SESSION 4. SUMMARY

Victor Miguel Terron Macias

23/2/2021

SESION 4. PANDAS Y ANÁLISIS EXPLORATORIO DE DATOS

OBJETIVO

- Aprender a utilizar módulos y paquetes en Python, además aprender a instalar paquetes de terceros.
- Identificar las dos estructuras de datos con las que van a estar trabajando en el resto del módulo: Series y DataFrames
- Leer archivos JSON e incrementar su capacidad para adquirir datos.
- Explorar un dataset para entenderlo mejor.

INTRODUCCIÓN

Con todo lo que hemos aprendido hasta ahora ya estamos listos para empezar a hacer procesamiento de datos formalmente. No sé si te has dado cuenta pero de hecho ya hicimos bastante procesamiento de datos.

Todo eso de transformar y filtrar listas usando filter y map. ¡Eso fue procesamiento de datos!

Pero si quisiéramos explorar, limpiar y estructurar grandes cantidades de datos con las herramientas que hemos aprendido hasta ahora, la cosa se pondría algo difícil. Es por eso que los científicos de datos han inventado algunas herramientas hechas especialmente para la ciencia de datos. Estas herramientas nos van a facilitar la vida muchísimo.

El día de hoy aprenderemos la primera de ellas: Pandas. Veremos cómo usarla, cómo adquirir conjuntos de datos y explorarlos un poco.

IMPORTACION DE PAQUETES

Pandas es lo que se llama un paquete de Python. Un paquete es un conjunto de módulos. ¿Qué es un módulo? Es un archivo .py que contiene código de Python que podemos reutilizar en otras secciones de nuestro programa. Un paquete entonces tiene muchos módulos, cada módulo conteniendo código que cualquier persona puede utilizar para extender las capacidades de su programa.

Podríamos programar todo siempre desde cero, pero en ese caso todo tomaría muchísimo tiempo y además nunca lograríamos tanta eficiencia. Usar paquetes que hen hecho otras personas es muy útil porque nos ahorra tiempo y energía y nos da “super poderes” que podemos utilizar en nuestro programa.

Para poder utilizar un paquete, lo primero que tenemos que hacer que instalarlo, pero afortunadamente Google Colab ya tiene instalados muchos de los paquetes más importantes de ciencia de datos. Por lo tanto, basta con realizar la importación en nuestro programa para poder utilizar estos paquetes.

Abre un Jupyter Notebook, escribe el comando import pandas as pd y corre la celda.

¡Listo! Ya podemos acceder a Pandas en nuestro programa. ¿Por qué agregamos lo de as pd? Bueno básicamente le estamos diciendo a Python que queremos poder escribir pd en vez de pandas cada vez que queramos usar el paquete en nuestro programa. Nos ahorra un poco de tecleo y además es un a convención. Todos los científicos de datos usan pd en vez de pandas.

¿Y ahora qué? ¿pandas con qué se come o qué?

Bueno, primero vamos a platicar de las dos estructuras de datos que pandas nos ofrece que vamos a estar utilizando muy constantemente: Las Series y los DataFrames. Tú ya conoces dos estructuras de datos: listas y diccionarios. Las estructuras de pandas se parecen bastante a éstas pero extienden sus funcionalidades. ¡Vamos a verlas!

SERIES DE PANDAS

Las Series son parecidas a las listas en que son secuencias ordenadas de 1 dimensión que pueden contener diferentes tipos de valores. ¿1 dimensión? ¿Dónde estamos, en la Matrix? No, no, tranquilo. No es tan esotérico como parece. En geometría, una línea tiene una sola dimensión. Un punto tiene 0 dimensiones, mientras que un plano tiene 2 y un cubo tiene 3:

```
In [2]: serie = pd.Series([4, 7, 6, 2])

        serie

Out[2]: 0    4
        1    7
        2    6
        3    2
        dtype: int64
```

Figure 1: IMG

Básicamente, cada vez que agregamos una nueva dimensión, tenemos una nueva "dirección" hacia donde podríamos "avanzar" (escribo "avanzar" entre comillas porque no estamos literalmente parados sobre la línea).

En Python, de las estructuras que conocemos, las listas y ahora las Series tienen 1 una sola dimensión. Eso significa que sólo puedes avanzar hacia adelante y hacia atrás.

Así se ve una Serie de pandas:

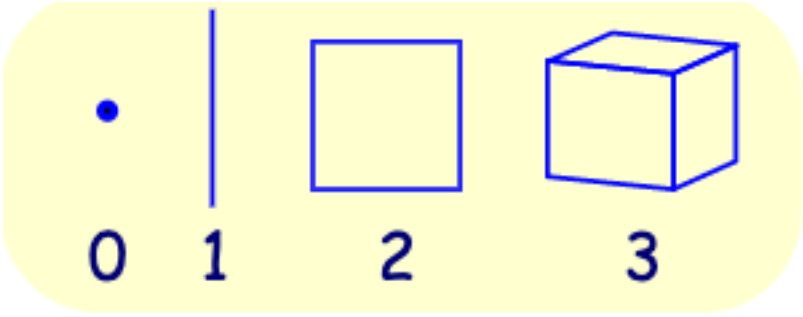

Figure 2: IMG

Al correr esta celda, podemos ver en el output cómo se ve la Serie que acabamos de crear:

¿Por qué hay dos columnas de números? ¿Y qué es eso de dtype: int64? ¿Y por qué Santa nunca me trajo el unicornio alado que le pedí?

INDICES

Veamos de cerca las columnas:

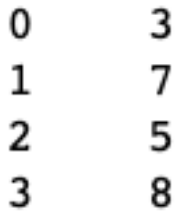

Figure 3: IMG

La columna de la derecha es la lista que le pasamos, ¿recuerdas? Es lo mismo sólo que aquí está representada verticalmente. Sigue teniendo una sola dimensión, no te preocupes.

La columna de la izquierda es el índice de nuestra Serie. Esto es algo nuevo que las listas de Python no tienen. En una Serie, cada elemento de la Serie tiene un índice asociado. Esto es similar a los diccionarios donde cada valor tiene una llave asociada.

Entonces, podríamos pensar una Serie como una especie de mezcla entre listas y diccionarios. Se parece a las listas en cuanto a que es una secuencia de elementos ordenados en una sola dimensión. Se parece a los diccionarios en que cada elemento tiene un índice asociado. Si accedemos a uno de los índices en nuestra Serie, obtendremos el elemento que está asociado a ese índice:

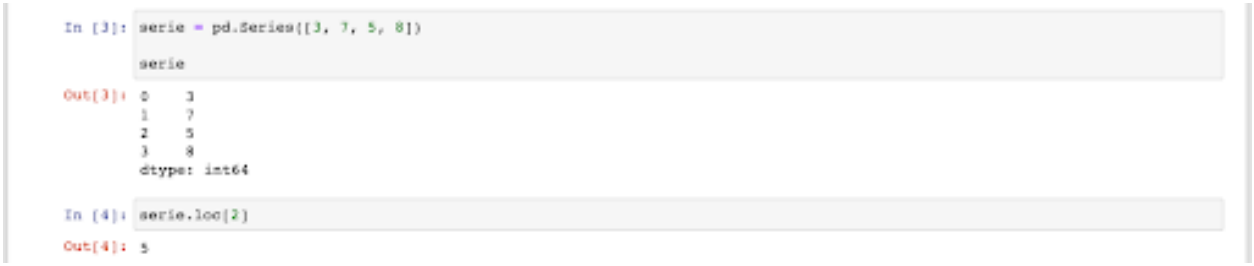

Figure 4: IMG

Bueno, ¡esto es prácticamente igual a cuando accedemos una lista por su índice! ¿Cuál es la diferencia? Primero, que tenemos que usar el operador loc para indicarle a la Serie que vamos a acceder a los elementos por el nombre del índice.

Otra gran diferencia es que el índice de una Serie puede tomar diferentes valores y diferentes órdenes. Si no le indicamos a la Serie de manera explícita qué índice tomar, la Serie va a asignar una secuencia numérica ascendente que comienza en 0 (exactamente igual a una lista). Pero podemos indicarle distintos índices a nuestra Serie de esta forma:

¡Wow! Le asignamos un índice que va desde 10 a 13 y a cada elemento le fue asignado el número que le correspondía. Ahora, para acceder a nuestro número 5 tuve que escribir serie.loc[12], ya que ese es el nuevo índice que le fue asignado.

Lo increíble es que también podemos asignarles strings como índices:

Esto se está poniendo interesante, ¿verdad?

```
In [4]: serie = pd.Series([3, 7, 5, 8], index=[10, 11, 12, 13])

        serie

Out[4]: 10    3
        11    7
        12    5
        13    8
        dtype: int64

In [5]: serie.loc[12]

Out[5]: 5
```

Figure 5: IMG

```
In [6]: serie = pd.Series([3, 7, 5, 8], index=['a', 'b', 'c', 'd'])

        serie

Out[6]: a    3
        b    7
        c    5
        d    8
        dtype: int64

In [7]: serie.loc['c']

Out[7]: 5
```

Figure 6: IMG

OTRAS ACCIONES RELACIONADAS CON ÍNDICES

Para asignar un valor nuevo a un índice en nuestra Serie basta con indexarlo y asignarle el nuevo valor (justo como sucede con las listas):

```
In [13]: serie.loc['b'] = 100

         serie

Out[13]: a      3
         b    100
         c      5
         d      8
         dtype: int64
```

Figure 7: IMG

Podemos también acceder a múltiples índices al mismo tiempo. Para pedir explícitamente los índices que queremos, le pasamos al operador de indexación [] una lista con los índices que queremos:

También podemos usar el operador dos puntos (:) para indicar lo siguiente:

En este primer ejemplo, le pasamos al operador de indexación el índice 5 y después dos puntos (:). Esto significa, “dame los índices desde el 5 hasta el final”.

También podemos pedir “dame desde el principio hasta el índice 5” de esta manera:

Podemos también pedir rangos explícitos:

TIPOS DE DATO

Regresemos por un momento a la primera Serie que creamos, pero ahora vamos a fijarnos en la información que viene en la parte inferior de nuestro output:

dtype: int64

¿Qué significa esto? Bueno, pandas tiene también sus tipos de dato, como los que tiene Python (int, float, bool, etc). Los tipos de datos más comunes en pandas son los siguientes:

```
In [21]: serie.loc[['b', 'c']]

Out[21]: b    100
         c      5
         dtype: int64
```

Figure 8: IMG

```
In [22]: serie = pd.Series([4, 7, 9, 5, 6, 0, 7, 2, 4, 5, 8])

         serie

Out[22]: 0     4
         1     7
         2     9
         3     5
         4     6
         5     0
         6     7
         7     2
         8     4
         9     5
         10    8
         dtype: int64

In [23]: serie.loc[5:]

Out[23]: 5     0
         6     7
         7     2
         8     4
         9     5
         10    8
         dtype: int64
```

Figure 9: IMG

```
In [24]: serie.loc[:5]

Out[24]: 0    4
         1    7
         2    9
         3    5
         4    6
         5    0
         dtype: int64
```

Figure 10: IMG

```
In [25]: serie.loc[3:7]

Out[25]: 3    5
         4    6
         5    0
         6    7
         7    2
         dtype: int64
```

Figure 11: IMG

- object
- int64
- float64
- bool

Hay otros, pero esos sólo te los toparas en sesiones más avanzadas.

Podemos pensar a int64 y a float64 como los equivalentes de int y float que ya conocemos. Veamos una serie con cada uno de estos tipos de datos. Primero int64:

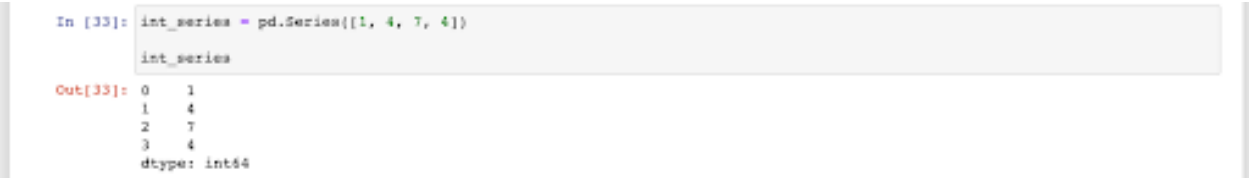

Figure 12: IMG

Ahora float64:

Figure 13: IMG

bool es exactamente lo mismo tanto en Python estándar como en pandas, como puedes ver:

Figure 14: IMG

object es el tipo de dato que es un poco inusual. Si hacemos una Serie con puras strings, pandas nos indica que el tipo de dato es object:

Pero también nos da object si tenemos una Serie con strings y otras cosas. O con una combinación entre números y otro tipo de dato:

Entonces, cuando nos topemos con el tipo de dato object, lo más apropiado es asumir que pueden ser strings o una combinación de tipos de datos.

Ahora que conocemos las características principales de las Series, vamos a pasar a la otra estructura de dato que nos ofrece pandas: los DataFrames.

DATAFRAMES

A diferencia de las Series, que son estructuras de 1 dimensión. Los DataFrames son estructuras bidimensionales. Esto quiere decir que podemos "recorrerlas" en dos direcciones. Una referencia muy sencilla para

```
In [29]: str_series = pd.Series(['a', 'b', 'c', 'd'])

         str_series

Out[29]: 0    a
         1    b
         2    c
         3    d
         dtype: object
```

Figure 15: IMG

```
In [30]: obj_series = pd.Series(['a', 1, 3.5, True])

         obj_series

Out[30]: 0       a
         1       1
         2     3.5
         3    True
         dtype: object

In [32]: mixed_series = pd.Series([1, 3, True, False])

         mixed_series

Out[32]: 0        1
         1        3
         2     True
         3    False
         dtype: object
```

Figure 16: IMG

entender cómo están estructurados los DataFrames son las tablas de MySQL o de Excel. Una tabla es una estructura bidimensional organizada en filas y columnas. Pues bueno, eso es exactamente cómo están organizados los DataFrames: como tablas. Vamos a ver cómo se ve uno:

	Nombre	Intensidad (1-10)	Es Parte de Nuestra Galaxia	Coleccionador que la encontró
0	Woopsie Doopsies	5	True	Marco P.
1	Omega-3	3	False	Marco P.
2	Justin Bieber	8	False	Jenny
3	La Twinkie	7	True	Marco P.
4	Rosaberta	2	True	Jenny

Figure 17: IMG

Es una tabla, ¿ves? Cada una de las columnas en un DataFrame es en realidad una Serie. Así que podemos pensar a un DataFrame como un diccionario de Series que comparten el mismo índice. Mira cómo fue que creamos este DataFrame:

¡Es un diccionario de listas! Las llaves se convierten en los nombres de las columnas, mientras que las listas contienen los valores que corresponden a cada fila de la tabla. Una vez creado el DataFrame, esas listas son convertidas automáticamente en Series.

INDEXACIÓN

Después de crear nuestro DataFrame podemos revisar columnas por separado usando el operador de indexación con el nombre de la columna que queremos ver:

```
In [11]: data = {
             "Nombre": ["Woopsie Doopsies", "Omega-3", "Justin Bieber", "La Twinkle", "Rosaberta"],
             "Intensidad (1-10)": [5, 3, 8, 7, 2],
             "Es Parte de Nuestra Galaxia": [True, False, False, True, True],
             "Coleccionador que la encontró": ["Marco P.", "Marco P.", "Jenny", "Marco P.", "Jenny"]
         }
```

Figure 18: IMG

```
In [43]: df["Intensidad (1-10)"]

Out[43]: 0    5
         1    3
         2    8
         3    7
         4    2
         Name: Intensidad (1-10), dtype: int64
```

Figure 19: IMG

¿Viste? Nuestra Serie tiene una nueva propiedad llamada Name. Esta propiedad es el nombre de la Serie y cuando esa Serie está en un DataFrame se convierte en el nombre de la columna.

También podemos observar más de una columna al mismo tiempo:

```
In [15]: df[["Intensidad (1-10)", "Nombre"]]
```

Out[15]:

	Intensidad (1-10)	Nombre
a	5	Woopsie Doopsies
b	3	Omega-3
c	8	Justin Bieber
d	7	La Twinkle
e	2	Rosaberta

Uso la palabra “observar” porque estas Series y DataFrames que obtenemos no son copias del DataFrame original. Sólo son “ventanas” para ver y modificar el DataFrame original con más detalle.

También podemos pedir combinaciones de filas y columnas usando el operador loc. El operador loc recibe nombres que se encuentren en nuestro índice y regresa la fila que le corresponde a ese índice:

```
In [29]: df.loc['b']

Out[29]: Nombre                          Omega-3
         Intensidad (1-10)                     3
         Es Parte de Nuestra Galaxia       False
         Coleccionador que la encontró    Marco P.
         Name: b, dtype: object
```

Figure 20: IMG

Podemos también pedir múltiples filas:

Podemos pasarle un segundo argumento especificando también las columnas que queremos:

¡También funciona con múltiples filas y columnas! Mira:

Genial

MANIPULACIÓN DE COLUMNAS

La asignación y creación de columnas nuevas a nuestro DataFrame funciona de una manera muy similar a cómo funcionan los diccionarios. Para asignar nuevos valores a una columna, simplemente accede a la columna y asígnale una Serie con los nuevos valores:

Para que todo funcione correctamente, la nueva Serie tiene que tener la misma longitud que el número de filas en el DataFrame, y además debe de compartir el mismo índice.

```
In [30]: df.loc[['a', 'b']]
Out[30]:
```

	Nombre	Intensidad (1-10)	Es Parte de Nuestra Galaxia	Coleccionador que la encontró
a	Woopsie Doopsies	5	True	Marco P.
b	Omega-3	3	False	Marco P.

Figure 21: IMG

```
In [31]: df.loc['a', 'Nombre']
Out[31]: 'Woopsie Doopsies'
```

Figure 22: IMG

Si quiero crear una nueva columna, basta con "acceder" a ella y pasarle una nueva Serie:

En este último ejemplo usé un pequeño truco para evitar tener que volver a escribir el índice del DataFrame al crear la nueva Serie. Usé df.index para obtener el índice del DataFrame y se lo pasé directamente a la Serie. DataFrame.index entonces nos regresa el índice del DataFrame desde donde lo llamemos:

Por último, vamos a ver cómo eliminar una columna. Los DataFrames de pandas tienen un método llamado drop, que podemos usar para eliminar columnas (o filas) en nuestro DataFrame. Simplemente llamamos la variable donde tenemos guardado nuestro DataFrame, llamamos el método drop y le pasamos a columns una lista con las columnas que queremos eliminar:

Como todos estos métodos sólo regresan "vistas" de nuestro DataFrame original (como si fueran lupas a través de las cuales vemos nuestro DataFrame desde distintas perspectivas), si queremos obtener un DataFrame nuevo donde las columnas que eliminamos ya no existan, tenemos que asignar el resultado a una nueva variable:

Nuestro DataFrame original está intacto:

ADQUISICIÓN DE DATOS

Al fin ha llegado nuestro momento. Hasta ahora hemos estado usando puros conjunto de datos inventados por mí (sí, aunque no lo crean, esos conjunto de datos eran inventados). Ha llegado el momento de explorar conjuntos de datos de verdad.

El primer paso en todo proyecto de Ciencia de Datos es la llamada Adquisición de Datos. La adquisición de datos es el proceso a través del cual nosotros (los científicos de datos) obtenemos datos para procesarlos, analizarlos y visualizarlos.

Hay veces que los conjuntos de datos que queremos o necesitamos no existen. En estos casos toca salir al mundo (o al Internet) a hacer una recolección de datos. Esto puede suceder a través de experimentos científicos, encuestas, medición de fenómenos, recolección de datos de uso de aplicaciones, web scraping, etc. En este módulo no vamos a aprender cómo hacer esto. Vamos a asumir que el conjuntos de datos que queremos ya existe y está esperándonos en alguna parte.

Ahora, ¿de dónde podemos conseguir estos conjuntos de datos? Los conjuntos de datos pueden estar en diversos lugares y estar almacenados en diversos formatos.

```
In [32]: df.loc[['a', 'b'], ['Nombre', 'Intensidad (1-10)']]
Out[32]:
```

	Nombre	Intensidad (1-10)
a	Woopsie Doopsies	5
b	Omega-3	3

Figure 23: IMG

```
In [35]: coleccionadores_correccion = pd.Series(['Jocolito Gutiérrez', 'Marco P.', 'Jenny', 'Jocolito Gutiérrez', 'Jenny'],
                                                index=['a', 'b', 'c', 'd', 'e'])

         df['Coleccionador que la encontró'] = coleccionadores_correccion

         df
```

Figure 24: IMG

```
In [41]: precio_en_rupias = pd.Series([25000, 500000, 4560000, 12000, 45000], index=df.index)

         df['Precio en rupias'] = precio_en_rupias

         df
```

Out[41]:

	Nombre	Intensidad (1-10)	Es Parte de Nuestra Galaxia	Coleccionador que la encontró	Precio en rupias
a	Woopsie Doopsies	5	True	Jocolito Gutiérrez	25000
b	Omega-3	3	False	Marco P.	500000
c	Justin Bieber	8	False	Jenny	4560000
d	La Twinkle	7	True	Jocolito Gutiérrez	12000
e	Rosaberta	2	True	NaN	45000

Figure 25: IMG

```
In [43]: df.index
Out[43]: Index(['a', 'b', 'c', 'd', 'e'], dtype='object')
```

Figure 26: IMG

```
In [45]: df.drop(columns=['Es Parte de Nuestra Galaxia', 'Precio en rupias'])
```

Out[45]:

	Nombre	Intensidad (1-10)	Coleccionador que la encontró
a	Woopsie Doopsies	5	Jocolito Gutiérrez
b	Omega-3	3	Marco P.
c	Justin Bieber	8	Jenny
d	La Twinkle	7	Jocolito Gutiérrez
e	Rosaberta	2	NaN

Figure 27: IMG

```
In [47]: df_dropped = df.drop(columns=['Es Parte de Nuestra Galaxia', 'Precio en rupias'])

         df_dropped
```

Out[47]:

	Nombre	Intensidad (1-10)	Coleccionador que la encontró
a	Woopsie Doopsies	5	Jocolito Gutiérrez
b	Omega-3	3	Marco P.
c	Justin Bieber	8	Jenny
d	La Twinkle	7	Jocolito Gutiérrez
e	Rosaberta	2	NaN

Figure 28: IMG

```
In [48]: df
```

Out[48]:

	Nombre	Intensidad (1-10)	Es Parte de Nuestra Galaxia	Coleccionador que la encontró	Precio en rupias
a	Woopsie Doopsies	5	True	Jocolito Gutiérrez	25000
b	Omega-3	3	False	Marco P.	500000
c	Justin Bieber	8	False	Jenny	4560000
d	La Twinkie	7	True	Jocolito Gutiérrez	12000
e	Rosaberta	2	True	NaN	45000

Figure 29: IMG

Los formatos más comunes son los siguientes:

- **JSON**
- **CSV**

Los datos pueden estar guardados en formato JSON en un archivo .json, o en formato CSV en un archivo .csv. Muchas veces obtenemos estos archivos de plataformas como Kaggle, que es uno de los repositorios de conjuntos de datos más importantes del mundo (así como una plataforma de aprendizaje y profesionalización para científicos de datos). En estas plataformas, basta con descargar los archivos y utilizarlos en nuestro programa.

Otras veces, los datos no están disponibles como archivos y tenemos que "pedirlos" de algún lado. Estos datos están almacenados remotamente y a veces podemos acceder a ellos usando algún tipo de interfaz. Las dos fuentes más comunes que usamos para "pedir" datos son:

- API's
- Bases de datos

En esta sesión vamos a aprender a leer archivos JSON. En sesiones posteriores adquiriremos nuestros datos de todas las demás fuentes.

¡Continuemos!

LECTURA DE JSONS

Vamos a empezar aprendiendo a leer un archivo JSON usando pandas. El primer paso hubiera sido ir a una plataforma y descargar este archivo. Ese paso lo he hecho por ustedes, pero si quieren saber de dónde conseguí el conjunto de datos que vamos a usar, pueden seguir este link.

El conjunto de datos que vamos a usar ha sido pre-procesado por mí para que sea adecuado para esta sesión. Entonces, ¿qué tenemos que hacer? El primer paso es importar nuestra librería pandas y una librería de Python para manejar formato JSON:

```
In [49]: import pandas as pd
         import json
```

Figure 30: IMG

Ahora, primero tenemos que leer el archivo antes de pasárselo a pandas. Esto se hace con el siguiente código:

No importa si no entiendes al 100% qué está pasando aquí, pero te voy a dar un pequeño tour de todas maneras.

Con esta parte del código le estamos diciendo a Python que queremos leer un archivo. El comando open sirve para leer archivos y recibe dos argumentos:

Figure 31: IMG

- El primero es la ruta (path) a nuestro archivo. Si todavía te confundes mucho con las rutas, la página de Wikipedia sobre rutas está bastante completa.
- El segundo es el tipo de acceso. En este caso hemos escrito 'r' que significa read. Para gran parte de lo que vamos a hacer como científicos de datos, el acceso 'r' bastará, pero si quieres conocer otros tipos de acceso y profundizar en el tema, puedes hacerlo acá.

Después de leer el archivo, lo hemos asignado a una variable f.

El archivo que hemos leído está en formato json y por lo tanto Python requiere convertirlo a un formato que él (Señor Python) pueda entender. Usando el comando json.load leemos el objeto en formato JSON y lo convertimos en un diccionario de Python. Ya que el formato JSON y los diccionarios tienen una estructura muy similar, para Python esto es pan comido.

Al finalizar, usamos f.close() para cerrar el archivo que abrimos y evitar que cosas raras puedan pasarle a nuestro programa.

Ahora sí, ¡es hora de ver que es lo que hay dentro!

Vamos a convertir nuestro diccionario en un DataFrame para poder revisar la información mejor. El objeto pd.DataFrame tiene un método llamado from_dict con el que podemos construir un DataFrame a partir de un diccionario:

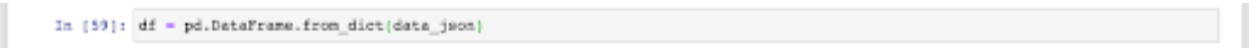

Figure 32: IMG

¡Qué emoción! Ya tenemos nuestro conjunto de datos en un DataFrame. Y, ¿ahora qué? Bueno, ahora es cuando las cosas se ponen buenas.

ANÁLISIS EXPLORATORIO DE DATOS

El siguiente paso después de la Adquisición de Datos es lo que se llama Análisis Exploratorio de Datos.

Este paso es donde exploramos nuestro conjunto de datos para entenderlo mejor. En este momento, por ejemplo, tenemos un conjunto de datos que estaba almacenado en un archivo llamado 'reviews_restaurantes.json', pero no tenemos la menor idea de qué hay dentro de ese archivo.

El Análisis Exploratorio de Datos es el proceso a través del cual nos enteramos de qué hay en ese conjunto de datos. Este paso es esencial antes de realizar cualquier tipo de Procesamiento, Análisis Estadístico y Visualización.

Vamos a aprender un poco de exploración de DataFrames usando este conjunto de datos que tenemos en este momento.

Lo primero que podemos hacer es simplemente pedirle al Jupyter Notebooks el output de nuestra variable df (le puse así por que son las siglas de DataFrame):

Ok, ya podemos ver algo de información. Este conjunto de datos tiene información acerca de algunos de los mejores restaurantes en diversas partes del mundo. La información fue extraída originalmente de una plataforma llamada Zomato, pero yo descargué el archivo JSON de Kaggle (de este link). El conjunto de

```
In [60]: df
```

Out[60]:

	has_online_delivery	price_range	currency	name	cuisines	location.address	location.city	user_rating.rating_text
0	1	3	Rs.	Hauz Khas Social	Continental, American, Asian, North Indian	9-A & 12, Hauz Khas Village, New Delhi	New Delhi	Very Good
1	0	3	Rs.	Qubitos - The Terrace Cafe	Thai, European, Mexican, North Indian, Chinese...	C-7, Vishal Enclave, Opposite Metro Pillar 417...	New Delhi	Excellent
2	1	2	Rs.	The Hudson Cafe	Cafe, Italian, Continental, Chinese	2524, 1st Floor, Hudson Lane, Delhi University...	New Delhi	Very Good
3	0	3	Rs.	Summer House Cafe	Italian, Continental	1st Floor, DDA Shopping Complex, Aurobindo Pla...	New Delhi	Very Good
4	0	3	Rs.	38 Barracks	North Indian, Italian, Asian, American	M-38, Outer Circle, Connaught Place, New Delhi	New Delhi	Very Good
...	...	...	...	...	...	...	...	...
1175	0	3	£	The Boozy Cow	Burger, Grill	17 Frederick Street, New Town, Edinburgh EH2 2EY	Edinburgh	Very Good
1176	0	3	£	La Favorita	Italian	325-331 Leith Walk, Leith, Edinburgh EH6 8SA	Edinburgh	Excellent
1177	0	3	£	Roseleaf Bar Cafe	Scottish, Cafe	23-24 Sandport Place, Leith, Edinburgh EH6 6EW	Edinburgh	Excellent
1178	0	3	£	Civerinos	Pizza, Italian	5 Hunter Square, Royal Mile, Old Town, Edinbur...	Edinburgh	Good
1179	0	3	£	The Hanging Bat	American	133 Lothian Road, Old Town, Edinburgh EH3 9AD	Edinburgh	Good

1180 rows × 8 columns

Figure 33: IMG

datos original tiene bastantes más datos pero yo los reduje a lo esencial para que nuestros procesos sean más sencillos (por el momento).

Al ver el output de nuestro DataFrame una de las primeras cosas en las que podemos fijarnos es en el tamaño del DataFrame. Esta información está localizada en la parte inferior:

1180 rows x 8 columns

Tenemos un conjunto de datos que tiene 1180 filas y 8 columnas.

Es posible acceder a esta información leyendo la propiedad shape de nuestro DataFrame:

```
In [67]: df.shape
Out[67]: (1180, 8)
```

Figure 34: IMG

También podemos observar que el índice tiene un orden secuencial, desde 0 hasta 1179:

Los 3 puntos que ves en medio (...) significan que hay más datos entre los índices 4 y 1175, pero que son demasiados como para mostrarlos.

También podemos ver los nombres de las columnas en la parte superior:

Veamos ahora qué métodos podemos utilizar para explorar un poco más nuestro DataFrame.

HEAD Y TAIL

Los métodos head y tail pueden usarse como un primer acercamiento para ver qué hay dentro de nuestro DataFrame. Al llamar head, pandas nos muestra las primeras 5 filas que hay en el DataFrame:

También podemos pasarle un número como argumento para indicarle cuántas filas queremos ver (siempre empezando por la cabeza del DataFrame, o sea la primera fila):

Dándole un primer vistazo a nuestro conjunto de datos vemos que tiene la siguiente información:

Figure 35: IMG

Figure 36: IMG

In [61]:
```
df.head()
```

Out[61]:

	has_online_delivery	price_range	currency	name	cuisines	location.address	location.city	user_rating.rating_text
0	1	3	Rs.	Hauz Khas Social	Continental, American, Asian, North Indian	9-A & 12, Hauz Khas Village, New Delhi	New Delhi	Very Good
1	0	3	Rs.	Qubitos - The Terrace Cafe	Thai, European, Mexican, North Indian, Chinese...	C-7, Vishal Enclave, Opposite Metro Pillar 417...	New Delhi	Excellent
2	1	2	Rs.	The Hudson Cafe	Cafe, Italian, Continental, Chinese	2524, 1st Floor, Hudson Lane, Delhi University...	New Delhi	Very Good
3	0	3	Rs.	Summer House Cafe	Italian, Continental	1st Floor, DDA Shopping Complex, Aurobindo Pla...	New Delhi	Very Good
4	0	3	Rs.	38 Barracks	North Indian, Italian, Asian, American	M-38, Outer Circle, Connaught Place, New Delhi	New Delhi	Very Good

Figure 37: IMG

In [62]:
```
df.head(10)
```

Out[62]:

	has_online_delivery	price_range	currency	name	cuisines	location.address	location.city	user_rating.rating_text
0	1	3	Rs.	Hauz Khas Social	Continental, American, Asian, North Indian	9-A & 12, Hauz Khas Village, New Delhi	New Delhi	Very Good
1	0	3	Rs.	Qubitos - The Terrace Cafe	Thai, European, Mexican, North Indian, Chinese...	C-7, Vishal Enclave, Opposite Metro Pillar 417...	New Delhi	Excellent
2	1	2	Rs.	The Hudson Cafe	Cafe, Italian, Continental, Chinese	2524, 1st Floor, Hudson Lane, Delhi University...	New Delhi	Very Good
3	0	3	Rs.	Summer House Cafe	Italian, Continental	1st Floor, DDA Shopping Complex, Aurobindo Pla...	New Delhi	Very Good
4	0	3	Rs.	38 Barracks	North Indian, Italian, Asian, American	M-38, Outer Circle, Connaught Place, New Delhi	New Delhi	Very Good
5	1	2	Rs.	Spezia Bistro	Cafe, Continental, Chinese, Italian	2525, 1st Floor, Hudson Lane, Delhi University...	New Delhi	Excellent
6	0	4	Rs.	Manhattan Brewery & Bar Exchange	Finger Food, American, Continental, North Indi...	1st Floor, Global Foyer Mall, Sector 43, Golf ...	Gurgaon	Excellent
7	0	4	Rs.	The Wine Company	Italian, European	Cyber Hub, DLF Cyber City, Gurgaon	Gurgaon	Poor
8	0	4	Rs.	Farzi Cafe	Modern Indian	38/39, Level 1, Block E , Inner Circle, Connau...	New Delhi	Very Good
9	1	3	Rs.	Indian Grill Room	North Indian, Mughlai	315, 3rd Floor, Suncity Business Tower, Golf C...	Gurgaon	Excellent

Figure 38: IMG

- El nombre del restaurante y el estilo de comida que vende (name y cuisines)
- Datos sobre localización (location.address y location.city)
- Datos sobre costos (price_range y currency)
- Un booleano indicando si el restaurante ofrece pedidos online (has_online_delivery)
- Un pequeño texto que es una evaluación de un usuario (user_rating.rating_text)

Usando tail podemos echarle un vistazo a las últimas filas de nuestro DataFrame:

```
In [63]: df.tail()
```

Out[63]:

	has_online_delivery	price_range	currency	name	cuisines	location.address	location.city	user_rating.rating_text
1175	0	3	£	The Boozy Cow	Burger, Grill	17 Frederick Street, New Town, Edinburgh EH2 2EY	Edinburgh	Very Good
1176	0	3	£	La Favorita	Italian	325-331 Leith Walk, Leith, Edinburgh EH6 8SA	Edinburgh	Excellent
1177	0	3	£	Roseleaf Bar Cafe	Scottish, Cafe	23-24 Sandport Place, Leith, Edinburgh EH6 6EW	Edinburgh	Excellent
1178	0	3	£	Civerinos	Pizza, Italian	5 Hunter Square, Royal Mile, Old Town, Edinbur...	Edinburgh	Good
1179	0	3	£	The Hanging Bat	American	133 Lothian Road, Old Town, Edinburgh EH3 9AD	Edinburgh	Good

Figure 39: IMG

COLUMNAS Y TIPOS DE DATOS

Si queremos saber un poco más acerca de la estructura de nuestro DataFrame podemos acceder a las propiedades columns y dtypes.

columns nos da un objeto que contiene los nombres de todas las columnas:

```
In [64]: df.columns
Out[64]: Index(['has_online_delivery', 'price_range', 'currency', 'name', 'cuisines',
                'location.address', 'location.city', 'user_rating.rating_text'],
               dtype='object')
```

Figure 40: IMG

Esto es muy útil cuando tenemos conjuntos de datos muy grandes y queremos saber qué columnas tenemos en un solo vistazo.

La propiedad dtypes nos da una Serie donde podemos ver cada columna y el tipo de dato que contiene:

```
In [68]: df.dtypes
Out[68]: has_online_delivery      int64
         price_range              int64
         currency                object
         name                    object
         cuisines                object
         location.address        object
         location.city           object
         user_rating.rating_text object
         dtype: object
```

Figure 41: IMG

Podemos observar aquí que sólo las columnas has_online_delivery y price_range tienen valores numéricos. Todas las demás columnas tienen tipo object que en este caso parece que se refiere a strings.

Éste fue un primer acercamiento muy básico al Análisis Exploratorio de Datos. En la siguiente sesión aprenderemos a adquirir datos de otras fuentes y exploraremos también herramientas de exploración mucho más poderosas.

WORK SESION 4. PANDAS Y ANÁLISIS EXPLORATORIO DE DATOS

OBJETIVOS

1. Identificar las características básicas de las series t DataFrames de Pandas
2. Leer JSON's usando Pandas
3. Utilizar herramientas básicas de exploración de datos

CONTENIDO

Las Series son una de las dos estructuras de datos que ofrece pandas que nos hacen la vida mucho más fácil como científicos de datos.

Las Series son una especie de híbrido entre listas y diccionarios.

He elegido enseñar el operador loc desde el principio. Usar el operador de indexación sin loc o iloc, aunque pareciera mejor por su similitud a las listas se presta a muchas confusiones y puede acarrear muchos errores. Creo que es mejor acostumbrarse a ser específicos al respecto de si estamos pidiendo los índices por su nombre o por su posición.

EJEMPLO 2. SERIES

1. OBJETIVOS:

```
- Entender qué son las ‘Series‘
- Aprender a crear ‘Series‘ de pandas
- Aprender los métodos básicos de indexación de las ‘Series‘
```

2. DESARROLLO:

```
import pandas as pd
```

Las `Series` son secuencias ordenadas de unidimensionales que pueden contener diferentes tipos de valores. En esto se parecen a las `listas`. De hecho podemos crear `Series` usando `listas`.

```
serie_1 = pd.Series([3, 7, 9, 8])
```

Una gran diferencia que tienen con las `listas` es que cada elemento en una `Serie` tiene un índice asociado que no necesariamente es una secuencia de enteros como en las `listas`. En este aspecto, nuestras `Series` se parecen a los `diccionarios`:

```
serie_1
```

```
## 0    3
## 1    7
## 2    9
## 3    8
## dtype: int64
```

La columna de la izquierda es nuestro índice, la columna de la derecha son los datos almacenados en la `Serie`. El texto en la parte inferior es el tipo de dato que tenemos en nuestra `Serie`.

Los tipos de datos más comunes que podemos encontrar son:

1. `int64`: Equivalente a `int`
2. `float64`: Equivalente a `float`
3. `bool`: Equivalente a `bool` (duh)
4. `object`: Equivalente a `str`, o indica que hay una mezcla de tipos de datos numéricos y no-numéricos en la `Serie`

   Importante: Tener `Series` que contengan diversos tipos de datos es una **muy mala** práctica. Lo recomendable es siempre tener homogeneidad de tipos de dato en cada `Serie` que tengamos. De todas maneras, se encontrarán por ahí algunos conjuntos de datos que contienen `Series` con tipos de datos diversos. Es por eso que cuando nos topemos con un tipo de dato `obj` tenemos que ser cuidadosos y no asumir automáticamente que el tipo de dato incluido son `strings`.

Podemos crear `Series` con un índice customizado:

```
serie_2 = pd.Series([4, 7, 9, 8], index=[10, 11, 12, 13])

serie_2
```

```
## 10    4
## 11    7
## 12    9
## 13    8
## dtype: int64
```

Incluso podemos usar `strings` en el índice:

```
serie_3 = pd.Series([5, 8, 7, 2], index=['a', 'b', 'c', 'd'])

serie_3
```

```
## a    5
## b    8
## c    7
## d    2
## dtype: int64
```

Debido a su similitud, podemos incluso crear `Series` usando `diccionarios`:

```
datos = {
    "Juan": 45,
    "Pepe": 56,
    "Alfonsina": 12,
    "Jenny": 49,
    "Marco P.": 12
}

serie_4 = pd.Series(datos)

serie_4
```

```
## Juan         45
## Pepe         56
## Alfonsina    12
## Jenny        49
## Marco P.     12
## dtype: int64
```

Al igual que en las listas, podemos acceder a nuestros datos usando el `operador de indexación`. La diferencia es que en una `Serie` tenemos que incluir el operador `loc` para indicarle a la `Serie` que estamos accesándola usando los nombres de los índices:

```
serie_1.loc[2]
```

```
## 9
```

```
serie_2.loc[12]
```

```
## 9
```

También podemos usar también `strings` como argumento:

```
serie_3.loc['c']
```

```
## 7
```

```
serie_4.loc['Marco P.']
```

```
## 12
```

RETO 1. SERIES

1. Objetivos:

`- Practicar la creación de 'Series' y la indexación básica de éstas`

```
#IMPORTACION DE PAQUETES PARA TEMA DE SERIES
import pandas as pd
```

CREACIÓN DE `SERIES`

A continuación tenemos unas variables que contienen los nombres de los ejecutivos más importantes de nuestra ya conocida EyePoker Inc. Debajo de eso hay una variable que no ha sido asignada aún:

```
#CREACION DE SERIES
ejecutivo_1 = 'Marco P.'
ejecutivo_2 = 'Jenny'
ejecutivo_3 = 'Britney Baby'
ejecutivo_4 = 'Pepe Guardabosques'
ejecutivo_5 = 'Lombardo El Destructor'

sueldos =pd.Series([10,20,30,40,50],
                   index=[ejecutivo_1,ejecutivo_2,ejecutivo_3,ejecutivo_4,ejecutivo_5])
```

Tu tarea es crear una `Serie` de `pandas` y asignarla a la variable `sueldos`. La información que hay dentro de esta variable son (oh, sorpresa) los sueldos de dichos ejecutivos.

Los valores de la `Serie` serán los sueldos, mientras que el índice de la `Serie` serán los nombres de los ejecutivos.

Crea una `Serie` y asígnala a `sueldos` de manera que el código que tenemos debajo funcione correctamente:

```
print('== Sueldos de los principales ejecutivos de EyePoker Inc. ==\n')

print(f'{("Ejecutivo"):25} | {("Sueldo")}')
print('----------------------------------------')
print(f'{ejecutivo_1:25} | ${(sueldos.loc[ejecutivo_1])} MXN')
print(f'{ejecutivo_2:25} | ${(sueldos.loc[ejecutivo_2])} MXN')
print(f'{ejecutivo_3:25} | ${(sueldos.loc[ejecutivo_3])} MXN')
print(f'{ejecutivo_4:25} | ${(sueldos.loc[ejecutivo_4])} MXN')
print(f'{ejecutivo_5:25} | ${(sueldos.loc[ejecutivo_5])} MXN')
```

```
== Sueldos de los principales ejecutivos de EyePoker Inc. ==

Ejecutivo                 | Sueldo

----------------------------------------

Marco P.                  | $10 MXN

Jenny                     | $20 MXN

Britney Baby              | $30 MXN

Pepe Guardabosques        | $40 MXN

Lombardo El Destructor    | $50 MXN
```

EJEMPLO 2. INDEXACIÓN DE SERIES

1. Objetivos:

- Aprender métodos avanzados de indexación de ‘Series‘

```
import pandas as pd
```

Nota: Algunas de las siguientes técnicas también son posibles en `listas`

Las `Series` nos permiten muchas más maneras de acceder a sus datos que vuelven a su sistema de indexación sumamente flexible. Veamos algunas de ellas:

```
serie = pd.Series(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i'])

serie
```

```
## 0    a
## 1    b
## 2    c
## 3    d
## 4    e
## 5    f
## 6    g
## 7    h
## 8    i
## dtype: object
```

Podemos pedir más de un elemento al mismo tiempo pasando una lista de índices:

```
serie.loc[[0, 1, 2]]
```

```
## 0    a
## 1    b
## 2    c
## dtype: object
```

Los elementos serán regresados en el orden en el que los pedimos:

```
serie.loc[[5, 8, 2, 4]]
```

```
## 5    f
## 8    i
## 2    c
## 4    e
## dtype: object
```

Podemos pedir "desde el principio hasta el índice x" o "desde el índice x hasta el final" de esta manera:

```
serie.loc[:4]
```

```
## 0    a
## 1    b
## 2    c
## 3    d
## 4    e
## dtype: object
```

```
serie.loc[6:]
```

```
## 6    g
## 7    h
## 8    i
## dtype: object
```

Podemos pedir incluso rangos:

```
serie.loc[3:8]
```

```
## 3    d
## 4    e
## 5    f
## 6    g
## 7    h
## 8    i
## dtype: object
```

También podemos usar rangos cuando nuestro índice no es numérico:

```
serie_2 = pd.Series([1, 2, 3, 4, 5, 6, 7, 8], index=['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])

serie_2
```

```
## a    1
## b    2
## c    3
## d    4
## e    5
## f    6
## g    7
## h    8
## dtype: int64
```

```
serie_2.loc['c':'f']
```

```
## c    3
## d    4
## e    5
## f    6
## dtype: int64
```

RETO 2. INDEXACIÓN DE SERIES

OBJETIVO

- Practicar técnicas avanzadas de indexación de `Series`

DESARROLLO

Indexación de `Series`

Tenemos una `Serie` que contiene los gastos mensuales totales (en MXN) de distintas divisiones de EyePoker Inc. Tú eres el Contador Oficial y tienes que obtener subconjuntos de datos que sirvan para agregar los gastos totales de diferentes combinaciones de divisiones.

Los datos son los siguientes:

```
import pandas as pd

gastos_mensuales = {
    'A': 15000,
    'B': 200000,
    'C': 3250000,
    'D': 120000,
    'E': 135000,
    'F': 55000,
    'G': 100000,
    'H': 25000
}

gastos_serie = pd.Series(gastos_mensuales)
```

```
gastos_serie
```

```
## A      15000
## B     200000
## C    3250000
## D     120000
## E     135000
## F      55000
## G     100000
## H      25000
## dtype: int64
```

El índice es el nombre de la división y los valores son los gastos mensuales en MXN.

Indexando la serie `gastos_serie` extrae las combinaciones de divisiones que se indican debajo para poder hacer los cálculos necesarios. El primero es un ejemplo:

```
# Los gastos de la división 'D' y 'G'
gastos_D_G =gastos_serie[[3,6]]
gastos_D_G
# Los gastos de la división 'A' y 'E'
```

```
## D    120000
## G    100000
## dtype: int64
```

```
gastos_A_E =gastos_serie[[0,4]]
gastos_A_E
# Los gastos de la división 'B', 'F' y 'H'
```

```
## A     15000
## E    135000
## dtype: int64
```

```
gastos_B_F_H =gastos_serie[[1,5,7]]
gastos_B_F_H
# Los gastos desde la primera división hasta la división 'E'
```

```
## B    200000
## F     55000
## H     25000
## dtype: int64
```

```
gastos_principio_a_E =gastos_serie.loc[:'E']
gastos_principio_a_E
# Los gastos desde la división 'D' hasta la 'G'
```

```
## A      15000
## B     200000
## C    3250000
## D     120000
## E     135000
## dtype: int64
```

```
gastos_D_a_G =gastos_serie.loc['D':'G']
gastos_D_a_G
# Los gastos desde la división 'C' hasta el la última división
```

```
## D    120000
## E    135000
## F     55000
## G    100000
## dtype: int64
```

```
gastos_C_a_final =gastos_serie.loc['C':]
gastos_C_a_final
```

```
## C    3250000
## D     120000
## E     135000
## F      55000
## G     100000
## H      25000
## dtype: int64
```

Pídele a tu experta la función de verificación `revisar_indexaciones` (encontrada en el archivo `helpers.py` de la carpeta donde se encuentra este Reto), pégala debajo y corre la celda para verificar tu resultado:

```
# Pega aquí la función de verificación
def revisar_indexaciones(gastos_serie, gastos_D_G, gastos_A_E, gastos_B_F_H,
                         gastos_principio_a_E, gastos_D_a_G, gastos_C_a_final):

    print(f'== Revisión de Indexaciones ==\n')
    print(f"{'Indexación':30} | {'Resultado':15} | {'Suma esperada ':15} | {'Suma recibida ':15}")
    print("-"*85)
    revisar_indexacion(gastos_serie.loc[['D', 'G']], gastos_D_G, 'División D y G')
    revisar_indexacion(gastos_serie.loc[['A', 'E']], gastos_A_E, 'División A y E')
    revisar_indexacion(gastos_serie.loc[['B', 'F', 'H']], gastos_B_F_H, 'División B, F y H')
    revisar_indexacion(gastos_serie.loc[:'E'], gastos_principio_a_E, 'Desde primera División a E')
    revisar_indexacion(gastos_serie.loc['D':'G'], gastos_D_a_G, 'División D y G')
    revisar_indexacion(gastos_serie.loc['C':], gastos_C_a_final, 'División C a última División')

def formatear_precio(precio):
    return f"${precio} MXN"

def revisar_indexacion(esperada, recibida, nombre):
    es_correcta = 'Correcta' if esperada.equals(recibida) else 'Incorrecta'
    suma_esperada = formatear_precio(sum(esperada))
    suma_recibida = formatear_precio(sum(recibida))
    print(f"{nombre:30} | {es_correcta:15} | {suma_esperada:15} | {suma_recibida:15}")

revisar_indexaciones(gastos_serie, gastos_D_G, gastos_A_E, gastos_B_F_H,
                         gastos_principio_a_E, gastos_D_a_G, gastos_C_a_final)
```

```
## == Revisión de Indexaciones ==
##
## Indexación                     | Resultado       | Suma esperada   | Suma recibida
## -------------------------------------------------------------------------------------
## División D y G                 | Correcta        | $220000 MXN     | $220000 MXN
## División A y E                 | Correcta        | $150000 MXN     | $150000 MXN
## División B, F y H              | Correcta        | $280000 MXN     | $280000 MXN
## Desde primera División a E     | Correcta        | $3720000 MXN    | $3720000 MXN
## División D y G                 | Correcta        | $410000 MXN     | $410000 MXN
## División C a última División   | Correcta        | $3685000 MXN    | $3685000 MXN
```

EJEMPLO 3. DATAFRAMES

Los DataFrames son la segunda estructura de datos de pandas que vamos estar usando constantemente. Un DataFrame está hecho de dos o más Series acomodadas de manera que obtenemos una estructura tabular.

Los DataFrames son bidimensionales, tienen filas y columnas. Cada columna es una Serie que tiene un nombre. Los DataFrames nos ayudan a manejar datos en estructura tabular de manera muy eficiente.

OBJETIVOS

- Aprender a crear ‘DataFrames‘ usando diccionarios de listas
- Aprender a indexar ‘DataFrames‘ para obtener subconjuntos de datos

	Nombre	Intensidad (1-10)	Es Parte de Nuestra Galaxia	Coleccionador que la encontró
0	Woopsie Doopsies	5	True	Marco P.
1	Omega-3	3	False	Marco P.
2	Justin Bieber	8	False	Jenny
3	La Twinkle	7	True	Marco P.
4	Rosaberta	2	True	Jenny

Figure 42: BANNER

DESARROLLO

```
import pandas as pd
```

Los `DataFrames` son entonces estructuras de datos bidimensionales. Tienen filas y columnas. Hay innumerables formas de crear `DataFrames` (si quieren ahondar en el tema, aquí hay una fuente muy completa). Vamos a aprender una de ellas: los diccionarios de listas.

Aquí tenemos un diccionario de listas:

```
datos = {
    'columna_1': ['valor_fila_0', 'valor_fila_1', 'valor_fila_2', 'valor_fila_3', 'valor_fila_4'],
    'columna_2': ['valor_fila_0', 'valor_fila_1', 'valor_fila_2', 'valor_fila_3', 'valor_fila_4'],
    'columna_3': ['valor_fila_0', 'valor_fila_1', 'valor_fila_2', 'valor_fila_3', 'valor_fila_4'],
    'columna_4': ['valor_fila_0', 'valor_fila_1', 'valor_fila_2', 'valor_fila_3', 'valor_fila_4']
}
```

Vamos a convertirlo en un `DataFrame`:

```
df = pd.DataFrame(datos)

df
```

```
      columna_1     columna_2     columna_3     columna_4
0  valor_fila_0  valor_fila_0  valor_fila_0  valor_fila_0
1  valor_fila_1  valor_fila_1  valor_fila_1  valor_fila_1
2  valor_fila_2  valor_fila_2  valor_fila_2  valor_fila_2
3  valor_fila_3  valor_fila_3  valor_fila_3  valor_fila_3
4  valor_fila_4  valor_fila_4  valor_fila_4  valor_fila_4
```

> **NOTA:** En la salida del RMARKDOWN no se aprecia como tabla estilizada sino como una tabla si estilo, para verlo es necesario ejecutar el codigo dentro de un jupyter notebook

También podemos pasarle explícitamente un índice para cambiar el índice default:

```
df = pd.DataFrame(datos, index=['a', 'b', 'c', 'd', 'e'])

df
```

```
      columna_1     columna_2     columna_3     columna_4
a  valor_fila_0  valor_fila_0  valor_fila_0  valor_fila_0
b  valor_fila_1  valor_fila_1  valor_fila_1  valor_fila_1
c  valor_fila_2  valor_fila_2  valor_fila_2  valor_fila_2
d  valor_fila_3  valor_fila_3  valor_fila_3  valor_fila_3
e  valor_fila_4  valor_fila_4  valor_fila_4  valor_fila_4
```

Para observar columnas individualmente, usamos el `operador de indexación` y le pasamos el nombre de la columna:

```
df['columna_1']
```

```
a    valor_fila_0
b    valor_fila_1
c    valor_fila_2
d    valor_fila_3
e    valor_fila_4
Name: columna_1, dtype: object
```

La columna que obtuvimos es una `Serie` de `pandas` con una propiedad `Name`.

También podemos ver más de una columna pasando una `lista` con los nombres de las columnas que queremos en el orden que las queremos:

```
df[['columna_3', 'columna_1']]
```

```
      columna_3     columna_1
a  valor_fila_0  valor_fila_0
b  valor_fila_1  valor_fila_1
c  valor_fila_2  valor_fila_2
d  valor_fila_3  valor_fila_3
e  valor_fila_4  valor_fila_4
```

> **IMPORTANTE**: Usamos las palabras `observar` o `ver` porque indexar columnas no regresa una copia de esas columnas, sino solamente una "vista" de esas columnas, como si estuviéramos viéndolas a través de una ventana. Eso quiere decir que los cambios que realicemos a las "vista" se verán reflejados en el `DataFrame` original.

Para indexar filas, podemos usar el operador `loc`.

Podemos pedir una sola fila:

```
df.loc['a']
```

```
columna_1    valor_fila_0
columna_2    valor_fila_0
columna_3    valor_fila_0
columna_4    valor_fila_0
Name: a, dtype: object
```

O podemos pedir varias:

```
df.loc[['c', 'a']]
```

```
      columna_1     columna_2     columna_3     columna_4
c  valor_fila_2  valor_fila_2  valor_fila_2  valor_fila_2
a  valor_fila_0  valor_fila_0  valor_fila_0  valor_fila_0
```

Podemos pedir rangos también:

```
df.loc['b':]
```

```
      columna_1     columna_2     columna_3     columna_4
b  valor_fila_1  valor_fila_1  valor_fila_1  valor_fila_1
c  valor_fila_2  valor_fila_2  valor_fila_2  valor_fila_2
d  valor_fila_3  valor_fila_3  valor_fila_3  valor_fila_3
e  valor_fila_4  valor_fila_4  valor_fila_4  valor_fila_4
```

```
df.loc['b':'d']
```

```
      columna_1     columna_2     columna_3     columna_4
b  valor_fila_1  valor_fila_1  valor_fila_1  valor_fila_1
c  valor_fila_2  valor_fila_2  valor_fila_2  valor_fila_2
d  valor_fila_3  valor_fila_3  valor_fila_3  valor_fila_3
```

Podemos pasarle un segundo argumento a `loc` para seleccionar solamente algunas columnas de las filas que pedimos. En este caso estamos pidiendo la columna 'columna_2' de la fila 'b', por lo que obtenemos un solo valor:

```
df.loc['b', 'columna_2']
```

```
'valor_fila_1'
```

También podemos pedir múltiples filas y columnas:

```
df.loc[['e', 'c'], ['columna_4', 'columna_2']]
```

```
      columna_4     columna_2
e  valor_fila_4  valor_fila_4
c  valor_fila_2  valor_fila_2
```

RETO 3 SESION 4. DATAFRAMES

OBJETIVO

- Aprender a crear DataFrames e indexar por columna y fila

DESARROLLO

a) Creación e indexación de `DataFrames`

Eres el Data Wrangler (procesador de datos) de EyePoker Inc. Tienes el siguiente diccionario con datos que se refieren a diferentes productos que vende la empresa. Este es tu conjunto de datos y el índice que le corresponde:

```
datos_productos = {
    "nombre": ["Pokemaster","Cegatron","Pikame Mucho","Lazarillo de Tormes","Stevie Wonder","Needle", "
    "precio": [10000, 5500, 3500, 750, 15500, 12250, 23000],
    "peso": [1.2, 1.5, 2.3, 5.5, 3.4, 2.4, 8.8],
    "capacidad de destrucción retinal": [3, 7, 6, 8, 9, 2, 10],
    "disponible": [True, False, True, True, False, False, True]
}

indice = [1, 2, 3, 4, 5, 6, 7]
```

El Analista de Datos de la empresa quiere realizar algunas visualizaciones con este conjunto de datos, pero no es muy buen procesador de datos, así que te pide a ti ayuda para crear los subconjuntos de datos que necesita para sus visualizaciones.

Te ha dado las descripciones de los subconjuntos que necesita, en el orden en el que los quiere.

Tu primer paso es convertir tu `diccionario` a `DataFrame` usando `datos_productos` e `indice`:

```
import pandas as pd

df_productos=pd.DataFrame(datos_productos,index=indice)
df_productos
```

```
                nombre  precio  ...  capacidad de destrucción retinal  disponible
1           Pokemaster   10000  ...                                 3        True
2             Cegatron    5500  ...                                 7       False
3         Pikame Mucho    3500  ...                                 6        True
4  Lazarillo de Tormes     750  ...                                 8        True
5        Stevie Wonder   15500  ...                                 9       False
6               Needle   12250  ...                                 2       False
7         El AyMeDuele   23000  ...                                10        True

[7 rows x 5 columns]
```

Ahora, indexa tu `DataFrame` para obtener los subconjuntos requeridos. Los productos en existencia tienen un orden específico en la base de datos. El orden correcto es el que está definido en `datos_productos`. Eso significa que el "Pokemaster" tiene el índice `1` y es el *primer* producto; y el "El AyMeDuele" tiene el ìndice `7` y es el *último* producto.

Realiza las indexaciones debajo. Recuerda ordenar tus `DataFrames` en el orden en el que los menciona el Analista:

```
# Quiero un DataFrame que contenga los productos "Pikame Mucho" y "Stevie Wonder"
pm_sw=df_productos.loc[[1,3]]
pm_sw
# Quiero un DataFrame que contenga desde el producto #4 hasta el último
```

```
##          nombre  precio  peso  capacidad de destrucción retinal  disponible
## 1    Pokemaster   10000   1.2                                 3        True
## 3  Pikame Mucho    3500   2.3                                 6        True
```

```
p4_final =df_productos.loc[4:]
p4_final
# Quiero un DataFrame que contenga los productos "El AyMeDuele", "Lazarillo de Tormes" y "Needle"
```

```
##                 nombre  precio  ...  capacidad de destrucción retinal  disponible
## 4  Lazarillo de Tormes     750  ...                                 8        True
## 5        Stevie Wonder   15500  ...                                 9       False
## 6              Needle   12250  ...                                 2       False
## 7         El AyMeDuele   23000  ...                                10        True
##
## [4 rows x 5 columns]
```

```
amd_lt_n =df_productos.loc[[7,4,6]]
amd_lt_n
# Quiero un DataFrame que contenga desde el primer producto hasta el producto #5
```

```
##                 nombre  precio  ...  capacidad de destrucción retinal  disponible
## 7         El AyMeDuele   23000  ...                                10        True
## 4  Lazarillo de Tormes     750  ...                                 8        True
## 6              Needle   12250  ...                                 2       False
##
## [3 rows x 5 columns]
```

```
primer_p5 =df_productos.loc[:5]
primer_p5
# Quiero un DataFrame que contenga los productos "Pikame Mucho" y "Lazarillo de Tormes", pero sólo con
```

```
##                 nombre  precio  ...  capacidad de destrucción retinal  disponible
## 1           Pokemaster   10000  ...                                 3        True
## 2             Cegatron    5500  ...                                 7       False
## 3         Pikame Mucho    3500  ...                                 6        True
## 4  Lazarillo de Tormes     750  ...                                 8        True
## 5        Stevie Wonder   15500  ...                                 9       False
##
## [5 rows x 5 columns]
```

```
pm_lt_pp =df_productos.loc[[1,3],['nombre','precio','peso']]
pm_lt_pp
# Quiero un DataFrame que contenga todos los productos pero con sólo las columnas 'nombre', 'precio' y
```

```
##          nombre  precio  peso
## 1    Pokemaster   10000   1.2
## 3  Pikame Mucho    3500   2.3
```

```
t_pcdr =df_productos.loc[:7, ['nombre', 'precio','capacidad de destrucción retinal']]
t_pcdr
# Quiero un DataFrame que contenga desde el producto #3 hasta el #6, pero sólo las columnas 'nombre', '
```

```
##                 nombre  precio  capacidad de destrucción retinal
## 1           Pokemaster   10000                                3
## 2             Cegatron    5500                                7
## 3         Pikame Mucho    3500                                6
## 4  Lazarillo de Tormes     750                                8
## 5        Stevie Wonder   15500                                9
## 6                Needle   12250                                2
## 7         El AyMeDuele   23000                               10
```

```
p3_p6_pd =df_productos.loc[3:6,['nombre','precio','disponible']]
p3_p6_pd
```

```
##                 nombre  precio  disponible
## 3         Pikame Mucho    3500        True
## 4  Lazarillo de Tormes     750        True
## 5        Stevie Wonder   15500       False
## 6                Needle   12250       False
```

Pídele a tu experta la función de verificación `verificar_indexaciones` (encontrada en el archivo `helpers.py` de la carpeta donde se encuentra este Reto), pégala debajo y corre la celda para verificar tu resultado:

```
# FUNCION DE VERIFICACION
def verificar_modificaciones(datos_productos, indice, df_productos, columna_nueva, df_productos_mas_colu
                             precios_descuento, df_productos_descuento, df_productos_sin_precio_ni_peso

    import pandas as pd

    df_productos_esperado = pd.DataFrame(datos_productos, index=indice)
    if not df_productos_esperado.equals(df_productos):
        print(f'df_productos ha sido creado incorrectamente ... Favor de revisar')
        return

    print(f'== Verificación de Modificaciones ==\n')
    columna_nueva_serie = pd.Series(columna_nueva, index=indice)
    df_productos_mas_columna_nueva_2 = df_productos.copy()
    df_productos_mas_columna_nueva_2['nivel de dolor'] = columna_nueva_serie
    verificar_modificacion(df_productos_mas_columna_nueva_2, df_productos_mas_columna_nueva, 'Agrega po

    precios_descuento_serie = pd.Series(precios_descuento, index=indice)
    df_productos_descuento_2 = df_productos.copy()
    df_productos_descuento_2['precio'] = precios_descuento_serie
    verificar_modificacion(df_productos_descuento_2, df_productos_descuento, 'Cambia por favor el 'Datal

    df_productos_sin_precio_ni_peso_2 = df_productos.drop(columns=['precio', 'peso'])
    verificar_modificacion(df_productos_sin_precio_ni_peso_2, df_productos_sin_precio_ni_peso, 'Elimina

def verificar_modificacion(esperada, recibida, descripcion):
    es_correcta = "Correcto" if esperada.equals(recibida) else "Incorrecto"
    respuesta = "Muchas gracias!" if es_correcta == "Correcto" else "Favor de revisar"
    print(f"\n- Descripción de pedido: {descripcion}")
    print(f"El pedido es {es_correcta} ... {respuesta}")
```

```
#VERIFICACION
verificar_indexaciones(datos_productos, indice, df_productos, pm_sw, p4_final, amd_lt_n, primer_p5, pm_
```

LECTURA DE ARCHIVOS JSON

Uno de los formatos más comunes en los que vamos a encontrar conjuntos de datos es el formato JSON. Como probablemente ya sabrás, el formato JSON se parece bastante al formato que tienen los diccionarios de Python:

```
{
    "llave_1": "valor_1",
    "llave_2": "valor_2",
    "llave_3": "valor_3",
    "llave_4": "valor_4"
}
```

Vamos a aprender a leer archivos JSON y a convertirlos en DataFrames.

> Lectura de CSVs y adquisición de datos por medio de APIs y Bases de Datos se estudian más adelante en el módulo.

MANIPULANDO OTRO DATAFRAME

```
import pandas as pd
datos={
    "Nombre":["Miguel","Victor","Terron","Macias","Claudia","Fernanda"],
    "Cereal favorito":["Honey Bunches","EXTRA","Fruti Loops","Chochitos","Choko Krispis","Zucaritas"]
}
#CREANDO NUESTRO DATRAFRAME
dataframe=pd.DataFrame(datos)
```

```
dataframe
#SI VA UN NUM DENTRO DEL ARGUMENTO SERÁ LA CANTIDAD DE DATOS A MOSTRAR
```

```
##      Nombre Cereal favorito
## 0    Miguel   Honey Bunches
## 1    Victor           EXTRA
## 2    Terron     Fruti Loops
## 3    Macias       Chochitos
## 4   Claudia   Choko Krispis
## 5  Fernanda       Zucaritas
```

```
dataframe.head(2)
```

```
##    Nombre Cereal favorito
## 0  Miguel   Honey Bunches
## 1  Victor           EXTRA
```

```
dataframe.tail(2)#CANTIDAD DE DATOS A MOSTRAR
```

```
##      Nombre Cereal favorito
## 4   Claudia   Choko Krispis
## 5  Fernanda       Zucaritas
```

```
dataframe["Hora del desayuno"]=pd.Series(["11:00","10:00","09:00","04:00","05:00","08:00"])
dataframe
```

```
##      Nombre Cereal favorito Hora del desayuno
## 0    Miguel   Honey Bunches             11:00
## 1    Victor           EXTRA             10:00
## 2    Terron     Fruti Loops             09:00
## 3    Macias       Chochitos             04:00
## 4   Claudia   Choko Krispis             05:00
## 5  Fernanda       Zucaritas             08:00
```

```
dataframe.drop(columns=["Hora del desayuno"])
```

```
##      Nombre Cereal favorito
## 0    Miguel   Honey Bunches
## 1    Victor           EXTRA
## 2    Terron     Fruti Loops
## 3    Macias       Chochitos
## 4   Claudia   Choko Krispis
## 5  Fernanda       Zucaritas
```

```
dataframe

#ELIMINACION POR NOMBRE DE COLUMNA
```

```
##      Nombre Cereal favorito Hora del desayuno
## 0    Miguel   Honey Bunches             11:00
## 1    Victor           EXTRA             10:00
## 2    Terron     Fruti Loops             09:00
## 3    Macias       Chochitos             04:00
## 4   Claudia   Choko Krispis             05:00
## 5  Fernanda       Zucaritas             08:00
```

```
dataframe1=dataframe.drop(columns=["Hora del desayuno"])
dataframe1
#ELIMINACION POR INDICES
```

```
##      Nombre Cereal favorito
## 0    Miguel   Honey Bunches
## 1    Victor           EXTRA
## 2    Terron     Fruti Loops
## 3    Macias       Chochitos
## 4   Claudia   Choko Krispis
## 5  Fernanda       Zucaritas
```

```
dataframe.drop(index=1)

#AGREGANDO COLUMNAS A UN DATAFRAME
```

```
##      Nombre Cereal favorito Hora del desayuno
## 0    Miguel   Honey Bunches             11:00
## 2    Terron     Fruti Loops             09:00
## 3    Macias       Chochitos             04:00
## 4   Claudia   Choko Krispis             05:00
## 5  Fernanda       Zucaritas             08:00
```

```
dataframe2=dataframe.copy(deep=True)
#MUY IMPORTANTE
dataframe2=dataframe.append({"Nombre":"Manuel","Cereal favorito":"Kellogs","Hora del desayuno":"07:00"}
dataframe2
```

```
##      Nombre Cereal favorito Hora del desayuno
## 0    Miguel   Honey Bunches             11:00
## 1    Victor           EXTRA             10:00
## 2    Terron     Fruti Loops             09:00
## 3    Macias       Chochitos             04:00
## 4   Claudia   Choko Krispis             05:00
## 5  Fernanda       Zucaritas             08:00
## 6    Manuel         Kellogs             07:00
```

Es sumamente importante colocar un **ignore_index=True**

```
# Cambiando algun valor estableciudo dentro de un dataframe
dataframe2.loc[6,"Cereal favorito"]="OREO"
dataframe2
```

```
##      Nombre Cereal favorito Hora del desayuno
## 0    Miguel   Honey Bunches             11:00
## 1    Victor           EXTRA             10:00
## 2    Terron     Fruti Loops             09:00
## 3    Macias       Chochitos             04:00
## 4   Claudia   Choko Krispis             05:00
## 5  Fernanda       Zucaritas             08:00
## 6    Manuel            OREO             07:00
```

RESUMEN DE ALGUNAS FUNCIONES

- head: Elementos al inicio
- tail: Elementos del final
- drop: Eliminación de datos
- append: Agregación de columna/fila
- loc[index,columnanombre]: Actualización de un dato

Es necesario colocar un deep copy si queremos que no se modifiquen los datos sobre el primer dataframe, para ello utilizamos el comando: df.copy(deep=True)

EJEMPLO 6. LECTURA DE JSON

OBJETIVOS

- Aprender a leer archivos JSON y crear `DataFrames` con ellos

DESARROLLO

```
import pandas as pd
```

Primero tenemos que importar la librería `json` que nos ayuda a lidiar con formato JSON en Python:

```
import json
```

Después leemos el archivo JSON usando `open`:

```
f = open('C:/Users/Victor Miguel Terron/Documents/PHASE2/DATA-SCIENCE-2PHASE/DATA PROCESSING AND ANALYSI
```

La ruta (o `path`) puede ser absoluto o relativo (en el Prework hay un link donde se explica esto con más claridad). El `'r'` significa que queremos leer el archivo (`read`).

Después convertimos nuestro archivo JSON en un diccionario de Python:

```
json_data = json.load(f)
```

Después cerramos nuestro archivo:

```
f.close()
```

Y finalmente pasamos el diccionario a a `pandas.DataFrame.from_dict` para crear un `DataFrame`:

```
df = pd.DataFrame.from_dict(json_data)
df
```

```
##       has_online_delivery  price_range  ... location.city user_rating.rating_text
## 0                       1            3  ...     New Delhi               Very Good
## 1                       0            3  ...     New Delhi               Excellent
## 2                       1            2  ...     New Delhi               Very Good
## 3                       0            3  ...     New Delhi               Very Good
## 4                       0            3  ...     New Delhi               Very Good
## ...                   ...          ...  ...           ...                     ...
## 1175                    0            3  ...     Edinburgh               Very Good
## 1176                    0            3  ...     Edinburgh               Excellent
## 1177                    0            3  ...     Edinburgh               Excellent
## 1178                    0            3  ...     Edinburgh                    Good
## 1179                    0            3  ...     Edinburgh                    Good
##
## [1180 rows x 8 columns]
```

¡Listo! Ahora vamos a ver qué podemos hacer con este `DataFrame`.

EJEMPLO 7. ANÁLISIS EXPLORATORIO DE DATOS

OBJETIVOS

- Aprender las herramientas básicas para darle un primer vistazo a los datos que tenemos en nuestros DataFrames

DESARROLLO

```
library(reticulate)
reticulate::py_install("requests")
```

```
import pandas as pd
import json as js
import requests

data=requests.get('https://raw.githubusercontent.com/beduExpert/Procesamiento-de-Datos-con-Python-Santa

json_data=js.loads(data.content)

dataframe=pd.DataFrame.from_dict(json_data)
dataframe
```

```
##       has_online_delivery  price_range  ... location.city user_rating.rating_text
## 0                       1            3  ...     New Delhi               Very Good
## 1                       0            3  ...     New Delhi               Excellent
## 2                       1            2  ...     New Delhi               Very Good
## 3                       0            3  ...     New Delhi               Very Good
## 4                       0            3  ...     New Delhi               Very Good
## ...                   ...          ...  ...           ...                     ...
## 1175                    0            3  ...     Edinburgh               Very Good
## 1176                    0            3  ...     Edinburgh               Excellent
## 1177                    0            3  ...     Edinburgh               Excellent
## 1178                    0            3  ...     Edinburgh                    Good
## 1179                    0            3  ...     Edinburgh                    Good
##
## [1180 rows x 8 columns]
```

```
dataframe.head(5)
```

```
##    has_online_delivery  price_range  ... location.city user_rating.rating_text
## 0                    1            3  ...     New Delhi               Very Good
## 1                    0            3  ...     New Delhi               Excellent
## 2                    1            2  ...     New Delhi               Very Good
## 3                    0            3  ...     New Delhi               Very Good
## 4                    0            3  ...     New Delhi               Very Good
##
## [5 rows x 8 columns]
```

```
dataframe.tail(5)

#FORMA MÁS EFICIENTE DE ABRIR EL ARCHIVO
```

```
##       has_online_delivery  price_range  ... location.city user_rating.rating_text
## 1175                    0            3  ...     Edinburgh                Very Good
## 1176                    0            3  ...     Edinburgh                Excellent
## 1177                    0            3  ...     Edinburgh                Excellent
## 1178                    0            3  ...     Edinburgh                     Good
## 1179                    0            3  ...     Edinburgh                     Good
##
## [5 rows x 8 columns]
```

```
import pandas as pd
import json as js
```

Ok, tenemos aquí nuestro conjunto de datos del Ejemplo pasado:

```
f = open('C:/Users/Victor Miguel Terron/Documents/PHASE2/DATA-SCIENCE-2PHASE/DATA PROCESSING AND ANALYSI
json_data = json.load(f)
f.close()

dataf = pd.DataFrame.from_dict(json_data)
dataf
```

```
##       has_online_delivery  price_range  ... location.city user_rating.rating_text
## 0                       1            3  ...     New Delhi                Very Good
## 1                       0            3  ...     New Delhi                Excellent
## 2                       1            2  ...     New Delhi                Very Good
## 3                       0            3  ...     New Delhi                Very Good
## 4                       0            3  ...     New Delhi                Very Good
## ...                   ...          ...  ...           ...                      ...
## 1175                    0            3  ...     Edinburgh                Very Good
## 1176                    0            3  ...     Edinburgh                Excellent
## 1177                    0            3  ...     Edinburgh                Excellent
## 1178                    0            3  ...     Edinburgh                     Good
## 1179                    0            3  ...     Edinburgh                     Good
##
## [1180 rows x 8 columns]
```

Podemos saber la forma de nuestro DataFrame accediendo a la propiedad shape. El primer valor es el número de filas, mientras que el segundo es el número de columnas:

```
dataf.shape
```

```
## (1180, 8)
```

Podemos ver las primeras filas del dataframe con el siguiente comando:

```
dataf.head(5)
```

```
##    has_online_delivery  price_range  ... location.city user_rating.rating_text
## 0                    1            3  ...     New Delhi               Very Good
## 1                    0            3  ...     New Delhi               Excellent
## 2                    1            2  ...     New Delhi               Very Good
## 3                    0            3  ...     New Delhi               Very Good
## 4                    0            3  ...     New Delhi               Very Good
##
## [5 rows x 8 columns]
```

Podemos ver las utlimas filas del dataframe con el siguiente comando:

```
dataf.tail(5)
```

```
##       has_online_delivery  price_range  ... location.city user_rating.rating_text
## 1175                    0            3  ...     Edinburgh               Very Good
## 1176                    0            3  ...     Edinburgh               Excellent
## 1177                    0            3  ...     Edinburgh               Excellent
## 1178                    0            3  ...     Edinburgh                    Good
## 1179                    0            3  ...     Edinburgh                    Good
##
## [5 rows x 8 columns]
```

Podemos ver los tipos de datos de cada columna del datarfmae accediendo a ellas con el siguientre comando:

```
dataf.dtypes
```

```
## has_online_delivery         int64
## price_range                 int64
## currency                   object
## name                       object
## cuisines                   object
## location.address           object
## location.city              object
## user_rating.rating_text    object
## dtype: object
```

También podemos usar la función info del DataFrame para ver más información sobre las columnas. Por ejemplo, el número de valores no nulos que hay en cada una.

```
dataf.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## Index: 1180 entries, 0 to 1179
## Data columns (total 8 columns):
##  #   Column                  Non-Null Count  Dtype
## ---  ------                  --------------  -----
##  0   has_online_delivery     1180 non-null   int64
##  1   price_range             1180 non-null   int64
##  2   currency                1180 non-null   object
```

```
##  3   name                    1180 non-null   object
##  4   cuisines                1180 non-null   object
##  5   location.address        1180 non-null   object
##  6   location.city           1180 non-null   object
##  7   user_rating.rating_text 1180 non-null   object
## dtypes: int64(2), object(6)
## memory usage: 83.0+ KB
```

En caso de que tengamos tantas columnas que no podamos verlas todas en el preview de arriba, podemos acceder a una lista de columnas usando las propiedad columns:

```
df.columns
```

```
## Index(['has_online_delivery', 'price_range', 'currency', 'name', 'cuisines',
##        'location.address', 'location.city', 'user_rating.rating_text'],
##       dtype='object')
```

RETO 5. ANÁLISIS EXPLORATORIO DE DATOS

OBJETIVOS

- Practicar leer archivos JSON utilizando pandas
- Practicar hacerse preguntas acerca de los conjuntos que tenemos

DESARROLLO

Vamos a practicar explorar conjuntos de datos y hacernos preguntas acerca de ellos.

Tenemos un conjunto de datos en formato JSON almacenado en '../../Datasets/new_york_times_bestsellers-clean.json' (en la carpeta /Datasets en el directorio raíz del módulo).

Primero que nada, lee el archivo JSON y crea un DataFrame con él:

```
import json as js
import pandas as pd
file=open('C:/Users/Victor Miguel Terron/Documents/PHASE2/DATA-SCIENCE-2PHASE/DATA PROCESSING AND ANALYS
json_info=js.load(file)
file.close()
dataframereto=pd.DataFrame.from_dict(json_info)
dataframereto
```

```
##                                   amazon_product_url  ... price.numberDouble
## 0     http://www.amazon.com/The-Host-Novel-Stephenie...  ...              25.99
## 1     http://www.amazon.com/Love-Youre-With-Emily-Gi...  ...              24.95
## 2     http://www.amazon.com/The-Front-Garano-Patrici...  ...              22.95
## 3     http://www.amazon.com/Snuff-Chuck-Palahniuk/dp...  ...              24.95
## 4     http://www.amazon.com/Sundays-at-Tiffanys-Jame...  ...              24.99
## ...                                                 ...  ...                ...
## 3028  http://www.amazon.com/Six-Years-Harlan-Coben/d...  ...              27.95
## 3029  http://www.amazon.com/The-Interestings-Novel-M...  ...              27.95
## 3030  http://www.amazon.com/Man-Without-Breath-Berni...  ...              26.95
```

```
## 3031  http://www.amazon.com/The-Storyteller-Jodi-Pic...  ...              28.99
## 3032  http://www.amazon.com/Z-A-Novel-Zelda-Fitzgera...  ...              25.99
##
## [3033 rows x 12 columns]
```

```
dataframereto.shape
```

```
## (3033, 12)
```

```
dataframereto.dtypes
```

```
## amazon_product_url              object
## author                          object
## description                     object
## publisher                       object
## title                           object
## oid                             object
## bestsellers_date.numberLong      int64
## published_date.numberLong        int64
## rank.numberInt                   int64
## rank_last_week.numberInt         int64
## weeks_on_list.numberInt          int64
## price.numberDouble             float64
## dtype: object
```

```
dataframereto.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## Index: 3033 entries, 0 to 3032
## Data columns (total 12 columns):
##  #   Column                        Non-Null Count  Dtype
## ---  ------                        --------------  -----
##  0   amazon_product_url            3033 non-null   object
##  1   author                        3033 non-null   object
##  2   description                   3033 non-null   object
##  3   publisher                     3033 non-null   object
##  4   title                         3033 non-null   object
##  5   oid                           3033 non-null   object
##  6   bestsellers_date.numberLong  3033 non-null   int64
##  7   published_date.numberLong    3033 non-null   int64
##  8   rank.numberInt                3033 non-null   int64
##  9   rank_last_week.numberInt      3033 non-null   int64
##  10  weeks_on_list.numberInt       3033 non-null   int64
##  11  price.numberDouble            3033 non-null   float64
## dtypes: float64(1), int64(5), object(6)
## memory usage: 308.0+ KB
```

```
dataframereto.columns
```

```
## Index(['amazon_product_url', 'author', 'description', 'publisher', 'title',
##        'oid', 'bestsellers_date.numberLong', 'published_date.numberLong',
##        'rank.numberInt', 'rank_last_week.numberInt', 'weeks_on_list.numberInt',
##        'price.numberDouble'],
##       dtype='object')
```

Ahora, usando todas las herramientas que hemos aprendido en esta sesión (indexación de filas y columnas, shape, dtypes, head, tail, columns, info, etc) explora tu dataset y debate con el experto y tus compañeros las siguientes preguntas:

1. ¿Qué podemos saber de este dataset con tan sólo leer el nombre del archivo y los nombres de las columnas? Que son los libros más vendidos
2. ¿Cuál es el tamaño (forma) de nuestro DataFrame? ¿Podríamos considerarlo un dataset grande o pequeño?3033 filas 12 columnas, es un dataset grande
3. ¿Crees que los nombres de las columnas son suficientemente descriptivos? ¿Podrían ser más claros o limpios? Suficientemente descriptivos, pueden ser mas limpios
4. ¿Qué tipos de datos tenemos? Enteros, decimales y objetos
5. ¿Qué significa el formato en el que tenemos las fechas? numero largo aparece
6. ¿Qué tipo de preguntas podríamos responder usando los datos numéricos de este dataset? Sacar tendencias de los libros mas comprados, con mejores reseñas
7. ¿Qué tipo de preguntas podríamos responder usando los datos no-numéricos? Historico de los datos, fecha de publicación