Sesion 2 Analisis anomalías

Jose Fernando Zea y Fernando López-Torrijos

8 de abril de 2021

Prueba de Grubbs

Una primera prueba para detección de datos extremos anterior a los cinco de Tukey fue el test de Grubbs, denominada así por Frank E. Grubbs, quien la publicó en 1950. Se conoce también como prueba residual máxima normalizada o prueba de desviación extrema studentizada. Se utiliza para detectar valores atípicos en un conjunto de datos univariados que se supone que proviene de una población distribuida normalmente.

Obsérvese el supuesto. Es importante.

La prueba de Grubbs detecta un valor atípico a la vez. Este valor atípico se elimina del conjunto de datos y la prueba se repite hasta que no se detectan valores atípicos adicionales. Sin embargo, las probabilidades de detección cambian en cada iteración. La prueba no debe usarse para tamaños de muestra de seis datos o menos, ya que con frecuencia etiqueta la mayoría de los puntos como valores atípicos.

La hipótesis nula H_0 es que no hay datos atípicos. Y la estadística de prueba es:

$$G = \frac{max_{1,\ldots,n} \mid y_i - \bar{y} \mid}{s}$$

siendo $\bar{y}$ la media y s la desviación estándar.

Mide la máxima desviación y la divide entre la desviación estándar. El resultado lo compara con respecto a un valor de referencia sacado a partir de la distribución t de student. De ahí el origen de su nombre alternativo.

La hipótesis se rechaza si $G > \frac{n-1}{n}\sqrt{\frac{t^2_{\alpha/(2n),n-2}}{n-2+t^2_{\alpha/(2n),n-2}}}$

donde $t^2_{\alpha/(2n),n-2}$ denota el valor crítico después del cual se debe considerar un valor extremo.

Se puede definir el test con la estadística sólo hacia un lado (test de una cola): $G = \frac{\bar{y}-y_{min}}{s}$ ó $G = \frac{y_{max}-\bar{y}}{s}$

Python no tiene una función para su cálculo. Se escribe la función *grubbs()* correspondiente:

```
# importar modulos
import os
import pandas as pd
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import datasets
from scipy.stats import t, zscore
import random
```

```
def grubbs(X, test='two-tailed', alpha=0.05):
```

```
    '''
    Ejecuta el test de Grubbs recursivamente hasta que la hipótesis nula sea cierta.

    Parametros
    ----------
    X : ndarray
        El arreglo de numeros (numpy) sobre el cual se desea hallar outliers.
    test : str
        Describe los tipos de outliers que se están buscando. Puede ser 'min'
        (si se buscan los outliers muy pequeños), 'max' (si se buscan los grandes),
        o 'two-tailed' (si se buscan ambos).
    alpha : float
        El nivel de significancia.

    Retorna
    -------
    X : ndarray
        El arreglo original con los outliers removidos.
    outliers : ndarray
        Un arreglo de outliers.
    '''

    Z = zscore(X, ddof=1)  # Z-score
    N = len(X)  # Número de muestras

    # Calcula valores extremos y en valor crítico de la t de student
    if test == 'two-tailed':
        extreme_ix = lambda Z: np.abs(Z).argmax()
        t_crit = lambda N: t.isf(alpha / (2.*N), N-2)
    elif test == 'max':
        extreme_ix = lambda Z: Z.argmax()
        t_crit = lambda N: t.isf(alpha / N, N-2)
    elif test == 'min':
        extreme_ix = lambda Z: Z.argmin()
        t_crit = lambda N: t.isf(alpha / N, N-2)
    else:
        raise ValueError("Test must be 'min', 'max', or 'two-tailed'")

    # Calcula el umbral
    thresh = lambda N: (N - 1.) / np.sqrt(N) * \
        np.sqrt(t_crit(N)**2 / (N - 2 + t_crit(N)**2))

    # Crea un arreglo donde almacena los outliers
    outliers = np.array([])

    # Bucle sobre el arreglo de datos y remueve los outliers
    while abs(Z[extreme_ix(Z)]) > thresh(N):

        # actualiza los outliers
        outliers = np.r_[outliers, X[extreme_ix(Z)]]
        # remueve los outlier del arreglo
        X = np.delete(X, extreme_ix(Z))
        # recalcula el Z score
```

```
        Z = zscore(X, ddof=1)
        N = len(X)

    return X, outliers
```

Sean simulados 20 datos mediante una distribución normal, y se añaden dos datos outlier:

```
np.random.seed(1548)
x1 = np.random.normal(30, 10, 20)
x2 = np.random.randint(50, 100, 2)
x = np.append(x1, x2)
print(x)
```

```
## [25.70407946 41.84081878 40.90397337 34.28815348 24.79200971 25.41526208
##  32.18174438 31.80755495 26.64518827 39.67046269 40.20563416 33.29813472
##  28.31732909 18.26280697 32.78518143 11.63954665 32.69773757 35.02533554
##  31.4466714  13.31562267 52.         68.        ]
```

Llámese a la función.

```
grubbs(x)
```

```
## (array([25.70407946, 41.84081878, 40.90397337, 34.28815348, 24.79200971,
##         25.41526208, 32.18174438, 31.80755495, 26.64518827, 39.67046269,
##         40.20563416, 33.29813472, 28.31732909, 18.26280697, 32.78518143,
##         11.63954665, 32.69773757, 35.02533554, 31.4466714 , 13.31562267,
##         52.        ]), array([68.]))
```

Consideró al número 52 como *normal*, y al 68 como *outlier*.

Se menciona como antecedente histórico de cómo en análisis unidimensional *datos extremos* y *datos anómalos* son equivalentes. Pero también es un ejemplo de una de las formas de determinar anomalías. Si el valor de la estadística G es mayor al umbral especificado, es anómalo. Si no, es *normal*. Se trata de una asignación binaria. Sin ambigüedades. Se verán métodos que asignan una probabilidad, y es deber del analista determinar de una manera razonable el umbral de probabilidad a partir del cual lo clasificará como dato anómalo.

Regresión lineal

El modelo de regresión lineal para muchas variables toma la siguiente forma:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p$$

, donde:

- y es la respuesta
- β_0 es el intercepto
- β_1 es el coeficiente asociado a x_1
- ...
- β_p es el coeficiente asociado a x_p

Los valores de β se conocen como los coeficientes de regresión. Estos valores se estiman en un proceso de ajuste usando la función de perdida cuadrático:

$$\sum_{i=1}^{n} (\hat{y}_i - y_i)^2$$

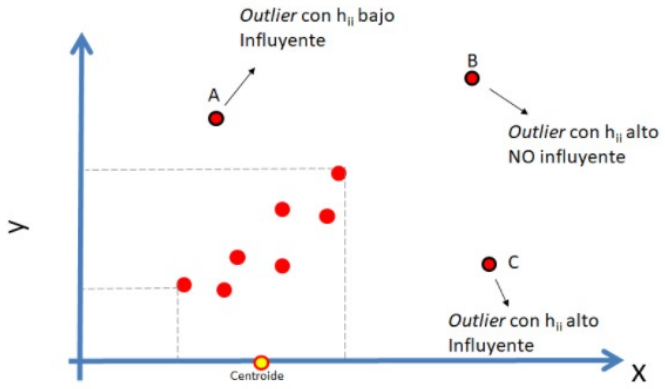
Esta función de perdida se conoce como suma de cuadrados del error.

Se van a exponer varias maneras de utilizar los resultados de una regresión para identificar datos anómalos.

Para exponer las ideas que se van a trabajar, se utiliza una base de datos con medidas corporales.

Fuente: https://fhernanb.github.io/libro_regresion/diag2.html#distancia-de-cook

Medidas de influencia




```
import numpy as np
import pandas as pd
import plotnine
# Una librería alternativa para realizar regresiones usando formula
import statsmodels.formula.api as smf
from plotnine import ggplot, geom_point, aes, geom_abline
```

```
datos = pd.read_csv("medidas_cuerpo2.csv")
```

La formulación del modelo y los coeficientes de regresión son:

```
lm=smf.ols(formula = "Peso ~ Estatura", data = datos).fit()
lm.params
```

```
## Intercept   -66.737734
## Estatura     78.540296
## dtype: float64
```

El coeficiente de determinación del modelo es:

```
lm.rsquared
```

```
## 0.43827250709468546
```

Los primeros seis registros:

```
datos.head()
```

```
##    ID  Peso Sexo  Estatura  circun_cuello  circun_muneca
## 0   1  47.6    F      1.57           29.5           13.9
## 1   2  68.1    M      1.66           38.4           16.0
## 2   3  68.0    M      1.90           36.5           16.6
## 3   4  80.0    M      1.76           38.0           17.1
## 4   5  68.1    M      1.83           38.0           17.1
```

Número de datos total:

```
n=datos.shape[0]
print(n)
```

```
## 26
```

Las medidas influyentes son:

```
p=2
influence = lm.get_influence()
print(influence)
```

```
## <statsmodels.stats.outliers_influence.OLSInfluence object at 0x0000000069888748>
```

Residuales estudentizados

Otra manera de identificar outliers es el uso de los residuales estandarizados:

$$t_i = \frac{\hat{\epsilon}_i}{\hat{\sigma}\sqrt{\frac{n-1}{n}}}$$

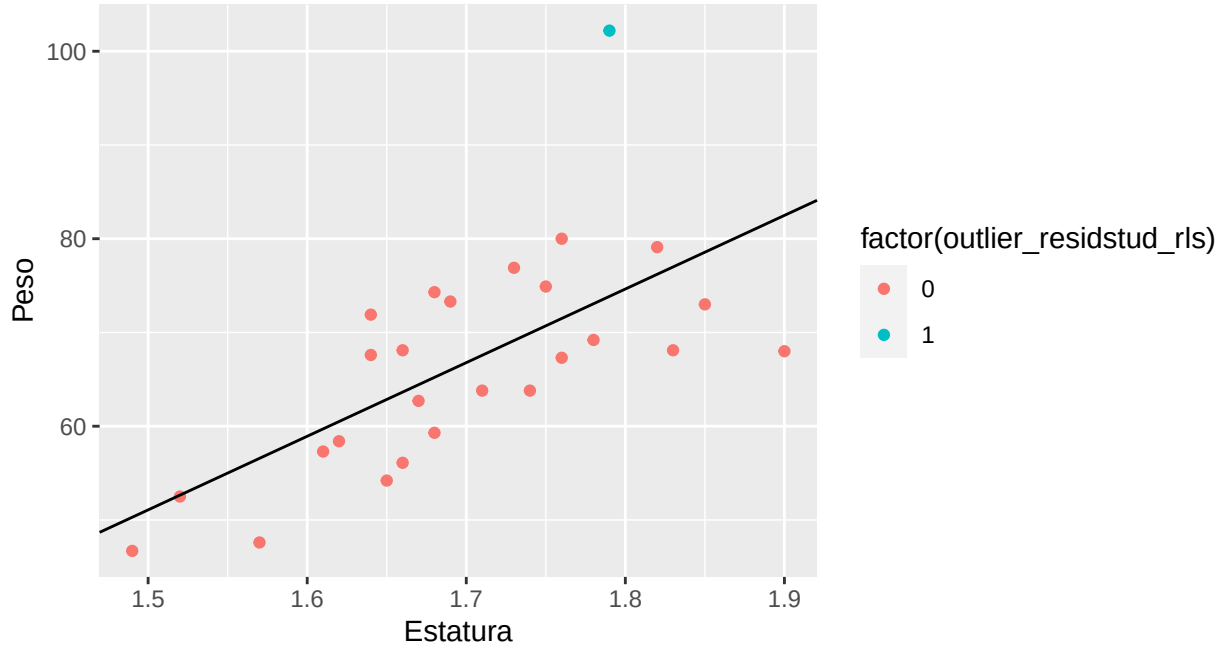
Un umbral razonable para identificar valores extremos según el modelo es considerar los puntos con valor absoluto de los residuales estudentizados mayores a dos:

$$|t_i| > 2$$

```
resid_student = influence.resid_studentized_external
datos[['outlier_residstud_rls']] = np.where(abs(resid_student) >= 2, 1, 0)
```

```
(ggplot(datos, aes('Estatura', 'Peso', color='factor(outlier_residstud_rls)'))
 + geom_point()
 + geom_abline(intercept = lm.params[0], slope = lm.params[1], color = "black"))
```

Se puede visualizar el punto atípico:



Distancia de Cook

Se define como

$$D_i = \frac{1}{p\sigma^2} \sum_{j=1}^{n} \left(\hat{y}_j - \hat{y}_{j(i)}\right)^2$$

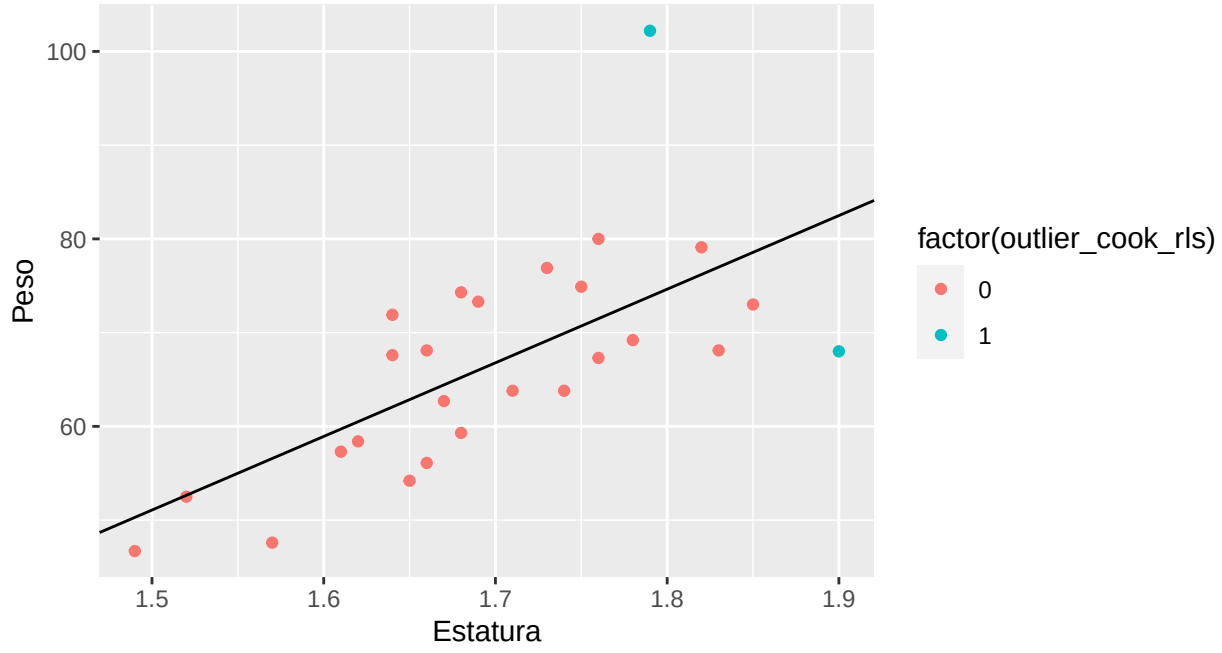
Donde σ corresponde a la varianza de los residuales, $\hat{y}_j$ es el valor pronosticado para la observación j-ésima usando todos los datos en el modelo de regresión y $\hat{y}_{j(i)}$ es el pronóstico para la observación j eliminando la observación i-ésima. p es el número de variables consideradas + 1 (el intercepto también se tiene en cuenta), por ejemplo si se consideran para el modelo de regresión dos variables independientes para pronosticar la variable de interés Y el valor de p = 3.

Un punto se considera influyente si $D_i > \frac{4}{n-k-2}$

```
(cooks, p) = influence.cooks_distance
datos[['outlier_cook_rls']] = np.where(cooks >= 4 /(n-2-2), 1, 0)
```

```
(ggplot(datos, aes('Estatura', 'Peso', color='factor(outlier_cook_rls)'))
 + geom_point()
 + geom_abline(intercept = lm.params[0], slope = lm.params[1], color = "black"))
```

Se puede visualizar el punto atípico que ya quedó también identificado con el método anterior, y un punto adicional:



Estadística dffit

La diferencia "estandarizada" entre el pronóstico de la obsevación i-esima con el modelo haciendo uso de todos los datos y el modelo quitando la observación i-ésima

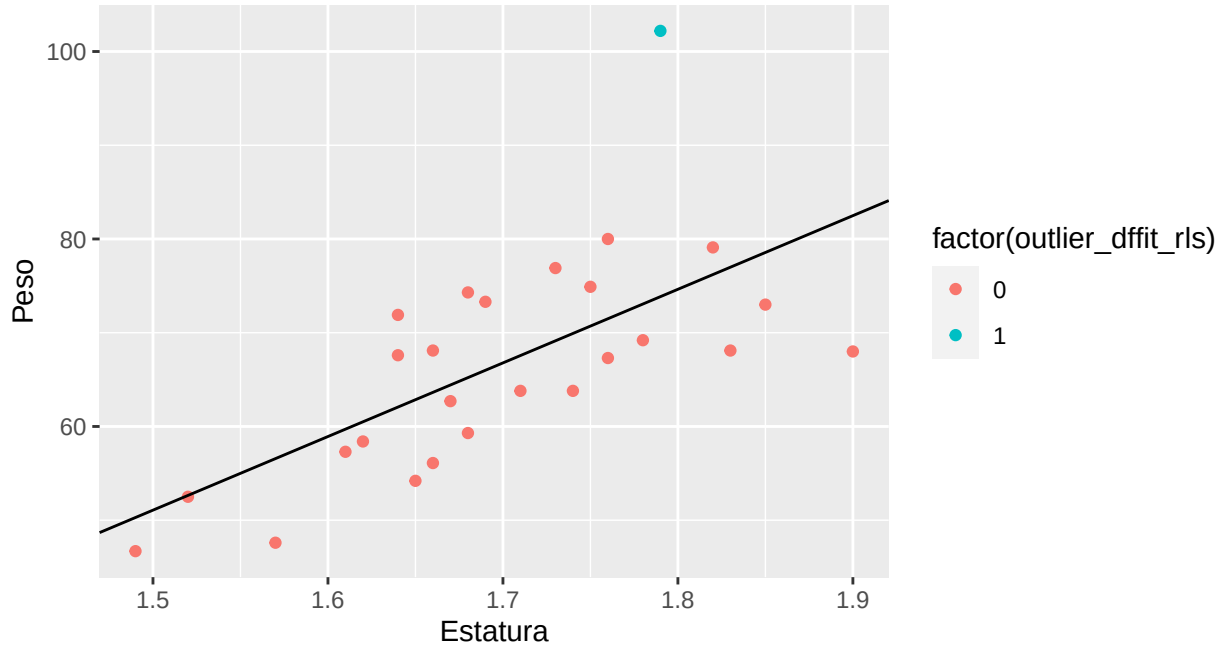
$$DFFIT_i = \frac{\hat{y}_i - \hat{y}_{i(i)}}{\sqrt{s^2_{(i)} h_{ii}}}$$

Si el valor de dffit de la i-ésima es mayor a $2\sqrt{p/n}$ la observación se considera un outlier.

```
(dffits, p) = influence.dffits
datos[['outlier_dffit_rls']] = np.where(dffits > 2 / np.sqrt(p/n), 1, 0)
```

```
(ggplot(datos, aes('Estatura', 'Peso', color='factor(outlier_dffit_rls)'))
 + geom_point()
 + geom_abline(intercept = lm.params[0], slope = lm.params[1], color = "black"))
```

Se pueden visualizar los puntos atípico detectados por la prueba dffit:



Leverege point

Es relevante identificar los valores atípicos en los predictores, para esto se utiliza la matriz de proyección (matriz "hat"):

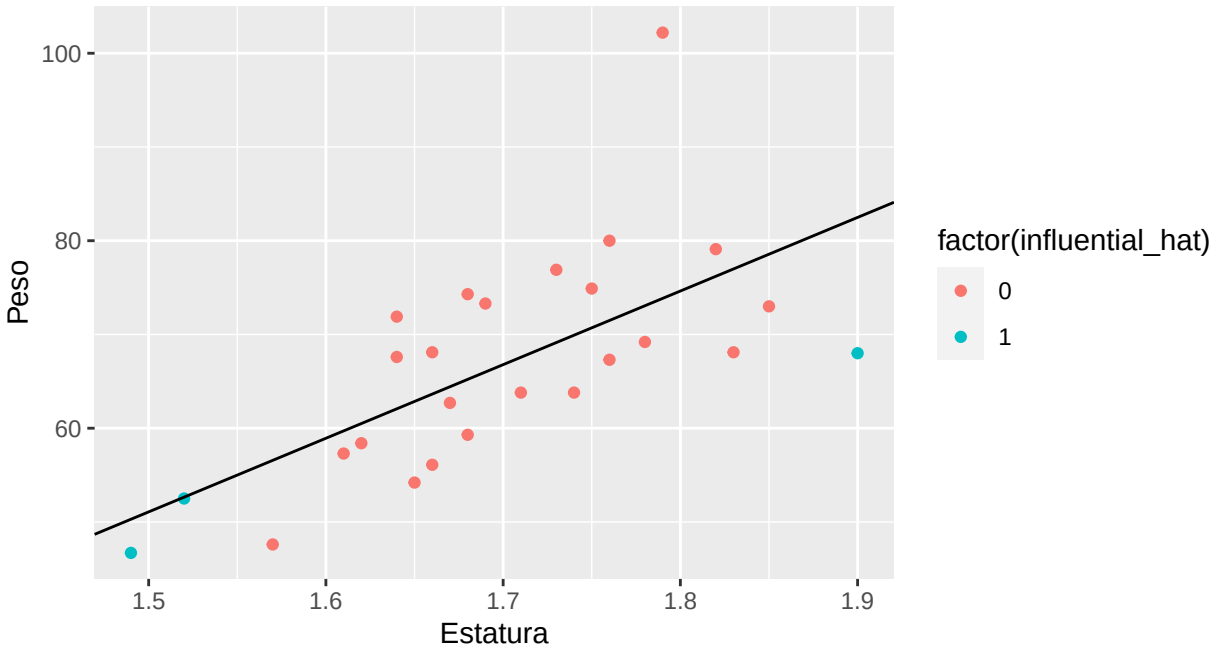
$$H = X(X^T X)^{-1} X^T$$

Identificaremos los valores que son extremos en el espacio de los predictores:

```
leverage = influence.hat_matrix_diag
datos[['influential_hat']] = np.where(leverage >= 2*p/n, 1, 0)
```

```
(ggplot(datos, aes('Estatura', 'Peso', color='factor(influential_hat)'))
 + geom_point()
 + geom_abline(intercept = lm.params[0], slope = lm.params[1], color = "black"))
```

Se puede visualizar los punto atípico detectados mediante la medida de apalancamiento (leverege):




```
datos.head()
```

```
##    ID  Peso Sexo  ...  outlier_cook_rls  outlier_dffit_rls  influential_hat
## 0   1  47.6    F  ...                 0                  0                1
## 1   2  68.1    M  ...                 0                  0                1
## 2   3  68.0    M  ...                 1                  0                1
## 3   4  80.0    M  ...                 0                  0                1
## 4   5  68.1    M  ...                 0                  0                1
##
## [5 rows x 10 columns]
```

Caso multivariado

Una buena estrategia es identificar valores atípicos por varios criteros. En el caso multivariado no es posible visualizar los outliers, sin embargo si se pueden marcar los puntos y analizarlos.

```
datos = pd.read_csv("medidas_cuerpo2.csv")
```

```
lm2 = smf.ols(formula = "Peso ~ Estatura + circun_cuello + circun_muneca", data = datos).fit()
n=datos.shape[0]
p=4
lm2.params
```

```
## Intercept       -62.018297
## Estatura          9.440395
## circun_cuello     2.258735
## circun_muneca     1.989082
## dtype: float64
```

```
lm2.rsquared
```

```
## 0.7582013458665
```

Calculamos las diferentes medidas de influencia bajo el modelo multivariado:

```
influence_rlm = lm2.get_influence()
resid_student_rlm = influence_rlm .resid_studentized_external
(cooks_rlm, p_cooks_rlm) = influence_rlm.cooks_distance
(dffits_rlm, p_dffits_rlm) = influence_rlm.dffits
leverage_rlm = influence_rlm.hat_matrix_diag
```

```
datos[['outlier_residstud_rlm']] = np.where(abs(resid_student_rlm) >= 2, 1, 0)
datos[['outlier_cook_rlm']] = np.where(cooks_rlm >= 4 /(n-p-2), 1, 0)
datos[['outlier_dffit_rlm']] = np.where(dffits_rlm > 2 * np.sqrt(p/n), 1, 0)
datos[['influential_hat_rlm']] = np.where(leverage_rlm >= 2*p/n, 1, 0)
```

```
df_outliers = \
    datos.query('outlier_residstud_rlm == 1 | outlier_cook_rlm == 1 | outlier_dffit_rlm == 1')
df_outliers
```

```
##     ID   Peso Sexo  ...  outlier_cook_rlm  outlier_dffit_rlm  influential_hat_rlm
## 7    8   69.2    M  ...                 1                  0                    0
## 10  11  102.2    M  ...                 1                  1                    0
##
## [2 rows x 10 columns]
```

```
df_outliers = df_outliers.drop(['outlier_residstud_rlm','outlier_cook_rlm', \
    'outlier_dffit_rlm', 'influential_hat_rlm'], 1)
df_outliers
```

```
##     ID   Peso Sexo  Estatura  circun_cuello  circun_muneca
## 7    8   69.2    M      1.78           40.5           16.5
## 10  11  102.2    M      1.79           41.5           17.1
```

Ejercicio

Considerar la estatura de las mujeres e identificar outliers

Random sample consensus

Random sample consensus (RANSAC) es una heurística (algoritmo iterativo) para calcular los parámetros de un modelo de regresión lineal sobre un conjunto de datos observados que contiene valores atípicos. Es no determinista en el sentido de que produce un resultado razonable solo con una cierta probabilidad, mayor a medida que se permita que el algoritmo explore un mayor número de iteraciones. El algoritmo fue publicado por primera vez por Fischler y Bolles SRI International en 1981.

El supuesto es que los datos consisten en "inliers", es decir, datos cuya distribución se explica por un conjunto de parámetros del modelo, aunque, como es usual en detección de anomalías, pueden estar sujetos a ruido, y "valores atípicos".

RANSAC asume que, dado un conjunto de *inliers* (generalmente pequeño), existe un procedimiento que permite estimar los parámetros del modelo.

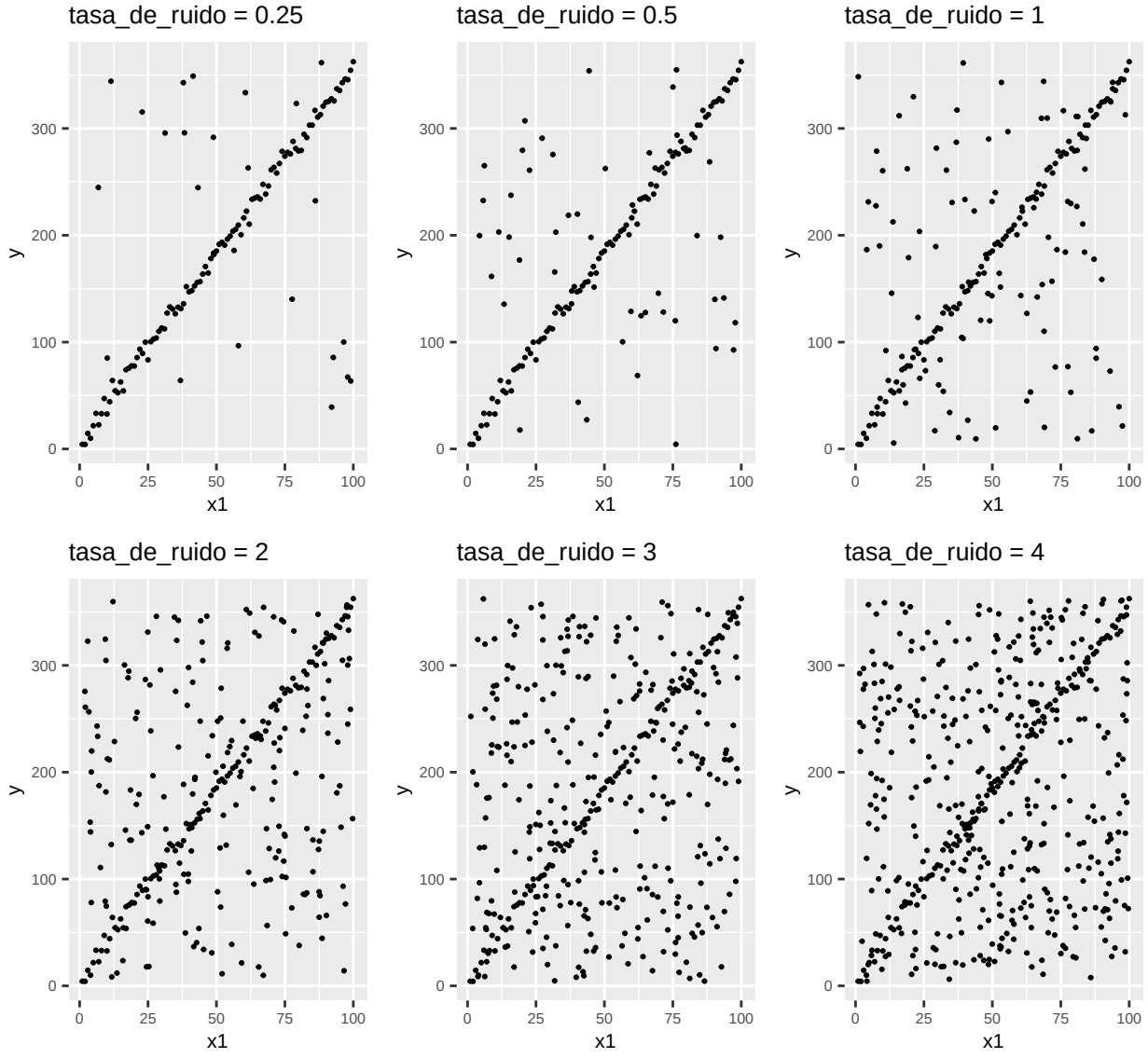
Una ventaja de RANSAC es su capacidad para hacer una estimación robusta de los parámetros del modelo, es decir, estima el modelo a pesar de haber un número significativo de valores atípicos. Una desventaja de RANSAC es que no hay tiempo máximo para calcular los parámetros del modelo, excepto el agotamiento del número de iteraciones. El número de iteraciones puede limitar la solución a una no óptima, y puede encontrar una solución que no se ajuste a los datos.

Ejemplo[1]

Se mostrará un ejemplo extremo, con diverso número de datos atípicos, para ejemplificar las ventajas y desventajas del método. Se hará mediante la incorporación de diversos *niveles de ruido* a los datos.

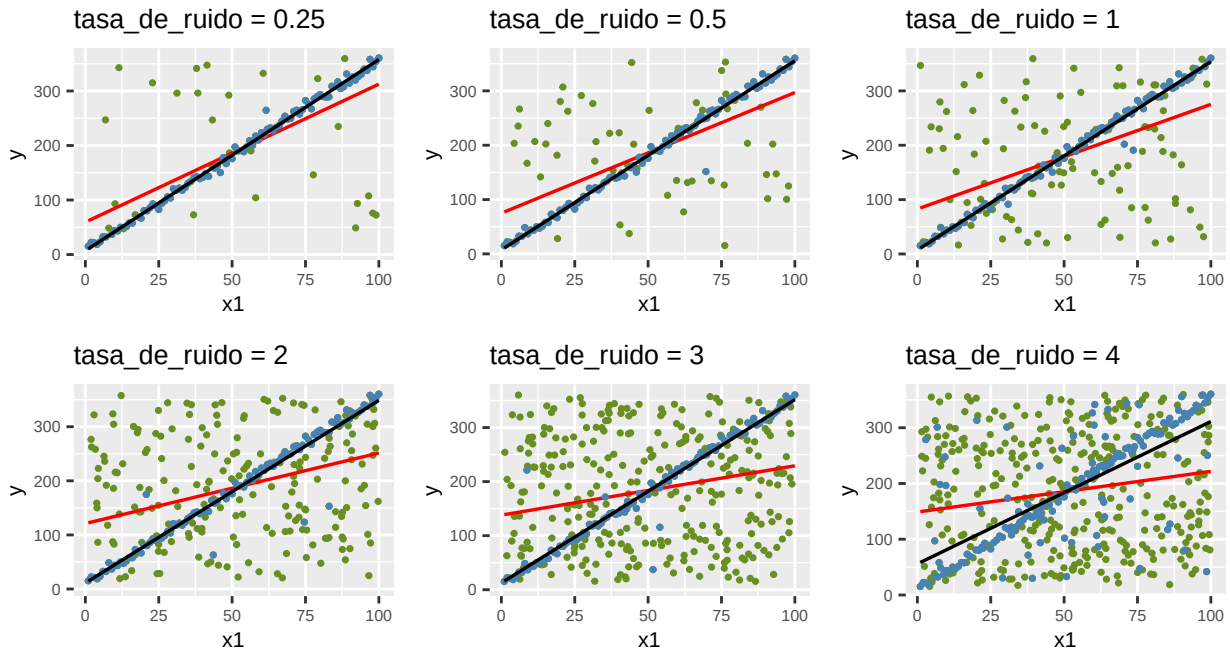
Para el modelamiento, supondremos una relación entre la variable de respuesta y un par de variables explicativas: $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \epsilon$

El siguiente gráfico presenta los datos de la primera variable, junto con el ruido incorporado. Una tasa de ruido = 0.50 implica que hay un 50% de puntos *outliers* adicionales a los *inliers*. Una tasa de ruido = 4 implica que hay cuatro veces más *outliers* que *inliers*. Este ejemplo tan ruidoso no es el usual en aplicaciones donde el interés es encontrar los *outliers*, pero es demostrativo de lo robusto que es el algoritmo.

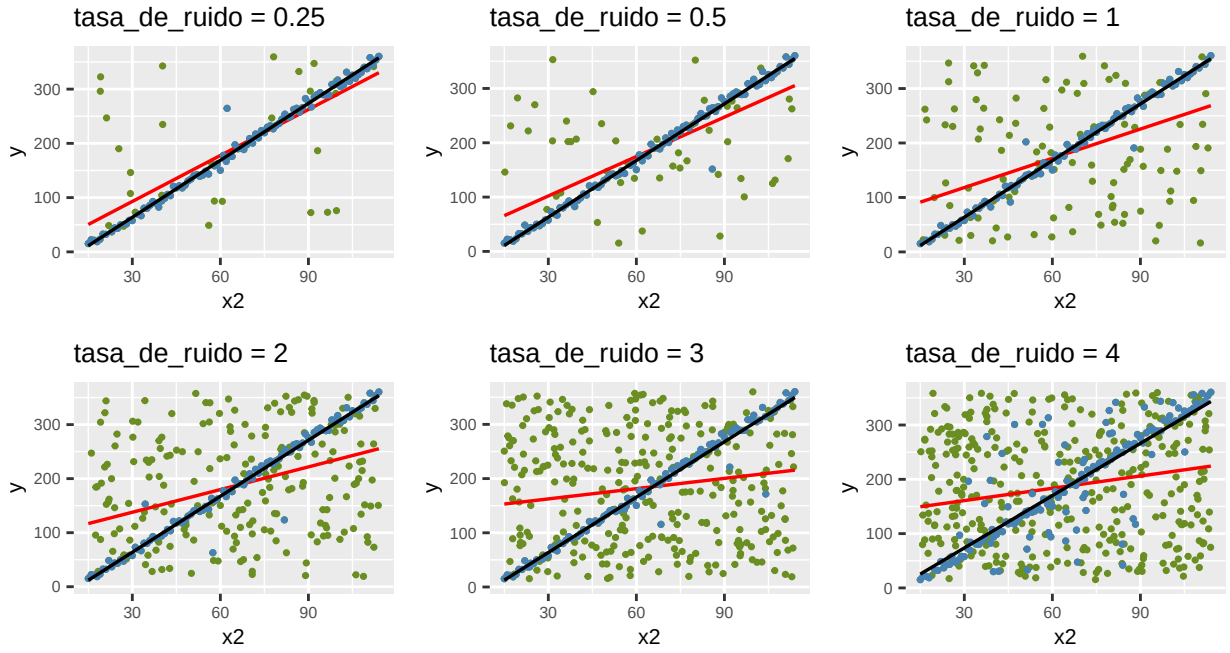


[1] Código tomado de http://www.pjthepooh.com/ransac_in_r.html y ajustado a dos variables independientes.

Este es el resultado sobre la primera variable. Obsérvese que los *outliers* están resaltados en color verde oliva. Tuvo equivocaciones sobre la tasa de ruido más alta.



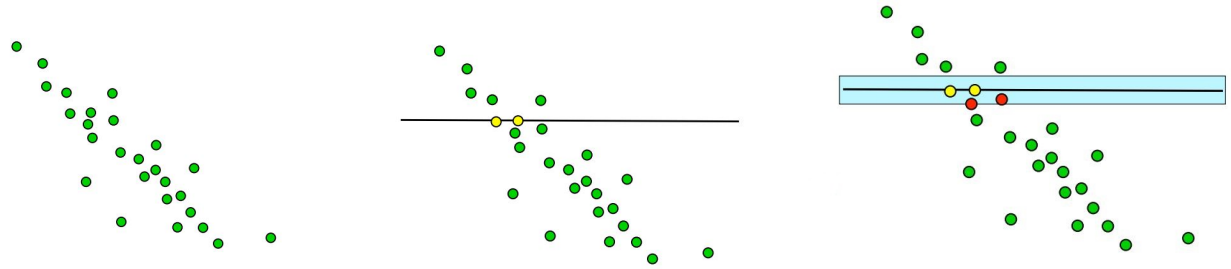
Sobre la segunda variable funcionó bastante bien.



Las equivocaciones pueden tener que ver con la parametrización del modelo. Hay que entender cómo funciona.

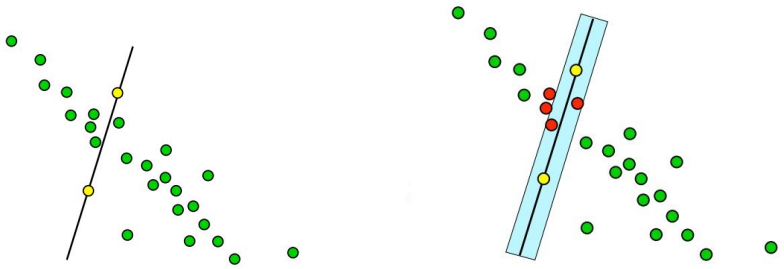
El algoritmo[2]

Una vez se tiene la información selecciona n puntos al azar y establece la regresión lineal. En este esquema de ejemplo n = 2. Cuenta cuántos puntos están dentro del umbral cercano a la regresión lineal.



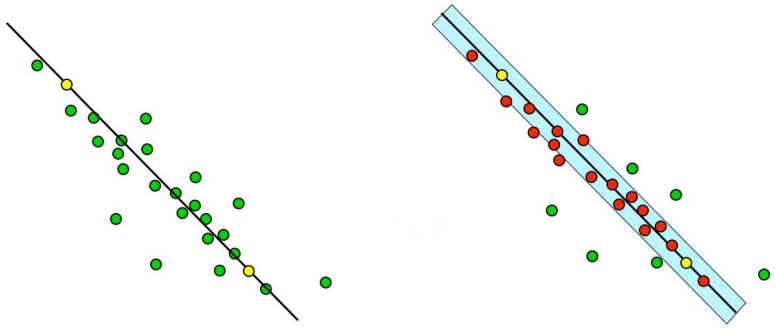
Hay 4 puntos.

Procede a seleccionar otro subconjunto de datos de tamaño n y a establecer otra regresión lineal. Y cuenta cuántos puntos están cercanos al umbral en la nueva regresión.



Hay 6 puntos.

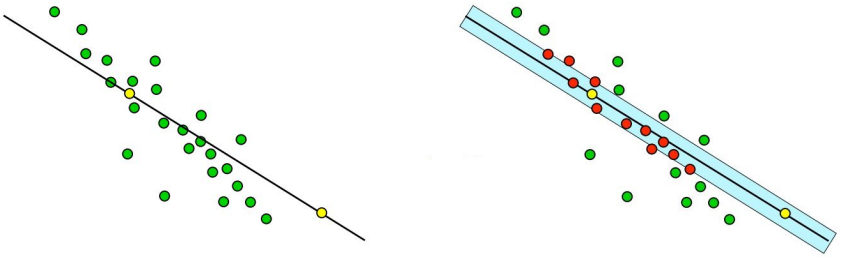
Lo realiza iterativamente.



Hay 19 puntos.

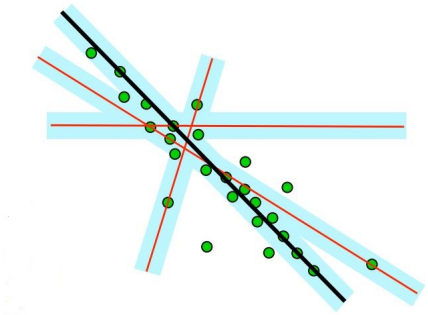
El número de veces que se le haya especificado.

[2]Gráficas y explicación del algoritmo tomadas del curso sobre procesamiento de imágenes de Roberts Collins, Penn State University



Hay 13 puntos.

Evalúa las diferentes opciones y selecciona aquellas que hayan llegado al número de puntos *inliers* esperado, y de éstas selecciona la mejor.



El mejor modelo es el de 19 puntos.

Dados el conjunto de datos y el modelo de regresión lineal, los parámetros del algoritmo son:

```
*n* - Un número mínimo de puntos con los cuales estimar la regresión.
*k* - El número máximo de iteraciones.
*t* - Un umbral que permita determinar que el modelo ajusta bien.
*d* - El número de puntos cercanos requeridos para asegurar que el modelo ajusta bien.
```

El algoritmo es el siguiente:

```
iteracion ← 0;
Mejor_ajuste ← inicia vacío;
Mejor_error ← inicia muy grande;
while iteracion < k do
    pueden_ser_inliers ← selección aleatoria de datos, de tamaño n;
    modelo_posible ← ajusta modelo y selecciona datos que pueden ser inliers;
    inliers_adicionales ← inicia vacío;
    for cada punto que no está en el conjunto pueden_ser_inliers do
        if el punto < umbral t then
            añada el punto al conjunto inliers_adicionales;
        end
    end
    if el número de elementos en inliers_adicionales > d then
        Ha encontrado el modelo;
        Mida qué tan bueno es;
        mejor_modelo_actual ←
          modelo ajustado con los puntos de los conjuntos pueden_ser_inliers e inliers_adicionales;
        este_error ← una medida de cuan bueno es el mejor_modelo_actual;
        if este_error < al Mejor_error then
            Mejor_ajuste ← mejor_modelo_actual;
            Mejor_error ← este_error;
        end
    end
    incremente iteracion;
end
```

Algorithm 1: Algoritmo de Ransac en pseudo código

¿Cómo seleccionar el tamaño del n y del k?

Se deben conocer (o suponer) los siguientes parámetros:

```
*n* - Un número mínimo de puntos con los cuales estimar la regresión.
*k* - El número máximo de iteraciones.
*e* - Probabilidad de que un punto sea un outlier. (Porcentaje esperado de outliers,
        entonces $d = N*(1 - e)$, si N es el número total de puntos).
*p* - Probabilidad deseada de haber llegado a una buena muestra.
```

$1 - e$ probabilidad de obtener un *inlier*.

$(1 - e)^n$ probabilidad de obtener **n** *inliers*.

$1 - (1 - e)^n$ probabilidad de que el conjunto esté contaminado con al menos un *outlier*.

$(1 - (1 - e)^n)^k$ probabilidad de que k muestras estén contaminadas con al menos un *outlier*.

$1 - (1 - (1 - e)^n)^k$ probabilidad de que al menos una muestra de k no esté contaminada con ningún *outlier*.

$1 - (1 - (1 - e)^n)^k = p$

Fijamos p en, por ejemplo, 99% y despejamos k.

$$k = \frac{log(1-p)}{log(1-(1-e)^n)} = \frac{-4.60517}{log(1-(1-e)^n)}$$

La tabla presenta el número de iteraciones que se deben especificar. Las filas de la tabla corresponden al número de puntos con respecto a los cuales se ajusta la regresión lineal y las columnas la proporción esperada de *outliers*.

	0.02	0.04	0.06	0.08	0.1	0.12	0.14	0.16	0.18	0.2	0.3	0.5
2	2	2	3	3	3	4	4	4	5	5	7	17
4	2	3	4	4	5	6	6	7	8	9	17	72
6	3	4	4	5	7	8	9	11	13	16	37	293
8	3	4	5	7	9	11	13	17	21	26	78	1177
10	3	5	6	9	11	15	19	24	32	41	161	4714
12	3	5	8	11	14	19	26	35	48	65	331	18861
14	4	6	9	13	18	26	36	51	72	103	677	75449
16	4	7	10	16	23	34	50	73	108	162	1384	301803
18	4	8	12	19	29	44	68	104	162	254	2826	1207216
20	5	8	14	23	36	58	92	149	242	398	5770	4828869
22	5	9	16	27	45	75	125	212	361	622	11777	19315482
24	5	10	18	32	56	97	170	301	537	973	24036	77261933
26	6	11	21	38	69	126	231	427	800	1522	49055	309047738
28	6	12	24	46	86	163	312	606	1191	2379	100114	1236190957
30	6	14	28	54	107	211	423	859	1772	3718	204315	4944763834

La tabla presenta que usar un n pequeño es mejor que uno grande. Y que el algoritmo es más rápido, por necesitar menor número de iteraciones, en presencia de una proporción pequeña de *outliers*.

Usos

Ransac se ideó para determinar la tendencia real a pesar del ruido. Actualmente es muy popular su uso en el área de procesamiento de imágenes. No es tradicional su uso para identificar outliers, pero en procesos en que sea lícito buscar puntos que no corresponden a un proceso de tendencia lineal (combinación lineal de variables), es una alternativa a explorar.

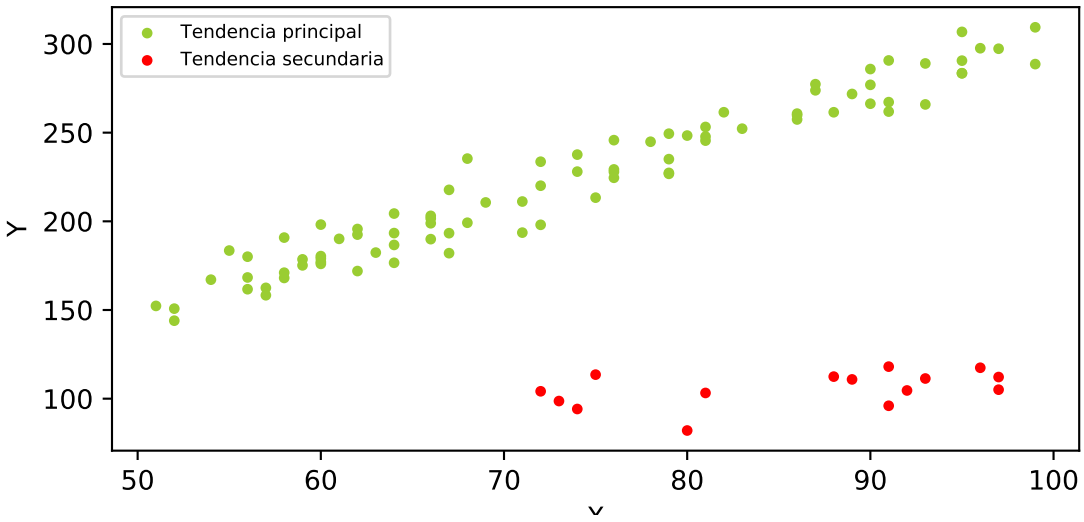
Práctica en python

```
# importar modulos
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# Librería de scikit_learn para regresion lineal
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import RANSACRegressor
from random import seed
```

Sea simulado un conjunto de 100 datos, en los cuales hay dos tendencias.

```
np.random.seed(158)
prin = 85
sec = 100 - prin
x1 = np.random.randint(50, 100, prin)
x2 = np.random.randint(70, 100, sec)
noise = np.random.normal(0, 10, prin)
y1 = 2 + 3 * x1 + noise
noise2 = np.random.normal(0, 10, sec)
y2 = 20 + x2 + noise2
x = np.append(x1, x2)
y = np.append(y1, y2)
```

```
lw = 2
plt.scatter(x1, y1, color='yellowgreen', marker='.',
            label='Tendencia principal')
plt.scatter(x2, y2, color='red', marker='.',
            label='Tendencia secundaria')
plt.legend(loc='upper left', fontsize='x-small')
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```



A continuación se aplica el algoritmo Ransac para separar la tendencia principal y la tendencia secundaria.

```
# Se requiere que el arreglo este en forma de columna
x1 = x.reshape(-1,1)
y1 = y.reshape(-1,1)
lr = LinearRegression()
lr.fit(x1, y1)
# Muestre el intercepto y el coeficiente
```

```
## LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
##          normalize=False)
```

```
lr.intercept_, lr.coef_
```

```
## (array([80.57262198]), array([[1.65194051]]))
```

```
ransac = RANSACRegressor(LinearRegression())
ransac.fit(x1, y1)
# El objeto guarda los inliers
```

```
## RANSACRegressor(base_estimator=LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
##          normalize=False),
##         is_data_valid=None, is_model_valid=None, loss='absolute_loss',
##         max_skips=inf, max_trials=100, min_samples=None, random_state=None,
##         residual_threshold=None, stop_n_inliers=inf, stop_probability=0.99,
##         stop_score=inf)
```

```
inlier_mask = ransac.inlier_mask_
outlier_mask = np.logical_not(inlier_mask)
# Variables que se pueden usar luego si se desea
outliers_x = x[outlier_mask]
outliers_y = y[outlier_mask]
```

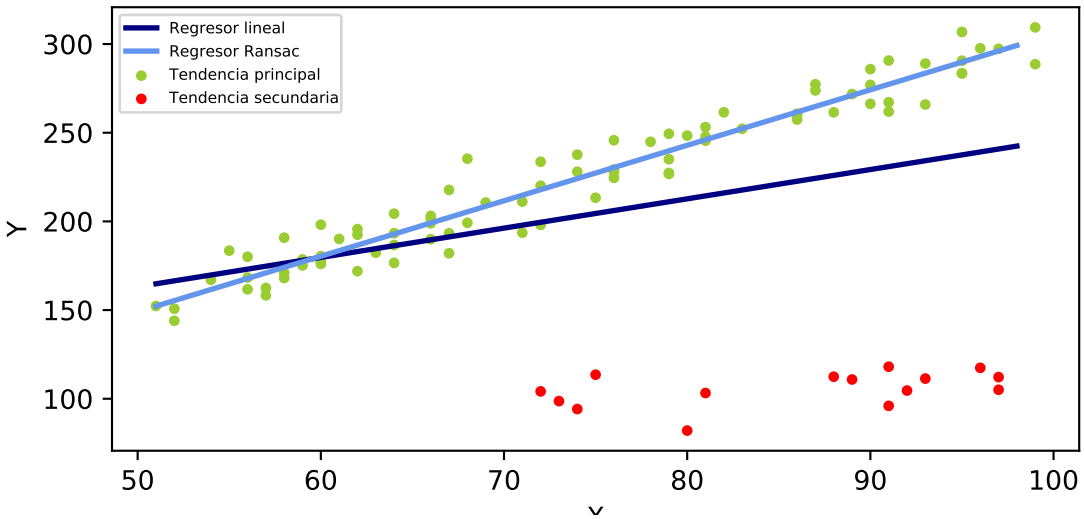
Obsérvese que el algoritmo elige internamente los parámetros con los cuales ejecutarse.

```
line_x = np.arange(x1.min(), x1.max())[:, np.newaxis]
line_y = lr.predict(line_x)
line_y_ransac = ransac.predict(line_x)
# Muestre el intercepto y el coeficiente
ransac.estimator_.intercept_, ransac.estimator_.coef_
```

```
## (array([-7.29538489]), array([[3.12758957]]))
```

Mediante la librería matplotlib se presenta el resultado.

```
lw = 2
plt.scatter(x[inlier_mask], y[inlier_mask], color='yellowgreen', marker='.',
            label='Tendencia principal')
plt.scatter(x[outlier_mask], y[outlier_mask], color='red', marker='.',
            label='Tendencia secundaria')
plt.plot(line_x, line_y, color='navy', linewidth=lw, label='Regresor lineal')
plt.plot(line_x, line_y_ransac, color='cornflowerblue', linewidth=lw,
         label='Regresor Ransac')
plt.legend(loc='upper left', fontsize='xx-small')
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```



Ejercicio

Identifique los outliers usando la base de datos de medidas del cuerpo.

```
datos = pd.read_csv("medidas_cuerpo2.csv")
# Se selecciona y y X.
# Por convención, para indicar que X es una matriz, se coloca en mayúsculas
X = datos[['Estatura', 'circun_cuello', 'circun_muneca']]
y = datos[['Peso']]
# Se estima el modelo de regresion lineal múltiple
lr = LinearRegression(normalize=True).fit(X, y)
lr.intercept_, lr.coef_
```

```
## (array([-62.01829679]), array([[9.44039465, 2.25873513, 1.98908223]]))
```

```
# Se estima el modelo RANSAC
ransac = RANSACRegressor(lr)
ransac.fit(X, y)
```

```
## RANSACRegressor(base_estimator=LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normali
##         is_data_valid=None, is_model_valid=None, loss='absolute_loss',
##         max_skips=inf, max_trials=100, min_samples=None, random_state=None,
##         residual_threshold=None, stop_n_inliers=inf, stop_probability=0.99,
##         stop_score=inf)
```

```
ransac.estimator_.intercept_, ransac.estimator_.coef_
```

```
## (array([-33.57956849]), array([[ 8.47306767,  3.02363927, -1.44216947]]))
```

```
inlier_mask = ransac.inlier_mask_
outlier_mask = np.logical_not(inlier_mask)
# Variables que se pueden usar luego si se desea
outliers_x = X[outlier_mask]
outliers_y = y[outlier_mask]
```

Este es el resultado:

```
outliers_x
```

```
##     Estatura  circun_cuello  circun_muneca
## 1       1.66           38.4           16.0
## 3       1.76           38.0           17.1
## 7       1.78           40.5           16.5
## 10      1.79           41.5           17.1
## 11      1.49           31.5           13.8
## 12      1.74           38.0           16.4
## 16      1.82           38.0           18.0
```