Business Analytics & Data Science

Día 3: Limpieza de Datos y Visualización

EAE Business School Barcelona

4 de febrero de 2026

Plan del Día

Primera Parte (9:00-11:00)

1. Calidad de datos: identificar problemas
2. Limpieza de datos: valores faltantes, duplicados, valores imposibles
3. Transformación de datos: nuevas columnas, ordenar

Segunda Parte (11:30-13:30)

1. Operaciones GroupBy
2. Por qué visualizar: Anscombe's Quartet
3. Visualización con Plotly Express

Repaso Rápido: Día 2

¿Qué vimos ayer?

- **Listas y diccionarios**: estructuras de datos básicas
- **pandas**: librería para trabajar con datos tabulares
- **DataFrames**: tablas con filas y columnas
- **Inspección**: `head()`, `info()`, `describe()`
- **Filtrado**: condiciones booleanas con `&` y `|`
- **Indexación**: `loc` (etiquetas) e `iloc` (posiciones)

Primera Parte: Calidad y Limpieza de Datos

La Realidad de los Datos

Mito: Los datos vienen limpios y listos para analizar

Realidad: Los datos del mundo real son desordenados

El 80% del tiempo de análisis de datos se gasta en limpieza

La limpieza de datos no es *glamourosa* pero es CRÍTICA para un buen análisis

Problemas comunes:

- **Valores faltantes** (NaN, NULL, vacíos)
- **Duplicados** (registros repetidos)
- **Valores imposibles** (edades negativas, precios de 0€)
- **Formatos inconsistentes** (fechas, texto)
- **Errores de entrada** (typos, códigos incorrectos)

Patrón de Calidad de Datos

Cuando trabajéis con datos nuevos, seguid este proceso:

1. **Identificar** → ¿Qué problemas hay?
2. **Clasificar** → ¿Qué tipo de problema? (faltante, duplicado, imposible)
3. **Decidir** → ¿Qué hacer? (eliminar, imputar, corregir)
4. **Documentar** → ¿Qué decisiones tomamos y por qué?

NUNCA eliminar datos sin entender por qué faltan o son raros

Valores Faltantes: Detectar

```
# Contar por columna
df.isnull().sum()

# Porcentaje faltante
(df.isnull().sum() / len(df)) * 100

# Ver filas con NaN
df[df.isnull().any(axis=1)]
```

Valores Faltantes: Estrategias

```
# Eliminar filas
df_clean = df.dropna()

# Eliminar columnas
df_clean = df.dropna(axis=1)

# Rellenar con media
df["price"] = df["price"].fillna(df["price"].mean())

# Rellenar con mediana
df["rooms"] = df["rooms"].fillna(df["rooms"].median())
```

Duplicados

Detectar...

```
# ¿Hay duplicados?
duplicados = df.duplicated()
print(f"Filas duplicadas: {duplicados.sum()}")

# Ver duplicados
df[df.duplicated(keep=False)]
```

...y eliminar:

```
# Eliminar (mantiene primera ocurrencia)
df_clean = df.drop_duplicates()

# Basado en columnas específicas
df_clean = df.drop_duplicates(subset=["id"])
```

Valores Imposibles

Decidir que un valor es imposible depende de la **lógica de negocio**; qué hacer con ellos también. Algunos ejemplos:

```
# Precios negativos
df[df["price"] < 0]

# Habitaciones = 0
df[df["rooms"] == 0]

# Outliers extremos
df[df["sqrmts"] > 1000]  # 1000m² para un piso?
```

Transformación de Datos: Columnas Derivadas

```
# Precio por m²
df["price_per_sqm"] = df["price"] / df["sqrmts"]

# Categorizar con pd.cut
df["price_category"] = pd.cut(df["price"],
                              bins=[0, 200000, 400000, 1000000],
                              labels=["Bajo", "Medio", "Alto"])
```

Transformación de Datos: Funciones Personalizadas

```python
# Definir función personalizada
def categorize_size(sqm):
    if sqm < 50:
        return "Pequeño"
    elif sqm < 100:
        return "Mediano"
    else:
        return "Grande"

# Aplicar a columna
df["size"] = df["sqrmts"].apply(categorize_size)
```

Ordenar Datos

```
# Ordenar por columna
df_sorted = df.sort_values("price")

# Orden descendente
df_sorted = df.sort_values("price", ascending=False)

# Múltiples columnas
df_sorted = df.sort_values(["neighborhood", "price"])

# Resetear índice
df_sorted = df.sort_values("price").reset_index(drop=True)
```

Operaciones GroupBy

GroupBy = "split-apply-combine"

GroupBy responde: "¿Cuál es el [métrica] por [categoría]?"

```
# Precio medio por barrio
df.groupby("neighborhood")["price"].mean()

# Múltiples agregaciones
df.groupby("neighborhood")["price"].agg(["mean", "median", "count"])
```

```python
# Múltiples columnas
df.groupby(["neighborhood", "type"])["price"].mean()

# Como DataFrame
precio_barrio = df.groupby("neighborhood")["price"].mean().reset_index()
```

🔬 Ahora al notebook

Ejercicio Limpieza de Datos

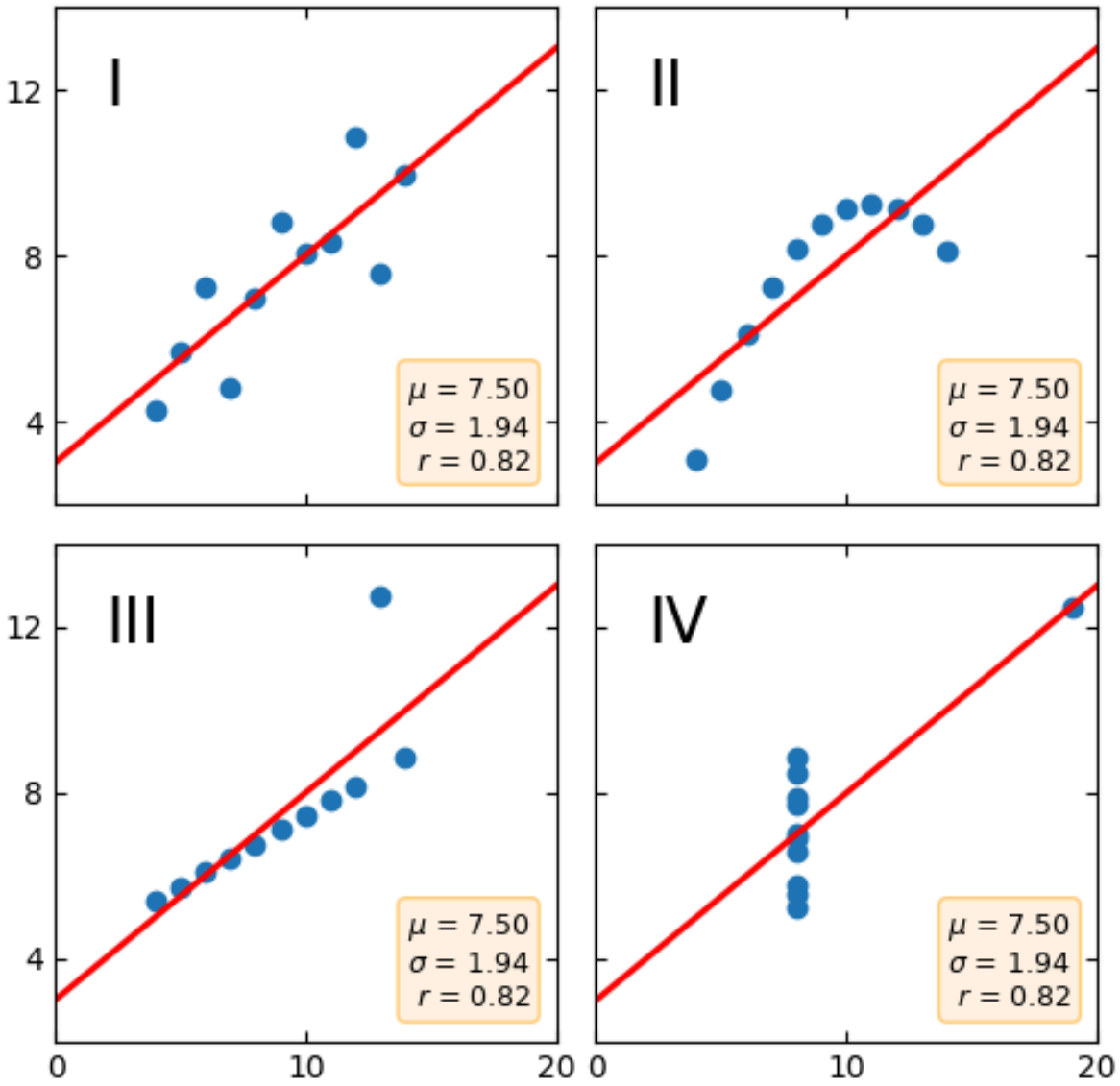
Segunda Parte: Visualización

¿Por Qué Visualizar?

Veamos el cuarteto de Anscombe.

Se trata de 4 datasets con las **mismas** estadísticas: media, desviación, correlación y recta de regresión lineal.

No obstante, las distribuciones son **muy** diferentes.


I
II
III
IV
$\mu = 7.50$
$\sigma = 1.94$
$r = 0.82$
12
8
4
0
10
20

Plotly Express

API simple para gráficos interactivos

¿Por qué Plotly?

- ✅ Interactivo (zoom, pan, hover)
- ✅ API consistente
- ✅ Funciona perfecto en Colab
- ✅ Visualizaciones profesionales con poco código

```python
import plotly.express as px

# Scatter plot simple
fig = px.scatter(df, x="sqrmts", y="price")
fig.show()
```

Scatter Plot: Básico

```
# Básico
fig = px.scatter(df, x="sqrmts", y="price")
fig.show()

# Con color por categoría
fig = px.scatter(df, x="sqrmts", y="price",
                 color="neighborhood")
fig.show()
```

Scatter plots revelan: correlaciones, outliers, clusters

Scatter Plot: Opciones Avanzadas

```
# Con tamaño de punto
fig = px.scatter(df, x="sqrmts", y="price",
                 color="neighborhood", size="rooms")
fig.show()

# Con información adicional en hover
fig = px.scatter(df, x="sqrmts", y="price",
                 color="neighborhood",
                 hover_data=["type", "rooms"])
fig.show()
```

Bar Chart

```
# Contar por barrio
counts = df["neighborhood"].value_counts().reset_index()
counts.columns = ["neighborhood", "count"]

fig = px.bar(counts, x="neighborhood", y="count")
fig.show()
```

Bar Chart: Agregaciones

```
# Precio medio por barrio
avg_price = df.groupby("neighborhood")["price"].mean().reset_index()
fig = px.bar(avg_price, x="neighborhood", y="price")
fig.show()

# Barras horizontales
fig = px.bar(avg_price, x="price", y="neighborhood",
             orientation="h")
fig.show()
```

Histogram

```
# Distribución de precios
fig = px.histogram(df, x="price", nbins=50)
fig.show()

# Con color por tipo
fig = px.histogram(df, x="price", color="type", nbins=30)
fig.show()
```

Histograms revelan: simetría, sesgos, multimodalidad, outliers

Box Plot

```
# Box plot simple
fig = px.box(df, y="price")
fig.show()

# Por categoría
fig = px.box(df, x="neighborhood", y="price")
fig.show()
```

Box plot muestra:

- Mediana (línea central)
- Cuartiles (caja)
- Whiskers (rango típico)
- Outliers (puntos)

🔬 Ahora al notebook

Ejercicio Visualización (30 minutos)

Resumen del Día 3

- ✅ Calidad de datos: identificar → clasificar → decidir → documentar
- ✅ Valores faltantes: `dropna()`, `fillna()`
- ✅ Duplicados: `drop_duplicates()`
- ✅ Transformaciones: nuevas columnas, `apply()`
- ✅ GroupBy: agregaciones por categorías
- ✅ Plotly Express: scatter, bar, histogram, box

Gracias!