1. Introducción

El concepto de modelo es un elemento fundamental en la representación de problemas y en el aprendizaje automático (*machine learning*).

En la RAE, entre otras definiciones, se encuentren:

♢ Arquetipo o punto de referencia para imitarlo o reproducirlo.

♢ Esquema teórico, generalmente en forma matemática, de un sistema o de una realidad compleja, como la evolución económica de un país, que se elabora para facilitar su comprensión y el estudio de su comportamiento.

Algoritmo

En *aprendizaje automático* es un proceso que utiliza datos para generar un modelo. Es decir, los algoritmos "*aprenden*" de los datos proporcionados.

Modelo

El modelo representa lo aprendido por el algoritmo de *machine learning*.

2. Aprendizaje no supervisado

En el *aprendizaje no supervisado* el proceso se realiza de manera automática, sin la necesidad de ningún supervisor externo (etiquetas de clase). Para ello se emplean técnicas de agrupamiento, reducción de dimensiones y descubrimiento de patrones, gracias a las cuales el sistema selecciona y aprende los objetos que poseen características similares, determinando automáticamente las clases, características y relaciones importantes.

2.1. Detección de anomalías y valores atípicos

Detección de anomalías (*outliers*) es la identificación de eventos u observaciones que no son acordes a los patrones esperados en cierto conjunto de datos. Se utiliza comúnmente para eliminar valores anómalos en conjuntos de datos, buscando mejorar el rendimiento de modelos de ML.

Detectar anomalías es muy útil cuando se está realizando el preprocesamiento de datos: puede ayudar a evitar sesgos no deseados.

Además, puede resultar útil en múltiples escenarios:

♢ Medicina

- Monitoreo de salud en vuelos espaciales

♢ Seguridad en aeropuertos

♢ Detección de ataques en redes de computadoras

♢ Detección de fraudes

♢ Actividad inusual en redes móviles, etc.

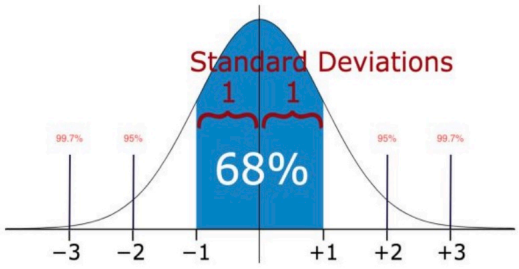
2.1.1. Desviación estándar

Es una forma simple de detectar valores anómalos en un conjunto de datos. Si los datos tienen una distribución normal:

♢ 68 % de los mismos se encuentran a una desviación estándar de la media,

♢ 95 % estarán a dos desviaciones y,

♢ 99.7 %, a tres desviaciones.



2.1.1.1. Normalidad de la distribución Para verificar la normalidad de una variable aleatoria es posible utilizar alguna de las siguientes técnicas:

♢ realizar un histograma y ver si *parece normal*

♢ hacer una gráfica *Quantile-Quantile Normal*

♢ realizar un *contraste de normalidad* Shapiro-Wilks

♢ hacer un diagrama de dispersión por pares de variables (*PairPlot*)

Ejemplos con Python

Importando bibliotecas útiles:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Datos:

```
cont = pd.read_csv('https://bit.ly/31B56KB')
cont.info()
```

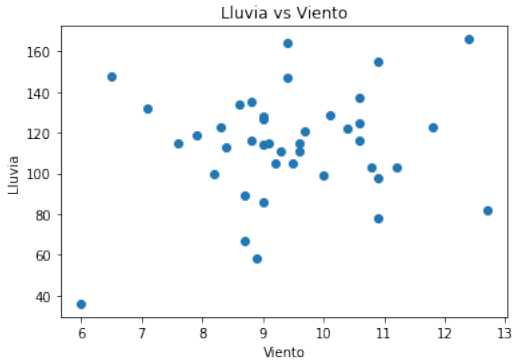
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41 entries, 0 to 40
Data columns (total 7 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Contaminacion_SO2   41 non-null     int64
 1   Temperatura         41 non-null     float64
 2   Fabricas            41 non-null     int64
 3   Habitantes          41 non-null     int64
 4   Velocidad_viento    41 non-null     float64
 5   Lluvia              41 non-null     float64
 6   Dias_Lluvia         41 non-null     int64
dtypes: float64(3), int64(4)
memory usage: 2.4 KB
```

```
cont.head()
```

	Contaminacion_SO2	Temperatura	Fabricas	Habitantes
0	10	70.3	213	582	6.0
1	13	61.0	91	132	8.2
2	12	56.7	453	716	8.7
3	17	51.9	454	515	9.0
4	56	49.1	412	158	9.0

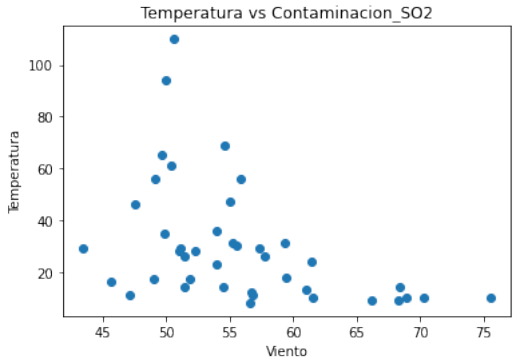
Se puede probar cómo se comportan pares de variables:

```
plt.scatter(cont.Velocidad_viento, cont.Dias_Lluvia)
plt.title('Lluvia vs Viento')
plt.xlabel('Viento')
plt.ylabel('Lluvia')
plt.show()
```



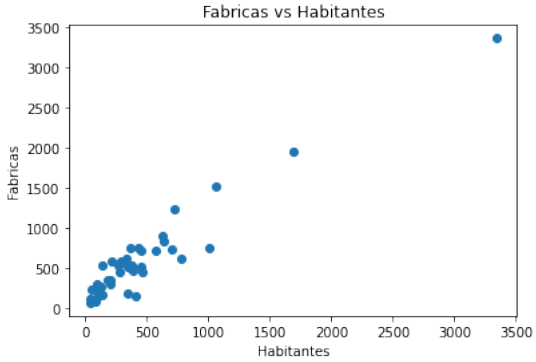
No parece mostrar evidencia de correlación lineal.

```
plt.scatter(cont.Temperatura, cont.Contaminacion_SO2)
plt.title('Temperatura vs Contaminacion_SO2')
plt.xlabel('Viento')
plt.ylabel('Temperatura')
plt.show()
```



Parece existir correlación.

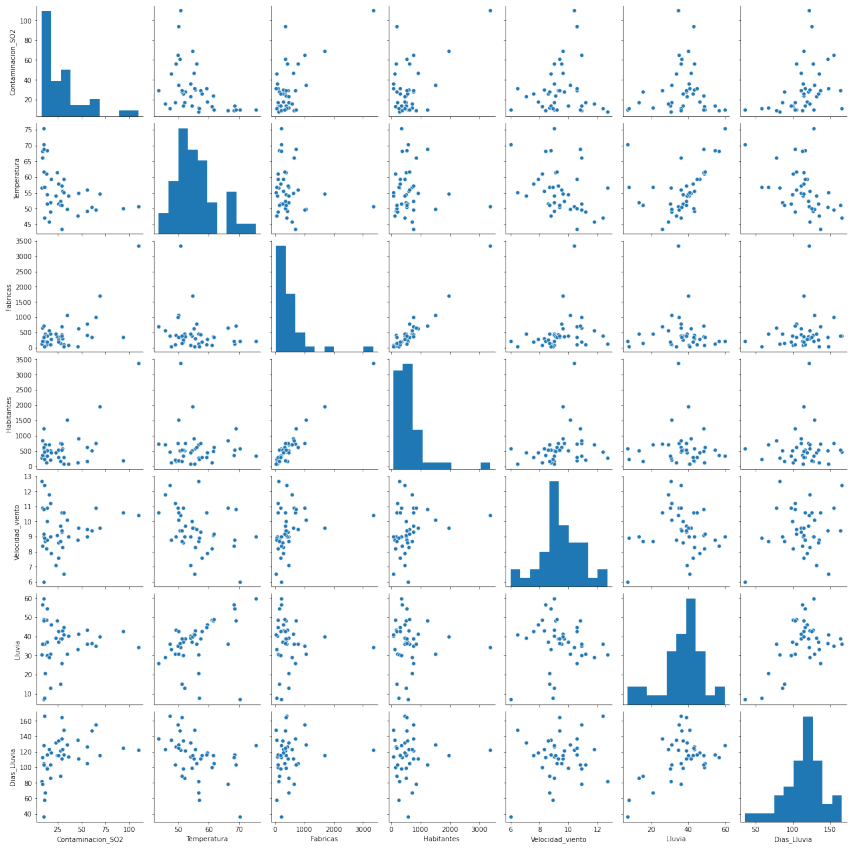
```
plt.scatter(cont.Fabricas, cont.Habitantes) # tiene 'outliers'
plt.title('Fabricas vs Habitantes')
plt.xlabel('Habitantes')
plt.ylabel('Fabricas')
plt.show()
```



Sin lugar a dudas existe correlación entre este par de variables, y contiene valores atípicos.

Para ver todas las variables juntos, podemos realizar una diagrama de dispersión por pares de variables:

```
sns.pairplot(cont)
```



En la diagonal, este diagrama muestra el histograma de cada variable, con esto podemos ver si tiene distribución normal.

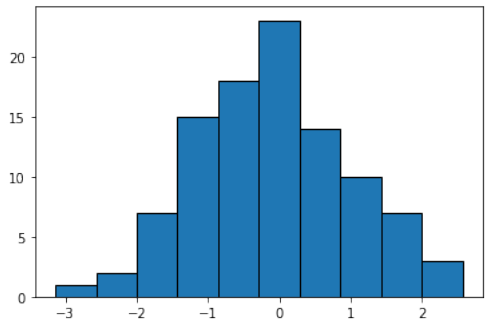
Prueba de normalidad

Para verificar si una variable tiene distribución normal, primero mostremos el resultado con una ficticia creada para este propçosito:

```
import numpy as np
data = np.random.normal(0,1,100)
```

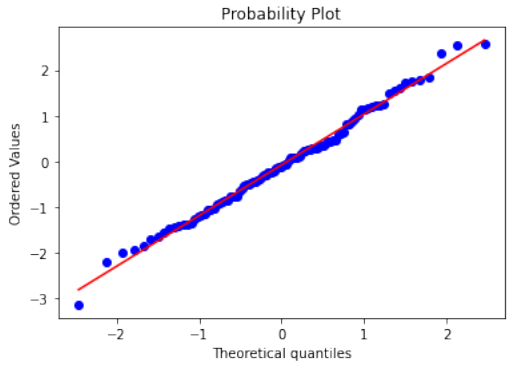
Veamos su histograma:

```
plt.hist(data, edgecolor='black', linewidth=1)
```



Y usemos una gráfica *Quantile-Quantile Normal* con *stats.probplot*:

```
import pylab
import scipy.stats as stats
stats.probplot(data, dist='norm', plot=pylab)
pylab.show()
```



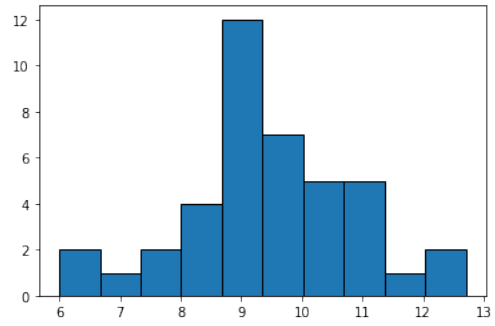
Esta gráfica indica que la distribución es normal si los puntos está distribuídos cerca de la recta. Además, podemos aplicar un contraste de normalidad con Shapiro:

```
from scipy.stats import shapiro
estad, p_value = shapiro(data)
print("Estadístico = %.3f, p_value=%.3f" % (estad, p_value))
# p_value > 0.05 => distribución normal
```

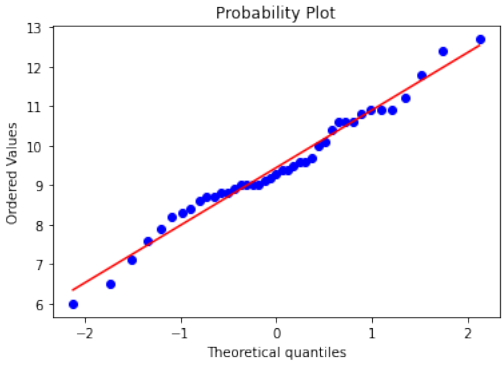
```
Estadístico = 0.993, p_value=0.887
```

Como el *p_ value* es mayor a 0.05, esta prueba indica que la variable tiene distribución normal. Regresando al conjunto de datos de contaminación:

```
plt.hist(cont.Velocidad_viento, edgecolor='black', linewidth=1)
```




```
import pylab
import scipy.stats as stats
stats.probplot(cont.Velocidad_viento, dist='norm', plot=pylab)
pylab.show()
```

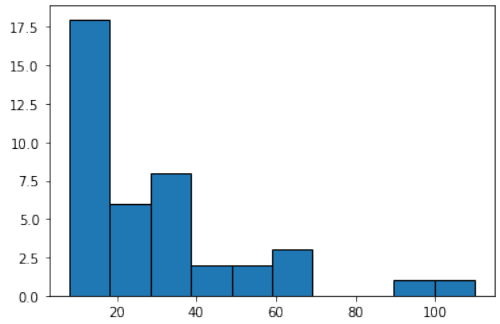


```
from scipy.stats import shapiro
estad, p_value = shapiro(cont.Velocidad_viento)
print("Estadístico = %.3f, p_value=%.3f" % (estad, p_value))
```

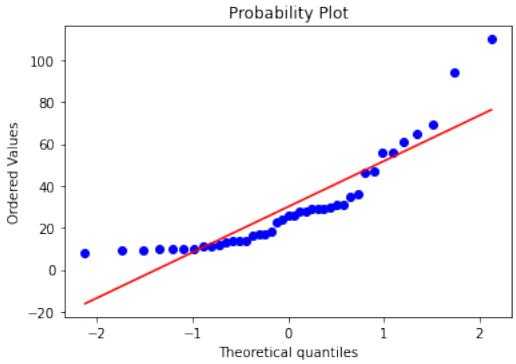
```
Estadístico = 0.981, p_value=0.697
```

Con estos resultados podemos concluir que la variable *Velocidad_viento* tiene distribución normal.

```
plt.hist(cont.Contaminacion_SO2, edgecolor='black', linewidth=1)
```




```
import pylab
import scipy.stats as stats
stats.probplot(cont.Contaminacion_SO2, dist='norm', plot=pylab)
pylab.show()
```



```
from scipy.stats
import shapiro estad, p_value = shapiro(cont.Contaminacion_SO2)
print("Estadístico = %.3f, p_value=%.3f" % (estad, p_value))
```

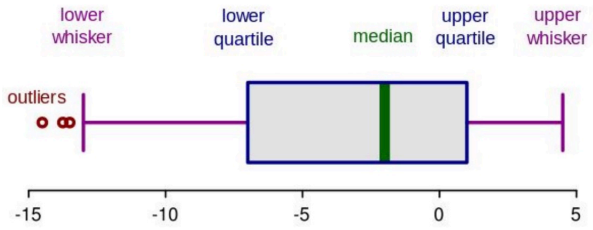
```
Estadístico = 0.812, p_value=0.000
```

Con estos resultados podemos concluir que la variable *Contaminacion_SO2* **no** tiene distribución normal.

2.1.2. Diagrama de *bigotes* (*boxplot*)

Un boxplot es una representación gráfica de datos números con sus cuartiles. Es una forma simple y muy efectiva de encontrar anomalías.

Los bigotes (*whiskers*) marcan los límites inferior y superior de la distribución de datos: cualquier dato fuera de esos límites se puede considerar anómalo.



Para construirlo se utiliza el concepto de rango intercuartil (IQR), una medida de dispersión que divide un conjunto de datos en cuatro subgrupos.

Dados n datos:

- ♢ $IQR = Q3 - Q1$
- ♢ $Q1$ =mediana de los $n/2$ datos menores
- ♢ $Q2$= mediana de todos los datos
- ♢ $Q3$= mediana de los $n/2$ datos mayores
- ♢ $low = Q1 - 1.5 * IQR$
- ♢ $upp = Q3 + 1.5 * IQR$

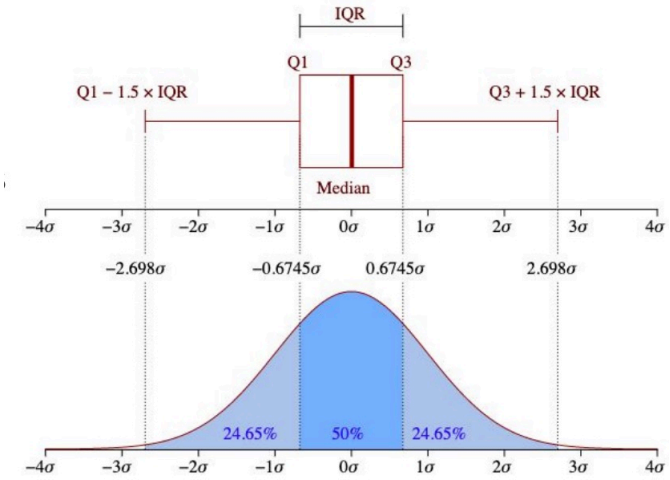


Imagen de: De Jhguch at en.wikipedia, CC BY-SA 2.5, https://commons.wikimedia.org/w/index.php?curid=14
Existen muchas otras formas de detección de valores atípicos, por ejemplo:

♢ Bosques de aislamiento

♢ Redes neuronales profundas

2.2. Análisis de agrupamiento (*clustering*)

En ocasiones es posible dividir una colección de observaciones en distintos subgrupos, basados solamente en los atributos de las observaciones; cuando se puede hacer esta separación, se facilita el entendimiento de la población o el proceso que generó las observaciones. La intención es realizar división de datos en *grupos* (*clusters*) de observaciones que son más similares dentro de un grupo que entre varios grupos. Los grupos son formados ya sea agregando observaciones o dividiendo un gran grupo de observaciones en una colección de grupos más pequeños.

El proceso de generación de grupos incluye dos variedades de algoritmos:

1. Combinar observaciones entre un número fijo de grupos buscando maximizar la similitud dentro de cada grupo

2. Comenzar con grupos de un solo elemento y, recursivamente combinarlos; alternativamente, se puede comenzar con un sólo grupo y recursivamente dividir nuevos grupos

En esta sección se revisarán dos algoritmos populares y representativos de cada una de estas propuestas: agrupación jerárquica y k-medias. Sea $D = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ el conjunto de datos, con $\mathbf{x}_i$ un vector de atributos medidos en una unidad observacional.

2.2.1. Agrupamiento por $k-$medias

A diferencia del agrupamiento jerárquico por aglomeración, en este algoritmo, el analista establece el número de grupos; el algoritmo comienza asignando arbitrariamente los vectores del conjunto de datos $D = \{x_1, \ldots, x_n\}$ a k grupos; estos grupos iniciales se denotan como $\{A_1, \ldots, A_k\}$. Después, se calculan los *centroides* de cada grupo; estos centroides son medias multivariadas de las observaciones de cada grupo y cada centroide representa al grupo para el cálculo de las distancias. También es posible asignar arbitrariamente k medias iniciales y con ellas generar los grupos.

Es muy poco probable que la configuración inicial sea una buena solución, por tanto el algoritmo itera entre dos pasos:

♢ asignar cada observación al grupo más cercano

♢ recalcular los centroides de cada grupo

Si al menos una observación es reasignada en otro grupo, los centroides cambiarán y debe realizarse otra iteración; el algoritmo continuará pasando entre estos dos pasos hasta que ya no hay reasignaciones. En ese momento, cada observación pertenece al grupo que le es más cercano. A partir de la configuración aleatoria inicial, la suma de cuadrados dentro de cada grupo ha sido minimizado; esto se debe a que dicha suma de cuadrados es equivalente a la suma de distancias euclideanas entre las observaciones y los centroides; además, mover cualquiera de las obervaciones incrementará la suma de distancias (y la suma de cuadrados dentro de los grupos). Por tanto, el algoritmo ha alcanzado *una mejor* asignación posible para las observaciones en los grupos y *un mejor* cálculo de los centroides.

Este algoritmo tiene un atractivo considerable porque minimza una función objetivo popular: la suma de cuadrados. Su inconveniente es que debe generar una configuración inicial aleatoria: una configuración diferente por lo general regresa un resultado distinto.

El centroide del grupo A_i es la media multivariada:

$$\overline{\mathbf{x}}_i = n_i^{-1} \sum_{\mathbf{x}_j \in A_i} \mathbf{x}_j,$$

donde, n_i es el número de observaciones que pertenecen a A_i y $\mathbf{x}_j = [x_{j,1}, \ldots, x_{j,h}]^T$; el número de atributos es h y el $l-$ésimo elemento de $\overline{\mathbf{x}}_i$ es:

$$\overline{x}_{i,l} = n_i^{-1} \sum_{\mathbf{x}_j \in A_i} x_{j,l},$$

para $l = 1, \ldots, h$; la distancia entre cada observación $\mathbf{x}_j$ y un grupo A_i, se define como la distancia entre la observación y el centroide del grupo. Como $\overline{\mathbf{x}}_i = [\overline{x}_{i,1}, \ldots, \overline{x}_{i,h}]^T$, la distancia euclideana cuadrada es:

$$d_E^2 (\mathbf{x}_j, \overline{\mathbf{x}}_i) = \sum_{l=1}^{h} (x_{j,l} - \overline{x}_{i,l})^2.$$

Se puede utilizar la distancia cuadrada en lugar de la distancia, dado que el ordenamiento de más cercano a más distante será el mismo.

Ejemplo

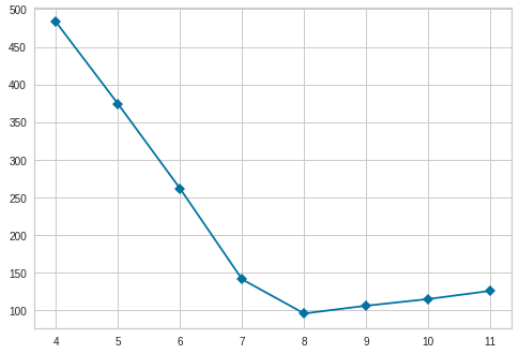
Clasificar las muestras siguientes utilizando $k = 2$:
$[8, 10], [3, 10.5], [7, 13.5], [5, 18], [5, 13], [6, 9], [9, 11], [3, 18], [8.5, 12], [8, 16]$

Ejemplo con Python

El conjunto de datos *xclara* es muy usado para probar este tipo de algoritmos

2.2.1.1. Determinar un buen valor para *k* Al igual que en la mayoría de los modelos, determinar los hiperparámetros requiere iterar el mismo y encontrar un indicador que nos permita elegir su valor.

Para el caso de $k-$medias, la *gráfica de codo* brinda una buena guía para este fin.



Este método recomienda usar como valor para k aquel que corresponde con el codo, es decir, en el que se tiene el mayor cambio en la pendiente,

Ejemplo en 3D

Tarea:

♢ Clasificar las muestras siguientes utilizando k=*2*:
$[1, 12.5], [3, 10.5], [3, 12.5], [3, 14.5], [3, 18], [5, 18], [5, 16], [5, 14.5], [5, 13], [6, 9], [8, 10], [9, 11], [8.5, 12], [7, 13.5], [8, 16], [0.5, 10.5]$

*

2.2.2. $k-$*modas*

Es un algoritmo de agrupamiento con una idea similar al aplicado en que se utiliza en $k-$medias, pero se aplica a conjuntos de datos que son mayoritariamente categóricos (como los usados en encuestas).

Sustuye al uso de $k-$medias porque este último algoritmo calcula distancias sobre datos *continuos;* para datos categóricos no es posible obtener esa métrica de distacia. En cambio, $k-$*modas* utiliza las diferencias (errores totales) entre las muestras del conjunto; entre menos diferencias existan los puntos de cada grupo serán más similares.

Al igual que en el caso de $k-$medias, el algoritmo comienza con la elección de k observaciones iniciales como representantes de cada grupo y posteriormente itera entre estos dos pasos mientras no haya convergencia:

♢ Asignar cada observación al grupo más *cercano*, es decir, aquella moda con la que tenga menos diferencias

♢ Recalcular los modas de cada grupo

Ejemplo

Clasificar las muestras siguientes utilizando $k = 2$:
$[x', y', z'], [y', z', x'], [z', x', x'], [y', z', z'], [x', z', y'], [z', y', x'], [x', x', y'], [z', y', x']$

Ejemplo con Python

Importando bibliotecas:

```
!pip install kmodes
```

2.2.2.1. Determinar un buen valor para *k* Al igual que en modelos anteriores, determinar los hiperparámetros requiere iterar el mismo y encontrar un indicador que nos permita elegir su valor.

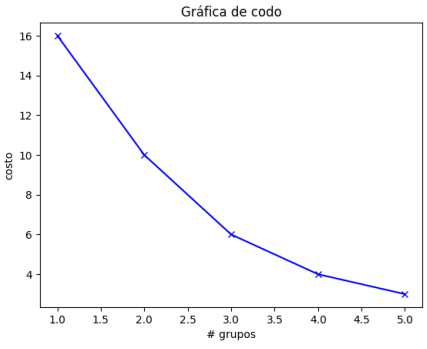
Para el caso de $k-$modas, también la *gráfica de codo* brinda una buena guía para este fin.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from kmodes.kmodes import KModes
```

```
col_cabello = np.array(['rubio', 'castaño', 'pelirrojo', 'negro', 'castaño', 'n
col_ojos = np.array(['azul', 'gris', 'verde', 'café', 'azul', 'gris', 'azul', '
tipo_cabello = np.array(['lacio', 'chino', 'ondulado', 'ondulado', 'chino', 'ch
personas = ['P1','P2','P3','P4','P5','P6','P7','P8']
data = pd.DataFrame({'person':personas, 'col_cabello':col_cabello, 'col_ojos':c
data = data.set_index('person') data
```

```
cost = []
K = range(1,5)
for num_clusters in list(K):
  kmode = KModes(n_clusters=num_clusters,init="random",n_init=5,verbose=False)
  kmode.fit_predict(data)
  cost.append(kmode.cost_)

plt.plot(K, cost, 'bx-')
plt.xlabel('# grupos')
plt.ylabel('costo')
plt.title(Gráfica de codo')
plt.show()
```



Tarea:

♢ Agrega al conjunto de datos una columna continua (como estatura y/o peso) e ingresa los valores correspondientes. Después genera rangos para obtener categorías y obtén los nuevos grupos generados con esa información adicional.

*

2.2.3. Agrupamiento jerárquico por aglomeración

Este enfoque comienza con cada observación definiendo un grupo de un sólo elemento: $\{\mathbf{x}_1\}, \ldots, \{\mathbf{x}_n\}$, donde n es el número de observaciones. El algoritmo reduce iterativamente el conjunto de grupos combinando grupos similares. La combinación de los conjuntos A y B en alguna iteración se representa como:

$$(A, B) \longrightarrow A \cup B$$

La elección de conjuntos a combinar se puede determinar calculando la distancia entre los grupos y combinando la pareja con la menos distancia intergrupal (más adelante se presentan otras opciones). Por tanto, es requiere una métrica para obtener la distancia entre grupos; por ejemplo, la distancia entre los grupos A y B puede definirse como la distancia más corta entre cualquier vector de A y cualquier vector de B. Matemáticamente:

$$d(A, B) = min\{d_v(\mathbf{x}_k, \mathbf{x}_l) | \mathbf{x}_k \in A, \mathbf{x}_l \in B\},$$

donde d_v es una distancia entre vectores. La distancia euclideana es muy popular para este cálculo. También se puede utilizar la distancia Manhattan (*city-block*), la distancia *city-block* entre $\mathbf{x}_0$ y cada $\mathbf{x}_i \in X$ se define como:

$$d_C(\mathbf{x}_i, \mathbf{x}_0) = \sum_{j=1}^{p} |x_{i,j} - x_{0,j}|,$$

donde p es el número de elementos de los vectores. Se pueden obtener grupos más compactos utilizando la métrica basada en centroides, éste se obtiene para cada grupo como:

$$\overline{\mathbf{x}}_A = n_A^{-1} \sum_{\mathbf{x}_k \in A} x_k,$$

donde n_A es el número de observaciones en el grupo A; entonces la distancia entre A y B es:

$$d_{cent}(A, B) = d(\overline{\mathbf{x}}_A, \overline{\mathbf{x}}_B)$$

Como se describió anteriormente, el algoritmo combinará grupos hasta obtener uno sólo; pero es necesario revisar conjuntos de grupos intermedios para identificar el que resulte más útil según sea el caso. Para poder identificar el agrupamiento óptimo, se puede utilizar un *dendrograma*, el cual muestra el historial de los agrupamientos, se realiza lo siguiente:

1. Determinar la mayor distancia vertical para la que no hay intersección con otros grupos
2. Trazar una línea horizontal entre ambos extremos
3. El agrupamiento óptimo es igual al número de líneas verticales cruzadas por el trazo anterior

Para el ejemplo de la figura 1 la mejor elección sería 4.

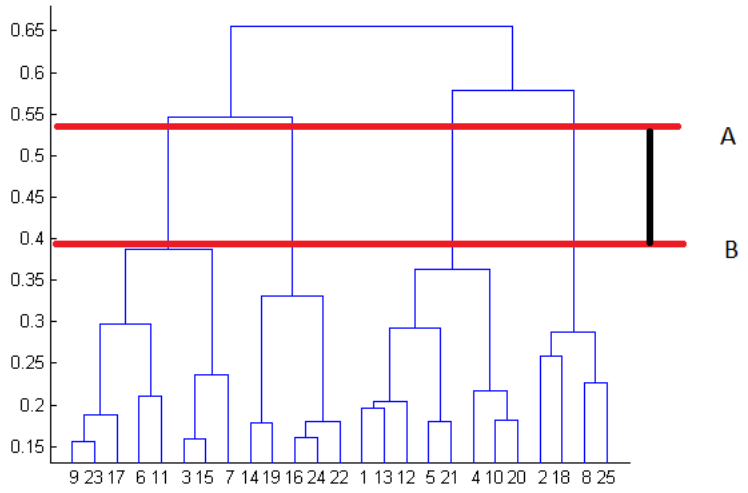


Figura 1: Ejemplo de uso de dendrogramas

Criterio de enlazamiento de grupos

Existen otros tipos de enlazamiento entre los grupos, cada uno entrega soluciones distintas.

♢ ***Single linkage***: utiliza la distancia más corta entre dos vectores de cada grupo
$L(A,B) = min\{d_v(\mathbf{x}_k, \mathbf{x}_l) | \mathbf{x}_k \in A, \mathbf{x}_l \in B\}$

♢ ***Complete linkage***: toma en cuenta la distancia más grande entre dos vectores de cada grupo
$L(A,B) = max\{d_v(\mathbf{x}_k, \mathbf{x}_l) | \mathbf{x}_k \in A, \mathbf{x}_l \in B\}$

♢ ***Average linkage***: usa la distancia promedio entre todos los vectores de cada grupo

♢ $L(A,B) = \frac{1}{n_A n_B} \sum_{k=1}^{n_A} \sum_{l=1}^{n_B} d_v(\mathbf{x}_k, \mathbf{x}_l), \mathbf{x}_k \in A, \mathbf{x}_l \in B$

♢ ***Ward linkage***: la distancia es la suma de las diferencias al cuadrado en los grupos

♢ $d_v(A,B) = (\overline{\mathbf{x}}_A - \mathbf{x}_k)^2 + (\overline{\mathbf{x}}_B - \mathbf{x}_l)^2, \mathbf{x}_k \in A, \mathbf{x}_l \in B$

Ejemplo

Para el conjunto de datos unidimensionales $\{7, 10, 20, 28, 35\}$, obtener el agrupamiento jerárquico y dibujar el dendrograma correspondiente.

Ejemplo con Python

Otro ejemplo con Python

Los grupos obtenidos:

```
plt.scatter(X[y_hc == 0,0],X[y_hc == 0,1],s=100,c='red',label='Cluster 1')
plt.scatter(X[y_hc == 1,0],X[y_hc == 1,1],s=100,c='blue',label='Cluster 2')
plt.scatter(X[y_hc == 2,0],X[y_hc == 2,1],s=100,c='green',label='Cluster 3')
plt.scatter(X[y_hc == 3,0],X[y_hc == 3,1],s=100,c='cyan',label='Cluster 4')
plt.scatter(X[y_hc == 4,0],X[y_hc == 4,1],s=100,c='magenta',label='Cluster 5')
plt.title('Grupos de clientes')
plt.xlabel('Ingreso anual (k$)')
plt.ylabel('Nivel de consumo (1-100)')
plt.legend()
plt.show()
```

2.2.4. Modelos de mezclas gausianas (*Gaussian Mixed Models*)

Similar a k-medias, en estos modelos se establece una cantidad k de grupos para iniciar y se inicializan tanto la medias como las varianzas.

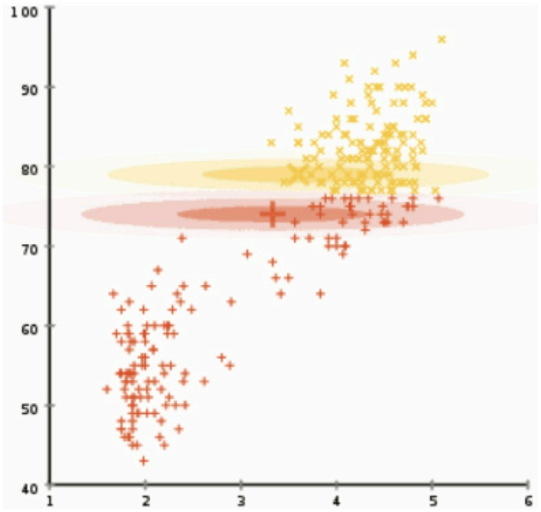


Figura 2: Paso inicial de un modelo de mezclas gaussianas

Iterativamente actualiza la media y la varianza de cada grupo, así como la probabilidad de pertenencia de cada punto a cada grupo y el peso del mismo (cantidad de muestras de cada grupo).

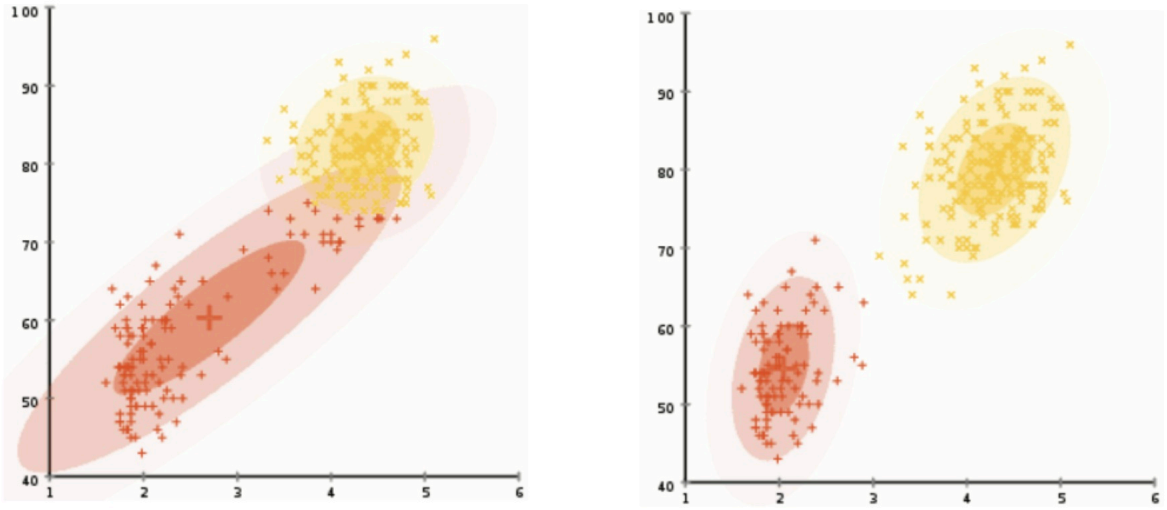


Figura 3: Evolución de un modelo de mezclas gaussianas

Esperanza-Maximización (*Expectation-Maximization*), es la técnica más popular para determinar los parámetros de un modelo de mezclas gaussianas. Se compone de dos pasos:

♢ Paso E: asignar de forma probabilística cada muestra a cada clase, basándose en la hipótesis actual de los parámetros.

♢ Paso M: actualizar las hipótesis de los parámetros, en función de las asignaciones actuales.

En el paso E se calcula el valor esperado de las asignaciones de grupos. En el paso M se obtiene la nueva máxima verosimilitud de las hipótesis.

Modelos de mezclas gaussianas vs k-medias

Si bien el algoritmo inicia con un número predefinido de grupos al igual que k-means, existen diferencias fundamentales:

♢ k-medias establece un círculo (hiperesfera) al centro del grupo, con radio definido por el punto más distante

♢ GMM funciona mejor cuando los datos no se ajustan a circunferencias (hiperesferas)

♢ k-medias realiza clasificación dura: devuelve la clase a la que pertenece una muestra, mientras GMM hace clasificación suave: devuelve la probabilidad de pertenencia a cada clase.

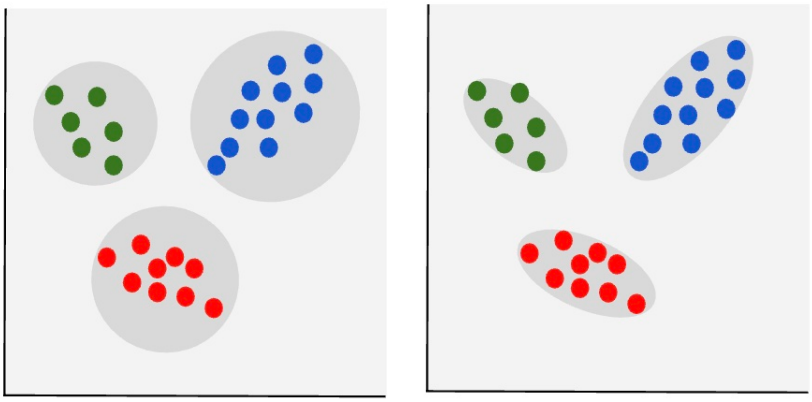


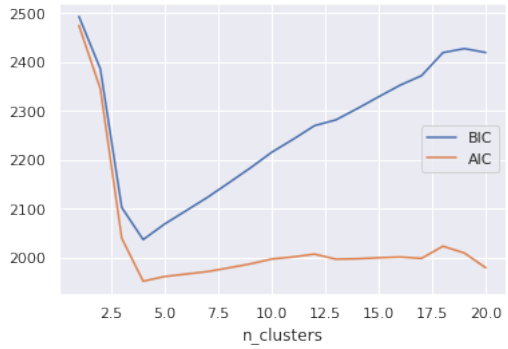
Figura 4: k-medias vs GMM

Ejemplo:

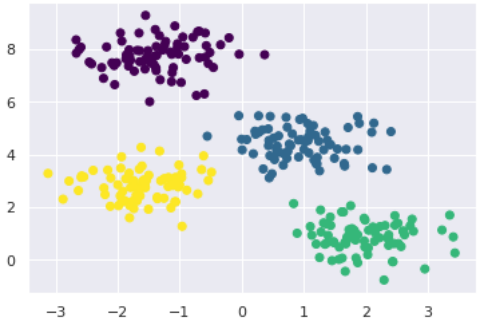
Antes de aplicar el modelo, podemos obtener el número óptimo de grupos con ayuda de los criterios:

♢ *Akaike information criterion*(AIC)

♢ *Bayesian information criterion*(BIC)



En este caso el criterio indica elegir el valor que minimiza el criterio elegido y para nuestro ejemplo coincide para ambos.



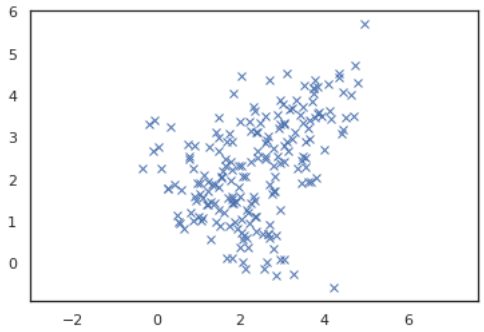
Ejemplo 2:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import pyplot as plt
```

Datos:

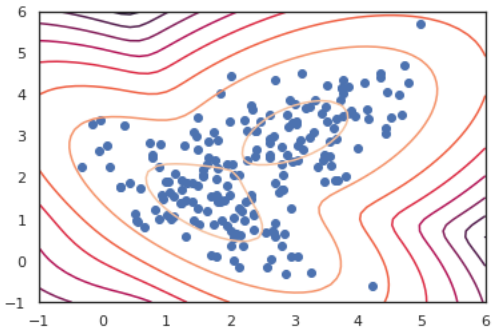
```
#https://bit.ly/3Bic3iu
X_train = np.load('data.npy')
```

```
plt.plot(X_train[:,0], X_train[:,1], 'bx')
plt.axis('equal')
plt.show()
```



Modelo y resultados:

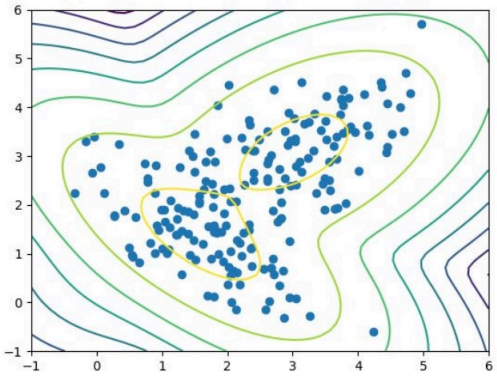
Gráfica de los grupos con curvas de nivel:



2.2.5. Agrupamientos para detectar anomalías

El análisis de agrupamientos puede usarse como método para detectar valores anómalos en conjuntos de datos.

La idea es simple: si un elemento tiene poca afinidad con todos los grupos, entonces es muy probable que se trate de una anomalía.



Se puede realizar con *k-medias*, pero es recomendable utilizar algo más sofisticado y que tome en cuenta la densidad, como un *modelo de mezclas gaussianas* o *DBS*CAN.

2.2.5.1. DBSCAN *Density Based Spacial Clustering of Applications with Noise*, es un algoritmo de agrupamiento basado en densidad: define los grupos como regiones continuas de alta densidad.

Esta técnica clasifica los puntos como **punto núcleo** (*core point*s) (densamente alcanzables) o **anomalías** (*noise point*s) de la forma siguiente:

♢ Para cada elemento del conjunto de datos, contar cuántos puntos se encuentran a una *distancia pequeña* ϵ, conocida como $\epsilon - vecindario$; si un elemento contiene más de cierto umbral *min* dentro de su $\epsilon - vecindario$, se considera como un **punto núcleo** (*core point*).

♢ Todos los elementos en el $\epsilon - vecindario$ de un punto núcleo pertenecen al mismo grupo, puede incluir otros *core points*. Los puntos alcanzables que no tienen más de *min* elementos en su $\epsilon - vecindario$ son llamados **puntos frontera** (*border points*).

♢ Cualquier elemento que no pertenezca a algún grupo, se considera como una **anomalía** o ruido (*noise point*).

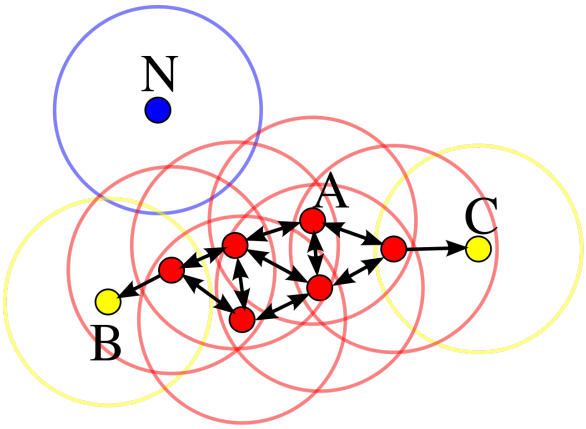


Figura 5: DBSCAN: Los puntos rojos son puntos núcleo; los puntos amarillos son puntos frontera alcanzables y pertenecen al mismo grupo; el punto azul no es alcanzable y se considera como una anomalía. (https://bit.ly/3Dy7qlf)

Una de las ventajas de DBSCAN es que no requiere que se especifique el número de grupos a generar; pero su principal atractivo es que puede agrupar los puntos con formas geométricas arbitrarias.

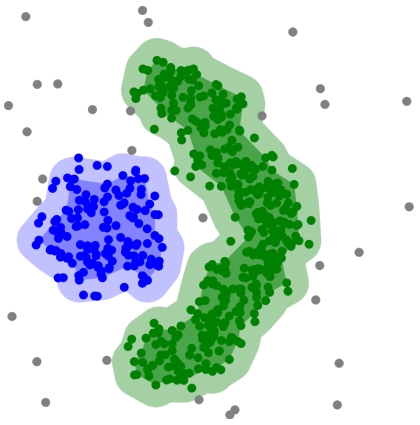


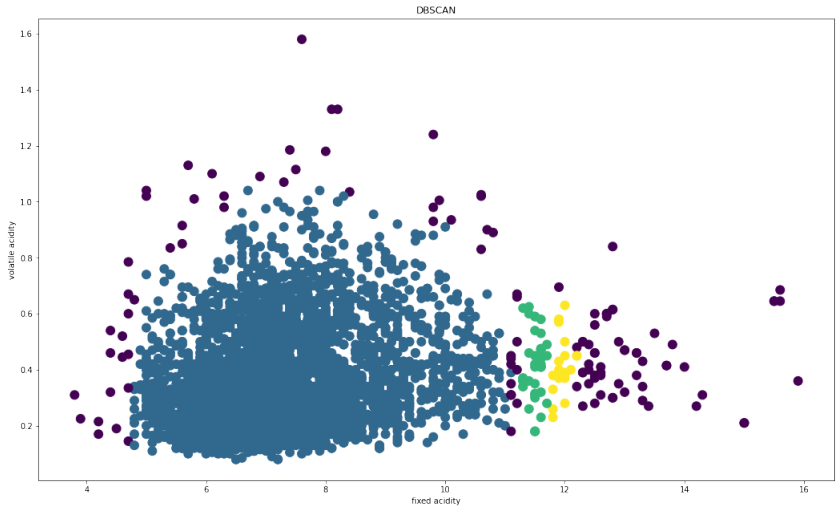
Figura 6: DBSCAN puede determinar grupos con formas complicadas. (https://bit.ly/3v3kb46)

2.2.5.1.1. Elección de hiperparámetros

♢ El valor para *min* puede ser obtenido a partir de la dimensionalidad de los puntos p como $min \geq p + 1$; cuanto más grande sea el conjunto de datos, mayor debe ser el valor asignado a *min*.

♢ Idealmente, el valor ϵ está dado por el problema a resolver, p.e. una distancia física. Cuando esto no sea el caso, debe tenerse en cuenta que un valor pequeño de ϵ puede provocar que muchos datos no se agrupen, mientras que con un valor grande los grupos se fusionarán y la mayoría de los elementos podrían quedar dentro de un solo grupo.

Ejemplo:


DBSCAN
volatile acidity
fixed acidity
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
4
6
8
10
12
14
16