```

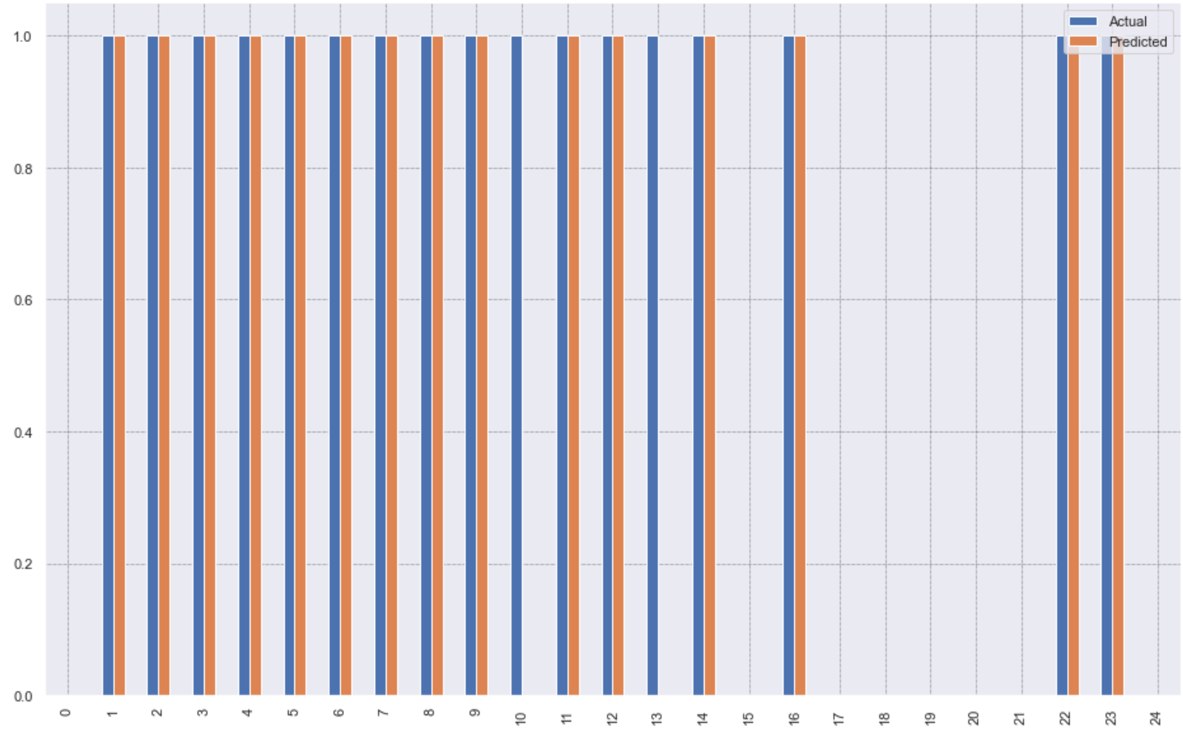
Out[74]:

	Actual	Predicted
0	0	0
1	1	1
2	1	1
3	1	1
4	1	1
...	...	...
166	0	0
167	0	0
168	1	1
169	1	1
170	1	1

171 rows × 2 columns

In [75]:

```
df1=df.head(25)
df1.plot(kind='bar',figsize=(16,10))
plt.grid(which='major',linestyle='-',linewidth='0.5',color='green')
plt.grid(which='major',linestyle=':',linewidth='0.5',color='black')
plt.show()
```



GaussianNB

In [76]:

```
ypred = gnb.predict(xts)
```

In [77]:

```
accuracy_score(yts,ypred)
```

Out[77]:

```
0.9239766081871345
```

In [78]:

```
print(classification_report(yts,ypred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.90      0.90        63
           1       0.94      0.94      0.94       108

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

In [79]:

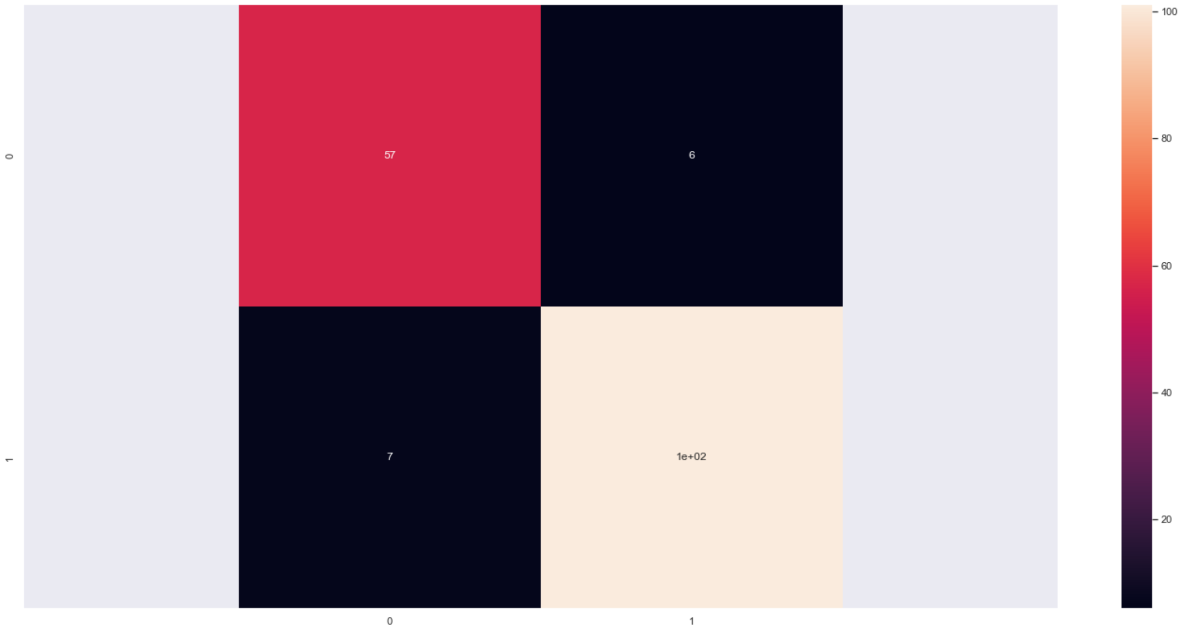
```
cf = confusion_matrix(yts,ypred)
cf
```

Out[79]:

```
array([[ 57,   6],
       [  7, 101]], dtype=int64)
```

In [80]:

```
sns.heatmap (cf,annot=True)
plt.axis('equal')
plt.show()
```



In [81]:

```
gnb.score(xts,yts)
```

Out[81]:

0.9239766081871345

In [82]:

```
df=pd.DataFrame({'Actual': yts.flatten(),'Predicted': y_pred.flatten()})
df
```

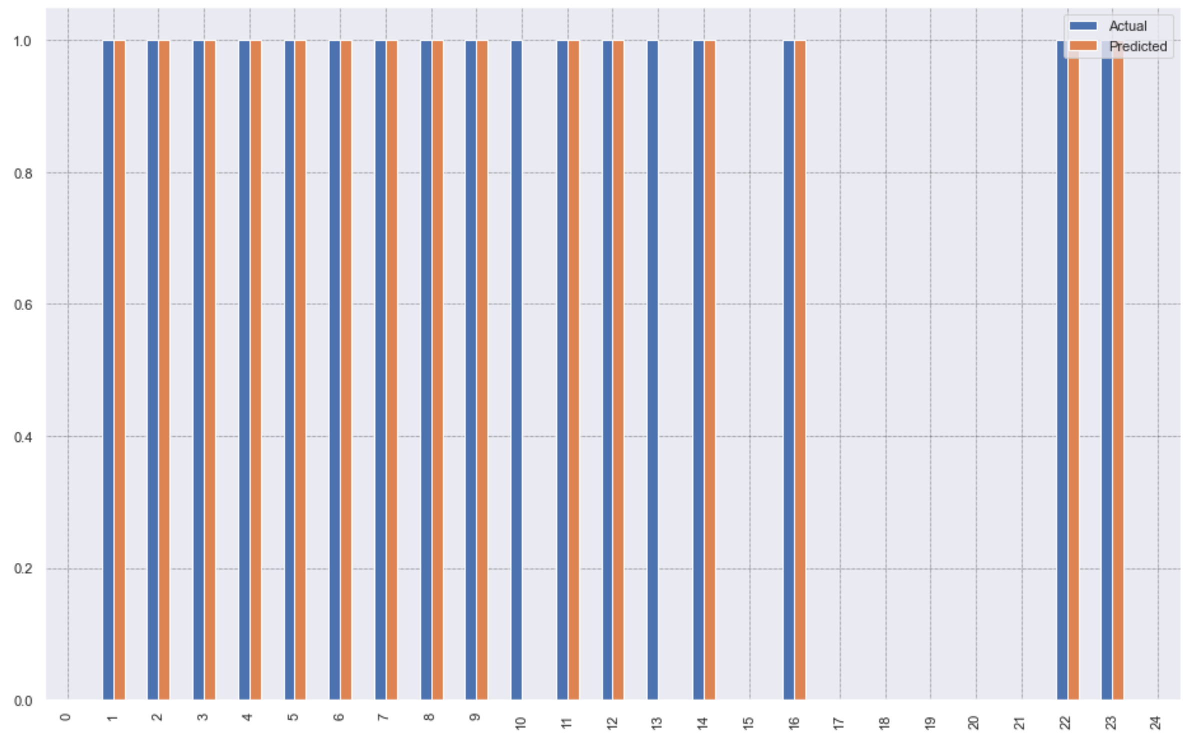
Out[82]:

	Actual	Predicted
0	0	0
1	1	1
2	1	1
3	1	1
4	1	1
...	...	...
166	0	0
167	0	0
168	1	1
169	1	1
170	1	1

171 rows × 2 columns

In [83]:

```
df1=df.head(25)
df1.plot(kind='bar',figsize=(16,10))
plt.grid(which='major',linestyle='-',linewidth='0.5',color='green')
plt.grid(which='major',linestyle=':',linewidth='0.5',color='black')
plt.show()
```



MultinomialNB

In [84]:

```
ypred = mnb.predict(xts)
```

In [85]:

```
accuracy_score(yts,ypred)
```

Out[85]:

```
0.9005847953216374
```

In [86]:

```
print(classification_report(yts,ypred))
```

```
              precision    recall  f1-score   support

           0       0.96      0.76      0.85        63
           1       0.88      0.98      0.93       108

    accuracy                           0.90       171
   macro avg       0.92      0.87      0.89       171
weighted avg       0.91      0.90      0.90       171
```

In [87]:

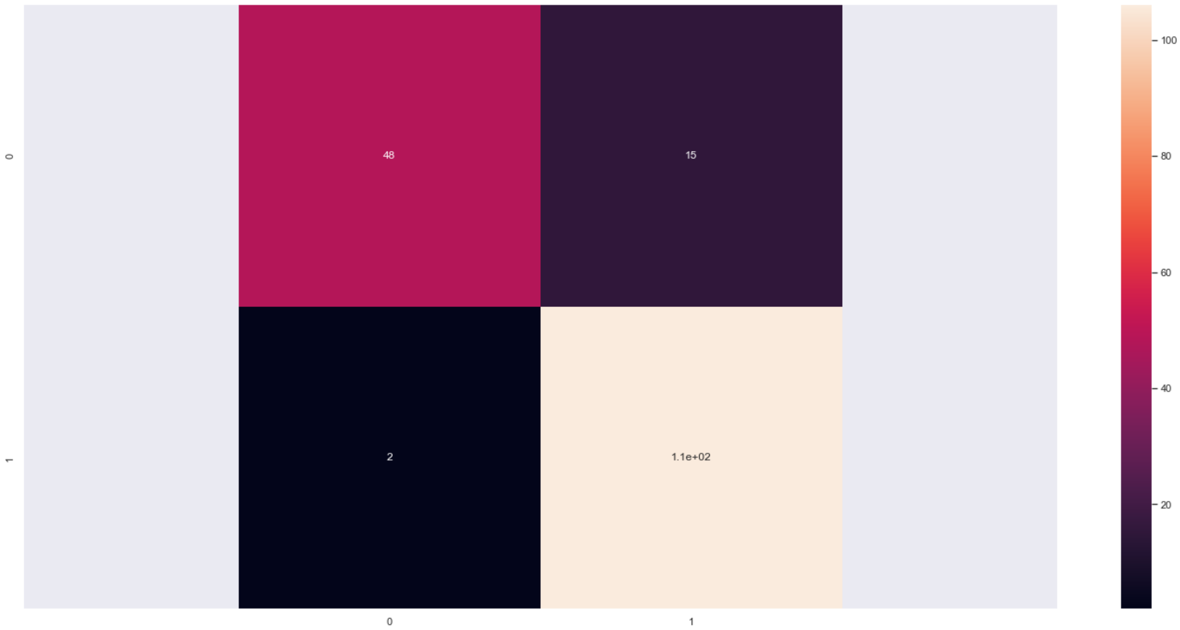
```
cf = confusion_matrix(yts,ypred)
cf
```

Out[87]:

```
array([[ 48,  15],
       [  2, 106]], dtype=int64)
```

In [88]:

```
sns.heatmap (cf,annot=True)
plt.axis('equal')
plt.show()
```



In [89]:

```
mnb.score(xts,yts)
```

Out[89]:

```
0.9005847953216374
```

In [90]:

```
df=pd.DataFrame({'Actual': yts.flatten(),'Predicted': y_pred.flatten()})
df
```

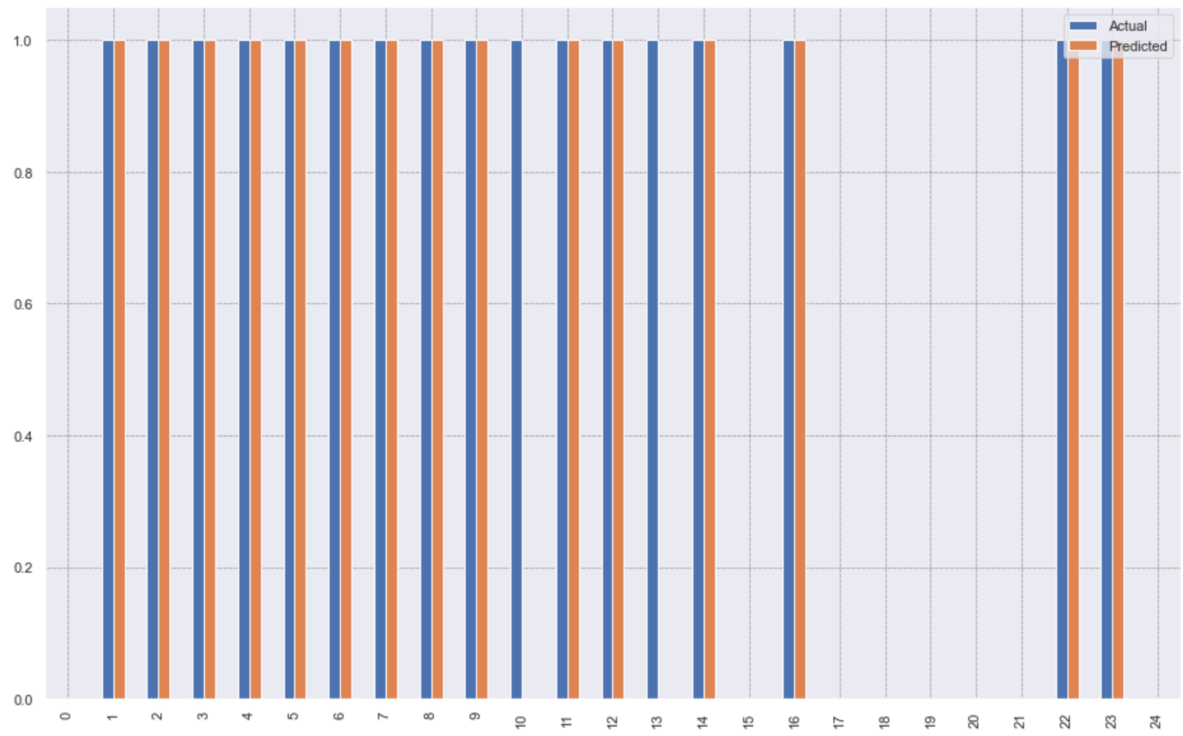
Out[90]:

	Actual	Predicted
0	0	0
1	1	1
2	1	1
3	1	1
4	1	1
...	...	...
166	0	0
167	0	0
168	1	1
169	1	1
170	1	1

171 rows × 2 columns

In [91]:

```
df1=df.head(25)
df1.plot(kind='bar',figsize=(16,10))
plt.grid(which='major',linestyle='-',linewidth='0.5',color='green')
plt.grid(which='major',linestyle=':',linewidth='0.5',color='black')
plt.show()
```



DT Regressor

In [92]:

```
myx.shape
```

Out[92]:

```
(569, 30)
```

In [93]:

```
myy.shape
```

Out[93]:

```
(569,)
```

In [94]:

```
feat = myx.values
classes = myy.values
```

In [95]:

```
(train_feat, test_feat, train_classes, test_classes) = train_test_split(feat, classes, rand
m = DecisionTreeRegressor().fit(train_feat, train_classes)
```

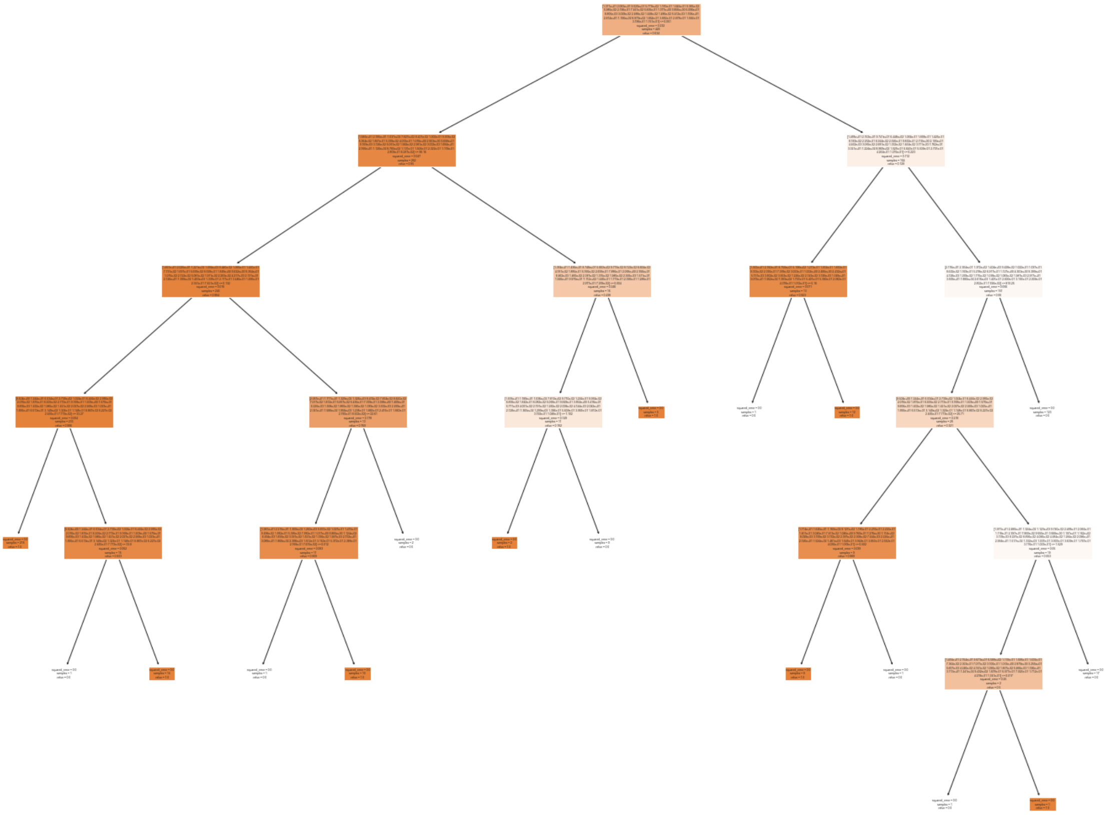
In [96]:

```
ypred = m.predict(test_feat)
print("MSE:",metrics.mean_squared_error(test_classes, ypred))
```

```
MSE: 0.06293706293706294
```

In [97]:

```
fig = plt.figure(figsize=(25, 20))
_= tree.plot_tree(m,
               feature_names = feat,
                  class_names = classes,
               filled = True)
```



In [98]:

```
text_representation = tree.export_text(m)
print(text_representation)
```

```
|--- feature_7 <= 0.05
|   |--- feature_13 <= 38.16
|   |   |--- feature_27 <= 0.13
|   |   |   |--- feature_21 <= 33.27
|   |   |   |   |--- value: [1.00]
|   |   |   |--- feature_21 >  33.27
|   |   |   |   |--- feature_21 <= 33.80
|   |   |   |   |   |--- value: [0.00]
|   |   |   |   |--- feature_21 >  33.80
|   |   |   |   |   |--- value: [1.00]
|   |   |--- feature_27 >  0.13
|   |   |   |--- feature_1 <= 22.61
|   |   |   |   |--- feature_18 <= 0.01
|   |   |   |   |   |--- value: [0.00]
|   |   |   |   |--- feature_18 >  0.01
|   |   |   |   |   |--- value: [1.00]
|   |   |   |--- feature_1 >  22.61
|   |   |   |   |--- value: [0.00]
|   |--- feature_13 >  38.16
|   |   |--- feature_19 <= 0.00
|   |   |   |--- feature_11 <= 1.10
|   |   |   |   |--- value: [1.00]
|   |   |   |--- feature_11 >  1.10
|   |   |   |   |--- value: [0.00]
|   |   |--- feature_19 >  0.00
|   |   |   |--- value: [1.00]
|--- feature_7 >  0.05
|   |--- feature_26 <= 0.22
|   |   |--- feature_8 <= 0.16
|   |   |   |--- value: [0.00]
|   |   |--- feature_8 >  0.16
|   |   |   |--- value: [1.00]
|   |--- feature_26 >  0.22
|   |   |--- feature_23 <= 810.25
|   |   |   |--- feature_21 <= 25.71
|   |   |   |   |--- feature_25 <= 0.50
|   |   |   |   |   |--- value: [1.00]
|   |   |   |   |--- feature_25 >  0.50
|   |   |   |   |   |--- value: [0.00]
|   |   |   |--- feature_21 >  25.71
|   |   |   |   |--- feature_12 <= 1.53
|   |   |   |   |   |--- feature_15 <= 0.02
|   |   |   |   |   |   |--- value: [0.00]
|   |   |   |   |   |--- feature_15 >  0.02
|   |   |   |   |   |   |--- value: [1.00]
|   |   |   |   |--- feature_12 >  1.53
|   |   |   |   |   |--- value: [0.00]
|   |   |--- feature_23 >  810.25
|   |   |   |--- value: [0.00]
```

In [99]:

```
m.score(test_feat,test_classes)
```

Out[99]:

```
0.7358374384236454
```

Cross Validation

In [100]:

```
# k-fold cross validation technique
kf = KFold(n_splits=5, random_state=1,shuffle=True)
# stratified kfold cross validation technique
skf = StratifiedKFold(n_splits=5)
# LeaveOneOut cross validation technique
loocv = LeaveOneOut()
# shuffle split cross validation technique
shvc = ShuffleSplit()
```

In [101]:

```
# evaluating the data sets with kfold and DecisionTreeClassifier
dst = DecisionTreeClassifier()
scores = cross_val_score(dst,feat,classes,scoring='accuracy',cv=kf)
print('Accuracy using Decision Tree: %2f%%'%(scores.mean()*100))
print(scores)
```

```
Accuracy using Decision Tree: 93.848781%
[0.95614035 0.92982456 0.89473684 0.97368421 0.9380531 ]
```

In [102]:

```
# evaluating the data sets with kfold and Naive Bayes Classifier
nb = GaussianNB()
scores = cross_val_score(nb,feat,classes,scoring='accuracy',cv=kf)
print('Accuracy using GaussianNB: %2f%%'%(scores.mean()*100))
print(scores)
```

```
Accuracy using GaussianNB: 93.851886%
[0.94736842 0.93859649 0.9122807  0.93859649 0.95575221]
```

In [103]:

```
# evaluating the data sets with kfold and SVM
svm = SVC()
scores = cross_val_score(svm,feat,classes,scoring='accuracy',cv=kf)
print('Accuracy using SVC: %2f%%'%(scores.mean()*100))
print(scores)
```

```
Accuracy using SVC: 91.386431%
[0.90350877 0.92982456 0.88596491 0.94736842 0.90265487]
```

In [104]:

```
# evaluating the data sets with kfold and KNN Classifier
knn = KNeighborsClassifier()
scores = cross_val_score(knn,feat,classes,scoring='accuracy',cv=kf)
print('Accuracy using KNN: %2f%%'%(scores.mean()*100))
print(scores)
```

```
Accuracy using KNN: 92.268281%
[0.93859649 0.89473684 0.88596491 0.96491228 0.92920354]
```

In [105]:

```
# evaluating the data sets with Stratified KFold and DecisionTree
scores = cross_val_score(dst,feat,classes,scoring='accuracy',cv=skf)
print('Accuracy using Decision Tree: %2f%%'%(scores.mean()*100))
print(scores)
```

```
Accuracy using Decision Tree: 91.210992%
[0.9122807  0.89473684 0.9122807  0.93859649 0.90265487]
```

In [106]:

```
# evaluating the data sets Stratified KFold and Naive Bayes Classifier
scores = cross_val_score(nb,feat,classes,scoring='accuracy',cv=skf)
print('Accuracy using GaussianNB: %2f%%'%(scores.mean()*100))
print(scores)
```

```
Accuracy using GaussianNB: 93.851886%
[0.92105263 0.92105263 0.94736842 0.94736842 0.95575221]
```

In [107]:

```
# evaluating the data sets Stratified KFold and SVM
scores = cross_val_score(svm,feat,classes,scoring='accuracy',cv=skf)
print('Accuracy using SVM: %2f%%'%(scores.mean()*100))
print(scores)
```

```
Accuracy using SVM: 91.217202%
[0.85087719 0.89473684 0.92982456 0.94736842 0.9380531 ]
```

In [108]:

```
# evaluating the data sets Stratified KFold and KNN
scores = cross_val_score(knn,feat,classes,scoring='accuracy',cv=skf)
print('Accuracy using KNN: %2f%%'%(scores.mean()*100))
print(scores)
```

```
Accuracy using KNN: 92.794597%
[0.88596491 0.93859649 0.93859649 0.94736842 0.92920354]
```

In [109]:

```python
# evaluating the data sets with LeaveOneOut and DecisionTree
scores = cross_val_score(dst,feat,classes,scoring='accuracy',cv=loocv)
print('Accuracy using GaussianNB: %2f%%'%(scores.mean()*100))
print(scores)
```

```
Accuracy using GaussianNB: 92.442882%
[1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1.
 1. 0. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 0. 1. 1. 1. 1. 0. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 0. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 1. 1. 0. 1. 1. 1. 1.
 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0.
 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1.]
```

In [110]:

```python
# evaluating the data sets LeaveOneOut and Naive Bayes Classifier
scores = cross_val_score(nb,feat,classes,scoring='accuracy',cv=loocv)
print('Accuracy using GaussianNB: %2f%%'%(scores.mean()*100))
print(scores)
```

```
Accuracy using GaussianNB: 93.848858%
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 1. 1. 0. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1.
 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 0. 1. 0. 1. 1. 1. 1.
 1. 1. 1. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 0. 1. 0. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 0.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 0. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

In [111]:

```python
# evaluating the data sets LeaveOneOut and SVM
scores = cross_val_score(svm,feat,classes,scoring='accuracy',cv=loocv)
print('Accuracy using SVM: %2f%%'%(scores.mean()*100))
print(scores)
```

```
Accuracy using SVM: 91.212654%
[1. 1. 1. 0. 1. 0. 1. 1. 0. 0. 1. 1. 1. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 0. 0. 0. 0. 1. 1. 0. 1. 1. 0.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 0. 1. 1. 1. 1.
 1. 1. 1. 0. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 0. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1.
 1. 0. 0. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 0. 0.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1.
 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1.
 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 0.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1.
 1. 1. 1. 1. 0. 0. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1.]
```

In [112]:

```
# evaluating the data sets LeaveOneOut and KNN
scores = cross_val_score(knn,feat,classes,scoring='accuracy',cv=loocv)
print('Accuracy using KNN: %2f%%'%(scores.mean()*100))
print(scores)
```

```
Accuracy using KNN: 93.321617%
[1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 1. 0. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 0. 0. 1. 1. 1.
 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1.
 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 0.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 0. 1. 1. 1. 1.
 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1.
 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 0.
 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

In [113]:

```
# evaluating the data sets with ShuffleSplit and DecisionTree
scores = cross_val_score(dst,feat,classes,scoring='accuracy',cv=shvc)
print('Accuracy using GaussianNB: %2f%%'%(scores.mean()*100))
print(scores)
```

```
Accuracy using GaussianNB: 92.982456%
[0.94736842 0.89473684 0.92982456 0.92982456 0.96491228 0.96491228
 0.89473684 0.9122807  0.94736842 0.9122807 ]
```

In [114]:

```
# evaluating the data sets with ShuffleSplit and Naive Bayes Classifier
scores = cross_val_score(nb,feat,classes,scoring='accuracy',cv=shvc)
print('Accuracy using GaussianNB: %2f%%'%(scores.mean()*100))
print(scores)
```

```
Accuracy using GaussianNB: 93.333333%
[0.9122807  0.89473684 0.9122807  0.9122807  0.92982456 0.96491228
 0.94736842 0.94736842 0.94736842 0.96491228]
```

In [115]:

```
# evaluating the data sets with ShuffleSplit and SVM
scores = cross_val_score(svm,feat,classes,scoring='accuracy',cv=shvc)
print('Accuracy using GaussianNB: %2f%%'%(scores.mean()*100))
print(scores)
```

```
Accuracy using GaussianNB: 92.631579%
[0.89473684 0.9122807  0.92982456 0.94736842 0.87719298 0.87719298
 0.94736842 0.94736842 0.96491228 0.96491228]
```

In [116]:

```
# evaluating the data sets with ShuffleSplit and KNN
scores = cross_val_score(knn,feat,classes,scoring='accuracy',cv=shvc)
print('Accuracy using KNN: %2f%%'%(scores.mean()*100))
print(scores)
```

```
Accuracy using KNN: 94.561404%
[0.92982456 0.92982456 0.9122807  0.96491228 0.9122807  0.94736842
 0.92982456 0.96491228 0.98245614 0.98245614]
```

In [21]:

```
#split the dataset into training and testing sets
features = df.drop(['diagnosis'],axis=1).values
classes=df['diagnosis'].values
```

In [22]:

```
feat_train, feat_test, class_train, class_test = train_test_split(features, classes, test_s
```

In [23]:

```
print('features train shape: ', feat_train.shape)
print('classes train shape: ', class_train.shape)
print('features test shape: ', feat_test.shape)
print('classes test shape: ', class_test.shape)
```

```
features train shape:  (455, 30)
classes train shape:  (455,)
features test shape:  (114, 30)
classes test shape:  (114,)
```

Decision Tree Classifier

Criterion=Gini

In [24]:

```
#Training
dectree=DecisionTreeClassifier(criterion='gini')

dectree.fit(feat_train,class_train)
```

Out[24]:

```
DecisionTreeClassifier()
```

In [25]:

```
#predict target values
pred=dectree.predict(feat_test)
print(pred)
```

```
['B' 'B' 'M' 'B' 'B' 'M' 'M' 'M' 'M' 'M' 'M' 'B' 'B' 'B' 'M' 'M' 'B' 'B'
 'B' 'B' 'M' 'B' 'B' 'M' 'M' 'B' 'B' 'B' 'B' 'M' 'M' 'B' 'M' 'M' 'M' 'M'
 'M' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'M' 'B' 'M' 'B' 'B' 'B' 'B'
 'M' 'B' 'B' 'M' 'B' 'B' 'M' 'M' 'M' 'B' 'B' 'B' 'M' 'B' 'M' 'B' 'B' 'B'
 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'M' 'B' 'B' 'B' 'B' 'M' 'M'
 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'M' 'B' 'M' 'B' 'B' 'B' 'B' 'M' 'B'
 'B' 'B' 'B' 'B' 'B' 'M']
```

In [26]:

```
#confusion matrix and accuracy
print("Accuracy",accuracy_score(class_test,pred))
print("Classification Report\n",classification_report(class_test,pred))
print("Confusion Matrix\n",confusion_matrix(class_test,pred))
```

```
Accuracy 0.9298245614035088
Classification Report
               precision    recall  f1-score   support

           B       0.95      0.95      0.95        74
           M       0.90      0.90      0.90        40

    accuracy                           0.93       114
   macro avg       0.92      0.92      0.92       114
weighted avg       0.93      0.93      0.93       114

Confusion Matrix
 [[70  4]
 [ 4 36]]
```

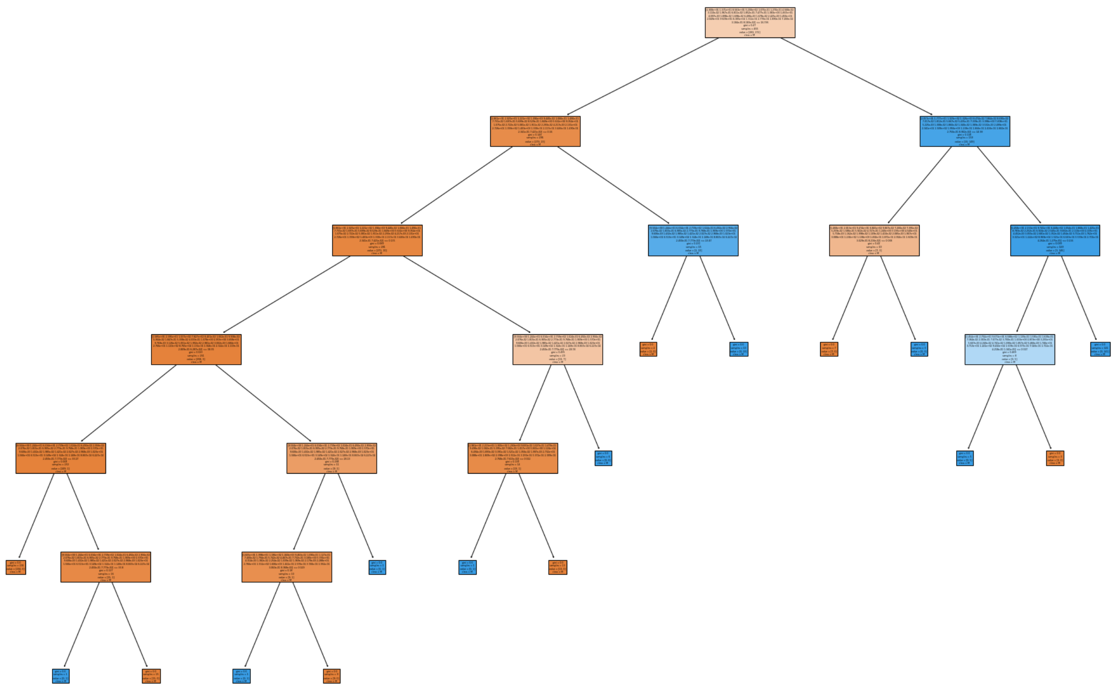
In [27]:

```
#Testing
pred=dectree.predict(feat_test)
print("Accuracy:",metrics.accuracy_score(class_test,pred))
```

```
Accuracy: 0.9298245614035088
```

In [28]:

```
from sklearn import tree
fig=plt.figure(figsize=(30,20))
_=tree.plot_tree(dectree,feature_names=features,class_names=classes,filled=True)
```



In [29]:

```
text_representation=tree.export_text(dectree)
print(text_representation)
```

```
|--- feature_20 <= 16.80
|   |--- feature_27 <= 0.16
|   |   |--- feature_27 <= 0.13
|   |   |   |--- feature_13 <= 38.35
|   |   |   |   |--- feature_21 <= 33.27
|   |   |   |   |   |--- class: B
|   |   |   |   |--- feature_21 >  33.27
|   |   |   |   |   |--- feature_21 <= 33.80
|   |   |   |   |   |   |--- class: M
|   |   |   |   |   |--- feature_21 >  33.80
|   |   |   |   |   |   |--- class: B
|   |   |   |--- feature_13 >  38.35
|   |   |   |   |--- feature_21 <= 28.13
|   |   |   |   |   |--- feature_6 <= 0.03
|   |   |   |   |   |   |--- class: M
|   |   |   |   |   |--- feature_6 >  0.03
|   |   |   |   |   |   |--- class: B
|   |   |   |   |--- feature_21 >  28.13
|   |   |   |   |   |--- class: M
|   |   |--- feature_27 >  0.13
|   |   |   |--- feature_21 <= 28.78
|   |   |   |   |--- feature_18 <= 0.01
|   |   |   |   |   |--- class: M
|   |   |   |   |--- feature_18 >  0.01
|   |   |   |   |   |--- class: B
|   |   |   |--- feature_21 >  28.78
|   |   |   |   |--- class: M
|   |--- feature_27 >  0.16
|   |   |--- feature_21 <= 23.47
|   |   |   |--- class: B
|   |   |--- feature_21 >  23.47
|   |   |   |--- class: M
|--- feature_20 >  16.80
|   |--- feature_1 <= 14.99
|   |   |--- feature_16 <= 0.04
|   |   |   |--- class: B
|   |   |--- feature_16 >  0.04
|   |   |   |--- class: M
|   |--- feature_1 >  14.99
|   |   |--- feature_26 <= 0.22
|   |   |   |--- feature_15 <= 0.02
|   |   |   |   |--- class: M
|   |   |   |--- feature_15 >  0.02
|   |   |   |   |--- class: B
|   |   |--- feature_26 >  0.22
|   |   |   |--- class: M
```

Criterion=Entropy

In [30]:

```
#Training
dectree=DecisionTreeClassifier(criterion='entropy')
dectree.fit(feat_train,class_train)
```

Out[30]:

```
DecisionTreeClassifier(criterion='entropy')
```

In [31]:

```
#confusion matrix and accuracy
pred=dectree.predict(feat_test)
print(pred)
print("Accuracy",accuracy_score(class_test,pred))
print("Classification Report\n",classification_report(class_test,pred))
print("Confusion Matrix\n",confusion_matrix(class_test,pred))
```

```
['B' 'B' 'M' 'B' 'B' 'M' 'M' 'M' 'M' 'M' 'M' 'B' 'B' 'B' 'M' 'M' 'B' 'B'
 'B' 'B' 'M' 'B' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'M' 'M' 'B' 'M' 'B' 'M' 'M'
 'M' 'B' 'M' 'B' 'B' 'B' 'B' 'B' 'B' 'B' 'M' 'M' 'B' 'M' 'M' 'B' 'B' 'B'
 'M' 'B' 'B' 'M' 'B' 'B' 'M' 'M' 'M' 'B' 'B' 'B' 'M' 'B' 'M' 'B' 'B' 'B'
 'B' 'B' 'M' 'M' 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'M' 'B' 'B' 'B' 'B' 'M' 'B'
 'B' 'B' 'B' 'B' 'B' 'M' 'B' 'B' 'B' 'M' 'B' 'M' 'B' 'B' 'B' 'B' 'M' 'B'
 'B' 'B' 'B' 'B' 'B' 'M']
Accuracy 0.956140350877193
Classification Report
               precision    recall  f1-score   support

           B       0.96      0.97      0.97        74
           M       0.95      0.93      0.94        40

    accuracy                           0.96       114
   macro avg       0.95      0.95      0.95       114
weighted avg       0.96      0.96      0.96       114

Confusion Matrix
 [[72  2]
 [ 3 37]]
```

In [32]:

```
text_representation=tree.export_text(dectree)
print(text_representation)
```

```
|--- feature_22 <= 105.95
|   |--- feature_27 <= 0.13
|   |   |--- feature_10 <= 0.64
|   |   |   |--- feature_21 <= 33.27
|   |   |   |   |--- class: B
|   |   |   |--- feature_21 >  33.27
|   |   |   |   |--- feature_21 <= 33.80
|   |   |   |   |   |--- class: M
|   |   |   |   |--- feature_21 >  33.80
|   |   |   |   |   |--- class: B
|   |   |--- feature_10 >  0.64
|   |   |   |--- feature_28 <= 0.21
|   |   |   |   |--- class: M
|   |   |   |--- feature_28 >  0.21
|   |   |   |   |--- class: B
|   |--- feature_27 >  0.13
|   |   |--- feature_21 <= 28.55
|   |   |   |--- feature_28 <= 0.36
|   |   |   |   |--- class: B
|   |   |   |--- feature_28 >  0.36
|   |   |   |   |--- feature_17 <= 0.03
|   |   |   |   |   |--- class: M
|   |   |   |   |--- feature_17 >  0.03
|   |   |   |   |   |--- class: B
|   |   |--- feature_21 >  28.55
|   |   |   |--- class: M
|--- feature_22 >  105.95
|   |--- feature_22 <= 117.45
|   |   |--- feature_21 <= 27.46
|   |   |   |--- feature_24 <= 0.14
|   |   |   |   |--- class: B
|   |   |   |--- feature_24 >  0.14
|   |   |   |   |--- feature_10 <= 0.22
|   |   |   |   |   |--- class: B
|   |   |   |   |--- feature_10 >  0.22
|   |   |   |   |   |--- class: M
|   |   |--- feature_21 >  27.46
|   |   |   |--- feature_4 <= 0.09
|   |   |   |   |--- feature_0 <= 15.06
|   |   |   |   |   |--- class: B
|   |   |   |   |--- feature_0 >  15.06
|   |   |   |   |   |--- class: M
|   |   |   |--- feature_4 >  0.09
|   |   |   |   |--- class: M
|   |--- feature_22 >  117.45
|   |   |--- feature_19 <= 0.00
|   |   |   |--- feature_3 <= 1016.55
|   |   |   |   |--- class: B
|   |   |   |--- feature_3 >  1016.55
|   |   |   |   |--- class: M
|   |   |--- feature_19 >  0.00
|   |   |   |--- class: M
```

KNN Classifier

In [33]:

```
feat_train, feat_test, class_train, class_test = train_test_split(features, classes, test_s
```

In [34]:

```
knn=KNeighborsClassifier(n_neighbors=4)
knn.fit(feat_train,class_train)
```

Out[34]:

```
KNeighborsClassifier(n_neighbors=4)
```

In [35]:

```
pred=knn.predict(feat_test)
print("Accuracy:",metrics.accuracy_score(class_test,pred))
```

```
Accuracy: 0.9210526315789473
```

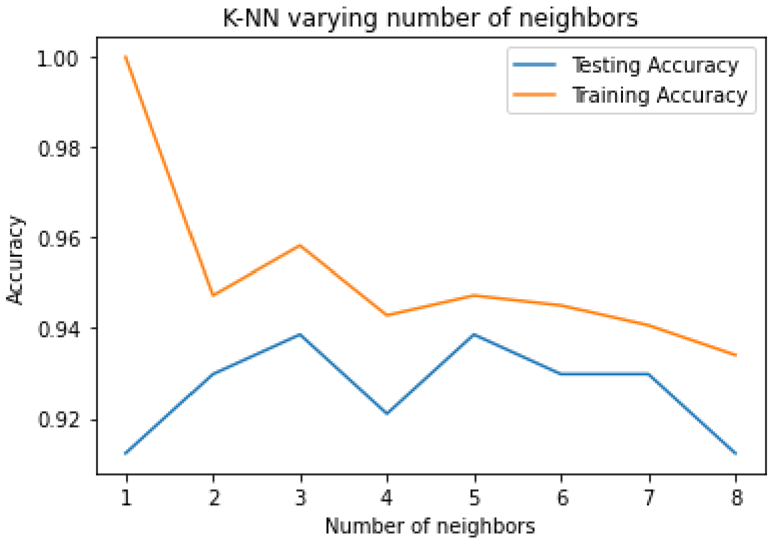
In [36]:

```
neighbors=np.arange(1,9)
train_accuracy=np.empty(len(neighbors))
test_accuracy=np.empty(len(neighbors))
for i,k in enumerate(neighbors):
    #setup as knn vlassifier with k neighbors
    knn1=KNeighborsClassifier(n_neighbors=k)
    #fit the model
    knn1.fit(feat_train,class_train)
    pred=knn1.predict(feat_test)
    #compute accuracy on the training set
    train_accuracy[i]=knn1.score(feat_train,class_train)
    #compute accuracy on the test set
    test_accuracy[i]=knn1.score(feat_test,class_test)
    print("Accuracy:",i,metrics.accuracy_score(class_test,pred))
print("train_accuracy\n",train_accuracy)
print("test_accuracy\n",test_accuracy)
```

```
Accuracy: 0 0.9122807017543859
Accuracy: 1 0.9298245614035088
Accuracy: 2 0.9385964912280702
Accuracy: 3 0.9210526315789473
Accuracy: 4 0.9385964912280702
Accuracy: 5 0.9298245614035088
Accuracy: 6 0.9298245614035088
Accuracy: 7 0.9122807017543859
train_accuracy
 [1.         0.94725275 0.95824176 0.94285714 0.94725275 0.94505495
 0.94065934 0.93406593]
test_accuracy
 [0.9122807  0.92982456 0.93859649 0.92105263 0.93859649 0.92982456
 0.92982456 0.9122807 ]
```

In [37]:

```
plt.title('K-NN varying number of neighbors')
plt.plot(neighbors,test_accuracy,label='Testing Accuracy')
plt.plot(neighbors,train_accuracy,label='Training Accuracy')
plt.legend()
plt.xlabel('Number of neighbors')
plt.ylabel('Accuracy')
plt.show()
```



Conclusion: From above graph we see training accuracy is more than that of testing accuracy

Support Vector Machine

In [38]:

```
df.head()
```

Out[38]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	com
0	M	17.99	10.38	122.80	1001.0	0.11840	
1	M	20.57	17.77	132.90	1326.0	0.08474	
2	M	19.69	21.25	130.00	1203.0	0.10960	
3	M	11.42	20.38	77.58	386.1	0.14250	
4	M	20.29	14.34	135.10	1297.0	0.10030	

5 rows × 31 columns

In [39]:

```
df['diagnosis'].unique()
```

Out[39]:

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

In [40]:

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

In [41]:

```
classess=df.diagnosis
classess
```

Out[41]:

```
0      1
1      1
2      1
3      1
4      1
      ..
564    1
565    1
566    1
567    1
568    0
Name: diagnosis, Length: 569, dtype: int64
```

In [42]:

```
featuress = df.drop(['diagnosis'],axis=1)
featuress
```

Out[42]:

	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 [43]:

```
from sklearn import preprocessing
#get col names
names = featuress.columns
#create scaler object
scaler = preprocessing.StandardScaler()
#fit data on the scaler object
scaled_df = scaler.fit_transform(featuress)
featuress = pd.DataFrame(scaled_df,columns=names)
```

In [44]:

```
featuress
```

Out[44]:

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_
0	1.097064	-2.073335	1.269934	0.984375	1.568466	3.2
1	1.829821	-0.353632	1.685955	1.908708	-0.826962	-0.4
2	1.579888	0.456187	1.566503	1.558884	0.942210	1.0
3	-0.768909	0.253732	-0.592687	-0.764464	3.283553	3.4
4	1.750297	-1.151816	1.776573	1.826229	0.280372	0.5
...	...	...	...	...	...	
564	2.110995	0.721473	2.060786	2.343856	1.041842	0.2
565	1.704854	2.085134	1.615931	1.723842	0.102458	-0.0
566	0.702284	2.045574	0.672676	0.577953	-0.840484	-0.0
567	1.838341	2.336457	1.982524	1.735218	1.525767	3.2
568	-1.808401	1.221792	-1.814389	-1.347789	-3.112085	-1.1

569 rows × 30 columns

In [45]:

```
feat_train, feat_test, class_train, class_test = train_test_split(featuress, classess, trai
```

In [46]:

```
svlassifier=SVC(kernel='linear')
svlassifier.fit(feat_train,class_train)
```

Out[46]:

```
SVC(kernel='linear')
```

In [47]:

```
print('features train shape: ', feat_train.shape)
print('classes train shape: ', class_train.shape)
print('features test shape: ', feat_test.shape)
print('classes test shape: ', class_test.shape)
```

```
features train shape:  (512, 30)
classes train shape:  (512,)
features test shape:  (57, 30)
classes test shape:  (57,)
```

In [48]:

```
pred=svlassifier.predict(feat_test)
```

In [49]:

```
accuracy_score(class_test,pred)
```

Out[49]:

```
0.9473684210526315
```

In [50]:

```
print(classification_report(class_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.92      1.00      0.96        35
           1       1.00      0.86      0.93        22

    accuracy                           0.95        57
   macro avg       0.96      0.93      0.94        57
weighted avg       0.95      0.95      0.95        57
```

In [51]:

```
cf=confusion_matrix(class_test,pred)
cf
```

Out[51]:

```
array([[35,  0],
       [ 3, 19]], dtype=int64)
```

In [52]:

```
kernels = ['linear','rbf','poly']
for kernel in kernels:
    sv = SVC(kernel=kernel).fit(feat_train,class_train)
    pred=sv.predict(feat_test)
    print("Accuracy:("+kernel+")", accuracy_score(class_test,pred))
```

```
Accuracy:(linear) 0.9473684210526315
Accuracy:(rbf) 0.9649122807017544
Accuracy:(poly) 0.8947368421052632
```

In [53]:

```
gammas = [0.1,1,10,100]
for gamma in gammas:
    sv = SVC(kernel='rbf', gamma=gamma).fit(feat_train,class_train)
    pred=sv.predict(feat_test)
    print("Accuracy:(", gamma , ")", accuracy_score(class_test,pred))
```

```
Accuracy:( 0.1 ) 0.9473684210526315
Accuracy:( 1 ) 0.6140350877192983
Accuracy:( 10 ) 0.6140350877192983
Accuracy:( 100 ) 0.6140350877192983
```

In [54]:

```
degrees = [0,1,2,3,4,5,20]
for degree in degrees:
    sv = SVC(kernel='poly', degree=degree).fit(feat_train,class_train)
    pred=sv.predict(feat_test)
    print("Accuracy:(", degree , "):", accuracy_score(class_test,pred))
```

```
Accuracy:( 0 ): 0.6140350877192983
Accuracy:( 1 ): 0.9473684210526315
Accuracy:( 2 ): 0.8771929824561403
Accuracy:( 3 ): 0.8947368421052632
Accuracy:( 4 ): 0.8596491228070176
Accuracy:( 5 ): 0.8596491228070176
Accuracy:( 20 ): 0.7543859649122807
```

Support Vector Regression

In [55]:

```
feat_train, feat_test, class_train, class_test = train_test_split(featuress, classess, trai
```

In [56]:

```
regressor=SVC(kernel='linear')
regressor.fit(feat_train,class_train)
```

Out[56]:

```
SVC(kernel='linear')
```

In [57]:

```
pred=svlassifier.predict(feat_test)
```

In [58]:

```
mean_squared_error(class_test,pred)
```

Out[58]:

```
0.05263157894736842
```

In [59]:

```
regressor.score(feat_test,class_test)
```

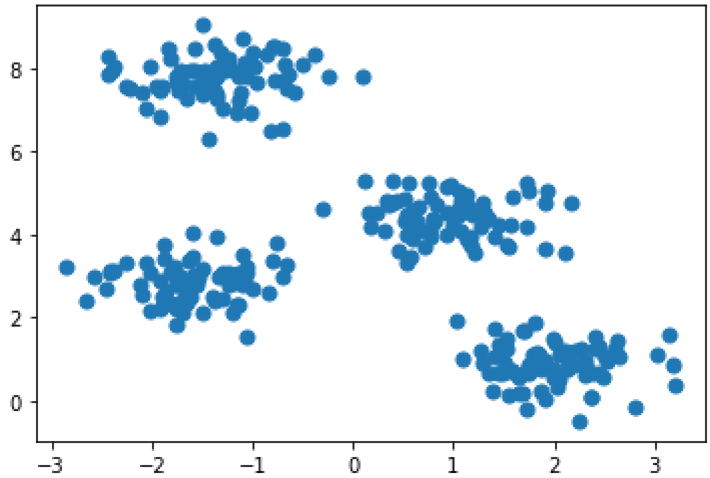
Out[59]:

```
0.9473684210526315
```

K Means Clustering

In [60]:

```
X,y_true = make_blobs(n_samples=300,centers=4,cluster_std=0.5,random_state=0)
plt.scatter(X[:, 0],X[:,1],s=50);
```



In [61]:

```
kmeans = KMeans(n_clusters=4)
kmeans.fit(X)
y_kmeans = kmeans.predict(X)
```

In [62]:

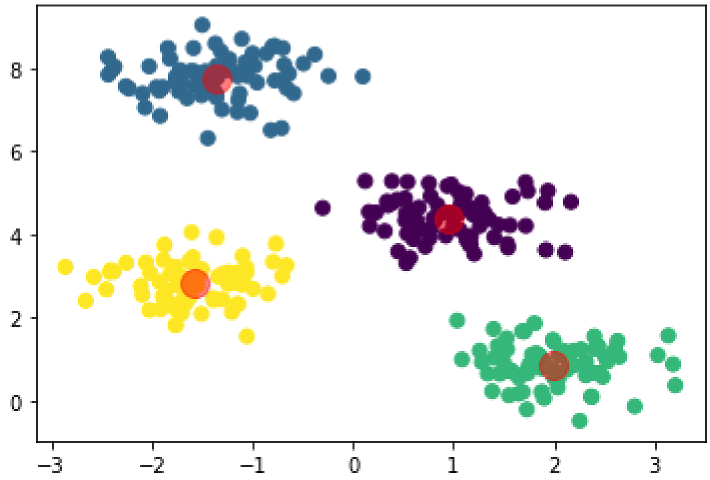
```
y_kmeans
```

Out[62]:

```
array([2, 1, 0, 1, 2, 2, 3, 0, 1, 1, 3, 1, 0, 1, 2, 0, 0, 2, 3, 3, 2, 2,
       0, 3, 3, 0, 2, 0, 3, 0, 1, 1, 0, 1, 1, 1, 1, 1, 3, 2, 0, 3, 0, 0,
       3, 3, 1, 3, 1, 2, 3, 2, 1, 2, 2, 3, 1, 3, 1, 2, 1, 0, 1, 3, 3, 3,
       1, 2, 1, 3, 0, 3, 1, 3, 3, 1, 3, 0, 2, 1, 2, 0, 2, 2, 1, 0, 2, 0,
       1, 1, 0, 2, 1, 3, 3, 0, 2, 2, 0, 3, 1, 2, 1, 2, 0, 2, 2, 0, 1, 0,
       3, 3, 2, 1, 2, 0, 1, 2, 2, 0, 3, 2, 3, 2, 2, 2, 2, 3, 2, 3, 1, 3,
       3, 2, 1, 3, 3, 1, 0, 1, 1, 3, 0, 3, 0, 3, 1, 0, 1, 1, 1, 0, 1, 0,
       2, 3, 1, 3, 2, 0, 1, 0, 0, 2, 0, 3, 3, 0, 2, 0, 0, 1, 2, 0, 3, 1,
       2, 2, 0, 3, 2, 0, 3, 3, 0, 0, 0, 0, 2, 1, 0, 3, 0, 0, 3, 3, 3, 0,
       3, 1, 0, 3, 2, 3, 0, 1, 3, 1, 0, 1, 0, 3, 0, 0, 1, 3, 3, 2, 2, 0,
       1, 2, 2, 3, 2, 3, 0, 1, 1, 0, 0, 1, 0, 2, 3, 0, 2, 3, 1, 3, 2, 0,
       2, 1, 1, 1, 1, 3, 3, 1, 0, 3, 2, 0, 3, 3, 3, 2, 2, 1, 0, 0, 3, 2,
       1, 3, 0, 1, 0, 2, 2, 3, 3, 0, 2, 2, 2, 0, 1, 1, 2, 2, 0, 2, 2, 2,
       1, 3, 1, 0, 2, 2, 1, 1, 1, 2, 2, 0, 1, 3])
```

In [63]:

```
plt.scatter(X[:, 0],X[:,1],c=y_kmeans,s=50,cmap='viridis')
centers = kmeans.cluster_centers_
plt.scatter(centers[:,0],centers[:,1],c='red',s=200,alpha=0.5);
```



In [64]:

```
df1 = datasets.load_breast_cancer()
df1.data.shape
```

Out[64]:

```
(569, 30)
```

In [65]:

```
df1.data
```

Out[65]:

```
array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01,
        1.189e-01],
       [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01,
        8.902e-02],
       [1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.613e-01,
        8.758e-02],
       ...,
       [1.660e+01, 2.808e+01, 1.083e+02, ..., 1.418e-01, 2.218e-01,
        7.820e-02],
       [2.060e+01, 2.933e+01, 1.401e+02, ..., 2.650e-01, 4.087e-01,
        1.240e-01],
       [7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 2.871e-01,
        7.039e-02]])
```

In [66]:

```
kmeans= KMeans(n_clusters=10, random_state=42)
clusters = kmeans.fit_predict(df1.data)
kmeans.cluster_centers_.shape
```

Out[66]:

```
(10, 30)
```

In [67]:

```
clusters
```

Out[67]:

```
array([9, 9, 1, 0, 1, 7, 1, 2, 7, 7, 5, 5, 5, 2, 7, 2, 2, 5, 4, 7, 7, 8,
       2, 4, 9, 5, 2, 1, 5, 5, 1, 2, 5, 9, 5, 5, 2, 0, 7, 7, 7, 0, 1, 2,
       7, 1, 8, 7, 0, 7, 0, 7, 0, 5, 2, 0, 9, 2, 7, 8, 8, 8, 2, 8, 2, 2,
       8, 0, 8, 0, 9, 8, 1, 2, 7, 5, 7, 1, 1, 7, 0, 7, 6, 5, 0, 1, 2, 1,
       0, 2, 2, 2, 2, 7, 2, 1, 0, 8, 0, 2, 2, 8, 0, 8, 8, 7, 0, 0, 4, 0,
       8, 0, 7, 8, 8, 0, 8, 2, 5, 5, 0, 1, 4, 7, 7, 7, 2, 1, 2, 1, 0, 5,
       5, 2, 1, 7, 0, 0, 2, 0, 8, 5, 0, 7, 0, 0, 0, 2, 7, 7, 7, 8, 8, 0,
       7, 0, 5, 5, 0, 0, 0, 1, 9, 0, 4, 2, 8, 5, 1, 2, 0, 2, 2, 8, 8, 8,
       8, 2, 7, 0, 6, 9, 5, 0, 2, 8, 5, 0, 0, 0, 7, 7, 8, 7, 2, 7, 2, 5,
       1, 2, 7, 5, 4, 2, 7, 2, 8, 5, 7, 2, 1, 0, 6, 5, 2, 7, 0, 8, 9, 4,
       7, 7, 8, 2, 7, 2, 8, 2, 7, 7, 5, 0, 0, 9, 8, 7, 6, 1, 7, 5, 7, 0,
       0, 7, 1, 8, 7, 7, 0, 0, 9, 0, 9, 5, 9, 2, 9, 2, 5, 2, 9, 5, 5, 2,
       5, 6, 8, 7, 7, 8, 7, 0, 4, 8, 5, 0, 0, 5, 7, 7, 1, 0, 1, 2, 7, 0,
       0, 7, 0, 0, 2, 2, 7, 0, 0, 7, 8, 0, 2, 8, 9, 0, 1, 8, 0, 0, 7, 8,
       7, 7, 0, 2, 7, 0, 8, 0, 0, 1, 8, 0, 8, 1, 7, 9, 0, 0, 7, 0, 5, 2,
       2, 7, 0, 0, 0, 5, 7, 9, 8, 6, 2, 8, 0, 1, 0, 8, 0, 2, 0, 0, 0, 2,
       6, 2, 0, 0, 7, 7, 8, 8, 0, 7, 0, 2, 7, 9, 1, 7, 6, 4, 5, 2, 1, 9,
       7, 2, 8, 7, 7, 0, 0, 0, 0, 0, 7, 2, 0, 7, 0, 5, 8, 8, 5, 9, 0, 7,
       7, 7, 0, 0, 5, 0, 7, 7, 0, 0, 2, 7, 5, 7, 0, 0, 8, 2, 2, 0, 8, 1,
       0, 0, 0, 2, 0, 7, 8, 8, 8, 0, 0, 7, 2, 0, 1, 1, 2, 2, 7, 7, 7, 7,
       0, 5, 7, 8, 5, 0, 5, 2, 2, 9, 0, 1, 0, 7, 7, 7, 0, 7, 7, 8, 1, 3,
       7, 0, 7, 7, 7, 8, 5, 0, 8, 0, 2, 0, 0, 7, 2, 7, 0, 2, 0, 2, 7, 7,
       2, 0, 2, 1, 0, 5, 7, 5, 5, 0, 7, 2, 7, 7, 1, 9, 2, 7, 0, 6, 8, 8,
       0, 8, 2, 2, 0, 2, 2, 2, 2, 0, 1, 1, 7, 7, 8, 6, 0, 7, 8, 8, 7, 0,
       7, 0, 0, 0, 7, 1, 8, 9, 7, 0, 8, 8, 0, 2, 2, 7, 7, 7, 8, 8, 8, 0,
       8, 0, 7, 8, 7, 8, 8, 8, 7, 0, 7, 0, 2, 9, 9, 1, 5, 9, 8])
```

In [68]:

```
kmeans.cluster_centers_
```

Out[68]:

```
array([[1.17684058e+01, 1.79862319e+01, 7.55610870e+01, 4.25743478e+02,
        9.38014493e-02, 7.83112319e-02, 4.32502659e-02, 2.46985362e-02,
        1.75413768e-01, 6.29150000e-02, 2.76519565e-01, 1.27060362e+00,
        1.96543261e+00, 1.99787174e+01, 7.22113043e-03, 2.04303768e-02,
        2.34826855e-02, 9.85089855e-03, 2.10551449e-02, 3.33913913e-03,
        1.29645652e+01, 2.40863043e+01, 8.42486957e+01, 5.13605797e+02,
        1.27910217e-01, 1.82056739e-01, 1.62445254e-01, 7.42174783e-02,
        2.76171014e-01, 7.95076812e-02],
       [1.91604762e+01, 2.14050000e+01, 1.26369048e+02, 1.14311905e+03,
        9.98419048e-02, 1.42173810e-01, 1.68243571e-01, 9.62930952e-02,
        1.93209524e-01, 5.99416667e-02, 7.13964286e-01, 1.27272143e+00,
        4.93688095e+00, 8.57366667e+01, 6.97321429e-03, 3.25688095e-02,
        4.46107143e-02, 1.63225714e-02, 2.27640476e-02, 3.93226190e-03,
        2.29930952e+01, 2.84192857e+01, 1.52921429e+02, 1.61173810e+03,
        1.38404762e-01, 3.40352381e-01, 4.35235714e-01, 1.82757143e-01,
        3.16050000e-01, 8.44071429e-02],
       [1.48048889e+01, 1.95548889e+01, 9.68246667e+01, 6.78477778e+02,
        9.82186667e-02, 1.18621889e-01, 1.05034444e-01, 5.72743333e-02,
        1.83950000e-01, 6.25370000e-02, 3.70153333e-01, 1.05712111e+00,
        2.65971111e+00, 3.36827778e+01, 6.25054444e-03, 2.72024333e-02,
        3.36346556e-02, 1.28329667e-02, 1.92910667e-02, 3.63720889e-03,
        1.71068889e+01, 2.64898889e+01, 1.13773333e+02, 8.96530000e+02,
        1.36545333e-01, 3.14547556e-01, 3.47453667e-01, 1.41027667e-01,
        3.04734444e-01, 8.76576667e-02],
       [2.74200000e+01, 2.62700000e+01, 1.86900000e+02, 2.50100000e+03,
        1.08400000e-01, 1.98800000e-01, 3.63500000e-01, 1.68900000e-01,
        2.06100000e-01, 5.62300000e-02, 2.54700000e+00, 1.30600000e+00,
        1.86500000e+01, 5.42200000e+02, 7.65000000e-03, 5.37400000e-02,
        8.05500000e-02, 2.59800000e-02, 1.69700000e-02, 4.55800000e-03,
        3.60400000e+01, 3.13700000e+01, 2.51200000e+02, 4.25400000e+03,
        1.35700000e-01, 4.25600000e-01, 6.83300000e-01, 2.62500000e-01,
        2.64100000e-01, 7.42700000e-02],
       [2.19266667e+01, 2.32366667e+01, 1.46633333e+02, 1.50011111e+03,
        1.05876667e-01, 1.77311111e-01, 2.41588889e-01, 1.27676667e-01,
        1.94844444e-01, 6.00200000e-02, 9.22911111e-01, 1.39977778e+00,
        6.71611111e+00, 1.31922222e+02, 7.83422222e-03, 4.41288889e-02,
        5.84833333e-02, 1.67322222e-02, 2.11144444e-02, 4.27744444e-03,
        2.75322222e+01, 3.09733333e+01, 1.88311111e+02, 2.32477778e+03,
        1.44977778e-01, 4.16088889e-01, 5.59288889e-01, 2.28133333e-01,
        3.10455556e-01, 8.43200000e-02],
       [1.71307843e+01, 2.14549020e+01, 1.12686275e+02, 9.14935294e+02,
        9.86378431e-02, 1.29746667e-01, 1.34528235e-01, 7.78741176e-02,
        1.88533333e-01, 6.05247059e-02, 5.50582353e-01, 1.26919020e+00,
        3.95376471e+00, 5.98396078e+01, 6.62015686e-03, 3.02961961e-02,
        3.82084314e-02, 1.48891176e-02, 1.94684314e-02, 4.03854902e-03,
        2.02923529e+01, 2.87207843e+01, 1.34923529e+02, 1.26694118e+03,
        1.36350588e-01, 3.13267843e-01, 3.74625490e-01, 1.63206863e-01,
        3.10478431e-01, 8.42862745e-02],
       [2.43160000e+01, 2.23750000e+01, 1.61910000e+02, 1.85420000e+03,
        1.03174000e-01, 1.68032000e-01, 2.35580000e-01, 1.40631000e-01,
        1.79210000e-01, 5.89640000e-02, 1.23297000e+00, 1.14835000e+00,
        8.82800000e+00, 1.99120000e+02, 6.61970000e-03, 2.92370000e-02,
        3.93590000e-02, 1.50810000e-02, 1.95370000e-02, 3.44310000e-03,
        3.09990000e+01, 2.98160000e+01, 2.08940000e+02, 2.93600000e+03,
        1.40180000e-01, 3.64520000e-01, 4.68620000e-01, 2.28060000e-01,
```

```
 2.76880000e-01, 8.10070000e-02],
[1.33393496e+01, 1.87629268e+01, 8.62058537e+01, 5.48933333e+02,
 9.22819512e-02, 8.97267480e-02, 5.94710976e-02, 3.34234309e-02,
 1.72340650e-01, 6.13792683e-02, 2.78245528e-01, 1.06987073e+00,
 1.97310813e+00, 2.29121138e+01, 5.74791870e-03, 2.19371138e-02,
 2.58573659e-02, 9.79302439e-03, 1.79366829e-02, 3.29744146e-03,
 1.48126016e+01, 2.49307317e+01, 9.69660163e+01, 6.73665041e+02,
 1.25623008e-01, 2.35954309e-01, 2.28506683e-01, 9.55779593e-02,
 2.80860976e-01, 8.24360163e-02],
[9.67473077e+00, 1.77228205e+01, 6.18244872e+01, 2.87002564e+02,
 9.72243590e-02, 8.23483333e-02, 4.67906282e-02, 1.86753077e-02,
 1.84562821e-01, 6.93942308e-02, 3.03461538e-01, 1.52944872e+00,
 2.08427821e+00, 1.81051923e+01, 1.03231923e-02, 2.62625513e-02,
 3.52811410e-02, 1.03885641e-02, 2.58941026e-02, 5.29807692e-03,
 1.06348590e+01, 2.28014103e+01, 6.85088462e+01, 3.45534615e+02,
 1.33208077e-01, 1.65925128e-01, 1.44518590e-01, 5.28589744e-02,
 2.72964103e-01, 8.68620513e-02],
[2.00092593e+01, 2.18959259e+01, 1.32700000e+02, 1.24794815e+03,
 1.03467407e-01, 1.61638519e-01, 1.97029259e-01, 1.09776296e-01,
 1.94181481e-01, 6.17955556e-02, 7.38003704e-01, 1.01643704e+00,
 5.13155556e+00, 9.47937037e+01, 5.93300000e-03, 3.06088889e-02,
 4.16533333e-02, 1.52032593e-02, 1.73874074e-02, 3.84103704e-03,
 2.52788889e+01, 2.92848148e+01, 1.68740741e+02, 1.95666667e+03,
 1.45407407e-01, 4.14522222e-01, 5.19233333e-01, 2.14877778e-01,
 3.23803704e-01, 9.26714815e-02]])
```

In [69]:

```
#fig, ax = plt.subplots(2,5, figsize=(8,3))
#centers = kmeans.cluster_centers_.reshape(1,1,1)
#for axi, center in zip(ax.flat,centers):
  #  axi.set(xticks=[],yticks=[])
 #   axi.imshow(center,interpolation='nearest',cmap=plt.cm.binary)
```

In [70]:

```
mat= confusion_matrix(df1.target,clusters)
mat
```

Out[70]:

```
array([[  5,  42,  52,   1,   9,  49,  10,  17,   0,  27],
       [133,   0,  38,   0,   0,   2,   0, 106,  78,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0]], dtype=int64)
```

Principal Component Analysis

In [71]:

```
from sklearn import datasets
cancer = datasets.load_breast_cancer()
```

In [72]:

```
X=cancer.data
y=cancer.target
```

In [73]:

```
pca = decomposition.PCA(n_components=3)
pca.fit(featuress)
X1 = pca.transform(featuress)
```

In [74]:

```
print(cancer.data.shape)
print(featuress.shape)
print(X1.shape)
```

```
(569, 30)
(569, 30)
(569, 3)
```

In [75]:

```
from sklearn.model_selection import train_test_split
(train_feat,test_feat,train_classes,test_classes)= train_test_split(featuress,classess,trai
dectree = DecisionTreeClassifier()
dectree.fit(train_feat,train_classes)
```

Out[75]:

```
DecisionTreeClassifier()
```

In [76]:

```
from sklearn import metrics
pred=dectree.predict(test_feat)
print("Accuracy:",metrics.accuracy_score(test_classes,pred))
```

```
Accuracy: 0.9228070175438596
```

In [77]:

```
from sklearn.model_selection import train_test_split
(train_feat,test_feat,train_classes,test_classes)= train_test_split(X1,classess,train_size=
dectree = DecisionTreeClassifier()
dectree.fit(train_feat,train_classes)
```

Out[77]:

```
DecisionTreeClassifier()
```

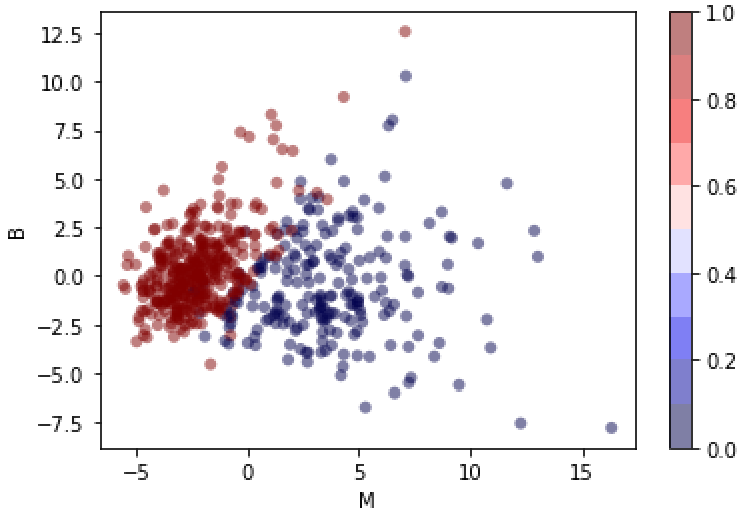
In [78]:

```
from sklearn import metrics
pred=dectree.predict(test_feat)
print("Accuracy:",metrics.accuracy_score(test_classes,pred))
```

```
Accuracy: 0.9228070175438596
```

In [79]:

```
plt.scatter(X1[:,0],X1[:,1],c=cancer.target,edgecolor='none',alpha=0.5,cmap=plt.cm.get_cmap
plt.xlabel('M')
plt.ylabel('B')
plt.colorbar();
```



Select K Percentile and K Best

In [80]:

```
X_train,X_test,y_train,y_test = train_test_split(featuress,classess,test_size=0.2,random_st
```

In [81]:

```
select = SelectPercentile(percentile=80)
select.fit(X_train,y_train)
```

Out[81]:

```
SelectPercentile(percentile=80)
```

In [82]:

```
X_train_selected = select.transform(X_train)
print("X_train.shape: {}".format(X_train.shape))
print("X_train_selected.shape: {}".format(X_train_selected.shape))
```

```
X_train.shape: (455, 30)
X_train_selected.shape: (455, 24)
```

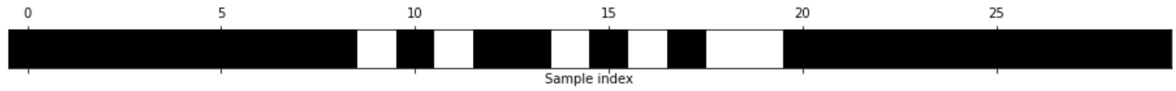
In [83]:

```
mask = select.get_support()
print(mask)
plt.matshow(mask.reshape(1,-1),cmap='gray_r')
plt.xlabel("Sample index")
plt.yticks(())
```

```
[ True  True  True  True  True  True  True  True  True False  True False
  True  True False  True False  True False False  True  True  True  True
  True  True  True  True  True  True]
```

Out[83]:

```
([], [])
```



In [84]:

```
from sklearn.tree import DecisionTreeClassifier
X_test_selected = select.transform(X_test)
lr = DecisionTreeClassifier()
lr.fit(X_train,y_train)
print("Score with all features: {:.3f}".format(lr.score(X_test,y_test)))
lr.fit(X_train_selected,y_train)
print("Score with all features: {:.3f}".format(lr.score(X_test_selected,y_test)))
```

```
Score with all features: 0.912
Score with all features: 0.939
```

In [85]:

```
select=SelectKBest(k=1)
select.fit(X_train,y_train)
```

Out[85]:

```
SelectKBest(k=1)
```

In [86]:

```
X_train_selected = select.transform(X_train)
print("X_train.shape: {}".format(X_train.shape))
print("X_train_selected.shape: {}".format(X_train_selected.shape))
```

```
X_train.shape: (455, 30)
X_train_selected.shape: (455, 1)
```

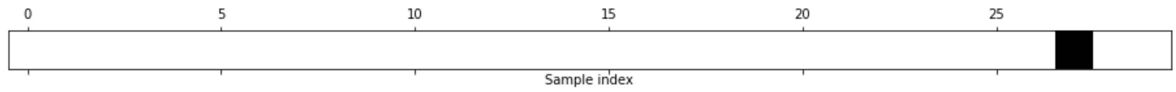
In [87]:

```
mask = select.get_support()
print(mask)
plt.matshow(mask.reshape(1,-1),cmap='gray_r')
plt.xlabel("Sample index")
plt.yticks(())
```

```
[False False False False False False False False False False False False
 False False False False False False False False False False False False
 False False False  True False False]
```

Out[87]:

```
([], [])
```



In [88]:

```
from sklearn.tree import DecisionTreeClassifier
X_test_selected = select.transform(X_test)
lr = DecisionTreeClassifier()
lr.fit(X_train,y_train)
print("Score with all features: {:.3f}".format(lr.score(X_test,y_test)))
lr.fit(X_train_selected,y_train)
print("Score with all features: {:.3f}".format(lr.score(X_test_selected,y_test)))
```

```
Score with all features: 0.904
Score with all features: 0.877
```

Feature Seletion-Model Based

In [126]:

```
from sklearn.feature_selection import SelectFromModel
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
select = SelectFromModel(RandomForestClassifier(n_estimators=25,random_state=1),threshold="
#select = SelectFromModel(SVC(kernel='linear'))
```

In [127]:

```
select.fit(X_train,y_train)
X_train_l1 = select.transform(X_train)
print("X_train.shape: {}".format(X_train.shape))
print("X_train_l1.shape: {}".format(X_train_l1.shape))
```

```
X_train.shape: (455, 30)
X_train_l1.shape: (455, 15)
```

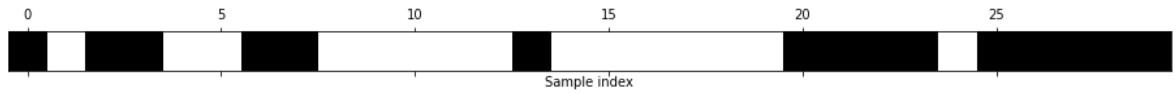
In [128]:

```
mask = select.get_support()
print(mask)
plt.matshow(mask.reshape(1,-1),cmap='gray_r')
plt.xlabel("Sample index")
plt.yticks(())
```

```
[ True False  True  True False False  True  True False False False False
 False  True False False False False False False  True  True  True  True
 False  True  True  True  True  True]
```

Out[128]:

```
([], [])
```



In [129]:

```
X_test_l1 = select.transform(X_test)
score = SVC().fit(X_train,y_train).score(X_test,y_test)
print("Test Score: {:.3f}".format(score))
score = SVC().fit(X_train_l1,y_train).score(X_test_l1,y_test)
print("Test Score: {:.3f}".format(score))
```

```
Test Score: 0.974
Test Score: 0.965
```

Iterative Feature Selection

In [109]:

```
from sklearn.feature_selection import RFE
select = RFE(RandomForestClassifier(n_estimators=1,random_state=0),n_features_to_select=10)
select.fit(X_train,y_train)
```

Out[109]:

```
RFE(estimator=RandomForestClassifier(n_estimators=1, random_state=0),
    n_features_to_select=10)
```

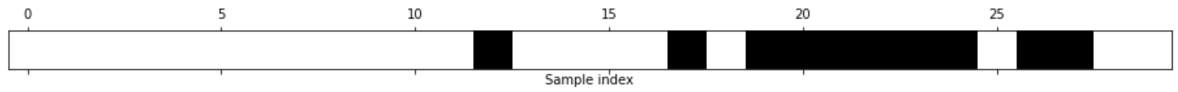
In [110]:

```
mask = select.get_support()
print(mask)
plt.matshow(mask.reshape(1,-1),cmap='gray_r')
plt.xlabel("Sample index")
plt.yticks(())
```

```
[False False False False False False False False False False False False
  True False False False False  True False  True  True  True  True  True
  True False  True  True False False]
```

Out[110]:

```
([], [])
```



In [113]:

```
from sklearn.linear_model import LogisticRegression
X_train_rfe =select.transform(X_train)
X_test_rfe = select.transform(X_test)
score = LogisticRegression().fit(X_train_rfe,y_train).score(X_test_rfe,y_test)
print("Test score: {:.3f}".format(score))
print("Test score: {:.3f}".format(select.score(X_test,y_test)))
```

```
Test score: 0.956
Test score: 0.965
```

Conclusion

```
Top 5 Accuracy:
    Random Forest=0.97
    SVM kernel:rbf= 0.96
    KNN shvc=95.96
    Multticlass Logistic Regression=95.3
    Decision Tree Classifier entropy=0.94
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```