PRÁCTICAS: Datos con Python

Documento elaborado por Marta Olmedilla Almarza

https://creativecommons.org/licenses/by-nc/4.0/deed.es

Índice:

PRÁCTICA: Datos con Python

Introducción

En esta práctica vamos a obtener un dataset de una BD pública en la web:

- Descargaremos los datos en formato CSV
- Crearemos una BD sqlite3 con una tabla cuya estructura pueda almacenar estos datos
- Realizaremos un programa Python que:
  - Lea los datos del CSV
  - Guarde los datos en la BD sqlite3
- Realizaremos otro programa Python que
  - Cargue los datos del CSV
  - Analice los datos
  - Muestre un gráfico con el resultado del análisis

1.- Obtención del dataset en formato csv

Vamos a acceder a la página de Kaggle para obtener el dataset: “Cars Datasets (2025)“
https://www.kaggle.com/datasets

Búscalo por su nombre, si no lo encuentras aquí tienes un enlace directo:
https://www.kaggle.com/datasets/abdulmalik1518/cars-datasets-2025

En la página del dataset, observa que ofrece 11 columnas de información. Puedes descargar el CSV con los campos que te interesen, desplegando la lista seleccionable de los campos disponibles.

Existen múltiples páginas con datasets públicos disponibles para descarga, como por ejemplo “Google Dataset Search” o github. Si prefieres puedes elegir otro dataset en Kaggle o bien otro dataset de otra página, siempre que esté en formato CSV.

2.- Creación de la BD y la tabla

Ahora vamos a crear una BD sqlite3 y una tabla con los campos del dataset
Abre un terminal y crea la BD:

```
$ sqlite3 coches.db
```

A continuación haz un sql llamado “coches.sql” para crear la tabla coches, que contendrá un campo por cada uno de los que vienen en el CSV:

- Añade una clave primaria autoincrementada
- Utiliza nombres cortos (y significativos) para las columnas
- Guarda como datos numéricos los correspondientes a la capacidad de la batería (quita el cc), los caballos (quita el hp), la velocidad total (quita el km/h), el rendimiento (quita los sec) y el precio (quita el $)

Ejecuta el sql desde sqlite3 para crear la tabla

```
sqlite> .read coches.sql
```

3.- Programa Python: Leer CSV y guardar en BD

Este programa debe:

- Crear una tabla en una BD sqlite3, con la estructura del CSV anterior
- Leer el CSV, y por cada línea, insertar un registro en la BD

A continuación veremos cómo se pueden realizar en Python las acciones anteriores.

3.1.- Lectura del fichero CSV y obtención de los datos

Nota: Para realizar esta parte necesitas conocer el manejo de strings, listas y ficheros en Python.

Una parte de tu código abre el fichero CSV y lee los datos. Cada campo irá en una variable. Los campos con información numérica deberán adaptarse para guardarse como números, eliminando cc, km/h, comas que no sean parte del CSV, etc.

Nota importante: Fíjate que algunos campos tienen comas, lo que se indica metiéndolos dentro de comillas. Tienes que parsear la línea para poder recuperar los campos, aquí un split por la coma no sirve ya que en ocasiones, un único campo también contiene comas.

Cómo parsear: Para parsear la línea, ve leyéndola de izquierda a derecha. Dentro del bucle de lectura, controla cuando empieza/termina un campo.

En el siguiente apartado veremos cómo guardarlos en la base de datos

3.2.- Guardado de los datos en la tabla sqlite3

En Python, existe una librería que maneja la bd sqlite3:

```
import sqlite3
```

Para utilizarla utilizamos lo siguiente:

Conexión

Para establecer una conexión y realizar una sentencia sql utiliza el siguiente código:

Ejemplo de conexión a sqlite3 desde Python

```
import sqlite3

conexion = sqlite3.connect("miBD.db")
cur = conexion.cursor()
sql = "Cualquier sentencia SQL correcta: Create, insert, drop, etc"
```

```
cur.execute(sql)
conexion.close()
```

- La librería **“sqlite3**” manipula una BD sqlite3 de forma integral. No es necesario tener ningún driver instalado, la librería maneja perfectamente la BD, creando los ficheros necesarios en el directorio indicado en la conexión
- **connect** realiza una conexión a la BD contenida en el fichero indicado. Si no existe, la crea.
- **cursor** ofrece una referencia a la base de datos. Este objeto dispone del método execute, que permite ejecutar sentencias sql.
- **execute** admite como parámetro cualquier tipo de sentencia sql. Si queremos una sentencia preparada, podemos utilizar placeholders con “?” y substituir los valores directamente añadiéndolos a una tupla, que será un segundo parámetro
- **close** cierra la conexión.

Sentencias preparadas

Para prevenir inyección de sql, tenemos que preparar nuestras sentencias. Con Python es muy sencillo, basta asignar los valores en una tupla como segundo parámetro del execute.
Ejemplo de inserción de datos preparados

```
sql = “insert into usuario (usuario, password) values (?,?)
cur.execute(sql, (“marta@edu.org”,”1234”) )
```

Gestión de excepciones y transacciones

Siempre debemos gestionar las excepciones y asegurarnos de cerrar la conexión a la BD. Además, debemos ejecutar commit para guardar todos los cambios, o se perderán al cerrar la conexión: La librería sqlite3 inicia una transacción en el momento en que ejecutamos la primera sentencia sql.

Ejemplo de gestión de errores y transacciones

```
# Este programa muestra la conexión a una BD sqlite3 que no existe inicialmente
# Luego crea una tabla e inserta registros en ella
import sqlite3

try:
  # Me conecto a la BD. Si no existe la crea
  conexion = sqlite3.connect("usuarios.db")
  cur = conexion.cursor()
  # Borro la tabla usuarios si existe
  sql = "drop table if exists usuarios"
  cur.execute(sql)
  # Creo la tabla usuarios
  sql = "create table usuarios (usuario text, password text)"
  cur.execute(sql)
```

```
  # Inserto un usuario en la tabla
  sql = "INSERT INTO usuarios VALUES (?,?)"
  cur.execute(sql, ("marta@edu.org","marta1234"))
  # Si todo fue bien guardo todos los execute en la BD
  conexion.commit()
except Exception as ex:
  print("Se ha producido el error: ", ex)
  # Si falló algo, dejo la BD sin ninguno de los cambios
  if conexion:
    conexion.rollback()
finally:
  # Tanto si funcionó como si falló algo, cierro la conexión
  if conexion:
    conexion.close()
```

- El ejemplo anterior se asegura de cerrar la conexión a la BD, tanto si hubo fallo como si no. Por ello el cierre está en el **finally** del try-except
- La **transacción** se inicia con el primer execute..
- Si todo fue bien, hacemos **commit**, es decir, le decimos a sqlite que todos los sql que hemos ejecutado son correctos, y que los guarde de forma física en la BD
- Si hubo algún error, hacemos un **rollback**, es decir, le decimos a sqlite que ningún sql que hemos ejecutado debe efectuarse de forma física en la BD, debe dejarlo todo como estaba antes del primer execute.
- La transacción finaliza con un commit o un rollback. Si haces un execute después de una de ambas, se iniciará una nueva transacción.

Nota: Como es posible que lo que haya fallado es la conexión, antes de ejecutar código sobre ella en los except / finally, debemos asegurarnos de que existe, mediante “if conexion:”

3.3.- Crea un nuevo CSV con los datos limpios

En el apartado anterior has preparado los datos para insertarlos en la tabla. Aprovecha el resultado y genera un nuevo CSV con los datos limpios:

- Abre un fichero csv nuevo llamado **coches_limpios.csv**, para escritura de texto
- Cada vez que leas y limpies una línea, a la vez que insertas en la BD, añade la línea al fichero.

Ejemplo de CSV generado:

```
marca,coche,motor,bateriacc,caballoshp,velocidadkmh,de0a100segundos,preciodolar,combustible,asientos
FERRARI,SF90 STRADALE,V8,3990,963,340,2.5,1100000,plug in hyrbrid,2
ROLLS ROYCE,PHANTOM,V12,6749,563,250,5.3,460000,Petrol,5
Ford,KA+,1.2L Petrol,1200,85,165,10.5,15000,Petrol,5
MERCEDES, GT 63 S,V8,3982,630,250,3.2,161000,Petrol,4
AUDI,AUDI R8 Gt,V10,5204,602,320,3.6,253290,Petrol,2
BMW,Mclaren 720s,V8,3994,710,341,2.9,499000,Petrol,2
ASTON MARTIN,VANTAGE F1,V8,3982,656,314,3.6,193440,Petrol,2
BENTLEY,Continental GT Azure,V8,3996,550,318,4.0,311000,Petrol,4
LAMBORGHINI,VENENO ROADSTER,V12,6498,750,356,2.9,4500000,Petrol,2
FERRARI,F8 TRIBUTO,V8,3900,710,340,2.9,280000,Petrol,2
```

4.- Programa Python: Análisis de datos

La siguiente es una práctica guiada: Vamos a hacer un programa en Python que trabaje con el CSV anterior para analizar los datos. Aprovecharemos el CSV con los datos numéricos que hemos generado, pues está más limpio.

Una fantástica y muy popular librería que utiliza Python para análisis y manipulación de datos es **Pandas**. Podríamos decir que Pandas es como el “Excel de Python”, pero maneja los datos con una velocidad y complejidad que Excel no podría alcanzar. Es la herramienta que se utiliza para preparar y entender los datos antes de pasarlos a un modelo de Machine Learning (aprendizaje automático).

Para utilizar esta librería, tenemos que instalarla primero.

4.1.- Gestor de paquetes pip

pip es un gestor de paquetes (librerías) para Python. Instala los que necesitamos gestionando las dependencias.

Si quiero instalar un paquete para Python, abro un terminal y escribo:

```
> py -m pip install nombre_del_paquete
```

4.2.- Instalación de la librería Pandas

Para instalar Pandas y todas sus dependencias, utilizo pip:

```
> py -m pip install pandas
```

Pandas ofrece una estructura llamada “**DataFrame**”, que es como una tabla bidimensional:

- Tiene filas y columnas.
- Las columnas tienen etiqueta (nombre de campo)
- Las filas tienen etiqueta (número de fila)
- La estructura, una vez creada, es fija: No podemos añadir o quitar filas o columnas
- Soportan tipos de datos mixtos

Con un dataframe podemos:

- Cargar y guardar datos desde un fichero (CSV, JSON, Excel, etc)

- Limpiar y manipular datos
- Analizar los datos: Obteniendo estadísticas descriptivas
- Visualizar la información mediante gráficas

4.3.- Lectura del CSV con Pandas

El código para importar la librería y cargar el CSV en un dataframe es el siguiente:

```
# Importo pandas y la utilizo con el alias pd
import pandas as pd

# Leo el CSV y lo guardo en un DataFrame de Pandas
df = pd.read_csv('coches_limpios.csv')

# Compruebo que se han cargado bien los datos, muestro las primeras filas
print(df.head())
```

Resultado de la ejecución

	marca	coche	motor	bateriacc	caballoshp	velocidadkmh	de0a100segundos	preciodolar	combustible	asientos
0	FERRARI	SF90 STRADALE	V8	3990	963	340	2.5	1100000	plug in hyrbrid	2
1	ROLLS ROYCE	PHANTOM	V12	6749	563	250	5.3	460000	Petrol	5
2	Ford	KA+	1.2L Petrol	1200	85	165	10.5	15000	Petrol	5
3	MERCEDES	GT 63 S	V8	3982	630	250	3.2	161000	Petrol	4
4	AUDI	AUDI R8 Gt	V10	5204	602	320	3.6	253290	Petrol	2

- `pd.read_csv()` Lee el CSV y pone cada campo en una columna del dataframe df
- `df.head()` Obtiene las primeras filas del dataframe df.
- `print(df)` muestra el dataframe completo: Las etiquetas de filas y columnas y los datos recuperados.

4.4.- Analizando la información

Pandas ofrece múltiples funciones estadísticas para analizar la información.

Ejemplo1: Máximos y Medias de varias columnas

Vamos a calcular los máximos y las medias de velocidades, segundos en alcanzar 100km/h, y precio en dólares:

```
# Análisis de datos en CSV
import pandas as pd

# Pandas carga el CSV en un dataframe
df = pd.read_csv('coches_limpios.csv')

# Echo un vistazo a las primeras filas
```

```
print(df.head())

# Es posible que algún valor no sea numérico en las columnas numéricas,
# El siguiente código convierte a numérico la columna. Si no puede pone NaN
# NaN significa "not a number", Pandas ignora el dato en los cálculos
df['velocidadkmh'] = pd.to_numeric(df['velocidadkmh'], errors='coerce')
df['de0a100segundos'] = pd.to_numeric(df['de0a100segundos'], errors='coerce')
df['preciodolar'] = pd.to_numeric(df['preciodolar'], errors='coerce')

# Realizo cálculos interesantes sobre los totales
print("Min Segundos:",df['de0a100segundos'].min(),"segundos")
print("Media Segundos:", df["de0a100segundos"].mean(),"segundos")
print("Max Velocidad:",df['velocidadkmh'].max(),"km/h")
print("Media Velocidad:", df["velocidadkmh"].mean(),"km/h")
print("Max Precio:",df['preciodolar'].max(),"$")
print("Media Precio:", df["preciodolar"].mean(),"$")
```

- `pd.to_numeric()`: Nos aseguramos de que todos los datos numéricos… son numéricos!!: Antes de pedirle a Pandas que calcule, tenemos que asegurarnos que todas las columnas numéricas contienen números: Es posible que el set de datos esté mal, y al limpiarlo no hayamos podido obtener todos los números. Este método de Pandas coge la columna indicada y convierte a numéricos todos los valores. Si no puede convertir alguno, pone **NaN** (not a number) y lo ignora en los cálculos.
- `max(), min(), mean()`: Son métodos que obtienen el máximo, mínimo y la media de una columna. Hemos utilizado éstas, pero hay muchas más funciones estadísticas.

Resultado de la ejecución:

```
Min Segundos: 1.9 segundos
Media Segundos: 9.145348837209303 segundos
Max Velocidad: 500.0 km/h
Media Velocidad: 216.3985209531635 km/h
Max Precio: 18000000.0 $
Media Precio: 139306.1947411668 $
```

Ejemplo2: Cálculos sobre agrupaciones

Podemos realizar cálclulos sobre agrupaciones, por ejemplo las medias y máximas por marca de coche.

```
# Antes de realizar una agrupación, la columna de agrupación
# tiene que tener valores unificados.
# Con pandas ponemos la marca en mayúsculas y quitamos espacios
df['marca'] = df['marca'].str.strip().str.upper()
```

```
# Agrupo los datos en otro dataframe
# Sus columnas serán cálculos agrupados
df_agrupado = df.groupby('marca').agg({
  'velocidadkmh': ['max','mean'],
  'de0a100segundos': ['min', 'mean'],
  'preciodolar': ['max','mean']
}).reset_index()
print("ESTADÍSTICAS POR MARCA")
print(df_agrupado)
```

- `str.strip()` elimina espacios y `str.upper()` pone en mayúsculas todos los datos de la columna indicada
- `df.groupby`: Crea otro dataframe, cuyas columnas serán:
  - La(s) columna(s) de agrupación
  - De columnas de datos, los cálculos que se indiquen
- `reset_index()`: Hace que las etiquetas de fila sean números. Si no lo ponemos utilizaría “marca” como etiqueta de fila, y “marca” no sería una columna del dataframe. Como queremos que sea columna, ponemos reset_index
- El nuevo dataframe tendrá una fila por cada marca, y una columna por cada cálculo
- En el ejemplo, hemos agrupado por marca y:
  - De la velocidadkmh pedimos el máximo y la media de cada marca
  - De los segundos en alcanzar los 100km/h el mínimo y la media de cada marca
  - Del precio, el máximo y la media de cada marca

Resultado de la ejecución:

```
ESTADÍSTICAS POR MARCA
                marca velocidadkmh            de0a100segundos            preciodolar
                               max        mean             min      mean         max          mean
0               ACURA        307.0  227.074074             2.9  5.677778    157000.0  6.325926e+04
1        ASTON MARTIN        402.0  330.000000             2.5  3.527273   3200000.0  7.529491e+05
2                AUDI        330.0  260.476190             3.2  5.161905    253290.0  8.287095e+04
3             BENTLEY        318.0  318.000000             4.0  4.000000    311000.0  3.110000e+05
4                 BMW        341.0  240.463415             2.9  6.880488    499000.0  6.731707e+04
5             BUGATTI        500.0  420.000000             2.2  2.400000  18000000.0  5.870000e+06
6            CADILLAC        320.0  227.250000             3.4  5.940000    149990.0  6.217850e+04
```

4.5.- Visualizando gráficamente los resultados

Instalando las librerías Matplotlib y Seaborn

Dos librerías muy utilizadas en Python para mostrar los datos de un dataframe de Pandas, son **Matplotlib** y **Seaborn**. Seaborn se apoya en Matpotlib, es más fácil de utilizar y ofrece gráficas más atractivas.

Para instalarlas:

> py -m pip install matplotlib

> py -m pip install seaborn

Ejemplo: Gráfico de Barras con Matplotlib

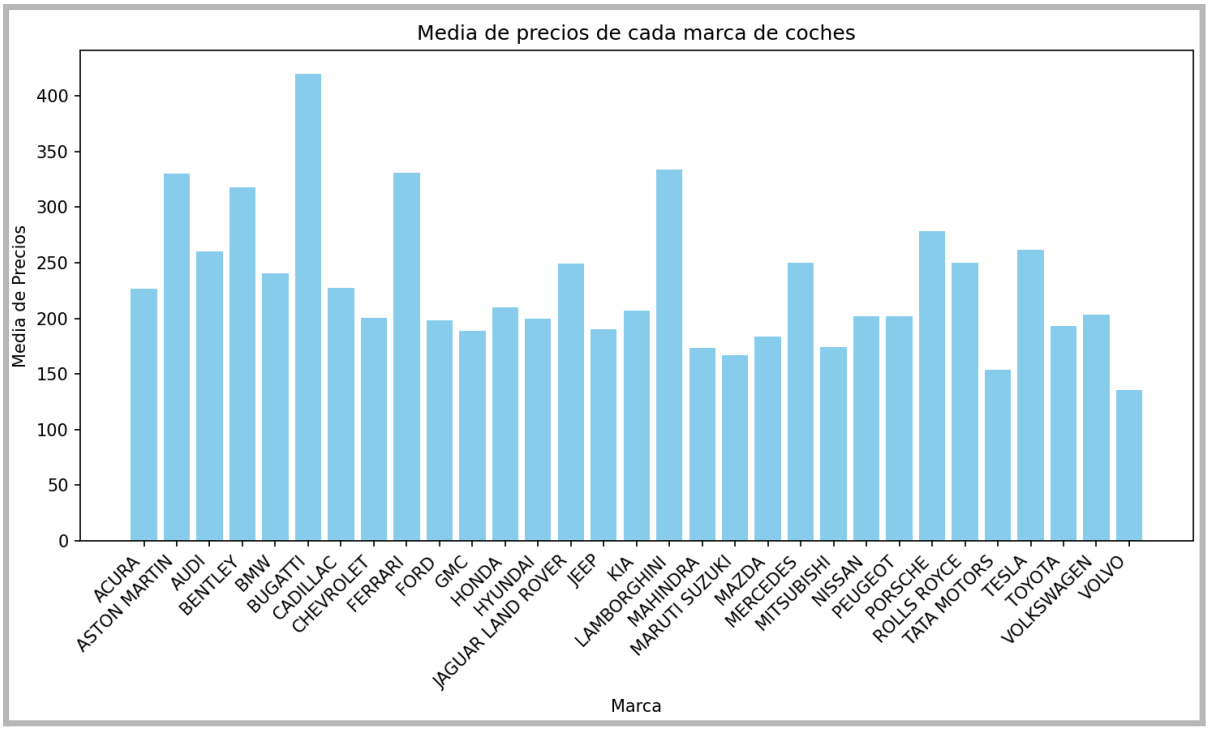
Con los dataframes obtenidos en el apartado anterior, vamos a obtener un gráfico de barras que muestre las medias de velocidades por cada marca. Utilizaremos matplotlib sobre el dataframe que tiene los cálculos agrupados: df_agrupado.

```
# ------------------------------
# VISUALIZACIÓN DE LOS RESULTADOS
# ------------------------------
import seaborn as sns
import matplotlib.pyplot as plt

# Creo el gráfico de barras con matplotlib
# Comparo la velocidad media de las marcas
plt.figure(figsize=(10,6))
plt.bar(df_agrupado['marca'], df_agrupado[('velocidadkmh','mean')],
color='skyblue')
plt.title('Media de precios de cada marca de coches')
plt.xlabel('Marca')
plt.ylabel('Media de Precios')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

- figure: Crea una figura, que se mostrará en el sistema como una ventana, del tamaño indicado
- bar: Crea un gráfico de barras bidimensional: Ponemos de parámetro los datos que se mostrarán en cada eje, en este caso la marca y la media de velocidades. Hemos utilizado una tupla para indicar cuál de las columnas “mean” se quiere utilizar. Esta función permite también indicar el color de las barras.
- title, xlabel, y label: Son textos que se mostrarán como título del gráfico, y de los ejes
- xticks: Permite rotar los datos del eje x (la marca aparecerá girada 45º a la derecha)
- tight_layout: Previene que los textos se corten en los bordes del gráfico, lo que hace es ajustarlo para que se vea todo bien.
- show: Muestra el gráfico

Ejemplo de ejecución:



Ejemplo: Gráfico de dispersión con Seaborn

Ejemplo: Gráfico de dispersión con Seaborn y Matpotlib

Seaborn trabaja conjuntamente con Matpotlib. Si definimos una figura en matpotlib, seaborn puede definir muchos gráficos y detalles para la misma, conjuntamente con matpotlib.

En este ejemplo vamos a comparar las velocidades máximas de las marcas con el mínimo de segundos que la marca necesita para llegar de 0 a 100 km/h.

```
# Creo el gráfico de dispersión con seaborn
# Estudio la relación entre rendimiento mínimo y velocidad máxima de las marcas
plt.rcParams.update({'font.size': 8})
plt.figure(figsize=(10,6))
sns.scatterplot(x=('velocidadkmh','max'), y=('de0a100segundos','min'), \
                data=df_agrupado, hue='marca', style='marca', s=200)
plt.title('Relación entre Velocidad y Rendimiento')
plt.xlabel('Velocidad máxima km/h')
plt.ylabel('Segundos Mínimos 0-100km/h')
plt.legend(title="MODELO")
plt.tight_layout()
plt.grid(True)
plt.show()
```

- rcParams: Permite establecer un tamaño de fuente (y otras características) a nivel global de matpotlib
- scatterplot: Permite definir con Seaborn un gráfico de dispersión, indicando las columnas mostradas en los ejes, y el dataframe del que va a coger los datos.

- hue y style: Permiten asignar color y estilo a los puntos del gráfico, según una tercera columna. En este caso la marca.
- s: Indica el tamaño de los puntos, en este caso es fijo
- grid: Con grid indicamos que queremos visualizar la cuadrícula

Ejemplo de ejecución