TP 1

Adan Rodriguez

DataSet usado: breast cancer diagnosis

Objetivo: saber si a partir de ciertos detalles del tumor, intentar predecir si un tumor podra ser maligno o benigno

In [257…

```
import pandas as pd
import statsmodels.api as sm
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
```

In [258…

```
df = pd.read_csv('data.csv')
df.head()
```

Out[258]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	s
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	

5 rows × 33 columns

Ahora limpiare el dataset de modo que omita campos como uri, artists_names etc y se persistira solo datos cuantitativos

In [259…

```
df = df.drop(['id'], axis=1)
df.head()
```

Out[259]:

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

5 rows × 32 columns

ya que la ultima columna esta constituida por NaN, la eliminamos del dataset

In [260…
```
df = df.dropna(axis=1)
df.head()
```

Out[260]:

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

5 rows × 31 columns

In [261…
```
'''Vemos si hay datos nulos en nuestro dataset'''
df.isnull().any()
```

Out[261]:
```
diagnosis                  False
radius_mean                False
texture_mean               False
perimeter_mean             False
area_mean                  False
smoothness_mean            False
compactness_mean           False
concavity_mean             False
concave points_mean        False
symmetry_mean              False
fractal_dimension_mean     False
radius_se                  False
texture_se                 False
perimeter_se               False
area_se                    False
smoothness_se              False
compactness_se             False
concavity_se               False
concave points_se          False
symmetry_se                False
fractal_dimension_se       False
radius_worst               False
texture_worst              False
perimeter_worst            False
area_worst                 False
smoothness_worst           False
compactness_worst          False
concavity_worst            False
concave points_worst       False
symmetry_worst             False
fractal_dimension_worst    False
dtype: bool
```

In [262…
```
#Resumen del dataframe
df.describe()
```

Out[262]:

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

8 rows × 30 columns

Ahora traduciremos el valor categorico M o B a uno numerico 1 o 0

In [263…

```python
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder_X = LabelEncoder()

#Transformo la columna
df.iloc[:, 0] = labelencoder_X.fit_transform(df.iloc[:, 0])


df.head()
```

Out[263]:

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

5 rows × 31 columns

In [264…

```python
#Separamos la variable independiente
y = pd.DataFrame(df.iloc[:, 0])
X = df.drop('diagnosis', axis=1)

X_df = X
```

Ahora para ver el p valor de cada campo le agregamos una fila de 1 a la matriz

In [265…

```python
# agregamos la columna de 1 al conjunto X original
X = np.append(arr = np.ones((569,1)).astype(int), values = X, axis = 1)
```

In [266…

```python
def EliminacionBackward(x, sl):
    numVars = len(x[0])
    for i in range(0, numVars):
        regressor_OLS = sm.OLS(y, x.tolist()).fit()
        maxVar = max(regressor_OLS.pvalues)
        if maxVar > sl:
            for j in range(0, numVars - i):
                if (regressor_OLS.pvalues[j] == maxVar):
                    x = np.delete(x, j, 1)
    regressor_OLS.summary()
    return x
```

In [267…

```
#Eliminamos las variables con mayor valor p automaticamente y mostramos e

SL = 0.05
y = y.astype(int)
X_opt = X

X_Modelado = EliminacionBackward(X_opt, SL)

pd.DataFrame(X_Modelado).head()
```

Out[267]:

	0	1	2	3	4	5	6	7	8	9	
0	1.0	17.99	0.27760	0.14710	1.0950	0.006399	0.05373	0.01587	25.38	17.33	2(
1	1.0	20.57	0.07864	0.07017	0.5435	0.005225	0.01860	0.01340	24.99	23.41	1!
2	1.0	19.69	0.15990	0.12790	0.7456	0.006150	0.03832	0.02058	23.57	25.53	1
3	1.0	11.42	0.28390	0.10520	0.4956	0.009110	0.05661	0.01867	14.91	26.50	!
4	1.0	20.29	0.13280	0.10430	0.7572	0.011490	0.05688	0.01885	22.54	16.67	1!

In [268…

```
#Eliminacion hecha a mano para comparar con la eliminacion automatica
X_opt = X[:, [0, 1, 6, 8, 11, 15, 17, 18, 21, 22, 24, 27, 29, 30]]
#Con este X auxiliar veremos y eliminaremos las columnas que menos influy
# X_opt, al inicio, tomará todas las filas y cada una de las columnas del

#convierto columna a tipo numerico
y = y.astype(int)

# tecnica OLS
SL = 0.05
regression_OLS = sm.OLS(endog = y, exog = X_opt.tolist()).fit()
# observar el p valor en el sumario
regression_OLS.summary()
```

Out[268]:

OLS Regression Results

Dep. Variable:	diagnosis	**R-squared:**	0.771
Model:	OLS	**Adj. R-squared:**	0.766
Method:	Least Squares	**F-statistic:**	144.1
Date:	Sat, 27 May 2023	**Prob (F-statistic):**	3.34e-168
Time:	14:34:02	**Log-Likelihood:**	26.008
No. Observations:	569	**AIC:**	-24.02
Df Residuals:	555	**BIC:**	36.80
Df Model:	13		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-2.0783	0.182	-11.449	0.000	-2.435	-1.722
x1	-0.0340	0.015	-2.224	0.027	-0.064	-0.004
x2	-3.3622	0.552	-6.092	0.000	-4.446	-2.278
x3	4.9320	0.943	5.230	0.000	3.080	6.784
x4	0.2147	0.069	3.105	0.002	0.079	0.351
x5	19.6509	4.335	4.533	0.000	11.135	28.167
x6	-3.0397	0.747	-4.067	0.000	-4.508	-1.571
x7	7.7568	3.348	2.317	0.021	1.180	14.334
x8	0.1657	0.021	8.041	0.000	0.125	0.206
x9	0.0102	0.002	5.552	0.000	0.007	0.014
x10	-0.0009	0.000	-7.663	0.000	-0.001	-0.001
x11	0.6533	0.144	4.537	0.000	0.370	0.936
x12	0.7751	0.213	3.634	0.000	0.356	1.194
x13	3.3066	1.139	2.903	0.004	1.070	5.544

Omnibus:	27.517	**Durbin-Watson:**	1.777
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	30.209
Skew:	0.554	**Prob(JB):**	2.76e-07
Kurtosis:	3.218	**Cond. No.**	4.71e+05

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 4.71e+05. This might indicate that there are strong multicollinearity or other numerical problems.

In [269…

```
#Ahora asignamos el dataFrame real con los campos que sabemos que su valo
X_df = X_df.iloc[:, [0, 5, 7, 10, 14, 16, 17, 20, 21, 23, 26, 28, 29]]
```

```
X_df
```

Out[269]:

	radius_mean	compactness_mean	concave points_mean	radius_se	smoothness_se	co
0	17.99	0.27760	0.14710	1.0950	0.006399	
1	20.57	0.07864	0.07017	0.5435	0.005225	
2	19.69	0.15990	0.12790	0.7456	0.006150	
3	11.42	0.28390	0.10520	0.4956	0.009110	
4	20.29	0.13280	0.10430	0.7572	0.011490	
...	...	...	...	...	...	
564	21.56	0.11590	0.13890	1.1760	0.010300	
565	20.13	0.10340	0.09791	0.7655	0.005769	
566	16.60	0.10230	0.05302	0.4564	0.005903	
567	20.60	0.27700	0.15200	0.7260	0.006522	
568	7.76	0.04362	0.00000	0.3857	0.007189	

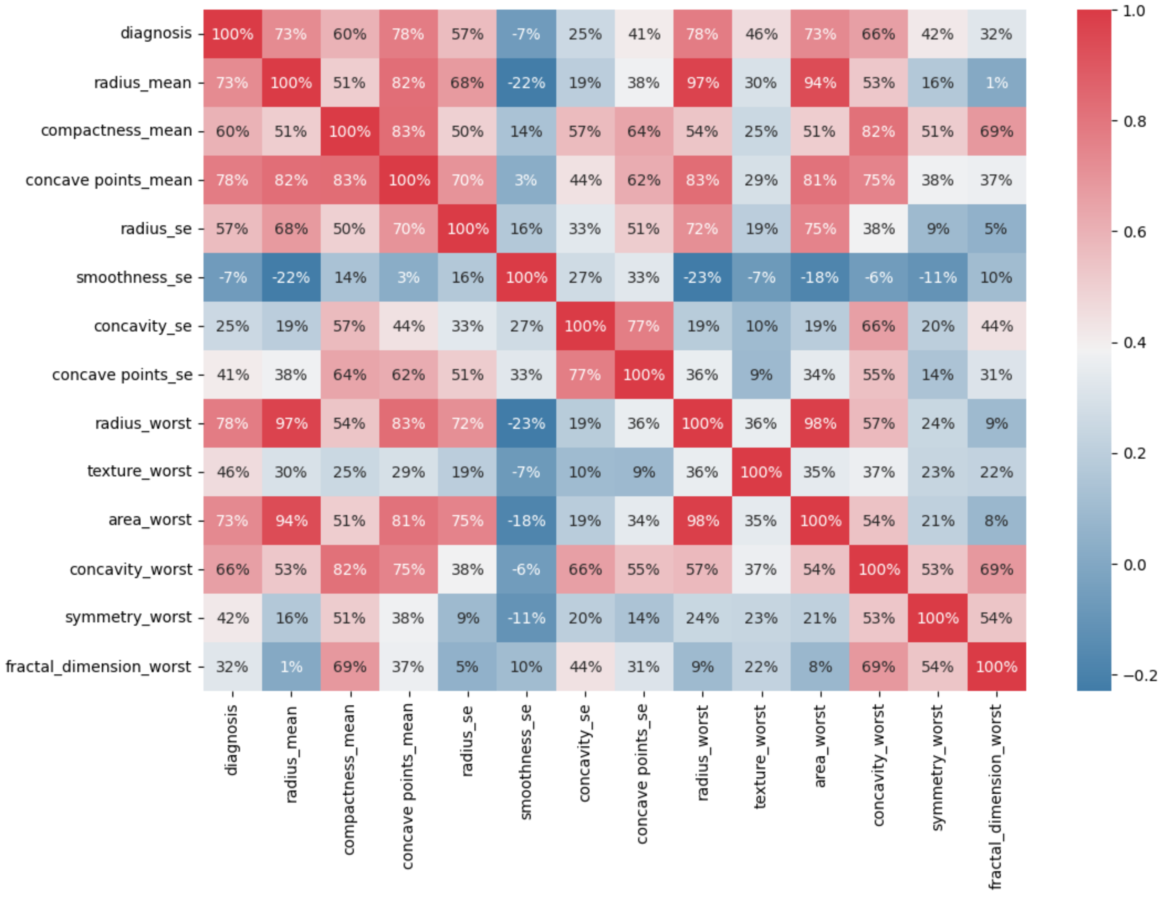
569 rows × 13 columns

In [270…

```
#Se muestra la matriz de correlacion resultante
X_aux = X_df.iloc[:, :]
X_aux.insert(0, 'diagnosis', y)
corr = pd.DataFrame(X_aux).corr()
plt.subplots(figsize=(12,8))
sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns, ann
            cmap=sns.diverging_palette(240, 10, as_cmap=True))
```

Out[270]:

```
<Axes: >
```



```
In [346…  X_aux.describe()
```

Out[346]:

	diagnosis	radius_mean	compactness_mean	concave points_mean	radius_se	smo
count	569.000000	569.000000	569.000000	569.000000	569.000000	
mean	0.372583	14.127292	0.104341	0.048919	0.405172	
std	0.483918	3.524049	0.052813	0.038803	0.277313	
min	0.000000	6.981000	0.019380	0.000000	0.111500	
25%	0.000000	11.700000	0.064920	0.020310	0.232400	
50%	0.000000	13.370000	0.092630	0.033500	0.324200	
75%	1.000000	15.780000	0.130400	0.074000	0.478900	
max	1.000000	28.110000	0.345400	0.201200	2.873000	

Definimos el significado de los campos:

diagnosis: diagnostico final del tumor (M/1: Maligno, B/0: Benigno)

radius_mean: media de las distancias desde el medio a varios puntos del tumor

compactness_mean: el perimetro cuadrado dividido por el area-1

concave points_mean: media del numero de porciones concavas del contorno

radius_se: error estandar de la medida del radio

smoothness_se: variacion local en longitud de radio

concavity_se: severidad de las porciones concavas del contorno

concave points_se: error estandar de las porciones concavas radius_worst: distancia mas amplia entre el centro y algun punto del tumor

texture_worst: peor desviacion de los valores en escala de grises

area_worst: mayor area del tumor

concavity_worst: concavidad mas severa del contorno

symmetry_worst: peor simetria del tumor

fractal_dimension_worst: peor irregularidad de los contornos del tumor

In [271…

```
corr['diagnosis'].abs().sort_values(ascending=False)
```

Out[271]:

```
diagnosis                  1.000000
concave points_mean        0.776614
radius_worst               0.776454
area_worst                 0.733825
radius_mean                0.730029
concavity_worst            0.659610
compactness_mean           0.596534
radius_se                  0.567134
texture_worst              0.456903
symmetry_worst             0.416294
concave points_se          0.408042
fractal_dimension_worst    0.323872
concavity_se               0.253730
smoothness_se              0.067016
Name: diagnosis, dtype: float64
```

Vemos la correlacion entre el diagnostico y las variables

Ahora reacomodamos nuestra matriz:

In [272…

```
#Asigno la matriz resultante a la matriz original de variables dependient
X = X_df
X
```

Out[272]:

	radius_mean	compactness_mean	concave points_mean	radius_se	smoothness_se	co
0	17.99	0.27760	0.14710	1.0950	0.006399	
1	20.57	0.07864	0.07017	0.5435	0.005225	
2	19.69	0.15990	0.12790	0.7456	0.006150	
3	11.42	0.28390	0.10520	0.4956	0.009110	
4	20.29	0.13280	0.10430	0.7572	0.011490	
...	...	...	...	...	...	
564	21.56	0.11590	0.13890	1.1760	0.010300	
565	20.13	0.10340	0.09791	0.7655	0.005769	
566	16.60	0.10230	0.05302	0.4564	0.005903	
567	20.60	0.27700	0.15200	0.7260	0.006522	
568	7.76	0.04362	0.00000	0.3857	0.007189	

569 rows × 13 columns

In [273…

```
#Separamos nuestros conjuntos
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
```

In [274…

```
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

Out[274]:

```
▾ LinearRegression
LinearRegression()
```

In [275…

```
y_pred = regressor.predict(X_test)

#Redondeamos los resultados para tener una respuesta binaria (tambien pod
y_test = np.round(y_test, decimals=0, out=None)
y_pred = np.round(y_pred, decimals=0, out=None)
```

In [328…

```
#Redimensionamos el array
y_test = np.array(y_test).reshape(-1)
y_pred = np.array(y_pred).reshape(-1)

#Cambiamos el tipo de valor a entero
y_pred = y_pred.astype(int)
y_test = y_test.astype(int)

#Creamos el dataframe para mostrar nuestra prediccion
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})

#Reemplazamos los valores numericos por categoricos para mejor visualizac
df['Actual'].replace(to_replace=0, value='B', regex=True, inplace=True)
df['Actual'].replace(to_replace=1, value='M', regex=True, inplace=True)

df['Predicted'].replace(to_replace=0, value='B', regex=True, inplace=True
df['Predicted'].replace(to_replace=1, value='M', regex=True, inplace=True
df['Predicted'].replace(to_replace=2, value='M', regex=True, inplace=True

df_toShow = df.head(25)
df_toShow.head()
```

Out[328]:

	Actual	Predicted
0	M	M
1	B	B
2	B	B
3	B	B
4	B	B

In [330…

```
def entropia(y):
    if isinstance(y, pd.Series):
        a = y.value_counts() / y.shape[0]
        entropy = np.sum(-a * np.log2(a + 1e-9))
        return entropy
    else:
        raise('el obj debe ser una serie Pandas')

entropia(X_aux.diagnosis)
```

```
Out[330]: 0.9526351195164697
```

Segun el calculo de la entropia, al estar cerca del 1 podemos afirmar que los datos varian bastante entre M y B

Ahora, veremos el conteo de diagnosticos con los que queriamos comparar nuestra prediccion

```
In [331… graf1 = pd.crosstab(index = ['Diagnosis'], columns=df['Actual'], margins=
          graf1
```

Out[331]:

Actual	B	M	All
row_0			
Diagnosis	67	47	114
All	67	47	114

Conteo de los resultados de la prediccion

```
In [332… graf2 = pd.crosstab(index = ['Diagnosis'], columns=df['Predicted'], margi
          graf2
```

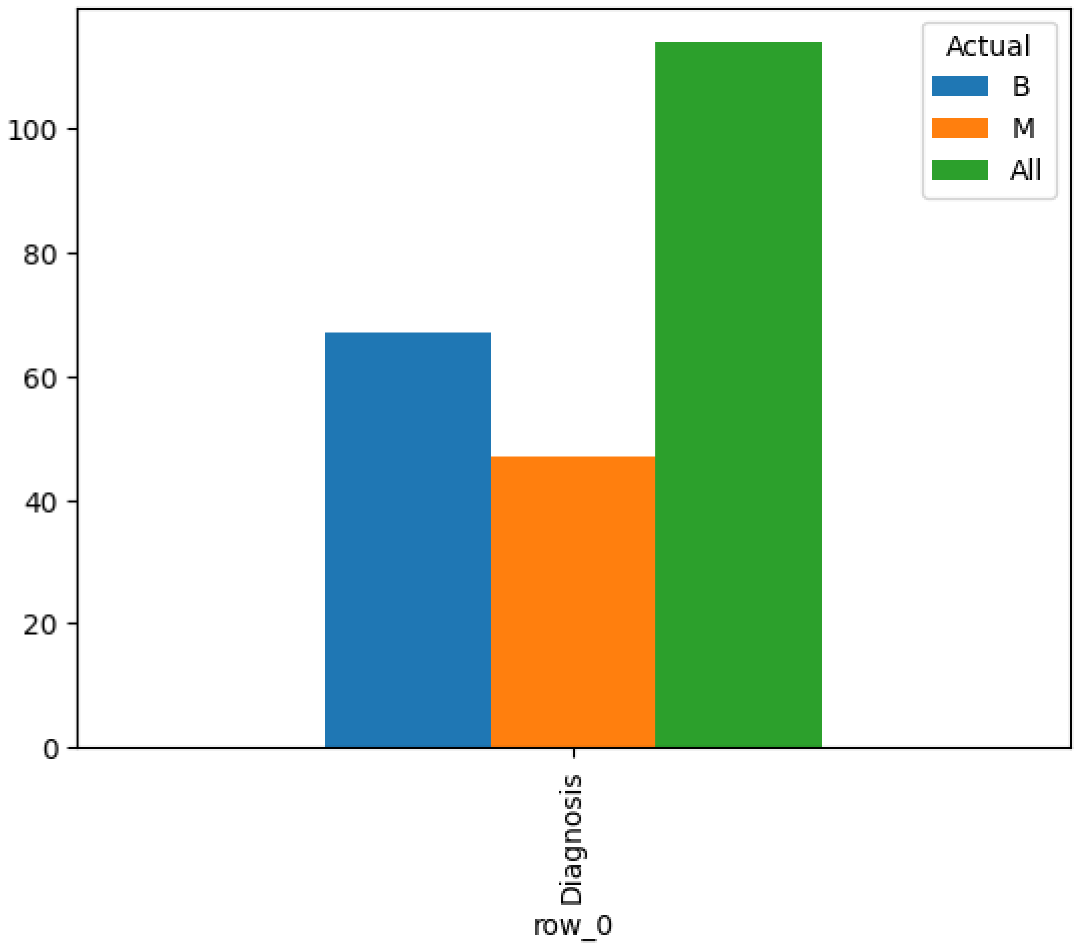
Out[332]:

Predicted	B	M	All
row_0			
Diagnosis	70	44	114
All	70	44	114

Visualizamos los datos

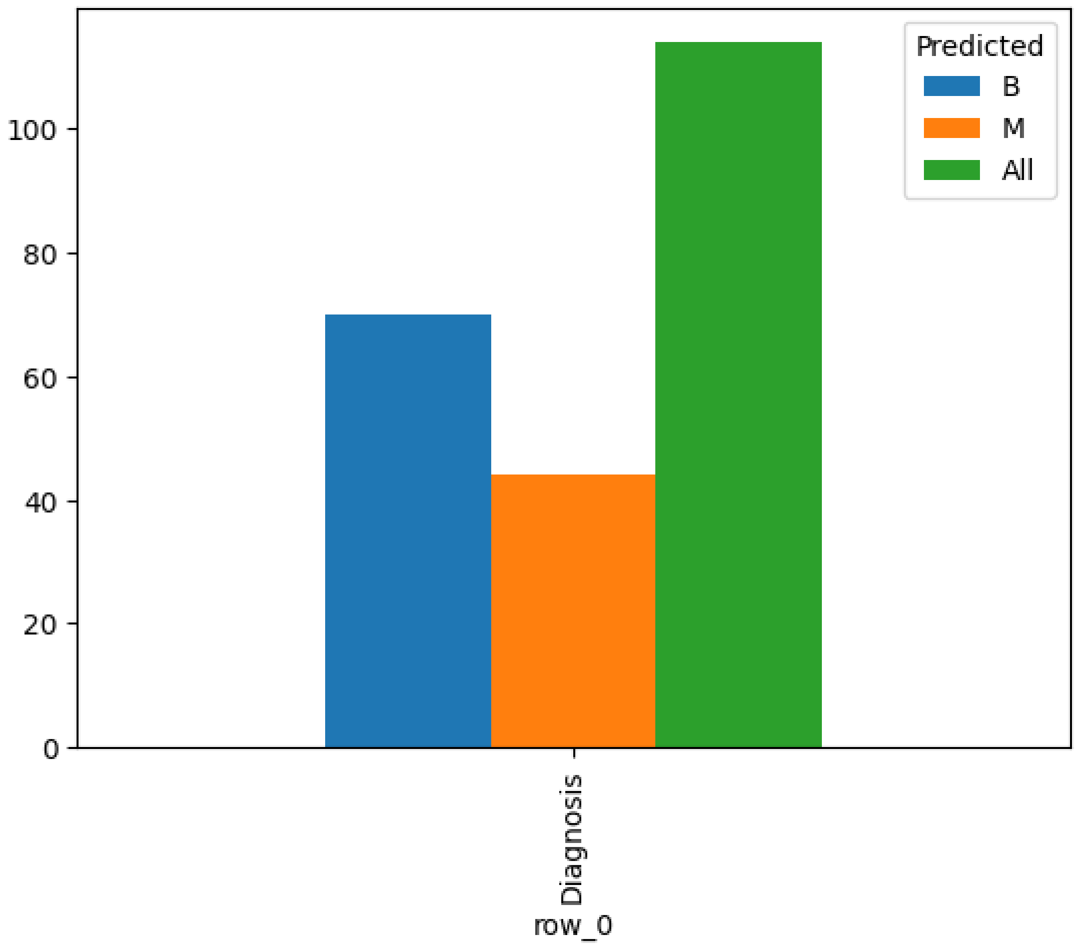
```
In [338… graf1[0:1].plot(kind='bar')
```

```
Out[338]: <Axes: xlabel='row_0'>
```



```
In [342… graf2[0:1].plot(kind='bar')
```

```
Out[342]: <Axes: xlabel='row_0'>
```



Por ultimo calculamos el indice de error del algoritmo

In [343…

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

```
Mean Absolute Error: 0.043859649122807015
Mean Squared Error: 0.043859649122807015
Root Mean Squared Error: 0.20942695414584775
```

Viendo que nuestras metricas son cercanas al cero, podemos tener confianza de que nuestro algoritmo tendrá buenas predicciones

En conclusion, pudimos ver que en el diagnostico cancerigeno de un tumor, antes de hacer una biopsia para saber con certeza la naturaleza del mismo, la concavidad, el radio y el area influye a la hora de saber si es maligno o benigno, por lo que con un algoritmo como el propuesto, podriamos monitorear el cambio de estos valores en algun paciente para saber de antemano que resultados podria esperar, y asi estudiarlo mas fondo en pro de diagnosticarle lo mas pronto posible y tratarle en una etapa temprana