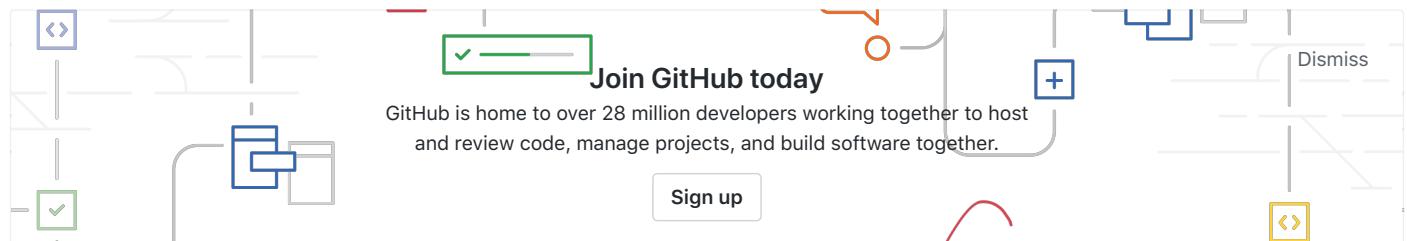


PORTAFOLIO DE EVIDENCIAS

CIENCIA DE DATOS

Mayra Cristina Berrones Reyes

Matricula | 1646291



Branch: master ▾ Ciencia_de_Datos / Practica 1.ipynb

Find file Copy path

mayraberrones94 Correcciones

19c18a8 9 minutes ago

1 contributor

147 lines (146 sloc) | 6.61 KB

MB CZ

Práctica 1: Preparación de Datos.

Introducción

En esta práctica se buscaron datos de la página de [Kaggle](#) tomando el conjunto de datos de Diagnóstico de Cáncer de mama de Wisconsin, Estados Unidos. Estos datos contienen el diagnóstico y una serie de características que se presentan en una imagen digitalizada de una masa en el seno, sin especificar el lado. Estos datos se usan en el ámbito de investigación científica, intentando caracterizar los principales marcadores que permitan separar el diagnóstico de una anomalía benigna a una maligna.

Las preguntas de interés para estudiar este tipo de datos pueden ser las siguientes:

- ¿Cuáles son las características más predominantes de las anomalías malignas?
- ¿Qué marcadores ~~de~~ permiten caracterizar una anomalía maligna?
- ¿Qué marcadores intersectan tanto en las anomalías malignas como en las benignas?
- ¿Cómo se relacionan todas estas características para que el diagnóstico final de la anomalía sea maligno, y cual será la relación con las anomalías benignas?
- Por último, la pregunta que puede resultar de mayor importancia, y a la cual se quiere guiar en esta investigación, es si al utilizar todos estos datos con pacientes no diagnosticados, es posible tener un valor aceptable de exactitud para poder hacer un diagnóstico inicial.

wave

Preparación de datos:

Se tiene una base de datos en formato .CSV, con registro de 532 mujeres que presentan una anomalía en el seno. Cada paciente es identificada por una clave numérica de 5 dígitos, y cualquier rasgo que pueda distinguir identidad no se revela en los datos.

Los datos que se tienen en este conjunto se dividen de la siguiente manera:

Columnas:

1. Id: Número de identificación.
2. Diagnóstico: El diagnóstico oficial del tejido del seno (M es maligno y B es benigno)
3. Radio medio: Media de las distancias desde el centro a los puntos en el perímetro.
4. Textura media: Desviación estándar de los valores de escala de grises.
5. Media del perímetro: Tamaño medio del n úcleo del tumor.
6. Área media.
7. Suavidad media: Es la media de la variación local en longitud de radios.
8. Compacidad media: Media del perímetro al cuadrado, entre el área menos 1.
9. Concavidad media: Media de la severidad de las porciones cóncavas de contorno.
10. Media de puntos cóncavos: Media para el número de porciones cóncavas en el contorno.

11. Media Simétrica
12. Radio-se: Es el error estándar de las distancias medias desde los puntos centrales en el perímetro.
13. Textura-se: Es el error de la desviación estándar de los valores de la escala de grises.
14. Perímetro-se
15. Área-se
16. Suavidad-se: Erros estándar para la variación local de longitud de radios.
17. Compacidad-se: Error estándar para el perímetro al cuadrado entre el área menos uno.
18. Concavidad-se: Error estándar para la severidad de las porciones cóncavas del contorno.
19. Puntos cóncavos-se: Error estándar por el numero de porciones cóncavas del contorno.
20. Simetría-se:
21. Dimensión Fractal-se: Error estándar para la “Aproximacion de costline” menos uno.
22. Peor Radio: Peor o mayor valor de la media de la distancia desde el centro a los puntos en el perímetro.
23. Peor textura: Peor o más grande valor de media para la desviación estándar de los valores de la escala de grises.
24. Peor perímetro
25. Peor área.
26. Peor suavidad: Peor o mayor valor de media para la variación local en el tamaño del radio.
27. Peor compacidad: Peor o mayor valor de media para el perímetro elevado al cuadrado entre el área menos uno.
28. Peor concavidad: Peor o mayor valor de media por la severidad de las partes cóncavas del contorno.
29. Peores puntos cóncavos: Peor o mas grande valor de la media por el número de porciones cóncavas del contorno.
30. Peor simetría.
31. Peor dimensión fractal: Peor o mayor valor de media para la aproximación “costline” menos uno.

Se conoce además que todos los datos están puestos hasta cuatro dígitos de significancia. Su Distribucion de clase es de 357 benignos y 212 malignos.

Los siguientes pasos se realizan con la finalidad de ordenar correctamente los datos y facilitar su uso a nivel estadístico.

Se inicia realizando una copia del archivo csv, que es llamado “w-data.csv”.

```
In [ ]: cp w-data.csv bp_w-data.csv
```

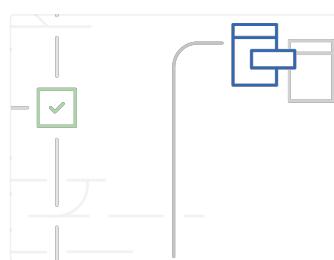
En esta copia se realizo una búsqueda para saber si en alguna de las columnas del archivo existe alguna celda que este vacía. Si este fuera el caso, se reemplaza la celda vacía con un NE, para No Existe.

```
In [ ]: sed 's/, $/&NE/' bp_w-data.csv
```

Todos los datos de la tabla son numéricos a excepción de la columna del diagnóstico, y con línea de código siguiente:

```
In [ ]: awk -F',' '{print $2}' bp_w-data.csv | sort | uniq -c
1 "diagnosis"
357 B
212 M
```

Podemos ver que las únicas dos opciones que existen en el diagnóstico es benigno o maligno, y nos comprueba que se tienen 357 casos de anomalías benignas y 212 casos de anomalías malignas.



Join GitHub today

GitHub is home to over 31 million developers working together to host and review code, manage projects, and build software together.

[Sign up](#)

Dismiss



[Find file](#) [Copy path](#)

Branch: master [Ciencia_de_Datos](#) / Práctica 2.ipynb

mayraberrones94 Cambios

18f6ffb 2 days ago

1 contributor

1600 lines (1599 sloc) | 59.8 KB

Práctica 2: Lectura y manipulación de datos en Pandas.

Pandas

El nombre de Pandas se deriva de las palabras "Panel Data" (Panel de datos) un término para conjuntos estructurados multidimensionales de datos. Esta es una librería disponible gratuita que ayuda analizar datos en conjunto con el lenguaje de programación de Python.

Metas de la práctica:

1. Leer data frames de archivos
2. Escribir data frames en CSV
3. Agregar columnas en data frames
4. Combinar dos o más data frames
5. Filtrar renglones de un data frame

Paso 1: Limpiar datos

Uno de los pasos más importantes al trabajar con datos es cerciorarse de que estos estén listos para cualquier manipulación que se les requiera hacer. En la práctica anterior se trabajó con la herramienta de bash para limpiar nuestros datos. En este caso, se utiliza la librería de Pandas para hacer esta tarea.

Primeramente, importamos la librería de Pandas y cargamos nuestro archivo csv.

In [1]: `import pandas as pd`

Una vez que se tiene importada la librería de Pandas, para poder leer los datos de entrada, se tiene que dar la dirección del directorio en el cual se encuentra el archivo .CSV.

In [2]: `data = pd.read_csv("/Users/mayraberrones/Documents/GitHub/Ciencia_de_Datos/w-data.csv")`

Como se realizó en la práctica anterior, hay que cerciorarse de que los datos están completos, y que no hay celdas en ninguna de las columnas en las cuales faltan datos. Para esto se utiliza el comando `isnull()`.

In [3]: `data.isnull().sum()`

Out[3]: `id` 0
..... ^

```

diagnosis          0
radius_mean        0
texture_mean       0
perimeter_mean    0
area_mean          0
smoothness_mean   0
compactness_mean  0
concavity_mean    0
concave_points_mean 0
symmetry_mean     0
fractal_dimension_mean 0
radius_se          0
texture_se         0
perimeter_se      0
area_se            0
smoothness_se     0
compactness_se    0
concavity_se      0
concave_points_se 0
symmetry_se       0
fractal_dimension_se 0
radius_worst       0
texture_worst      0
perimeter_worst   0
area_worst         0
smoothness_worst  0
compactness_worst 0
concavity_worst   0
concave_points_worst 0
symmetry_worst    0
fractal_dimension_worst 0
Unnamed: 32         569
dtype: int64

```

Con el comando de `IsNull()` podemos ver si en alguna de las columnas existe una celda que no tenga ningún valor. En el caso del `sum()` complementario, es para evitar que imprima en el Output todo el csv y en su lugar dé la suma de las celdas que se encuentran vacias dentro de cada una de las columnas que conforman el archivo.

Paso 2: Separar los datos.

Uno de los objetivos que se tienen en este trabajo es el de encontrar relaciones entre la información de anomalías benignas y malignas. Para esto se quiere sacar primero información por separado de ambas anomalías para poder compararlas.

En el código siguiente se lee el archivo csv, y en este se toman los casos que en el diagnóstico se presentan como malignos, y se copian a un archivo .CSV nuevo, llamado Malignant-data.csv.

```
In [4]: reader = pd.read_csv('w-data.csv')
writer = reader[reader['diagnosis']=='M']
writer.to_csv('Malignant-data.csv', index=False)
```

Se realiza el mismo procedimiento para los datos de anomalías benignas, y se guardan en un archivo llamado Benign-data.csv.

```
In [ ]: writer = reader[reader['diagnosis']=='B']
writer.to_csv('Benign-data.csv', index=False)
```

Una vez que se tienen estos dos archivos por separado, se mandan a llamar de nuevo con el comando de `read`, para poder iniciar a manipularlos.

```
In [6]: data1 = pd.read_csv("/Users/mayraberrones/Documents/GitHub/Ciencia_de_Datos/Malignant-data.csv")
data2 = pd.read_csv("/Users/mayraberrones/Documents/GitHub/Ciencia_de_Datos/Benign-data.csv")
```

Una vez que se tienen los documentos separados, podemos ver con la función `Shape` de cuantas filas y columnas están conformados cada uno, con esto se cerciora de que se hayan copiado el número de elementos correctos a cada uno de los archivos.

```
In [7]: data1.shape
Out[7]: (212, 33)
```

```
In [8]: data2.shape
```

```
Out[8]: (357, 33)
```

Paso 3: Extraer información.

Ya que se tienen los archivos separados, en pandas existe la función de `Describe()`, la cual nos da un resumen de la información que se tiene en el .csv. Con esta función se pueden ver rápidamente los siguientes elementos:

- Count: El conteo de valores existentes en la columna, excluyendo los que NaN (Sin contenido)
- Mean: Promedio de todos los valores numéricos de la columna.
- Std: Desviación estandar de todos los valores de la columna.
- Min: Valor numérico mínimo encontrado en la columna.
- 25, 50, 75: En este caso estos son los valores dados por default, pero pueden cambiarse dentro del comando si se requiere. Representan diferentes cuartiles de todos los valores numericos en la columna.
- Max: Valor numérico máximo encontrado en la columna.

```
In [10]: data1.describe()
```

```
Out[10]:
```

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dimension_mean
count	2.120000e+02	212.000000	212.000000	212.000000	212.000000	212.000000	212.000000	212.000000	212.000000	212.000000	212.000000
mean	3.681805e+07	17.462830	21.604906	115.365377	978.376415	0.102898	0.145188	0.102898	0.145188	0.145188	0.145188
std	1.378965e+08	3.203971	3.779470	21.854653	367.937978	0.012608	0.053987	0.012608	0.053987	0.053987	0.053987
min	8.670000e+03	10.950000	10.380000	71.900000	361.600000	0.073710	0.046050	0.073710	0.046050	0.046050	0.046050
25%	8.613450e+05	15.075000	19.327500	98.745000	705.300000	0.094010	0.109600	0.094010	0.109600	0.109600	0.109600
50%	8.953665e+05	17.325000	21.460000	114.200000	932.000000	0.102200	0.132350	0.102200	0.132350	0.132350	0.132350
75%	8.911290e+06	19.590000	23.765000	129.925000	1203.750000	0.110925	0.172400	0.110925	0.172400	0.172400	0.172400
max	9.112962e+08	28.110000	39.280000	188.500000	2501.000000	0.144700	0.345400	0.144700	0.345400	0.345400	0.345400

8 rows × 32 columns

```
In [11]: data2.describe()
```

```
Out[11]:
```

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dimension_mean
count	3.570000e+02	357.000000	357.000000	357.000000	357.000000	357.000000	357.000000	357.000000	357.000000	357.000000	357.000000
mean	2.654382e+07	12.146524	17.914762	78.075406	462.790196	0.092478	0.080085	0.092478	0.080085	0.080085	0.080085
std	1.167397e+08	1.780512	3.995125	11.807438	134.287118	0.013446	0.033750	0.013446	0.033750	0.033750	0.033750
min	8.913000e+03	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.052630	0.019380	0.019380	0.019380
25%	8.746620e+05	11.080000	15.150000	70.870000	378.200000	0.083060	0.055620	0.083060	0.055620	0.055620	0.055620
50%	9.089160e+05	12.200000	17.390000	78.180000	458.400000	0.090760	0.075290	0.090760	0.075290	0.075290	0.075290
75%	8.812816e+06	13.370000	19.760000	86.100000	551.100000	0.100700	0.097550	0.100700	0.097550	0.097550	0.097550
max	9.113205e+08	17.850000	33.810000	114.600000	992.100000	0.163400	0.223900	0.163400	0.223900	0.223900	0.223900

8 rows × 32 columns

Estos datos van a servir más adelante cuando se analicen más a detalle. Por el momento, se guardan en archivos `Describe1.csv` y `Describe2.csv`

```
In [9]: des1 = data1.describe()
des2 = data2.describe()
des1.to_csv("Describe1.csv", encoding='utf-8', index=True)
des2.to_csv("Describe2.csv", encoding='utf-8', index=True)
```

En caso de requerir la información toda junta, se hace un sólo archivo con la descripción de las anomalías malignas y benignas en el

archivo Concatenar.csv

```
In [12]: conc = des1.append(des2)
conc.to_csv("Concatenar.csv", encoding='utf-8', index=True)

In [13]: resultado = pd.read_csv("/Users/mayraberrones/Documents/GitHub/Ciencia_de_Datos/Concatenar.csv")

In [14]: resultado.shape

Out[14]: (16, 33)
```

```
In [15]: resultado.head(8)
```

```
Out[15]:
```

	Unnamed: 0	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_i
0	count	2.120000e+02	212.000000	212.000000	212.000000	212.000000	212.000000	212.000000
1	mean	3.681805e+07	17.462830	21.604906	115.365377	978.376415	0.102898	0.145188
2	std	1.378965e+08	3.203971	3.779470	21.854653	367.937978	0.012608	0.053987
3	min	8.670000e+03	10.950000	10.380000	71.900000	361.600000	0.073710	0.046050
4	25%	8.613450e+05	15.075000	19.327500	98.745000	705.300000	0.094010	0.109600
5	50%	8.953665e+05	17.325000	21.460000	114.200000	932.000000	0.102200	0.132350
6	75%	8.911290e+06	19.590000	23.765000	129.925000	1203.750000	0.110925	0.172400
7	max	9.112962e+08	28.110000	39.280000	188.500000	2501.000000	0.144700	0.345400

8 rows × 33 columns

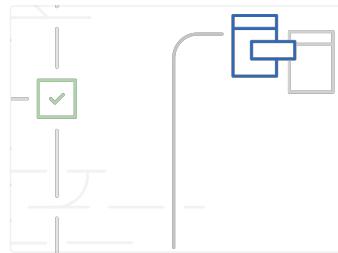
```
In [16]: resultado.tail(8)
```

```
Out[16]:
```

	Unnamed: 0	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_i
8	count	3.570000e+02	357.000000	357.000000	357.000000	357.000000	357.000000	357.000000
9	mean	2.654382e+07	12.146524	17.914762	78.075406	462.790196	0.092478	0.080085
10	std	1.167397e+08	1.780512	3.995125	11.807438	134.287118	0.013446	0.033750
11	min	8.913000e+03	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380
12	25%	8.746620e+05	11.080000	15.150000	70.870000	378.200000	0.083060	0.055620
13	50%	9.089160e+05	12.200000	17.390000	78.180000	458.400000	0.090760	0.075290
14	75%	8.812816e+06	13.370000	19.760000	86.100000	551.100000	0.100700	0.097550
15	max	9.113205e+08	17.850000	33.810000	114.600000	992.100000	0.163400	0.223900

8 rows × 33 columns

Por último, se tiene la función de Tail y Head, que solo son utilizadas aquí para demostrar que el archivo de Concatenar.csv se guardó con los datos adecuadamente.



Join GitHub today

GitHub is home to over 31 million developers working together to host and review code, manage projects, and build software together.

[Sign up](#)

Dismiss



Branch: master ▾ Ciencia_de_Datos / Práctica 3.ipynb

mayraberrones94 cambios

b00f604 16 hours ago

1 contributor

[Find file](#) [Copy path](#)

2296 lines (2295 sloc) | 95 KB



Práctica 3: Estadística descriptiva básica

Como objetivo de esta práctica se tiene sacar algunos elementos básicos de información estadística de los datos, como por ejemplo, conteos, promedios, unas cuantas correlaciones, junto con limpieza, categorización y cuantificación de la información.

En la [Práctica 2](#) se realizaron algunas de estas opciones por medio del comando de `Describe()` de la herramienta de Pandas, pero existen opciones separadas a este comando, las cuales se describen a continuación.

Primeramente, se vuelve a importar la librería de Pandas, junto con el archivo CSV con el que se está trabajando.

```
In [1]: import pandas as pd  
data = pd.read_csv("/Users/mayraberrones/Documents/GitHub/Ciencia_de_Datos/w-data.csv")
```

Ya que se tiene el archivo cargado, se sacará el promedio de cada una de las columnas. Para esto hay que asegurarse de que los valores de las columnas sean todos sean de la misma categoría.

```
In [2]: data.dtypes
```

```
Out[2]: 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

```

Con el Output se puede ver que los únicos valores de columnas que son diferentes son el de la columna de ID y el de Diagnóstico.
Las demás columnas se representan por valores flotantes.

Se sacan ahora los promedios por columnas.

```
In [19]: columnas = data.columns.tolist()
for col in columnas:
    if col != "id":
        if col != "diagnosis":
            print("Promedio de <{0}>: {1}".format(col, data[col].mean()))

Promedio de <radius_mean>: 14.127291739894552
Promedio de <texture_mean>: 19.289648506151142
Promedio de <perimeter_mean>: 91.96903339191564
Promedio de <area_mean>: 654.8891036906855
Promedio de <smoothness_mean>: 0.09636028119507908
Promedio de <compactness_mean>: 0.10434098418277679
Promedio de <concavity_mean>: 0.0887993158172232
Promedio de <concave points_mean>: 0.04891914586994728
Promedio de <symmetry_mean>: 0.18116186291739894
Promedio de <fractal_dimension_mean>: 0.06279760984182776
Promedio de <radius_se>: 0.40517205623901575
Promedio de <texture_se>: 1.2168534270650264
Promedio de <perimeter_se>: 2.866059226713533
Promedio de <area_se>: 40.337079086116
Promedio de <smoothness_se>: 0.007040978910369067
Promedio de <compactness_se>: 0.025478138840070295
Promedio de <concavity_se>: 0.03189371634446397
Promedio de <concave points_se>: 0.011796137082601054
Promedio de <symmetry_se>: 0.02054229876977153
Promedio de <fractal_dimension_se>: 0.0037949038664323374
Promedio de <radius_worst>: 16.269189806678387
Promedio de <texture_worst>: 25.677223198594024
Promedio de <perimeter_worst>: 107.26121265377857
Promedio de <area_worst>: 880.5831282952548
Promedio de <smoothness_worst>: 0.13236859402460457
Promedio de <compactness_worst>: 0.25426504393673116
Promedio de <concavity_worst>: 0.27218848330404216
Promedio de <concave points_worst>: 0.11460622319859401
Promedio de <symmetry_worst>: 0.2900755711775044
Promedio de <fractal_dimension_worst>: 0.08394581722319859
Promedio de <Unnamed: 32>: nan
```

Se sabe que en el caso de la columna del ID no es necesario sacar el promedio, ya que no tiene significancia la información, y en la columna de Diagnóstico es de tipo objeto, por lo que no es posible promediar. En tales casos, se excluyen dichas columnas.

Se puede repetir este proceso para la varianza, y la desviación estándar.

```
In [20]: columnas = data.columns.tolist()
for col in columnas:
    if col != "id":
        if col != "diagnosis":
            print("Varianza de <{0}>: {1}".format(col, data[col].var()))

Varianza de <radius_mean>: 12.418920129526722
Varianza de <texture_mean>: 18.49890867905146
Varianza de <perimeter_mean>: 590.4404795217704
Varianza de <area_mean>: 123843.55431768115
Varianza de <smoothness_mean>: 0.00019779970027290278
Varianza de <compactness_mean>: 0.0027891874004381295
Varianza de <concavity_mean>: 0.006355247900423129
Varianza de <concave points mean>: 0.0015056607691635436
```

```

----- -----
Varianza de <symmetry_mean>: 0.0007515428211713162
Varianza de <fractal_dimension_mean>: 4.984872279821283e-05
Varianza de <radius_se>: 0.07690235187622219
Varianza de <texture_se>: 0.304315949077143
Varianza de <perimeter_se>: 4.08789583770081
Varianza de <area_se>: 2069.4315828687345
Varianza de <smoothness_se>: 9.015114003075571e-06
Varianza de <compactness_se>: 0.00032070288676061906
Varianza de <concavity_se>: 0.0009111982378230953
Varianza de <concave points_se>: 3.80724191290626e-05
Varianza de <symmetry_se>: 6.833289825212876e-05
Varianza de <fractal_dimension_se>: 7.001691562872347e-06
Varianza de <radius_worst>: 23.360224175177606
Varianza de <texture_worst>: 37.77648276875665
Varianza de <perimeter_worst>: 1129.1308469423748
Varianza de <area_worst>: 324167.38510216837
Varianza de <smoothness_worst>: 0.0005213198325267952
Varianza de <compactness_worst>: 0.024754770743704052
Varianza de <concavity_worst>: 0.04352409045926073
Varianza de <concave points_worst>: 0.004320740679099743
Varianza de <symmetry_worst>: 0.003827583539505929
Varianza de <fractal_dimension_worst>: 0.00032620937824822397
Varianza de <Unnamed: 32>: nan

```

```
In [21]: columnas = data.columns.tolist()
for col in columnas:
    if col != "id":
        if col != "diagnosis":
            print("Desviación Estándar de <{0}>: {1}".format(col, data[col].std()))
```

```

Desviación Estándar de <radius_mean>: 3.5240488262120775
Desviación Estándar de <texture_mean>: 4.301035768166949
Desviación Estándar de <perimeter_mean>: 24.298981038754906
Desviación Estándar de <area_mean>: 351.914129181653
Desviación Estándar de <smoothness_mean>: 0.014064128137673618
Desviación Estándar de <compactness_mean>: 0.052812757932512194
Desviación Estándar de <concavity_mean>: 0.07971980870789348
Desviación Estándar de <concave points_mean>: 0.038802844859153605
Desviación Estándar de <symmetry_mean>: 0.027414281336035715
Desviación Estándar de <fractal_dimension_mean>: 0.007060362795084459
Desviación Estándar de <radius_se>: 0.2773127329861039
Desviación Estándar de <texture_se>: 0.5516483926172023
Desviación Estándar de <perimeter_se>: 2.0218545540421076
Desviación Estándar de <area_se>: 45.49100551613181
Desviación Estándar de <smoothness_se>: 0.0030025179438390656
Desviación Estándar de <compactness_se>: 0.017908179325677388
Desviación Estándar de <concavity_se>: 0.03018606032298841
Desviación Estándar de <concave points_se>: 0.006170285174046869
Desviación Estándar de <symmetry_se>: 0.008266371528798397
Desviación Estándar de <fractal_dimension_se>: 0.002646070967089195
Desviación Estándar de <radius_worst>: 4.833241580469323
Desviación Estándar de <texture_worst>: 6.146257623038319
Desviación Estándar de <perimeter_worst>: 33.602542269036356
Desviación Estándar de <area_worst>: 569.356992669949
Desviación Estándar de <smoothness_worst>: 0.022832429404835465
Desviación Estándar de <compactness_worst>: 0.15733648891374197
Desviación Estándar de <concavity_worst>: 0.2086242806081323
Desviación Estándar de <concave points_worst>: 0.06573234119594207
Desviación Estándar de <symmetry_worst>: 0.06186746753751869
Desviación Estándar de <fractal_dimension_worst>: 0.018061267348893986
Desviación Estándar de <Unnamed: 32>: nan

```

Es importante también saber los valores más altos y más bajos de cada una de las columnas, por lo que se utiliza el comando de Max y Min para localizarlos.

```
In [22]: columnas = data.columns.tolist()
for col in columnas:
    if col != "id":
        if col != "diagnosis":
            print("Valor Máximo y Mínimo de <{0}>: {1} -- {2}".format(col, data[col].max(), data[col].min()))
```

Valor MÁXIMO y MÍNIMO de `smoothness_mean`: 0.161 0.001

```

valor maximo y minimo de <radius_mean>: 28.11 -- 0.981
Valor Máximo y Mínimo de <texture_mean>: 39.28 -- 9.71
Valor Máximo y Mínimo de <perimeter_mean>: 188.5 -- 43.79
Valor Máximo y Mínimo de <area_mean>: 2501.0 -- 143.5
Valor Máximo y Mínimo de <smoothness_mean>: 0.1634 -- 0.05262999999999996
Valor Máximo y Mínimo de <compactness_mean>: 0.3454 -- 0.01938
Valor Máximo y Mínimo de <concavity_mean>: 0.4268 -- 0.0
Valor Máximo y Mínimo de <concave points_mean>: 0.2012 -- 0.0
Valor Máximo y Mínimo de <symmetry_mean>: 0.304 -- 0.106
Valor Máximo y Mínimo de <fractal_dimension_mean>: 0.09744 -- 0.04996000000000004
Valor Máximo y Mínimo de <radius_se>: 2.873 -- 0.1115
Valor Máximo y Mínimo de <texture_se>: 4.885 -- 0.3602
Valor Máximo y Mínimo de <perimeter_se>: 21.98 -- 0.757
Valor Máximo y Mínimo de <area_se>: 542.2 -- 6.802000000000005
Valor Máximo y Mínimo de <smoothness_se>: 0.03113 -- 0.001713
Valor Máximo y Mínimo de <compactness_se>: 0.1354 -- 0.002252
Valor Máximo y Mínimo de <concavity_se>: 0.396 -- 0.0
Valor Máximo y Mínimo de <concave points_se>: 0.05279 -- 0.0
Valor Máximo y Mínimo de <symmetry_se>: 0.07895 -- 0.007882
Valor Máximo y Mínimo de <fractal_dimension_se>: 0.02984 -- 0.0008948000000000001
Valor Máximo y Mínimo de <radius_worst>: 36.04 -- 7.93
Valor Máximo y Mínimo de <texture_worst>: 49.54 -- 12.02
Valor Máximo y Mínimo de <perimeter_worst>: 251.2 -- 50.41
Valor Máximo y Mínimo de <area_worst>: 4254.0 -- 185.2
Valor Máximo y Mínimo de <smoothness_worst>: 0.2226 -- 0.07117000000000001
Valor Máximo y Mínimo de <compactness_worst>: 1.058 -- 0.02729
Valor Máximo y Mínimo de <concavity_worst>: 1.252 -- 0.0
Valor Máximo y Mínimo de <concave points_worst>: 0.2910000000000004 -- 0.0
Valor Máximo y Mínimo de <symmetry_worst>: 0.6638 -- 0.1565
Valor Máximo y Mínimo de <fractal_dimension_worst>: 0.2075 -- 0.05504
Valor Máximo y Mínimo de <Unnamed: 32>: nan -- nan

```

Todos estos valores son en conjunto de los resultados tanto malignos como benignos. Ahora podemos hacer algunas correlaciones entre los diferentes diagnósticos y los resultados de sus columnas. Por ejemplo, se puede saber los mismos datos de Promedio, Varianza y Desviación pero por diagnóstico separado con el comando groupby.

In [23]: `data.groupby('diagnosis').mean()`

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean
diagnosis							
B	2.654382e+07	12.146524	17.914762	78.075406	462.790196	0.092478	0.080085
M	3.681805e+07	17.462830	21.604906	115.365377	978.376415	0.102898	0.145188

2 rows × 32 columns

In [24]: `data.groupby('diagnosis').var()`

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean
diagnosis							
B	13628148690161474	3.170222	15.961021	139.415582	18033.030100	0.000181	0.001139
M	19015458403133984	10.265431	14.284393	477.625870	135378.355365	0.000159	0.002915

2 rows × 32 columns

In [25]: `data.groupby('diagnosis').std()`

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean
diagnosis							
B	1.167397e+08	1.780512	3.995125	11.807438	134.287118	0.013446	0.033750
M	1.378965e+08	3.203971	3.779470	21.854653	367.937978	0.012608	0.053987

2 rows × 32 columns

In [26]: `data.groupby('diagnosis').max()`

Out[26]:

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave_points_mean	symmetry_mean	fractal_dimension_mean	radius_se	texture_se	perimeter_se	area_se	smoothness_se	compactness_se	concavity_se	concave_points_se	symmetry_se	fractal_dimension_se	radius_worst	texture_worst	perimeter_worst	area_worst	smoothness_worst	compactness_worst	concavity_worst	concave_points_worst	symmetry_worst	fractal_dimension_worst
diagnosis																															
B	911320502	17.85	33.81	114.6	992.1	0.1634	0.2239	0.05263	0.01938	0.00000	0.565369	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
M	911296202	28.11	39.28	188.5	2501.0	0.1447	0.3454	0.07371	0.04605	0.02399	0.565369	0.143048	0.679090	0.275869	0.691765	0.732562	0.301467	0.497473	0.143048	0.679090	0.275869	0.691765	0.732562	0.301467	0.497473	0.143048	0.679090	0.275869	0.691765	0.732562	0.301467

2 rows × 32 columns

In [27]: `data.groupby('diagnosis').min()`

Out[27]:

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave_points_mean	symmetry_mean	fractal_dimension_mean	radius_se	texture_se	perimeter_se	area_se	smoothness_se	compactness_se	concavity_se	concave_points_se	symmetry_se	fractal_dimension_se	radius_worst	texture_worst	perimeter_worst	area_worst	smoothness_worst	compactness_worst	concavity_worst	concave_points_worst	symmetry_worst	fractal_dimension_worst
diagnosis																															
B	8913	6.981	9.71	43.79	143.5	0.05263	0.01938	0.00000	0.00000	0.00000	0.565369	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000		
M	8670	10.950	10.38	71.90	361.6	0.07371	0.04605	0.02399	0.04605	0.02399	0.565369	0.143048	0.679090	0.275869	0.691765	0.732562	0.301467	0.497473	0.143048	0.679090	0.275869	0.691765	0.732562	0.301467	0.497473	0.143048	0.679090	0.275869	0.691765	0.732562	0.301467

2 rows × 32 columns

Con el comando de groupby, se pueden apreciar los datos de una manera más limpia y ordenada que con las líneas de código anteriores.

Ahora que ya se tiene una idea de como hacer los comandos estadísticos con comando distintos al de Describe, se puede hacer algo diferente, como relacionar algunos de los datos.

Una de las maneras en las que pandas tiene para mostrar la correlación entre los valores de una tabla es con el comando corr() en los cuales se puede cambiar el parámetro del Método de correlación que quiera utilizarse. Estos parámetros son:

1. Pearson: Este coeficiente cuantifica el grado en el que una relación entre dos variables puede ser descrito como una línea.
2. Spearman: A diferencia de la correlación de Pearson, Spearman no está restringido a seguir relaciones lineales. En lugar de comparar las medias y varianzas, el coeficiente de Spearman se fija en el orden relativo del orden de los valores por cada variable. Esto lo hace más apropiado para datos continuos y discretos.
3. Kendall's Tau: Esta correlación, a diferencia de las otras, no toma en cuenta la diferencia entre sus rangos. Este tipo es más apropiado para datos discretos.

El parámetro default de este comando es el de pearson.

In [41]: `data.corr(method = 'pearson')`

Out[41]:

	id	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave_points_mean	symmetry_mean	fractal_dimension_mean	radius_se	texture_se	perimeter_se	area_se	smoothness_se	compactness_se	concavity_se	concave_points_se	symmetry_se	fractal_dimension_se	radius_worst	texture_worst	perimeter_worst	area_worst	smoothness_worst	compactness_worst	concavity_worst	concave_points_worst	symmetry_worst	fractal_dimension_worst
id	1.000000	0.074626	0.099770	0.073159	0.096893	-0.012968	0.000006	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000		
radius_mean	0.074626	1.000000	0.323782	0.997855	0.987357	0.170581	0.506124	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000		
texture_mean	0.099770	0.323782	1.000000	0.329533	0.321086	-0.023389	0.236702	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000			
perimeter_mean	0.073159	0.997855	0.329533	1.000000	0.986507	0.207278	0.556936	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000		
area_mean	0.096893	0.987357	0.321086	0.986507	1.000000	0.177028	0.177028	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000		
smoothness_mean	-0.012968	0.170581	-0.023389	0.207278	0.177028	1.000000	0.659123	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000		
compactness_mean	0.000096	0.506124	0.236702	0.556936	0.498502	0.659123	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000		
concavity_mean	0.050080	0.676764	0.302418	0.716136	0.685983	0.521984	0.883121	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000		
concave_points_mean	0.044158	0.822529	0.293464	0.850977	0.823269	0.553695	0.831135	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000		
symmetry_mean	-0.022114	0.147741	0.071401	0.183027	0.151293	0.557775	0.602641	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000		
fractal_dimension_mean	-0.052511	-0.311631	-0.076437	-0.261477	-0.283110	0.584792	0.565369	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000		
radius_se	0.143048	0.679090	0.275869	0.691765	0.732562	0.301467	0.497473	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000		

texture_se	-0.007526	-0.097317	0.386358	-0.086761	-0.066280	0.068406	0.046205
perimeter_se	0.137331	0.674172	0.281673	0.693135	0.726628	0.296092	0.548905
area_se	0.177742	0.735864	0.259845	0.744983	0.800086	0.246552	0.455653
smoothness_se	0.096781	-0.222600	0.006614	-0.202694	-0.166777	0.332375	0.135295
compactness_se	0.033961	0.206000	0.191975	0.250744	0.212583	0.318943	0.738722
concavity_se	0.055239	0.194204	0.143293	0.228082	0.207660	0.248396	0.570517
concave points_se	0.078768	0.376169	0.163851	0.407217	0.372320	0.380676	0.642262
symmetry_se	-0.017306	-0.104321	0.009127	-0.081629	-0.072497	0.200774	0.229977
fractal_dimension_se	0.025725	-0.042641	0.054458	-0.005523	-0.019887	0.283607	0.507318
radius_worst	0.082405	0.969539	0.352573	0.969476	0.962746	0.213120	0.535315
texture_worst	0.064720	0.297008	0.912045	0.303038	0.287489	0.036072	0.248133
perimeter_worst	0.079986	0.965137	0.358040	0.970387	0.959120	0.238853	0.590210
area_worst	0.107187	0.941082	0.343546	0.941550	0.959213	0.206718	0.509604
smoothness_worst	0.010338	0.119616	0.077503	0.150549	0.123523	0.805324	0.565541
compactness_worst	-0.002968	0.413463	0.277830	0.455774	0.390410	0.472468	0.865809
concavity_worst	0.023203	0.526911	0.301025	0.563879	0.512606	0.434926	0.816275
concave points_worst	0.035174	0.744214	0.295316	0.771241	0.722017	0.503053	0.815573
symmetry_worst	-0.044224	0.163953	0.105008	0.189115	0.143570	0.394309	0.510223
fractal_dimension_worst	-0.029866	0.007066	0.119205	0.051019	0.003738	0.499316	0.687382
Unnamed: 32	NaN	NaN	NaN	NaN	NaN	NaN	NaN

32 rows × 32 columns

Una vez que se tienen los datos se puede apreciar que el valor máximo de correlación es uno, el cual existe en las columnas/filas con el mismo nombre. El valor disminuye a menor relación existente entre los datos.

Branch: master ▾ Ciencia_de_Datos / Práctica 4.ipynb

mayraberrones94 ei

1 contributor

269 lines (268 sloc) | 447 KB

Práctica 4: Visualización de información con plotly.

Para esta práctica, se utilizan herramientas de visualización de gráficas para poder darle forma a los datos que se tienen en el archivo csv, mostrando algunos de los datos que se formaron la práctica anterior de manera mas visual.

La herramienta que se utiliza en esta práctica es la de seaborn.

```
In [1]: import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from pandas.tools import plotting
from scipy import stats
plt.style.use("ggplot")
import warnings
warnings.filterwarnings("ignore")
from scipy import stats
```

Como se realiza en prácticas anteriores, se cargan los datos del archivo csv con la herramienta de pandas. En este caso se tiene una columna sin nombre que causaba ruido en la práctica anterior, por lo que se incluye la orden de *drop* para remover dicha columna.

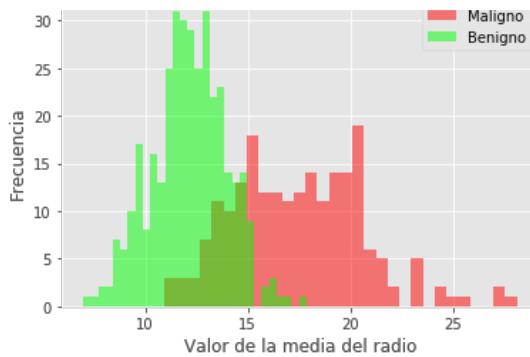
```
In [2]: data = pd.read_csv("/Users/mayraberrones/Documents/GitHub/Ciencia_de_Datos/w-data.csv")
```

```
In [3]: data = data.drop(['Unnamed: 32','id'],axis = 1)
```

Una vez que se tienen los datos cargados, lo primero que se quiere hacer es una gráfica tipo histograma para mostrar el contraste que existe entre los resultados benignos y malignos de alguna de las columnas. En este caso se toma la columna de *radius_mean* que representa la media del radio de la anomalía.

```
In [10]: m = plt.hist(data[data["diagnosis"] == "M"].radius_mean,bins=30,fc = (1,0,0,0.5),label = "Maligno")
)
b = plt.hist(data[data["diagnosis"] == "B"].radius_mean,bins=30,fc = (0,1,0,0.5),label = "Benigno")
)
plt.legend()
plt.xlabel("Valor de la media del radio")
plt.ylabel("Frecuencia")
plt.title("Histograma")
plt.show()
```

Histograma

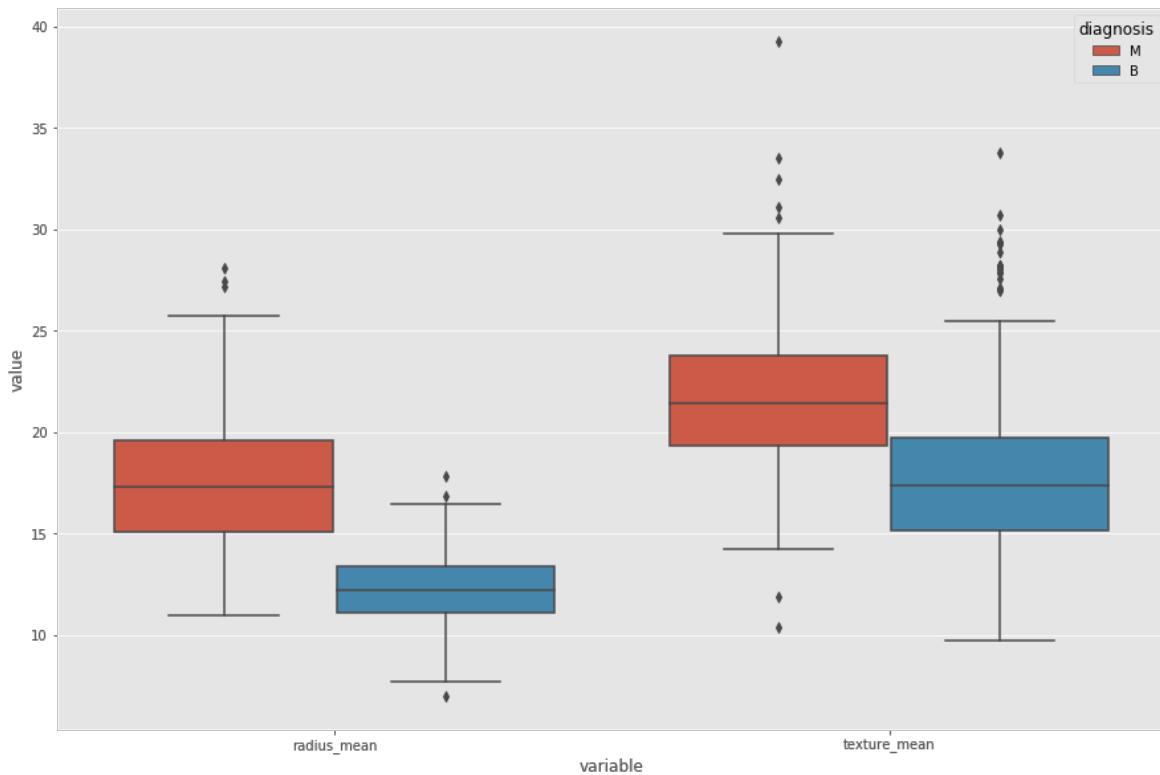


<Figure size 2160x1800 with 0 Axes>

Con esto se puede apreciar de manera más sencilla las similitudes y desigualdades que existen dentro de esta columna para las anomalías malignas y benignas, con lo que se puede sacar diferentes conclusiones para los rangos que se quieren formar de los valores de diferencia entre ambas anomalías.

Después se puede jugar con otro tipo de gráficas, como por ejemplo, las conocidas cajas bigote. Con esta herramienta se pueden comparar varias columnas para ver que tanto están separadas en cuanto a características que se observan en la práctica anterior (promedios, cuartiles, valor máximo y valor mínimo)

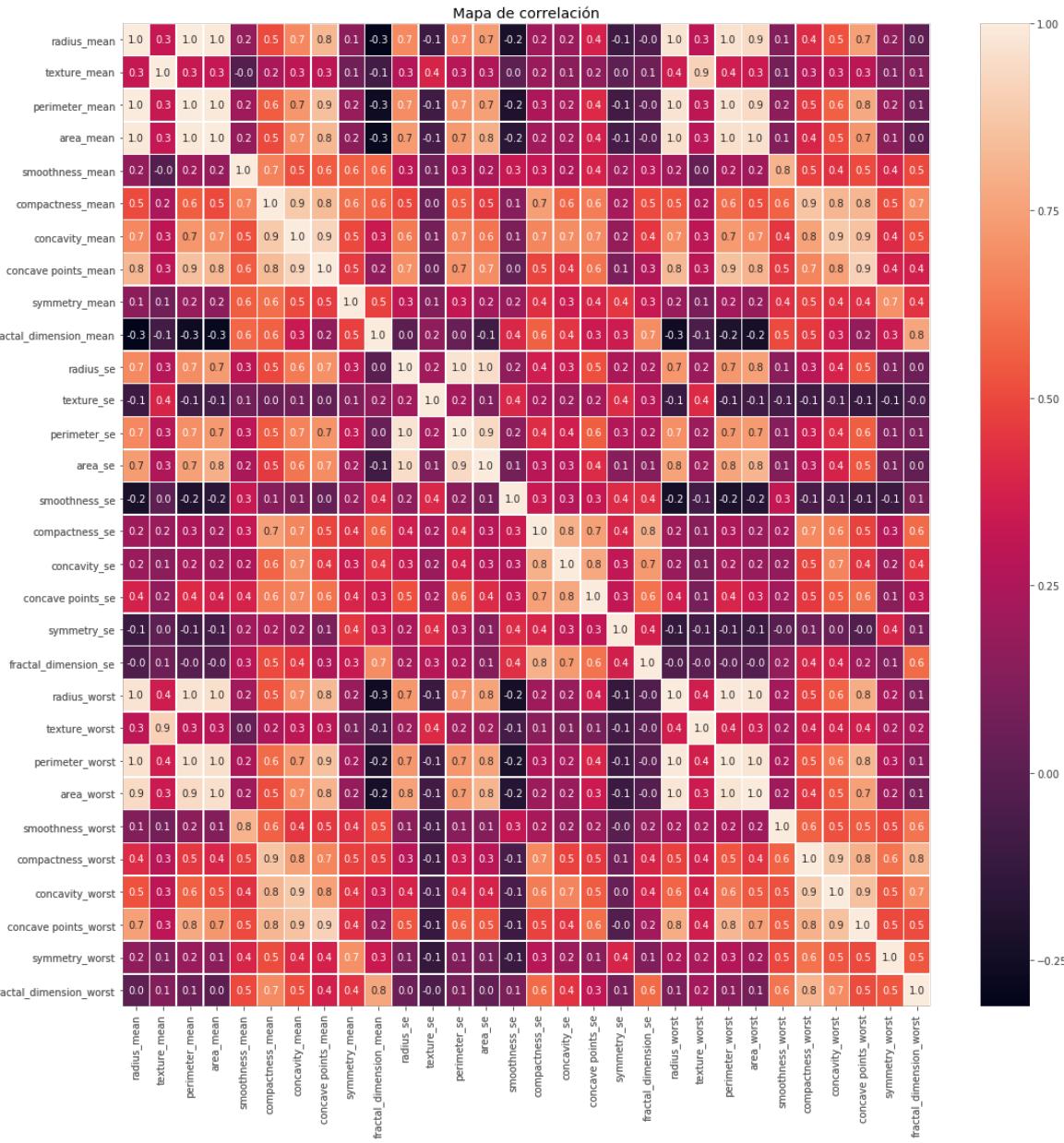
```
In [12]: mdata = pd.melt(data,id_vars = "diagnosis",value_vars = ['radius_mean', 'texture_mean'])
plt.figure(figsize = (15,10))
sns.boxplot(x = "variable", y = "value", hue="diagnosis",data= mdata)
plt.show()
```



Por último se tiene la tabla de correlación que se hizo en la práctica pasada. Con la herramienta de `heatmap` de seaborn, se puede hacer la tabla de manera más gráfica, teniendo un gradiente de color para diferenciar los valores más significativos de los menos significativos. En este caso, vemos que los colores más claros son los que tienen una mayor correlación entre sí.

```
In [14]: f,ax=plt.subplots(figsize = (18,18))
sns.heatmap(data.corr(),annot= True,linewidths=0.5,fmt = ".1f",ax=ax)
plt.xticks(rotation=90)
plt.yticks(rotation=0)
```

```
plt.title('Mapa de correlación')
plt.savefig('graph.png')
plt.show()
```



En cuanto a la herramienta propuesta por la profesora, se encontraron varias dificultades al momento de instalarlas en la terminal. Al parecer cuando se abre la terminal en python directamente, no tiene problemas para importar la librería, pero al importarla en el jupyter es cuando no la reconoce. La herramienta plotly puede utilizarse en su página web de manera gratuita, más al momento de importar las gráficas que se crearon existe el problema de importación que se menciona. (El plan es enmendar este error con la profesora en clases siguientes.)

In [4]: `from IPython.display import IFrame`
`IFrame(src='https://plot.ly/~MayraBerrones/6', width=700, height=600)`

Out[4]: La gráfica que se muestra es similar a la que se realizó con la herramienta de seaborn, más la diferencia que puede apreciarse es que en las

gráficas de plotly se puede interactuar mejor con la gráfica, mirando las características como el promedio, los cuartiles, el mayor y menor valor, al pasar el cursor por encima.

Boxplot1.png

(No funcionó)

Práctica 5

June 3, 2019

1 Práctica 5: Pruebas estadísticas.

Para esta práctica se realizan algunas pruebas estadísticas para los datos que se tienen en el csv con el que se ha estado trabajando, usando librerías como la de plotly, matplotlib, y pruebas estadísticas de python.

Primero, y lo que hizo falta en la práctica anterior era importar plotly al notebook, esto se logró instalando directamente el plotly en el notebook usando conda para importar los paquetes. Esto se hizo en la práctica anterior como una actualización. Para ver como se logró esto, se puede verificar en este [link](#)

Ya que se tiene instalado el plotly y todas las demás librerías necesarias, se puede continuar con la práctica.

In [4]: `import plotly`

`plotly.tools.set_credentials_file(username='MayraBerrones', api_key='BubmIKE5Gw0uVv3WK`

In []:

Una vez que se conecta con la cuenta de plotly, se puede empezar a crear algunas gráficas. En este caso, se hizo un histograma overlay como el que se creó en la práctica anterior, pero en lugar de ser una imagen fija en png tenemos la herramienta interactiva que nos ofrece plotly.

In [7]: `import plotly.plotly as py`
`import plotly.graph_objs as go`
`import pandas as pd`
`import numpy as np`

```
data = pd.read_csv("https://raw.githubusercontent.com/mayraberrones94/Ciencia_de_Datos/master/wdbc.csv")

x0 = data.loc[data.diagnosis == 'M'].radius_mean
x1 = data.loc[data.diagnosis == 'B'].radius_mean
Maligno = go.Histogram(
    x=x0,
    opacity=0.75,
    name='Maligno'
)
Benigno = go.Histogram(
    x=x1,
    opacity=0.75,
```

```

        name='Benigno'
    )
data = [Maligno, Benigno]
layout = go.Layout(
    title='Plot Title',
    xaxis=dict(
        title='x Axis',
        titlefont=dict(
            family='Courier New, monospace',
            size=18,
            color='#7f7f7f'
        )
    ),
    yaxis=dict(
        title='y Axis',
        titlefont=dict(
            family='Courier New, monospace',
            size=18,
            color='#7f7f7f'
        )
    )
)
layout = go.Layout(barmode='overlay')
fig = go.Figure(data=data, layout=layout)

py.iplot(fig, filename='overlaid histogram')

```

/Users/mayraberrones/miniconda3/lib/python3.6/site-packages/IPython/core/display.py:689: UserWarning:

Consider using IPython.display.IFrame instead

Out[7]: <plotly.tools.PlotlyDisplay object>

In []:

Ya que se tiene esta gráfica, se puede empezar a hacer experimentos con los análisis estadísticos. En este caso, se utilizaron dos de los datos que presentan comportamientos similares en el csv, que son el de 'radius_mean' y 'texture_mean' para ver si son datos normales.

```

In [5]: import plotly.plotly as py
import plotly.graph_objs as go
import pandas as pd
from statsmodels.graphics.gofplots import qqplot
import matplotlib as plt
from numpy.random import randn
from numpy import concatenate

```

```

d = pd.read_csv("https://raw.githubusercontent.com/mayraberrones94/Ciencia_de_Datos/master/datos/breast_cancer.csv")
cfep = d.loc[d.diagnosis == 'M'].radius_mean # elisa primera
cfes = d.loc[d.diagnosis == 'M'].texture_mean # elisa segunda
cfmp = d.loc[d.diagnosis == 'B'].radius_mean # moi primera
cfms = d.loc[d.diagnosis == 'B'].texture_mean # moi segunda

layout = {'xaxis': {'range': [0, 100]}, 'yaxis': {'range': [0, 1]}, \
'shapes': [{ 'type': 'line', 'x0': 70, 'y0': 0, 'x1': 70, 'y1': 1, \
'line': { 'color': 'rgb(255, 0, 0)', 'width': 2}}]}

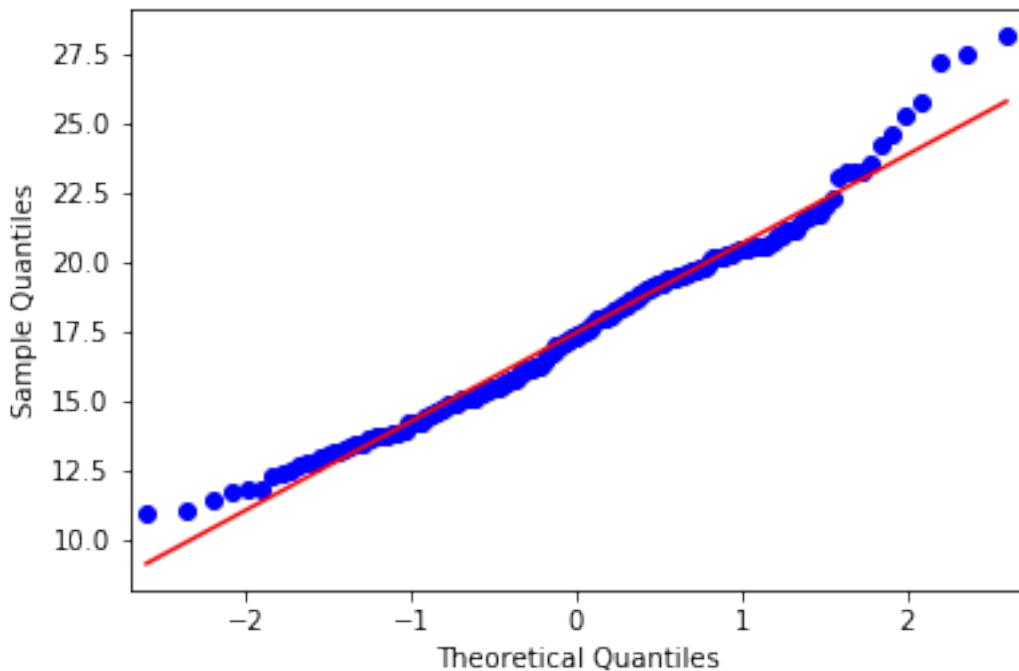
py.iplot({'data': [go.Histogram(x = cfms, histnorm='probability')], 'layout': layout}, \
unimodal = 15 * randn(1000) + 75 # varianza 15, media 75
bimodal = concatenate((unimodal, 20 * randn(1000) + 45)) # lo mismo pero ahora con un bimodal

f = qqplot(cfep, line='s')
# plt.savefig(f, "qq_cfep.png")

```

/Users/mayraberrones/miniconda3/lib/python3.6/site-packages/IPython/core/display.py:689: UserWarning:

Consider using IPython.display.IFrame instead



Cuando se observa que se tiene esa línea roja atravesando los datos, significa que son más o menos normales.

```

In [9]: import pandas as pd
        from numpy.random import randn
        from numpy.random import seed
        from numpy import concatenate, isnan
        import ssl

        if getattr(ssl, '_create_unverified_context', None):
            ssl._create_default_https_context = ssl._create_unverified_context
        d = pd.read_csv("https://raw.githubusercontent.com/mayraberrones94/Ciencia_de_Datos/master/iris.csv")

        seed(7)
        nd = 15 * randn(1000) + 75
        datos = {'Maligno - Radio': d.loc[d.diagnosis == 'M'].radius_mean , \
                  'Maligno - Textura': d.loc[d.diagnosis == 'M'].texture_mean, \
                  'Benigno - Radio': d.loc[d.diagnosis == 'B'].radius_mean, \
                  'Benigno - Textura': d.loc[d.diagnosis == 'B'].texture_mean, \
                  'unim': nd, \
                  'bim': concatenate((nd, 20 * randn(1000) + 45))}

        from scipy.stats import shapiro
        for alpha in [0.05, 0.01]:
            for data in datos:
                crudos = datos[data]
                s, p = shapiro(crudos[~isnan(crudos)]) # es MUY importante quitar los NaN
                print('{:s} {:.2f} {:.3f}'.format(data, s, p))
                if p > alpha:
                    print('aceptablemente normal con nivel de significancia', alpha)
                else:
                    print('no parece ser normal con nivel de significancia', alpha)

Maligno - Radio 0.98 0.002
no parece ser normal con nivel de significancia 0.05
Maligno - Textura 0.97 0.000
no parece ser normal con nivel de significancia 0.05
Benigno - Radio 1.00 0.668
aceptablemente normal con nivel de significancia 0.05
Benigno - Textura 0.94 0.000
no parece ser normal con nivel de significancia 0.05
unim 1.00 0.834
aceptablemente normal con nivel de significancia 0.05
bim 0.99 0.000
no parece ser normal con nivel de significancia 0.05
Maligno - Radio 0.98 0.002
no parece ser normal con nivel de significancia 0.01
Maligno - Textura 0.97 0.000
no parece ser normal con nivel de significancia 0.01
Benigno - Radio 1.00 0.668
aceptablemente normal con nivel de significancia 0.01

```

```

Benigno - Textura 0.94 0.000
no parece ser normal con nivel de significancia 0.01
unim 1.00 0.834
aceptablemente normal con nivel de significancia 0.01
bim 0.99 0.000
no parece ser normal con nivel de significancia 0.01

```

Después tenemos las pruebas con la librería ssl y numpy, que nos dice que la mayoría de los datos no son normales.

```

In [14]: import pandas as pd
        from numpy.random import randn
        from numpy.random import seed
        from numpy import concatenate, isnan
        from scipy.stats import mannwhitneyu
        import ssl

        if getattr(ssl, '_create_unverified_context', None):
            ssl._create_default_https_context = ssl._create_unverified_context
d = pd.read_csv("https://raw.githubusercontent.com/mayraberrones94/Ciencia_de_Datos/master/iris.csv")

seed(7)
nd = 15 * randn(1000) + 75
datos = {'Maligno-Radio': d.loc[d.diagnosis == 'M'].radius_mean , \
          'Maligno-Textura': d.loc[d.diagnosis == 'M'].texture_mean, \
          'Benigno-Radio': d.loc[d.diagnosis == 'B'].radius_mean, \
          'Benigno-Textura': d.loc[d.diagnosis == 'B'].texture_mean, \
          'unim': nd, \
          'bim': concatenate((nd, 20 * randn(1000) + 45))}

def prueba(c1, c2):
    d1 = c1[~isnan(c1)]
    d2 = c2[~isnan(c2)]
    n1 = len(d1)
    n2 = len(d2)
    if min(n1, n2) < 20:
        print('hay muy pocos datos como para obtener un resultado confiable')
        return
    for alpha in [0.05, 0.01]:
        s, p = mannwhitneyu(d1, d2)
        print('{:d} {:d} {:.2f} {:.3f}'.format(n1, n2, s, p))
        if p > alpha:
            print('son igualmente distribuidos con nivel de significancia', alpha)
        else:
            print('se ven differentemente distribuidos con nivel de significancia', alpha)

print("Maligno")

```

```

prueba(datos['Maligno-Radio'], datos['Maligno-Textura'])
print("Benigno")
prueba(datos['Benigno-Radio'], datos['Benigno-Textura'])

Maligno
212 212 8487.50 0.000
se ven differentemente distribuidos con nivel de significancia 0.05
212 212 8487.50 0.000
se ven differentemente distribuidos con nivel de significancia 0.01
Benigno
357 357 8299.50 0.000
se ven differentemente distribuidos con nivel de significancia 0.05
357 357 8299.50 0.000
se ven differentemente distribuidos con nivel de significancia 0.01

```

Cuando se realiza una prueba más, se puede observar que los datos introducidos tienen una distribución distinta, dandole dos niveles de significancia, en ambos casos de diagnóstico Benigno y Maligno.

```

In [13]: import pandas as pd
         from numpy import isnan, nan
         import ssl

         if getattr(ssl, '_create_unverified_context', None):
             ssl._create_default_https_context = ssl._create_unverified_context

         d = pd.read_csv("https://raw.githubusercontent.com/mayraberrones94/Ciencia_de_Datos/main/datos.csv")
         from scipy.stats import wilcoxon

         pares = dict()
         pedazo = d.loc[~isnan(d.diagnosis == 'M') & ~isnan(d.diagnosis == 'B')]
         pares['diagnosis'] = (pedazo.radius_mean, pedazo.texture_mean)

         for datos in pares:
             (d1, d2) = pares[datos]
             n1 = len(d1)
             n2 = len(d2)
             if min(n1, n2) < 20:
                 print('hay muy pocos datos de {:s} como para obtener un resultado confiable'.format(datos))
             else:
                 for alpha in [0.05, 0.01]:
                     s, p = wilcoxon(d1, d2)
                     print('{:s} {:d} {:d} {:.2f} {:.3f}'.format(datos, n1, n2, s, p))
                     if p > alpha:
                         print('son igualmente distribuidos con nivel de significancia', alpha)
                     else:
                         print('se ven differentemente distribuidos con nivel de significancia')

```

diagnosis 569 569 7362.50 0.000
se ven differentemente distribuidos con nivel de significancia 0.05
diagnosis 569 569 7362.50 0.000
se ven differentemente distribuidos con nivel de significancia 0.01

En el último dato no se separan los diagnosticos en maligno y benigno, y aun asi, con ambas significancias da como resultado que los datos son differentemente distribuidos.

Join GitHub today
GitHub is home to over 31 million developers working together to host and review code, manage projects, and build software together.

Sign up

Branch: master ▾ Ciencia_de_Datos / Práctica 6.ipynb

mayraberrones94 fin

1 contributor

6.01 MB

7309f7f 18 hours ago

Find file Copy path

Dismiss

Práctica 6: Modelos lineales con scipy.stats

El objetivo de esta práctica es modelar por los menos dos aspectos que tengan los datos con modelos lineales simples. Se utiliza la nueva he

En la práctica anterior se realizó un mapa de calor para revisar las correlaciones que tienen entre ellos los datos dentro del csv.

```
In [38]: import plotly
plotly.tools.set_credentials_file(username='MayraBerrones', api_key='BubmIKE5Gw0uVv3WKe1l')

In [41]: import plotly.offline as py
import numpy as np
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
import plotly.tools as tls
import plotly.figure_factory as ff

py.init_notebook_mode()

e = pd.read_csv("https://raw.githubusercontent.com/mayraberrones94/Ciencia_de_Datos/master/w-data.csv")
correlation = e.corr()
#tick labels
matrix_cols = correlation.columns.tolist()
#convert to array
corr_array = np.array(correlation)
#Plotting
trace = go.Heatmap(z = corr_array,
                     x = matrix_cols,
                     y = matrix_cols,
                     xgap = 2,
                     ygap = 2,
                     colorscale='Rainbow',
                     colorbar = dict(),
                     )
layout = go.Layout(dict(title = 'Correlation Matrix for variables',
                        autosize = False,
                        height = 720,
                        width = 800,
                        margin = dict(r = 0 ,l = 210,
                                     t = 25,b = 210,
                                     ),
                        yaxis = dict(tickfont = dict(size = 9)),
                        xaxis = dict(tickfont = dict(size = 9))
                       )
```

```
        )
fig = go.Figure(data = [trace], layout = layout)
py.iplot(fig)
```

NOTA: Por alguna razón el plotly offline no quiere mostrar mi gráfica. Si quiere verla, puede seguir el siguiente [link](#)

La diagonal principal es la que se encuentra una columna con si misma, pero gracias a la coloración de este mapa, se puede apreciar rápidamente

Una vez que tenemos identificados dichos elementos, podemos empezar a hacer comparaciones entre los que están muy correlacionados y los que no.

Se ven a continuación 4 gráficas de elementos relacionados, y otras cuatro después de elementos no relacionados.

In [6]:

```
import pandas as pd
from numpy import isnan
from statsmodels.graphics.gofplots import qqplot
import matplotlib.pyplot as plt
from scipy.stats import shapiro
import ssl

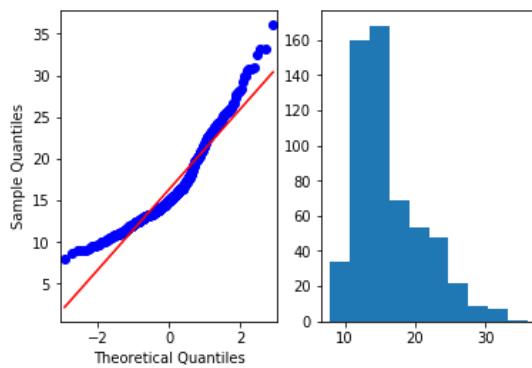
if getattr(ssl, '_create_unverified_context', None):
    ssl._create_default_https_context = ssl._create_unverified_context
e = pd.read_csv("https://raw.githubusercontent.com/mayraberrones94/Ciencia_de_Datos/master/w-data.csv")
d1 = e.loc[e.radius_worst]
d2 = e.loc[~isnan(e.radius_mean)]
m = pd.concat([d1, d2])
d = m.radius_worst.dropna()
f, ax = plt.subplots(1, 2)
qqplot(d, line='s', ax = ax[0])
ax[1] = plt.hist(d)
for a in [0.05, 0.01]:
    s, p = shapiro(d)
    print(s, p, a, "normal" if p > a else "no normal")
plt.show()
```

/Users/mayraberrones/miniconda3/lib/python3.6/site-packages/ipykernel_launcher.py:11: FutureWarning:
Passing list-likes to .loc or [] with any missing label will raise
KeyError in the future, you can use .reindex() as an alternative.

See the documentation here:

<https://pandas.pydata.org/pandas-docs/stable/indexing.html#deprecate-loc-reindex-listlike>
This is added back by InteractiveShellApp.init_path()

0.9138855338096619 1.7342190612392723e-17 0.05 no normal
0.9138855338096619 1.7342190612392723e-17 0.01 no normal



In [7]:

```
d1 = e.loc[e.radius_worst]
d2 = e.loc[~isnan(e.perimeter_mean)]
m = pd.concat([d1, d2])
d = m.radius_worst.dropna()
f, ax = plt.subplots(1, 2)
qqplot(d, line='s', ax = ax[0])
ax[1] = plt.hist(d)
for a in [0.05, 0.01]:
    s, p = shapiro(d)
    print(s, p, a, "normal" if p > a else "no normal")
```

```

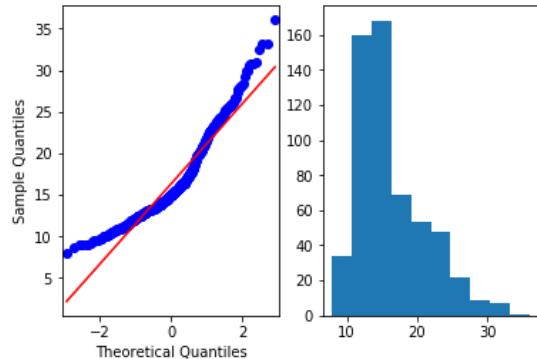
    print(s, p, a, "normal" if p > a else "no normal")
plt.show()

/Users/mayraberrones/miniconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: FutureWarning:
Passing list-like to .loc or [] with any missing label will raise
KeyError in the future, you can use .reindex() as an alternative.

See the documentation here:
https://pandas.pydata.org/pandas-docs/stable/indexing.html#deprecate-loc-reindex-listlike
    """Entry point for launching an IPython kernel.

0.9138855338096619 1.7342190612392723e-17 0.05 no normal
0.9138855338096619 1.7342190612392723e-17 0.01 no normal

```



```

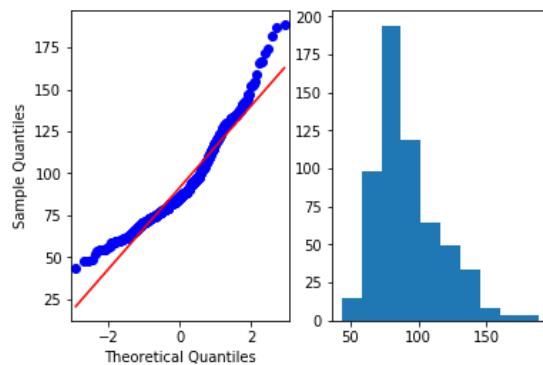
In [9]: d1 = e.loc[e.perimeter_mean]
d2 = e.loc[~isnan(e.radius_worst)]
m = pd.concat([d1, d2])
d = m.perimeter_mean.dropna()
f, ax = plt.subplots(1, 2)
qqplot(d, line='s', ax = ax[0])
ax[1] = plt.hist(d)
for a in [0.05, 0.01]:
    s, p = shapiro(d)
    print(s, p, a, "normal" if p > a else "no normal")
plt.show()

/Users/mayraberrones/miniconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: FutureWarning:
Passing list-like to .loc or [] with any missing label will raise
KeyError in the future, you can use .reindex() as an alternative.

See the documentation here:
https://pandas.pydata.org/pandas-docs/stable/indexing.html#deprecate-loc-reindex-listlike
    """Entry point for launching an IPython kernel.

0.9331727623939514 1.4505056998112445e-15 0.05 no normal
0.9331727623939514 1.4505056998112445e-15 0.01 no normal

```



```

In [10]: d1 = e.loc[e.area_mean]
d2 = e.loc[~isnan(e.radius_worst)]
m = pd.concat([d1, d2])
d = m.area_mean.dropna()
f, ax = plt.subplots(1, 2)
qqplot(d, line='s', ax = ax[0])

```

```

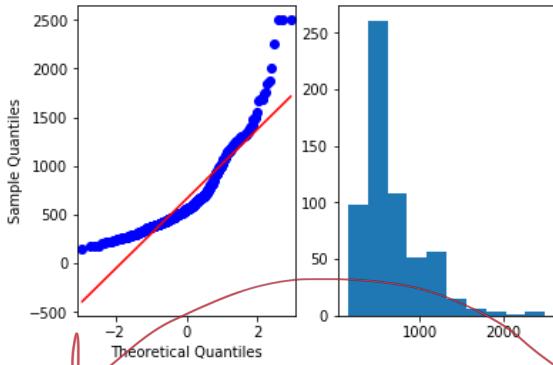
ax[1] = plt.hist(d)
for a in [0.05, 0.01]:
    s, p = shapiro(d)
    print(s, p, a, "normal" if p > a else "no normal")
plt.show()

/Users/mayraberrones/miniconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: FutureWarning:
Passing list-like to .loc or [] with any missing label will raise
KeyError in the future, you can use .reindex() as an alternative.

See the documentation here:
https://pandas.pydata.org/pandas-docs/stable/indexing.html#deprecate-loc-reindex-listlike
    """Entry point for launching an IPython kernel.

0.854504406452179 3.9272559797657846e-23 0.05 no normal
0.854504406452179 3.9272559797657846e-23 0.01 no normal

```



```

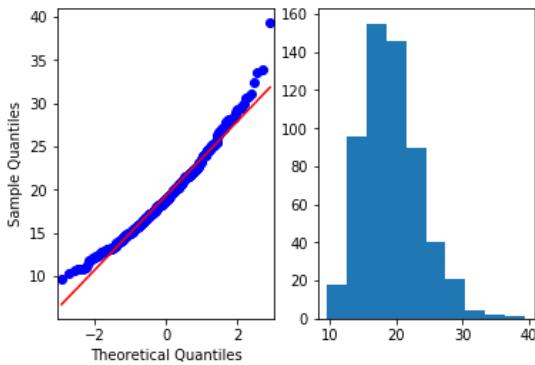
In [11]: d1 = e.loc[e.texture_mean]
d2 = e.loc[~isnan(e.texture_worst)]
m = pd.concat([d1, d2])
d = m.texture_mean.dropna()
f, ax = plt.subplots(1, 2)
qqplot(d, line='s', ax = ax[0])
ax[1] = plt.hist(d)
for a in [0.05, 0.01]:
    s, p = shapiro(d)
    print(s, p, a, "normal" if p > a else "no normal")
plt.show()

/Users/mayraberrones/miniconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: FutureWarning:
Passing list-like to .loc or [] with any missing label will raise
KeyError in the future, you can use .reindex() as an alternative.

See the documentation here:
https://pandas.pydata.org/pandas-docs/stable/indexing.html#deprecate-loc-reindex-listlike
    """Entry point for launching an IPython kernel.

0.9769935607910156 7.756307951467534e-08 0.05 no normal
0.9769935607910156 7.756307951467534e-08 0.01 no normal

```



```

In [12]: d1 = e.loc[e.area_worst]
d2 = e.loc[~isnan(e.radius_worst)]

```

```

m = pd.concat([d1, d2])
d = m.area_worst.dropna()
f, ax = plt.subplots(1, 2)
qqplot(d, line='s', ax = ax[0])
ax[1] = plt.hist(d)
for a in [0.05, 0.01]:
    s, p = shapiro(d)
    print(s, p, a, "normal" if p > a else "no normal")
plt.show()

```

/Users/mayraberrones/miniconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: FutureWarning:
Passing list-like to .loc or [] with any missing label will raise
KeyError in the future, you can use .reindex() as an alternative.

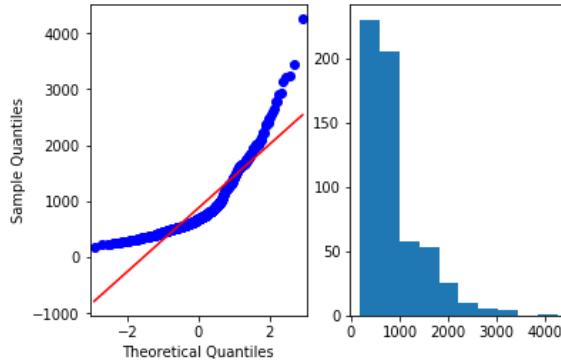
See the documentation here:

<https://pandas.pydata.org/pandas-docs/stable/indexing.html#deprecate-loc-reindex-listlike>
""Entry point for launching an IPython kernel.

```

0.8131364583969116 1.3208961865742285e-25 0.05 no normal
0.8131364583969116 1.3208961865742285e-25 0.01 no normal

```



```

In [14]: d1 = e.loc[e.area_mean]
d2 = e.loc[~isnan(e.fractal_dimension_mean)]
m = pd.concat([d1, d2])
d = m.area_mean.dropna()
f, ax = plt.subplots(1, 2)
qqplot(d, line='s', ax = ax[0])
ax[1] = plt.hist(d)
for a in [0.05, 0.01]:
    s, p = shapiro(d)
    print(s, p, a, "normal" if p > a else "no normal")
plt.show()

```

/Users/mayraberrones/miniconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: FutureWarning:
Passing list-like to .loc or [] with any missing label will raise
KeyError in the future, you can use .reindex() as an alternative.

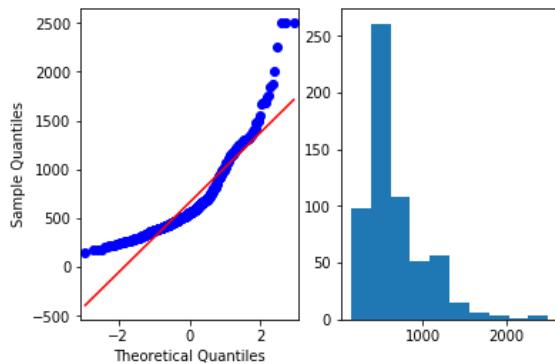
See the documentation here:

<https://pandas.pydata.org/pandas-docs/stable/indexing.html#deprecate-loc-reindex-listlike>
""Entry point for launching an IPython kernel.

```

0.854504406452179 3.9272559797657846e-23 0.05 no normal
0.854504406452179 3.9272559797657846e-23 0.01 no normal

```



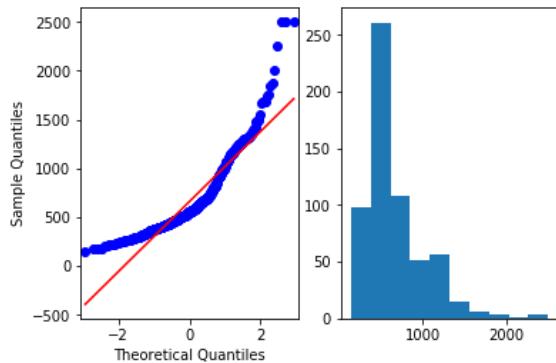
```
In [16]: d1 = e.loc[e.area_mean]
d2 = e.loc[~isnan(e.smoothness_se)]
m = pd.concat([d1, d2])
d = m.area_mean.dropna()
f, ax = plt.subplots(1, 2)
qqplot(d, line='s', ax = ax[0])
ax[1] = plt.hist(d)
for a in [0.05, 0.01]:
    s, p = shapiro(d)
    print(s, p, a, "normal" if p > a else "no normal")
plt.show()

/Users/mayraberrones/miniconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: FutureWarning:
Passing list-likes to .loc or [] with any missing label will raise
KeyError in the future, you can use .reindex() as an alternative.
```

See the documentation here:

<https://pandas.pydata.org/pandas-docs/stable/indexing.html#deprecate-loc-reindex-listlike>
 """Entry point for launching an IPython kernel.

```
0.854504406452179 3.9272559797657846e-23 0.05 no normal
0.854504406452179 3.9272559797657846e-23 0.01 no normal
```



Se realizan entonces más pruebas para tener datos más contundentes.

```
In [30]: import pandas as pd
from scipy.stats import shapiro, mannwhitneyu, wilcoxon, kruskal
import ssl

def prueba(d1, d2, a = 0.05, k = 20):
    if min(len(d1), len(d2)) < k:
        return None # no se puede concluir nada con tan pocos datos
    else:
        return mannwhitneyu(d1, d2)[1] > a # True si es la misma distribución

if getattr(ssl, '_create_unverified_context', None):
    ssl._create_default_https_context = ssl._create_unverified_context
m = pd.read_csv("https://raw.githubusercontent.com/mayraberrones94/Ciencia_de_Datos/master/w-data.csv")

pares = dict()
x = m.loc[~isnan(m.perimeter_mean) & ~isnan(m.radius_worst)]
pares['perimeter mean vs. radius worst'] = (x.perimeter_mean, x.radius_worst)
x = m.loc[~isnan(m.area_mean) & ~isnan(m.radius_worst)]
pares['area mean vs. radius worst'] = (x.area_mean, x.radius_worst)
x = m.loc[~isnan(m.texture_mean) & ~isnan(m.texture_worst)]
pares['texture mean vs. texture worst'] = (x.texture_mean, x.texture_worst)
x = m.loc[~isnan(m.area_mean) & ~isnan(m.radius_worst)]
pares['area mean vs. radius worst'] = (x.area_mean, x.radius_worst)
a = 0.05
for d in pares:
    (d1, d2) = pares[d]
    print('datos insuficientes' if min(len(d1), len(d2)) < 20 else \
          "iguales" if wilcoxon(d1, d2)[1] > a else "diferentes", "para", d)
```

```

niv = {'radius mean': [g[1].diagnosis.dropna() for g in m.groupby(['radius_mean'])], \
        'area mean': [g[1].diagnosis.dropna() for g in m.groupby(['area_mean'])], \
        'perimeter mean': [g[1].diagnosis.dropna() for g in m.groupby(['perimeter_mean'])], \
        'texture mean': [g[1].diagnosis.dropna() for g in m.groupby(['texture_mean'])], \
        'radius worst': [g[1].diagnosis.dropna() for g in m.groupby(['radius_worst'])], \
        'texture worst': [g[1].diagnosis.dropna() for g in m.groupby(['texture_worst'])], \
        }

print('\n'.join([("insuficientes datos" if min([len(x) for x in niv[d]]) < 1 else \
                  "sin diferencia" if kruskal(*niv[d])[1] > a else "hay diferencia") + \
                  " para " + d for d in niv])))

diferentes para perimeter mean vs. radius worst
diferentes para area mean vs. radius worst
diferentes para texture mean vs. texture worst
hay diferencia para radius mean
sin diferencia para area mean
sin diferencia para perimeter mean
sin diferencia para texture mean
hay diferencia para radius worst
sin diferencia para texture worst

```

Como se puede apreciar después de realizar las pruebas, se tiene que el radius mean y el radius worst son los únicos que presentan diferencia

```

In [34]: import pandas as pd
from numpy import isnan
from statsmodels.graphics.gofplots import qqplot
import matplotlib.pyplot as plt
from scipy.stats import shapiro
import ssl

if getattr(ssl, '_create_unverified_context', None):
    ssl._create_default_https_context = ssl._create_unverified_context
d = pd.read_csv("https://raw.githubusercontent.com/mayraberrones94/Ciencia_de_Datos/master/datosnw.csv")

moi = d.loc[d.diagnosis == 1] # filtramos los de moi ahora
print(len(moi))
d1 = moi.loc[moi.radius_mean]
print(len(d1))
d2 = moi.loc[~isnan(moi.radius_worst)]
print(len(d2))
m = pd.concat([d1, d2])
print(len(m))
d = m.radius_mean.dropna()
plt.rcParams["figure.figsize"] = [6, 9]
f = plt.figure()
qqplot(d, line='s', ax = f.add_subplot(311))
sf = f.add_subplot(312)
sf.hist(d)
sf.set_xlabel("Radio")
sf.set_ylabel("Frecuencia absoluta")
a = 0.05
print("normal" if shapiro(d)[1] > a else "no normal")
print("estimado inicial: antes", o.radius_mean.corr(o.radius_worst), "después", m.radius_mean.corr(m.radius_worst))

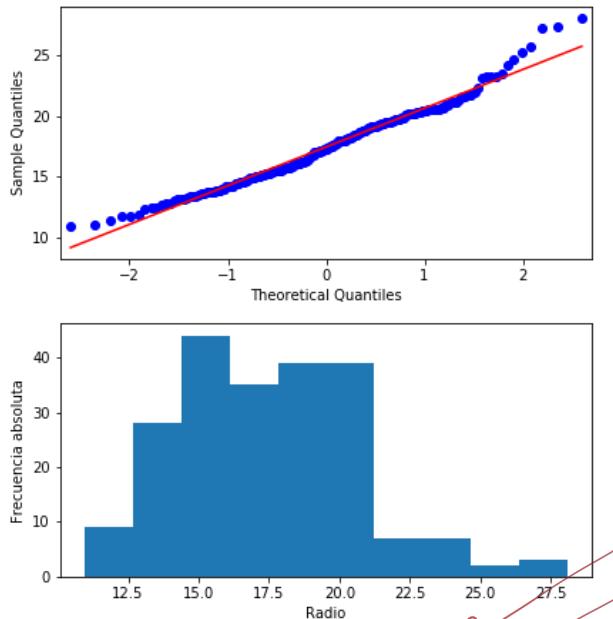
plt.tight_layout()
plt.show()

212
212
212
424
no normal
estimado inicial: antes 0.9695389726112065 después 0.9213005308411946
/Users/mayraberrones/miniconda3/lib/python3.6/site-packages/ipykernel_launcher.py:15: FutureWarning:
Passing list-like to .loc or [] with any missing label will raise
KeyError in the future, you can use .reindex() as an alternative.

See the documentation here:
https://pandas.pydata.org/pandas-docs/stable/indexing.html#deprecate-loc-reindex-listlike

```

```
from ipykernel import kernelapp as app
```



Se puede apreciar que las colitas de la gráfica no están tan dispersas, y gracias al histograma podemos ver cuales son los elementos más concurridos.

Por último conviene hacer una exploración de como están de diferentes las anomalías malignas y benignas en estas dos columnas mediante función.

```
In [15]: import matplotlib.pyplot as plt
from scipy import stats
from numpy import isnan
import pandas as pd
import ssl

if getattr(ssl, '_create_unverified_context', None):
    ssl._create_default_https_context = ssl._create_unverified_context
o = pd.read_csv("https://raw.githubusercontent.com/mayraberrones94/Ciencia_de_Datos/master/datosnw.csv")
m = pd.concat([o.loc[o.radius_mean], o.loc[~isnan(o.radius_worst)]])

y = m.radius_mean # lo que se modela
x = m.diagnosis # en función de qué se modela, es decir, queremos un  $y = f(x) = a * x + b$ 
mascara = ~isnan(x) & ~isnan(y)
x = x[mascara]
y = y[mascara]

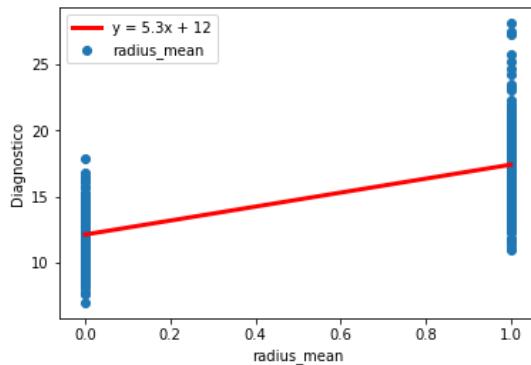
a, b, r, p, e = stats.linregress(x, y)
print("y = f(x) = {:.4f} x + {:.4f}".format(a, b))
print("error", e)
print("valor p", p)
print("pendiente {:s}significativo".format("no " if p >= 0.05 else ""))
print("R^2", r**2)

plt.plot(x, (a * x + b), label = 'y = {:.1f}x + {:.0f}'.format(a, b), color = 'red', linewidth = 3)
plt.scatter(x, y)
plt.legend(loc='upper left')
plt.xlabel("radius_mean")
plt.ylabel("Diagnóstico")
plt.show()

y = f(x) = 5.2697 x + 12.1504
error 0.20686216282911718
valor p 2.143852600098201e-96
pendiente significativo
R^2 0.5302106288502086

/Users/mayraberrones/miniconda3/lib/python3.6/site-packages/ipykernel_launcher.py:10: FutureWarning:
Passing list-like to .loc or [] with any missing label will raise
KeyError in the future, you can use .reindex() as an alternative.
```

See the documentation here:
<https://pandas.pydata.org/pandas-docs/stable/indexing.html#deprecate-loc-reindex-listlike>
 # Remove the CWD from sys.path while we load stuff.



```
In [35]: import matplotlib.pyplot as plt
from scipy import stats
from numpy import isnan
import pandas as pd
import ssl

if getattr(ssl, '_create_unverified_context', None):
    ssl._create_default_https_context = ssl._create_unverified_context
o = pd.read_csv("https://raw.githubusercontent.com/mayraberrones94/Ciencia_de_Datos/master/datosnw.csv")
m = pd.concat([o.loc[o.radius_mean], o.loc[~isnan(o.radius_worst)]])

y = m.radius_worst # lo que se modela
x = m.diagnosis # en función de qué se modela, es decir, queremos un  $y = f(x) = a * x + b$ 
mascara = ~isnan(x) & ~isnan(y)
x = x[mascara]
y = y[mascara]

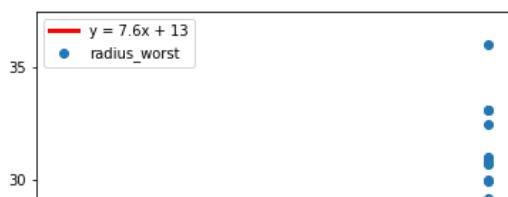
a, b, r, p, e = stats.linregress(x, y)
print("y = f(x) = {:.4f} x + {:.4f}".format(a, b))
print("error", e)
print("valor p", p)
print("pendiente {:s}significativo".format("no " if p >= 0.05 else ""))
print("R^2", r**2)

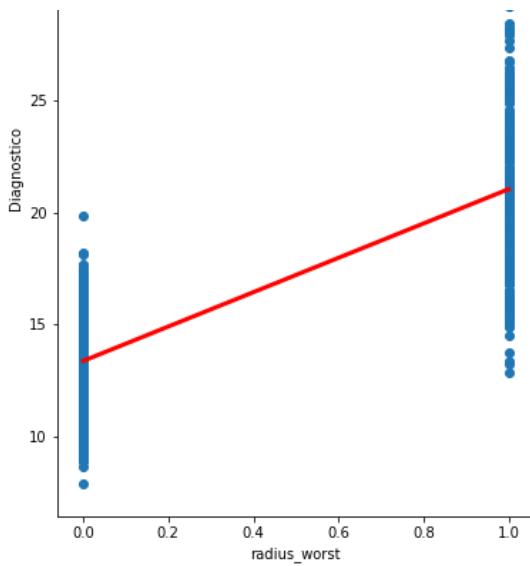
plt.plot(x, (a * x + b), label = 'y = {:.1f}x + {:.0f}'.format(a, b), color = 'red', linewidth = 3)
plt.scatter(x, y)
plt.legend(loc='upper left')
plt.xlabel("radius_worst")
plt.ylabel("Diagnostico")
plt.show()

/Users/mayraberrones/miniconda3/lib/python3.6/site-packages/ipykernel_launcher.py:10: FutureWarning:
Passing list-likes to .loc or [] with any missing label will raise
KeyError in the future, you can use .reindex() as an alternative.
```

See the documentation here:
<https://pandas.pydata.org/pandas-docs/stable/indexing.html#deprecate-loc-reindex-listlike>
 # Remove the CWD from sys.path while we load stuff.

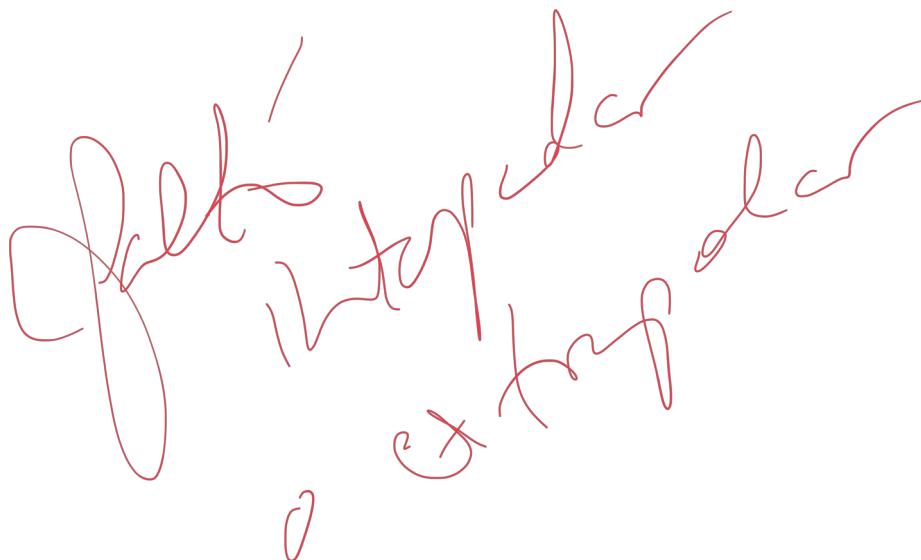
```
y = f(x) = 7.6454 x + 13.3846
error 0.2622946898839285
valor p 2.2434157833955287e-115
pendiente significativo
R^2 0.596379622336683
```





Como se observa en las gráficas anteriores, las dos columnas que se mostraron con mayor correlación resultan tener una pendiente significativa en el radio peor.

En cuanto a los valores de la función, comparandolo con los resultados demostrados en la práctica de la Dra. Elisa, se puede concluir que es de media del radio, y de 13 en la media del peor radio, lo cual da como resultado los primeros umbrales para saber cuando una anomalía tiene mayor riesgo.



Join GitHub today
GitHub is home to over 31 million developers working together to host and review code, manage projects, and build software together.

Sign up

Branch: master ▾ Ciencia_de_Datos / Práctica 7.ipynb

mayaberrones94 ult 1 contributor

fcd292e 3 days ago

541 lines (540 sloc) | 136 KB

Raw Blame History

Práctica 7: Regresión múltiple con statsmodels.

El objetivo de esta práctica es encontrar por lo menos un modelo de regresión múltiple para intentar tomar los resultados como un clasificador de una variable de interés en el proyecto.

Para los datos que se han utilizado a lo largo de este trabajo, el dato de mayor interés y en el que se basa uno de los principales objetivos que se tenían en la práctica 1, es la de clasificar entre anomalías benignas y malignas.

Regresando a las prácticas anteriores, se puede ver que las características que se describen en las columnas entre las anomalías malignas y benignas muchas veces se cruzan, por lo que no es posible hacer una regresión lineal simple para la clasificación. En esta práctica se utilizaran herramientas de statsmodels para poder hacer un clasificador decente.

Primeramente, y para poder utilizar la columna de diagnostico contra las demás columnas que se han estado utilizando, se convierte la M por un 1 y la B por un 0. De esta manera, podemos interactuar con la columna de diagnostico como nuestro dato objetivo.

Después se toman las dos columnas que resultaron ser más contundentes en la práctica anterior, las cuales son 'Radius_mean' y 'Radius_worst'.

```
In [3]: import matplotlib.pyplot as plt
import statsmodels.api as sm
from numpy import isnan
import pandas as pd
import ssl

if getattr(ssl, '_create_unverified_context', None):
    ssl._create_default_https_context = ssl._create_unverified_context
o = pd.read_csv("https://raw.githubusercontent.com/mayaberrones94/Ciencia_de_Datos/master/bp_w-data.csv")
o.diagnosis.replace(to_replace = dict(M = 1, B = 0), inplace = True)

dim = pd.concat([o.loc[o.diagnosis == 1], o.loc[~isnan(o.radius_mean)]])
dib = pd.concat([o.loc[o.diagnosis == 0], o.loc[~isnan(o.radius_mean)]])

for datos in [dim, dib]:
    d = pd.DataFrame(datos, columns = ["diagnosis", "radius_mean", "radius_worst"])
    d = d.dropna()
    n = len(d)
    if n >= 8:
        y = d["diagnosis"]
        x = d[["radius_mean", "radius_worst"]]
```

```

x = sm.add_constant(x)
m = sm.OLS(y, x).fit()
print(datos.diagnosis.unique()[0])
print(m.summary())

```

1

OLS Regression Results

Dep. Variable:	diagnosis	R-squared:	0.564			
Model:	OLS	Adj. R-squared:	0.563			
Method:	Least Squares	F-statistic:	502.5			
Date:	Fri, 22 Mar 2019	Prob (F-statistic):	7.75e-141			
Time:	17:24:43	Log-Likelihood:	-240.10			
No. Observations:	781	AIC:	486.2			
Df Residuals:	778	BIC:	500.2			
Df Model:	2					
Covariance Type:	nonrobust					
coef	std err	t	P> t	[0.025	0.975]	
const	-0.6502	0.053	-12.225	0.000	-0.755	-0.546
radius_mean	-0.0287	0.012	-2.388	0.017	-0.052	-0.005
radius_worst	0.0924	0.009	10.589	0.000	0.075	0.110

Omnibus: 63.143 Durbin-Watson: 1.471
 Prob(Omnibus): 0.000 Jarque-Bera (JB): 33.321
 Skew: 0.342 Prob(JB): 5.81e-08
 Kurtosis: 2.255 Cond. No. 110.

Warnings:
 [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
 0

OLS Regression Results

Dep. Variable:	diagnosis	R-squared:	0.617			
Model:	OLS	Adj. R-squared:	0.617			
Method:	Least Squares	F-statistic:	744.6			
Date:	Fri, 22 Mar 2019	Prob (F-statistic):	2.86e-193			
Time:	17:24:43	Log-Likelihood:	-66.178			
No. Observations:	926	AIC:	138.4			
Df Residuals:	923	BIC:	152.8			
Df Model:	2					
Covariance Type:	nonrobust					
coef	std err	t	P> t	[0.025	0.975]	
const	-0.7317	0.043	-17.157	0.000	-0.815	-0.648
radius_mean	-0.0826	0.011	-7.197	0.000	-0.105	-0.060
radius_worst	0.1363	0.009	16.027	0.000	0.120	0.153

Omnibus: 155.016 Durbin-Watson: 1.608
 Prob(Omnibus): 0.000 Jarque-Bera (JB): 270.780
 Skew: 1.034 Prob(JB): 1.59e-59
 Kurtosis: 4.655 Cond. No. 106.

Warnings:
 [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Una vez que se tienen estos datos, se puede seguir a ver la relevancia que tienen estas columnas en contraste contra el diagnóstico, algo que no se había realizado en prácticas anteriores.

Se separa los datos de las anomalías malignas y benignas, para ver si el resultado es diferente.

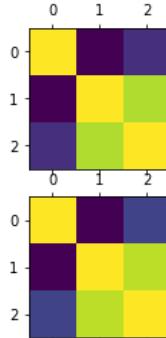
```
In [4]: f = plt.figure()
d = dim
d = pd.DataFrame(d, columns = ["diagnosis", "radius_mean", "radius_worst"])
d = d.dropna()
print(d.corr())
sf = f.add_subplot(211)
tmp = sf.matshow(d.corr())
```

```

d = dib
d = pd.DataFrame(d, columns = ["diagnosis", "radius_mean", "radius_worst"])
d = d.dropna()
print(d.corr())
sf = f.add_subplot(212)
tmp = sf.matshow(d.corr())

```

	diagnosis	radius_mean	radius_worst
diagnosis	1.000000	0.707661	0.748644
radius_mean	0.707661	1.000000	0.965054
radius_worst	0.748644	0.965054	1.000000
	diagnosis	radius_mean	radius_worst
diagnosis	1.000000	0.714760	0.771940
radius_mean	0.714760	1.000000	0.971174
radius_worst	0.771940	0.971174	1.000000



Por los resultados de la tabla que se generó, se puede apreciar que los datos de benigno tienen una correlación apenas más alta que la de maligno en cuanto al resultado del diagnóstico. Despues se puede realizar una gráfica para comprobar lo que se puede observar en el mapa de calor.

```

In [5]: d = pd.DataFrame(o, columns = ["diagnosis", "radius_mean", "radius_worst"])
d = d.dropna()
print(len(d))
tmp = pd.plotting.scatter_matrix(d, figsize = (9, 9), s = 500)
y = d["diagnosis"]
x = d[["radius_mean", "radius_worst"]]
x = sm.add_constant(x)
m = sm.OLS(y, x).fit()
print(m.summary())

```

569

OLS Regression Results

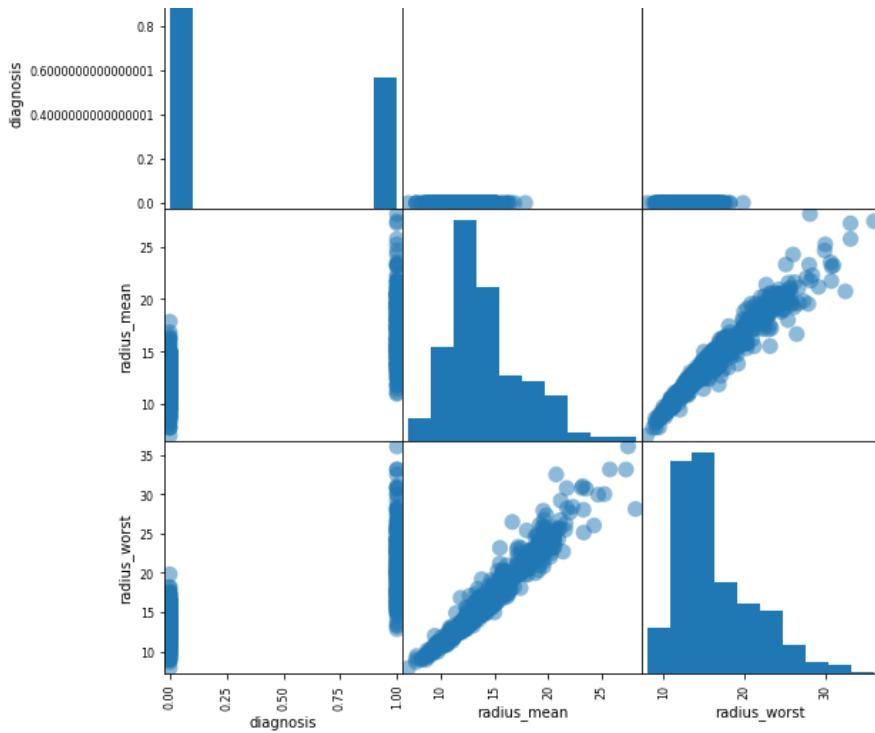
```

=====
Dep. Variable: diagnosis R-squared:      0.612
Model:          OLS   Adj. R-squared:   0.610
Method:         Least Squares F-statistic:    445.5
Date: Fri, 22 Mar 2019 Prob (F-statistic):  6.16e-117
Time: 17:24:52 Log-Likelihood: -124.87
No. Observations: 569 AIC:            255.7
Df Residuals: 566 BIC:            268.8
Df Model: 2
Covariance Type: nonrobust
=====
      coef  std err      t      P>|t|      [ 0.025      0.975]
-----
const     -0.7553    0.059   -12.823    0.000     -0.871     -0.640
radius_mean -0.0521    0.015   -3.549    0.000     -0.081     -0.023
radius_worst  0.1146    0.011   10.700    0.000      0.094     0.136
=====
Omnibus: 37.687 Durbin-Watson: 1.720
Prob(Omnibus): 0.000 Jarque-Bera (JB): 43.574
Skew: 0.671 Prob(JB): 3.45e-10
Kurtosis: 3.186 Cond. No. 106.
=====
```

Warnings:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.





```
In [13]: d = pd.DataFrame(dib, columns = ["diagnosis", "radius_mean", "radius_worst"])
d = d.dropna()
print(len(d))
tmp = pd.plotting.scatter_matrix(d, figsize = (9, 9), s = 500)
y = d["diagnosis"]
x = d[["radius_mean", "radius_worst"]]
x = sm.add_constant(x)
m = sm.OLS(y, x).fit()
print(m.summary())
```

926

OLS Regression Results

```
=====
Dep. Variable: diagnosis R-squared:      0.617
Model: OLS Adj. R-squared:      0.617
Method: Least Squares F-statistic:   744.6
Date: Fri, 22 Mar 2019 Prob (F-statistic): 2.86e-193
Time: 14:21:26 Log-Likelihood: -66.178
No. Observations: 926 AIC:            138.4
Df Residuals:    923 BIC:            152.8
Df Model:        2
Covariance Type: nonrobust
=====
```

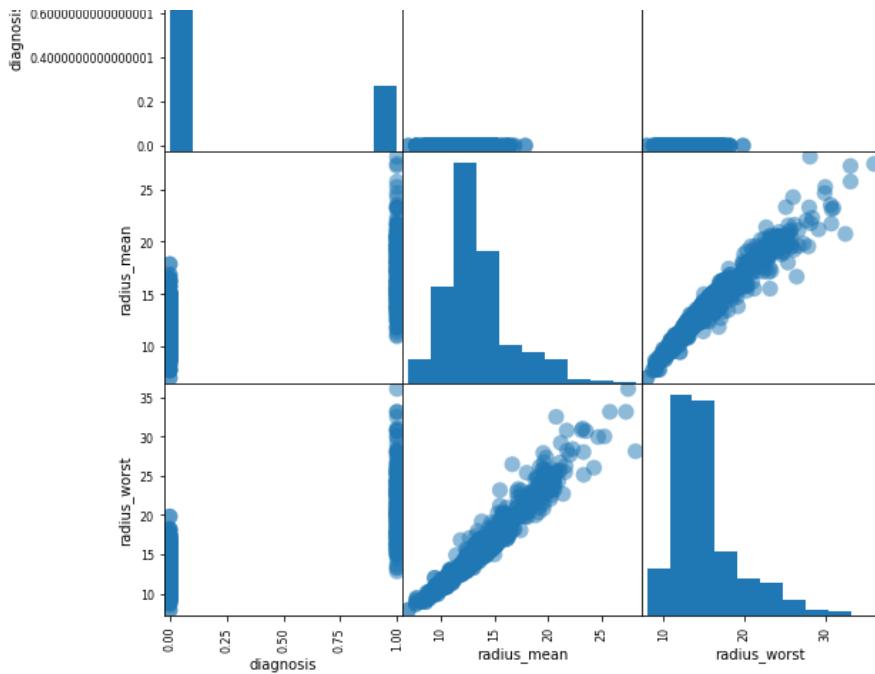
	coef	std err	t	P> t	[0.025	0.975]
const	-0.7317	0.043	-17.157	0.000	-0.815	-0.648
radius_mean	-0.0826	0.011	-7.197	0.000	-0.105	-0.060
radius_worst	0.1363	0.009	16.027	0.000	0.120	0.153

```
=====
Omnibus:           155.016 Durbin-Watson:       1.608
Prob(Omnibus):    0.000 Jarque-Bera (JB):    270.780
Skew:              1.034 Prob(JB):          1.59e-59
Kurtosis:          4.655 Cond. No.         106.
=====
```

Warnings:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.





A partir de aquí, se quiere lograr formar un clasificador que prediga la clasificación de diagnóstico de una anomalía basándose en los datos que se le alimente de estas dos características de radio.

Como se explica en el ejemplo de práctica realizado por la Dra. Elisa, el objetivo de clasificar es el de diagnóstico, por lo que lo se coloca en el valor de Y. En el valor de X se colocan las otras dos características que ayudarán a formar el clasificador.

```
In [6]: d = pd.DataFrame(o, columns = ["diagnosis", "radius_mean", "radius_worst"])
d = d.dropna()
y = d["diagnosis"]
x = d[["radius_mean", "radius_worst"]]
m = sm.OLS(y, x).fit()
comp = pd.DataFrame(d, columns = ["diagnosis"])
comp['pron'] = m.predict(x)
print(comp['pron'])
comp['error'] = comp.diagnosis - comp.pron
comp['absE'] = pd.DataFrame.abs(comp['error'])
orden = comp.sort_values(by = ['absE'])
mejores = orden.head(10)
mejores.insert(0, 'tipo', 'mejores')
peores = orden.tail(10)
peores.insert(0, 'tipo', 'peores')
pd.concat([mejores, peores])
n = len(comp)
for e in [.1, .5, 1]:
    k = sum(comp.absE < e)
    print(k, "de", n, "= {:.2f}% estuvieron dentro de".format(100 * k / n), e, "punto del correcto")
0      1.346098
1      0.824218
2      0.726702
3      0.641404
4      0.438694
5      0.560028
6      0.856954
7      0.621027
8      0.467072
9      0.490857
10     0.593593
11     0.853917
12     0.354904
13     0.206509
14     0.257377
15     0.546358
```

```

--      -----
16    0.807422
17    0.888307
18    1.370932
19    0.304907
20    0.277400
21    0.147312
22    0.514208
23    1.462267
24    1.772819
25    0.939949
26    0.567725
27    0.515256
28    0.911527
29    0.467103
...
539   0.190084
540   0.150213
541   0.338652
542   0.342726
543   0.231526
544   0.236360
545   0.333448
546   0.185093
547   0.121109
548   0.240091
549   0.413151
550   0.163082
551   0.179574
552   0.220025
553   0.109013
554   0.204272
555   0.117619
556   0.106721
557   0.207651
558   0.186314
559   0.194507
560   0.249130
561   0.149551
562   0.437688
563   0.638619
564   0.732119
565   0.670788
566   0.454569
567   0.952011
568   0.316002
Name: pron, Length: 569, dtype: float64
30 de 569 = 5.27% estuvieron dentro de 0.1 punto del correcto
500 de 569 = 87.87% estuvieron dentro de 0.5 punto del correcto
568 de 569 = 99.82% estuvieron dentro de 1 punto del correcto

```

Se imprimió la parte de predicción porque necesitaba ver cual era el resultado para darle los datos correctos al siguiente paso, que es el que va a decir si la herramienta que se crea con estos parámetros es suficientemente buena para clasificar las anomalías benignas y malignas.

```

In [41]: d = pd.DataFrame(o, columns = ["diagnosis", "radius_mean", "radius_worst"])
d = d.dropna()
y = d["diagnosis"]
x = d[["radius_mean", "radius_worst"]]
m = sm.OLS(y, x).fit()
print(m.summary())
comp = pd.DataFrame(d, columns = ["diagnosis"])
comp['pron'] = m.predict(x)
comp['error'] = comp.diagnosis - comp.pron
comp['absE'] = pd.DataFrame.abs(comp['error'])
orden = comp.sort_values(by = ['absE'])
mejores = orden.head(10)
mejores.insert(0, 'tipo', 'mejores')
peores = orden.tail(10)
peores.insert(0, 'tipo', 'peores')
pd.concat([mejores, peores])
n = len(comp)
for e in [1, 5, 10]:

```

```

k = sum(comp.absE < e)
print(k, "de", n, "= {:.2f}% estuvieron dentro de".format(100 * k / n), e, \
      "punto{:s} del CF correcto".format(" if e == 1 else "s"))
fp = sum((comp.diagnosis == 0) & (comp.pron >= .5))
fn = sum((comp.diagnosis == 1) & (comp.pron < .5))
tp = sum((comp.diagnosis == 1) & (comp.pron >= .5))
tn = sum((comp.diagnosis == 0) & (comp.pron < .5))

print('Clasificación')
print(fp, fn, tp, tn)
tt = tp + tn + fp + fn
print('sensibilidad', tp / (tp + fn))
print('especificidad', tn / (tn + fp))
print('precisión', (tp + tn) / tt)

```

```

OLS Regression Results
=====
Dep. Variable: diagnosis R-squared:          0.685
Model:           OLS   Adj. R-squared:        0.684
Method:          Least Squares F-statistic:     617.8
Date:            Fri, 22 Mar 2019 Prob (F-statistic): 3.90e-143
Time:             15:19:58 Log-Likelihood:    -197.43
No. Observations: 569   AIC:                  398.9
Df Residuals:    567   BIC:                  407.5
Df Model:         2
Covariance Type: nonrobust
=====
      coef    std err      t      P>|t|      [ 0.025    0.975]
-----
radius_mean    -0.1755    0.013   -13.927    0.000     -0.200    -0.151
radius_worst     0.1774    0.011    16.414    0.000      0.156     0.199
=====
Omnibus:            57.716 Durbin-Watson:       1.691
Prob(Omnibus):      0.000 Jarque-Bera (JB):    73.780
Skew:                 0.877 Prob(JB):        9.52e-17
Kurtosis:              2.803 Cond. No.          25.8
=====
```

Warnings:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
568 de 569 = 99.82% estuvieron dentro de 1 punto del CF correcto
569 de 569 = 100.00% estuvieron dentro de 5 puntos del CF correcto
569 de 569 = 100.00% estuvieron dentro de 10 puntos del CF correcto
Clasificación
1 63 149 356
sensibilidad 0.7028301886792453
especificidad 0.9971988795518207
precisión 0.8875219683655536

Para la medición de eficiencia en cualquier herramienta de aprendizaje de máquina, se tienen la sensibilidad, especificidad, y precisión. Dependiendo de donde se quiera utilizar la herramienta, cada una de estas opciones adquiere **mas relevancia**.

Para poder realizar los falsos positivos, falsos negativos, verdaderos positivos y verdaderos negativos, se tuvo que actualizar a los valores que se esperan del diagnostico. Trabajando con una base de datos diferente en Keras, esto es lo que hace para poder darme el valor de especificidad.

En el caso de estos resultados, a pesar de que tiene buena especificidad, también se le tiene que dar importancia a la precisión, ya que esos 63 falsos negativos significa que clasificó a 63 pacientes con que su cáncer era benigno, siendo que el resultado era maligno.

Claramente, esto en una herramienta de la vida real resultaría muy poco confiable, pero en investigación puede ayudar el tomar otros parámetros tal vez no tan relacionados como lo estaban el 'radius_mean' y el 'radius_worst'

Join GitHub today
GitHub is home to over 31 million developers working together to host and review code, manage projects, and build software together.

Sign up

Branch: master ▾ Ciencia_de_Datos / Práctica 8.ipynb

mayaberrones94 cambio 364be56 20 hours ago

1 contributor

366 lines (365 sloc) | 521 KB

Raw Blame History

Práctica 8: Análisis de varianza y de componentes principales.

En esta práctica se realizará una prueba de ANOVA para revisar los datos con los que se ha estado trabajando a lo largo de estos trabajos. Una prueba ANOVA es una herramienta para cuantificar si o no una variable o factor tiene un efecto estadísticamente significativo en una variable de interés. Se realiza a partir de la salida del modelo de regresión que se obtiene con otra versión de ols.

Adicionalmente se realiza un PCA (Principal Component Analysis) es una técnica utilizada para describir un conjunto de datos en términos de nuevas "componentes" no correlacionadas. Los componentes se ordenan por la cantidad de varianza original que describen, por lo que la técnica es útil para reducir la dimensionalidad de un conjunto de datos.

Se inicia cargando de nueva cuenta los datos que se utilizarán.

```
In [1]: import statsmodels.api as sm
from statsmodels.formula.api import ols
from numpy import isnan
import pandas as pd
import ssl

if getattr(ssl, '_create_unverified_context', None):
    ssl._create_default_https_context = ssl._create_unverified_context
o = pd.read_csv("https://raw.githubusercontent.com/mayaberrones94/Ciencia_de_Datos/master/bp_w-data.csv")
o.diagnosis.replace(to_replace = dict(M = 1, B = 0), inplace = True)

d = pd.DataFrame(o, columns = ["diagnosis", "radius_mean", "radius_worst"])
d = d.dropna()
m = ols('diagnosis ~ radius_mean + radius_worst', data = d).fit()
a = sm.stats.anova_lm(m, typ = 2)
print(a)
n = len(a)
alpha = 0.01
for i in range(n):
    print("{:s} {:s}es significativo".format(a.index[i], "" if a['PR(>F)'][i] < alpha else "no "))
    sum_sq      df          F      PR(>F)
radius_mean   1.149872   1.0   12.595381  4.188203e-04
radius_worst  10.452598   1.0   114.494906 1.846153e-24
Residual     51.671911  566.0        NaN         NaN
radius_mean es significativo
radius_worst es significativo
Residual no es significativo
```

Resumen de los diagnósticos

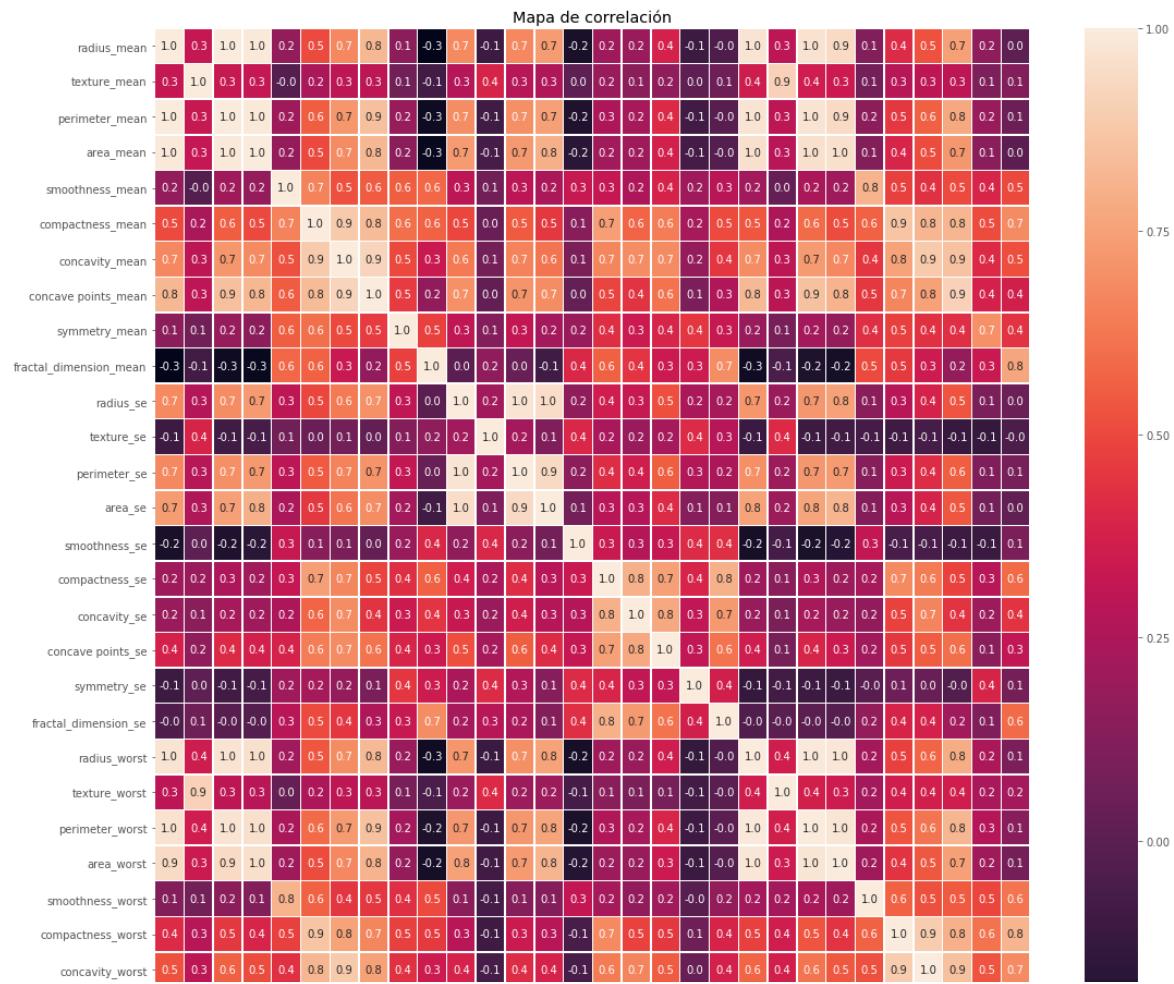
Se utilizan los mismos datos que se encontraron en la práctica pasada como los que tienen mayor impacto en el diagnóstico, más sin embargo podemos ver que el residuo es muy grande, y como se muestra en la práctica de la Dra Elisa, eso significa que nuestro modelo no es muy bueno aún, confirmando lo que se observa en la práctica anterior en la parte en que se revisan los resultados y se sabe que la sensibilidad y presión del modelo no son tan buenas.

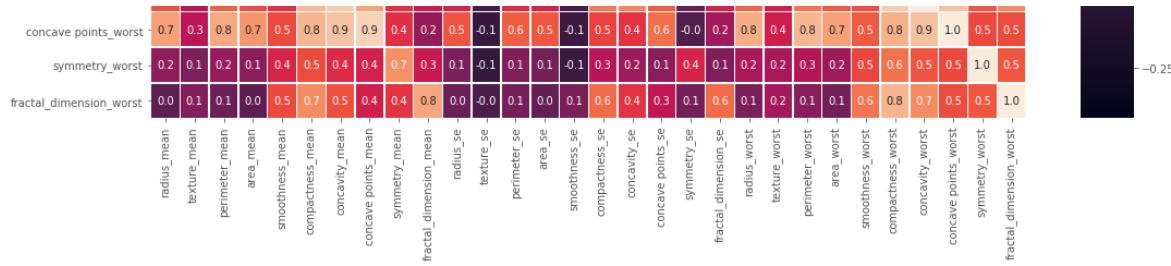
Lo que se recomienda aquí es agregar datos que puedan resultar importantes. Para recordar los datos que tenían mejor correlación, se realiza de nueva cuenta el mapa de calor, sacando como datos principales el 'radius mean', 'radius worst', y agregando los datos de 'perimeter mean', 'area mean' y 'perimeter worst' que son con los que mejor se relacionaron.

In [3]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from pandas.tools import plotting
from scipy import stats
plt.style.use("ggplot")
import warnings
warnings.filterwarnings("ignore")
from scipy import stats

data = pd.read_csv("/Users/mayraberrones/Documents/GitHub/Ciencia_de_Datos/w-data.csv")
data = data.drop(['Unnamed: 32', 'id'], axis = 1)
f,ax=plt.subplots(figsize = (18,18))
sns.heatmap(data.corr(), annot= True, linewidths=0.5, fmt = ".1f", ax=ax)
plt.xticks(rotation=90)
plt.yticks(rotation=0)
plt.title('Mapa de correlación')
plt.savefig('graph.png')
plt.show()
```





De nueva cuenta se realizan los cálculos del ANOVA. El residuo disminuye un poco, y se muestra el 'perimeter mean' como el que no tiene significancia.

```
In [3]: d = pd.DataFrame(o, columns = ["diagnosis", "radius_mean", "radius_worst", "perimeter_mean", "area_mean", "perimeter_worst"])
d = d.dropna()
m = ols('diagnosis ~ radius_mean + radius_worst + perimeter_mean + area_mean + perimeter_worst', d)
a = sm.stats.anova_lm(m, typ = 2)
print(a)
n = len(a)
alpha = 0.01
for i in range(n):
    print("{:s} {:s}es significativo".format(a.index[i], "" if a['PR(>F)'][i] < alpha else "no "))

```

	sum_sq	df	F	PR(>F)
radius_mean	2.823529	1.0	35.721341	4.042227e-09
radius_worst	2.279007	1.0	28.832421	1.155901e-07
perimeter_mean	3.698251	1.0	46.787723	2.067895e-11
area_mean	2.501221	1.0	31.643728	2.921120e-08
perimeter_worst	0.294411	1.0	3.724690	5.411561e-02
Residual	44.501319	563.0	Nan	Nan

radius_mean es significativo
 radius_worst es significativo
 perimeter_mean es significativo
 area_mean es significativo
 perimeter_worst no es significativo
 Residual no es significativo

Después se hacen los cálculos de las interacciones entre los nuevos datos que se le están dando al ANOVA. Aquí se muestra que ninguna de las interacciones que estan con el 'area mean' son significativas, y la suma cuadrada del residuo disminuyó significativamente, lo que es bueno.

```
In [5]: d = pd.DataFrame(o, columns = ["diagnosis", "radius_mean", "radius_worst", "perimeter_mean", "area_mean", "perimeter_worst"])
d = d.dropna()
m = ols('diagnosis ~ radius_worst * perimeter_worst + radius_worst * radius_mean + radius_mean * area_mean + radius_worst * area_mean + perimeter_worst * area_mean', \
        data = d)
a = sm.stats.anova_lm(m, typ = 2)
print(a)
n = len(a)
alpha = 0.01
for i in range(n):
    print("{:s} {:s}es significativo".format(a.index[i], "" if a['PR(>F)'][i] < alpha else "NO "))

```

	sum_sq	df	F	PR(>F)
radius_worst	0.040954	1.0	0.646218	4.218099e-01
perimeter_worst	1.825917	1.0	28.811187	1.171105e-07
radius_worst:perimeter_worst	2.724444	1.0	42.989062	1.253716e-10
radius_mean	5.488121	1.0	86.597183	2.997318e-19
radius_worst:radius_mean	2.716680	1.0	42.866556	1.328781e-10
area_mean	0.160617	1.0	2.534379	1.119560e-01
radius_mean:area_mean	0.315565	1.0	4.979308	2.604795e-02
radius_worst:area_mean	0.310076	1.0	4.892702	2.737407e-02
perimeter_worst:area_mean	0.148233	1.0	2.338979	1.267378e-01
Residual	35.426781	559.0	Nan	Nan

radius_worst NO es significativo
 perimeter_worst es significativo
 radius_worst:perimeter_worst es significativo
 radius_mean es significativo

```

radius_mean es significativo
radius_worst:radius_mean es significativo
area_mean NO es significativo
radius_mean:area_mean NO es significativo
radius_worst:area_mean NO es significativo
perimeter_worst:area_mean NO es significativo
Residual NO es significativo

```

Por último se realiza una gráfica de dos componentes, mostrando todos los elementos menos el 'area mean' que es el que no tenía significancia en la interacción con los demás elementos.

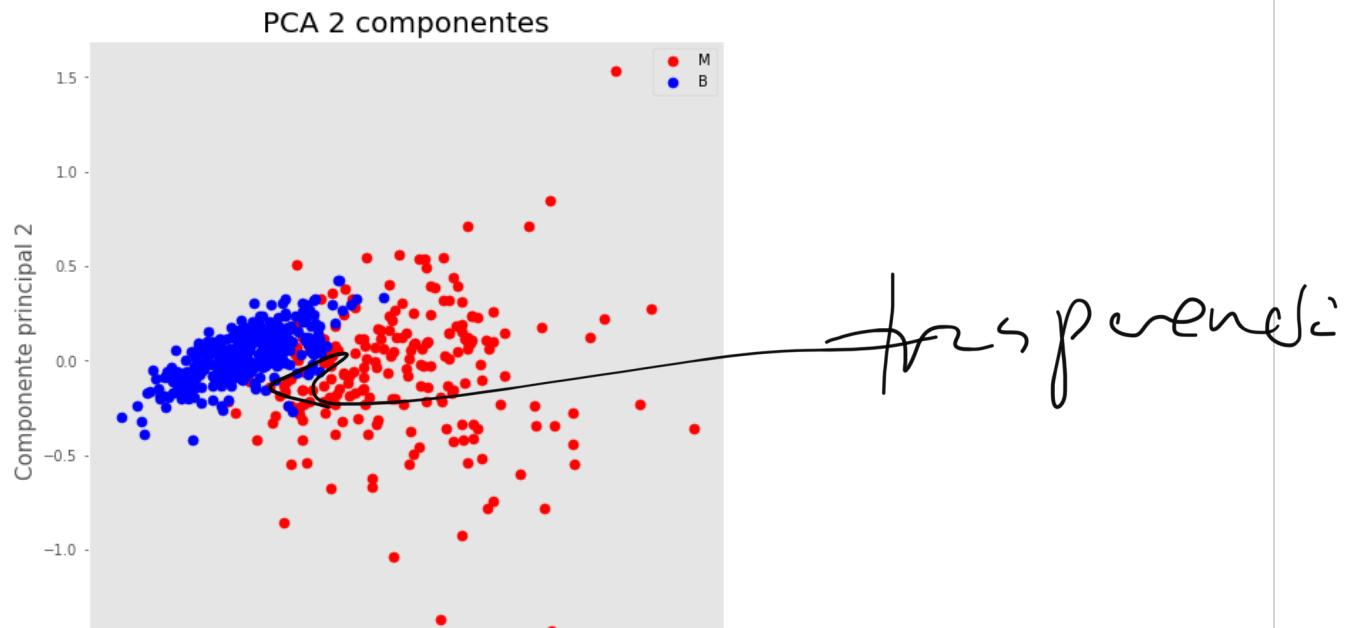
```

In [7]: import pandas as pd
from sklearn.preprocessing import StandardScaler

df = pd.DataFrame(data, columns = ["diagnosis", "radius_mean", "radius_worst", "perimeter_mean", "perimeter_worst"])

features = ["radius_mean", "radius_worst", "perimeter_mean", "perimeter_worst"]
# Separating out the features
x = df.loc[:, features].values
# Separating out the target
y = df.loc[:,['diagnosis']].values
# Standardizing the features
x = StandardScaler().fit_transform(x)
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(x)
principalDf = pd.DataFrame(data = principalComponents
                            , columns = ['principal component 1', 'principal component 2'])
finalDf = pd.concat([principalDf, df[['diagnosis']]], axis = 1)
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Componente principal 1', fontsize = 15)
ax.set_ylabel('Componente principal 2', fontsize = 15)
ax.set_title('PCA 2 componentes', fontsize = 20)
targets = ['M', 'B']
colors = ['r', 'b']
for target, color in zip(targets,colors):
    indicesToKeep = finalDf['diagnosis'] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
              , finalDf.loc[indicesToKeep, 'principal component 2']
              , c = color
              , s = 50)
ax.legend(targets)
ax.grid()

```





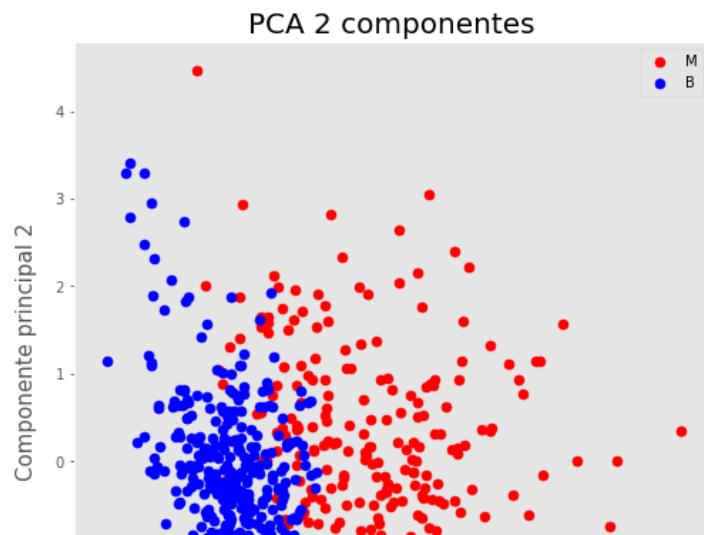
Viendo la gráfica podemos sacar en conclusión que la parte difícil de clasificar sería el cluster ese que se hace en la parte media, ya que muchos de los puntos rojos y azules pueden separarse claramente, lo cual muestra porque el resultado de la práctica anterior la especificidad resulta de un 99 porciento, mientras que la sensibilidad baja a ser de un 70 porciento.

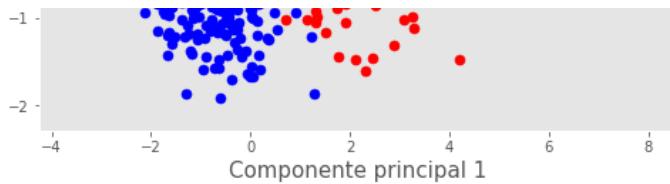
Por último solo se quería apreciar la diferencia de incluir las columnas que no mostraban correlación con los demás datos ('area mean', 'texture mean' y 'fractal dimension mean') y quitando los datos que parecían tener mas relación entre ellos y los demás ('radius mean' y 'radius worst'). Esto se hace con el fin de sobresaltar la cualidad de la herramienta de PCA.

```
In [11]: import pandas as pd
from sklearn.preprocessing import StandardScaler

df = pd.DataFrame(data, columns = ["diagnosis", "perimeter_mean", "perimeter_worst", "area_mean", "texture_mean", "fractal_dimension_mean"])

features = [ "perimeter_mean", "perimeter_worst", "area_mean", "texture_mean", "fractal_dimension_mean"]
# Separating out the features
x = df.loc[:, features].values
# Separating out the target
y = df.loc[:,['diagnosis']].values
# Standardizing the features
x = StandardScaler().fit_transform(x)
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(x)
principalDf = pd.DataFrame(data = principalComponents
                           , columns = ['principal component 1', 'principal component 2'])
finalDf = pd.concat([principalDf, df[['diagnosis']]], axis = 1)
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Componente principal 1', fontsize = 15)
ax.set_ylabel('Componente principal 2', fontsize = 15)
ax.set_title('PCA 2 componentes', fontsize = 20)
targets = ['M', 'B']
colors = ['r', 'b']
for target, color in zip(targets,colors):
    indicesToKeep = finalDf['diagnosis'] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
              , finalDf.loc[indicesToKeep, 'principal component 2']
              , c = color
              , s = 50)
ax.legend(targets)
ax.grid()
```





Ahora se puede apreciar una separación mas visible de los datos, estando estos más dispersos, pero aun así, existen muchos de los puntos que se mezclan entre ellos. Será interesante ver si el prácticas siguientes podemos averiguar que hacer en este caso para desbaratar el cluster.

Join GitHub today

GitHub is home to over 31 million developers working together to host and review code, manage projects, and build software together.

Sign up

Dismiss

Branch: master ▾ Ciencia_de_Datos / Práctica 9.ipynb

Find file Copy path

mayaberrones94 pr9 8ef795b 11 hours ago

1 contributor

1709 lines (1708 sloc) | 1020 KB

Raw Blame History

Práctica 9: Pronósticos con statsmodels.

Para practicar el arte de pronosticar, es decir, predecir un valor que tomará una variable a partir de valores que haya tomado en la anterioridad. Para los datos que se han estado utilizando a lo largo de las prácticas anteriores no es posible hacer un pronóstico como el que se realizó en el ejemplo de la práctica de la Dra Elisa, ya que los datos no tienen un máximo o mínimo fijo, en el sentido en el que muchos de los resultados de una anomalía maligna se sobreponen en los resultados de benigno, por lo que no es confiable predecir el resultado de la anomalía con estos datos.

Para poder utilizar las herramientas recomendadas se hará uso de una nueva serie de datos. En este caso se investigó en la página de Kaggle, y se encontró una base de datos sobre la ciudad de Barcelona, España sobre la contaminación, nacimientos, defunciones, accidentes automovilísticos, etc. Todos ellos transcurridos en el año 2017.

Los únicos datos que cumplían con los requerimientos para estas experimentaciones son los de accidentes de automóviles, ya que tiene divisiones por mes, día de la semana, y hasta hora ocurrida en los diferentes municipios de Barcelona.

Como en otras prácticas, se inicia cargando los datos en pandas.

```
In [2]: import matplotlib.pyplot as plt
from numpy import asarray
import pandas as pd
import ssl

if getattr(ssl, '_create_unverified_context', None):
    ssl._create_default_https_context = ssl._create_unverified_context
df = pd.read_csv('/Users/mayaberrones/Downloads/barcelona-data-sets/accidents_2017.csv')
```

```
In [89]: df.shape
```

```
Out[89]: (10339, 15)
```

A continuación se muestra cómo está conformado este conjunto.

```
In [90]: df.head(15)
```

	Id	District	Neig	Street	Weekday	Month	Day	acc	p_day	m_injuries	s_injuries	Victims
0	2017S008429	Unknown	Unknown	Número 27 ...	Friday	October	13	8	Morning	2	0	2

1	2017S007316	Unknown	Unknown	3 Zona Franca / Número 50 Zona Franca ...	Friday	September	1	13	Morning	2	0	2
2	2017S010210	Unknown	Unknown	Litoral (Besòs) ...	Friday	December	8	21	Afternoon	5	0	5
3	2017S006364	Unknown	Unknown	Número 3 Zona Franca ...	Friday	July	21	2	Night	1	0	1
4	2017S004615	Sant Martí	el Camp de l'Arpa del Clot	Las Navas de Tolosa ...	Thursday	May	25	14	Afternoon	1	0	1
5	2017S007775	Sant Martí	el Camp de l'Arpa del Clot	Indústria / Trinxant ...	Wednesday	September	20	12	Morning	1	0	1
6	2017S004484	Sant Martí	el Camp de l'Arpa del Clot	Trinxant / Indústria ...	Saturday	May	20	21	Afternoon	1	0	1
7	2017S010680	Sant Martí	el Camp de l'Arpa del Clot	Indústria ...	Tuesday	December	26	20	Afternoon	2	0	2
8	2017S005152	Sant Martí	el Camp de l'Arpa del Clot	Indústria ...	Monday	June	12	15	Afternoon	1	0	1
9	2017S003932	Sant Martí	el Camp de l'Arpa del Clot	Maragall ...	Wednesday	May	3	20	Afternoon	1	0	1
10	2017S010348	Sant Martí	el Camp de l'Arpa del Clot	Indústria ...	Thursday	December	14	20	Afternoon	1	0	1
11	2017S000245	Sant Martí	el Camp de l'Arpa del Clot	ST ANTONI M CLARET / Pg Maragall ...	Wednesday	January	11	7	Morning	1	0	1
12	2017S005695	Sant Martí	el Camp de l'Arpa del Clot	Sant Antoni Maria Claret / Guinardó ...	Friday	June	30	12	Morning	1	0	1
13	2017S007384	Sant Martí	el Camp de l'Arpa del Clot	Sant Antoni Maria Claret ...	Monday	September	4	16	Afternoon	0	1	1
14	2017S008080	Sant	el Camp de l'Arpa	Sant Antoni	Saturday	September	30	19	Afternoon	1	0	1

		Martí	del Clot	Maria Claret ...	Saturday	Sunday	Monday	Tuesday	Wednesday	Thursday	Friday		
--	--	-------	----------	------------------	----------	--------	--------	---------	-----------	----------	--------	--	--

Para saber con cuantas localidades se cuenta, se utiliza la herramienta de 'unique()', con la cual se observa que se tienen 10 localidades.

```
In [91]: df['District'].unique()
```

```
Out[91]: array(['Unknown', 'Sant Martí', 'Ciutat Vella', 'Eixample',
   'Sants-Montjuïc', 'Les Corts', 'Sarrià-Sant Gervasi', 'Gràcia',
   'Horta-Guinardó', 'Nou Barris', 'Sant Andreu'], dtype=object)
```

Porque en los nombres de los conjuntos se utilizan nombres extraños y algunas letras y formas de puntuación que causan conflicto con el csv al momento de mandar a llamar sus diferentes variables, se opta por cambiarlas a números. Se cambian las columnas de 'District', 'Weekday' y 'Month', ya que son con las que se estará trabajando a lo largo de esta práctica.

```
In [92]: df['District'] = pd.factorize(df.District)[0] + 1
df['Weekday'] = pd.factorize(df.Weekday)[0] + 1
df['Month'] = pd.factorize(df.Month)[0] + 1
df.head(2)
```

	Id	District	Neig	Street	Weekday	Month	Day	acc	p_day	m_injuries	s_injuries	Victims	Vehicles	Longitude	Latitude
0	2017S008429	1	Unknown	Número 27 ...	1	1	13	8	Morning	2	0	2	2	2.125624	41.340045
1	2017S007316	1	Unknown	Número 3 Zona Franca / Número 50 Zona Franca ...	1	2	1	13	Morning	2	0	2	2	2.120452	41.339426

Una vez que se tienen de manera más legible las partes importantes, se puede despreciar las que causan ruido solamente. Estas columnas son la de 'Id', 'Neig' y 'Street'.

```
In [93]: d = df.drop(["Id", "Neig", "Street"], axis=1)
d.head(5)
```

	District	Weekday	Month	Day	acc	p_day	m_injuries	s_injuries	Victims	Vehicles	Longitude	Latitude
0	1	1	1	13	8	Morning	2	0	2	2	2.125624	41.340045
1	1	1	2	1	13	Morning	2	0	2	2	2.120452	41.339426
2	1	1	3	8	21	Afternoon	5	0	5	2	2.167356	41.360885
3	1	1	4	21	2	Night	1	0	1	2	2.124529	41.337668
4	2	2	5	25	14	Afternoon	1	0	1	3	2.185272	41.416365

```
In [97]: d.to_csv("nuevos.csv", index = False)
```

Una vez que se acomodaron los datos de manera más bonita, se pasa a un nuevo csv para poder hacerle otras manipulaciones.

```
In [95]: data = pd.read_csv('nuevos.csv')
```

```
In [96]: data.head(5)
```

	District	Weekday	Month	Day	acc	p_day	m_injuries	s_injuries	Victims	Vehicles	Longitude	Latitude
0	1	1	1	13	8	Morning	2	0	2	2	2.125624	41.340045
1	1	1	2	1	13	Morning	2	0	2	2	2.120452	41.339426
2	1	1	3	8	21	Afternoon	5	0	5	2	2.167356	41.360886
3	1	1	4	21	2	Night	1	0	1	2	2.124529	41.337668

o	i	t	+	<1	<	TWENTY	i	u	i	c	L124029	+1.337000
4	2	2	5	25	14	Afternoon	1	0	1	3	2.185272	41.416365

```
In [70]: for i in range(2, 12):
    for k in range(1, 13):
        for j in range(1, 8):
            e = data.loc[data.District == i]
            e1 = e.loc[e.Month == k]
            e2 = e1.loc[e1.Weekday == j]
            # print(e2.Hour.sum())
            #print('-----')
```

Para poder realizar los mismos procedimientos que se muestran en la práctica de la Dra Elisa, se revisó la manera en la que tenía guardados sus datos, para saber de que manera se tenía que acomodar los propios.

Al final, lo que se les realizó fue buscar por distrito - mes - dia, y se sumaron todos los accidentes que pasaron en esa configuración. Esto se repite con todos los datos. Al finalizar se guardan en un nuevo CSV. (Se llama Madrid porque me confundí de ciudad española)

```
In [3]: d = pd.read_csv('/Users/mayraberrones/Downloads/madrid.csv')
```

```
In [74]: d.head(5)
```

```
Out[74]:
```

	Month	weekday	a	b	c	d	e	f	g	h	i	j
0	1	1	262	99	504	252	119	130	117	194	110	68
1	1	2	229	53	499	169	97	118	49	157	100	105
2	1	3	276	89	644	228	131	320	128	189	116	116
3	1	4	199	67	286	194	77	151	48	55	105	46
4	1	5	198	97	641	296	151	250	229	89	159	206

```
In [100]: d.shape
```

```
Out[100]: (84, 12)
```

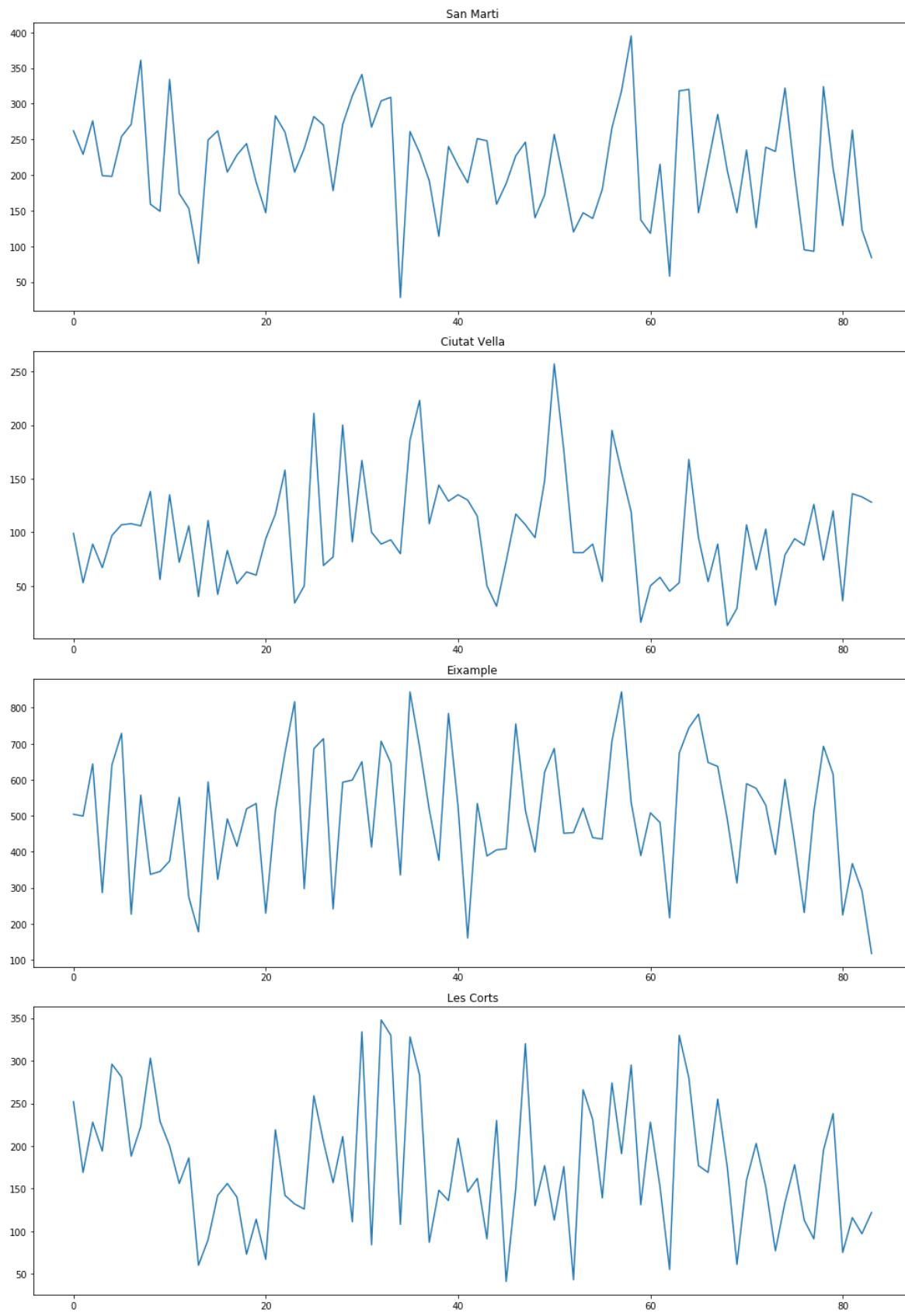
Al finalizar con todos los acomodos de los datos ya se puede iniciar con las pruebas. Para esto se tomaron solo las cuatro localidades que los nombres me gustaron más, para no tener 11 gráficas.

En este caso se utiliza un modelo ARIMA (promedios deslizantes autoregresivos, por auto-regressive integrated moving average en inglés).

```
In [75]: import matplotlib.pyplot as plt
from numpy import asarray
import pandas as pd
import ssl

x = range(len(d))
lbls = ['{:d} {:d}'.format(s, a) for (s, a) in zip(d['weekday'], d['Month'])]
plt.rcParams["figure.figsize"] = [14, 20]
f = plt.figure()
sf = f.add_subplot(411)
y = asarray(d['a'])
sf.plot(x, y)
sf.set_title('San Marti')
sf = f.add_subplot(412)
y = asarray(d['b'])
sf.set_title('Ciutat Vella')
sf.plot(x, y)
sf = f.add_subplot(413)
y = asarray(d['c'])
sf.set_title('Eixample')
sf.plot(x, y)
sf = f.add_subplot(414)
y = asarray(d['d'])
sf.set_title('Les Corts')
sf.plot(x, y)
plt.tight layout()
```

```
plt.show()
```



Esta información se está expresando a lo largo de un año. Que se muestren un poco caóticas puede significar que es difícil predecir cuantos accidentes pueden presentarse en una semana. Los números que se muestran en la parte izquierda de la gráfica no están tan dispersos, por lo que imagino que solo se ven así de largos los picos porque esta redicido el espacio en el que se muestran. Si la gráfica se alargara, se vería más como una linea.

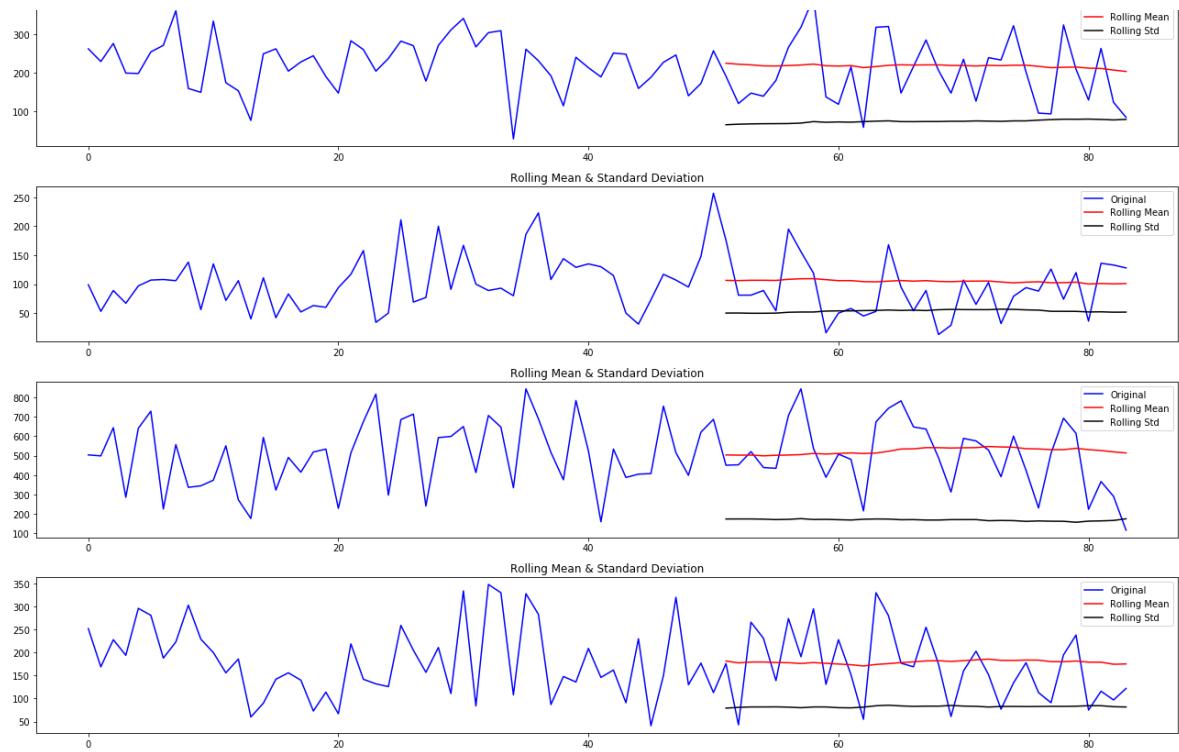
```
In [80]: from statsmodels.tsa.stattools import adfuller
import matplotlib.pyplot as plt
import pandas as pd
import ssl

# rutina de https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/
def test_stationarity(ts, w, r, i):
    rolmean = ts.rolling(w).mean()
    rolstd = ts.rolling(w).std()
    plt.subplot(r, 1, i)
    orig = plt.plot(ts, color='blue', label='Original')
    mean = plt.plot(rolmean, color='red', label='Rolling Mean')
    std = plt.plot(rolstd, color='black', label = 'Rolling Std')
    plt.legend(loc = 'best')
    plt.title('Rolling Mean & Standard Deviation')
    dfoutput = adfuller(ts, autolag='AIC')
    dfoutput = pd.Series(dfoutput[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])
    for key,value in dfoutput[4].items():
        dfoutput['Critical Value {:.3f}'.format(key)] = value
    return 'Results of Dickey-Fuller Test:\n' + '\n'.join(['{:s}\t{:s}'.format(k, v) for (k, v) in dfoutput.items()])

plt.rcParams["figure.figsize"] = [18, 12]
f = plt.figure()
i = 1
lvls = ['a', 'b', 'c', 'd']
r = len(lvls)
t = ''
w = 52
for c in lvls:
    t += test_stationarity(d[c], w, r, i)
    i += 1
plt.tight_layout()
print(t)
```

```
Results of Dickey-Fuller Test:
Test Statistic -7.277
p-value 0.000
#Lags Used 1.000
Number of Observations Used 82.000
Critical Value (1%) -3.513
Critical Value (5%) -2.897
Critical Value (10%) -2.586Results of Dickey-Fuller Test:
Test Statistic -7.319
p-value 0.000
#Lags Used 0.000
Number of Observations Used 83.000
Critical Value (1%) -3.512
Critical Value (5%) -2.897
Critical Value (10%) -2.586Results of Dickey-Fuller Test:
Test Statistic -2.050
p-value 0.265
#Lags Used 7.000
Number of Observations Used 76.000
Critical Value (1%) -3.519
Critical Value (5%) -2.900
Critical Value (10%) -2.587Results of Dickey-Fuller Test:
Test Statistic -8.838
p-value 0.000
#Lags Used 0.000
Number of Observations Used 83.000
Critical Value (1%) -3.512
Critical Value (5%) -2.897
Critical Value (10%) -2.586
```





Se sabe que estos datos son estacionarios. Es sobre los accidentes ocurridos en los 12 meses del año 2017. El link de tutoria no servía en la práctica de la Dra, por lo que estoy guiandome por lo que dice su descripción. En su caso menciona que los valores p son muy grandes, en este caso, el único valor p que brinca es el valor de la tercera gráfica, la cual representa los valores de la localidad de Eixample.

```
In [81]: from statsmodels.tsa.stattools import adfuller
import matplotlib.pyplot as plt
from scipy import stats
from numpy import asarray
import pandas as pd
import ssl

# rutina de https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/
def test_stationarity(ts, w, r, i):
    rolmean = ts.rolling(w).mean()
    rolstd = ts.rolling(w).std()
    plt.subplot(r, 1, i)
    orig = plt.plot(ts, color='blue', label='Original')
    mean = plt.plot(rolmean, color='red', label='Rolling Mean')
    std = plt.plot(rolstd, color='black', label = 'Rolling Std')
    plt.legend(loc = 'best')
    plt.title('Rolling Mean & Standard Deviation')
    dftest = adfuller(ts, autolag='AIC')
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])
    for key,value in dftest[4].items():
        dfoutput['Critical Value {:.3f}'.format(key)] = value
    return 'Results of Dickey-Fuller Test:\n' + '\n'.join(['{:s}\t{:.3f}'.format(k, v) for (k, v) in dfoutput.items()])

plt.rcParams["figure.figsize"] = [18, 12]
f = plt.figure()
lvls = ['a', 'b', 'c', 'd']
t = ''
n = len(d)
x = [i for i in range(n)]
i = 1
w = 52
for c in lvls:
    y = asarray(d[c])
```

```

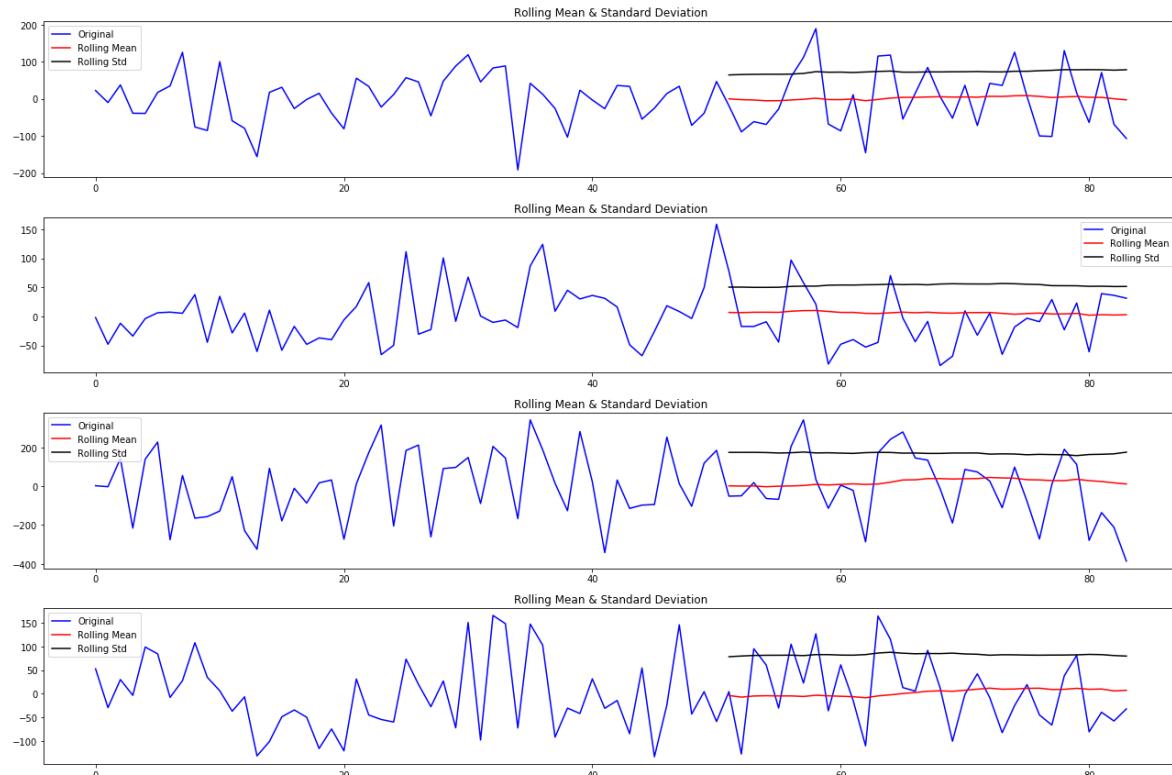
a, b, r, p, e = stats.linregress(x, y)
m = [a * v + b for v in x] # valores del modelo lineal
repl = c + 'v2'
d[repl] = y - m # los restamos
t += test_stationarity(d[repl], w, len(lvls), i)
i += 1
plt.tight_layout()
print(t)

```

```

Results of Dickey-Fuller Test:
Test Statistic -7.675
p-value 0.000
#Lags Used 1.000
Number of Observations Used 82.000
Critical Value (1%) -3.513
Critical Value (5%) -2.897
Critical Value (10%) -2.586Results of Dickey-Fuller Test:
Test Statistic -7.322
p-value 0.000
#Lags Used 0.000
Number of Observations Used 83.000
Critical Value (1%) -3.512
Critical Value (5%) -2.897
Critical Value (10%) -2.586Results of Dickey-Fuller Test:
Test Statistic -2.050
p-value 0.265
#Lags Used 7.000
Number of Observations Used 76.000
Critical Value (1%) -3.519
Critical Value (5%) -2.900
Critical Value (10%) -2.587Results of Dickey-Fuller Test:
Test Statistic -3.733
p-value 0.004
#Lags Used 2.000
Number of Observations Used 81.000
Critical Value (1%) -3.514
Critical Value (5%) -2.898
Critical Value (10%) -2.586

```



Después de las segundas pruebas, el valor p de la tercera gráfica sigue fijo, mientras que el valor p de la última gráfica, la localidad de

Les Corts, aumenta un poco. A pesar de eso, las líneas rojas, aunque un poco temblorosas, parecen estar en línea recta. De cualquier manera, se realiza una nueva experimentación, para quitar de dudas.

```
In [82]: from statsmodels.tsa.stattools import adfuller
import matplotlib.pyplot as plt
from pandas import Series
from numpy import asarray
import pandas as pd
import ssl

# de https://machinelearningmastery.com/time-series-forecast-study-python-monthly-sales-french-champagne/
def difference(dataset, interval=1):
    diff = list()
    for i in range(interval, len(dataset)):
        value = dataset[i] - dataset[i - interval]
        diff.append(value)
    return Series(diff)

plt.rcParams["figure.figsize"] = [8, 12]
f = plt.figure()
i = 1
for c in ['a', 'b', 'c', 'd']:
    ts = difference(asarray(d[c]), interval = 52)
    ts = ts.dropna()
    plt.subplot(4, 1, i)
    plt.title(c)
    plt.plot(ts)
    i += 1
    result = adfuller(ts)
    print('{:s}\nADF Stat\t{:3f}\np-value\t{:3f}\n'.format(c, result[0], result[1]))
    for key, value in result[4].items():
        print('\t{:s}: {:.3f}'.format(key, value))
plt.tight_layout()
```

a
ADF Stat -5.046
p-value 0.000

1%: -3.661
5%: -2.961
10%: -2.619

b
ADF Stat -4.644
p-value 0.000

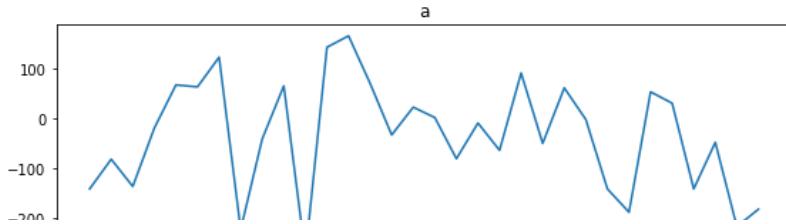
1%: -3.670
5%: -2.964
10%: -2.621

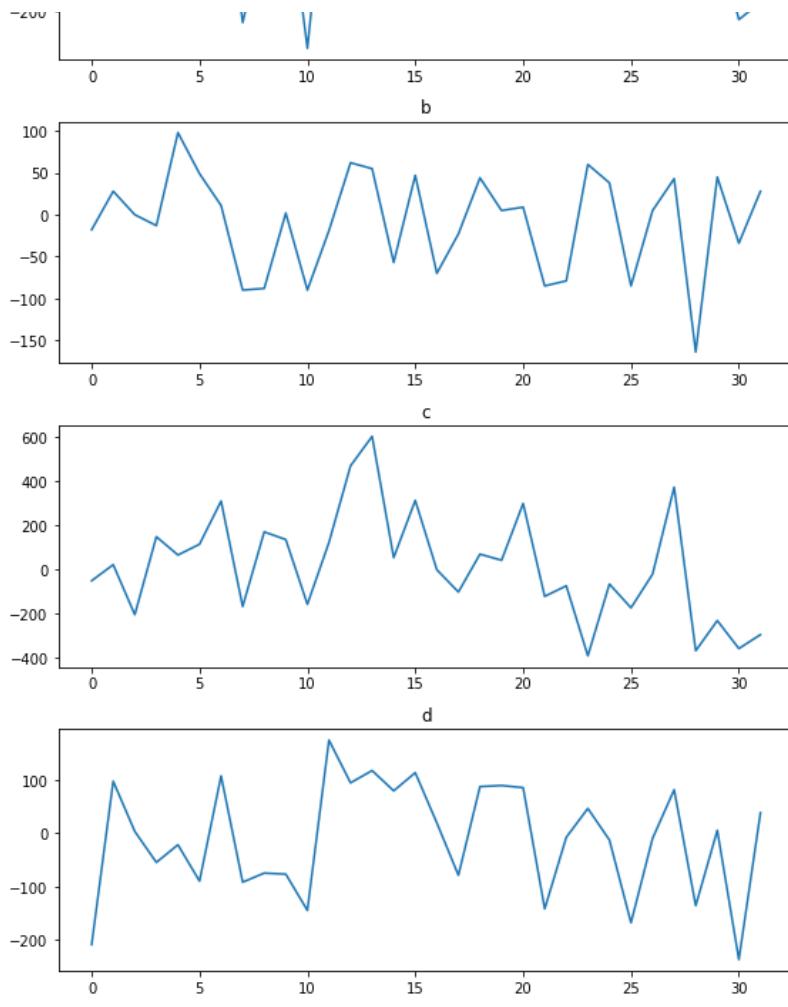
c
ADF Stat -1.188
p-value 0.679

1%: -3.770
5%: -3.005
10%: -2.643

d
ADF Stat -5.970
p-value 0.000

1%: -3.661
5%: -2.961
10%: -2.619





Después de esta prueba, se supone que el objetivo es que, a pesar de los picos, estos no hagan los valores a la izquierda de la gráfica ser tan separados, lo cual se logra en tres de las cuatro gráficas. La única localidad en la cual el valor p aumenta, y los valores del costado son más grandes es en la de Eixample, la cual es la que su valor p nunca disminuyó a lo largo de la práctica.

```
In [85]: from statsmodels.tsa.stattools import acf, pacf
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf
import matplotlib.pyplot as plt
from math import sqrt
import pandas as pd
import ssl

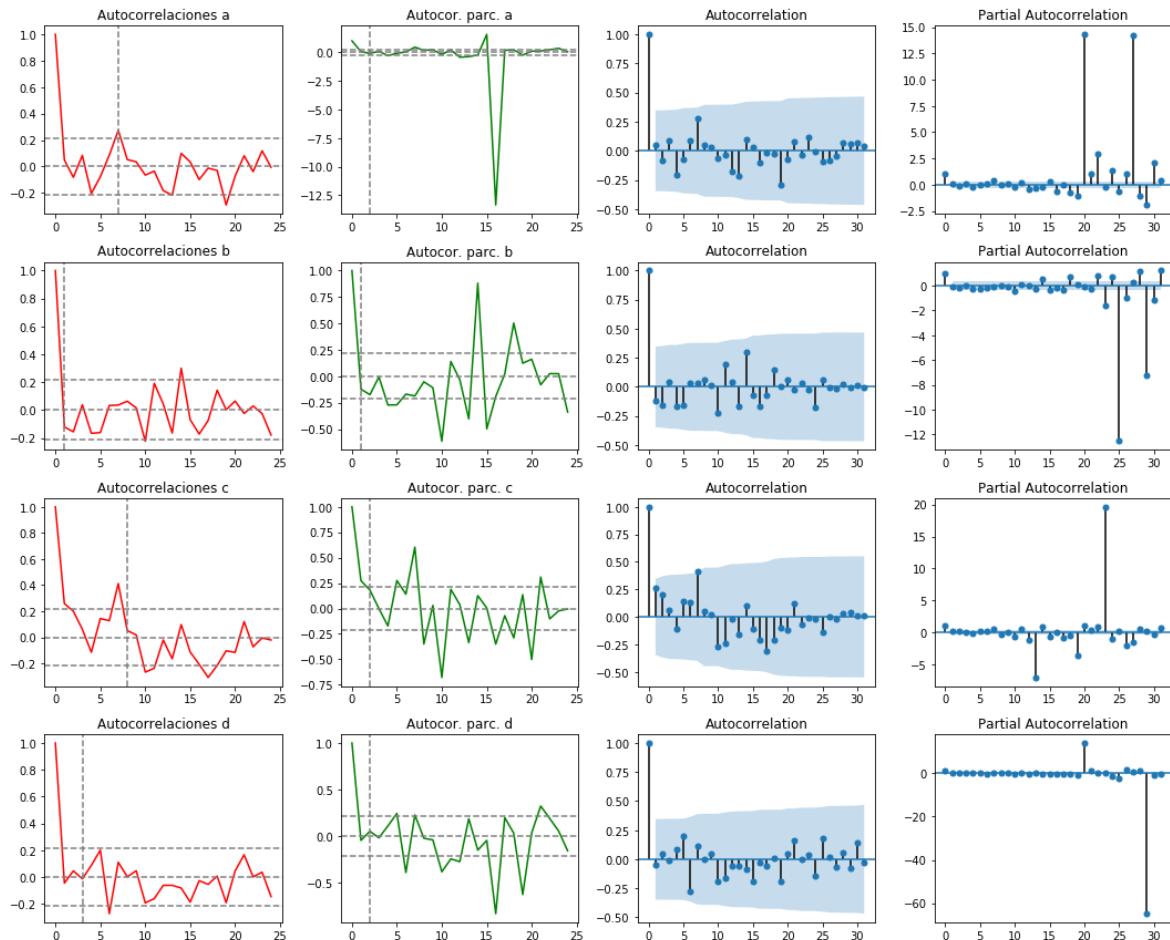
# de https://machinelearningmastery.com/time-series-forecast-study-python-monthly-sales-french-champagne/
def difference(dataset, interval=1):
    diff = list()
    for i in range(interval, len(dataset)):
        value = dataset[i] - dataset[i - interval]
        diff.append(value)
    return pd.Series(diff)

plt.rcParams["figure.figsize"] = [15, 12]
f = plt.figure()
i = 1
n = len(d)
q = {'a': 7, 'b': 1, 'c': 8, 'd': 3}
p = {'a': 2, 'b': 1, 'c': 2, 'd': 2}
for c in ['a', 'b', 'c', 'd']:
    ts = difference(asarray(d[c]), interval = 52).dropna()
    # manera del primer tutorial
```

```

plt.subplot(4, 4, i)
i += 1
plt.plot(acf(ts, nlags = 24), 'r')
plt.axhline(y = 0, linestyle='--', color = 'gray')
plt.axvline(x = q[c], linestyle = '--', color='gray')
plt.axhline(y = -1.96 / sqrt(n), linestyle = '--', color='gray')
plt.axhline(y = 1.96 / sqrt(n), linestyle = '--', color='gray')
plt.title('Autocorrelaciones ' + c)
plt.subplot(4, 4, i)
i += 1
plt.plot(pacf(ts, nlags = 24, method='ols'), 'g')
plt.axhline(y = 0, linestyle = '--', color = 'gray')
plt.axvline(x = p[c], linestyle = '--', color='gray')
plt.axhline(y = -1.96 / sqrt(n), linestyle = '--', color='gray')
plt.axhline(y = 1.96 / sqrt(n), linestyle = '--', color='gray')
plt.title('Autocor. parc. ' + c)
# manera del segundo tutorial
plt.subplot(4, 4, i)
i += 1
plot_acf(ts, ax = plt.gca())
plt.subplot(4, 4, i)
i += 1
plot_pacf(ts, ax = plt.gca())
plt.tight_layout()

```



Entonces, siguiendo el procedimiento del primer tutorial, buscando la coordenada x del primer cruce con el intervalo de confianza superior, se obtiene $q = \{\text{'San Martí': 6, 'Ciutat Vella': 1, 'Eixample': 7, 'Les Corts': 2}\}$ $p = \{\text{'San Martí': 2, 'Ciutat Vella': 1, 'Eixample': 2, 'Les Corts': 2}\}$

```
In [9]: from statsmodels.tsa.arima_model import ARIMA
import pandas as pd
import ssl
```

```

# https://machinelearningmastery.com/time-series-forecast-study-python-monthly-sales-french-champagne/
def difference(dataset, interval=1):
    diff = list()
    for i in range(interval, len(dataset)):
        value = dataset[i] - dataset[i - interval]
        diff.append(value)
    return pd.Series(diff)

guardar = d.iloc[-1:, :]
d = d.iloc[:-1, :]
n = len(d)
q = {'a': 6, 'b': 1, 'c': 7, 'd': 3}
p = {'a': 2, 'b': 1, 'c': 2, 'd': 2}
prons = []
for c in ['a', 'b', 'c', 'd']:
    ts = difference(asarray(d[c]), interval = 7).dropna()
    try:
        m = ARIMA(ts, order = (p[c], 1, q[c]))
        f = m.fit(trend = 'nc', disp = 0)
        print('{:s}\tpron.\t'.format(c), int(round(f.forecast()[0][0] + d[c].iloc[n - 52])))
        print('{:s}\treal\t'.format(c), guardar[c].iloc[0])
    except:
        print(c, "no se pudo pronosticar con esos parámetros")
        print('...')

a      pron.    328
a      real     324
...
b      pron.     70
b      real     74
...
c no se pudo pronosticar con esos parámetros
...
d      pron.    193
d      real     195
...

```

Supongo que en el c, que es la localidad donde el valor p siempre fue alto, el pronóstico no pudo hacerse por esa misma razón. En los otros tres casos, el pronóstico y su realidad no están tan separados. El intervalo se cambió a 7 porque son los 7 días de la semana en las que están separados los datos.

```

In [11]: from statsmodels.tsa.arima_model import ARIMA
import pandas as pd
import ssl

# https://machinelearningmastery.com/time-series-forecast-study-python-monthly-sales-french-champagne/
def difference(dataset, interval=1):
    diff = list()
    for i in range(interval, len(dataset)):
        value = dataset[i] - dataset[i - interval]
        diff.append(value)
    return pd.Series(diff)

guardar = d.iloc[-1:, :]
d = d.iloc[:-1, :]
n = len(d)
q = {'a': 1, 'b': 1, 'c': 2, 'd': 1}
p = {'a': 2, 'b': 1, 'c': 2, 'd': 0}
prons = []
for c in ['a', 'b', 'c', 'd']:
    ts = difference(asarray(d[c]), interval = 7).dropna()
    try:
        m = ARIMA(ts, order = (p[c], 1, q[c]))
        f = m.fit(trend = 'nc', disp = 0)
        print('{:s}\tpron.\t'.format(c), int(round(f.forecast()[0][0] + d[c].iloc[n - 52])))
        print('{:s}\treal\t'.format(c), guardar[c].iloc[0])
    except:
        print(c, "no se pudo pronosticar con esos parámetros")
        print('...')

```

a	pron.	231
a	real	95
...		
b	pron.	54
b	real	88
...		
c	pron.	272
c	real	231
...		
d	pron.	78
d	real	113
...		

Se dejó el mismo intervalo, pero se cambió los datos de entrada por localidad a los mismos que se tenían en la práctica de la Dra Elisa. Este último experimento se hizo por curiosidad, ya que no entendía porque los datos de la q y la p que describió, no eran los mismos que en el código.

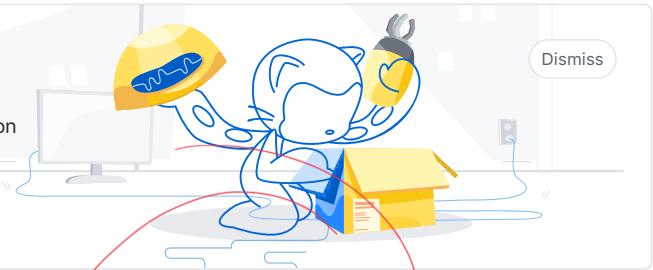
En este resultado el pronóstico del c si fue posible, pero es el único que se acerca, y los demás los deja peor, por lo que creo que sería interesante hacer estas pruebas con todos los demás datos de localidades, y agrupar aquellas que se comporten como el que está más desfasado, para así poder tener predicciones más concretas.

All your code in one place

GitHub makes it easy to scale back on context switching. Read rendered documentation, see the history of any file, and collaborate with contributors on projects across GitHub.

[Sign up for free](#)

[See pricing for teams and enterprises](#)



Dismiss

Branch: master ▾ [Ciencia_de_Datos / Práctica 10.ipynb](#)

[Find file](#) [Copy path](#)

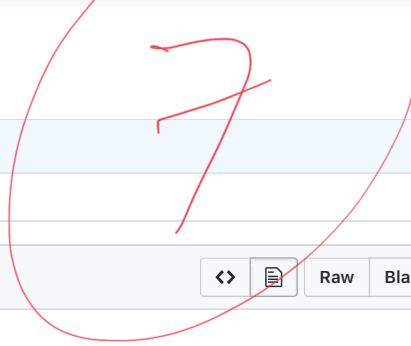
mayaberrones94 ultimo

d015f25 9 hours ago

1 contributor

905 lines (904 sloc) | 984 KB

[Raw](#) [Blame](#) [History](#) [Edit](#) [Delete](#)



Práctica 10: Clasificación de datos con sklearn.

El término clasificación hace referencia a la asignación automática de datos en grupos previamente conocidos de acuerdo a sus atributos. En el caso estudiado para los datos que se presentan en esta práctica, los dos grupos de interés son el conjunto benigno y maligno de datos, de donde se han explorado diferentes métodos para separar sus características de manera que permita hacer una clasificación con exactitud buena.

```
In [8]: import ssl
import pandas as pd
from math import ceil, sqrt
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from matplotlib.colors import ListedColormap
from numpy import isnan, nan, arange, meshgrid, c_
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
import numpy as np
from sklearn.datasets import make_moons, make_circles, make_classification
from sklearn.neural_network import MLPClassifier
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
```

El primer paso es cargar los datos del CSV. Se sigue con el procedimiento utilizado en prácticas anteriores y se reescriben los datos de M y B con 0 y 1 como las etiquetas deseadas de salida.

Después, para poder tener una idea de cómo se separan los datos del clasificador, se toman los máximos de los valores de radius_mean y radius_worst en los valores de diagnóstico benignos, con la idea de que cualquier elemento que se encuentre por encima de esos valores, tiene que ser maligno.

Esto surge con la idea que en prácticas anteriores, utilizando los valores de parámetros con menos relación se podía lograr una clasificación más exacta. Una de las conclusiones a las que se llegó anteriormente fue que el 'cluster' que se notaba en gráficas de clasificaciones podría deshacerse si se utilizan las características de cada columna del csv.

En este caso se tiene la idea de que el máximo de los benignos y el mínimo de los malignos son valores en donde se encuentra el 'cluster' mencionado, por lo que en esta práctica se buscará deshacer esto utilizando los máximos y mínimos de varias columnas. El 'radius_mean' y el 'radius_worst' son los dos datos en los que se puede hacer un umbral similar.

```
In [9]: if getattr(ssl, '_create_unverified_context', None):
    ssl._create_default_https_context = ssl._create_unverified_context
d = pd.read_csv("https://raw.githubusercontent.com/mayraberrones94/Ciencia_de_Datos/master/bp_w-d
ta.csv")
d.diagnosis.replace(to_replace = dict(M = 1, B = 0), inplace = True)

m = d.loc[d.diagnosis == 1]
b = d.loc[d.diagnosis == 0]
print(m.radius_mean.max(), m.radius_worst.max())
print(m.radius_mean.min(), m.radius_worst.min())
print(b.radius_mean.max(), b.radius_worst.max())
print(b.radius_mean.min(), b.radius_worst.min())

28.11 36.04
10.95 12.84
17.85 19.82
6.981 7.93
```

Una vez que se tienen los dos valores de máximo y mínimo, se colocan en los valores de las etiquetas con las que se va a realizar el clasificador. En el primer intento de corrida, el programa marca error diciendo que faltaba declarar el valor de la 'h' utilizada en la línea 25. Despues de revisar el código en el link proporcionado en la práctica de ejemplo, se ve que el valor de la h se utiliza como parámetro de tamaño del paso que se va a dar en la parte de entrenamiento y prueba. Se toma el mismo valor ahí mencionado, 0.02.

```
In [10]: pri = d.radius_mean >= 17.85
seg = d.radius_worst >= 19.82
ter = ~pri & ~seg

d['etiquetas'] = [1 if pri[i] else (2 if seg[i] else (0 if ter[i] else "NA")) for i in pri.keys()]
# etiquetas
print(d.etiquetas.value_counts())
y = d.etiquetas
xVars = ['radius_mean', 'radius_worst']
x = d.loc[:, xVars].values
x = StandardScaler().fit_transform(x)
pca = PCA(n_components = 2) # pedimos uno bidimensional
X = pca.fit_transform(x)
# código de https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_compariso
n.html
h = .02 # step size in the mesh
names = ["Nearest Neighbors", "Linear SVM", "RBF SVM", "Gaussian Process", \
         "Decision Tree", "Random Forest", "AdaBoost", "Naive Bayes"]
classifiers = [KNeighborsClassifier(3), SVC(kernel="linear", C=0.025), \
               SVC(gamma=2, C=1), GaussianProcessClassifier(1.0 * RBF(1.0)), \
               DecisionTreeClassifier(max_depth=5), RandomForestClassifier(max_depth=5, n_estimators=10, max_
features=1), \
               AdaBoostClassifier(), GaussianNB()]
k = int(ceil(sqrt(len(classifiers) + 1)))
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.3, random_state=42) # divisió
n
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
xx, yy = meshgrid(arange(x_min, x_max, h), arange(y_min, y_max, 0.02))
cm = plt.cm.RdBu
cm_bright = ListedColormap(['#FF0000', '#0000FF'])
plt.rcParams["figure.figsize"] = [16, 16]
figure = plt.figure()
ax = plt.subplot(k, k, 1)
ax.set_title("Datos de entrada")
ax.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cm_bright, alpha=0.2, edgecolors='k') # e
ntrenamiento
ax.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=cm_bright, alpha=0.2, edgecolors='k') # vali
dación
```

```

ax.set_xlim(xx.min(), xx.max())
ax.set_ylim(yy.min(), yy.max())
ax.set_xticks(())
ax.set_yticks(())
i = 2
for name, clf in zip(names, classifiers):
    ax = plt.subplot(k, k, i)
    clf.fit(X_train, y_train)
    score = clf.score(X_test, y_test)
    if hasattr(clf, "decision_function"):
        Z = clf.decision_function(c_[xx.ravel(), yy.ravel()])[:, 1]
    else:
        Z = clf.predict_proba(c_[xx.ravel(), yy.ravel()])[:, 1]
    Z = Z.reshape(xx.shape)
    ax.contourf(xx, yy, Z, cmap=cm, alpha=.8)
    ax.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cm_bright, edgecolors='k')
    ax.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=cm_bright, edgecolors='k', alpha=0.6)
    ax.set_xlim(xx.min(), xx.max())
    ax.set_ylim(yy.min(), yy.max())
    ax.set_xticks(())
    ax.set_yticks(())
    ax.set_title(name)
    ax.text(xx.max() - .3, yy.min() + .3, ('%.3f' % score).lstrip('0'), size=40, horizontalalignment='right')
    i += 1
plt.tight_layout()
plt.show()

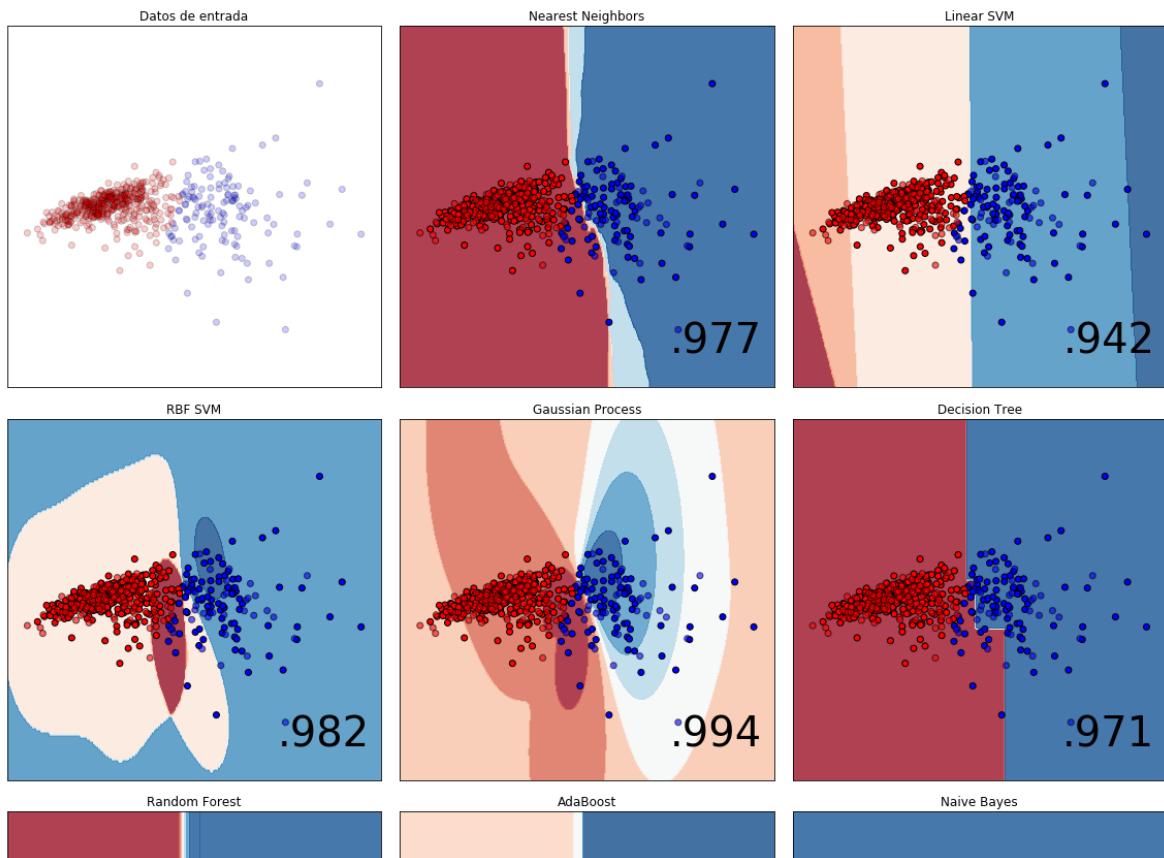
```

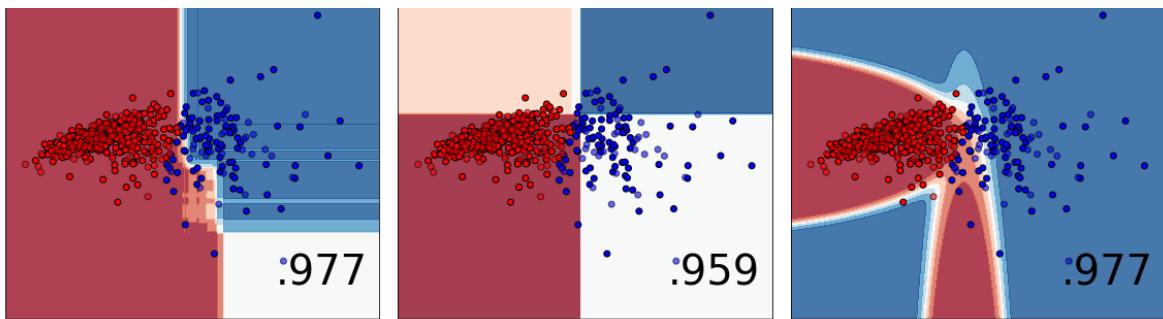
0 441
1 98
2 30
Name: etiquetas, dtype: int64

```

/Users/mayraberrones/anaconda3/lib/python3.7/site-packages/sklearn/gaussian_process/gpc.py:434: ConvergenceWarning: fmin_l_bfgs_b terminated abnormally with the state: {'grad': array([-0.13121889, 1.95281997]), 'task': b'ABNORMAL_TERMINATION_IN_LNSRCH', 'funcalls': 77, 'nit': 6, 'warnflag': 2}
ConvergenceWarning)

```





Los datos representados en el 0, son los que estarían dentro del 'cluster' difícil de clasificar. Los datos que están en el 1 y el 2 son los datos que no se tiene duda de su clasificación, ya que están dentro de los rangos de máximos y mínimos que se establecieron anteriormente.

Como resultado en las gráficas se tiene una buena separación de los datos y una alta exactitud de clasificación en todos los clasificadores. El siguiente paso es saber la precisión, el "recall" y el "F1-score", que son las mediciones utilizadas para reportar el rendimiento de una herramienta de aprendizaje de máquina.

```
In [12]: import ssl
import pandas as pd
from math import ceil, sqrt
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from matplotlib.colors import ListedColormap
from numpy import isnan, nan
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
names = ["Nearest Neighbors", "Linear SVM", "RBF SVM", "Gaussian Process", \
         "Decision Tree", "Random Forest", "AdaBoost", "Naive Bayes"]
classifiers = [KNeighborsClassifier(3), SVC(kernel="linear", C=0.025), \
               SVC(gamma=2, C=1), GaussianProcessClassifier(1.0 * RBF(1.0)), \
               DecisionTreeClassifier(max_depth=5), RandomForestClassifier(max_depth=5, n_estimators=10, max_\
               features=1), \
               AdaBoostClassifier(), GaussianNB()]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.3, random_state=42) # la mism\
a división
for name, clf in zip(names, classifiers):
    clf.fit(X_train, y_train)
    print(name, clf.score(X_test, y_test))
    expected, predicted = y_test, clf.predict(X_test)
    print(metrics.classification_report(expected, predicted))
    print(metrics.confusion_matrix(expected, predicted))
    print('-' * 60)

Nearest Neighbors 0.9766081871345029
      precision    recall  f1-score   support
          0       0.99     0.99     0.99      133
          1       0.93     1.00     0.97      28
          2       0.88     0.70     0.78      10

      micro avg       0.98     0.98     0.98      171
      macro avg       0.93     0.90     0.91      171
  weighted avg       0.98     0.98     0.98      171

[[132    0    1]
 [  0   28    0]
 [  1    2   7]]]

-----
Linear SVM 0.9415204678362573
```

```

          precision    recall   f1-score   support
          0         0.96     1.00     0.98     133
          1         0.85     1.00     0.92      28
          2         0.00     0.00     0.00      10

      micro avg       0.94     0.94     0.94     171
      macro avg       0.60     0.67     0.63     171
  weighted avg       0.89     0.94     0.91     171

[[133  0  0]
 [ 0  28  0]
 [ 5  5  0]]
```

RBF SVM 0.9824561403508771

	precision	recall	f1-score	support
0	1.00	0.99	1.00	133
1	0.93	1.00	0.97	28
2	0.89	0.80	0.84	10
micro avg	0.98	0.98	0.98	171
macro avg	0.94	0.93	0.93	171
weighted avg	0.98	0.98	0.98	171

```

[[132  0  1]
 [ 0  28  0]
 [ 0  2  8]]
```

```

/Users/mayraberrones/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classification.py:1143:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with
no predicted samples.
    'precision', 'predicted', average, warn_for)
/Users/mayraberrones/anaconda3/lib/python3.7/site-packages/sklearn/gaussian_process/gpc.py:434:
ConvergenceWarning: fmin_l_bfgs_b terminated abnormally with the state: {'grad': array([-0.13121889
,  1.95281997]), 'task': b'ABNORMAL_TERMINATION_IN_LNSRCH', 'funcalls': 77, 'nit': 6, 'warnflag':
2}
    ConvergenceWarning)
```

Gaussian Process 0.9941520467836257

	precision	recall	f1-score	support
0	1.00	0.99	1.00	133
1	1.00	1.00	1.00	28
2	0.91	1.00	0.95	10
micro avg	0.99	0.99	0.99	171
macro avg	0.97	1.00	0.98	171
weighted avg	0.99	0.99	0.99	171

```

[[132  0  1]
 [ 0  28  0]
 [ 0  0  10]]
```

Decision Tree 0.9707602339181286

	precision	recall	f1-score	support
0	1.00	0.98	0.99	133
1	0.90	1.00	0.95	28
2	0.78	0.70	0.74	10
micro avg	0.97	0.97	0.97	171
macro avg	0.89	0.89	0.89	171
weighted avg	0.97	0.97	0.97	171

```

[[131  0  2]
 [ 0  28  0]
 [ 0  3  7]]
```

Random Forest 0.9766081871345029

	precision	recall	f1-score	support
0	0.99	1.00	1.00	133

```

      1      0.90    1.00    0.95    28
      2      1.00    0.60    0.75    10

  micro avg      0.98    0.98    0.98    171
  macro avg      0.97    0.87    0.90    171
weighted avg     0.98    0.98    0.97    171

[[133  0  0]
 [ 0  28  0]
 [ 1  3  6]]
-----
AdaBoost 0.9590643274853801
      precision    recall   f1-score   support
      0       0.99    0.98    0.99    133
      1       0.88    1.00    0.93    28
      2       0.71    0.50    0.59    10

  micro avg      0.96    0.96    0.96    171
  macro avg      0.86    0.83    0.84    171
weighted avg     0.96    0.96    0.96    171

[[131  0  2]
 [ 0  28  0]
 [ 1  4  5]]
-----
Naive Bayes 0.9766081871345029
      precision    recall   f1-score   support
      0       0.99    0.99    0.99    133
      1       0.90    1.00    0.95    28
      2       1.00    0.70    0.82    10

  micro avg      0.98    0.98    0.98    171
  macro avg      0.97    0.90    0.92    171
weighted avg     0.98    0.98    0.98    171

[[132  1  0]
 [ 0  28  0]
 [ 1  2  7]]
-----
```

Ahora que ya se tiene un buen resultado, se intenta poner más datos de los que se tomaron inicialmente. En la práctica de prueba de la Dra. Elisa, esto es con la intención de que salgan más alumnos que reprobaron en primera oportunidad. En el caso de esta práctica, la cantidad de casos malignos es más pequeña que la de los benignos por casi la mitad, por lo que un conjunto de datos de prueba más grande permitirá que tenga más posibilidad de tener datos balanceados. Se repite el procedimiento con solo un dato, que es el “radius_mean” y se aumenta el valor del conjunto de prueba de 0.3 a 0.4. Los resultados tanto en las gráficas como en la parte de tablas de confusión mejoran.

En este caso, los datos del 0 siguen siendo los de la parte del cluster, y la del 1 de los resultados benignos. No tener el resultado del 2 no es problema, ya que se asume que son los resultados sobrantes, o sea los de la parte que son seguramente malignos.

```
In [16]: import ssl
import pandas as pd
from math import ceil, sqrt
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from matplotlib.colors import ListedColormap
from numpy import isnan, nan, arange, meshgrid, c_
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB

if getattr(ssl, '_create_unverified_context', None):
    ssl._create_default_https_context = ssl._create_unverified_context
```

```

d = pd.read_csv("https://raw.githubusercontent.com/mayraberrones94/Ciencia_de_Datos/master/bp_w-data.csv")
d.diagnosis.replace(to_replace = dict(M = 1, B = 0), inplace = True)

pri = d.radius_mean >= 17.85

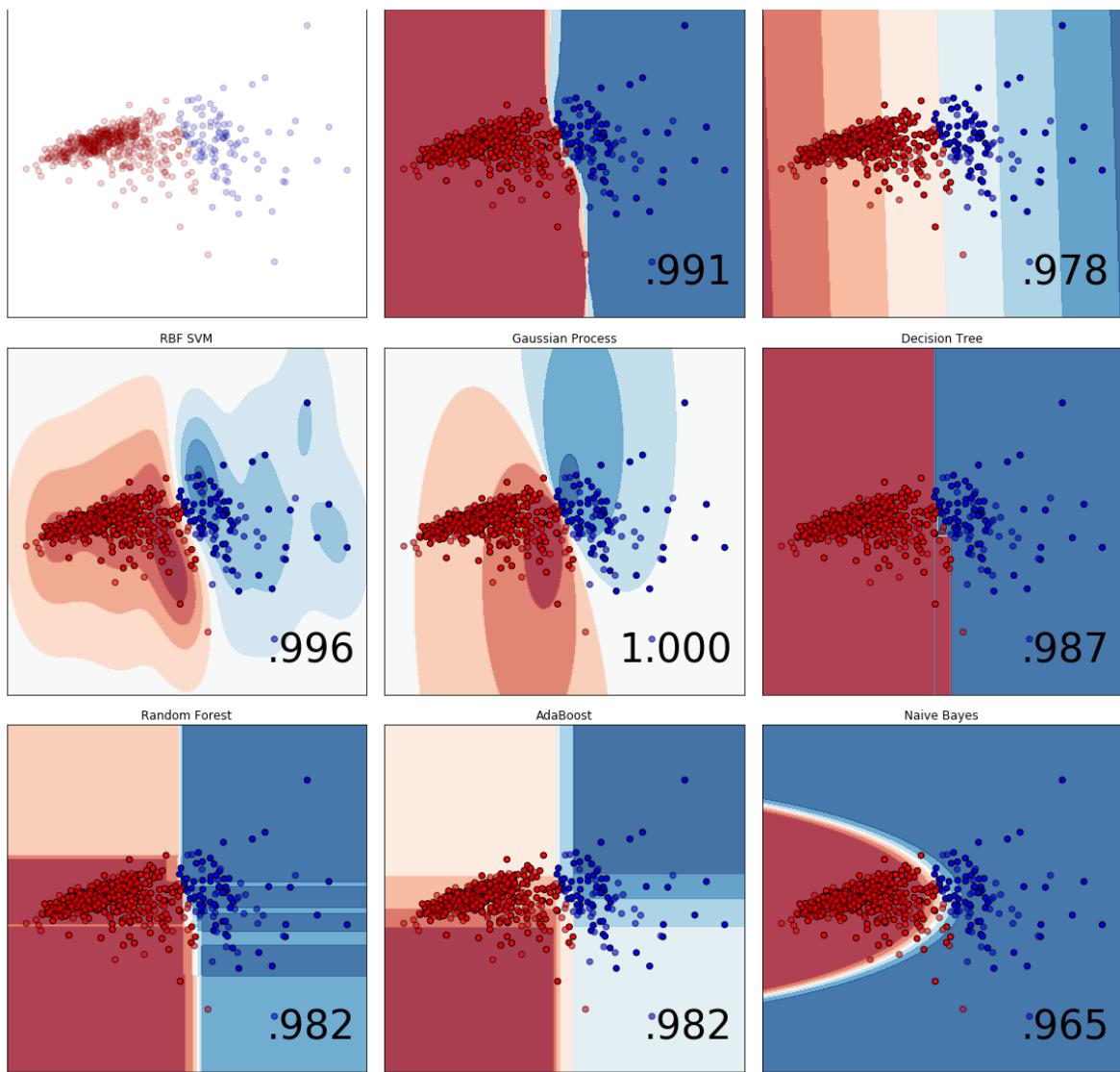
d['etiquetas'] = [1 if pri[i] else 0 for i in pri.keys()] # etiquetas
print(d.etiquetas.value_counts())
y = d.etiquetas
xVars = ['radius_mean', 'radius_worst']
x = d.loc[:, xVars].values
x = StandardScaler().fit_transform(x)
pca = PCA(n_components = 2) # pedimos uno bidimensional
X = pca.fit_transform(x)

h = .02 # step size in the mesh
names = ["Nearest Neighbors", "Linear SVM", "RBF SVM", "Gaussian Process", \
         "Decision Tree", "Random Forest", "AdaBoost", "Naive Bayes"]
classifiers = [KNeighborsClassifier(3), SVC(kernel="linear", C=0.025), \
               SVC(gamma=2, C=1), GaussianProcessClassifier(1.0 * RBF(1.0)), \
               DecisionTreeClassifier(max_depth=5), RandomForestClassifier(max_depth=5, n_estimators=10, max_\
               features=1), \
               AdaBoostClassifier(), GaussianNB()]
k = int(ceil(sqrt(len(classifiers) + 1)))
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.4, random_state=42) # división
n
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
xx, yy = meshgrid(arange(x_min, x_max, h), arange(y_min, y_max, 0.02))
cm = plt.cm.RdBu
cm_bright = ListedColormap([('FF0000', '#0000FF')])
plt.rcParams["figure.figsize"] = [16, 16]
figure() = plt.figure()
ax = plt.subplot(k, k, 1)
ax.set_title("Datos de entrada")
ax.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cm_bright, alpha=0.2, edgecolors='k') # en entrenamiento
ax.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=cm_bright, alpha=0.2, edgecolors='k') # validación
ax.set_xlim(xx.min(), xx.max())
ax.set_ylim(yy.min(), yy.max())
ax.set_xticks(())
ax.set_yticks(())
i = 2
for name, clf in zip(names, classifiers):
    ax = plt.subplot(k, k, i)
    clf.fit(X_train, y_train)
    score = clf.score(X_test, y_test)
    if hasattr(clf, "decision_function"):
        Z = clf.decision_function(c_[xx.ravel(), yy.ravel()])
    else:
        Z = clf.predict_proba(c_[xx.ravel(), yy.ravel()])[:, 1]
    Z = Z.reshape(xx.shape)
    ax.contourf(xx, yy, Z, cmap=cm, alpha=.8)
    ax.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cm_bright, edgecolors='k')
    ax.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=cm_bright, edgecolors='k', alpha=0.6)
    ax.set_xlim(xx.min(), xx.max())
    ax.set_ylim(yy.min(), yy.max())
    ax.set_xticks(())
    ax.set_yticks(())
    ax.set_title(name)
    ax.text(xx.max() - .3, yy.min() + .3, ('%.3f' % score).lstrip('0'), size=40, horizontalalignme
nt='right')
    i += 1
plt.tight_layout()
plt.show()

```

0 471
1 98
Name: etiquetas, dtype: int64





```
In [17]: import ssl
import pandas as pd
from math import ceil, sqrt
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from matplotlib.colors import ListedColormap
from numpy import isnan, nan
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_moons, make_circles, make_classification
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis

if getattr(ssl, '_create_unverified_context', None):
    ssl._create_default_https_context = ssl._create_unverified_context
d = pd.read_csv("https://raw.githubusercontent.com/mayraberrones94/Ciencia_de_Datos/master/bp_w-dato.csv")
d.diagnosis.replace({0: 1, 1: 0}, inplace = True)
```

```

d.replace(to_replace = dict(m = 1, b = 0), inplace = True)

pri = d.radius_mean >= 17.85

d['etiquetas'] = [1 if pri[i] else 0 for i in pri.keys()] # etiquetas
print(d.etiquetas.value_counts())
y = d.etiquetas
xVars = ['radius_mean', 'radius_worst']
x = d.loc[:, xVars].values
x = StandardScaler().fit_transform(x)
pca = PCA(n_components = 2) # pedimos uno bidimensional
X = pca.fit_transform(x)
# código de https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_compariso
n.html
names = ["Nearest Neighbors", "Linear SVM", "RBF SVM", "Gaussian Process", \
         "Decision Tree", "Random Forest", "AdaBoost", "Naive Bayes"]
classifiers = [KNeighborsClassifier(3), SVC(kernel="linear", C=0.025), \
               SVC(gamma=2, C=1), GaussianProcessClassifier(1.0 * RBF(1.0)), \
               DecisionTreeClassifier(max_depth=5), RandomForestClassifier(max_depth=5, n_estimators=10, max_
features=1), \
               AdaBoostClassifier(), GaussianNB()]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.4, random_state=42) # la mism
a división
for name, clf in zip(names, classifiers):
    clf.fit(X_train, y_train)
    print(name, clf.score(X_test, y_test))
    expected, predicted = y_test, clf.predict(X_test)
    print(metrics.classification_report(expected, predicted))
    print(metrics.confusion_matrix(expected, predicted))
    print('-' * 60)

0    471
1     98
Name: etiquetas, dtype: int64
Nearest Neighbors 0.9912280701754386
      precision    recall  f1-score   support
          0       1.00     0.99     0.99      193
          1       0.95     1.00     0.97      35

   micro avg       0.99     0.99     0.99      228
   macro avg       0.97     0.99     0.98      228
weighted avg       0.99     0.99     0.99      228

[[191   2]
 [ 0  35]]
-----
Linear SVM 0.9780701754385965
      precision    recall  f1-score   support
          0       1.00     0.97     0.99      193
          1       0.88     1.00     0.93      35

   micro avg       0.98     0.98     0.98      228
   macro avg       0.94     0.99     0.96      228
weighted avg       0.98     0.98     0.98      228

[[188   5]
 [ 0  35]]
-----
RBF SVM 0.9956140350877193
      precision    recall  f1-score   support
          0       1.00     0.99     1.00      193
          1       0.97     1.00     0.99      35

   micro avg       1.00     1.00     1.00      228
   macro avg       0.99     1.00     0.99      228
weighted avg       1.00     1.00     1.00      228

[[192   1]
 [ 0  35]]
-----
Gaussian Process 1.0

```

```

Gaussian Process 1.0
      precision    recall  f1-score   support

          0       1.00     1.00     1.00      193
          1       1.00     1.00     1.00       35

   micro avg       1.00     1.00     1.00      228
   macro avg       1.00     1.00     1.00      228
weighted avg       1.00     1.00     1.00      228

[[193  0]
 [ 0  35]]
```

```

Decision Tree 0.9868421052631579
      precision    recall  f1-score   support

          0       1.00     0.98     0.99      193
          1       0.92     1.00     0.96       35

   micro avg       0.99     0.99     0.99      228
   macro avg       0.96     0.99     0.98      228
weighted avg       0.99     0.99     0.99      228

[[190  3]
 [ 0  35]]
```

```

Random Forest 0.9868421052631579
      precision    recall  f1-score   support

          0       1.00     0.98     0.99      193
          1       0.92     1.00     0.96       35

   micro avg       0.99     0.99     0.99      228
   macro avg       0.96     0.99     0.98      228
weighted avg       0.99     0.99     0.99      228

[[190  3]
 [ 0  35]]
```

```

AdaBoost 0.9824561403508771
      precision    recall  f1-score   support

          0       0.99     0.98     0.99      193
          1       0.92     0.97     0.94       35

   micro avg       0.98     0.98     0.98      228
   macro avg       0.96     0.98     0.97      228
weighted avg       0.98     0.98     0.98      228

[[190  3]
 [ 1  34]]
```

```

Naive Bayes 0.9649122807017544
      precision    recall  f1-score   support

          0       1.00     0.96     0.98      193
          1       0.81     1.00     0.90       35

   micro avg       0.96     0.96     0.96      228
   macro avg       0.91     0.98     0.94      228
weighted avg       0.97     0.96     0.97      228

[[185  8]
 [ 0  35]]
```

Por último se realiza el experimento en el cual se utilizan muchas más columnas del csv que solo las dos de 'radius_mean' y 'radius_worst'. En este caso ya no se utilizan los valores de umbral máximo y mínimo porque los datos son demasiado diferentes entre si para poder darle solo dos valores en los rangos máximos y mínimos (unos son enteros, otros son flotantes).

```
In [18]: import ssl
import pandas as pd
from math import ceil, sqrt
```

```

from math import ceil, sqrt
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from matplotlib.colors import ListedColormap
from numpy import isnan, nan
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB

if getattr(ssl, '_create_unverified_context', None):
    ssl._create_default_https_context = ssl._create_unverified_context
d = pd.read_csv("https://raw.githubusercontent.com/mayraberrones94/Ciencia_de_Datos/master/bp_w-data.csv")
d.diagnosis.replace(to_replace = dict(M = 1, B = 0), inplace = True)

xv = ['diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean', 'compactness_mean',
      'concavity_mean', 'radius_worst', 'texture_worst', 'perimeter_worst',
      'area_worst', 'compactness_worst']
d = d.loc[:, xv]
for c in ['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean', 'compactness_mean',
          'concavity_mean', 'radius_worst', 'texture_worst', 'perimeter_worst',
          'area_worst', 'compactness_worst']:
    cat = pd.Categorical(d[c])
    d[c] = cat.codes
d = d.dropna()
d['etiqueta'] = d.diagnosis == 1
y = d.etiqueta
print(y.value_counts())
d = d.drop(['etiqueta'], 1)
X = StandardScaler().fit_transform(d.values) # ahora SIN ejecutar PCA antes
names = ["Nearest Neighbors", "Linear SVM", "RBF SVM", "Gaussian Process",
         "Decision Tree", "Random Forest", "AdaBoost", "Naive Bayes"]
classifiers = [KNeighborsClassifier(3), SVC(kernel="linear", C=0.025),
               SVC(gamma=2, C=1), GaussianProcessClassifier(1.0 * RBF(1.0)),
               DecisionTreeClassifier(max_depth=5), RandomForestClassifier(max_depth=5, n_estimators=10, max_features=1),
               AdaBoostClassifier(), GaussianNB()]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.5) # cada ejecución tiene una división al azar
for name, clf in zip(names, classifiers):
    clf.fit(X_train, y_train)
    y_m = clf.predict(X_test)
    print(name, metrics.classification_report(y_test, y_m))
    print(metrics.confusion_matrix(y_test, y_m))
    print('-' * 60)

False      357
True       212
Name: etiqueta, dtype: int64
Nearest Neighbors          precision     recall   f1-score   support
False        1.00      1.00      1.00      174
True         1.00      1.00      1.00      111

micro avg     1.00      1.00      1.00      285
macro avg     1.00      1.00      1.00      285
weighted avg   1.00      1.00      1.00      285

[[174   0]
 [  0 111]]
-----
Linear SVM          precision     recall   f1-score   support
False        1.00      1.00      1.00      174
True         1.00      1.00      1.00      111

```

```

true    1.00    1.00    1.00    111
micro avg 1.00    1.00    1.00    285
macro avg 1.00    1.00    1.00    285
weighted avg 1.00    1.00    1.00    285

[[174  0]
 [ 0 111]]
```

RBF SVM		precision	recall	f1-score	support
False	0.84	1.00	0.91	174	
True	1.00	0.69	0.82	111	
micro avg	0.88	0.88	0.88	285	
macro avg	0.92	0.85	0.87	285	
weighted avg	0.90	0.88	0.88	285	

```

[[174  0]
 [ 34 77]]
```

```

/Users/mayraberrones/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.py:595: DataCo
nversionWarning: Data with input dtype int64 was converted to float64 by StandardScaler.
    warnings.warn(msg, DataConversionWarning)
/Users/mayraberrones/anaconda3/lib/python3.7/site-packages/sklearn/utils/validation.py:595: DataCo
nversionWarning: Data with input dtype int64 was converted to float64 by StandardScaler.
    warnings.warn(msg, DataConversionWarning)
```

Gaussian Process		precision	recall	f1-score	support
False	1.00	1.00	1.00	174	
True	1.00	1.00	1.00	111	
micro avg	1.00	1.00	1.00	285	
macro avg	1.00	1.00	1.00	285	
weighted avg	1.00	1.00	1.00	285	

```

[[174  0]
 [ 0 111]]
```

Decision Tree		precision	recall	f1-score	support
False	1.00	1.00	1.00	174	
True	1.00	1.00	1.00	111	
micro avg	1.00	1.00	1.00	285	
macro avg	1.00	1.00	1.00	285	
weighted avg	1.00	1.00	1.00	285	

```

[[174  0]
 [ 0 111]]
```

Random Forest		precision	recall	f1-score	support
False	0.97	0.97	0.97	174	
True	0.95	0.95	0.95	111	
micro avg	0.96	0.96	0.96	285	
macro avg	0.96	0.96	0.96	285	
weighted avg	0.96	0.96	0.96	285	

```

[[169  5]
 [ 5 106]]
```

AdaBoost		precision	recall	f1-score	support
False	1.00	1.00	1.00	174	
True	1.00	1.00	1.00	111	
micro avg	1.00	1.00	1.00	285	
macro avg	1.00	1.00	1.00	285	
weighted avg	1.00	1.00	1.00	285	

```
111 111
```

```
[[1 1 / 4      0 ]  
 [ 0 1 1 1 ]]
```

Naive Bayes	precision	recall	f1-score	support
False	1.00	1.00	1.00	174
True	1.00	1.00	1.00	111
micro avg	1.00	1.00	1.00	285
macro avg	1.00	1.00	1.00	285
weighted avg	1.00	1.00	1.00	285

```
[[174      0]  
 [ 0 111 ]]
```

En los datos de False serían los resultados benignos, mientras que los resultados del True representan los que deberán ser malignos.

Como se puede apreciar, con este nueva forma de clasificar el desempeño mejora en todos los casos, la mayoría de ellos alcanzando una clasificación perfecta. En el primer experimento tanto como en el segundo se pueden apreciar aún algunos datos que se intersectan y que resulta difícil clasificarlos en dos clases separadas, pero es un margen mucho más pequeño de los que se notaban en prácticas anteriores, lo cual es un buen avance.

Con los resultados de estos clasificadores se cumple uno de los objetivos que se tenían desde la primer práctica con estos datos, que era el de utilizar los valores máximos y mínimos por separado de los valores de diagnóstico malignos y benignos, resultando en un clasificador que permita separarlos y clasificarlos.

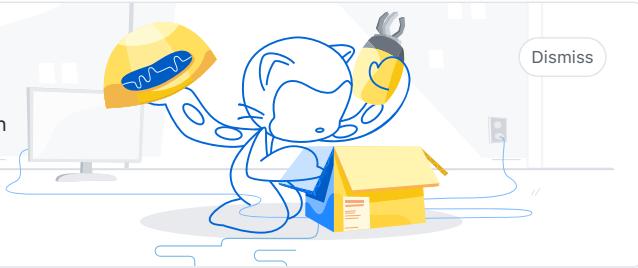
bien !!

All your code in one place

GitHub makes it easy to scale back on context switching. Read rendered documentation, see the history of any file, and collaborate with contributors on projects across GitHub.

[Sign up for free](#)

[See pricing for teams and enterprises](#)



Branch: master ▾ [Ciencia_de_Datos / Práctica 11.ipynb](#)

[Find file](#) [Copy path](#)

mayaberrones94 cambio

5e4e238 9 hours ago

1 contributor

685 lines (684 sloc) | 79.3 KB

[Raw](#) [Blame](#) [History](#) [Edit](#) [Delete](#)

Práctica 11: Agrupamiento de Datos.

El agrupamiento o "clustering" es la tarea de agrupar un conjunto de objetos de tal manera que los miembros del mismo grupo sean más similares, en algún sentido u otro. Es la tarea principal de la minería de datos exploratoria y es una técnica común en el análisis de datos estadísticos. Además es utilizada en múltiples campos como el aprendizaje automático, el reconocimiento de patrones, el análisis de imágenes, la búsqueda y recuperación de información.

El análisis de grupos no es en sí un algoritmo específico, se puede hacer el agrupamiento utilizando varios algoritmos que difieren significativamente en su idea de qué constituye un grupo y cómo encontrarlos eficientemente.

En esta práctica se utilizarán datos diferentes a los que se han utilizado en prácticas anteriores, ya que estos son más sencillos de relacionar con grupos. Se trata de un csv sacado de la página de 'Kaggle' donde describen el género, la raza, el nivel educativo de los padres, y las calificaciones de estudiantes, entre otras cosas.

Lo primero que se realiza, al igual que en prácticas anteriores, es llamar el csv por medio de la herramienta de pandas.

```
In [29]:  
import ssl  
import pandas as pd  
from sklearn import metrics  
from numpy.random import seed  
import matplotlib.pyplot as plt  
from sklearn.cluster import KMeans  
from numpy import isnan, nan, take, where  
import numpy as np  
seed(42)  
  
if getattr(ssl, '_create_unverified_context', None):  
    ssl._create_default_https_context = ssl._create_unverified_context  
e = pd.read_csv("https://raw.githubusercontent.com/mayaberrones94/Ciencia_de_Datos/master/Student  
sPerformance.csv")
```

Se tienen identificados los distintos elementos dentro del csv, por lo que se etiquetan con números para hacer más sencilla su manipulación.

```
In [30]:  
e['race'] = pd.factorize(e.race)[0] + 1  
e['gender'] = pd.factorize(e.race)[0] + 1  
e['prep'] = pd.factorize(e.race)[0] + 1
```

```
In [31]: e.head()
```

	gender	race	parental-ed	lunch	prep	math	read	write	prom
0	1	1	bachelor's degree	standard	1	72	72	74	72.666667
1	2	2	some college	standard	2	69	90	88	82.333333
2	1	1	master's degree	standard	1	90	95	93	92.666667
3	3	3	associate's degree	free/reduced	3	47	57	44	49.333333
4	2	2	some college	standard	2	76	78	75	76.333333


```
In [32]: e['prom-int'] = e['prom'].apply(np.int64)
e.head()
```

	gender	race	parental-ed	lunch	prep	math	read	write	prom	prom-int
0	1	1	bachelor's degree	standard	1	72	72	74	72.666667	72
1	2	2	some college	standard	2	69	90	88	82.333333	82
2	1	1	master's degree	standard	1	90	95	93	92.666667	92
3	3	3	associate's degree	free/reduced	3	47	57	44	49.333333	49
4	2	2	some college	standard	2	76	78	75	76.333333	76


```
In [36]: e['parental-ed'].unique()
```

```
Out[36]: array(['bachelor's degree', 'some college', 'master's degree',
       'associate's degree', 'high school', 'some high school'],
      dtype=object)
```



```
In [40]: e['parental-ed'] = pd.factorize(e.race)[0] + 1
e['lunch'] = pd.factorize(e.race)[0] + 1
e.head()
```

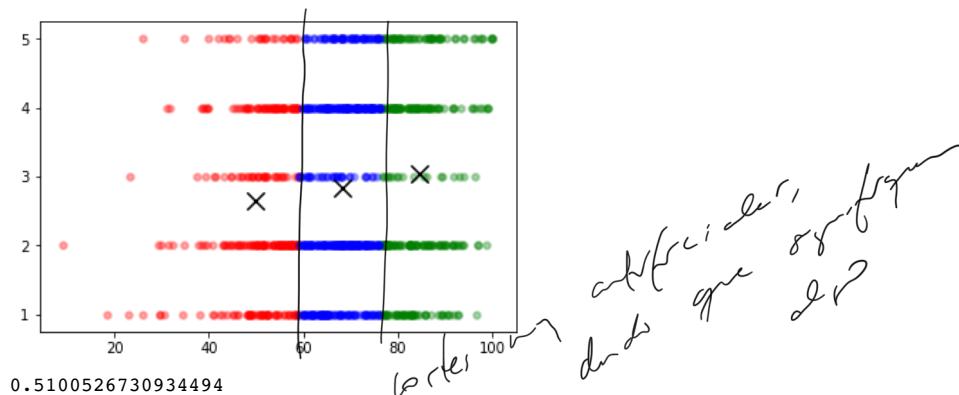
	gender	race	parental-ed	lunch	prep	math	read	write	prom	prom-int
0	1	1	1	1	1	72	72	74	72.666667	72
1	2	2	2	2	2	69	90	88	82.333333	82
2	1	1	1	1	1	90	95	93	92.666667	92
3	3	3	3	3	3	47	57	44	49.333333	49
4	2	2	2	2	2	76	78	75	76.333333	76

Una vez que se tienen los datos de manera más ordenada, se continua siguiendo los pasos dictados en la práctica de prueba para saber la relación que se tienen en los diferentes grupos. En este caso parece interesante saber si el nivel educativo de los padres tiene un papel importante para el resultado final del promedio en calificación del alumno. Se toman entonces como referencia la columna de 'parental-ed' y 'prom'.

```
In [91]: keep = ['prom', 'parental-ed']
d = e.loc[:, keep]
x = d.values
k = 3
m = KMeans(init = 'random', n_clusters = k, n_init = 10)
m.fit(x)
centroides = m.cluster_centers_
grupos = m.predict(x)
plt.figure(1)
plt.scatter(centroides[:, 0], centroides[:, 1], marker='x', s=150, linewidths=3, color='black', zorder=10)
colores = ['r', 'g', 'b']
for g in range(k):
    incl = where(grupos == g)[0]
    print(len(incl), "Grupo", g)
    grupo = take(x, incl, 0)
    plt.scatter(grupo[:, 0], grupo[:, 1], marker='o', s=20, linewidths=2, color=colores[g], alpha = 0.3, zorder=5)
plt.show()
```

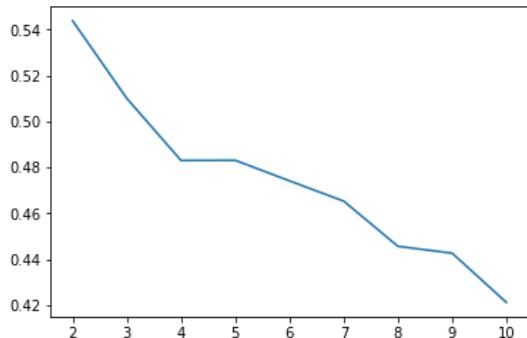
```
print(metrics.silhouette_score(x, grupos, metric='euclidean'))
```

```
274 Grupo 0  
287 Grupo 1  
439 Grupo 2
```



Según la interpretación vista en la práctica de prueba de la Dra. Elisa, se puede concluir con la gráfica que el nivel de educación que tienen los padres influye un poco en la calificación final del alumno, ya que las calificaciones más altas parecen coincidir con el centroide que indica una educación arriba del promedio, mientras que se aprecia también que cuando la calificación disminuye, los centroides de la educación van moviéndose a un nivel más bajo.

```
In [100]: import ssl  
import pandas as pd  
from sklearn import metrics  
from numpy.random import seed  
import matplotlib.pyplot as plt  
from numpy import isnan, nan, take, where  
from sklearn.cluster import KMeans  
  
ks = [k for k in range(2, 11)]  
sil = []  
for k in ks:  
    m = KMeans(init = 'random', n_clusters = k, n_init = 2)  
    m.fit(x)  
    sil.append(metrics.silhouette_score(x, m.predict(x), metric='euclidean'))  
plt.figure(1)  
plt.plot(ks, sil)  
plt.show()
```



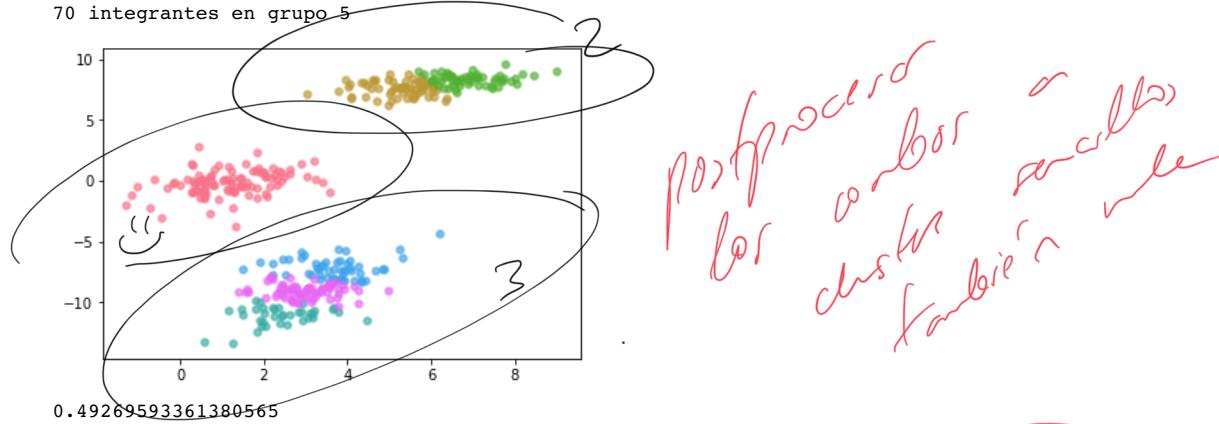
```
In [104]: import ssl  
import pandas as pd  
import seaborn as sns  
from sklearn import metrics  
import matplotlib.pyplot as plt  
from numpy.random import multivariate_normal, seed  
from numpy import take, where, unique, concatenate  
from sklearn.cluster import AffinityPropagation  
seed(17)  
g1 = multivariate_normal([1.3, 0.0], [[1, 0.4], [0.4, 1]], 100)  
g2 = multivariate_normal([6.0, 8.0], [[1.2, 0.3], [0.3, 0.5]], 120)
```

```

g3 = multivariate_normal([3.0, -9.0], [[0.75, 0.6], [0.6, 2]], 160)
x = concatenate((g1, g2, g3))
m = AffinityPropagation(damping = 0.9, convergence_iter = 30)
c = m.fit(x)
grupos = c.labels_
plt.clf()
plt.figure(1)
k = len(unique(grupos))
colores = sns.color_palette("husl", k)
for g in range(k):
    incl = where(grupos == g)[0]
    print(len(incl), "integrantes en grupo", g)
    grupo = take(x, incl, 0)
    plt.scatter(grupo[:, 0], grupo[:, 1], marker='o', s=20, linewidths=2, color=colores[g], alpha = 0.6, zorder=5)
plt.show()
print(metrics.silhouette_score(x, grupos, metric='euclidean'))

```

100 integrantes en grupo 0
 63 integrantes en grupo 1
 57 integrantes en grupo 2
 37 integrantes en grupo 3
 53 integrantes en grupo 4
 70 integrantes en grupo 5

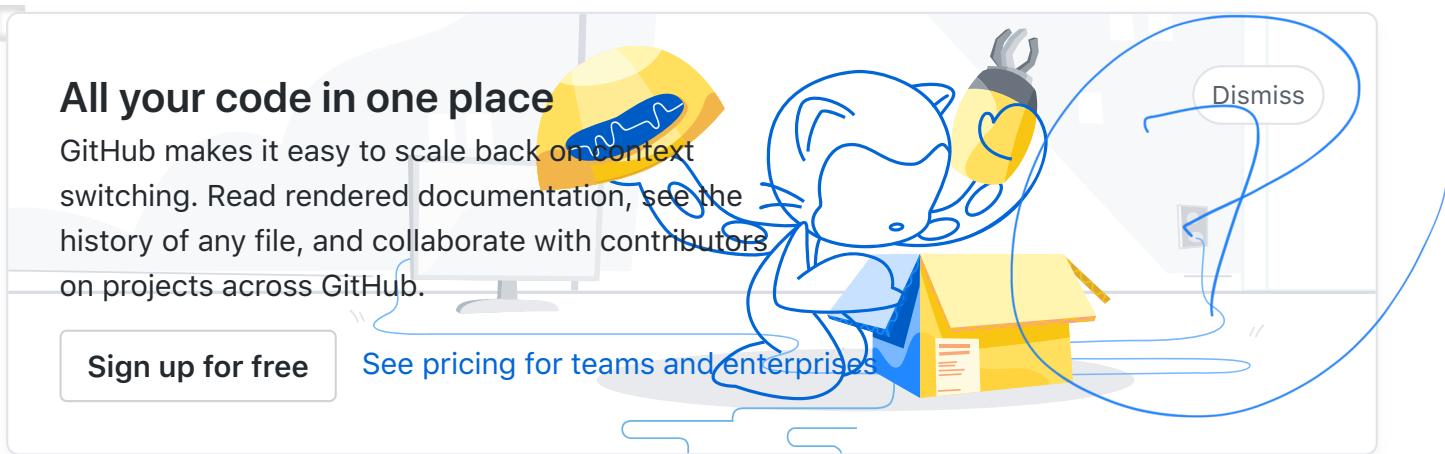


En estos últimos dos experimentos que se realizaron, se intentó mover algunos de los parámetros presentados con la idea de encontrar los valores que dieran una respuesta fácil de interpretar, como pasó en la primera gráfica, pero la única que logró capturar más la atención fue esta final. En este caso se jugó con diferentes valores para la parte de 'multivariate_normal' y no importa que tan grande sea el primer número, ese cluster que se forma en la parte superior sigue siendo igual de separado de los otros de abajo.

Esto puede significar el grupo de alumnos con buena calificación, y esos dos colores que se notan son los dos tipos de nivel de educación a los que llegaron sus padres. En el caso del cluster de en medio, su color se queda fijo y su tamaño de puntos también, por lo que supongo que se traduce a una educación promedio, tiene como resultado calificaciones promedio.

El último cluster representa un misterio, ya que algunas veces, aumentando los otros valores, este cambiaba de color también.

En general, y con los resultados obtenidos puedo sacar como conclusión con estos datos que el nivel de educación que tienen los padres de los alumnos si afecta un poco en la calificación global que estos puedan obtener. La razón porque la diferencia sea tan pequeña en la primera gráfica, puede ser que el nivel académico no refleje el compromiso o la inteligencia de los padres, más sin embargo, los recursos disponibles por estas personas para poder completar sus estudios.



The illustration shows a person sitting at a desk, looking at a computer screen. The screen displays a GitHub interface with the message "All your code in one place". The person is surrounded by various icons representing different GitHub features like code review, issues, and pull requests. A "Dismiss" button is visible in the top right corner of the illustration's frame.

All your code in one place

GitHub makes it easy to scale back on context switching. Read rendered documentation, see the history of any file, and collaborate with contributors on projects across GitHub.

[Sign up for free](#) [See pricing for teams and enterprises](#)

Branch: master ▾

[Ciencia_de_Datos](#) / Práctica 12.ipynb

[Find file](#)

[Copy path](#)



mayraberrones94 im

1587b0b 6 hours ago

1 contributor

1182 lines (1181 sloc) | 447 KB



[Raw](#)

[Blame](#)

[History](#)



Práctica 12: Análisis de texto con nltk y wordcloud.

Para realizar esta práctica se utilizaron datos diferentes de cualquier otro trabajo anterior. Al igual que los demás, este se tomó de la página de Kaggle. Se trata de un csv en donde se toman las descripciones de diferentes tipos de vinos en diferentes partes del mundo.

Para poder iniciar con la práctica es necesario tener todas las herramientas que se van a utilizar instaladas. Se inicia con la herramienta de nltk.

NLTK, es un conjunto de bibliotecas y programas para el procesamiento del lenguaje natural (PLN) simbólico y estadísticos de la distribución de Python. NLTK incluye demostraciones gráficas y datos de muestra. NLTK está

Este módulo contiene funciones y datos de trabajo útil para destinado a apoyar la investigación y la enseñanza en PLN o áreas muy relacionadas, que incluyen la lingüística empírica, las ciencias cognitivas, la inteligencia artificial, la recuperación de información, y el aprendizaje de la máquina.

```
In [1]: import sys
!conda install --yes --prefix {sys.prefix} nltk

Solving environment: done

## Package Plan ##

environment location: /Users/mayraberrones/anaconda3

added / updated specs:
- nltk

The following packages will be downloaded:

      package          |      build
-----+-----+
  conda-4.6.14       |    py37_0
  2.1 MB

The following packages will be UPDATED:

  conda: 4.5.12-py37_0 --> 4.6.14-py37_0

Downloading and Extracting Packages
  conda-4.6.14      | 2.1 MB      | #####
#|#####| 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
```

Después de haber instalado la nueva librería, podemos empezar a hacer las pruebas. En el caso de esta práctica se tuvo que hacer todos los procesos de terminal fuera de la herramienta de jupyter notebook por que se necesitaba abrir una ventana externa para instalar todas las bases de datos necesarias para realizar el trabajo, por esta razón aparece que todos los paquetes están actualizados.

```
In [2]: import nltk
import ssl
```

```

In [9]: if hasattr(ssl, '_create_unverified_context'):
    ssl._create_default_https_context = ssl._create_unverified_context
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('vader_lexicon')
from nltk.corpus import stopwords
print(stopwords.words("english")[:10])
from nltk.sentiment.vader import SentimentIntensityAnalyzer
s = SentimentIntensityAnalyzer() # en inglés hasta podemos distinguir entre palabras positivas y negativas
print(s.polarity_scores('useless'))
print(s.polarity_scores('marvelous'))
```

```

['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves',
 'you', "you're"]
{'neg': 1.0, 'neu': 0.0, 'pos': 0.0, 'compound': -0.4215}
{'neg': 0.0, 'neu': 0.0, 'pos': 1.0, 'compound': 0.5994}

[nltk_data] Downloading package punkt to
[nltk_data]      /Users/mayraberrones/nltk_data...
[nltk_data] Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]      /Users/mayraberrones/nltk_data...
[nltk_data] Package stopwords is already up-to-date!
[nltk_data] Downloading package vader_lexicon to
[nltk_data]      /Users/mayraberrones/nltk_data...
[nltk_data] Package vader_lexicon is already up-to-date!
```

Una vez que si tienen importados los paquetes necesarios, se procede a mandar llamar el csv por medio de la librería de pandas.

El comando de 'head' se agrega para poder observar como estan distribuidos los datos.

In [10]: `import pandas as pd
d = pd.read_csv("https://raw.githubusercontent.com/mayraberrones94/Ciencia_de_Datos/master/wine-reviews/wine-rev.csv")
d.head(5)`

Out[10]:

	Unnamed: 0	country	description	designation	points	price	province	region
0	0	Italy	Aromas include tropical fruit	Vulkà Bianco	87	NaN	Sicily & Sardinia	East

			broom, brimston...					
1	1	Portugal	This is ripe and fruity, a wine that is smooth...	Avidagos	87	15.0	Douro	N
2	2	US	Tart and snappy, the flavors of lime flesh and...	NaN	87	14.0	Oregon	V V
3	3	US	Pineapple rind, lemon pith and orange blossom ...	Reserve Late Harvest	87	13.0	Michigan	L N S
4	4	US	Much like the regular bottling from 2012, this...	Vintner's Reserve Wild Child Block	87	65.0	Oregon	V V

Ya que se tiene el csv, el siguiente paso es el de limpiar los datos, como se ha hecho en todas las prácticas anteriores en donde se utiliza información nueva.

Los datos que de interés para este trabajo son el de 'country' que es el lugar de donde proviene el vino, 'description' que es la descripción que dieron los consumidores, y por último 'variety' que es el tipo de vino que se maneja.

```
In [31]: header = ['country', 'description', 'variety']
d.to_csv('solo-rev.csv', columns = header)
e = pd.read_csv('solo-rev.csv')
e.head(4)
```

Out[31]:

Unnamed:	country	description	variety
----------	---------	-------------	---------

0	country	description	variety
0	Italy	Aromas include tropical fruit, broom, brimston...	White Blend
1	Portugal	This is ripe and fruity, a wine that is smooth...	Portuguese Red
2	US	Tart and snappy, the flavors of lime flesh and...	Pinot Gris
3	US	Pineapple rind, lemon pith and orange blossom ...	Riesling

```
In [32]: e.drop(e.columns[[0]], axis=1, inplace=True)
e.head(2)
```

Out[32]:

	country	description	variety
0	Italy	Aromas include tropical fruit, broom, brimston...	White Blend
1	Portugal	This is ripe and fruity, a wine that is smooth...	Portuguese Red

Ya que se tienen los datos sin información innecesaria, se hacen los mismos análisis para ver cuales son los datos con los que se puede realizar en la práctica, como por ejemplo, saber cuales son los datos únicos de 'country', ya que en la práctica de ejemplo de la Dra Elisa se utilizan las diferentes calificaciones de exámenes, en este se utilizarán distintos países.

```
In [6]: e['country'].unique()
```

```
Out[6]: array(['Italy', 'Portugal', 'US', 'Spain', 'France', 'Germany',
       'Argentina', 'Chile', 'Australia', 'Austria', 'South Africa',
       'New Zealand', 'Israel', 'Hungary', 'Greece', 'Romania', 'Mexico',
       'Canada', nan, 'Turkey', 'Czech Republic', 'Slovenia',
       'Luxembourg', 'Croatia', 'Georgia', 'Uruguay', 'England',
       'Lebanon', 'Serbia', 'Brazil', 'Moldova', 'Morocco',
       'Peru', 'India', 'Bulgaria', 'Cyprus', 'Armenia', 'Switzerland',
       'Bosnia and Herzegovina', 'Ukraine', 'Slovakia', 'M
```

```
acedonia',
    'China', 'Egypt'], dtype=object)
```

```
In [81]: e.to_csv('s-rev.csv', index=False)
read = pd.read_csv('s-rev.csv')
read.head(2)
```

	country	description	variety
0	Italy	Aromas include tropical fruit, broom, brimston...	White Blend
1	Portugal	This is ripe and fruity, a wine that is smooth...	Portuguese Red

Fuera de la herramienta de jupyter notebook acomodaron los datos dentro del csv 's-rev', reenombrandolo 'rev-solo' con el programa 'puntuacion.py' proporcionado en la presentación de prueba, más un pequeño ajuste para que reacomodara y quitara el título de las columnas.

```
In [40]: import ssl
import pandas as pd
import matplotlib.pyplot as plt
from wordcloud import WordCloud
from nltk import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import SnowballStemmer
from nltk.tokenize import RegexpTokenizer

if getattr(ssl, '_create_unverified_context', None):
    ssl._create_default_https_context = ssl._create_unverified_context
import pandas as pd
pr = pd.read_csv('rev-solo.csv', header = None)
pr.head(3)
```

	0	1	2
0	Aromas include tropical fruit broom brimstone ...	Italy	White Blend
1	This is ripe and fruity a wine that is smooth ...	Portugal	Portuguese Red
2	Tart and snappy the flavors of lime flesh and ...	US	Pinot Gris

Aquí solo se observa una simple demostración de como quedaron los datos re

acomodados antes de iniciar con los demás experimentos.

Se toman los nombres de las tres columnas en el orden en el que están en el csv. El idioma de dichos datos está en inglés, y cuando se revisaron si había filas sin contestar, estas ya estaban rellenas con la leyenda de 'NaN', por lo que se modifica esa parte también.

```
In [41]: pr.columns = ['description', 'country', 'variety']
n = len(pr)
spa = stopwords.words("english")
stemmer = SnowballStemmer('english')
tokenizer = RegexpTokenizer(r'\w+') # para eliminar puntuación
reemplazos = []
for r in range(n):
    original = pr.description[r]
    reemplazo = ''
    if original != 'NaN':
        quedar = [stemmer.stem(p) for p in tokenizer.tokenize(original) if p.lower() not in spa]
        reemplazo = ' '.join(quedar)
    reemplazos.append(reemplazo)
pr['limpios'] = reemplazos
texto = ' '.join(reemplazos)
nube = WordCloud().generate(texto)
plt.rcParams["figure.figsize"] = [15, 7]
plt.imshow(nube)
plt.axis("off")
plt.show()
```



Al correr el experimento se pueden ver las palabras más repetidas en las descripciones de los vinos. Parecen ser congruentes, ya que se concentran en

los sabores que se aprecian en un vino.

Ahora, para separar los datos, se hacen 3 archivos csv diferentes, con tres de los países que se encuentran en la columna 'country'. Estos fueron elegidos porque eran los primeros tres que aparecían en la lista. Los nombres son 'US', 'Italy' y 'Portugal'.

```
In [52]: header = ['description', 'country', 'variety']
writer = e[e['country'] == 'US']
writer.to_csv('us-wine.csv', columns = header, index=False)

lol = pd.read_csv('us-wine.csv')
lol.head(4)
```

Out[52]:

	description	country	variety
0	Tart and snappy, the flavors of lime flesh and...	US	Pinot Gris
1	Pineapple rind, lemon pith and orange blossom ...	US	Riesling
2	Much like the regular bottling from 2012, this...	US	Pinot Noir
3	Soft, supple plum envelopes an oaky structure ...	US	Cabernet Sauvignon

```
In [53]: header = ['description', 'country', 'variety']
writer = e[e['country'] == 'Italy']
writer.to_csv('italy-wine.csv', columns = header, index=False)

lol = pd.read_csv('italy-wine.csv')
lol.head(4)
```

Out[53]:

	description	country	variety
0	Aromas include tropical fruit, broom, brimston...	Italy	White Blend
1	Here's a bright, informal red that opens with ...	Italy	Frappato
2	This is dominated by oak and oak-driven aromas...	Italy	Nerello Mascalese
3	Delicate aromas recall white flower and

3 citrus...

Italy

White Blend

```
In [54]: header = ['description', 'country', 'variety']
writer = e[e['country'] == 'Portugal']
writer.to_csv('portugal-wine.csv', columns = header, index =False)

lol = pd.read_csv('portugal-wine.csv')
lol.head(4)
```

Out[54]:

	description	country	variety
0	This is ripe and fruity, a wine that is smooth...	Portugal	Portuguese Red
1	Grown on the sandy soil of Tejo, the wine is t...	Portugal	Touriga Nacional
2	This bottling shows a rich, wood-aged wine, fu...	Portugal	Portuguese White
3	From an estate in the south of the Alentejo, t...	Portugal	Portuguese Red

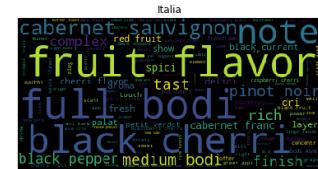
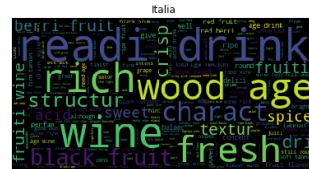
```
In [68]: import ssl
import pandas as pd
import matplotlib.pyplot as plt
from wordcloud import WordCloud
from nltk import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import SnowballStemmer
from nltk.tokenize import RegexpTokenizer

if getattr(ssl, '_create_unverified_context', None):
    ssl._create_default_https_context = ssl._create_unverified_context
dini = pd.read_csv("portugal-wine1.csv", header = None)
dmcu = pd.read_csv("us-wine1.csv", header = None)
dord = pd.read_csv("italy-wine1.csv", header = None)
cols = ['description', 'country', 'variety']
dini.columns = cols
dmcu.columns = cols
dord.columns = cols
spa = stopwords.words("english")
stemmer = SnowballStemmer('english')
tokenizer = RegexpTokenizer(r'\w+') # para eliminar puntuación
plt.rcParams["figure.figsize"] = [18, 9]
f = plt.figure()
lbls = ["Portugal", "US", "Italia"]
```

```

i = 1
for d in [dini, dmcu, dord]:
    lbl = lbls.pop(0)
    reemplazos = []
    for r in range(len(d)):
        original = d.description[r]
        reemplazo = ''
        if original != 'NaN':
            quedar = [stemmer.stem(p) for p in tokenizer.tokenize(original) if p.lower() not in spa]
            reemplazo = ' '.join(quedar)
            reemplazos.append(reemplazo)
    d['limpios'] = reemplazos
    for tipo in lbls:
        sf = plt.subplot(3, 3, i)
        i += 1
        texto = ' '.join(reemplazos)
        sf.set_title(tipo)
        nube = WordCloud().generate(texto)
        sf.imshow(nube)
        sf.axis("off")
plt.tight_layout()
plt.show()

```



Siguiendo los pasos del ejemplo se tienen tres mapas de texto diferentes. En estos casos ya se puede apreciar una diferencia notoria en cada uno, dejando más claro cuales son las características que los distinguen de los demás países.

Para la siguiente parte del experimento se necesitó descargar una librería nueva, llamada 'pyspellchecker'. Esta herramienta es utilizada para asegurar de que no existan errores de ortografía dentro de los bloques de texto que se le introduce al código como descripción.

```
In [72]: import sys
!{sys.executable} -m pip install pyspellchecker
```

```
Collecting pyspellchecker
  Downloading https://files.pythonhosted.org/packages/de/4a/9d85c7ddeb8761ab71ef064d2ac1b1af6cae728367f5c4f74a5d1adc1360/pyspellchecker-0.4.0-py3-none-any.whl (1.9MB)
  100% |██████████| 1 9MB 5 3MB/s
```

```
eta 0:00:01
Installing collected packages: pyspellchecker
Successfully installed pyspellchecker-0.4.0
You are using pip version 19.0.3, however version 19.1.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
```

```
In [1]: import ssl
import pandas as pd
import matplotlib.pyplot as plt
from wordcloud import WordCloud
from nltk import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import SnowballStemmer
from nltk.tokenize import RegexpTokenizer
from spellchecker import SpellChecker
sc = SpellChecker()
adicionales = [ 'ready', 'texture' ]
if getattr(ssl, '_create_unverified_context', None):
    ssl._create_default_https_context = ssl._create_unverified_context
d = pd.read_csv("portugal-winel.csv", header = None)
d.columns = [ 'description', 'country', 'variety' ]
spa = stopwords.words("english")
stemmer = SnowballStemmer('english')
tokenizer = RegexpTokenizer(r'\w+') # para eliminar puntuación
plt.rcParams["figure.figsize"] = [12, 6]
texto = ''
for original in d.description:
    palabras = tokenizer.tokenize(original)
    revisadas = []
    for p in sc.known(palabras):
        revisadas += [p] * palabras.count(p) # para no alterar las frecuencias
    for p in sc.unknown(palabras):
        revisadas += [sc.correction(p)] * palabras.count(p)
    inicial = [stemmer.stem(p) for p in revisadas if p.lower() not in spa]
    quedar = [p for p in inicial if p not in adicionales]
    texto += ' '.join(quedar) + ' '
plt.title('')
nube = WordCloud().generate(texto)
plt.imshow(nube)
plt.axis("off")
plt.show()
```

```
-----
-----
ValueError                                Traceback (most
recent call last)
~/anaconda3/lib/python3.7/site-packages/spellchecker/spell
checker.py in _check_if_should_check(word)
    228         try: # check if it is a number (int, floa
t, etc)
```

All your code in one place

GitHub makes it easy to scale back on context switching. Read rendered documentation, see the history of any file, and collaborate with contributors on projects across GitHub.

Dismiss

Sign up for free See pricing for teams and enterprises

Branch: master ▾ Ciencia_de_Datos / Práctica 13.ipynb Find file Copy path

mayraberrones94实践13 4cc650f 4 days ago

1 contributor

1.99 MB Download History

Práctica 13: Análisis de imágenes.

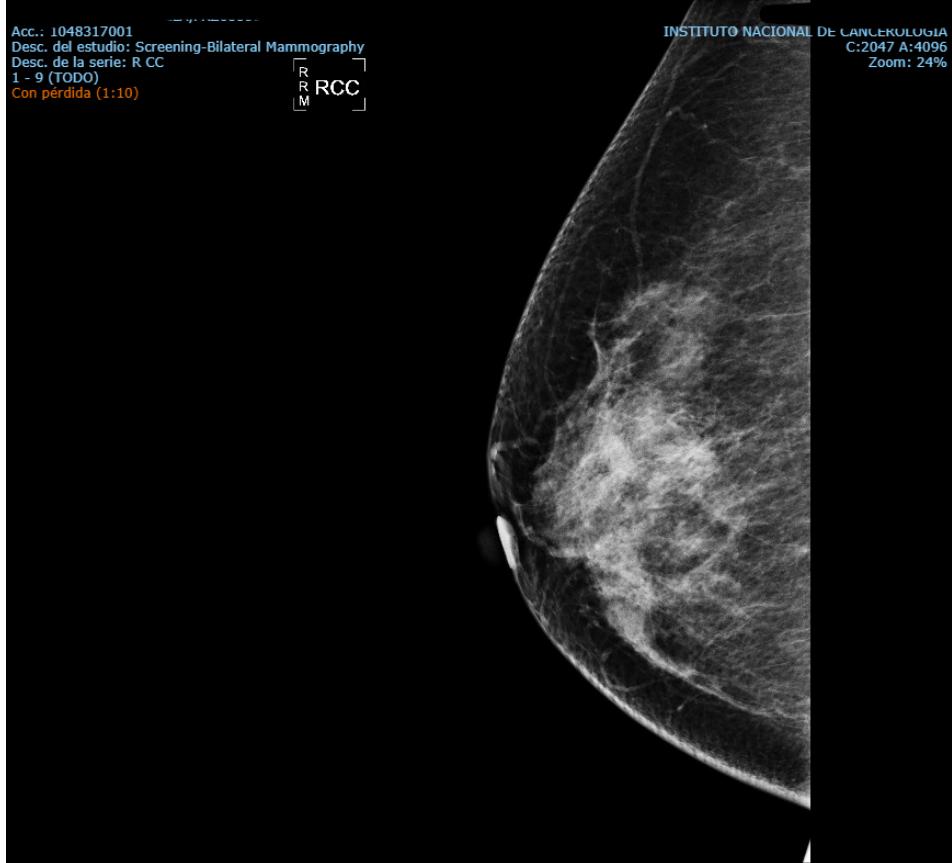
Al iniciar el trabajo en estas prácticas se tomaron como datos de entrada un csv con el título de "Breast Cancer Wisconsin (Diagnostic) Data Set" que contiene las características de una aspiración de aguja de una anomalía encontrada en una mamografía. Estos datos se utilizaron con la finalidad de complementar los conocimientos sobre el cáncer de mama y las posibles herramientas de diagnóstico que pueden crearse.

En esta práctica se quieren tomar las herramientas de manipulación de imágenes tales como OpenCV, ImageMagick, y Pillow, todas ellas controlables en base a Python o la terminal del ordenador, y modificar una imagen de mamografía para en este caso poder realizar lo que se conoce como máscara ROI, que es básicamente un parche que utilizan los médicos para localizar las anomalías dentro de las imágenes.

Para iniciar se importa la imagen de prueba, y las librerías que se van a utilizar. Ya que esta base de datos aún está en proceso de ser publica, aún tiene marcados los datos en sus esquinas. Se borró a mano el nombre del paciente y los datos de fecha de publicación, siguiendo la normativa programada por el Instituto Nacional de Cancerología para proteger los datos de los pacientes.

```
In [8]: from PIL import Image, ImageFilter, ImageEnhance  
i = Image.open("maligna-1.png")  
i
```

Out[8]:



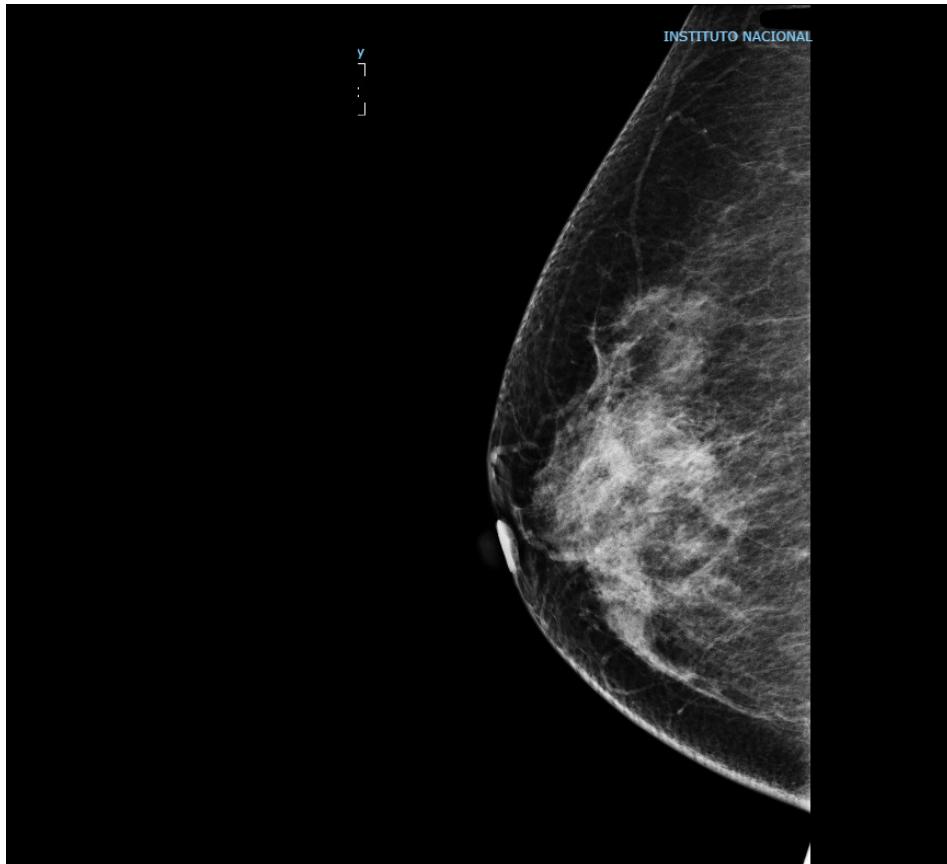
A pesar de que ya no está el nombre del paciente, aún causa ruido todo el demás texto, por lo que se utilizan parches negros para tapar los datos creados con la herramienta PIL. Una vez creados, se guarda la imagen nueva para poder seguir manipulandola.

```
In [9]: from PIL import Image, ImageChops, ImageFont, ImageD  
raw, ImageEnhance  
import numpy as np  
  
draw = ImageDraw.Draw(i)
```

```
#draw.rectangle(((0, 0), (0, 0)), fill="black")
draw.rectangle(((0, 0), (320, 110)), fill="black")
draw.rectangle(((740, 0), (900, 90)), fill="black")

i.save("crop-mall.png")
crop = Image.open("crop-mall.png")
crop
```

Out[9]:



Teniendo la imagen de esta manera, se pueden recortar los bordes negros, para que la imagen pueda ser más pequeña y ocupe menos procesamiento. El recorte se realiza fuera del 'notebook' dentro de la terminal del sistema. El resultado es el de la imagen llamada 'image_out.png'.

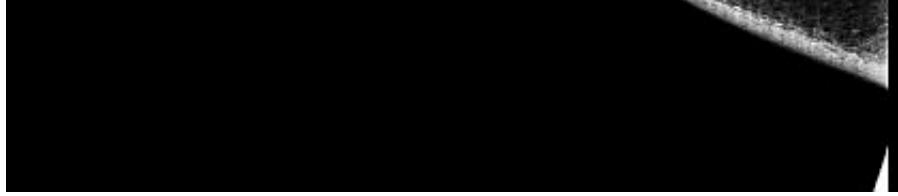
Después de realizar este recorte, se le da a la imagen una serie de filtros, todos documentados en la página de la librería PIL.

Los filtros utilizados son 'Sharp', 'Detail' y 'Contour'. Este último se desprecia ya que lo que se quiere lograr es un detalle de las posibles anomalías, y este filtro le quita muchas de las características importantes para ayudar esta tarea.

```
In [6]: i = Image.open("image_out.png")
sharpener = ImageEnhance.Sharpness(i)
sharpened = sharpener.enhance(3.0)
sharpened.save('Sharp-mall.png')
ima2 = Image.open("Sharp-mall.png")
#im.save('normales/no-{}.png'.format(i))
ima2
```

Out[6]:



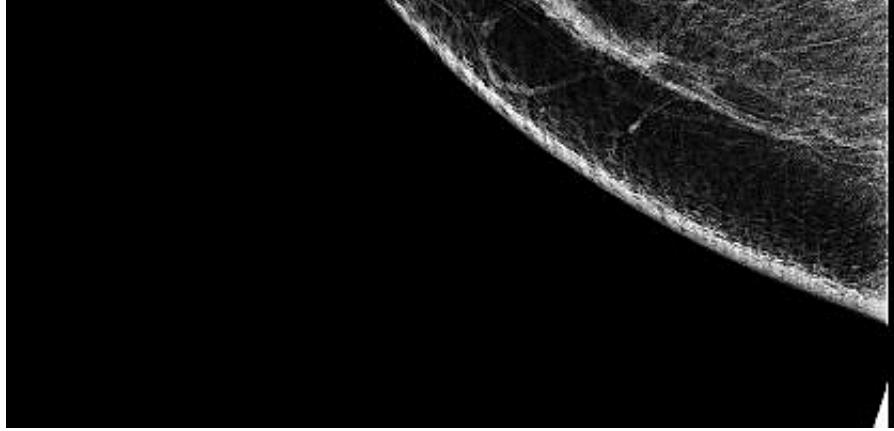


```
In [32]: from PIL import Image, ImageFilter, ImageEnhance
i = Image.open("Sharp-mall.png")

i.filter(ImageFilter.DETAIL).save("detail-m1.png")
detail = Image.open("detail-m1.png")
#im.save('normales/no-{}.png'.format(i))
detail
```

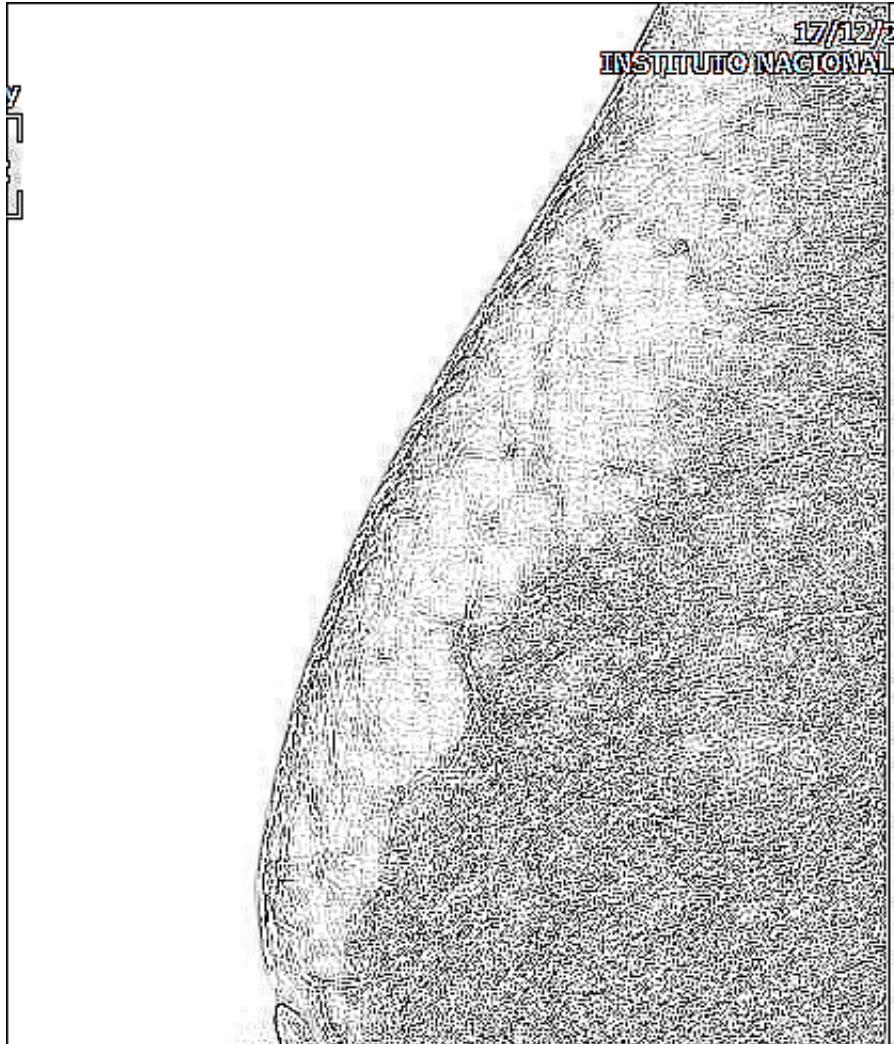
Out[32]:

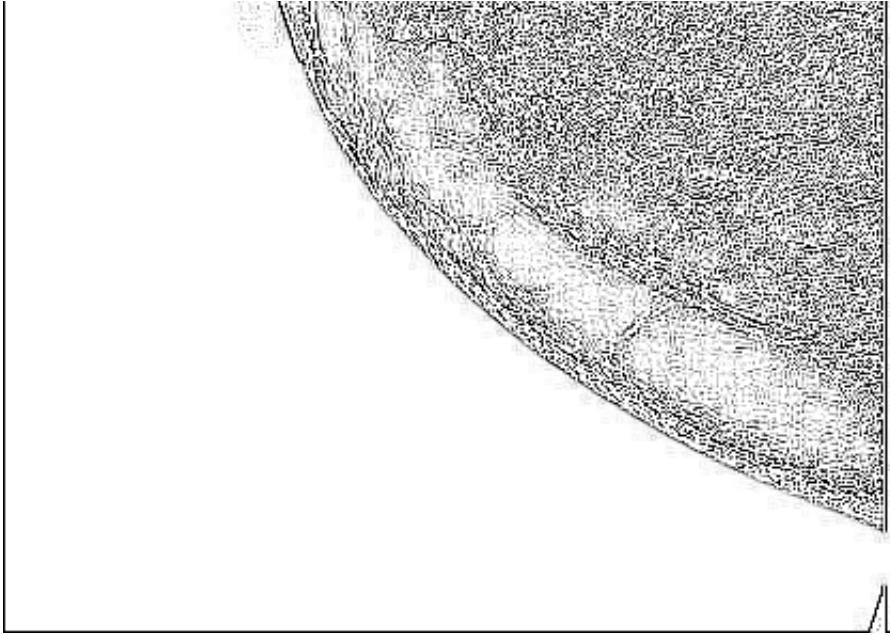




```
In [33]: from PIL import Image, ImageFilter, ImageEnhance  
i = Image.open("detail-m1.png")  
  
i.filter(ImageFilter.CONTOUR).save("contour-m1.png")  
contour = Image.open("contour-m1.png")  
contour
```

Out[33]:





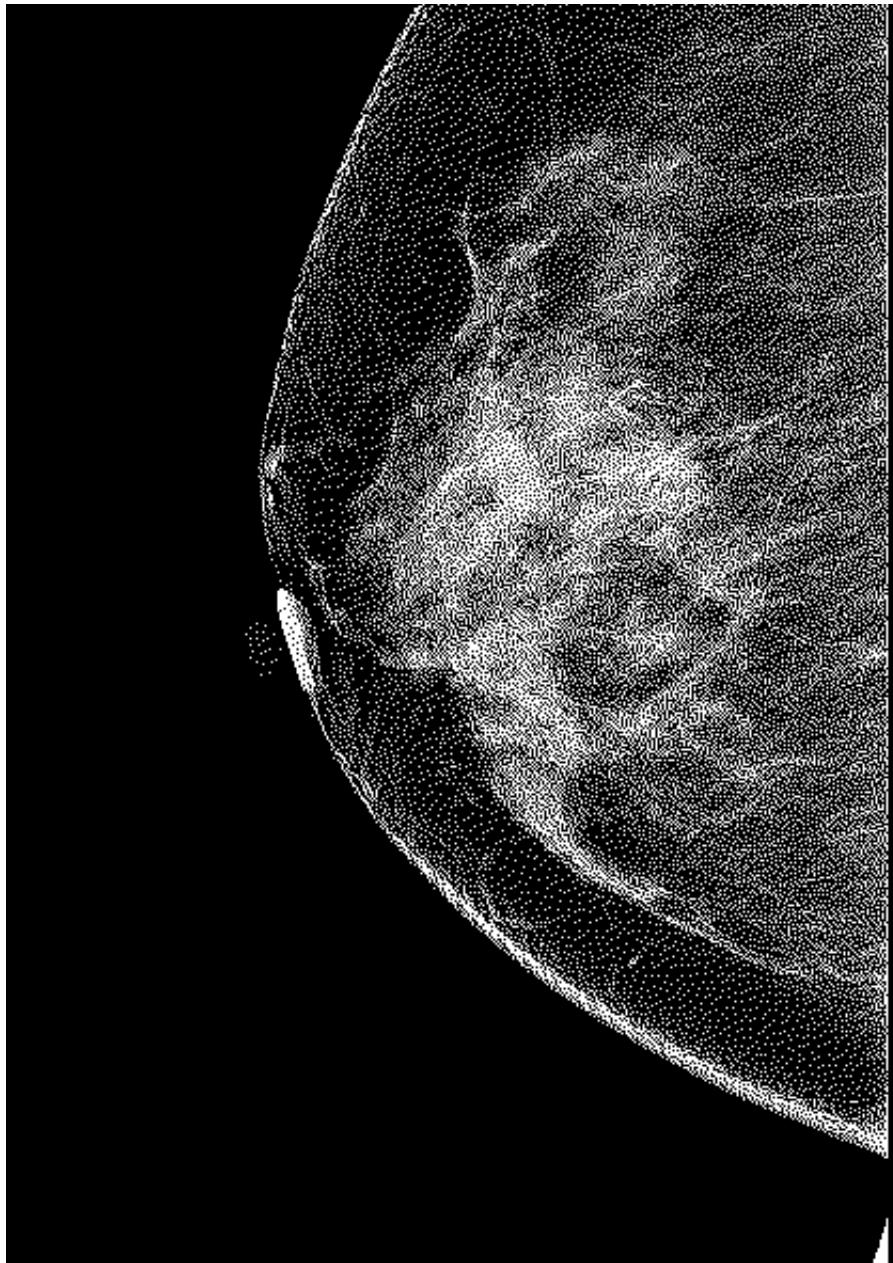
Las imágenes de mamografía más antiguas están en formato de blanco y negro, como es el ejemplo de la base de datos Mini-Mias, que ha estado disponible desde los inicios del año 1999. Bases de datos más recientes, como lo es la base de la que se tomó la imagen de ejemplo para esta práctica ya están desarrolladas en escala RGB para poder mejorar su calidad. Como paso siguiente en esta práctica se transforma la imagen a blanco y negro, siguiendo los pasos de la práctica de prueba de la Dra. Elisa.

```
In [34]: import ssl
import requests
imagen = Image.open("detail-m1.png")
nuevo = imagen.convert('1')
print(nuevo.size)
nuevo
```

(419, 786)

```
Out[34]:
```

A binary (black and white) version of the mammogram image. The image is mostly black, with a large, irregular white shape representing the breast tissue. In the top right corner, there is text that reads "17/12/2" and "INSTITUTO NACIONAL". The image is framed by a thick black border.



Con la imagen de esta manera se puede apreciar un poco más donde se encuentra la mayor concentración de masa. No es la norma, pero usualmente cuando existe una gran concentración de masa en la imagen es probable que ahí se encuentre una anomalía. Dependiendo de la forma de dicha masa es cuando los expertos la pueden clasificar en anomalía benigna o maligna.

El siguiente paso será el de limpiar esos pequeños puntos alrededor de la masa principal para poder crear la máscara final. En el ejemplo de la práctica, la Dra. Elisa establece que el umbral tomó varios intentos antes de llegar a uno significativo, por lo que se decidió también mover

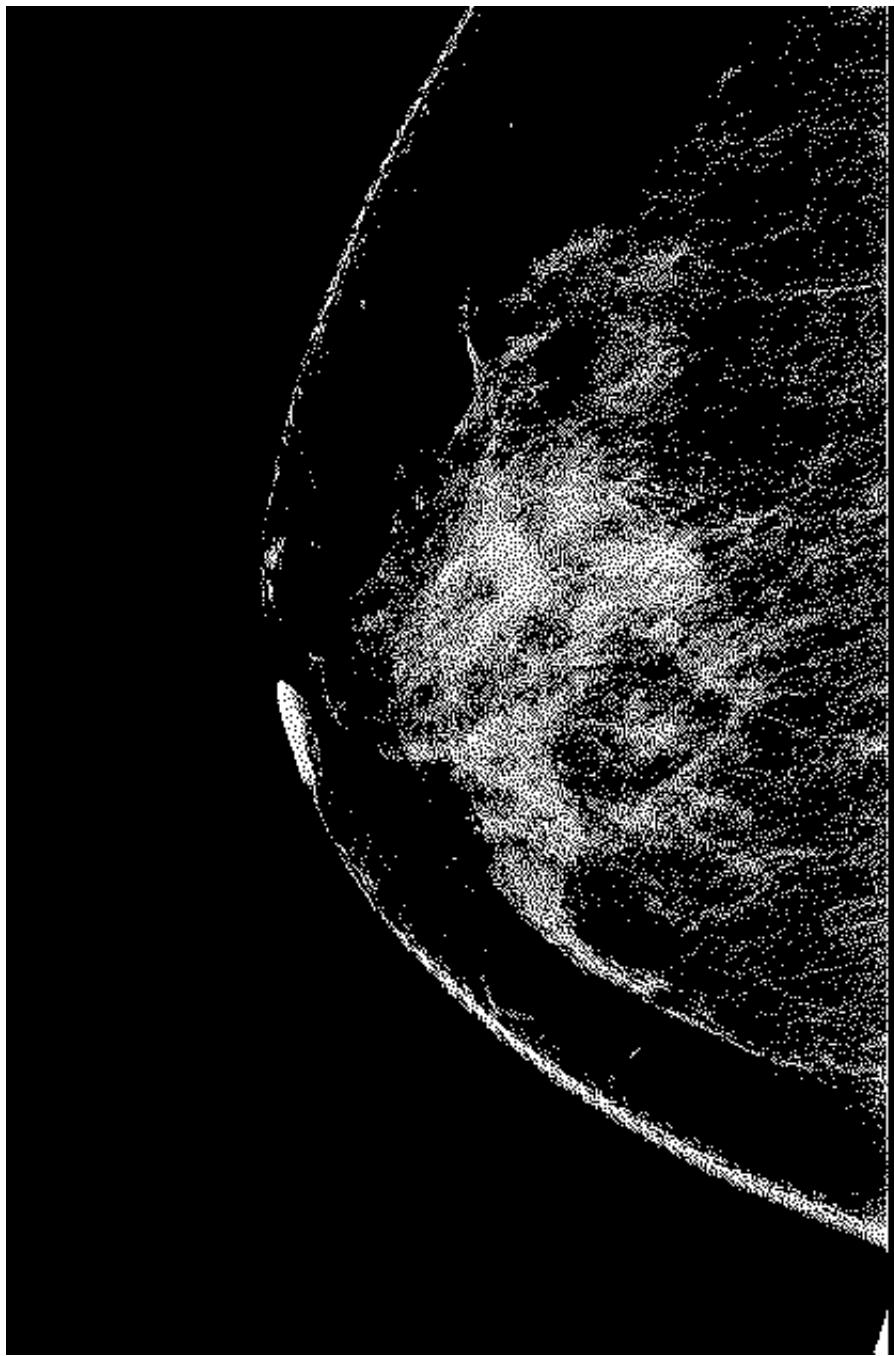
los datos. Entre menor el umbral, menos puntos aparecen, pero después de probar con números desde 50, y subiendo en intervalos de 10, se llegó a la conclusión de que el 150 es un umbral bueno para quitar los puntos, sin perder muchas características de la masa principal.

```
In [38]: import ssl
import requests
from PIL import Image, ImageDraw
if getattr(ssl, '_create_unverified_context', None):
    ssl._create_default_https_context = ssl._create_unverified_context

negro = (0, 0, 0)
blanco = (255, 255, 255)
umbral = 150 #90 era el primero
n = Image.open("detail-m1.png")
w, h = n.size
P = n.load()
for f in range(h): # bordes verticales
    if P[0, f] == blanco:
        ImageDraw.floodfill(n, (0, f), negro)
    if P[w - 1, f] == blanco:
        ImageDraw.floodfill(n, (w - 1, f), negro)
for c in range(w): # bordes horizontales
    if P[c, 0] == blanco:
        ImageDraw.floodfill(n, (c, 0), negro)
    if P[c, h - 1] == blanco:
        ImageDraw.floodfill(n, (c, h - 1), negro)
for f in range(h):
    for c in range(w):
        rgb = P[c, f]
        if max(rgb) - min(rgb) > umbral: # tiene un
color que no es gris
            P[c, f] = negro
b = n.convert('1') # binaricemos lo que queda
b.save('rgb-m1.png')
b
```

Out[38]:





Se tiene la imagen más limpia, pero aún puede hacerse una modificación para quitar esa parte de poca densidad en la parte derecha de la imagen. En el código anterior se quitaron los pixeles blancos que no tenían vecinos, para poder quitar esos puntos solitarios alrededor de la masa. En este siguiente código se quitan los puntos sin vecino, y con un solo vecino.

In [23]:

```
import ssl
import requests
```

```

from PIL import Image

if getattr(ssl, '_create_unverified_context', None):
    ssl._create_default_https_context = ssl._create_unverified_context

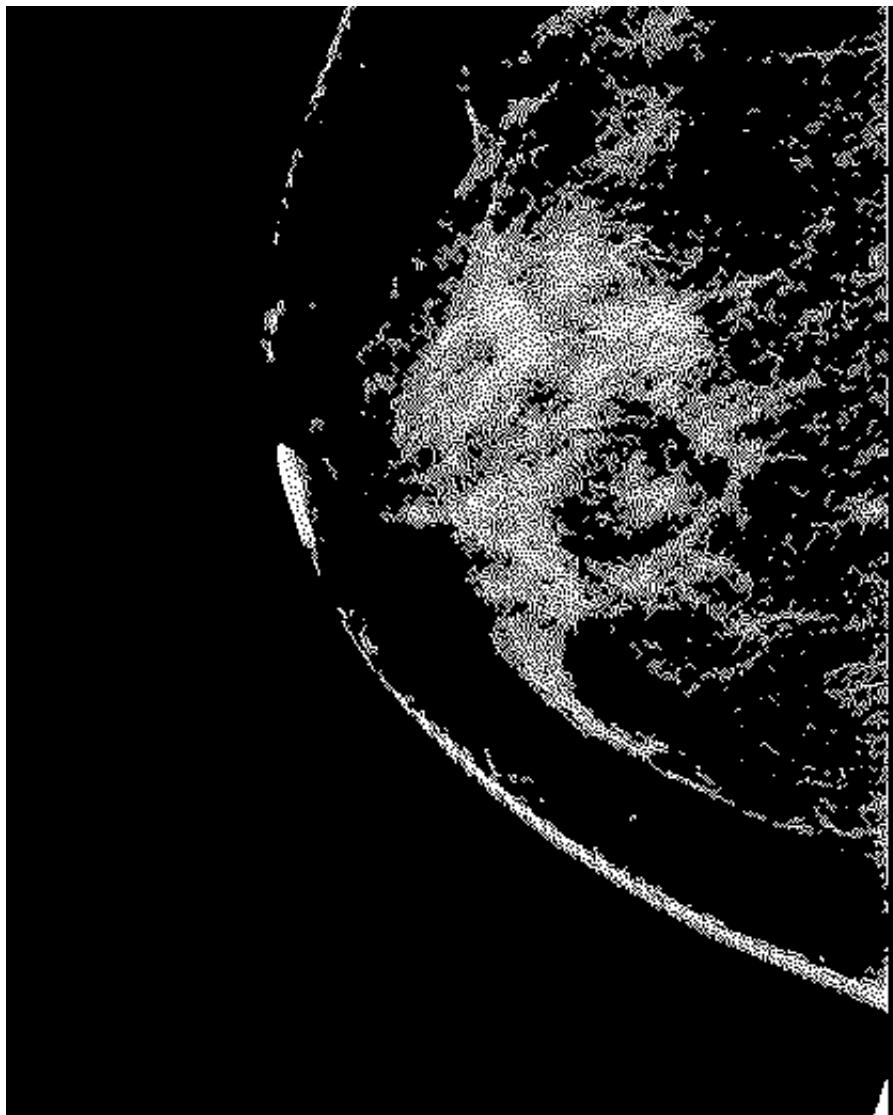
nuevo = b
P = nuevo.load()
ancho, altura = nuevo.size
borrados = 0
vecinos = [(-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 1),
           (1, -1), (1, 0), (1, 1)]
for fila in range(altura):
    for columna in range(ancho):
        if P[columna, fila] == 255: # pixel es negro
            contador = 0
            for (df, dc) in vecinos:
                vf = fila + df
                vc = columna + dc
                if vf >= 0 and vc >= 0 and vf < altura and vc < ancho: # si existe el vecino
                    if P[vc, vf] == 255:
                        contador += 1
            if contador < 2: # uno o cero vecinos negros
                P[columna, fila] = 0 # será blanco
                borrados += 1
print(borrados, "pixeles negros eliminados")
nuevo.save('pixeles-m1.png')
pixeles = Image.open("pixeles-m1.png")
pixeles

```

13121 pixeles negros eliminados

Out[23]:





Para quitar el ruido final de la imagen, se utilizó el ejemplo de la práctica de la Dra Elisa en donde una de las imágenes se queda con ruidos y líneas pequeñas, que es más o menos lo que pasa con esta imagen.

```
In [24]: import ssl
import requests
from PIL import Image, ImageDraw

if getattr(ssl, '_create_unverified_context', None):
    ssl._create_default_https_context = ssl._create_
unverified_context

n = nuevo
w, h = n.size
rgb = n.convert('RGB')
```

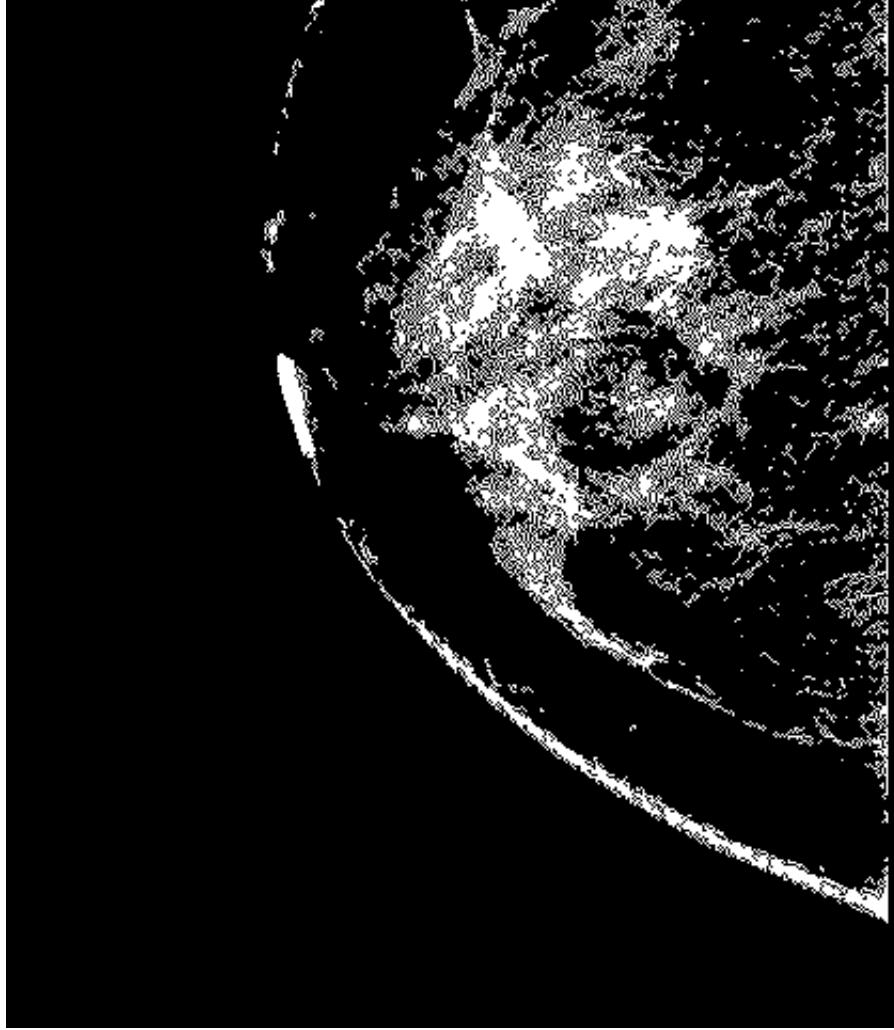
```

P = rgb.load()
negro = (0, 0, 0)
blanco = (255, 255, 255)
for f in range(h): # bordes verticales
    if P[0, f] == blanco:
        ImageDraw.floodfill(rgb, (0, f), negro)
    if P[w - 1, f] == blanco:
        ImageDraw.floodfill(rgb, (w - 1, f), negro)
for c in range(w): # bordes horizontales
    if P[c, 0] == blanco:
        ImageDraw.floodfill(rgb, (c, 0), negro)
    if P[c, h - 1] == blanco:
        ImageDraw.floodfill(rgb, (c, h - 1), negro)
n = rgb.convert('1')
P = n.load()
V = [(-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 1), (1, -1), (1, 0), (1, 1)]
for f in range(1, h - 1): # sin bordes ahora
    for c in range(1, w - 1):
        if P[c, f] == 0:
            cont = 0
            for (df, dc) in V:
                if P[c + dc, f + df] == 0: # siempre
existen
                    cont += 1
            if cont < 2:
                P[c, f] = 255 # blanco
n.save('vecinos-m1.png')
vecinos = Image.open("vecinos-m1.png")
vecinos

```

Out[24]:





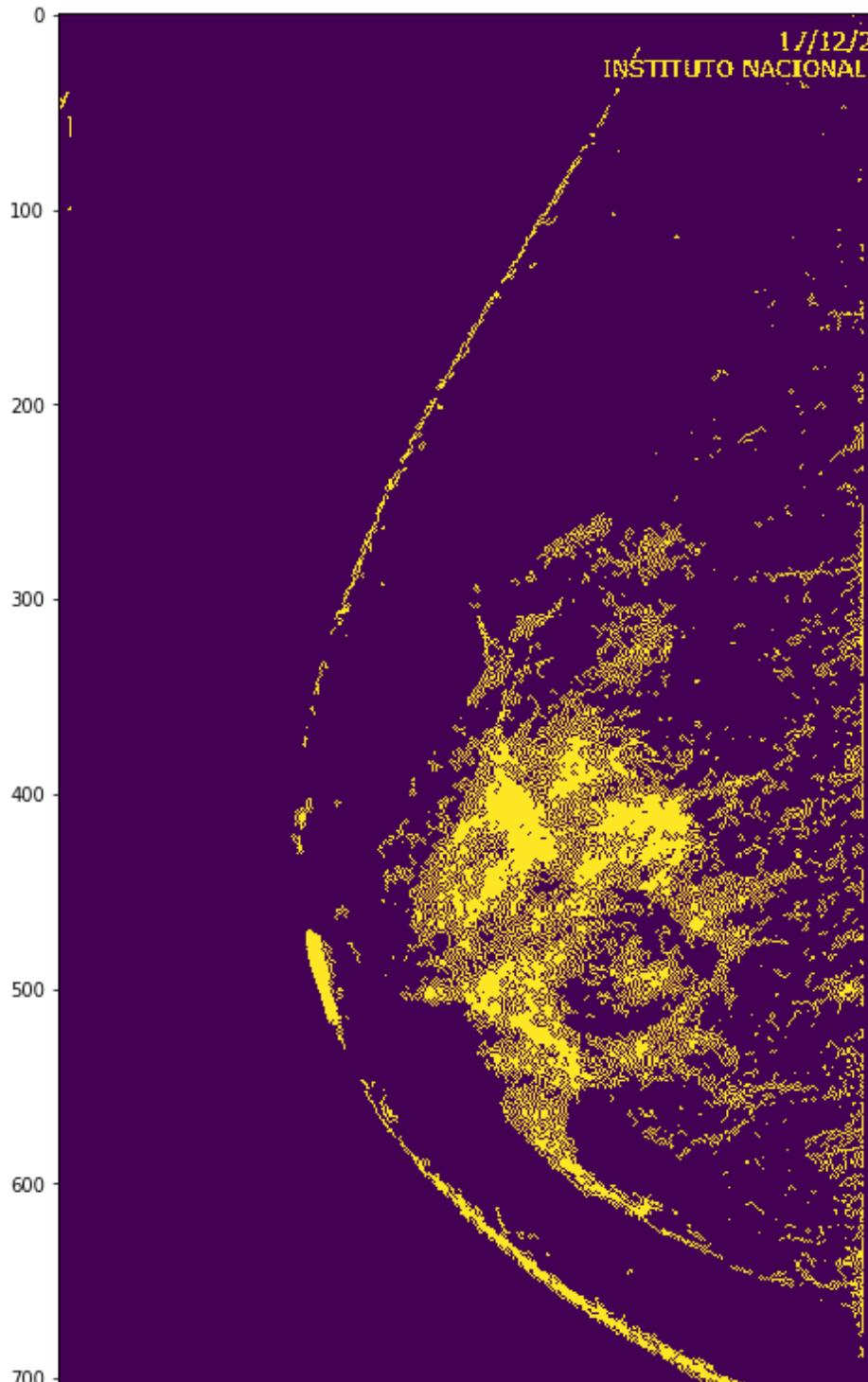
Después de hacer esto, es mucho más claro cuales son las zonas de más alta concentración. Ahora se cambia de herramienta de PIL a OpenCV, intentando sacar solo el contorno de las figuras en blanco que se ven en la imagen, logrando realizar una máscara.

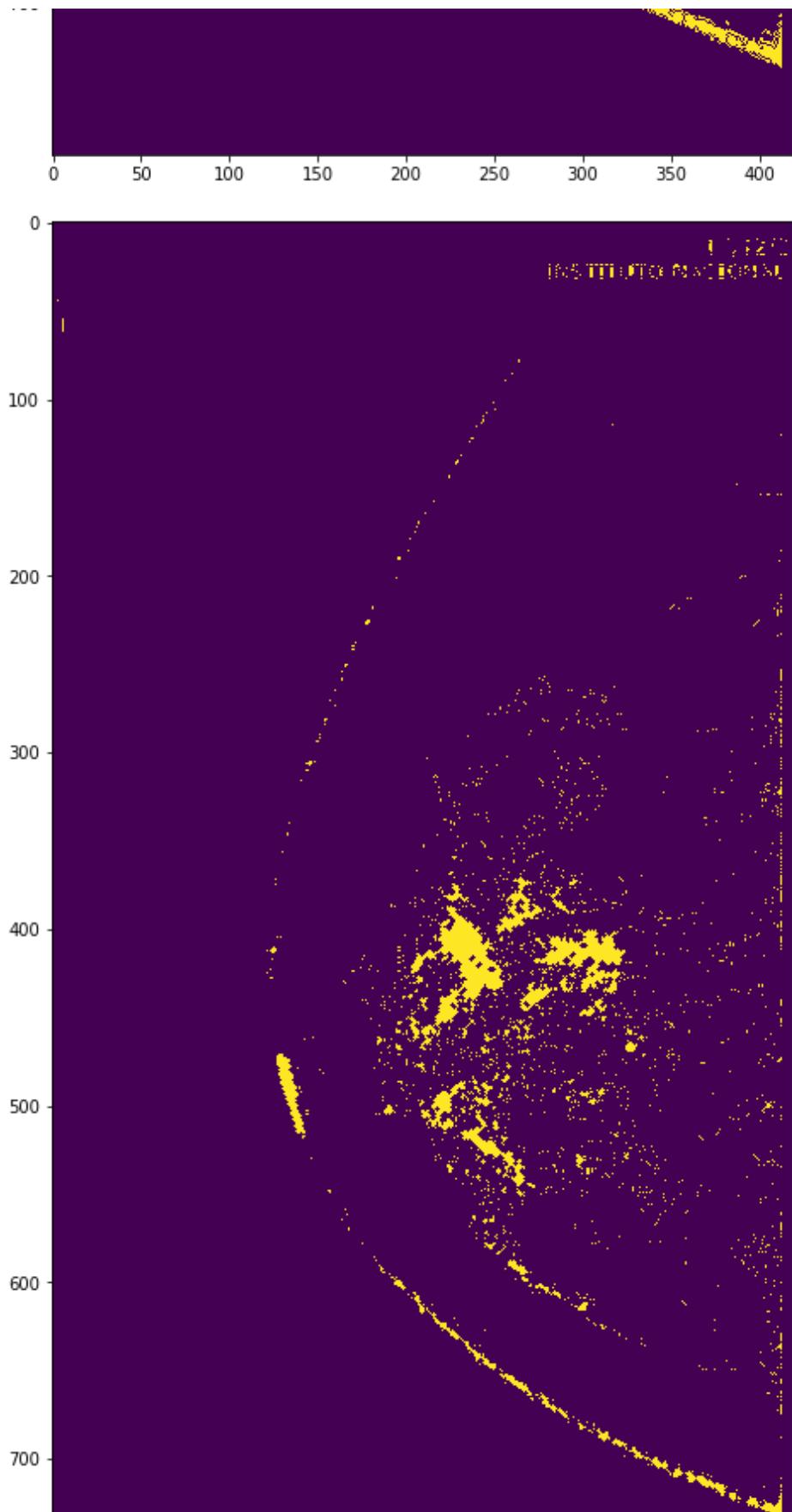
```
In [34]: import cv2
import numpy as np
import matplotlib.pyplot as plt
from skimage import io
%matplotlib inline
im = io.imread('vecinos-m1.png')
plt.figure(figsize=(15,15))
plt.imshow(im)

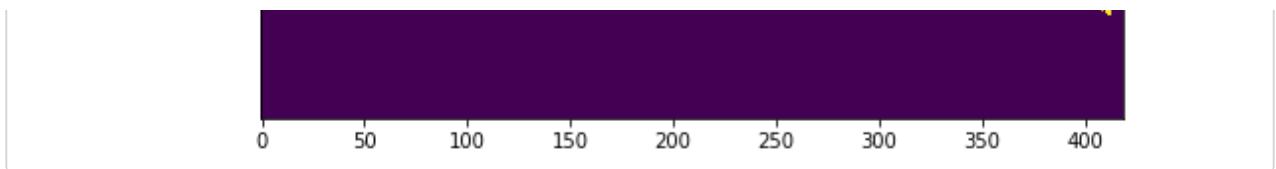
# Contoured image
ret,thresh = cv2.threshold(im, 120,255, cv2.THRESH_BINARY)
```

```
contours = cv2.findContours(threshn, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)[-2]
for contour in contours:
    cv2.drawContours(im, contour, -1, (0, 255, 0), 1)
plt.figure(figsize=(15,15))
plt.imshow(im)
```

Out[34]: <matplotlib.image.AxesImage at 0x1c2d560898>







Comentarios de las prácticas.

- Práctica 1 En esta práctica iniciamos escogiendo los datos con los que se trabajaría el resto del curso. Por conveniencia, se buscó un tema que concordara con el tema que se tiene para la tesis de maestría. En la página de Kaggle se encontró una base de datos sobre la información que se sacaban de las biopsias hechas a anomalías de cáncer de seno.
- En esta primera parte nos concentraremos en limpiar los datos y saber un poco más como estaban conformados. También se definieron una serie de preguntas que debían de responderse para el final del curso.
- Práctica 2 En esta segunda práctica se iniciaron procedimientos un poco más avanzados para revisar los datos y seguir modificándolos para poder hacer el siguiente grupo de experimentos para sacar información pertinente.
- Práctica 3 En esta práctica se quería hacer uso estadístico básico con la información que teníamos, como conteos, promedios, correlaciones etc. Otra cosa que incluimos aquí fue a categorizar la información, incluso haciendo nuevos documentos con estos nuevos datos. Todo esto manejando instrucciones sencillas en Python y la herramienta de Pandas.
- Práctica 4 En esta práctica empezamos a utilizar la herramienta de Plotly para la visualización de gráficas que realizamos con los datos sacados de la práctica anterior. Aquí tuve problemas técnicos con el jupyter por lo cual opté por usar otra herramienta para la creación de gráficas llamada Seaborn.
- Aquí se concentró más en ver las diferencias entre ver las características que tenían los datos con la etiqueta de malignos a comparación de aquellos benignos.
- Práctica 5 Esta práctica se entregó después de tiempo. Aquí el trabajo consiste en realizar algunas pruebas estadísticas con las librerías de plotly, Python y matplotlib, buscando saber si los datos que se tienen son normales o no mediante gráficas. Más aquí en esta práctica la parte que más me llamó la atención es la gráfica de histograma en donde comparamos los dos estados de maligno y benigno, ya que se puede ver una área en medio que comparten características, y la cual se supuso como uno de los problemas que se tendrían más adelante para poder hacer un clasificador.
- Práctica 6 A partir de esta práctica en la cual nos pedían modelar los aspectos de los datos con modelos lineales simples, pudimos sacar dos de las características que nos ayudarían en las prácticas siguientes para realizar el clasificador con mayor exactitud.

Práctica 7 En esta práctica se inició a trabajar con los modelos de regresión multiple para poder crear una clasificación. Aquí se tomaron los dos datos que se estudiaron en la práctica anterior y se intentó hacer un clasificador con las herramientas de Python. Al finalizar todas las experimentaciones, se tuvo un resultado bueno en los tres parámetros de sensibilidad, precisión y especificidad, más revisando los resultados de los falsos positivos y negativos, nos dimos cuenta que este resultado, aunque bueno, aún deja mucho que desear por la gravedad que implica tener tantos resultados incorrectos.

Práctica 8 En esta práctica realizamos pruebas estadísticas más sofisticadas de las que utilizamos en prácticas anteriores. Estas herramientas son las del ANOVA y el PCA.

Los resultados de esta práctica me sirvieron mucho para poder avanzar con mi tesis, ya que llegue a un punto en donde tenía tantas variables y parámetros que muchas de las funciones que sabía en RStudio no me servían. Pero con la forma de programarlo en Python pude darle seguimiento a mi parte de Análisis de resultados y diseño de experimentos.

Práctica 9 Para esta práctica empezamos a trabajar con pronósticos, pero para esto requieres ventanas de tiempo, las cuales no contaba mis datos. En este caso volví a hacer una revisión de bases de datos en Kaggle y encontré una de cantidad de accidentes que pasaban en Barcelona a lo largo del año 2017 para intentar hacer una predicción de la cantidad de accidentes que podían pasar en cualquier otro año en un mes en específico.

Práctica 10 En esta práctica ya volvimos con los datos del cáncer de mama, ya que empezamos a trabajar con clasificación de datos con la herramienta de Sklearn. En estos experimentos se utilizaron los datos que se traían desde la práctica 6, y con sus resultados fue posible hacer un clasificador bueno con dos de sus características, lo que cumple con uno de los objetivos planteados desde la primer práctica.

Práctica 11 En esta práctica también se tuvo que hacer cambio de base de datos utilizada, ya que trataba de hacer agrupamiento de datos. El único agrupamiento que podía hacer con los datos anteriores era el de benigno con maligno y eso creaba una bolita no muy descriptiva, ya que el cluster de datos era muy cercano.

Con los nuevos datos se pudo hacer una hipótesis inicial distinta. Trataba sobre si los estudios de los padres pueden afectar al nivel de estudios que llegan sus hijos.

Práctica 12 Para esta práctica también se utilizaron datos distintos a los que se iniciaron. En este caso se tomó una base de datos de opiniones acerca del sabor de vinos de diferentes lugares y diferentes clases.

Esta práctica me llamó mucho la atención, ya que es una manera bastante fácil de poder agrupar las palabras claves de algún bulto de texto, o darte cuenta de cuales son las que más se repiten dentro del mismo.

Práctica 13 Por ultimo se llega a la práctica final que es la de procesamiento de imágenes. Como el tema que traía desde un inicio era para poder enriquecer un poco lo que tenía para mi tesis, se incluyeron algunas de las imágenes que utilzo para poder realizar mi clasificador. En este caso intenté hacer una mascara ROI de una imagen de mamografía, utilizando las herramientas de binarización y filtros de imágenes que tiene la herramienta de PIL. Esta práctica me dio varias ideas para los pasos siguientes de mi clasificador.