Guía Profesional de Limpieza y Análisis Exploratorio de Datos

Domina las herramientas clave para transformar datos crudos en conocimiento útil.

20 de abril de 2025

Palabras clave:
pandas data cleaning, exploratory data analysis with Python, data preprocessing pandas, EDA python notebook, missing values handling pandas, outlier detection python, data wrangling pandas, data science portfolio project, feature engineering pandas, data cleaning cheat sheet.

Índice

Nivel Básico **2**

Nivel Intermedio **3**

Nivel Avanzado **5**

Creación de Nuevas Columnas **6**

Análisis Exploratorio de Datos (EDA) **6**

Nivel Básico

Importar y cargar datos

Antes de analizar datos, es importante saber cómo cargarlos desde diferentes fuentes. Aquí usamos archivos CSV, Excel y bases de datos SQLite.

```
# Importamos las bibliotecas necesarias
import pandas as pd
import sqlite3

# Leer un archivo CSV y cargarlo como DataFrame
# Muy común para datasets estructurados
df_csv = pd.read_csv('archivo.csv')

# Leer un archivo Excel
# Requiere tener instalada la librería openpyxl
df_excel = pd.read_excel('archivo.xlsx')

# Conectar a una base de datos SQLite y leer una tabla
# Útil cuando los datos están en bases relacionales
db = sqlite3.connect('mi_base.db')
df_sql = pd.read_sql_query("SELECT * FROM tabla", db)
db.close()
```

Exploración de datos

Una vez cargados los datos, es fundamental inspeccionarlos para entender su estructura, contenido y calidad.

```
# Ver las primeras 5 filas del DataFrame (muestra inicial)
df.head()

# Ver las últimas 5 filas (útil para ver datos recientes)
df.tail()

# Ver el número de filas y columnas (dimensiones)
df.shape

# Mostrar nombres de columnas (cabeceras)
df.columns

# Mostrar tipos de datos por columna (int, float, object, etc)
df.dtypes

# Información general del DataFrame (memoria, tipos y nulos)
df.info()

# Estadísticas descriptivas de columnas numéricas
df.describe()

# Conteo de valores únicos en una columna categórica
df['columna'].value_counts()
```

Tipos de datos y conversiones

Asegúrate de que las columnas estén en el formato correcto. Esto facilita el análisis posterior y evita errores.

```python
# Convertir columna a formato de fecha (timestamp)
df['fecha'] = pd.to_datetime(df['fecha'])

# Convertir columna numérica a entero
df['edad'] = df['edad'].astype(int)

# Convertir columna a booleano (True/False)
df['bool_col'] = df['bool_col'].astype(bool)
```

Limpieza básica de texto

Las columnas de texto muchas veces vienen desordenadas: mayúsculas, espacios, símbolos, etc. Esta limpieza es básica pero poderosa.

```python
# Convertir todo a minúsculas y quitar espacios al inicio/final
df['col'] = df['col'].str.lower().str.strip()

# Eliminar caracteres que no sean letras ni espacios
df['col'] = df['col'].str.replace(r'[^a-zA-Z ]', '', regex=True)

# Reemplazar múltiples espacios por uno solo (normalización)
df['col'] = df['col'].str.replace(r'\s+', ' ', regex=True)
```

Detección y eliminación de valores faltantes

Los valores nulos o vacíos son comunes. Aquí vemos cómo identificarlos y manejarlos.

```python
# Contar cuántos valores nulos hay por columna
df.isnull().sum()

# Eliminar filas que tengan al menos un valor nulo
df.dropna()

# Rellenar todos los valores nulos con 0
df.fillna(0)

# Rellenar valores nulos en una columna con la media
df['col'].fillna(df['col'].mean(), inplace=True)
```

Detección y eliminación de duplicados

Evita que registros repetidos contaminen tu análisis.

```python
# Detectar qué filas son duplicadas (retorna True/False)
df.duplicated()

# Eliminar duplicados, conservando solo la primera aparición
df.drop_duplicates()
```

Nivel Intermedio

Outliers

Los outliers (valores atípicos) son datos que se alejan significativamente del resto y pueden distorsionar los análisis. Se pueden detectar con métodos estadísticos como el IQR o la desviación estándar.

```
# Método IQR (Interquartile Range)
Q1 = df['col'].quantile(0.25)
Q3 = df['col'].quantile(0.75)
IQR = Q3 - Q1
outliers = df[(df['col'] < Q1 - 1.5*IQR) | (df['col'] > Q3 + 1.5*IQR)]

# Método de 3 desviaciones estándar desde la media
media = df['col'].mean()
std = df['col'].std()
outliers = df[(df['col'] < media - 3*std) | (df['col'] > media + 3*std)]
```

Mapeo y reemplazo de valores

Reemplazar valores específicos en una columna es útil para normalizar entradas (como codificaciones o abreviaciones).

```
# Mapeo con diccionario simple (ideal para categorías pequeñas)
mapeo = {'m': 'masculino', 'f': 'femenino'}
df['sexo'] = df['sexo'].map(mapeo)

# Reemplazo manual de valores (más flexible)
df['ciudad'] = df['ciudad'].replace({'cdmx': 'Ciudad de México'})
```

Actualización condicional

Es común necesitar cambiar los valores de una columna según una condición lógica.

```
# Si la edad es mayor a 30, asignar 'adulto mayor' a la columna 'grupo'
df.loc[df['edad'] > 30, 'grupo'] = 'adulto mayor'

# Crear una nueva columna condicional: 'adulto' o 'menor'
df['etiqueta'] = np.where(df['edad'] >= 18, 'adulto', 'menor')
```

Visualización

Visualizar datos ayuda a entender su distribución y detectar patrones, sesgos o valores atípicos.

```
import matplotlib.pyplot as plt
import seaborn as sns

# Histograma: muestra la distribución de frecuencias
plt.hist(df['columna'], bins=10)
plt.show()

# Boxplot: ideal para ver la mediana, cuartiles y outliers
sns.boxplot(x=df['columna'])
plt.show()
```

Manejo de fechas

Las fechas deben convertirse a formato datetime para poder extraer partes útiles como el año o el mes.

```
# Convertir a datetime
df['fecha'] = pd.to_datetime(df['fecha'])

# Extraer año y mes
df['año'] = df['fecha'].dt.year
```

```
df['mes'] = df['fecha'].dt.month
```

Codificación de variables

Las variables categóricas deben ser transformadas en números para modelos de Machine Learning.

```
# Codificación con LabelEncoder (0, 1, 2...)
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
df['categoria_cod'] = encoder.fit_transform(df['categoria'])
```

Nivel Avanzado

Imputación avanzada

Cuando los valores faltantes no pueden eliminarse ni imputarse con valores simples, se puede usar KNNImputer. Este algoritmo busca valores similares y predice los faltantes en base a sus vecinos más cercanos.

```
from sklearn.impute import KNNImputer
imputer = KNNImputer(n_neighbors=3)
df[['col1', 'col2']] = imputer.fit_transform(df[['col1', 'col2']])
```

Limpieza por lógica de negocio

A veces la limpieza depende de reglas específicas del contexto. Por ejemplo, si un cliente no tiene actividad registrada, su saldo debería ser cero.

```
df.loc[(df['actividad'] == 'ninguna') & (df['saldo'] > 0), 'saldo'] = 0
```

Validación de formatos

Comprobar que los valores tengan un formato válido es esencial, sobre todo en datos como correos electrónicos, códigos o identificadores.

```
df['email_valido'] = df['email'].str.contains(r'^[\w\.-]+@[\w\.-]+$', regex
    =True)
```

Perfilado automático

Para una visión general del dataset sin escribir mucho código, puedes usar 'ydata-profiling' (antes pandas-profiling). Genera un informe HTML interactivo.

```
from ydata_profiling import ProfileReport
profile = ProfileReport(df)
profile.to_file('informe.html')
```

Preprocesamiento con Pipeline

Combinar pasos de procesamiento (como imputación y escalado) en un solo pipeline es clave para preparar datos antes de entrenar modelos de Machine Learning.

```
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler

pipeline = Pipeline([
```

```
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())
])
df[['edad', 'ingresos']] = pipeline.fit_transform(df[['edad', 'ingresos']])
```

Creación de Nuevas Columnas

Columnas numéricas

```
# Crear una columna nueva como suma de otras dos
df['total'] = df['ventas'] + df['impuestos']

# Operaciones condicionales
df['es_grande'] = df['total'] > 1000
```

Columnas datetime

```
# Extraer año, mes, día de una fecha existente
df['fecha'] = pd.to_datetime(df['fecha'])
df['año'] = df['fecha'].dt.year
df['mes'] = df['fecha'].dt.month
df['dia_semana'] = df['fecha'].dt.day_name()
```

Columnas de texto

```
# Crear columnas basadas en transformación de texto
df['nombre_completo'] = df['nombre'] + ' ' + df['apellido']
df['inicial'] = df['nombre'].str[0].str.upper()
```

Análisis Exploratorio de Datos (EDA)

Filtrado, ordenamiento y agrupamiento

```
# Filtrar por condición
df_filtrado = df[df['ventas'] > 100]

# Ordenar de forma descendente
df.sort_values('ventas', ascending=False)

# Agrupar por categoría y resumir
df.groupby('categoria')['ventas'].mean()
```

Visualización con Pandas

```
# Gráfico de barras
df['categoria'].value_counts().plot(kind='bar')

# Histograma
df['ventas'].plot(kind='hist', bins=20)

# Boxplot por grupo
df.boxplot(column='ventas', by='categoria')
```

Gráficos de dispersión y correlaciones

```
# Scatter plot
import matplotlib.pyplot as plt
plt.scatter(df['edad'], df['ingresos'])
plt.xlabel('Edad')
plt.ylabel('Ingresos')
plt.show()

# Correlación entre variables
df.corr()
df.corr().style.background_gradient(cmap='coolwarm')
```

Pair plots y distribuciones

```
import seaborn as sns

# Pair plot de varias columnas
sns.pairplot(df[['edad', 'ingresos', 'ventas']])

# Distribución de una sola variable
sns.histplot(df['ventas'], kde=True)

# Distribución normal (simulada)
import numpy as np
sns.histplot(np.random.normal(loc=0, scale=1, size=1000), kde=True)
```