Universidad de SanAndrés

Clase 2:
Análisis exploratorio y procesamiento de datos

Analítica de Datos

Maestría en Ciencias del Comportamiento

Primavera 2026

¿Quiénes dejan una empresa, y por qué?

Un caso real de *people analytics*: rotación de personal (dataset HR Attrition, 1470 empleados).

Edad	Departamento	Puesto	Ingreso	Horas extra	Se fue?
41	Ventas	Sales Executive	$5,993	Si	**Si**
49	I+D	Research Scientist	$5,130	No	**No**
37	I+D	Laboratory Technician	$2,090	Si	**Si**
33	I+D	Research Scientist	$2,909	Si	**No**
27	I+D	Laboratory Technician	$3,468	No	**No**
32	I+D	Laboratory Technician	$3,068	No	**No**

De qué vamos a hablar hoy

Universidad de SanAndrés

Objetivo

Aprender los **fundamentos de Python** para el análisis de datos y recorrer, de punta a punta, el **análisis exploratorio (EDA)** sobre datos reales.

1. **Fundamentos de Python y carga de datos.** Variables, estructuras y funciones; del archivo al DataFrame.
2. **Explorar.** Tipos, estructura y estadísticos descriptivos.
3. **Procesar y limpiar.** Filtrar y agrupar; tratar faltantes, duplicados y outliers.
4. **Transformar.** Crear variables con sentido (*feature engineering*) y fijar el vocabulario.

Para las prácticas vamos a ir a **Colab**.

BLOQUE 1 DE 4

Fundamentos de Python y carga de datos

Del lenguaje a la primera tabla de datos.

Python, con R al lado

Nos apoyamos en lo que ya saben: cada operación tiene su equivalente en R.

en R (tidyverse)	en Python (pandas)	qué hace
`x <- 42`	`x = 42`	asignar una variable
`c(29, 41, 35)`	`[29, 41, 35]`	colección ordenada
`list(a = 1)`	`{"a": 1}`	colección con nombres
`for (x in xs) { }`	`for x in xs:`	recorrer
`if (c) { } else { }`	`if c: ... else:`	condicional
`f <- function(x) { }`	`def f(x):`	definir una función

Ojo: en Python la **indentación** (no las llaves) define los bloques, y se cuenta **desde 0**.

Variables, tipos y estructuras

```
edad = 34                  # int   (entero)
ingreso = 4919.0           # float (decimal)
puesto = "Sales Exec"      # str   (texto)
renuncio = False           # bool  (True / False)

# una lista de diccionarios ya ES una tabla en miniatura
equipo = [{"nombre": "Ana",   "renuncio": True},
          {"nombre": "Bruno", "renuncio": False}]
```

- El **tipo** determina qué podés hacer con el valor.
- **Lista**: valores en orden (el primero es el `[0]`). **Diccionario**: pares *nombre: valor*, una observación.

Control de flujo y funciones

```python
def tasa_de_renuncia(personas):
    renuncias = 0
    for p in personas:           # recorrer la coleccion
        if p["renuncio"]:        # decidir segun una condicion
            renuncias += 1
    return renuncias / len(personas)
```

- `for` recorre, `if` decide, `def` empaqueta un cálculo para **reutilizarlo**.
- Idea que usaremos todo el tiempo: la **media de una columna de True/False es una proporción**.

Del archivo al DataFrame

```
import pandas as pd
url = "https://.../emp_attrition.csv"
df = pd.read_csv(url)      # tambien read_excel(...), read_parquet(...)
df.shape                   # (1470, 35): filas x columnas
```

- Un **DataFrame** es una tabla: cada **fila** una observación, cada **columna** una variable.
- Cargamos desde una **URL** pública para que funcione en Colab sin archivos locales.

Importar y exportar: CSV, Excel, parquet

```
df = pd.read_csv(url)              # tambien read_excel(...), read_parquet(...)
df.to_csv("hr.csv", index=False)   # guardar: to_excel(...), to_parquet(...)
```

- Por cada `read_*` hay un `to_*`: cargar y guardar son **simétricas**.
- **CSV** (universal), **Excel** (planillas), **parquet** (columnar: rápido y liviano para datos grandes).
- `read_csv` tiene parámetros para archivos sucios: `sep`, `na_values`, `usecols`, `encoding`.

A la notebook

Ejercicio 1: clasificar ingresos con una función y un `for`.

Practicamos los fundamentos y cargamos el dataset, en Python o en R.

BLOQUE 2 DE 4

Explorar: tipos, estructura y descriptivos

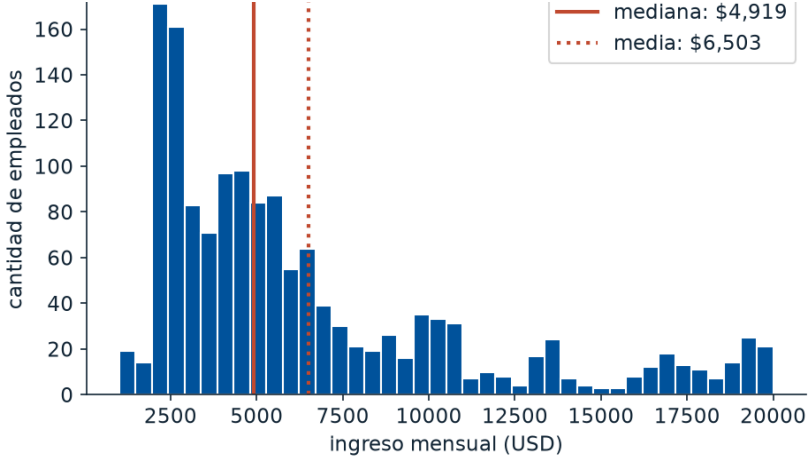
Antes de modelar, mirar; antes de mirar, resumir.

Primer vistazo al dataset

```
df.shape            # (1470, 35): 1470 empleados, 35 variables
df.info()           # tipo y no-nulos de cada columna
df["Attrition"].value_counts()     # No: 1233 | Yes: 237
```

- Leer los **tipos** es el primer chequeo de calidad: 26 columnas numéricas y 9 de texto.
- Una variable numérica que aparece como *texto* es señal de **datos sucios**.

La mitad gana menos de $4.919: la media la infla la cola



Distribución del ingreso mensual (n = 1470). **Mediana $4.919** frente a **media $6.503**: la cola de sueldos altos empuja la media.

La trampa: media frente a mediana

Cuidado

Cuando la distribución es **asimétrica**, la **media** deja de describir al caso “típico”.

- Los ingresos tienen **asimetría a derecha**: unos pocos sueldos altos empujan la media hacia arriba.
- Para el ingreso “típico” se reporta la **mediana** ($4.919), no la media ($6.503).
- Regla general: **mirá la distribución**, no solo un promedio.

BLOQUE 3 DE 4

Procesar y limpiar con pandas

Filtrar, agrupar, y dejar los datos listos para modelar.

Los verbos de todos los días

El 80 % del trabajo real son cuatro verbos que ya usaron en `dplyr`.

verbo	en dplyr (R)	en pandas
elegir columnas	`select(df, a, b)`	`df[["a", "b"]]`
filtrar filas	`filter(df, cond)`	`df[cond]`
ordenar	`arrange(df, desc(a))`	`df.sort_values("a", ascending=False)`
renombrar	`rename(df, n = viejo)`	`df.rename(columns={"viejo": "n"})`
agrupar y resumir	`group_by() + summarise()`	`df.groupby("g")["x"].median()`

El filtro es una **máscara booleana**: `df["OverTime"] == "Yes"` da una columna de True/False, y `df[mascara]` se queda con las filas True.

Seleccionar con `loc` e `iloc`

```
df.loc[df["OverTime"]=="Yes", ["JobRole","MonthlyIncome"]]  # por etiqueta
df.iloc[:5, :3]                                             # por posicion (desde 0)
df["Age"]                                                   # es una Series
```

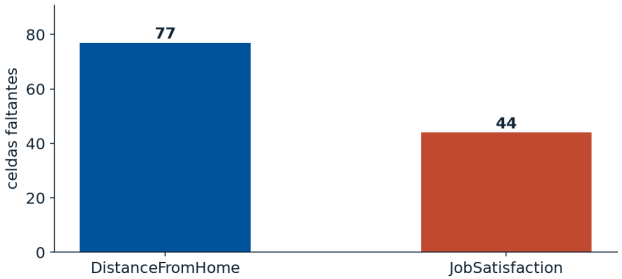
- `loc`: por **etiqueta** (nombres, índice). `iloc`: por **posición** entera (desde 0).
- Una sola columna es una **Series** (un vector 1D con índice), el bloque con el que se arma el DataFrame.

Limpieza estructural

```python
# columnas que valen lo mismo en las 1470 filas: no aportan
constantes = [c for c in df.columns if df[c].nunique() == 1]
df = df.drop(columns=constantes)  # EmployeeCount, Over18, StandardHours
df.duplicated().sum()             # 0 (en datos reales suelen aparecer)
```

- Una columna **constante** no distingue a nadie: fuera.
- **Duplicados** exactos: `duplicated()` para detectarlos, `drop_duplicates()` para sacarlos.

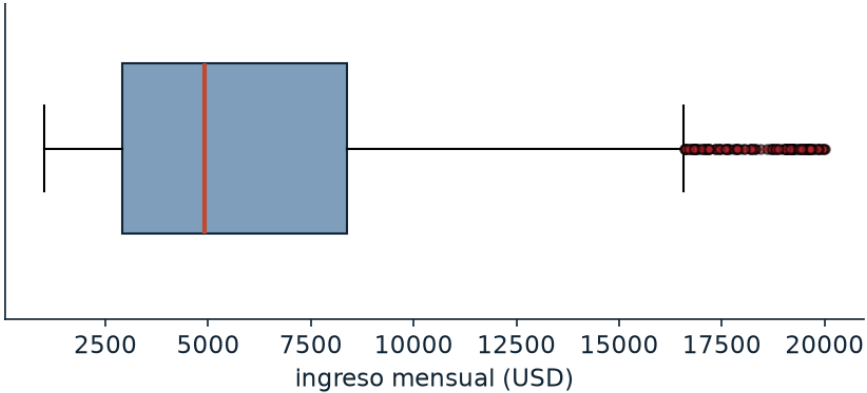
Faltantes: primero detectar, después decidir



- **Eliminar** (`dropna`): simple, pero tira información.
- **Imputar** (`fillna`): rellenar con un valor razonable (mediana para numéricas, moda para categorícas).

Faltantes introducidos a propósito para practicar (el dataset viene completo). La decisión depende de cuánto y por qué falta.

Los 114 ingresos atípicos: gerentes, no errores



Regla del IQR: atípico si supera $Q_3 + 1{,}5 \times IQR$. Los 114 puntos rojos son de nivel jerárquico 4 y 5: **atípico no es sinónimo de error**.

A la notebook

Ejercicios 2 y 3: tasa de rotación por puesto, e imputación por grupo.

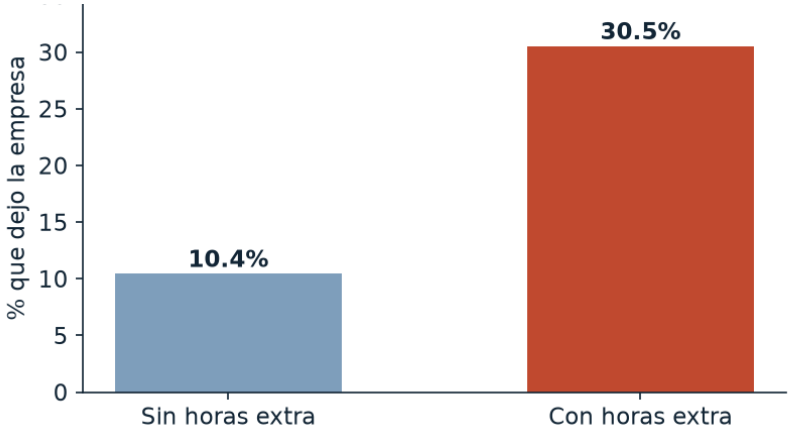
Procesar con pandas y tratar faltantes, en Python o en R.

BLOQUE 4 DE 4

Transformar: crear variables y nombrar

Del dato limpio a variables con sentido para el análisis.

Con horas extra, la rotación casi se **triplica**



Tasa de attrition según horas extra: **10,4 %** sin, **30,5 %** con (n = 1470). Un `groupby` + la media de una columna booleana responde cualquier “tasa por grupo”.

Feature engineering: crear variables con sentido

```
# ingreso por anio de experiencia (dos trayectorias, mismo sueldo)
df["IngresoPorAnioExp"] = df["MonthlyIncome"] / (df["TotalWorkingYears"] + 1)

# discretizar la edad en tramos con etiqueta
df["TramoEdad"] = pd.cut(df["Age"], bins=[17, 30, 40, 50, 61],
                         labels=["18-30", "31-40", "41-50", "51-60"])
```

- Cuando ninguna columna captura el fenómeno, se **crea** una nueva.
- La justificación es el contexto (el dominio), no el algoritmo.

Un vocabulario común

término	qué es
datos crudos	la fuente original, intacta
limpieza	corregir errores, duplicados, tipos
procesamiento	manipular: filtrar, agrupar, resumir
preprocesamiento	lo que pide el algoritmo (estandarizar, imputar)
feature engineering	crear variables con sentido sustantivo
feature selection	elegir qué variables entran al modelo

No es pedantería: en la Clase 4, confundir *preprocesamiento* con *procesamiento* puede invalidar un modelo (*data leakage*).

A la notebook

Ejercicio 4: el costo de viajar seguido (*feature engineering*).

Crear variables con sentido, en Python o en R.

Lo que nos llevamos

Universidad de SanAndrés

Ya podés hacer un EDA de punta a punta

Cargar datos desde la nube, explorarlos, limpiarlos (faltantes, duplicados, outliers) y transformarlos (*feature engineering*), en Python o en R.

El hallazgo de hoy, en una frase: en esta empresa la rotación se concentra en quienes

- hacen **horas extra** (30,5 % frente a 10,4 %),
- **viajan seguido** (24,9 %),
- y trabajan en **ventas** (39,8 % entre representantes).

En la Clase 4 vamos a intentar **predecir** esa rotación.

Lo que viene

- **Clase 3:** visualización de datos y reportes reproducibles (sobre este mismo dataset).
- **Lectura:** McKinney, *Python for Data Analysis*, caps. 5 a 7 (gratis online).
- **Practicar:** las dos notebooks de hoy (Python y R) están en el sitio de la materia.

¿Preguntas?