

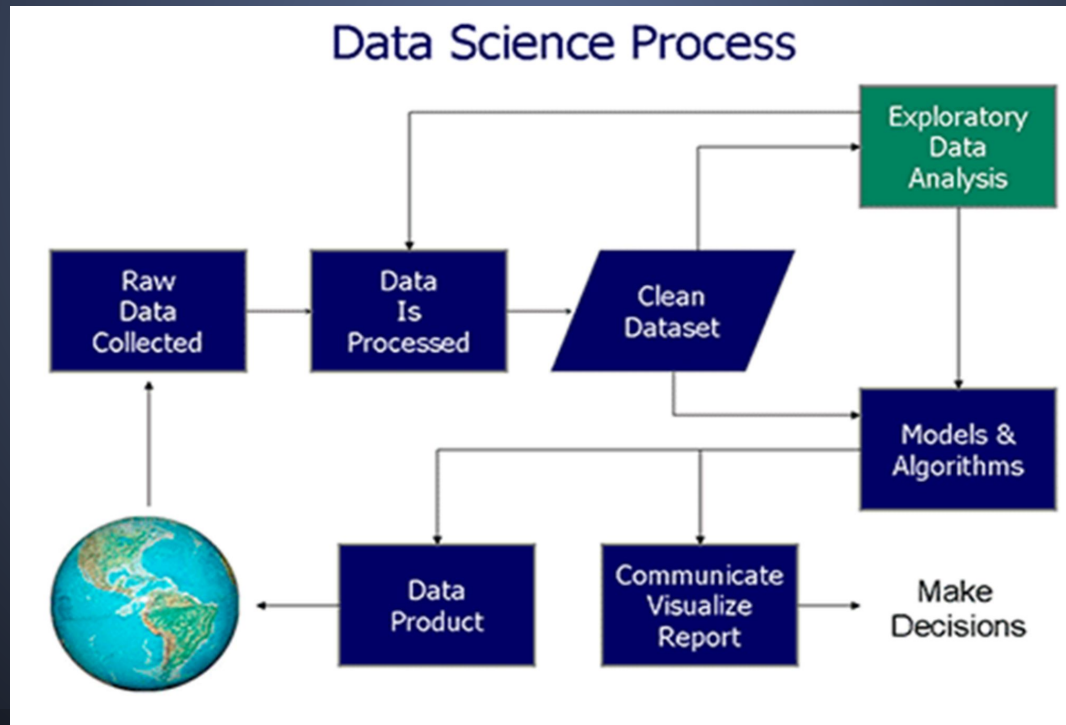


Aprendizaje Automático y Minería de Datos

Minería de datos
Fundamentos y ciclo de vida

Motivación

- La Ciencia de Datos sirve para **generar informes** que facilitan tomar decisiones o **desarrollar productos software** “basados en datos”



Motivación

- En Estadística, ser “minero de datos” era un término *despectivo*
- La academia en 1989 prefiere llamarlo **Descubrimiento de Conocimiento en Bases de Datos** KNOWLEDGE DISCOVERY IN DATABASES (KDD)
- **Fayyad, Piatetsky-Shapiro y Smyth (1996)** lo definen como “proceso no trivial de identificar patrones válidos, novedosos, potencialmente útiles y comprensibles a partir de los datos” y ofrecen 5 pasos a seguir

I have a joke about
a Data Miner



But you probably
won't dig it

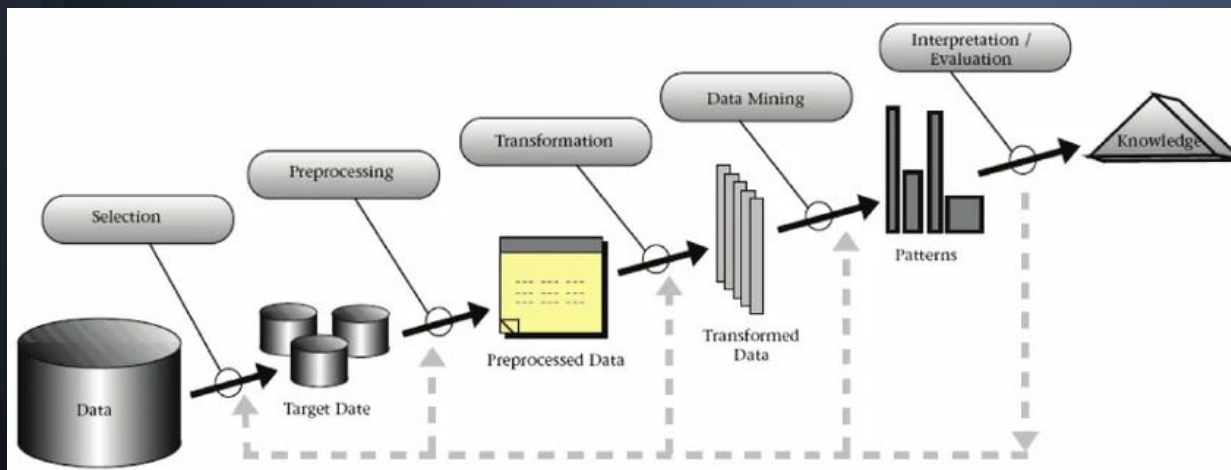
Proceso de Fayyad et al.



Inicio: Datos brutos

1. **Selección y captura** FILTERING
2. **Preprocesamiento y limpieza** DATA CLEANING / IMPUTATION
3. **Transformación y reducción** FEATURE ENGINEERING
4. **Minería de datos** (propiamente dicha, ahora AA)
5. **Evaluación e interpretación** PATTERN EVALUATION

Final: Conocimiento útil / Decisión



Proceso Estándar Interindustrial

CROSS-INDUSTRIAL STANDARD PROCESS FOR DATA MINING (CRISP-DM)

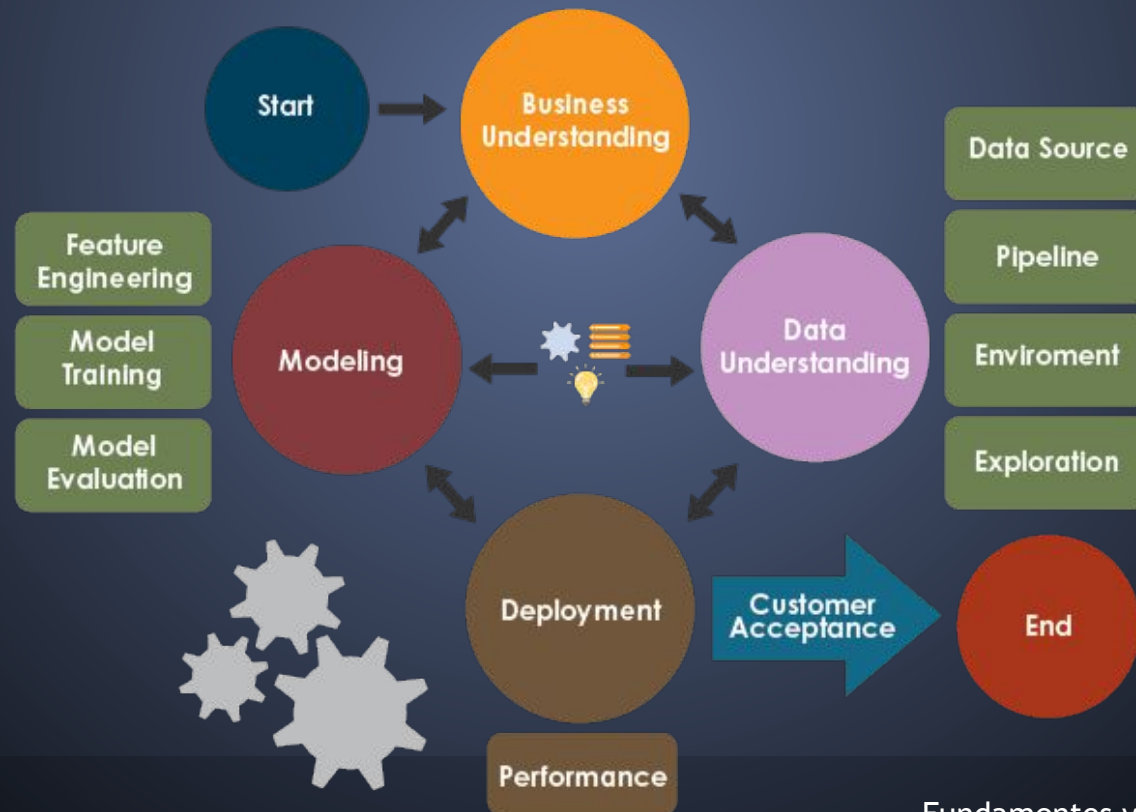
- Creado por un consorcio empresarial en 1996, y publicado en el 2000
 - Comprender el negocio
 - Comprender los datos
 - Preparar los datos
 - Modelar
 - Evaluar
 - Desplegar



Proceso de Ciencia de Datos en Equipo

TEAM DATA SCIENCE PROCESS (TDSP)

- Mejora creada por Microsoft en 2016 pensando en AA, Big Data, metodologías ágiles...



Proyectos reales



- La **mayor parte del tiempo** de un proyecto es MD (*conseguir, limpiar y transformar datos*)
 - Durante el entrenamiento de modelos en AA también se tarda mucho en los *ajustes finos*
 - Los procesos nunca son *lineales*, sino **iterativos**
- INSIGHTS ○ Los **hallazgos** modelando o evaluando pueden redefinir el problema y hacernos volver al MD

Ejemplos

Data-Informed mejor:
Art first, Science second!

DATA-DRIVEN GAME DEVELOPMENT

- Desarrollo de videojuegos dirigido por datos
 - King, Riot Games, Supercell...



Nuestro proceso

- **Adquisición** (incluye captura y selección) y **exploración** (analítica e incluso *visual*)
- **Limpieza y calidad** (preprocesamiento, volviendo a explorar para confirmar datos limpios y fiables)
- **Transformación y reducción** (conversión a un formato adecuado y computable)
- De *Modelado* y de *Evaluación e Interpretación* hablaremos más en AA
- El *Despliegue* a escala industrial NO lo vemos (aunque ahora hay *modelos hasta para móviles*)

Mantenimiento

- Los datos (y por tanto los modelos) pueden perder relevancia y **degradarse** DATA / MODEL DECAY
 - **Deriva de concepto**, ya no existen la misma relación entre entradas y salidas
CONCEPT DRIFT
 - **Deriva de datos**, las nuevas entradas ya no son como las que teníamos
DATA DRIFT
 - **Deriva de etiquetas**, las salidas correctas ya no son como las que teníamos (en **Aprendizaje Supervisado**)
LABEL DRIFT
- Mantener el sistema implica *actualizar* los conjuntos de datos y *reentrenar* modelos

Canalizaciones

PIPELINES

- Secuencias ordenadas y automatizadas de procesos de manipulación de datos
 - Conectamos la salida de un proceso con la entrada del siguiente y así evitamos errores
 - Simplifica el código y evita fuga de datos
- En bibliotecas como **PySpark**, **Scikit-Learn**, etc. hay clases que implementan esto

```
mi_pipeline = Pipeline(steps=[
    ('imputador', SimpleImputer(strategy='mean')),
    ('escalador', StandardScaler()),
    ('modelo', LogisticRegression())
])
```

Participación

- Entonces, ¿qué vamos a ver **en este tema**?
 - A. Adquisición, Exploración, Modelado y Despliegue
 - B. Adquisición, Exploración, Modelado y Transformación
 - C. Adquisición, Exploración, Limpieza y Transformación
 - D. Selección, Exploración, Modelado y Transformación
 - E. Selección, Exploración, Modelado y Despliegue



- Biblioteca para el **manejo eficiente de vectores, matrices, tipos de datos de bajo nivel** y realización de **operaciones vectorizadas**
 - ¡El motor *superoptimizado* de cálculo numérico sobre el que trabaja **Pandas**, **Scikit-Learn**, **TensorFlow**, **PyTorch**...!
- Tiene la estructura de datos n-dimensional **ndarray** (deja obsoleto al *array* y es mejor que *list* para hacer cálculos)
 - Permite operar *sin* bucles explícitos y *decenas de veces* más rápido que *list* (nada en **C**)

```
import numpy as np

# En Python nativo: [x * 2 for x in datos]
# En NumPy (operación vectorizada):
arr = np.array([1, 2, 3, 4])
resultado = arr * 2 # Multiplica cada elemento sin bucles
```

*Lleva paréntesis Y corchete

NDArray

- Propiedades clave de esta estructura
 - **shape**, tupla con sus dimensiones
 - **dtype**, tipo de dato *para todos sus elementos*
 - **axis**, eje sobre el que se trabaja
 - En una matriz (2 dimensiones) el **0** son las **columnas** verticales y el **1** las **filas** horizontales

```
# Matriz 2x3 con float32 (para ver el dtype en la salida)
matriz = np.array([[1.0, 2.0, 3.0],
                  [4.0, 5.0, 6.0]], dtype=np.float32)

suma_col = matriz.sum(axis=0) # axis=0: por columnas
suma_fil = matriz.sum(axis=1) # axis=1: por filas

# La última línea muestra shape, resultados y dtype
(matriz.shape, suma_col, suma_fil)
```



```
((2, 3),
 array([5., 7., 9.], dtype=float32),
 array([ 6., 15.], dtype=float32))
```

**Se omite el cero decimal,
y si el tipo es float64 también se omite*

NDArray

SLICING

- Segmentación multidimensional
- Indexación booleana, siempre devuelve un vector (1 dimensión) y se usa para filtrar

```
# Matriz de ejemplo de 3x3
datos = np.array([[10, 20, 30],
                  [40, 50, 60],
                  [70, 80, 90]])

# 1. Segmentación multidimensional: filas 0 y 1, columnas 1 y 2
submatriz = datos[0:2, 1:3]

# 2. Indexación booleana: elementos mayores que 40
mascara = datos > 40
filtrados = datos[mascara]

# Salida interactiva en la última línea
(submatriz, mascara, filtrados)
```



```
(array([[20, 30],
        [50, 60]]),
 array([[False, False, False],
        [False,  True,  True],
        [ True,  True,  True]]),
 array([50, 60, 70, 80, 90]))
```

NDArray

BROADCASTING

- **Difusión automática**, operar con estructuras de distintas dimensiones “estirando” la menor

- **Reorganización sin copia**

RESHAPE

- ¡Ojo! Segmentar y reorganizar devuelven una **vista** (**no copia**) del original que permite modificarlo

```
# 100 muestras, 3 características
X = np.random.rand(100, 3)

# Calcula la media de cada columna (recorre el eje vertical) -> Tamaño (3,)
media = X.mean(axis=0)

# Difusión automática: resta a cada fila el vector de medias
X_centrado = X - media
```

```
vector = np.arange(12)           # Shape (12,)
matriz = vector.reshape(4, 3)    # Shape (4, 3)
transpuesta = matriz.T           # Transpuesta: Shape (3, 4)
```

NDArray

- Estadística rápida, álgebra lineal, generación de números aleatorios, etc.

```
import numpy as np

# 1. Generación de datos aleatorios (con el generador recomendado)
rng = np.random.default_rng(42) # Semilla para reproducibilidad
datos = rng.random((3, 4))      # Matriz de 3x4 con valores entre 0 y 1
```

```
# 2. Estadística rápida (especificando axis)
media_col = datos.mean(axis=0) # axis=0: colapsa filas (resultado por columna)
desv_fil = datos.std(axis=1)   # axis=1: colapsa columnas (resultado por fila)
suma_col = datos.sum(axis=0)
min_fil = datos.min(axis=1)
max_col = datos.max(axis=0)
```

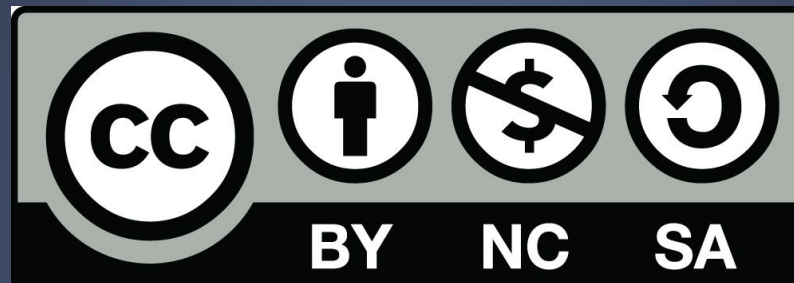
```
# 3. Álgebra lineal: Producto matricial (@)
pesos = rng.random((4, 2))    # Matriz de 4x2
proyeccion = datos @ pesos     # Multiplicación: (3x4) @ (4x2) -> (3x2)
```

```
# Salida interactiva para la celda del Notebook
(datos.shape, media_col, proyeccion)
```



```
((3, 4),
 array([0.62723612, 0.61205345, 0.63806208, 0.40788647]),
 array([[0.86541539, 0.83856114],
        [0.85290264, 0.77194689],
        [0.88081699, 1.0537446 ]]))
```

Críticas, dudas, sugerencias...



* Excepto el contenido multimedia de terceros autores

Federico Peinado (2026)

www.federicopeinado.es

