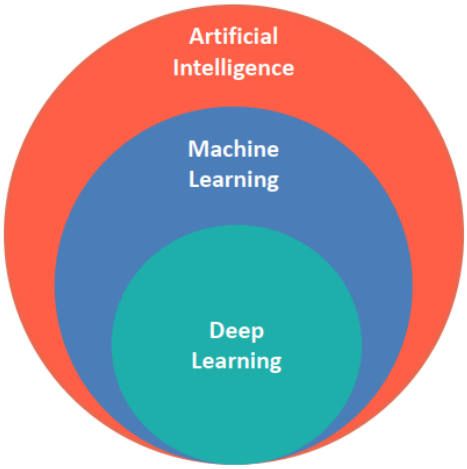
😎 Python y Ciencia de Datos (primera parte)

La ciencia de los datos es un campo que intenta extraer de los datos digitalizados información útil; analizándolos, aplicando distintos procesos, etc.

Las etapas que se distinguen son:

- Etapa 1 ⇒ Detectar el problema: Qué queremos estimar o predecir?
- Etapa 2 ⇒ Obtener y preparar los datos: Qué recursos tenemos?, qué información es relevante? limpiar los datos.
- Etapa 3 ⇒ Explorar los datos: visualizarlos y localizar en los gráficos posibles tendencias, correlaciones o patrones.
- Etapa 4 ⇒ Modelizar y evaluar los datos: crear el modelo con un algoritmo innovador. Evaluar el modelo.
- Etapa 5 ⇒ Comunicar los resultados: Qué resultados hemos obtenido?, Los resultados tienen sentido?



ARTIFICIAL INTELLIGENCE (Inteligencia Artificial)
Ciencia que faculta a las computadoras para imitar la inteligencia humana.

MACHINE LEARNING (Aprendizaje automático)
El Machine Learning es una rama de la Inteligencia Artificial que permite a las máquinas mejorar en una tarea determinada con la experiencia. Es importante aclarar que todas las técnicas de Machine Learning se clasifican como técnicas de Inteligencia Artificial. Sin embargo, no toda la Inteligencia Artificial puede contar como Machine Learning ya que algunos motores básicos basados en reglas pueden ser clasificados como IA pero no aprenden de la experiencia por lo que no pertenecen a la categoría de Machine Learning.

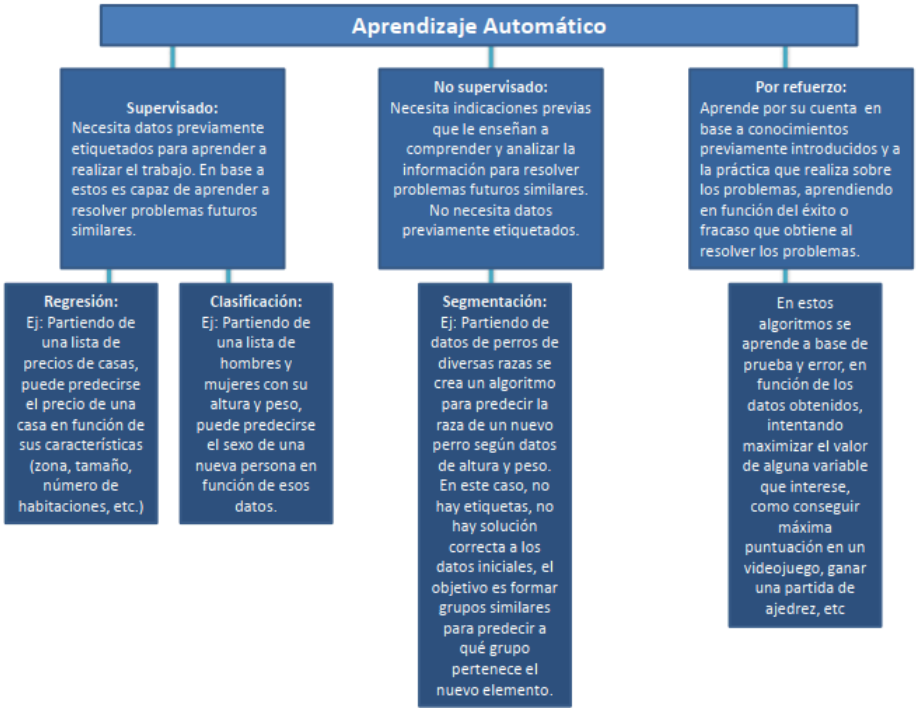
DEEP LEARNING (Aprendizaje Profundo)
El Deep Learning es una rama especializada del Machine Learning que se basa en el entrenamiento de Redes Neuronales Artificiales Profundas usando un gran conjunto de datos como imágenes o textos. Son modelos de procesamiento de información inspirados en el cerebro humano.

	Aprendizaje automático	Aprendizaje profundo
Formato de datos	Datos estructurados	Datos no estructurados
Base de datos	Base de datos manejable	Más de un millón de puntos de datos
Entrenamiento	Se necesita un entrenador humano	El sistema aprende por sí solo
Algoritmo	Algoritmo variable	Red neuronal de algoritmos
Aplicación	Tareas rutinarias sencillas	Tareas complejas
	Algunos ámbitos de aplicación	
	Marketing online	Seguridad de sistemas informáticos
	Atención al cliente	Atención al cliente
	Ventas	Creación de contenidos
	Inteligencia empresarial	Asistentes de voz
	Además de los campos mencionados, ambas tecnologías también se utilizan en muchos otros ámbitos de la vida, como la medicina, la ciencia o la movilidad.	

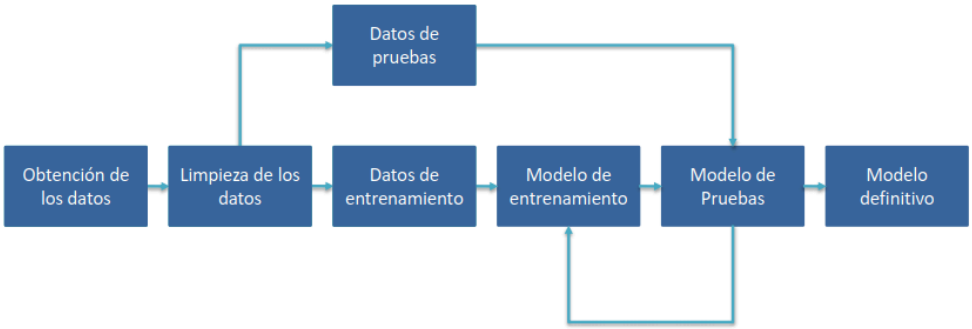
Machine Learning (Aprendizaje automático)

⇒ Es una rama de la IA cuyo objetivo es desarrollar técnicas que permitan que las computadoras aprendan.
⇒ Es un método de análisis de datos que automatiza la construcción de modelos analíticos.
⇒ Permite a las computadoras encontrar soluciones a problemas, sin ser explícitamente programadas para ello, mediante el uso de algoritmos que aprenden de los datos.

Aprendizaje automático clásico



Procesos



Preprocesado de Datos

Análisis previo de los datos a preprocesar: Se debe analizar el conjunto de datos, distinguiendo que nos está mostrando. Datos de ejemplo:

1	Pais,Edad,Salario,Compro
2	Brasil,44,72000,No
3	Argentina,27,48000,Si
4	Chile,30,54000,No
5	Argentina,38,61000,No
6	Chile,40,,Si
7	Brasil,35,58000,Si
8	Argentina,,52000,No
9	Brasil,48,79000,Si
10	Chile,50,83000,No
11	Brasil,37,67000,Si

- 4 columnas: Pais, Edad, Salario y Compro (variables)
- 10 observaciones
- Clientes de una empresa que han comprado un determinado producto.
- Las 3 primeras columnas tienen información personal, origen del cliente, edad y salario.
- La 4ta columna contiene información acerca de cómo ese cliente se relaciona con la empresa (compró o no compró el producto?). Entonces qué diferencia hay entre las variables?

Distinguir variables:

1. Primeras tres columnas: datos que soy capaz de observar, en este caso personas y características, serán las **variable independientes**. $Pais$, $Edad$ y $Salario$, forman una matriz que hay que crear y son las que suministramos al algoritmo de ML

2. La última columna, $Compro$, es la que el algoritmo de ML va a intentar predecir, a partir de las variables independientes, es decir la **variable dependiente**. O sea que intentamos predecir si hubo o no hubo compra.

Importación de los datos

In [1]:
```
import pandas as pd

df = pd.read_csv('archs/Datos.csv')
df
```

Out[1]:

	Pais	Edad	Salario	Compro
0	Brasil	44.0	72000.0	No
1	Argentina	27.0	48000.0	Si
2	Chile	30.0	54000.0	No
3	Argentina	38.0	61000.0	No
4	Chile	40.0	NaN	Si
5	Brasil	35.0	58000.0	Si
6	Argentina	NaN	52000.0	No
7	Brasil	48.0	79000.0	Si
8	Chile	50.0	83000.0	No
9	Brasil	37.0	67000.0	Si

Exploración de los datos

In [2]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 4 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   Pais     10 non-null     object 
 1   Edad     9 non-null      float64
 2   Salario  9 non-null      float64
 3   Compro   10 non-null     object 
dtypes: float64(2), object(2)
memory usage: 448.0+ bytes
```

In [3]:
```
desc1 = df.describe()
desc1
```

Out[3]:

	Edad	Salario
count	9.000000	9.000000
mean	38.777778	63777.777778
std	7.693793	12265.579662
min	27.000000	48000.000000
25%	35.000000	54000.000000
50%	38.000000	61000.000000
75%	44.000000	72000.000000
max	50.000000	83000.000000

Dado el archivo estudiantes2.csv

1. cargar el dataset
2. explorar el mismo con el fin de determinar qué informa cada columna
3. obtener información de los datos
4. obtener la descripción estadística

Preprocesamiento

Valores Desconocidos

Resolver el problema de los datos faltantes: En particular existen *diferentes enfoques* que se pueden llevar a cabo para resolver los **NaN** es decir, decidir qué se debe hacer con esos datos faltantes.

Por ejemplo, en el conjunto de datos original, se puede observar que faltan 2 valores, hay un dato que falta en la columna Edad y luego tenemos una persona de la cual desconocemos su Salario.

1	Pais,Edad,Salario,Compro
2	Brasil,44,72000,No
3	Argentina,27,48000,Si
4	Chile,30,54000,No

```
>>> print(dataset)
       Pais  Edad  Salario Compro
0    Brasil  44.0  72000.0     No
1  Argentina  27.0  48000.0     Si
2     Chile  30.0  54000.0     No
```

In [4]:
```
df.isnull().values.any()
```

Out[4]: True

Devolverá True si hay algún valor NaN en nuestro DataFrame.

In [5]:
```
df.isnull().any()
```

Out[5]:
```
Pais       False
Edad        True
Salario     True
Compro     False
dtype: bool
```

Emite en qué columnas se encuentran nuestros valores NaN.

In [6]:
```
df.isnull().sum().sum()
```

Out[6]: 2

Emite cuántos NaN tenemos en total.

In [7]:
```
nan_rows = df[df.isnull().any(1)]
nan_rows
```

Out[7]:

	Pais	Edad	Salario	Compro
4	Chile	40.0	NaN	Si
6	Argentina	NaN	52000.0	No

Extrae las filas que contienen valores NaN.

a. Hay que encontrar una mejor solución que el eliminar toda la fila.
b. Una posibilidad, para este caso, es reemplazar el valor desconocido de la columna Edad por la media de todos los valores de la columna de Edad.
c. Con el mismo criterio se resolvería el valor desconocido de la columna Salario.

Instalar: **pip install -U scikit-learn** ⇒ https://scikit-learn.org/stable/ (https://scikit-learn.org/stable/)

El método **fit()** se utiliza para calcular la media y la desviación estándar que se utilizará posteriormente para escalar, pero no aplica la transformación real. El cálculo se almacena como un objeto de ajuste. El método no devuelve nada.

El método **transform()** se utiliza para realizar el escalado utilizando la media y la desviación estándar calculadas mediante el método .fit(), aplica la transformación real en la columna, por lo tanto fit() y transform() es un proceso de dos pasos. A diferencia del método fit(), transform() devuelve la matriz transformada.

Con el método **fit_transform()** lo expresado anteriormente se puede reducir a un proceso de un solo paso utilizando fit_transform()

El método **inverse_transform()** permite transformar las etiquetas generadas en los valores originales.

In [8]:
```
from sklearn.impute import SimpleImputer
import numpy as np

'''
SimpleImputer permite sustituir valores nulos por otros valores según varias estrategias disponibles.
La estrategia a ejecutar se indica mediante el parámetro strategy y lo utilizamos para resolver el
problema de los NaN
'''

imputer = SimpleImputer(missing_values = np.nan, strategy="mean")

'''
fit() y transform() se utilizan para centrar/escalar características de un dato dado.
Básicamente ayuda a normalizar los datos dentro de un rango particular.
'''

imputer = imputer.fit(df.iloc[:, 1:3])
df.iloc[:, 1:3] = imputer.transform(df.iloc[:, 1:3])
df
```

Out[8]:

	Pais	Edad	Salario	Compro
0	Brasil	44.000000	72000.000000	No
1	Argentina	27.000000	48000.000000	Si
2	Chile	30.000000	54000.000000	No
3	Argentina	38.000000	61000.000000	No
4	Chile	40.000000	63777.777778	Si
5	Brasil	35.000000	58000.000000	Si
6	Argentina	38.777778	52000.000000	No
7	Brasil	48.000000	79000.000000	Si
8	Chile	50.000000	83000.000000	No
9	Brasil	37.000000	67000.000000	Si

In [9]:
```
desc2=df.describe()
```

Comparamos los datos estádisticos de antes y después de procesar los datos faltantes

In [10]:
```
pd.concat([desc1,desc2], axis=1)
```

Out[10]:

	Edad	Salario	Edad	Salario
count	9.000000	9.000000	10.000000	10.000000
mean	38.777778	63777.777778	38.777778	63777.777778
std	7.693793	12265.579662	7.253777	11564.099406
min	27.000000	48000.000000	27.000000	48000.000000
25%	35.000000	54000.000000	35.500000	55000.000000
50%	38.000000	61000.000000	38.388889	62388.888889
75%	44.000000	72000.000000	43.000000	70750.000000
max	50.000000	83000.000000	50.000000	83000.000000

Dado el archivo estudiantes2.csv

1. Obtener información sobre valores faltantes.
2. Si se encuentran valores faltantes, imputar en las columnas correspondientes eligiendo la estrategia.
3. Comparar las descripciones y obtener conclusiones.

Valores atípicos (outliers)

Los datos que se encuentran a más de 2 o 3 desviaciones estándar del promedio pueden considerarse atípicos. Una primera exploración a la columna 'Salario':

In [11]:
```
np.sort(df['Salario'])
```

Out[11]:
```
array([48000.        , 52000.        , 54000.        , 58000.        ,
       61000.        , 63777.77777778, 67000.        , 72000.        ,
       79000.        , 83000.        ])
```

Existen varias otras técnicas de detección, una es caclular el rango intercuartílico (IQR):

In [12]:
```
Q1 = df['Salario'].quantile(0.25)
Q3 = df['Salario'].quantile(0.75)
IQR = Q3 - Q1

# Definir límites para identificar valores atípicos
limite_inferior = Q1 - 1.5 * IQR
limite_superior = Q3 + 1.5 * IQR

# Identificar valores atípicos
valores_atipicos = df[(df['Salario'] < limite_inferior) | (df['Salario'] > limite_superior)]

print("El valor del IRQ es: ", IQR)
print("El valor del límite inferior es :", limite_inferior)
print("El valor del límite superior es :", limite_superior)
print("Valores Atípicos Detectados :", valores_atipicos)
```

```
El valor del IRQ es:  15750.0
El valor del límite inferior es : 31375.0
El valor del límite superior es : 94375.0
Valores Atípicos Detectados : Empty DataFrame
Columns: [Pais, Edad, Salario, Compro]
Index: []
```

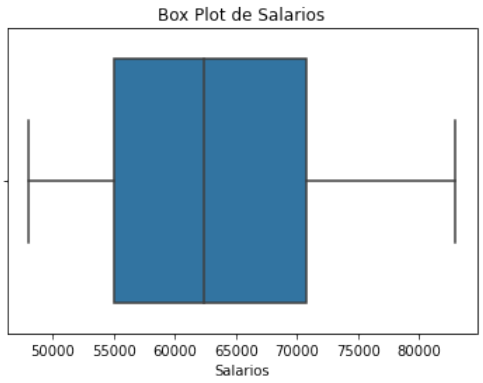
También pueden utilizarse diagramas de caja u otros gráficos:

In [13]:
```
import seaborn as sns

g = sns.boxplot(data = df, x = 'Salario')

# Add a title and change xlabel
g.set_title('Box Plot de Salarios')
g.set_xlabel('Salarios')
```

Out[13]: Text(0.5, 0, 'Salarios')



Z-Score

Mide la distancia de un punto respecto a la media de un grupo, expresada en términos de desviación estándar. Un Z-Score alto puede indicar un valor atípico.

In [14]:
```
# Calcular el Z-Score de los datos
df['z_score'] = (df['Salario'] - df['Salario'].mean()) / df['Salario'].std()

# Identificar valores atípicos
# Por ejemplo, los puntos de datos donde el Z-Score es mayor que 2 o menor que -2
valores_atipicos = df[np.abs(df['z_score']) > 2]
print("Valores Atípicos Detectados:",valores_atipicos)
```

```
Valores Atípicos Detectados: Empty DataFrame
Columns: [Pais, Edad, Salario, Compro, z_score]
Index: []
```

In [15]:
```
df = df.drop('z_score', axis=1)
```

Es recomendable acompañar de más técnicas adicionales para determinar con exactitud la ubicación de los valores atípicos.

Imputar

La imputación implica reemplazar los valores atípicos con otros valores como la mediana o la media. Esto se suele hacer cuando se quiere conservar la mayor cantidad de datos, pero eliminando el efecto de los outliers.

```
Dado el archivo estudiantes2.csv:

1. Detectar valores atípicos.
2. Si se los encuentra, elegir una estrategia para imputar los mismos.
```

Separación de variables independientes de la variable dependiente

In [16]:
```
1 df.head()
```

Out[16]:

	Pais	Edad	Salario	Compro
0	Brasil	44.0	72000.000000	No
1	Argentina	27.0	48000.000000	Si
2	Chile	30.0	54000.000000	No
3	Argentina	38.0	61000.000000	No
4	Chile	40.0	63777.777778	Si

In [17]:
```
1 X = df.iloc[:, :-1].values
2 X
```

Out[17]:
```
array([['Brasil', 44.0, 72000.0],
       ['Argentina', 27.0, 48000.0],
       ['Chile', 30.0, 54000.0],
       ['Argentina', 38.0, 61000.0],
       ['Chile', 40.0, 63777.77777777778],
       ['Brasil', 35.0, 58000.0],
       ['Argentina', 38.77777777777778, 52000.0],
       ['Brasil', 48.0, 79000.0],
       ['Chile', 50.0, 83000.0],
       ['Brasil', 37.0, 67000.0]], dtype=object)
```

In [18]:
```
1 y = df.iloc[:, 3].values
2 y
```

Out[18]:
```
array(['No', 'Si', 'No', 'No', 'Si', 'Si', 'No', 'Si', 'No', 'Si'],
      dtype=object)
```

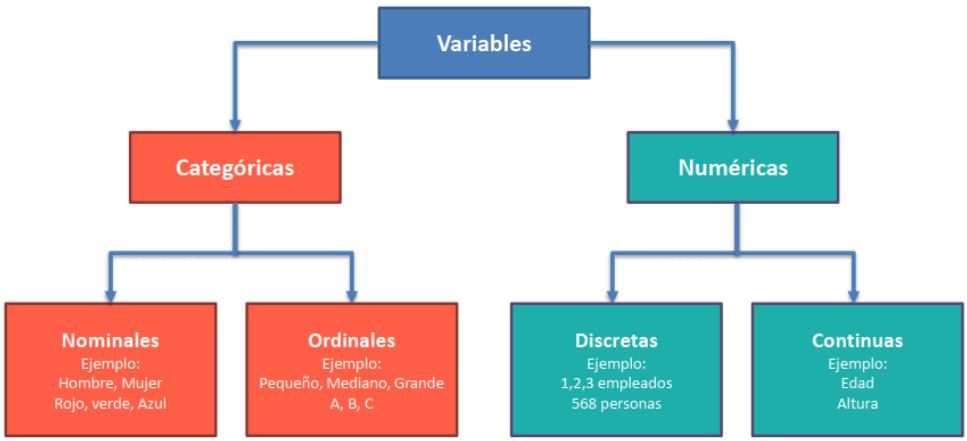
Observar que para X ([[ ]]) devuelve una matriz y para 'y' un vector ([ ])

Dado el archivo estudiantes2.csv:

1. Separar la variable dependiente de las independientes.
2. Emitir los resultados.

Datos categóricos

Tipos de Variables



Datos categóricos son las columnas Pais y Compro, observar que no son datos numéricos. Entonces la *primera face* de limpieza es traducir esos datos a números.

In [19]:
```
1 X
```

Out[19]:
```
array([['Brasil', 44.0, 72000.0],
       ['Argentina', 27.0, 48000.0],
       ['Chile', 30.0, 54000.0],
       ['Argentina', 38.0, 61000.0],
       ['Chile', 40.0, 63777.77777777778],
       ['Brasil', 35.0, 58000.0],
       ['Argentina', 38.77777777777778, 52000.0],
       ['Brasil', 48.0, 79000.0],
       ['Chile', 50.0, 83000.0],
       ['Brasil', 37.0, 67000.0]], dtype=object)
```

Dado el archivo pizzaplace2.csv

1. Eliminar las columnas "id"y "time".
2. Detectar valores faltantes.
3. Si los hay, imputar los mismos.
4. Separar las variables independientes de la dependiente.
5. Detectar las variables categóricas.

In [20]:
```
from sklearn.preprocessing import LabelEncoder
'''
Utilizamos LabelEnconder para convertir a números los datos categóricos.
LabelEnconder codifica etiquetas de una característica categórica en valores numéricos entre 0 y el
número de clases menos 1. Una vez instanciado, el método fit lo entrena (creando el mapeado entre las etiquetas
y los números) y el método transform() transforma las etiquetas que se incluyan como argumento en los
números correspondientes. El método fit_transform realiza ambas acciones simultáneamente.
'''
labelencoder_X = LabelEncoder()
X[:, 0] = labelencoder_X.fit_transform(X[:, 0])
X
```

Out[20]:
```
array([[1, 44.0, 72000.0],
       [0, 27.0, 48000.0],
       [2, 30.0, 54000.0],
       [0, 38.0, 61000.0],
       [2, 40.0, 63777.77777777778],
       [1, 35.0, 58000.0],
       [0, 38.77777777777778, 52000.0],
       [1, 48.0, 79000.0],
       [2, 50.0, 83000.0],
       [1, 37.0, 67000.0]], dtype=object)
```

Cualquier modelo tomará los números de la columna 1 en forma ordinal $(0 < 1 < 2)$... pero los países no son comparables. Surge aquí el concepto de variables dummy o One Hot Encode

Variables Dummy

La introducción de **variables ficticias**, o codificación **One-hot**, es una forma de incluir $variables\ nominales$ en un modelo de ML.

$Ejemplo$: Se tiene una variable nominal que representa una ocupación, con 3 niveles:

- comerciante,
- ingeniero y
- empresario

Se desea incluir esta variable en el modelo para predecir los ingresos por lo tanto hay que convertirla en un $número$, pero **al ser una variable nominal, no existe un orden natural que se pueda utilizar para asignarle un número a cada categoría**.

Es decir que no se puede utilizar un mapeo arbitrario, por ejemplo asignando 1 para comerciante, 2 para ingeniero y 3 para empresario. Si ocurriese, la consecuencia sería que luego de ajustar el modelo, sin importar el coeficiente que obtenga, los ingresos de ingeniero siempre estarán entre los de comerciante y empresario, por lo tanto el mapeo arbitrario no es una forma adecuada de incluir variables nominales.

La forma correcta de incluir variables nominales es la codificación **One-Hot**. Con esta codificación **si la variable tiene n niveles, agrega n-1 columnas a la matriz**. En el ejemplo dado, agregaría 2 columnas, porque hay 3 ocupaciones.

La primera columna, supongamos que pertenece a ingenieros. Esa variable tomaría el valor 1 si la persona es un ingeniero, 0 en caso contrario. La segunda columna, digamos comerciante, sería un indicador similar, pero para comerciante.

Estas variables comerciante e ingeniero son las que se denominan **variables ficticias**. **Para cada nivel de la variable nominal, existe una configuración única de variables ficticias**. Hay que tener en cuenta que, si la variable toma el nivel empresario, ambas variables ficticias son cero. **Dos o más variables ficticias no toman el valor 1 simultáneamente**.

In [21]:
```
X
```

Out[21]:
```
array([[1, 44.0, 72000.0],
       [0, 27.0, 48000.0],
       [2, 30.0, 54000.0],
       [0, 38.0, 61000.0],
       [2, 40.0, 63777.77777777778],
       [1, 35.0, 58000.0],
       [0, 38.77777777777778, 52000.0],
       [1, 48.0, 79000.0],
       [2, 50.0, 83000.0],
       [1, 37.0, 67000.0]], dtype=object)
```

Dado el archivo pizzaplace2.csv

1. Convertir la columna categórica "type"
2. Emitir los datos.

In [22]:
```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer
'''
Utilizamos OneHotEncoder para codificar características categóricas como una matriz numérica
y make_column_transformer permite aplicar transformaciones de datos de forma selectiva a diferentes
columnas del conjunto de datos. Es decir que calcula los datos categóricos y sobreescribe.
'''
onehotencoder = make_column_transformer((OneHotEncoder(), [0]), remainder = "passthrough")
X = onehotencoder.fit_transform(X)
X
```

Out[22]:
```
array([[0.0, 1.0, 0.0, 44.0, 72000.0],
       [1.0, 0.0, 0.0, 27.0, 48000.0],
       [0.0, 0.0, 1.0, 30.0, 54000.0],
       [1.0, 0.0, 0.0, 38.0, 61000.0],
       [0.0, 0.0, 1.0, 40.0, 63777.77777777778],
       [0.0, 1.0, 0.0, 35.0, 58000.0],
       [1.0, 0.0, 0.0, 38.77777777777778, 52000.0],
       [0.0, 1.0, 0.0, 48.0, 79000.0],
       [0.0, 0.0, 1.0, 50.0, 83000.0],
       [0.0, 1.0, 0.0, 37.0, 67000.0]], dtype=object)
```

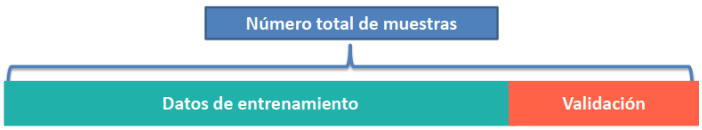
In [23]:
```
labelencoder_y = LabelEncoder()
y = labelencoder_y.fit_transform(y)
y
```

Out[23]:
```
array([0, 1, 0, 0, 1, 1, 0, 1, 0, 1])
```

Dado el archivo pizzaplace2.csv

1. Codificar las características categóricas como una matriz numérica.
2. Eliminar las columnas que no son necesarias para el análisis.
3. Emitir los datos.

Dividir el dataset



Aproximadamente entre el 70% y 80% del conjunto de datos se utilizará para la fase de entrenamiento, mientras que porcentaje restante será utilizado para evaluar, o sea para la fase de testing, definimos:

- **X_train** es el conjunto de entrenamiento, las **variables independientes** que se utilizarán para entrenar el algoritmo.
- **X_test** datos con los cuales se va a testear que el algoritmo funciona correctamente,
- **y_train** los valores de predicción que también se suministran al algoritmo para que aprenda a predecirlos .
- **y_test** se usa para validar si las predicciones de testing, son o no son correctas, para evaluar la performance, la eficacia del algoritmo.

Posibles problemas:

- **Overfitting o sobre ajuste** ocurre cuando se desempeña bien con los datos de entrenamiento, **pero su precisión es notablemente más baja con los datos de evaluación**, esto se debe a que el modelo ha memorizado los datos que ha visto y no pudo generalizar las reglas para predecir los datos que no ha visto.
- **Underfitting o subajuste** se refiere a un modelo que no puede modelar los datos de entrenamiento, **no puede generalizar a nuevos datos**, esto ocurre cuando el modelo es muy simple.

Dividir el dataset en conjunto de entrenamiento y conjunto de test

In [24]:
```
from sklearn.model_selection import train_test_split
'''
train_test_split permite dividir un dataset en bloques, típicamente bloques
destinados al entrenamiento y validación del modelo.
'''
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

In [25]:
```
X_train
```

Out[25]:
```
array([[0.0, 0.0, 1.0, 40.0, 63777.77777777778],
       [0.0, 1.0, 0.0, 37.0, 67000.0],
       [1.0, 0.0, 0.0, 27.0, 48000.0],
       [1.0, 0.0, 0.0, 38.77777777777778, 52000.0],
       [0.0, 1.0, 0.0, 48.0, 79000.0],
       [1.0, 0.0, 0.0, 38.0, 61000.0],
       [0.0, 1.0, 0.0, 44.0, 72000.0],
       [0.0, 1.0, 0.0, 35.0, 58000.0]], dtype=object)
```

In [26]:
```
X_test
```

Out[26]:
```
array([[0.0, 0.0, 1.0, 30.0, 54000.0],
       [0.0, 0.0, 1.0, 50.0, 83000.0]], dtype=object)
```

In [27]:
```
1 y_train
```

Out[27]:
```
array([1, 1, 1, 0, 1, 0, 0, 1])
```

In [28]:
```
1 y_test
```

Out[28]:
```
array([0, 0])
```

Dado el archivo pizzaplace2.csv

1. Dividir el dataset en X_train, X_test, y_train, y_test con un tamaño de 1/3.
2. Emitir los datos.

Escalar o normalizar los datos

En **escala**, se cambia el rango de distribución de los datos. Es decir que se van a transformar los valores de las características de forma que estén dentro de un rango, típicamente $[0, 1]$ o $[-1, 1]$. En general, es posible que se necesite escalar datos para problemas de aprendizaje automático de modo que todas las variables tengan un rango de distribución bastante similar para evitar problemas.

En **normalización**, se cambia la forma de distribución de los datos. La normalización va a transformar las características de forma que todas compartan un mismo valor medio y una misma desviación media. Es una transformación más radical. El punto de la normalización es cambiar las observaciones para que puedan describirse como una distribución normal.

Más información (https://statologos.com/normalizar-datos-entre-0-y-1/)

La escala y la normalización son muy similares, a menudo se aplican indistintamente, pero tienen diferentes efectos en los datos, se deben comprender las diferencias y saber cuándo aplicar una en lugar de la otra.

Más información (https://towardsai.net/p/data-science/scaling-vs-normalizing-data-5c3514887a84)

En el ejemplo, ocurre que la columna de Edad y la columna de Salario no se encuentran dentro del mismo rango de valores. La diferencia es grande por lo tanto el algoritmo lo tiene que tener en cuenta. La solución a la diferencia es escalar los datos.

Escalar significa normalizar los datos para que estén definidos en el mismo rango de valores, ejemplo típico es escalar entre -1 y 1, de modo que la $Edad$ más pequeña correspondería al valor -1 y la mayor a 1.

Con el $Salario$ (u otras columnas numéricas) es igual, el salario menor corresponde a -1 y a partir de ahí se escalaría linealmente hasta 1 que representaría el salario más elevado.

Esta normalización o estandarización es muy importante para evitar que unas variables dominen sobre otras dentro del algoritmo de ML y que el propio algoritmo pueda discernir que peso le dará a cada una de las variables, no por tener un rango grande o pequeño sino porque aportan en el proceso de predicción o clasificación.

Hay que hacer lo mismo con y_train, y_test ? en este caso no, porque lo que se quiere es clasificar compra o no compra. No obstante hay casos, como en **la regresión lineal**, donde se recomienda que se realice la normalización del array de datos a predecir para guardar una consistencia.

In [29]:
```
1 from sklearn.preprocessing import StandardScaler
2
3 sc_X = StandardScaler()
4 X_train = sc_X.fit_transform(X_train)
5 X_test = sc_X.transform(X_test)
```

In [30]:
```
1 X_train
```

Out[30]:
```
array([[-0.77459667, -1.        ,  2.64575131,  0.26306757,  0.12381479],
       [-0.77459667,  1.        , -0.37796447, -0.25350148,  0.46175632],
       [ 1.29099445, -1.        , -0.37796447, -1.97539832, -1.53093341],
       [ 1.29099445, -1.        , -0.37796447,  0.05261351, -1.11141978],
       [-0.77459667,  1.        , -0.37796447,  1.64058505,  1.7202972 ],
       [ 1.29099445, -1.        , -0.37796447, -0.0813118 , -0.16751412],
       [-0.77459667,  1.        , -0.37796447,  0.95182631,  0.98614835],
       [-0.77459667,  1.        , -0.37796447, -0.59788085, -0.48214934]])
```

In [31]:
```
1 X_test
```

Out[31]:
```
array([[-0.77459667, -1.        ,  2.64575131, -1.45882927, -0.90166297],
       [-0.77459667, -1.        ,  2.64575131,  1.98496442,  2.13981082]])
```

Dado el archivo pizzaplace2.csv

1. Normalizar los datos.
2. Emitir los datos.