Análisis exploratorio de los datos

Aprendizaje Automatico

Marco Teran
EAFIT

2025

Contenido

Objetivos y mapa

Objetivos de aprendizaje del capítulo

- Entender CRISP–DM y su lógica iterativa.
- Dominar EDA: métricas, visualización y robustez.
- Practicar ingeniería de características efectiva y segura.
- Diferenciar sobreajuste vs. subajuste; usar regularización.
- Validar con rigor: *holdout*, CV y *mismatch*.
- Diseñar *pipelines* reproducibles y medibles.
- Evitar *data leakage* y sesgos comunes.

Mapa mental

CRISP–DM $\rightarrow$ EDA $\rightarrow$ Ingeniería de características $\rightarrow$ Modelado $\rightarrow$ Validación $\rightarrow$ Despliegue $\rightarrow$ Iteración

Capítulo en una página

- **Fundamentos**: objetivo de negocio y métricas.
- **EDA**: localización, dispersión, forma, dependencia.
- **Visualización**: principios perceptuales y gramática.
- **Características**: codificación, escalado, transformaciones.
- **Teoría clave**: sesgo–varianza, capacidad, información.
- **Validación**: particiones, CV, *mismatch*, *drift*.
- **Prácticas**: *pipelines*, monitoreo y retraining.

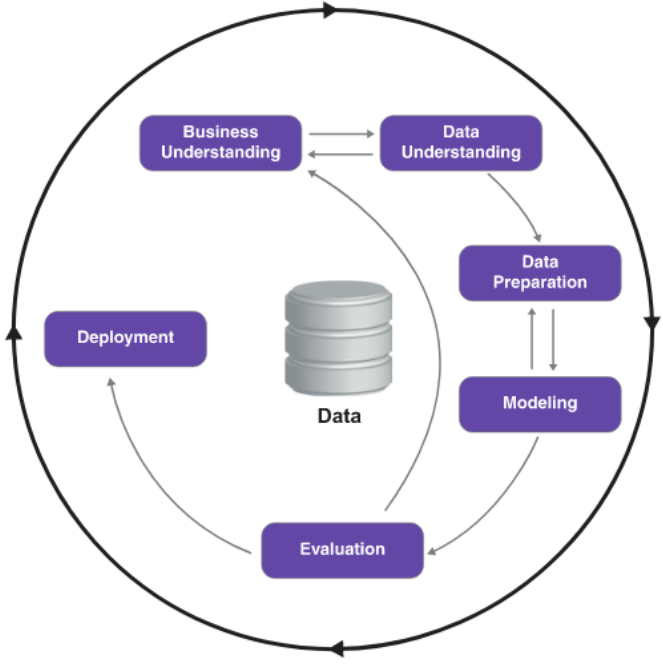
Introducción

CRISP-DM

- Un proyecto de ML trasciende elegir y entrenar modelos.
- Existen marcos para organizar el ciclo de vida completo.
- CRISP–DM es el estándar intersectorial (1996).

Idea central

Del objetivo de negocio a producción, con iteraciones informadas por datos.


Business Understanding
Data Understanding
Data Preparation
Modeling
Evaluation
Deployment
Data

CRISP-DM: fases y preguntas guía

- **Comprensión del negocio**
  - ¿Qué KPI mover? ¿Por qué ahora?
  - ¿ML agrega valor frente a reglas?
- **Comprensión de los datos**
  - ¿Fuentes confiables? ¿Cobertura temporal?
  - ¿Calidad y sesgos de muestreo?
- **Preparación de datos**
  - Estructurar, limpiar, enriquecer, etiquetar.
- **Modelado**
  - Objetivo, pérdida y validación coherentes.
- **Evaluación**
  - ¿Mueve el KPI empresarial?
  - ¿Riesgos y *trade-offs* aceptables?
- **Despliegue**
  - Integración, monitoreo y plan de retraining.

Ejemplo

Supongamos que queremos construir un sistema de detección de spam: para cada correo electrónico que recibimos, queremos determinar si es spam o no. Si lo es, queremos meterlo en la carpeta de "spam".

Reflexión

Métrica de negocio: quejas y clics en "Report spam".


Business Understanding
Data Understanding
Data Preparation
Modeling
Evaluation
Deployment

Etapa de comprensión del negocio

- Analizar quejas de usuarios y volumen de spam.
- Evaluar si ML supera reglas y filtros actuales.
- Definir éxito: reducción de reportes o quejas.
- Considerar alternativas no–ML si procede.

Pitfall

KPI mal definido $\rightarrow$ optimización de la métrica equivocada.

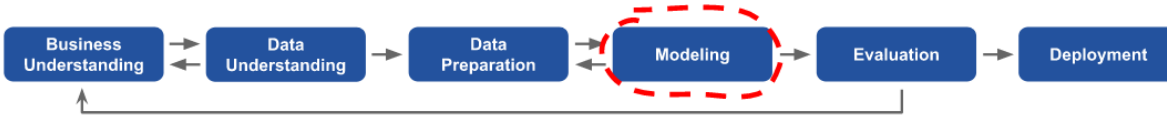

Business Understanding
Data Understanding
Data Preparation
Modeling
Evaluation
Deployment

Paso de comprensión de datos

- Inventariar fuentes: etiquetas de reporte de spam, logs.
- Auditar tamaño, ruido, faltantes y cobertura.
- Detectar sesgo de muestreo y *drift* temporal.
- Si falta calidad: mejorar captura o adquirir datos.

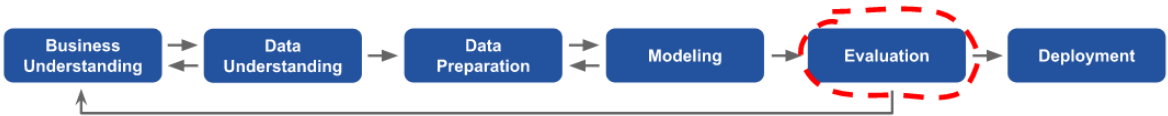
Nota histórica

Tukey impulsó el EDA: primero explorar, luego formalizar.


Business Understanding
Data Understanding
Data Preparation
Modeling
Evaluation
Deployment

Paso de modelado

- Seleccionar algoritmos y definir pérdidas y métricas.
- Ajustar preparación según validación inicial.
- Diseñar un marco de validación honesto.
- Promover el mejor candidato a evaluación final.


Business Understanding
Data Understanding
Data Preparation
Modeling
Evaluation
Deployment

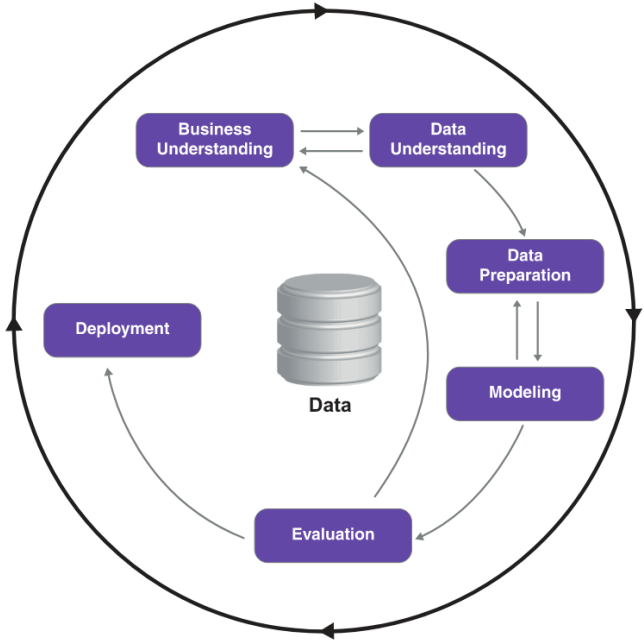
Etapa de evaluación

- Comprobar alineación con expectativas del negocio.
- Validar que el KPI empresarial mejora realmente.
- Usar *holdout* ciego para estimar generalización.
- Evaluar riesgos: *false positives* y experiencia.


Business Understanding
Data Understanding
Data Preparation
Modeling
Evaluation
Deployment

Etapa de despliegue

- *Ramp-up* por cohortes; medir impacto causal.
- Comparar grupo tratado vs. control (A/B).
- Reciclar aprendizaje: retroalimentar al inicio.
- Reevaluar el objetivo si cambia el entorno.


Business Understanding
Data Understanding
Data Preparation
Modeling
Evaluation
Deployment
Data

Iterar

- Iteraciones planificadas según evidencia y riesgo.
- El valor está antes y después del modelado.
- Mejor pregunta correcta aproximada que exactitud vana.
- EDA disciplinado evita sorpresas en producción.

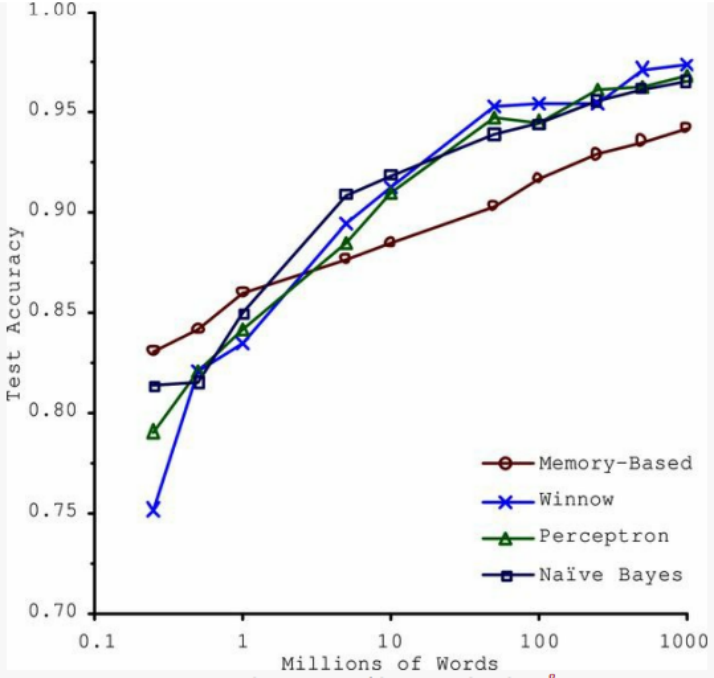
Principales desafíos del Machine Learning

Principales desafíos del Machine Learning

- Dos ejes: calidad de datos y capacidad del modelo.
- Peligros: poca data, sesgos, ruido y *leakage*.
- Antídotos: EDA robusto y validación honesta.

Cantidad insuficiente de datos de entrenamiento

- Muchos algoritmos requieren grandes volúmenes.
- Tareas complejas: millones o *transfer learning*.
- Curva de datos guía inversión y expectativas.


1.00
0.95
0.90
0.85
0.80
0.75
0.70
Test Accuracy
0.1
1
10
100
1000
Millions of Words
Memory-Based
Winnow
Perceptron
Naïve Bayes

Datos de entrenamiento no representativos

Para generalizar, la muestra debe representar producción.

- Muestras grandes también pueden sesgarse.
- Cuide el muestreo estratificado y temporal.
- Documente el proceso de muestreo siempre.

Pitfall

Sesgo de muestreo $\rightarrow$ métricas infladas en validación.

Datos de baja calidad

- Errores, outliers y ruido degradan patrones.
- Limpieza: reglas, imputación y validación cruzada.
- Estrategias: descartar, corregir o modelar faltantes.

Características irrelevantes

- El éxito depende de buenas representaciones.
- Ingeniería: selección, extracción y reducción.
- Cuidado con alta cardinalidad y multicolinealidad.

Sobreajuste (overfitting) y regularización

- Ajuste perfecto al ruido no generaliza.
- Modelos complejos requieren control de capacidad.

$$\underbrace{\mathbb{E}(y-\hat{f})^2}_{\text{riesgo}} = \sigma^2 + \text{bias}^2 + \text{varianza}$$

Teorema clave

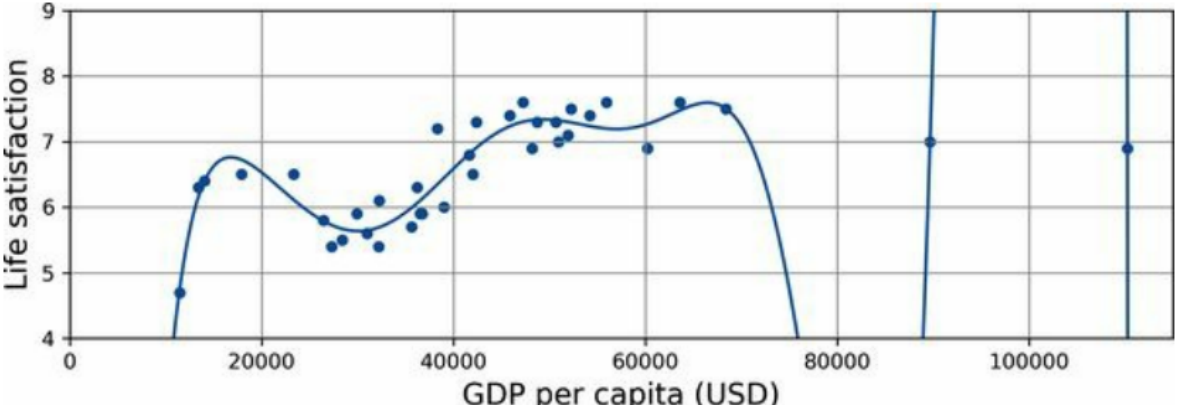
Descomposición sesgo–varianza: guía el *trade-off* óptimo.

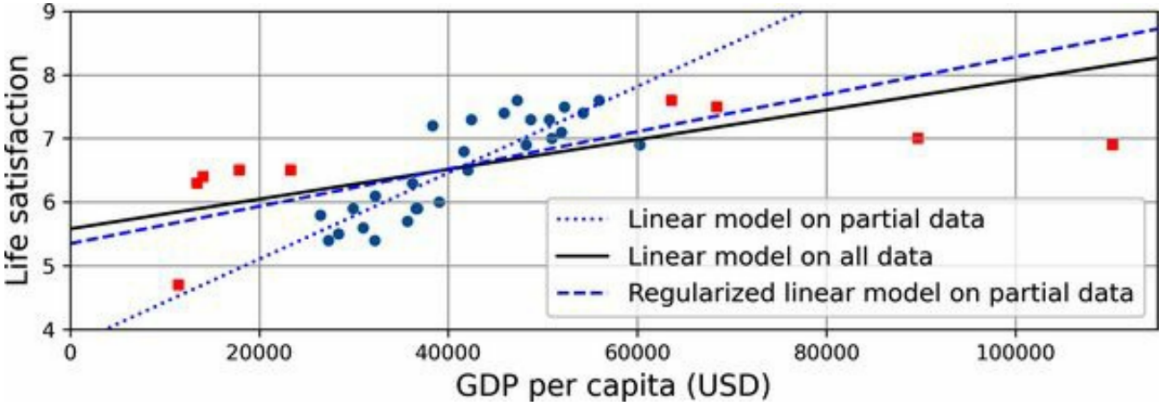
Cómo reducir el sobreajuste

- Simplificar modelo o usar regularización.
- Conseguir más y mejores datos.
- Reducir ruido y estabilizar etiquetas.

Regla robusta

Valide cada decisión de limpieza con CV estratificada.


Life satisfaction
9
8
7
6
5
4
0
20000
40000
60000
80000
100000
GDP per capita (USD)


Life satisfaction
GDP per capita (USD)
Linear model on partial data
Linear model on all data
Regularized linear model on partial data
9
8
7
6
5
4
0
20000
40000
60000
80000
100000

Ajuste insuficiente (underfitting)

- Modelo demasiado simple para la estructura.
- Soluciones: más capacidad o mejores rasgos.
- Ajustar hiperparámetros y aflojar restricciones.

Resumen (I)

- ML aprende de datos para mejorar tareas específicas.
- Tipos: supervisado, no supervisado, por lotes, en línea.
- Entrenamiento ajusta parámetros para predecir bien fuera.
- Éxito depende de datos, validación y mantenimiento.

Resumen (II)

- EDA disciplinado descubre estructura y anomalías.
- Ingeniería de rasgos conecta dominio y algoritmo.
- Regularización y CV controlan la generalización.
- CRISP–DM guía el ciclo de vida completo.

Pruebas y validación

Principios de validación

- Generalización se estima con datos no vistos.
- Separar entrenamiento, validación y prueba.
- Tamaño del *test* depende de la data.
- Evitar adaptar decisiones al conjunto de prueba.

Ajuste de hiperparámetros y selección de modelos

Conjuntos y flujo de selección

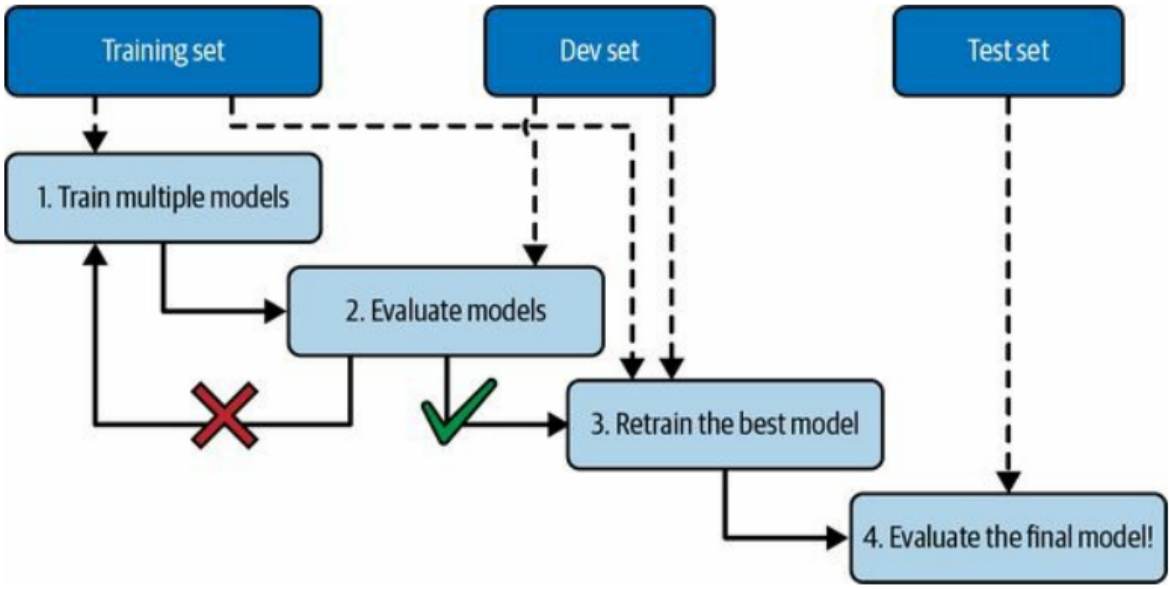
- Comparar candidatos en validación, no en prueba.
- Regularizar para minimizar riesgo esperado.
- Barrido de hiperparámetros con CV estratificada.
- Elegir el mejor y reentrenar en todo el *train*.

Cuándo usar CV y cuánto

- K–fold reduce varianza de estimación.
- CV repetida mejora estabilidad con costo extra.
- Para series temporales use *forward chaining*.

Pitfall

No mezcle información futura en particiones temporales.


Training set
Dev set
Test set
1. Train multiple models
2. Evaluate models
3. Retrain the best model
4. Evaluate the final model!

Divergencia de datos

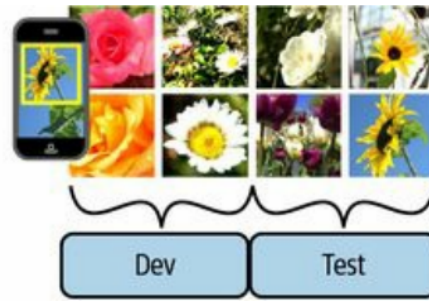
Cuando validación no representa producción

- Entrenamiento puede no reflejar el *live traffic*.
- Validación y prueba deben emular producción.
- Deduplicar entre *splits* y periodos temporales.
- Mantener un conjunto de desarrollo para diagnóstico.

Diagnóstico de *mismatch*

- Mal en train y dev: simplifique, limpie, más datos.
- Bien en train y dev, mal en test: preprocesar como producción.
- Sólo entonces, estimar con prueba final una vez.


Train
Train-dev


Dev
Test

Proyecto de Machine Learning de principio a fin

Exploración de datos

Exploración de datos: propósito

- Comprender estructura, calidad y señales útiles.
- Guiar decisiones de rasgos y validación.
- Formular hipótesis y detectar anomalías.

Checklist esencial de EDA

1. Auditar faltantes y outliers sistemáticos.
2. Distribuciones: centro, dispersión y forma.
3. Dependencias: correlación e información mutua.
4. Estabilidad temporal y *drift* de covariables.
5. Balance de clases y separación de etiquetas.

Ventajas de un EDA riguroso

1. Evita errores caros en producción.
2. Acelera selección de modelos adecuados.
3. Mejora interpretabilidad y gobernanza.
4. Reduce sobreajuste por decisiones informadas.

Técnicas de EDA recomendadas

1. Estadística descriptiva robusta: mediana, IQR, MAD.
2. Gráficos: histogramas, cajas, dispersión, facetas.
3. Reducción: PCA para explorar multivariado.
4. Dependencia: Spearman y mutual information.

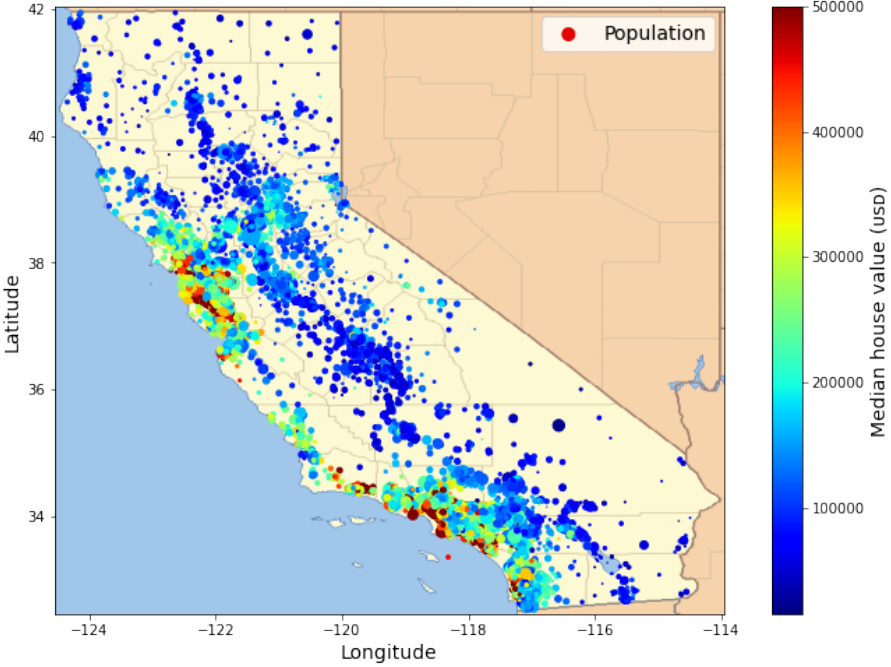
Pasos para resolver con EDA

1. Analizar el panorama general del negocio.
2. Obtener y versionar los datos crudos.
3. Explorar y visualizar con criterios robustos.
4. Preparar datos para algoritmos de ML.

Dataset de precios de casas de California

Dataset de precios de casas de California

1. Usaremos California Housing Prices (censo 1990).
2. Apto para aprender EDA y regresión tabular.
3. Se añadió un atributo categórico para práctica.
4. Se eliminaron rasgos para simplificar ejercicios.


Population
Latitude
Longitude
42
40
38
36
34
−124
−122
−120
−118
−116
−114
Median house value (usd)
500000
400000
300000
200000
100000

California Housing: rasgos y tareas

- 20,640 instancias, 8 atributos demográficos y geográficos.
- Rangos heterogéneos: considerar escalado o transformaciones.
- Tareas: predecir valor medio y analizar patrones.

Tener una visión amplia

- Objetivo: predecir precio medio por distrito.
- Insumos: población, ingresos, habitaciones, localización.
- La unidad: distrito del censo (600–3,000 personas).

Lista de comprobación de proyecto

- Usar checklist de ML adaptado al contexto.
- Documentar supuestos, métricas y decisiones.
- Separar datos crudos, intermedios y limpios.

Enmarcar el problema

- ¿Cómo generará valor el modelo?
- Integración aguas abajo define tolerancias.
- Métrica empresarial condiciona la técnica.

Pitfall

Optimizar RMSE sin impacto en el ROI definido.

Pipelines

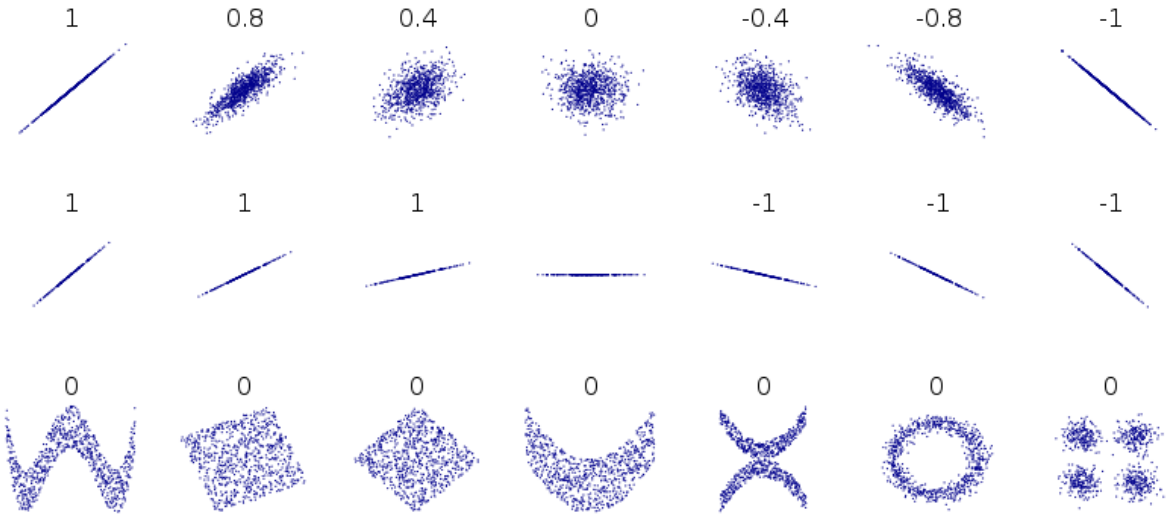
Pipelines

Una secuencia de datos que procesan componentes se llama pipeline de datos. Las pipelines son muy comunes en los sistema de machine learning, puesto que hay muchos datos que manipular y muchas transformaciones de datos que aplicar.

- Componentes asíncronos con almacenamiento compartido.
- Equipos independientes, acoplamiento débil y resiliencia.
- Monitoreo evita *stale data* y degradación.

Buenas prácticas

Encapsular, versionar, ser deterministas y trazables.


1
0.8
0.4
0
-0.4
-0.8
-1
1
1
1
-1
-1
-1
0
0
0
0
0
0
0

Conceptos fundamentales de EDA

¿Qué es EDA?

- *Exploratory Data Analysis*: metodología para descubrir estructura.
- Combina estadísticas descriptivas y visualización.
- Objetivos: patrones, anomalías, hipótesis, suposiciones.
- Pasos: coleccionar/concatenar, explorar, procesar, reportar.
- **Más un arte que una ciencia**.

Localización y dispersión robustas

- Media y varianza: sensibles a outliers.
- Mediana e IQR/MAD: opciones robustas por defecto.

$$IQR = Q_3 - Q_1, \quad \text{Outlier} : x \notin [Q_1 - 1.5\,IQR,\ Q_3 + 1.5\,IQR]$$

Regla práctica

Reporte siempre media$\pm$DE y mediana/IQR.

Forma y dependencia

- Asimetría y curtosis: forma de la distribución.
- Pearson/Spearman: lineal y monótona, respectivamente.
- Información mutua: dependencias no lineales.

$$I(X;Y) = \sum_{x,y} p(x,y) \log \frac{p(x,y)}{p(x)p(y)}$$

Estadística descriptiva: visión general

- **Generales**: `df.head()`, `df.shape`, `df.info()`
- **Univariada numérica**: media, varianza, histograma (`df.describe()`)
- **Univariada categórica**: moda, frecuencias, % únicos.
- **Variable objetivo**: distribución de clases.

Código: numéricas y categóricas

```
# Característica numérica
import matplotlib.pyplot as plt
df[num_feature].plot.hist(bins=7)
plt.show()

# Característica categórica
import matplotlib.pyplot as plt

df[cat_feature].value_counts().plot.bar()
plt.show()
```

Estadística de la variable objetivo

- `df[target].value_counts()` para distribución de clases.
- `np.bincount(y)` alternativa para y numérico.
- Verificar desbalance y rareza de clases.

Dependencias y correlaciones

- Dispersión: `df.plot.scatter('f1','f2')`.
- Matriz de correlación: `df[cols].corr()`.
- Rango: $[-1, 1]$; 0 indica linealmente independiente.
- Multicolinealidad afecta regresiones (inversas de matrices).
- Árboles son menos sensibles a colinealidad.
- Correlación alta con el objetivo ayuda a lineales.

Visualización efectiva

- Aproveche canales preatentivos: forma, tamaño, posición.
- Facetar por grupos; evite *chartjunk*.
- Seleccione escalas lineales o log según rango.

Gramática de gráficos

Datos, estéticas, geometrías, facetas, estadísticas y temas.

Bases de datos desbalanceadas

Desbalance: definición y ejemplos

- Distribución de clases no equitativa.
- Riesgo: ignorar la clase rara.
- Casos: fraude, anomalías, diagnóstico médico.
- Ej.: *Amazon reviews*: 5 estrellas domina.

Cómo manejar el desbalance

- Ajustes **sólo** en *train*; dev/test conservan distribución.
- Submuestreo de la clase mayoritaria.
- Remuestreo de clases minoritarias.
- Generación sintética (p.ej., SMOTE/ADASYN).
- Ponderar la pérdida: pesos por clase/muestra.
- Recurso: imbalanced-learn.

Ingeniería de características

Ingeniería de características: principios

- Usa conocimiento del dominio para nuevas variables.
- **Intuición**: ¿Qué usaría un humano para predecir?
- Selección: filtrar irrelevantes y redundantes.
- Construcción: productos, polinomios, logaritmos, kernels.
- Codificación: representar categóricas numéricamente.

Imputación y escalado

Estrategias de imputación

- **Descartar** filas/columnas (cuidado con pérdida de señal).
- **Imputar** numéricas: media/mediana; categóricas: moda.
- **Placeholder**: valor constante reservado.
- **Avanzado**: imputación con ML (p.ej., Datawig).
- Código: `df['col'].fillna(df['col'].mean())`, `df['col'].fillna(df['col'].mode())`.

SimpleImputer en `sklearn`

- `SimpleImputer(missing_values=np.nan, strategy='mean', fill_value=None)`.
- Numéricas: `strategy='mean'` o `'median'`.
- Numéricas/Categóricas: `'most_frequent'` o `'constant'`.
- Métodos: `.fit()` aprende; `.transform()` aplica; `.fit_transform()`.

Estandarización de características

- Motivación: kNN, redes, distancias dependen de escala.
- **StandardScaler**: media 0 y DE 1 por columna.
- **MinMaxScaler**: mapea a $[0, 1]$ por columna.
- Siempre *fit* en *train*; *transform* en dev/test.
- *Standard* vs. *MinMax* vs. *Robust*.
- Box–Cox y log para positividad y colas.
- *Quantile* para normalización marginal.

Codificación (encoding)

- **Ordinal**: categorías con orden (ej.: talla S<M<L).
- **Nominal**: sin orden (ej.: color {green, red, blue}).
- Cuidado: enteros ordinales pueden inducir órdenes ficticios.

Codificación categórica

- One–hot: ortogonalidad y simplicidad.
- Eliminar categoría de referencia en modelos lineales.
- Target encoding con *shrinkage* para alta cardinalidad.

$$\widehat{y}_c = \frac{n_c \bar{y}_c + \lambda \bar{y}}{n_c + \lambda}$$

LabelEncoder y OrdinalEncoder

- `LabelEncoder`: codifica una sola columna o el objetivo.
- `OrdinalEncoder`: codifica múltiples columnas categóricas.

Código: OrdinalEncoder

```python
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
y_enc = le.fit_transform(df["classlabel"])

# Inversa:
y_inv = le.inverse_transform(y_enc)

# Mapeo etiqueta->id:
classes = {lab:i for i, lab in enumerate(le.classes_)}
```

Código: OrdinalEncoder

```
from sklearn.preprocessing import OrdinalEncoder

oe = OrdinalEncoder(
                    categories=[["blue","green","red"], ["XS","S","M","L","XL"]],
                    handle_unknown="use_encoded_value", unknown_value=-1
        )
X_ord = oe.fit_transform(df[["color","size"]])
```

One–Hot Encoding

- Evita orden artificial de enteros ordinales.
- Expande en variables binarias por categoría.
- `OneHotEncoder` (`handle_unknown='ignore'`).
- Pandas: `get_dummies`.
- Para una sola variable, `LabelBinarizer`.

Demasiadas categorías

- Jerarquías: región $\rightarrow$ estado $\rightarrow$ ciudad.
- Elegir nivel apropiado de agregación.
- Agrupar en *bins* (ej.: grupos de edades).

Codificación usando la variable objetivo

- Reemplazar categorías por la media del objetivo (con *shrinkage*).
- Úsese con CV para evitar *leakage*.

Interacciones y no linealidades

- Polinomios e interacciones controladas.
- Fourier para estacionalidad y periodicidad.
- Autoencoders para representación compacta.

Evitar *data leakage*

- **Target leakage**: rasgos derivados de Y.
- **Contaminación**: estadísticas de test en preprocesado.
- **Temporal**: usar información futura sin intención.

Regla de oro

Calcular estadísticas *dentro* de cada partición de CV.

Pipelines y *sklearn*

Pipeline en `sklearn`

- Secuencia de transformaciones que termina en estimador.
- Implementa `.fit()` y `.predict()`.
- Reduce *leakage*: las estadísticas se aprenden sólo en *train*.

Ejemplo de `Pipeline` en Python

```
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
from sklearn.neighbors import KNeighborsClassifier

knn_pipe = Pipeline([
                    ("imputer", SimpleImputer(strategy="median")),
                    ("scaler", MinMaxScaler()),
                    ("clf", KNeighborsClassifier(n_neighbors=5)),
            ])
knn_pipe.fit(X_train, y_train)
y_pred = knn_pipe.predict(X_test)
```

Atajo con make_pipeline

```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression

pipe = make_pipeline(StandardScaler(),
                     LogisticRegression(max_iter=1000))
pipe.fit(X_train, y_train)
y_proba = pipe.predict_proba(X_new)
```

Transformers comunes

- `SimpleImputer`, `StandardScaler`, `MinMaxScaler`.
- `LabelEncoder`, `OrdinalEncoder`, `OneHotEncoder`.
- `CountVectorizer` para texto.
- Todos: `.fit()`, `.transform()`, `.fit_transform()`.

ColumnTransformer en `sklearn`

- Aplica transformaciones por subconjunto de columnas.
- Concatena salidas en una sola matriz de rasgos.
- Métodos: `.fit()` y `.transform()`.

ColumnTransformer + Pipeline (ejemplo)

```
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.neighbors import KNeighborsClassifier

num_cols, cat_cols = ["feature1","feature3"], ["feature0","feature2"]

num_proc = Pipeline([("imp", SimpleImputer(strategy="median")),
                                    ("scaler", MinMaxScaler())])
                                    cat_proc = Pipeline([("imp", SimpleImputer(strategy="constant",
                                                                               fill_value="missing")),
                                    ("ohe", OneHotEncoder(handle_unknown="ignore",
                                                                                  sparse_output=False))])

pre = ColumnTransformer([("num", num_proc, num_cols),
                                               ("cat", cat_proc, cat_cols)])

model = Pipeline([("prep", pre),
                                ("clf", KNeighborsClassifier(n_neighbors=5))])

model.fit(X_train, y_train)
y_hat = model.predict(X_test)
```

Síntesis y referencias

Lo esencial para llevar

- EDA riguroso antes de cualquier modelado.
- Rasgos bien diseñados superan modelos exóticos.
- Validación honesta gana a métricas infladas.
- *Pipelines* versionados y monitoreados.
- Iterar con propósito: negocio primero, siempre.

Referencias y lecturas recomendadas I

- Tukey, *Exploratory Data Analysis* (1977).
- Cleveland, *Visualizing Data* (1993).
- Wilkinson, *The Grammar of Graphics* (2005).
- Cover & Thomas, *Elements of Information Theory* (2006).
- Hastie, Tibshirani, Friedman, *ESL* (2009).
- Kuhn & Johnson, *Feature Engineering and Selection* (2019).
- scikit–learn, *Preprocessing Module* (docs).
- imbalanced-learn (lib).
- Datawig (imputación con NN).

¡Muchas gracias por su atención!

¿Preguntas?

Contacto: Marco Teran
webpage: marcoteran.github.io/
e-mail: mtteranl@eafit.edu.co