Nombre: Josue Alejandro Sauca Pucha

Fecha: 12-07-2023

Evaluación Unidad 2 July 12, 2023 1 Evaluación Unidad 2 Parte1

1. Descargue el archivo Breast Cancer Wisconsin (Diagnostic) ubicado en el repositorio UCI
2. Lea la descripción del archivo

In [42]:
```
import pandas as pd
from sklearn.model_selection import train_test_split

#Leemos el archivo descargado con extension .data
data1 =  pd.read_csv('wdbc.data',delimiter=',',header=None)
data1.head()
```

Out[42]:

	0	1	2	3	4	5	6	7	8	9	...	22	23
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	25.38	17.33
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	24.99	23.41
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	23.57	25.53
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	14.91	26.50
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	22.54	16.67

rows × 32 columns

In [43]:
```
#Asiganmos el nombre a las columnas con la documentacion de la pagina https://
data1.columns =['id','diagnosis','radius1','texture1',
                'perimeter1','area1','smoothness1','compactness1'
                ,'concavity1','concave_points1','symmetry1'
                ,'fractal_dimension1','radius2','texture2','perimeter2',
               'area2','smoothness2','compactness2','concavity2',
                'concave_points2','symmetry2','fractal_dimension2','radius3',
               'texture3','perimeter3','area3','smoothness3','compactness3'
               ,'concavity3','concave_points3','symmetry3','fractal_dimension3
```

```
3. Divida el archivo de dos partes que ocupen el 95% para entrenamiento y el
5% para test
```

In [184]:
```
test_data
```

Out[184]:

nmetry1	fractal_dimension1	...	radius3	texture3	perimeter3	area3	smoothness3	compactness3
0.1925	0.06373	...	14.97	24.64	96.05	677.9	0.14260	0.2378(
0.1582	0.05461	...	24.86	26.58	165.90	1866.0	0.11930	0.2336(
0.1931	0.05796	...	19.26	26.00	124.90	1156.0	0.15460	0.2394(
0.1811	0.07102	...	12.88	22.91	89.61	515.8	0.14500	0.2629(
0.1818	0.06782	...	12.26	19.68	78.78	457.8	0.13450	0.2118(
0.2397	0.07016	...	25.74	39.42	184.60	1821.0	0.16500	0.8681(
0.1824	0.06140	...	27.66	25.80	195.00	2227.0	0.12940	0.3885(
0.1739	0.06149	...	20.01	19.52	134.90	1227.0	0.12550	0.2812(
0.1942	0.06902	...	15.53	23.19	96.66	614.9	0.15360	0.4791(
0.1813	0.05536	...	15.14	21.80	101.20	718.9	0.09384	0.2006(
0.1619	0.05584	...	14.35	34.23	91.29	632.9	0.12890	0.1063(
0.1893	0.05534	...	20.05	26.30	130.70	1260.0	0.11680	0.2119(
0.1515	0.05266	...	15.98	25.82	102.30	782.1	0.10450	0.0999(
0.1869	0.06532	...	17.73	25.21	113.70	975.2	0.14260	0.2116(
0.1689	0.05808	...	13.61	19.27	87.22	564.9	0.12920	0.2074(
0.1634	0.07224	...	20.33	32.72	141.30	1298.0	0.13920	0.2817(
0.1659	0.05348	...	15.61	17.58	101.70	760.2	0.11390	0.1011(
0.1791	0.06331	...	10.65	22.88	67.88	347.3	0.12650	0.1200(
0.1930	0.07818	...	7.93	19.54	50.41	185.2	0.15840	0.1202(
0.2127	0.06251	...	24.30	25.48	160.20	1809.0	0.12680	0.3135(
0.1662	0.06566	...	16.57	20.86	110.30	812.4	0.14110	0.3542(
0.1937	0.06161	...	13.56	25.80	88.33	559.5	0.14320	0.1773(
0.2162	0.06606	...	26.23	28.74	172.00	2081.0	0.15020	0.5717(
0.1630	0.06439	...	11.11	28.94	69.92	376.3	0.11260	0.0709(
0.1935	0.05878	...	12.44	31.62	81.39	476.5	0.09545	0.1361(
0.2403	0.06641	...	14.08	12.49	91.36	605.5	0.14510	0.1379(
0.1617	0.05594	...	14.24	17.37	96.59	623.7	0.11660	0.2685(
0.1593	0.06127	...	10.84	34.91	69.57	357.6	0.13840	0.1710(
0.1759	0.06183	...	13.75	25.99	87.82	579.7	0.12980	0.1839(

In [225]:

```
# Con la funcion train_test_split dividimos en dos parte el datase,
#la primera parte para entrenamiento y la segunda para test, en este caso el t
#y lo demas queda para entrenamiento el 95%

X = data1.iloc[:, 2:]  # La variable X contiene los datos desde el radio hasta
y = data1.iloc[:, 1]   # Contiene todos los datos del diagnostico

train_data, test_data, train_labels, test_labels = train_test_split(X, y, test

print("Training set:")
print(train_data.head())

print("Test set:")
print(test_data.head())
```

Training set:

	radius1	texture1	perimeter1	area1	smoothness1	compactness1	\
72	17.200	24.52	114.20	929.4	0.10710	0.18300	
551	11.130	22.44	71.49	378.4	0.09566	0.08194	
158	12.060	12.74	76.84	448.6	0.09311	0.05241	
424	9.742	19.12	61.93	289.7	0.10750	0.08333	
532	13.680	16.33	87.76	575.5	0.09277	0.07255	

	concavity1	concave_points1	symmetry1	fractal_dimension1	...	\
72	0.169200	0.07944	0.1927	0.06487	...	
551	0.048240	0.02257	0.2030	0.06552	...	
158	0.019720	0.01963	0.1590	0.05907	...	
424	0.008934	0.01967	0.2538	0.07029	...	
532	0.017520	0.01880	0.1631	0.06155	...	

	texture3	perimeter3	area3	smoothness3	compactness3	concavity3	\
72	33.82	151.60	1681.0	0.1585	0.7394	0.65660	
551	28.26	77.80	436.6	0.1087	0.1782	0.15640	
158	18.41	84.08	532.8	0.1275	0.1232	0.08636	
424	23.17	71.79	380.9	0.1398	0.1352	0.02085	
532	20.20	101.60	773.4	0.1264	0.1564	0.12060	

	concave_points3	symmetry3	fractal_dimension3	Naive_Bayes_Predicion
72	0.18990	0.3313	0.13390	
551	0.06413	0.3169	0.08032	
158	0.07025	0.2514	0.07898	
424	0.04589	0.3196	0.08009	
532	0.08704	0.2806	0.07782	

[5 rows x 31 columns]

Test set:

	radius1	texture1	perimeter1	area1	smoothness1	compactness1	\
204	12.47	18.60	81.09	481.9	0.09965	0.1058	
70	18.94	21.31	123.60	1130.0	0.09009	0.1029	
131	15.46	19.48	101.70	748.9	0.10920	0.1223	
431	12.40	17.68	81.47	467.8	0.10540	0.1316	
540	11.54	14.44	74.65	402.9	0.09984	0.1120	

	concavity1	concave_points1	symmetry1	fractal_dimension1	...	\
204	0.08005	0.03821	0.1925	0.06373	...	
70	0.10800	0.07951	0.1582	0.05461	...	
131	0.14660	0.08087	0.1931	0.05796	...	
431	0.07741	0.02799	0.1811	0.07102	...	
540	0.06737	0.02594	0.1818	0.06782	...	

	texture3	perimeter3	area3	smoothness3	compactness3	concavity3	\
204	24.64	96.05	677.9	0.1426	0.2378	0.2671	
70	26.58	165.90	1866.0	0.1193	0.2336	0.2687	
131	26.00	124.90	1156.0	0.1546	0.2394	0.3791	
431	22.91	89.61	515.8	0.1450	0.2629	0.2403	
540	19.68	78.78	457.8	0.1345	0.2118	0.1797	

	concave_points3	symmetry3	fractal_dimension3	Naive_Bayes_Predicion
204	0.10150	0.3014	0.08750	
70	0.17890	0.2551	0.06589	
131	0.15140	0.2837	0.08019	
431	0.07370	0.2556	0.09359	

```
540          0.06918     0.2329              0.08134

[5 rows x 31 columns]
```

In [209]:
```
X
```

Out[209]:

	radius1	texture1	perimeter1	area1	smoothness1	compactness1	concavity1	concave_pc
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.1
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.0
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.1
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.1
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.1
...	...	...	...	...	...	...	...	
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.1
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.0
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.0
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.1
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.0

569 rows × 31 columns

In [210]:
```
y
```

Out[210]:
```
0      M
1      M
2      M
3      M
4      M
      ..
564    M
565    M
566    M
567    M
568    B
Name: diagnosis, Length: 569, dtype: object
```

4. Utilice el archivo para aplicar los siguientes algoritmos de clasificaicón supervisada: NB, SVM, DT, pude utilizar weka o python.
5. Con el archivo de test realice la predicción de los datos con los 3 modelos

In [51]:
```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import svm
from sklearn.tree import DecisionTreeClassifier

"""
A continuacion se va a entrenar con Naive Bayes, para lo cual se envia los dat
"""
nb = GaussianNB()
nb.fit(train_data, train_labels) #con esta funcion se ajusta el modelo a los a
nb_score = nb.score(test_data, test_labels)#con esta funcion se compara las pr
print("Naive Bayes porcentaje:", nb_score)

"""
A continuacion se va a entrenar con Naive Bayes, para lo cual se envia los dat
"""
svm_clf = svm.SVC(kernel='linear')
svm_clf.fit(train_data, train_labels)
svm_score = svm_clf.score(test_data, test_labels)
print("SVM porcentaje:", svm_score)

# Entrenar y probar el clasificador Decision Tree
dt = DecisionTreeClassifier()
dt.fit(train_data, train_labels)
dt_score = dt.score(test_data, test_labels)
print("Decision Tree porcentaje:", dt_score)
```

```
Naive Bayes porcentaje: 0.9655172413793104
SVM porcentaje: 0.9655172413793104
Decision Tree porcentaje: 0.896551724137931
```

6. Agregue una columna por cada modelo denominada predicciones al archivo de test y súbalo al eva

In [111]:
```
test_data
```

Out[111]:

	radius1	texture1	perimeter1	area1	smoothness1	compactness1	concavity1	concave_pc
204	12.470	18.60	81.09	481.9	0.09965	0.10580	0.080050	0.0
70	18.940	21.31	123.60	1130.0	0.09009	0.10290	0.108000	0.0
131	15.460	19.48	101.70	748.9	0.10920	0.12230	0.146600	0.0
431	12.400	17.68	81.47	467.8	0.10540	0.13160	0.077410	0.0
540	11.540	14.44	74.65	402.9	0.09984	0.11200	0.067370	0.0
567	20.600	29.33	140.10	1265.0	0.11780	0.27700	0.351400	0.1
369	22.010	21.90	147.20	1482.0	0.10630	0.19540	0.244800	0.1
29	17.570	15.05	115.00	955.1	0.09847	0.11570	0.098750	0.0
81	13.340	15.86	86.49	520.0	0.10780	0.15350	0.116900	0.0
477	13.900	16.62	88.97	599.4	0.06828	0.05319	0.022240	0.0
457	13.210	25.25	84.10	537.9	0.08791	0.05205	0.027720	0.0
167	16.780	18.80	109.30	886.3	0.08865	0.09182	0.084220	0.0
165	14.970	19.76	95.50	690.2	0.08421	0.05352	0.019470	0.0
329	16.260	21.88	107.50	826.8	0.11650	0.12830	0.179900	0.0
527	12.340	12.27	78.94	468.5	0.09003	0.06307	0.029580	0.0
83	19.100	26.29	129.10	1132.0	0.12150	0.17910	0.193700	0.1
511	14.810	14.70	94.66	680.7	0.08472	0.05016	0.034160	0.0
556	10.160	19.59	64.73	311.7	0.10030	0.07504	0.005025	0.0
101	6.981	13.43	43.79	143.5	0.11700	0.07568	0.000000	0.0
535	20.550	20.86	137.80	1308.0	0.10460	0.17390	0.208500	0.1
73	13.800	15.79	90.43	584.1	0.10070	0.12800	0.077890	0.0
394	12.100	17.72	78.07	446.2	0.10290	0.09758	0.047830	0.0
393	21.610	22.28	144.40	1407.0	0.11670	0.20870	0.281000	0.1
425	10.030	21.28	63.19	307.3	0.08117	0.03912	0.002470	0.0
305	11.600	24.49	74.23	417.2	0.07474	0.05688	0.019740	0.0
76	13.530	10.94	87.91	559.2	0.12910	0.10470	0.068770	0.0
384	13.280	13.72	85.79	541.8	0.08363	0.08575	0.050770	0.0
555	10.290	27.61	65.67	321.4	0.09030	0.07658	0.059990	0.0
362	12.760	18.84	81.87	496.6	0.09676	0.07952	0.026880	0.0

29 rows × 30 columns

In [126]:
```
type(nb.predict(test_data))
```

Out[126]:
```
numpy.ndarray
```

In [127]:
```
type(dt.predict(test_data))
```

Out[127]:
```
numpy.ndarray
```

In [90]:
```
svm_clf.predict(test_data)
```

Out[90]:
```
array(['B', 'M', 'M', 'B', 'B', 'M', 'M', 'M', 'B', 'B', 'B', 'M', 'B',
       'M', 'B', 'M', 'B', 'B', 'B', 'M', 'B', 'B', 'M', 'B', 'B', 'B',
       'B', 'B', 'B'], dtype=object)
```

In [172]:
```
test_data
```

Out[172]:

	radius1	texture1	perimeter1	area1	smoothness1	compactness1	concavity1	concave_pc
204	12.470	18.60	81.09	481.9	0.09965	0.10580	0.080050	0.0
70	18.940	21.31	123.60	1130.0	0.09009	0.10290	0.108000	0.0
131	15.460	19.48	101.70	748.9	0.10920	0.12230	0.146600	0.0
431	12.400	17.68	81.47	467.8	0.10540	0.13160	0.077410	0.0
540	11.540	14.44	74.65	402.9	0.09984	0.11200	0.067370	0.0
567	20.600	29.33	140.10	1265.0	0.11780	0.27700	0.351400	0.1
369	22.010	21.90	147.20	1482.0	0.10630	0.19540	0.244800	0.1
29	17.570	15.05	115.00	955.1	0.09847	0.11570	0.098750	0.0
81	13.340	15.86	86.49	520.0	0.10780	0.15350	0.116900	0.0
477	13.900	16.62	88.97	599.4	0.06828	0.05319	0.022240	0.0
457	13.210	25.25	84.10	537.9	0.08791	0.05205	0.027720	0.0
167	16.780	18.80	109.30	886.3	0.08865	0.09182	0.084220	0.0
165	14.970	19.76	95.50	690.2	0.08421	0.05352	0.019470	0.0
329	16.260	21.88	107.50	826.8	0.11650	0.12830	0.179900	0.0
527	12.340	12.27	78.94	468.5	0.09003	0.06307	0.029580	0.0
83	19.100	26.29	129.10	1132.0	0.12150	0.17910	0.193700	0.1
511	14.810	14.70	94.66	680.7	0.08472	0.05016	0.034160	0.0
556	10.160	19.59	64.73	311.7	0.10030	0.07504	0.005025	0.0
101	6.981	13.43	43.79	143.5	0.11700	0.07568	0.000000	0.0
535	20.550	20.86	137.80	1308.0	0.10460	0.17390	0.208500	0.1
73	13.800	15.79	90.43	584.1	0.10070	0.12800	0.077890	0.0
394	12.100	17.72	78.07	446.2	0.10290	0.09758	0.047830	0.0
393	21.610	22.28	144.40	1407.0	0.11670	0.20870	0.281000	0.1
425	10.030	21.28	63.19	307.3	0.08117	0.03912	0.002470	0.0
305	11.600	24.49	74.23	417.2	0.07474	0.05688	0.019740	0.0
76	13.530	10.94	87.91	559.2	0.12910	0.10470	0.068770	0.0
384	13.280	13.72	85.79	541.8	0.08363	0.08575	0.050770	0.0
555	10.290	27.61	65.67	321.4	0.09030	0.07658	0.059990	0.0
362	12.760	18.84	81.87	496.6	0.09676	0.07952	0.026880	0.0

29 rows × 30 columns

In [174]:
```
test_data.to_csv("test.csv", index=False)
```

In [175]:
```
#Leemos el archivo descargado con extension .data
datos_test =  pd.read_csv('test.csv',delimiter=',')
datos_test
```

Out[175]:

nmetry1	fractal_dimension1	...	radius3	texture3	perimeter3	area3	smoothness3	compactness3
0.1925	0.06373	...	14.97	24.64	96.05	677.9	0.14260	0.2378(
0.1582	0.05461	...	24.86	26.58	165.90	1866.0	0.11930	0.2336(
0.1931	0.05796	...	19.26	26.00	124.90	1156.0	0.15460	0.2394(
0.1811	0.07102	...	12.88	22.91	89.61	515.8	0.14500	0.2629(
0.1818	0.06782	...	12.26	19.68	78.78	457.8	0.13450	0.2118(
0.2397	0.07016	...	25.74	39.42	184.60	1821.0	0.16500	0.8681(
0.1824	0.06140	...	27.66	25.80	195.00	2227.0	0.12940	0.3885(
0.1739	0.06149	...	20.01	19.52	134.90	1227.0	0.12550	0.2812(
0.1942	0.06902	...	15.53	23.19	96.66	614.9	0.15360	0.4791(
0.1813	0.05536	...	15.14	21.80	101.20	718.9	0.09384	0.2006(
0.1619	0.05584	...	14.35	34.23	91.29	632.9	0.12890	0.1063(
0.1893	0.05534	...	20.05	26.30	130.70	1260.0	0.11680	0.2119(
0.1515	0.05266	...	15.98	25.82	102.30	782.1	0.10450	0.0999[
0.1869	0.06532	...	17.73	25.21	113.70	975.2	0.14260	0.2116(
0.1689	0.05808	...	13.61	19.27	87.22	564.9	0.12920	0.2074(
0.1634	0.07224	...	20.33	32.72	141.30	1298.0	0.13920	0.2817(
0.1659	0.05348	...	15.61	17.58	101.70	760.2	0.11390	0.1011(
0.1791	0.06331	...	10.65	22.88	67.88	347.3	0.12650	0.1200(
0.1930	0.07818	...	7.93	19.54	50.41	185.2	0.15840	0.1202(
0.2127	0.06251	...	24.30	25.48	160.20	1809.0	0.12680	0.3135(
0.1662	0.06566	...	16.57	20.86	110.30	812.4	0.14110	0.3542(
0.1937	0.06161	...	13.56	25.80	88.33	559.5	0.14320	0.1773(
0.2162	0.06606	...	26.23	28.74	172.00	2081.0	0.15020	0.5717(
0.1630	0.06439	...	11.11	28.94	69.92	376.3	0.11260	0.0709[
0.1935	0.05878	...	12.44	31.62	81.39	476.5	0.09545	0.1361(
0.2403	0.06641	...	14.08	12.49	91.36	605.5	0.14510	0.1379(
0.1617	0.05594	...	14.24	17.37	96.59	623.7	0.11660	0.2685(
0.1593	0.06127	...	10.84	34.91	69.57	357.6	0.13840	0.1710(
0.1759	0.06183	...	13.75	25.99	87.82	579.7	0.12980	0.1839(

In [176]:
```
datos_test['SVM_Predicion'] = dt.predict(test_data)
```

In [177]:
```
datos_test['Naive_Bayes_Predicion'] = nb.predict(test_data)
```

In [178]:
```
datos_test["Decision_Tree_Predicion"] = svm_clf.predict(test_data)
```

In [179]:
```
datos_test.to_csv("test.csv", index=False)
```

Parte 2

1. Realice una análisis de correlación de las varibles del archivo completo descargado.
2. Utilice las 5 variables que más se correlacionan con el diagnóstico de cancer.
3. Con los registros de las 5 features aplique nuevamente los algoritmos NB, SVM, DT, en weka o python. Debe utilizar el archivo de train.
4. Ahora realice la predicción con los 3 modelos generados y el archivo de test.
5. Agregue una columna por cada modelo denominada predicciones al archivo de test y súbalo al eva.

In [156]:
```
#Leemos el archivo descargado con extension .data
data2 =  pd.read_csv('wdbc.data',delimiter=',',header=None)
data2.head()
```

Out[156]:

	0	1	2	3	4	5	6	7	8	9	...	22	23
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	25.38	17.33
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	24.99	23.41
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	23.57	25.53
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	14.91	26.50
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	22.54	16.67

5 rows × 32 columns

In [157]:
```
#Asiganmos el nombre a las columnas con la documentacion de la pagina https://
data2.columns =['id','diagnosis','radius1','texture1',
                'perimeter1','area1','smoothness1','compactness1'
                ,'concavity1','concave_points1','symmetry1'
                ,'fractal_dimension1','radius2','texture2','perimeter2',
               'area2','smoothness2','compactness2','concavity2',
                'concave_points2','symmetry2','fractal_dimension2','radius3',
               'texture3','perimeter3','area3','smoothness3','compactness3'
               ,'concavity3','concave_points3','symmetry3','fractal_dimension3
```

In [159]:
```
columnas_seleccionadas = ['perimeter3','radius3','perimeter1','area1','concave
data2 = data2.loc[:, columnas_seleccionadas]
```

In [160]:
```
data2
```

Out[160]:

	perimeter3	radius3	perimeter1	area1	concave_points1
0	184.60	25.380	122.80	1001.0	0.14710
1	158.80	24.990	132.90	1326.0	0.07017
2	152.50	23.570	130.00	1203.0	0.12790
3	98.87	14.910	77.58	386.1	0.10520
4	152.20	22.540	135.10	1297.0	0.10430
...	...	...	...	...	...
564	166.10	25.450	142.00	1479.0	0.13890
565	155.00	23.690	131.20	1261.0	0.09791
566	126.70	18.980	108.30	858.1	0.05302
567	184.60	25.740	140.10	1265.0	0.15200
568	59.16	9.456	47.92	181.0	0.00000

569 rows × 5 columns

In [227]:
```
X = data2.iloc[:, :]  # La variable X contiene los datos desde el radio hasta
```

In [226]:
```
y
```

Out[226]:
```
0      M
1      M
2      M
3      M
4      M
      ..
564    M
565    M
566    M
567    M
568    B
Name: diagnosis, Length: 569, dtype: object
```

In [228]:
```
X
```

Out[228]:

	perimeter3	radius3	perimeter1	area1	concave_points1
0	184.60	25.380	122.80	1001.0	0.14710
1	158.80	24.990	132.90	1326.0	0.07017
2	152.50	23.570	130.00	1203.0	0.12790
3	98.87	14.910	77.58	386.1	0.10520
4	152.20	22.540	135.10	1297.0	0.10430
...	...	...	...	...	...
564	166.10	25.450	142.00	1479.0	0.13890
565	155.00	23.690	131.20	1261.0	0.09791
566	126.70	18.980	108.30	858.1	0.05302
567	184.60	25.740	140.10	1265.0	0.15200
568	59.16	9.456	47.92	181.0	0.00000

569 rows × 5 columns

In [229]:
```
train_data1, test_data1, train_labels1, test_labels1 = train_test_split(X, y,
```

In [231]:
```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import svm
from sklearn.tree import DecisionTreeClassifier

nb = GaussianNB()
nb.fit(train_data1, train_labels1)
nb_score = nb.score(test_data1, test_labels1)
print("Naive Bayes porcentaje:", nb_score)


svm_clf = svm.SVC(kernel='linear')
svm_clf.fit(train_data1, train_labels1)
svm_score = svm_clf.score(test_data1, test_labels1)
print("SVM porcentaje:", svm_score)

# Entrenar y probar el clasificador Decision Tree
dt = DecisionTreeClassifier()
dt.fit(train_data1, train_labels1)
dt_score = dt.score(test_data1, test_labels1)
print("Decision Tree porcentaje:", dt_score)
```

```
Naive Bayes porcentaje: 0.8275862068965517
SVM porcentaje: 0.8620689655172413
Decision Tree porcentaje: 0.896551724137931
```

In [232]:
```
dt.predict(test_data1)
```

Out[232]:
```
array(['B', 'B', 'B', 'B', 'B', 'B', 'B', 'M', 'B', 'M', 'B', 'M', 'M',
       'M', 'M', 'B', 'B', 'B', 'B', 'B', 'B', 'M', 'B', 'B', 'B', 'B',
       'B', 'M', 'B'], dtype=object)
```

In [233]:
```
svm_clf.predict(test_data1)
```

Out[233]:
```
array(['B', 'B', 'B', 'B', 'B', 'B', 'M', 'M', 'B', 'M', 'B', 'M', 'B',
       'M', 'M', 'B', 'B', 'B', 'B', 'M', 'B', 'M', 'B', 'B', 'B', 'B',
       'B', 'M', 'B'], dtype=object)
```

In [234]:
```
nb.predict(test_data1)
```

Out[234]:
```
array(['B', 'B', 'B', 'B', 'B', 'B', 'B', 'M', 'B', 'M', 'B', 'M', 'B',
       'M', 'M', 'B', 'B', 'B', 'B', 'M', 'B', 'M', 'B', 'B', 'B', 'B',
       'B', 'M', 'B'], dtype='<U1')
```

In [235]:
```
test_data1.to_csv("test1.csv", index=False)
```

In [236]:
```
datos_test1 =  pd.read_csv('test1.csv',delimiter=',')
datos_test1
```

Out[236]:

	perimeter3	radius3	perimeter1	area1	concave_points1
0	67.03	10.410	60.11	269.4	0.015040
1	75.79	12.020	59.82	278.6	0.014060
2	96.05	14.970	81.09	481.9	0.038210
3	94.17	14.670	88.06	582.7	0.019170
4	76.43	11.620	70.79	365.6	0.027880
5	95.78	14.960	87.76	571.1	0.031600
6	108.60	16.990	88.05	582.7	0.044890
7	184.60	25.380	122.80	1001.0	0.147100
8	58.08	9.092	53.27	203.9	0.021680
9	171.10	25.300	135.90	1264.0	0.150400
10	91.88	14.240	83.19	506.3	0.023150
11	127.10	19.820	114.60	992.1	0.041780
12	106.00	16.010	92.33	595.9	0.067590
13	129.10	19.760	117.40	1024.0	0.057780
14	114.60	17.310	97.40	668.3	0.090290
15	92.80	14.040	82.82	504.8	0.034000
16	78.44	12.360	70.79	386.8	0.006434
17	97.90	15.270	88.37	585.9	0.009937
18	91.11	13.750	78.01	457.9	0.016920
19	115.90	17.710	104.30	800.0	0.045280
20	86.65	13.350	78.83	463.7	0.016540
21	114.10	16.460	98.22	656.1	0.063000
22	96.08	14.670	87.21	571.8	0.017230
23	57.17	9.077	47.98	178.8	0.000000
24	82.74	13.070	76.09	446.0	0.005592
25	86.82	13.720	79.42	491.9	0.005449
26	87.54	13.670	76.53	438.6	0.020080
27	202.40	30.670	155.10	1747.0	0.141000
28	72.42	11.370	69.14	363.7	0.013690

In [237]:
```
datos_test1['SVM_Predicion'] = dt.predict(test_data1)
```

In [238]:
```
datos_test1['Naive_Bayes_Predicion'] = nb.predict(test_data1)
```

In [239]:
```
datos_test1["Decision_Tree_Predicion"] = svm_clf.predict(test_data1)
```

In [240]:
```
datos_test1.to_csv("test1.csv", index=False)
```

Parte 3

1. Realice una análisis de la ejecución de los algoritmos aplicados en las dos partes anteriores y elija cuál de las dos partes es la mejor para aplicar en este caso. Para la explicación debe realizar una comparación de los algoritmos y argumentar porque elije la opción.

En el caso aplicado se pudo evidenciar el uso de los dos algoritmos pero yo personalmente me quedaria con el primero y es porque:

```
* El primero tiene mas argumentos respecto a la variables para determ
inar de mejor manera la deteccion de cancer.
* Para determinar el diagnostico si se aplica el segundo algoritmo se
ra mas reducido ya que tiene menos datos a tomar en cuenta a diferenc
ia del primero que tiene mas argumento y no segmenta cierta informaci
on
* Se puede ver que la tendencia del segundo clasifica con mayor frecu
encia los casos benignos a diferencia de los malignos, esto no es del
todo cierto ya que el dataset original tiene mayor casos malignos que
benignos y esto puede dar a malas interpretaciones
* El dataset del primero obtiene mejores relaciones con el dataset or
iginal a diferencia del segundo.
```