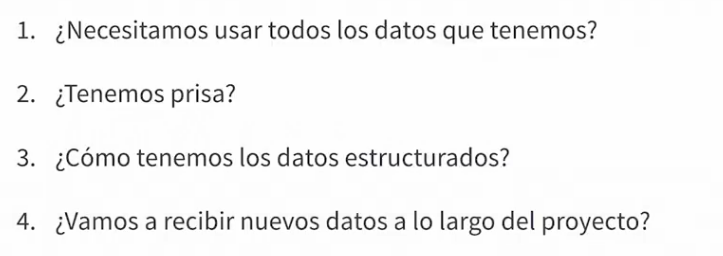
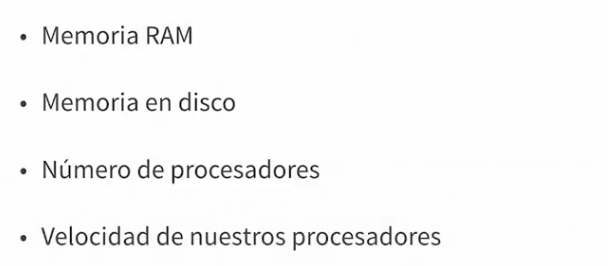
"C:/Users/MIGUEL/Desktop/programacion/python/linkeding/Python para data science y big data esencial/base\_datos\_2008.csv"

Evaluación de las necesidades de big data

Escuchamos hablar del "big data" varias veces a lo largo del día, sobre todo si nos encontramos en un contexto empresarial. Pero ¿es siempre de "big data" de lo que estamos hablando? Aunque el concepto "big data" no tiene una definición clara, lo que sí que está claro es que su definición varía con el tiempo, ya que lo que antes no cabía en un ordenador entero, ahora puede procesarse en milisegundos. Así que, cuando hablamos de qué podemos hacer con nuestro ordenador y para qué vamos a necesitar plataformas más potentes, tenemos que ser conscientes de que en unos años habrá cambiado y, a su vez, vamos a poder manejar una cantidad aún mayor de información. Voy a proponerte algunas preguntas previas. Una de las **primeras preguntas que debemos hacernos** es

si realmente **necesitamos procesar todos los datos que tenemos**. Puede sonar una trivialidad. Pero si solo queremos obtener información básica sobre el perfil de nuestros clientes y no pretendemos realizar ningún análisis más profundo, no va a hacer falta cargar en nuestro programa todo el volumen de datos del que disponemos, sino solo la fracción que nos interese y eso puede simplificar muchísimo nuestras necesidades a la hora de programar. Por ejemplo, podríamos no necesitar la opinión escrita que nos den miles de clientes sobre nuestros productos, lo cual nos podría llegar a ahorrar asignar "gigabytes" de datos. Una segunda pregunta tiene que ser **si el tiempo es un factor determinante o no**. Hay muchos procesos que podemos ejecutar perfectamente en nuestro ordenador sin necesidad de pensar en nuestras "clusters" o código más avanzado. Únicamente vamos a tardar más en obtener nuestros resultados. Si eso no es un problema, bien porque podemos desarrollar otras actividades mientras se procesan los resultados, o bien porque sea un análisis que necesitamos solo ocasionalmente y no es urgente, puede que no merezca la pena salir de nuestra zona de confort por un par de veces al año. Una tercera cuestión que debemos tener clara a la hora de pensar en cómo vamos a afrontar el proceso de análisis de datos es **el tipo de bases de datos con el que vamos a estar trabajando**/**como tenemos los datos estructurados.** Si tenemos una base de datos gigante con millones de filas y queremos procesar toda la información a la vez, por ejemplo, para entrenar un modelo de "Machine Learning", seguramente no vamos a tener otra opción que usar paquetes o librerías más específicos para su procesamiento. Pero si, por otro lado, tenemos la información fragmentada en bases de datos más reducidas, por ejemplo, según los diferentes años o productos, y tenemos que realizar análisis que no requieran cruzar los datos, es muy probable que, aunque el volumen de datos total sea muy elevado, seamos perfectamente capaces de usar los métodos más convencionales para gestionar las bases de datos por separado. En cambio, si tenemos muchas bases de datos separadas y queremos unirlas en nuestros análisis, tenemos que ser conscientes de que, dependiendo de cómo estén relacionadas entre ellas, vamos a necesitar usar mucho más código. Una cuarta y última pregunta que es conveniente hacernos antes de empezar con un proyecto de datos es **si vamos a trabajar con una base de datos estática o dinámica**. Dicho de otra forma: ¿**vamos a disponer de los mismos datos al iniciar el proyecto que al terminarlo o vamos a estar actualizando la información de la cual disponemos a lo largo de nuestros análisis?** Si nos encontramos en el caso en que no vamos a obtener nueva información, las tareas de preprocesamiento, limpieza y gestión de memoria seguramente solo necesitemos realizarlas una vez. En cambio, si vamos a estar recibiendo nuevos datos, tendremos que realizar un esfuerzo extra a la hora de pensar en cómo vamos a procesar la información y puede limitar o complicar la utilidad de ciertas estructuras o procesos que hubiésemos pensado para datos estáticos.

después de todas estas preguntas, pero en muchas ocasiones el "big data" es una necesidad real e inevitable.



Si nos encontramos en uno de estos casos, nos conviene controlar cuál es la capacidad de nuestro ordenador. Vamos a necesitar saber de cuánta **memoria RAM** disponemos, que va a determinar cuánta información podemos procesar al mismo tiempo. Y **cuánta memoria disponemos en el disco**, que a grandes rasgos va a determinar el volumen total de información que podemos llegar a gestionar, usando el código adecuado, claro. Si tenemos previsto procesar información más allá de estos límites, vamos a necesitar más que solo nuestro ordenador. Otra información realmente útil a la hora de procesar grandes volúmenes de datos es el **número de procesadores de los cuales disponemos y su velocidad,** que van a determinar en gran medida nuestra habilidad para paralelizar los cálculos y la rapidez en la cual se ejecute nuestro código, respectivamente.

Para finalizar este vídeo, un último consejo. antes de empezar a procesar todos nuestros datos de golpe, puede ser una buena idea seleccionar solo una fracción de ellos, por ejemplo, el 10 %, y ver cómo responde nuestro ordenador a algunos análisis básicos. Si vemos que es rápido, quizás con todos los datos podamos usar un código más convencional. Si, por otro lado, tarda bastante, sin duda tendremos que usar métodos más avanzados a la hora de gestionar toda la información.

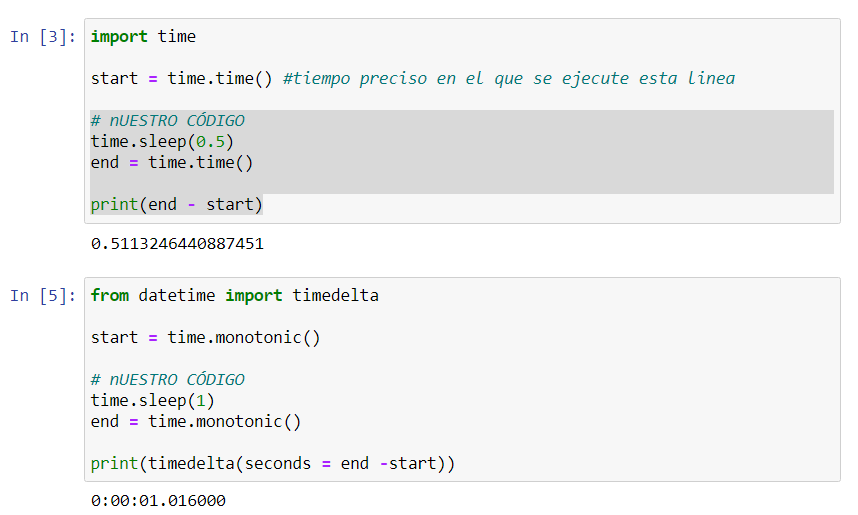
## Instalar Jupyter Notebook

Un dato curioso de Jupyter Notebook es que aquí tenemos el orden de ejecución que hemos realizado. Cada vez que ejecutamos una instrucción va a aparecer un asterisco momentáneamente y posteriormente nos va a parecer que esta celda se ha ejecutado en tercer lugar, justo después de esta. Esto es muy útil, porque nos va a permitir mantener un control del orden de ejecución de nuestros comandos. Si queremos insertar una nueva celda, vamos a pulsar este botón.

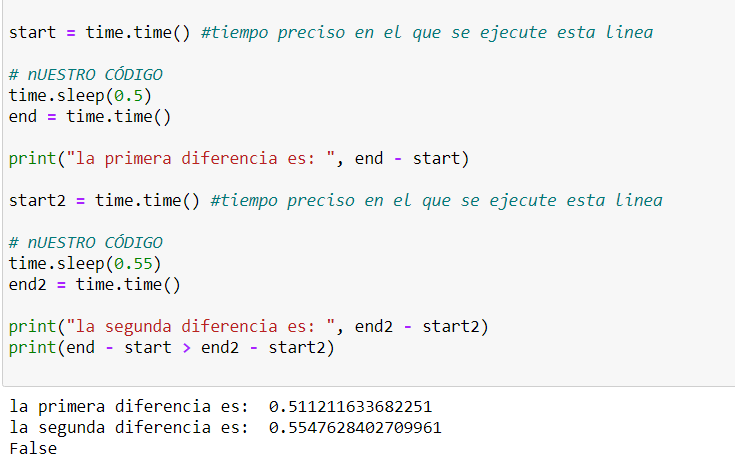
cómo se instalan nuevos paquetes en Anaconda. Salimos de Pantalla completa, vamos al Menú y vamos a poner Anaconda Prompt. Y aquí vamos a poner 'conda install' y el nombre del paquete. Si ejecutamos este comando, vamos a estar instalando en Anaconda el paquete deseado.

## Evaluar la eficiencia del código

Vamos a explorar algunas de las maneras más sencillas de evaluar la velocidad a la cual se procesa nuestro código, lo que nos va a permitir comparar diferentes maneras de realizar nuestros análisis para escoger posteriormente cuál es la más eficiente para nuestros objetivos. La manera más sencilla de realizar la comparación es tan simple como lo siguiente. Vamos a importar el **paquete 'time'**. Vamos a crear una nueva variable, que vamos a llamar 'start', que va a ser el tiempo preciso en el que se ejecute esta línea. Vamos a dejar un espacio donde vamos a poner nuestro código. Lo dejo comentado. Y vamos a crear una nueva variable que sea 'end', que va a ser otra vez el tiempo en el que se ejecute esto. Como lo que nos interesa no es ni el tiempo de inicio ni el tiempo de final, nos interesa la diferencia, vamos a tener que mostrar por pantalla con la **función 'print'** la diferencia entre los dos. Es decir, el final menos el principio. Si ejecutamos todo esto, vamos a ver que nos devuelve cero. ¿Por qué cero? Porque esto no tarda prácticamente nada en ejecutarse, así que vamos a ponerle, solo para ver el resultado, que duerma un poco. Y vamos a ponerle que duerma, por ejemplo, medio segundo. Si ejecutamos esta instrucción ahora, vemos que ha tardado medio segundo más algunos milisegundos extra, que es lo que tarda en procesar todo el código. Si, por otro lado, lo que queremos es un código un poco más complejo, pero con un "output" más humano, podemos usar el siguiente paquete. Vamos a importar 'timedelta'. Y vamos a hacer lo mismo que antes, pero usando una función distinta. Por ejemplo, vamos a crear 'start', que se va a sobrescribir al 'start' que teníamos antes y vamos a decir 'time.monotonic'. Voy a dejar un espacio para nuestro código. Vamos a hacerle dormir otra vez. Un segundo, para variar un poco. Vamos a guardarlo también. Y ahora viene el cambio. Y aquí damos a imprimir 'timedelta', que es una función, los segundos igual a el final menos al principio. Si ejecutamos todo esto, nos da un error. Este error es porque no he especificado bien el argumento de la función. Esto podemos leerlo aquí, que nos dice: "Second is an invalid keyword...". Entonces lo corregimos, ejecutamos, se espera un segundo y nos dice: "ha tardado cero horas, cero minutos y un segundo exacto". Esto es un redondeo, obviamente, pero nos permite leer mucho mejor el "output" de la función.



Por otro lado, si queremos comparar los tiempos o almacenarlos, es tan sencillo como ser el primer código que hemos mostrado o el segundo y almacenar los resultados para luego compararlos. Por ejemplo, podemos crear una nueva celda, pegar el primer código que teníamos. Y ahora, en vez de imprimir la diferencia, volvemos a seleccionar todo esto. Lo creamos aquí y vamos a cambiarle los nombres, poner 'start2'. En 2 hacerlo dormir un poquito más y estudiar la diferencia. Esto podemos hacerlo haciendo un 'print' de la primera diferencia, saber si esto es más grande, por ejemplo, que la segunda diferencia. Si ejecutamos todo esto, va a tardar un segundo y nos dice falso. ¿Falso cómo lo interpretamos? Como la primera diferencia no es más grande que la segunda diferencia. Así que nuestro primer código, el que hemos visto aquí, es un poquito más rápido que el segundo.



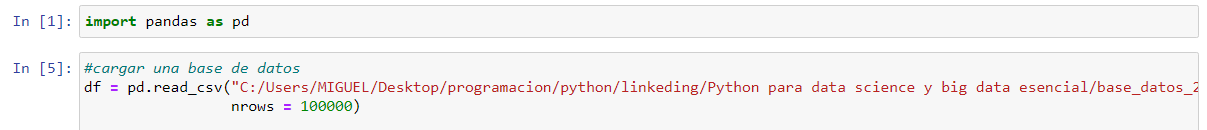
Para terminar este vídeo, un último consejo. Una buena manera de obtener una estimación más precisa del tiempo necesario para ejecutar un código concreto y así minimizar el impacto que puedan tener procesos que esté ejecutando el ordenador en segundo plano, es realizar el proceso que queremos evaluar dentro de un "loop" y almacenar los resultados en una lista, en vez de en una variable. Finalmente podremos comparar la media, la mediana, el máximo o hasta la distribución entera de tiempos de ejecución de diversos procesos y así escoger el más adecuado con un fundamento mucho más sólido y con un código que solo es ligeramente más complejo que el que hemos mostrado aquí.

**GESTIÓN DE DATOS EN PYTHON**

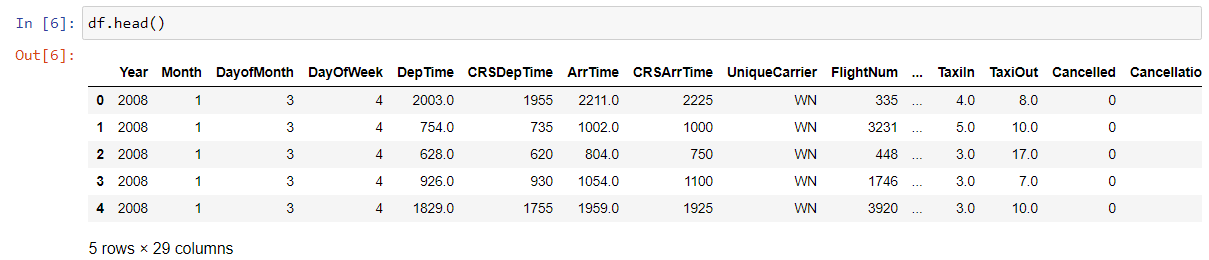
## Introducción al Pandas

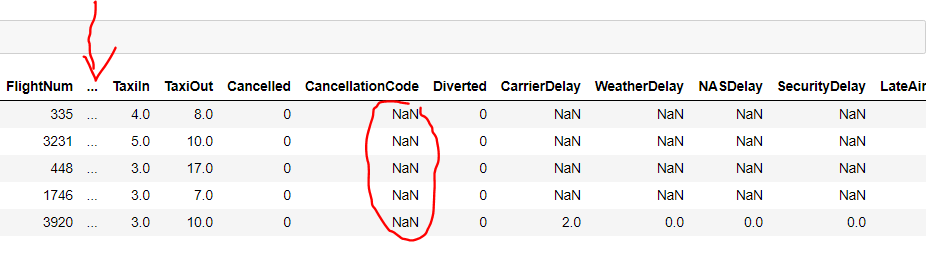
Vamos a explorar uno de los paquetes principales que nos ofrece Python para el análisis de datos. Este paquete se llama Pandas y es **lo más parecido que tenemos en el lenguaje de programación Python a la hoja de cálculo de Excel**. Nos permite una visualización muy agradable de los datos, a la vez que muchas funcionalidades interesantes. Vamos a empezar importando el paquete, usando el comando **'import**'. Y vamos a usar la anotación estándar que vamos a encontrar en Internet en todos los foros y en todas las ayudas, que es **cargar el paquete con el nombre 'pd'**. Esto nos va ayudar mucho a la hora de comunicar nuestro código o de leer el código de otros programadores.

Vamos a ejecutar esta línea y seguidamente vamos a **cargar una base de datos**, también usando la anotación estándar, que es llamar a nuestra base de datos 'df', "data frame". Vamos a usar una función que se llama **'read\_ csv'**, que depende de un parámetro, que es el "pad" donde tenemos nuestro archivo, que ya lo tengo aquí copiado, y vamos a darle un parámetro a esta función. El parámetro que vamos a darle es '**nrows**' y nos permite fijar el número de filas que queremos cargar en este archivo. El archivo original tiene más de 600 MB y esto va a acelerar algunos de los procesos que realicemos. Si ejecutamos este comando, lo que vamos a hacer es crear un **objeto llamado 'df', que contiene nuestro "data frame"**.

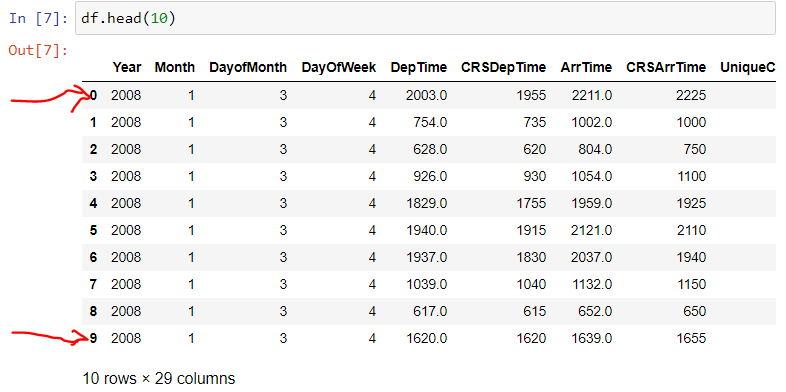


**¿Cómo podemos explorarlo? Con esta función 'head'** que nos muestra las cinco primeras filas de nuestro "data frame". Podemos explorarlo, vemos que nos dice cinco filas y 29 columnas. No nos lo muestra todo, porque, como vemos aquí con estos puntitos, nos está diciendo que hay algunas columnas que no nos está mostrando. Y podemos ver que tiene algunos datos perdidos.

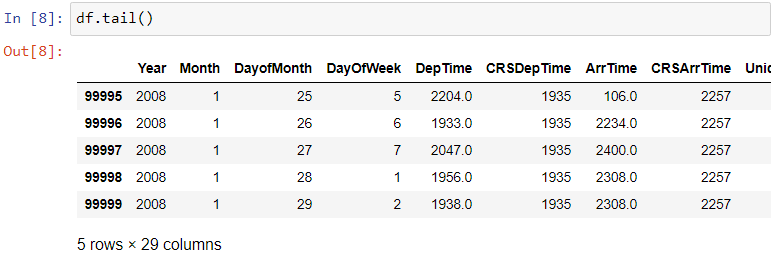




 ¿Qué pasa si queremos ver más datos? Podemos modificarlo tan fácil como añadiendo un 10 aquí, por ejemplo, y veríamos las diez primeras filas. **Un hecho importante en Python es que el índice de las filas empieza por cero siempre. Esto es muy importante**, porque cuando estemos realizando análisis, la mayoría de secuencias van a ir de cero a diez y si queremos, por ejemplo, diez casos, nos va a dar de cero a nueve, y no de uno a diez como sería lo intuitivo.

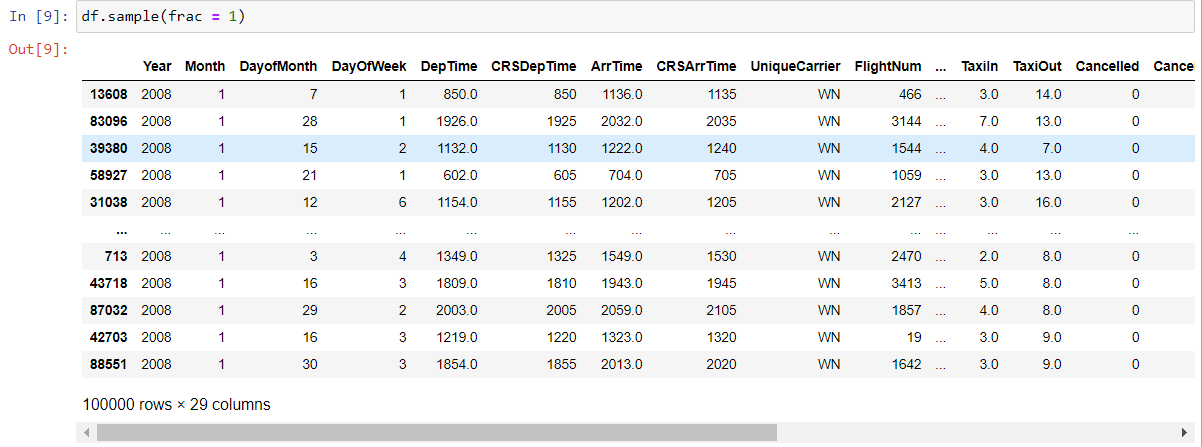


Una función muy parecida a 'head', pero que hace lo contrario –podemos ejecutarla aquí– es 'tail'. **'Tail' lo que nos da es las cinco últimas filas de nuestra base de datos**.



Estas dos funciones que acabamos de ver son muy interesantes, porque nos permiten explorar en la parte superior y en la parte inferior de la base de datos. Pero, como podemos ver, nuestros datos están ordenados. Todos los datos corresponden al 'Month' 1, porque hemos cargado pocos de la base de datos que teníamos y los números del 'DayofMonth' van ordenados.

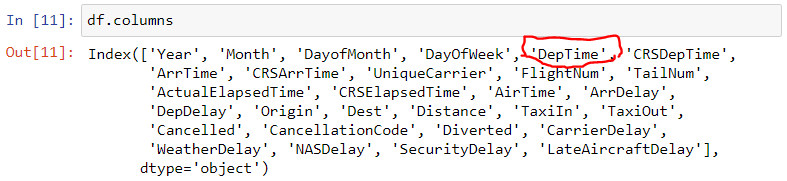
**¿Qué podemos hacer si queremos ver un poco de todo y no ver la ordenación en la que vienen los datos?** Una opción muy interesante a considerar es **reordenar nuestra base de datos**. ¿Cómo podemos hacerlo? Podemos hacer **'dt.sample'** y vamos a necesitar un parámetro, que ahora os explico lo que es, que lo que va a hacer es reordenarnos nuestra base de datos.



Lo ejecutamos y vemos que nos ha movido todos los "index". Esto ya no está ordenado y nos muestra toda la base de datos entera. Pero esto no ha guardado el resultado. Esta función de aquí lo que ha hecho es mostrarnos el "data frame" desordenado. Si lo que queremos es que lo guarde, podemos decirle 'df = dt.**sample(frac = 1)**'. Si ahora lo ejecutamos, podemos ver que no muestra nada, no tenemos un "output", pero **ahora el "data frame" guardado ya está remezclado.**



¿Qué significa 'frac = 1'? Significa que queremos que mantenga el 100 % de las filas que tenemos. Si, por otro lado, lo que nos interesa es explorar las columnas que tenemos, podemos usar esta instrucción. La voy a ejecutar ahora.

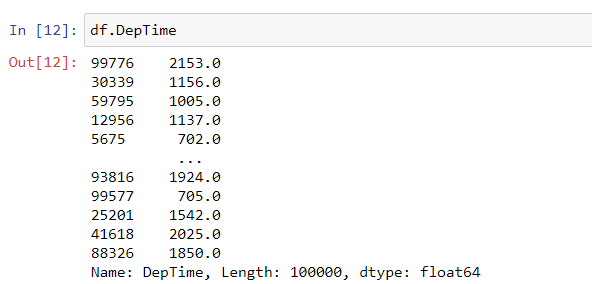


Y podemos ver en una lista todos los nombres de las columnas que tenemos.

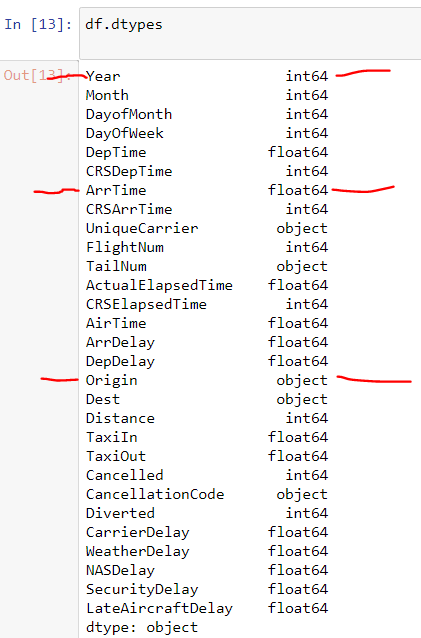
¿Por qué aquí no tenemos paréntesis?

* Porque aquí lo que estamos llamando es a uno de los atributos de nuestro "data frame" (una propiedad, programación orientada a objetos).
* Cuando aplicamos una función en nuestro "data frame", usamos paréntesis.

Cuando lo que queremos es obtener alguna información de este "data frame" que ya existe, por ejemplo, queremos ver el 'DepTime', si ahora ejecutamos esto, nos va mostrar una columna entera. ¿Qué columna? 'DepTime'. Y podemos verlo en el formato que nos muestra aquí, que es una sola columna.

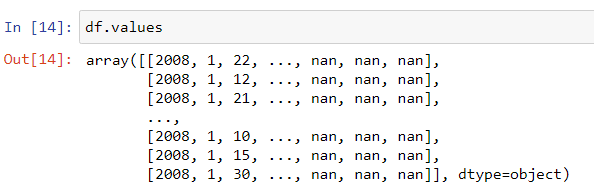


La mayoría de "data frames" con los que trabajemos van a tener distintos tipos de variables. Esto podemos explorarlo con esta instrucción **dtypes**, que lo que nos va a mostrar es qué tipo de variable es cada una.



Por ejemplo, 'Year' es un "int year", que esto es un valor entero. El tiempo de llegada es un 'float', que es un valor decimal. Y otras variables como, por ejemplo, el origen, el aeropuerto de donde salen los aviones, es un 'object', que en este caso vendría a ser una variable textual, un texto.

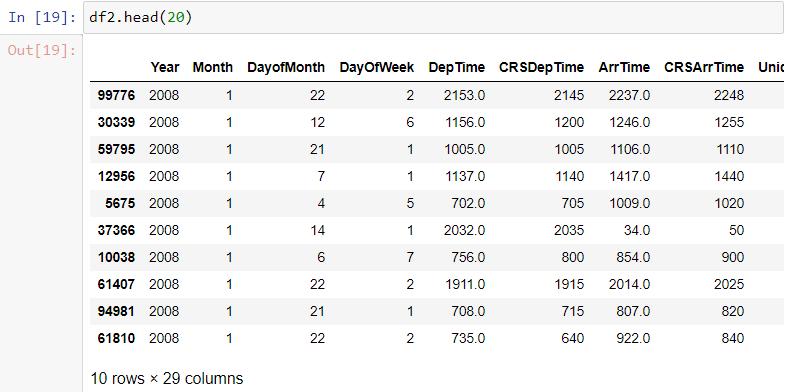
Si solo estamos interesados en los valores que tiene nuestra base de datos y no en verlos en un formato atractivo, podemos usar esta instrucción, **'values'**, que lo que nos va a devolver es todos los datos que tenemos en un formato un poco menos atractivo. **En una "array**". **Un "array" va a permitir que realicemos operaciones matriciales y otros tipos de cálculo que con un "data frame" no son tan cómodos**.



Por último, vamos a ver **cómo podemos guardar la base de datos en otra estructura**. Por ejemplo, si queremos guardar otro "data frame", crear uno nuevo, que esto va a consumir memoria, pero nos puede interesar **por si queremos hacer alguna prueba** o si queremos aplicar funciones que son muy costosas en tiempo, podemos definir un nuevo "data frame", llamémosle 'df2', que sean las primeras diez filas del "data frame" con el que estamos trabajando. Esto lo ejecutamos y no nos va a devolver nada, pero habremos guardado en memoria un nuevo objeto.



Si exploramos este nuevo objeto, por ejemplo, le pedimos que nos muestren las 20 primeras filas de este objeto y lo ejecutamos. Vamos a ver que solo nos está mostrando diez de ellas, porque hemos creado este objeto diciéndole que solo tenía diez filas del "data frame" original y no nos va a poder mostrar 20, porque no las tiene.

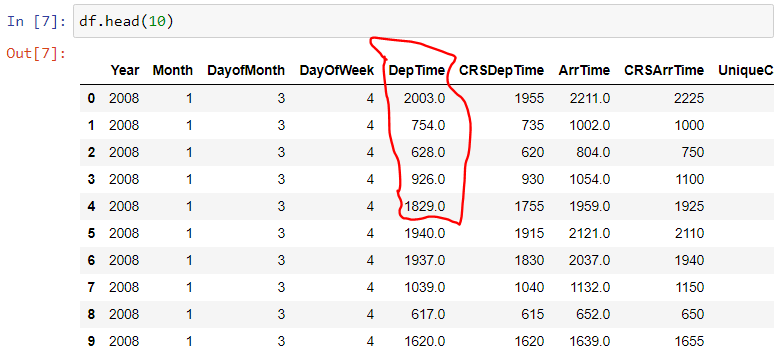


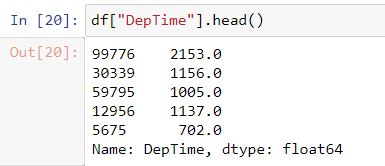
Hemos mostrado algunas de las funcionalidades más básicas que necesitarás seguro para trabajar con bases de datos. Pero esto solo es una pequeñísima parte de lo que Pandas puede ofrecernos.

## Filtrar datos en Python

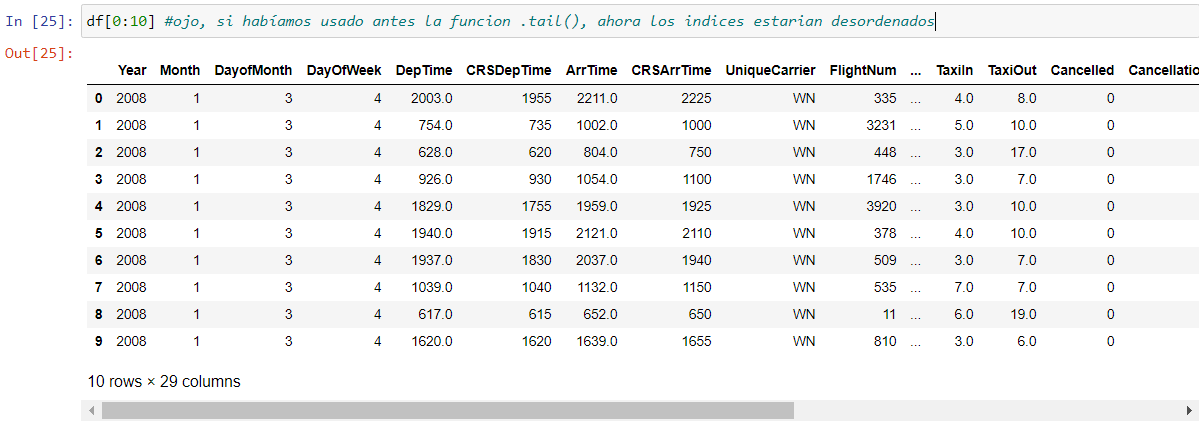
En este vídeo vamos a ver las principales herramientas de filtrado que nos ofrece Pyhton, y más concretamente el paquete Pandas. **Por filtrado entendemos la obtención de información relevante en filas y/o columnas de nuestras bases de datos**.

Por ejemplo, si lo que queremos es obtener información de una columna concreta, por ejemplo, 'ArrDelay', que es el retraso de los vuelos, podemos obtenerlo con esta sintaxis. Aquí estamos obteniendo los cinco primeros valores, que es lo que nos permite la función 'head' de la columna 'ArrDelay'.

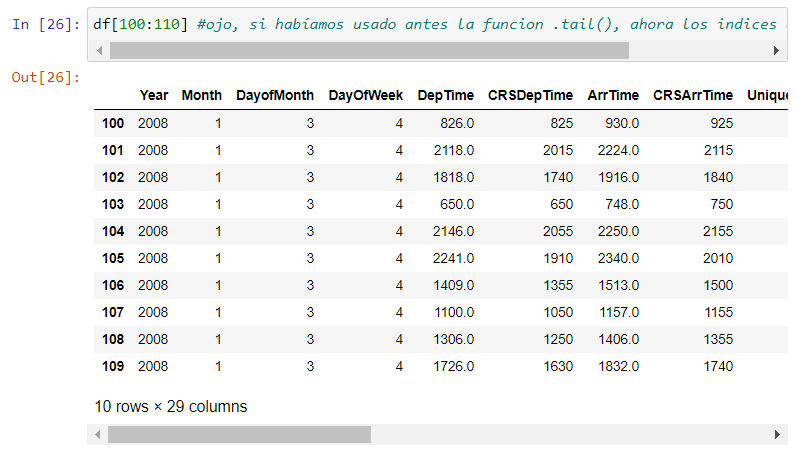




Si lo que queremos es obtener información por filas, lo que podemos hacer es usar esta sintaxis. Lo que obtenemos aquí es exactamente lo mismo que obtendríamos si usásemos 'df[0:10]'. Pero este tipo de sintaxis nos permite que, si queremos seleccionar otro intervalo, podemos usar lo que queramos.

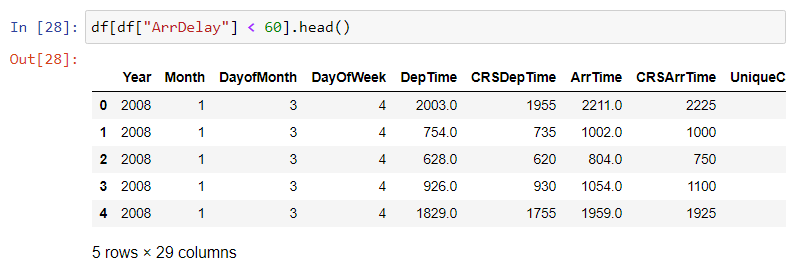


Y aquí vemos que hemos cogido de la fila 100 a la 109 porque lo hemos especificado así.



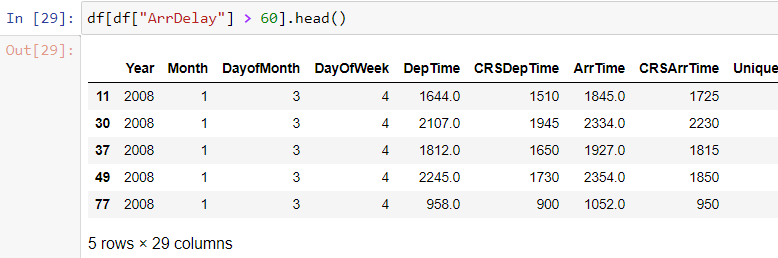
Si lo que queremos es **obtener información sobre valores concretos**, lo que podemos usar es este tipo de sintaxis.

Tenemos que **coger el "data frame" entero y, dentro de los corchetes, aplicarle un filtro concreto**. Por ejemplo, todos aquellos que se han retrasado menos de 60 minutos, y vamos a mostrar las cinco primeras filas. Si ejecutamos esto, vemos que nos ofrece exactamente las primeras cinco filas.

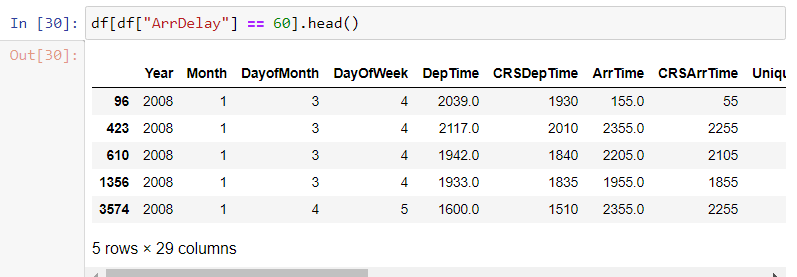


¿Por qué pasa esto? Porque los cinco primeros vuelos no se han retrasado tanto.

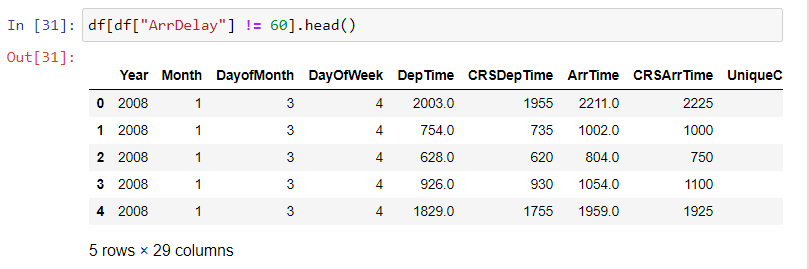
Por otro lado, lo qué podríamos hacer es ver los que se han retrasado más de una hora. Si hacemos este cambio y ejecutamos, lo que vamos a ver es que nos devuelve cinco vuelos distintos. ¿Qué vuelos? El 11, el 30, el 37, el 49 y el 77.



Las principales sintaxis para este tipo de filtro son las dos que hemos visto, el doble igual, si lo que queremos es ver aquellos vuelos que se han retrasado exactamente una hora.



Esta sintaxis, si lo que queremos es ver aquellos que no se han retrasado una hora, esto significaría de 59 para abajo y de 61 para arriba, todos los valores excepto el 60 exacto.

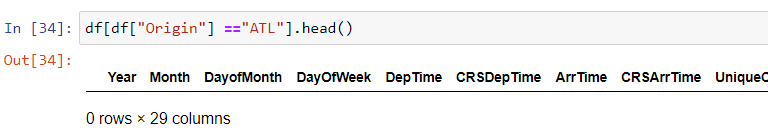


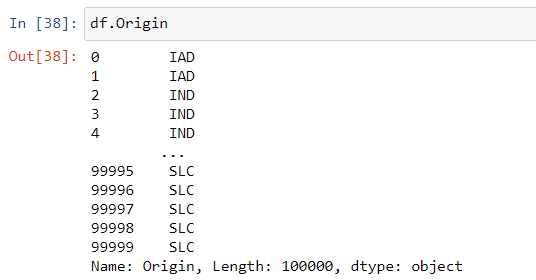
Y luego, si queremos incluir en una desigualdad el mismo valor, usaríamos esto o esto para más pequeño o igual que 60 o más grande o igual que 60.

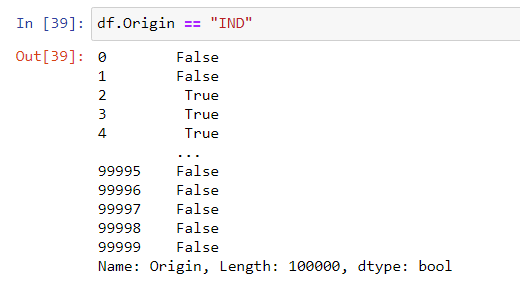


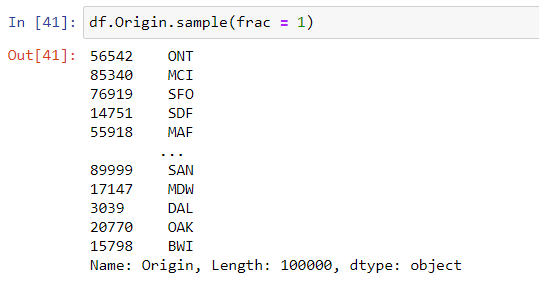


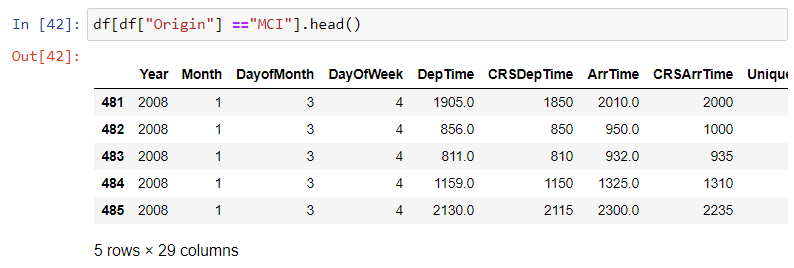
**Esto evidentemente también funciona por filtros de texto**. Por ejemplo, si lo que queremos es ver todos los vuelos que salen de un cierto aeropuerto, por ejemplo, Atlanta, vamos a usar este tipo de sintaxis. Si ejecutamos esto, vamos a ver exactamente todos los vuelos que han salido de Atlanta.





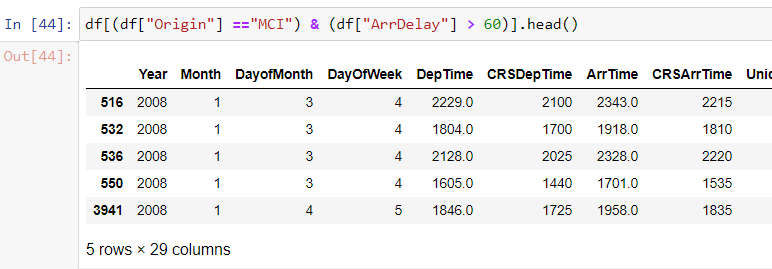






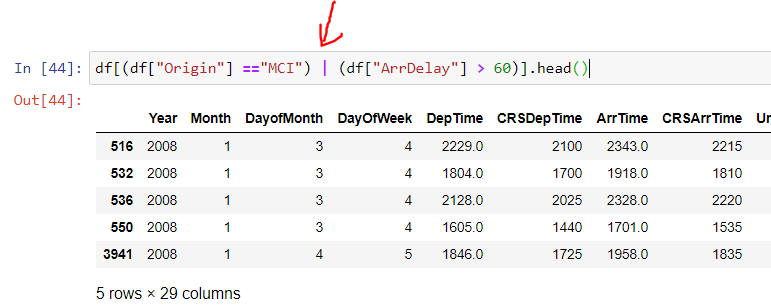
Si lo que queremos hacer **es usar dos filtros conjuntamente**, lo que vamos a tener que hacer es usar una sintaxis combinada.

Por ejemplo, vamos a filtrar nuestro "data frame", entre paréntesis el filtro que queramos y vamos a tener que añadirle el segundo filtro o tercero, cuarto... los que queramos, con este tipo de sintaxis. Por ejemplo, todos aquellos vuelos que se hayan retrasado más de 60 minutos. Y vamos a mostrar, como siempre, los cinco primeros.



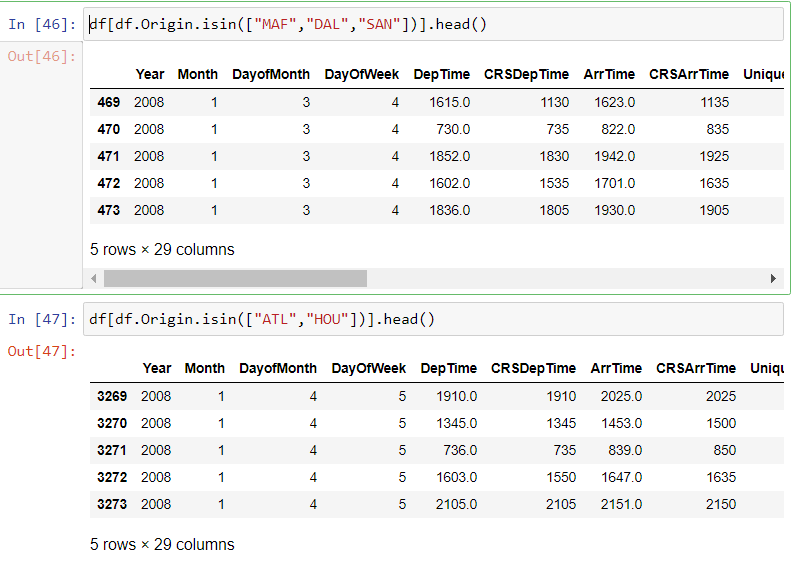
**Cuando usamos la sintaxis '&'**, el símbolo este, le estamos diciendo a Python que tienen que cumplirse las dos condiciones. La primera y la segunda.

Si, por otro lado, usamos este **símbolo |** lo que le estaremos diciendo es una o la otra.



Esto es importante porque puede interesarnos, para depende de qué tipo de análisis, un tipo de filtro u otro.

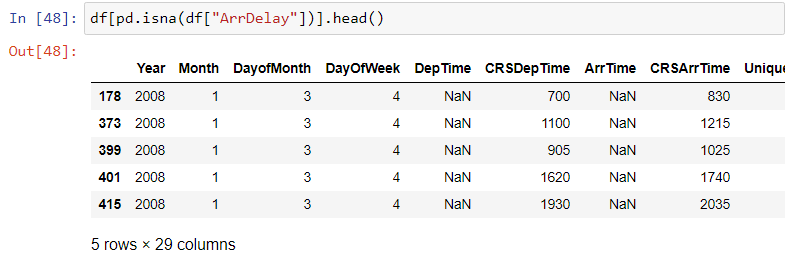
Si lo que nos interesase sería, por ejemplo, escoger diferentes orígenes a la vez, por ejemplo, Atlanta y Houston, podríamos usar esta misma sintaxis con la barra vertical o podríamos usar otro tipo de sintaxis. Vamos a ver cuál.



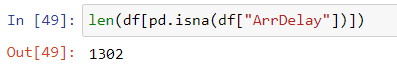
Podríamos usar aquellos vuelos con origen en, usar esta función y dentro de una lista ponerle todos los nombres que queremos. Y añadiríamos, como siempre, 'head'.

Para terminar esta introducción al filtrado de bases de datos**, podríamos usar otro tipo de filtro que no dependiese del "data frame" con el que trabajamos, sino que fuese, por ejemplo, con el mismo paquete Pandas, usando una función**.

Por ejemplo, **si los datos están perdidos**, **'isna', 'not available'**, e introduciríamos aquí pues la columna que queremos filtrar. Por ejemplo, 'ArrDelay'. Nos permitiría ver todos aquellos casos donde no tenemos ningún valor en la columna 'ArrDelay'.



Si nos interesa saber cuántos valores en concreto, podemos usar la instrucción **'len**'. Vamos a pegar el filtro que tenemos antes. En este caso, nos permitiría saber **cuántas filas** de nuestra base de datos no tienen ningún valor en la columna 'ArrDelay'.



En resumen, hemos visto cómo seleccionar filas o columnas, así como aplicar filtros a nuestros datos basándonos en valores o en condiciones múltiples.

(En los ejemploas anteriores, no me dejaba bajar para ver todas las ccolumnas porque había estbalecido nwos = 100000)

## Transformaciones de la base de datos [ ]

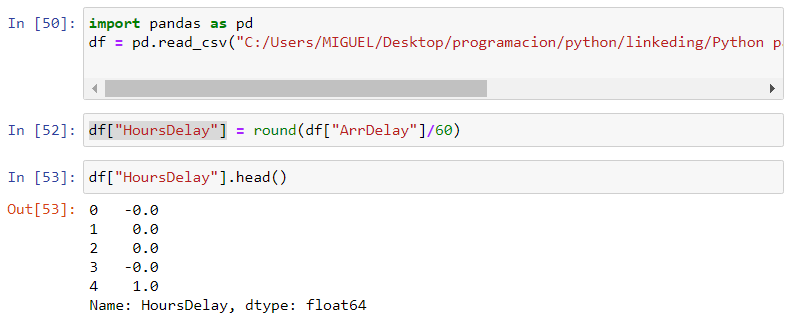
En este vídeo, vamos a ver las principales transformaciones que nos permite aplicar Python, y más concretamente Pandas, en nuestras bases de datos.

**¿Qué entendemos por transformación**?

**Una transformación es una modificación de la estructura de nuestra base de datos**.**Por ejemplo, añadirle filas, quitárselas, crear una columna nueva, etc…**

Vamos a empezar viendo **cómo se crea una columna nueva**.

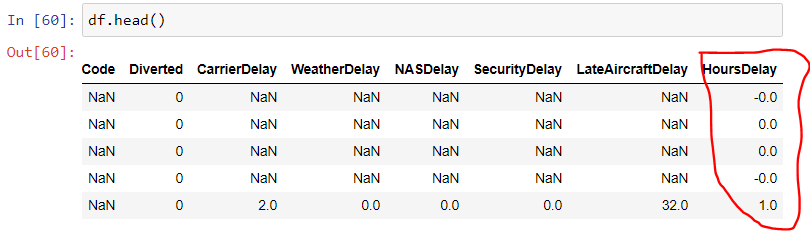
Para crear una columna nueva, debemos darle un nombre. Así, por ejemplo, yo voy a crear una que se llame 'HoursDelay' y será una transformación. Concretamente va a ser el redondeo con esta función 'round' de la columna ya existente, 'ArrDelay', dividida por 60.

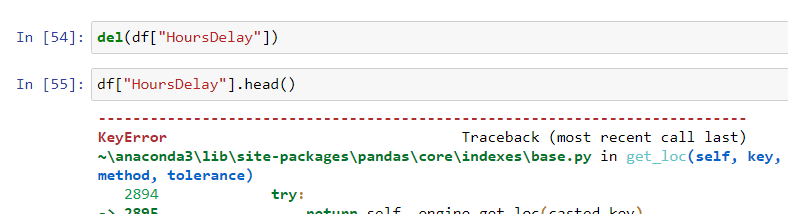


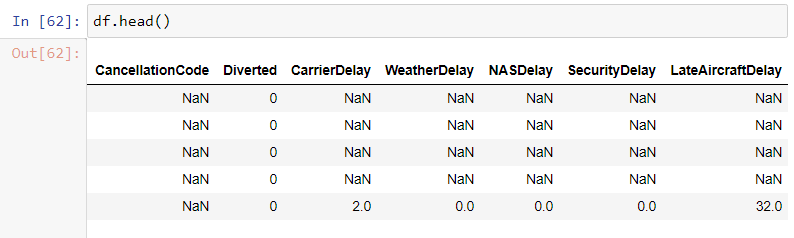
Una vez hemos ejecutado este código, la transformación ya se ha realizado, aunque no la veamos.

Así que, si queremos verla, tendremos que hacer... Y nos lo mostrará. Como podemos ver, ha creado los nuevos valores y ha creado unos particularmente especiales, que son el - 0. Esto no tiene que preocuparnos, porque sencillamente es que Python entiende que un redondeo por debajo de a cero, que sea cero, lleva el signo negativo. Esto lo interpreta así y lo va a mostrar así, pero será un cero, si queremos realizar cualquier tipo de cálculo.

Si, por otro lado, lo que **queremos es eliminar una columna en concreto**, por ejemplo, la que acabamos de crear, porque no nos gusta que el cero aparezca con un signo negativo delante, aunque no importe, podemos usar esta sintaxis que es muy sencilla. Es sencillamente **'del', "delete"**. La ejecutamos y nuestra base de datos, aunque no muestre nada, ya no tiene esta columna. Podemos comprobarlo sencillamente haciendo el 'head' de nuestra base de datos. Y vamos a ver que la columna que habíamos creado ya no está.

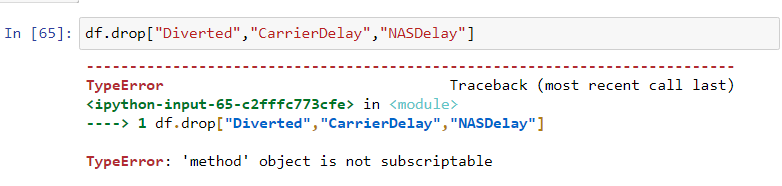




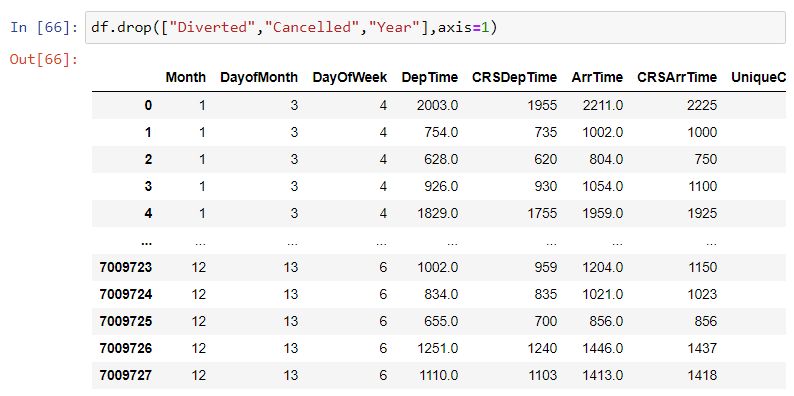


Esto podríamos verlo al final de nuestra base de datos. Por ejemplo, puedo volver a crearla ejecutando 'HoursDelay'. Ahora volvería a ejecutar el 'head'. Y veríamos que al final de todo tenemos esta nueva columna. Ahora podemos mantenerla porque tampoco nos molesta.

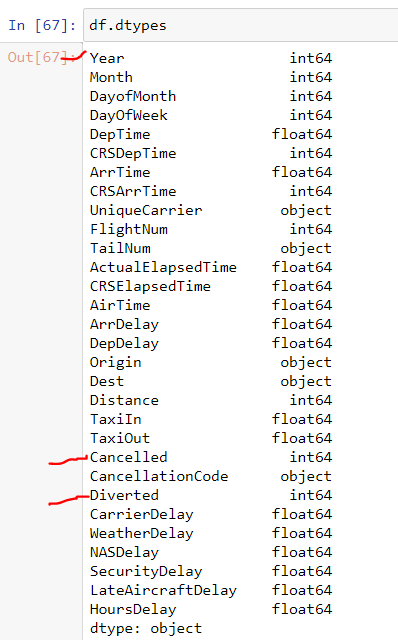
Si lo que queremos, por otro lado, es **borrar varias columnas a la vez**, esto vamos a hacerlo con una función que es **'drop' y dentro de 'drop' vamos a crear una lista con todos los nombres que no queramos. Por ejemplo, 'Diverted', 'Cancelled', y 'Year', por ejemplo**. Si ejecutamos esta función así, nos va a dar un error, porque no le hemos especificado dónde queremos borrarlas.

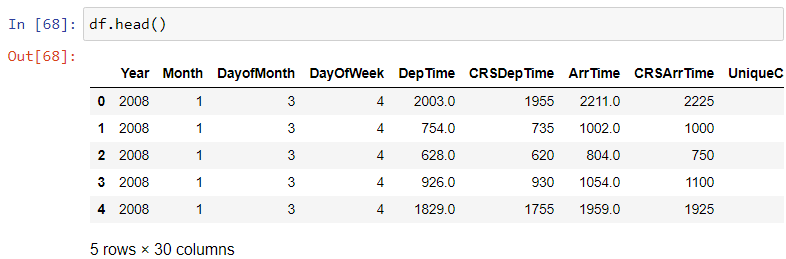


Así que tendremos que ponerle el eje y ejecutarlo. Esto puede tardar un poco. Pero nos va a devolver la base de datos. Nos está mostrando la base de datos, donde ya no tenemos la columna 'Year' ni 'Cancelled' ni 'Diverted'.

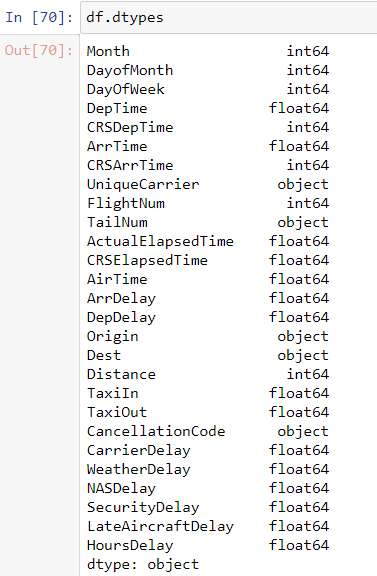


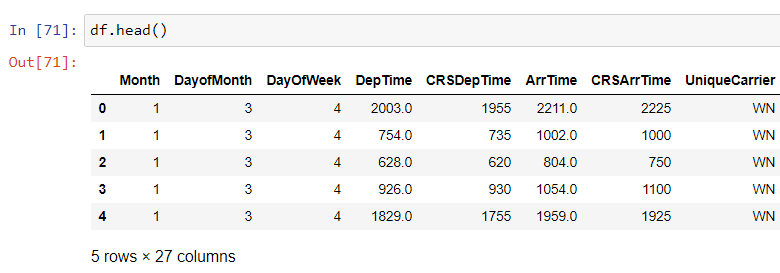
Pero si ahora vamos al final y mostramos el 'head', como hemos hecho siempre, vamos a ver que 'Year' sigue apareciendo.





Esto ocurre porque **no hemos igualado a nada nuestra función 'drop'. Entonces lo ejecuta, pero no lo guarda. Si lo que queremos es que se borre en nuestra base de datos que tenemos guardada, tenemos que igualarlo en nuestro "data frame" y aplicarle la función**. Si ahora ejecutamos esto, ahora ya ha borrado de nuestra base de datos y ya no será accesible.



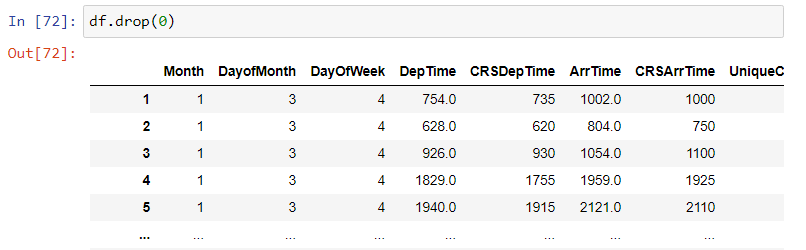


Otra **alternativa que podríamos usar para borrar columnas** sería usar esto mismo y aquí especificarle que, en el mismo objeto donde le estamos aplicando la función, queremos que lo borre. Entonces así estaríamos obteniendo exactamente lo mismo que con la fila anterior.



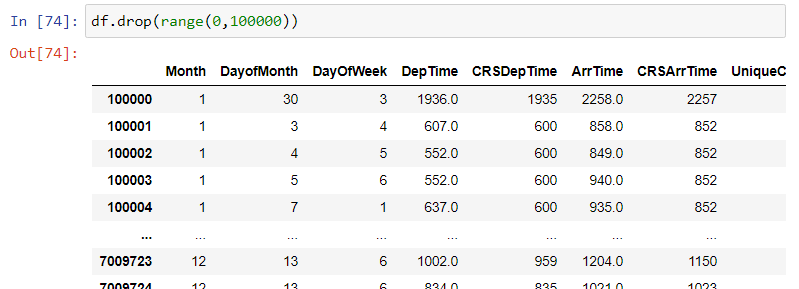
No vamos a ejecutarlo, porque estas columnas ya las hemos borrado y nos devolvería un error.

La función **'drop' también funciona para eliminar filas**. Así que si ejecutamos 'df.drop' y le ponemos 0, por ejemplo, vamos a estar eliminando la primera fila de nuestra base de datos.



Como estamos viendo, cuando ejecutamos esto nos muestra el 'head' sin la fila número 0.

Si quisiésemos eliminar muchas filas de golpe, esto podríamos hacerlo también con 'drop'. Dentro de la función, tendríamos que especificarle un rango con la función 'range'. Por ejemplo, las primeras 100 000 filas. Esto puede tardar un poco.



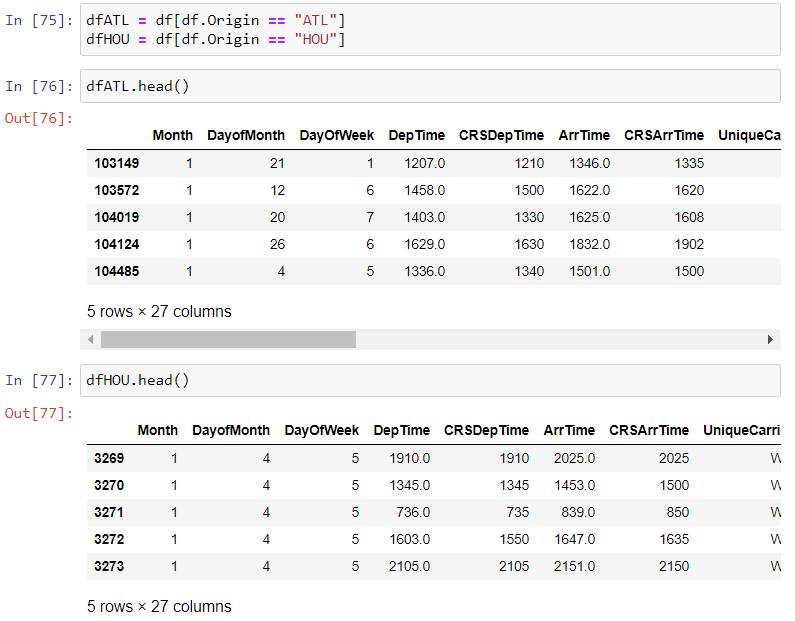
Pero recordemos que esto no lo ha guardado porque no hemos igualado a nada. Si quisiésemos guardar esto, a partir de la fila 100 000 tendríamos que añadirle el 'inplace=true' o poner 'dt=df.drop', etc.





Vamos a ver **cómo se añaden nuevas filas a una base de datos**.

Esto, para hacerlo, vamos a crear dos bases de datos. Una, por ejemplo, con todos los vuelos que tengan un origen en Atlanta. Y, por otro lado, todos los vuelos –con otro nombre, por supuesto– que tengan un origen en Houston. Si lo ejecutamos, ahora tenemos dos nuevos objetos, 'data frame Atlanta' y 'data frame Houston', que son solo con las filas que tengan los orígenes que hemos especificado.

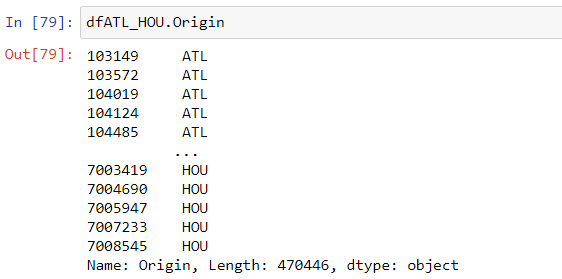


Si lo que queremos es ahora **sumar estos dos "data frames**", lo que tendríamos que hacer es, por ejemplo, en el "data frame" de Atlanta, le añadimos con la **función 'append**' el "data frame" de Houston. Esto podemos hacerlo porque ambos "data frames" tienen la misma estructura de columnas.

Y entonces lo que podríamos crear es 'newdf' y, si lo ejecutamos, esta 'newdt' va a tener exactamente todas las filas con un origen en Atlanta o Houston.



Podemos comprobarlo. Si hacemos 'newdf', 'Origin', podemos ver que primero tenemos todos los de Atlanta, ya que hemos hecho 'dfATL.append' y luego hemos añadido al final aquellas del "data frame" de Houston.



En este vídeo hemos visto las transformaciones más básicas que podemos aplicar a nuestras bases de datos, añadiendo o quitando filas o creando nuevas columnas.

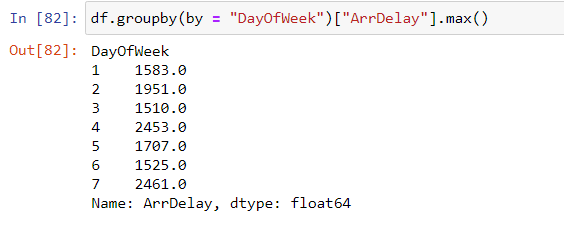
## Groupby: obtener información esencial

Vamos a mostrar una de las funcionalidades más interesantes que nos ofrece el paquete Pandas, el 'groupby'.

El **'groupby',** que seguro que te resultará familiar si alguna vez has trabajado con bases de datos relacionales, como el SQL, **es una herramienta de resumen de nuestros datos que nos permite realizar operaciones matemáticas sencillas agrupando por categorías**.

La manera más directa de entender su funcionamiento es viendo un ejemplo práctico, así que vamos a trabajar con la base de datos de aerolíneas completa y estudiar el retraso de los pueblos al llegar a su aeropuerto de destino.

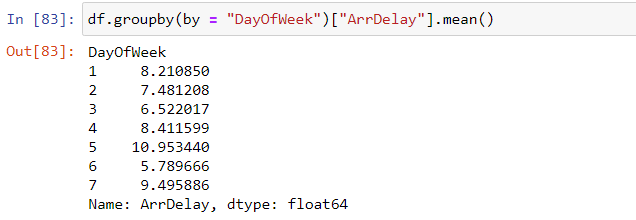
La primera instrucción que quiero mostrarte es esta, que depende de tres elementos principales. La instrucción se aplica sobre nuestro "data frame" y va a depender de los tres parámetros que he comentado. El primero va a ser un nombre de columna, el segundo otro nombre de columna y el tercero una función. Vamos a empezar con el máximo, por ejemplo. Y ahora lo ejecutamos.



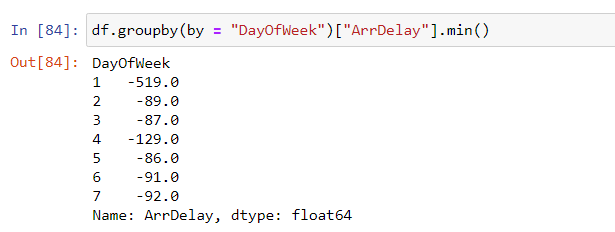
¿Cómo interpretamos este resultado? El 'groupby' lo que está haciendo es nos agrupa por cada día de la semana, lo vemos aquí, 'by = "DayofWeek"', y nos da un valor de esta columna, de los retrasos de los vuelos, y le aplica una **función, el máximo**. ¿Cómo interpretamos esto entonces?

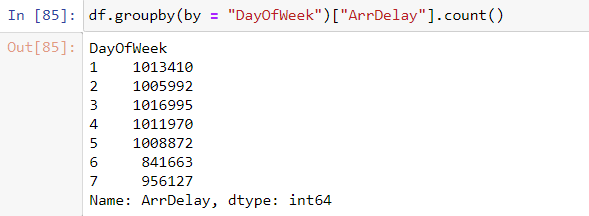
Esto puede interpretarse como el retraso máximo que ha tenido cada día de la semana en toda nuestra base de datos.

Hemos escogido la función 'max', pero podríamos haber escogido cualquier otra de las que nos ofrece el paquete Pandas. Por ejemplo, podríamos llamarlas **función 'mean'** y nos daría la media de retraso para cada día de la semana.

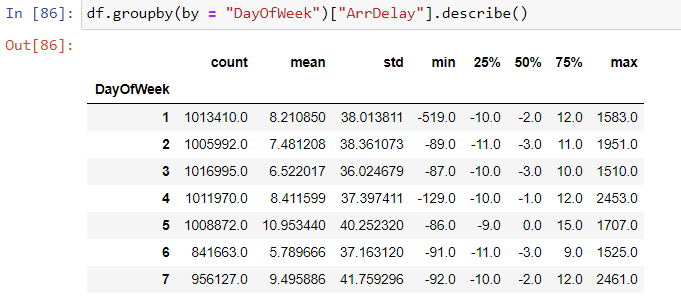


Otras funciones que podrían interesarnos sería el **mínimo**, por ejemplo. También podría interesarnos el número de casos que tenemos. Podemos ver que en fines de semana hay menos vuelos, por ejemplo.



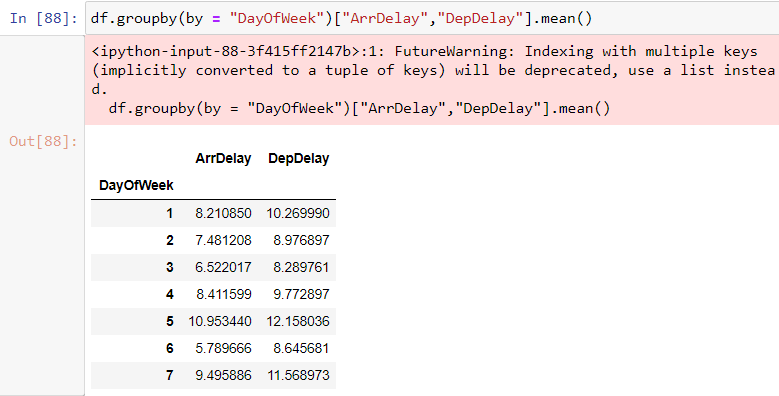


Y quizás la más interesante de todas sea **'describe'**, que lo que hace, y por eso tarda más, es mostrarnos un **resumen estadístico** agrupado para cada uno de los días de la semana. Aquí podemos encontrar resumida la **información de cuántos casos hay, la media, la desviación estándar, el mínimo, los tres cuartiles, la central es la mediana, y el máximo**.



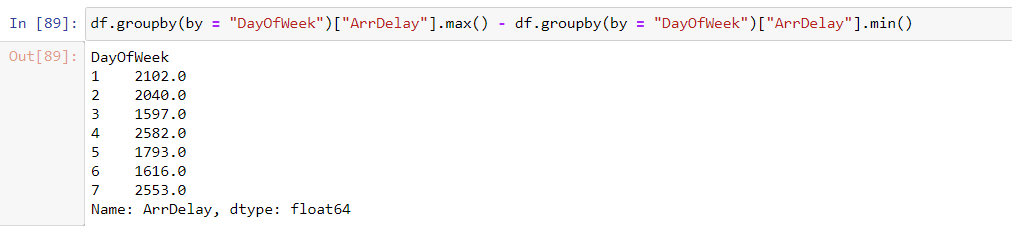
Este solo es un ejemplo de los más básicos del tipo de análisis que podemos realizar con la función 'groupby'. Vamos a ver algo un poco más complejo.

Copiamos esta instrucción y le añadimos otra columna aquí. Vamos a calcular la media sobre estas columnas y ejecutamos este código.



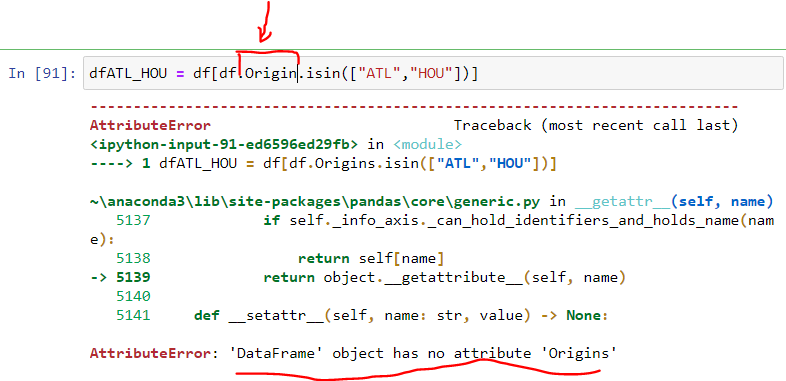
Obtenemos el mismo resultado que anteriormente, pero ahora podemos ver el resumen simultáneo y **comparado de estas dos variables**. Esto podemos realizarlo con tantas columnas como queramos, pero es preferible que, si lo hacemos, **estemos seguros de que estas columnas son del mismo tipo**. Si no, estamos comparando variables que son enteras con variables que son decimales o, por ejemplo, variables de texto con variables enteras, lo que generaría un "output" difícil de leer.

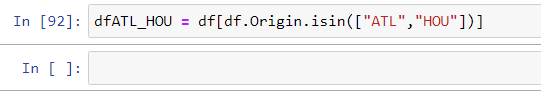
Es muy interesante saber también que podemos realizar operaciones con nuestros objetos 'groupby'. Por ejemplo, vamos a usar el máximo que hemos visto hasta ahora y vamos a restarle a este mismo objeto el mínimo.



Esto lo que nos va a permitir es ver el rango de valores para cada día de la semana. Si vemos el "output", vemos que le distancia mayor es el jueves y la menor es el miércoles.

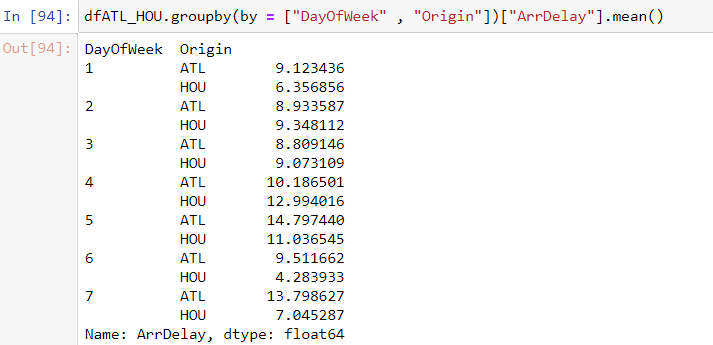
Para realizar el siguiente tipo de análisis, que es un poco más complejo, vamos a crear una nueva base de datos, que la llamaremos 'dfATLHOU'. Vamos a seleccionar solamente los vuelos que salgan de estas dos ciudades. Esta función, como puedes ver, no nos devuelve nada, pero tenemos un nuevo "data frame" creado.





Ahora lo que vamos a hacer es aplicarle una función 'groupby' a este nuevo "data frame". Y aquí vamos a variar un poco la sintaxis. Vamos a explorar otra vez el día de la semana y también el origen.

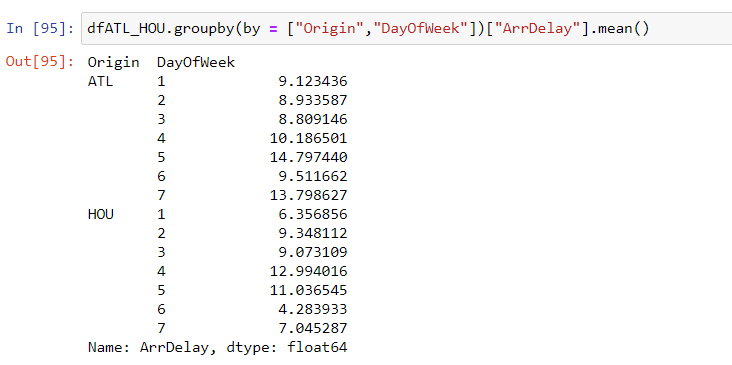
Hemos reducido esta base de datos para que este resultado sea más agradable de ver. Si no, en la base de datos original tenemos miles de orígenes y entonces es muy complicada. Y aquí vamos a volver a usar el retraso de llegada de los vuelos y vamos a comparar, por ejemplo, las medias otra vez, que es la medida más popular. Ejecutamos y podemos ver una nueva tabla, que es un poco más complicada de leer, pero que nos ofrece mucha más información.



Aquí lo que estamos viendo es para cada día de la semana por separado aquellos vuelos que salen de Atlanta y aquellos vuelos que salen de Houston y sus medias.

Esto permite comparar, por ejemplo, en cuál de las dos ciudades los vuelos llegan más retrasados y en qué día las diferencias son mayores.

Para este tipo de análisis, el orden es muy importante, ya que así copio y pego la misma instrucción que tenemos aquí y en vez de poner de 'DayofWeek', 'Origin', vamos a hacerlo al revés y vamos a poner 'Origin, 'DayofWeek'. Volvemos a ejecutar y aquí el "output" que obtenemos es muy parecido, pero las conclusiones que podemos sacar a simple vista son muy distintas



Aquí comparar entre Atlanta y Houston el lunes, por ejemplo, no es nada intuitivo porque tenemos que ir a mirar la primera entrada en la octava. En cambio, es mucho más sencillo obtener más o menos una evolución de cómo cambia el retraso por días de la semana dentro de cada uno de los dos aeropuertos.

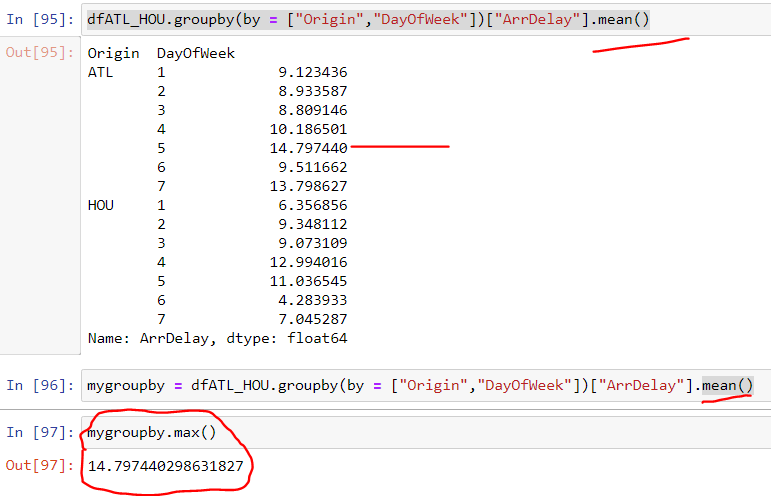
Hago esta distinción porque **el orden en el que pongamos las columnas en el 'by' va a afectar al tipo de conclusiones que podamos sacar o, como mínimo, a la velocidad y la intuitividad de estas**.

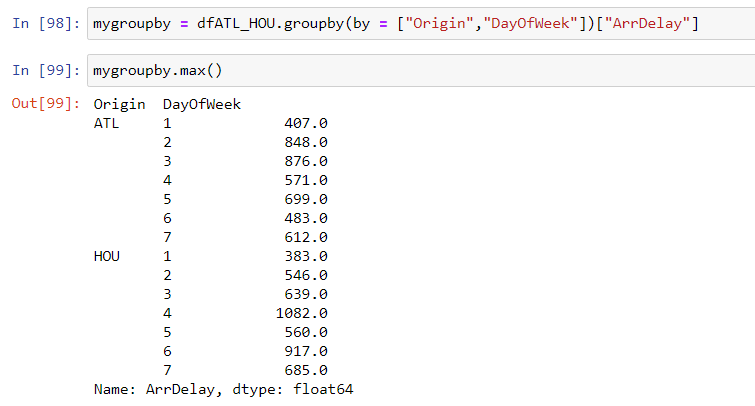
Por último, comentar que **los objetos 'groupby' pueden guardarse**. ¿Esto qué quiere decir?

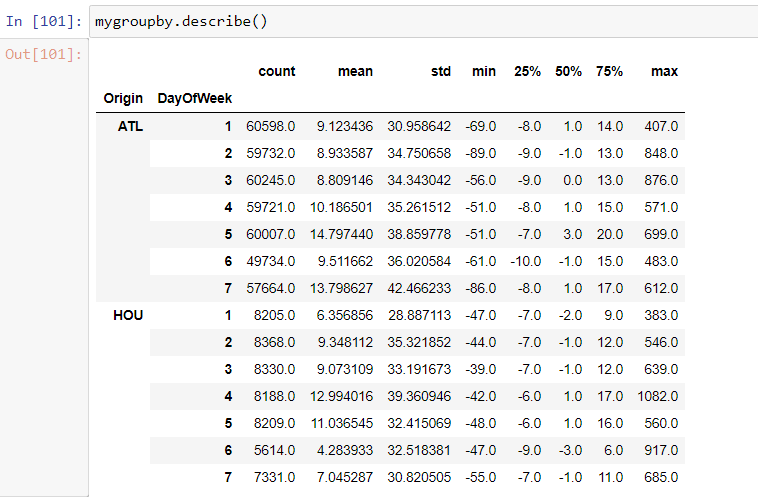


Si pegamos este objeto, por ejemplo, yo puedo guardarlo y decirle 'groupby'. Lo ejecuto y no voy a tener ningún "output", pero aun así yo puedo explotar este objeto. ¿Cómo puedo hacerlo? Si ejecuto este comando de aquí, lo que vamos a ver es que nos devuelve el mismo tipo de tablas que habíamos visto anteriormente. ¿Por qué es interesante esto?

Porque ahora podemos modificar aquí mismo las funciones. Es mucho más rápido y claro para nuestros análisis si usamos este tipo de estructura.







## Tratar datos duplicados y perdidos

En los siguientes minutos vamos a explorar dos temas fundamentales para el análisis de datos:

**la gestión de la información duplicada y la gestión de la información faltante o "missing data**".

**Vamos a empezar hablando de los casos de duplicados**, que, a no ser que se trate de casos muy específicos, donde nos interese contabilizar cuántas veces se repiten los casos concretos, **el procedimiento habitual es eliminarlos**.

Vamos a ver un ejemplo de código sobre cómo gestionar estos datos duplicados. Para hacerlo, hemos cargado un millón de datos. He usado esta sintaxis, que es 10 elevado a las 6, así que estoy cargando un millón de datos.



El código principal que vamos a usar **es 'drop\_duplicates'**. Pero primero vamos a generar una base de datos donde estemos seguros de que hay muchos datos duplicados.

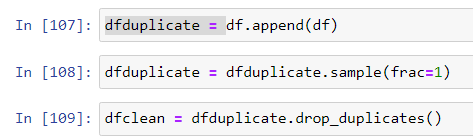
Para hacerlo, vamos a crear un nuevo "data frame", que va a ser el "data frame" original, por duplicado.



para que quede claro que no estamos haciendo ninguna trampa, vamos a reordenarlo

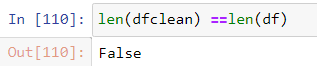


y ahora vamos a aplicarle la función que va a eliminar los casos faltantes. Vamos a llamarle 'dfclean' y se lo clicamos a la base de datos artificial que acabamos de generar.



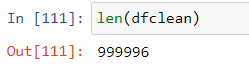
**Con esta sencilla instrucción ya hemos eliminado todos los casos duplicados, no necesitamos ningún código más.**

Lo que es también interesante y normalmente necesario es comprobar si el número de filas que tienen nuestro "data frame" nuevo es igual al número de filas que esperábamos que tuviese, es decir, las que tenía el original.



Como podemos ver aquí, esto no es verdad. Nos está diciendo que el número de filas del nuevo "data frame" es distinto al número de filas del "data frame" original. ¿Qué ha pasado?

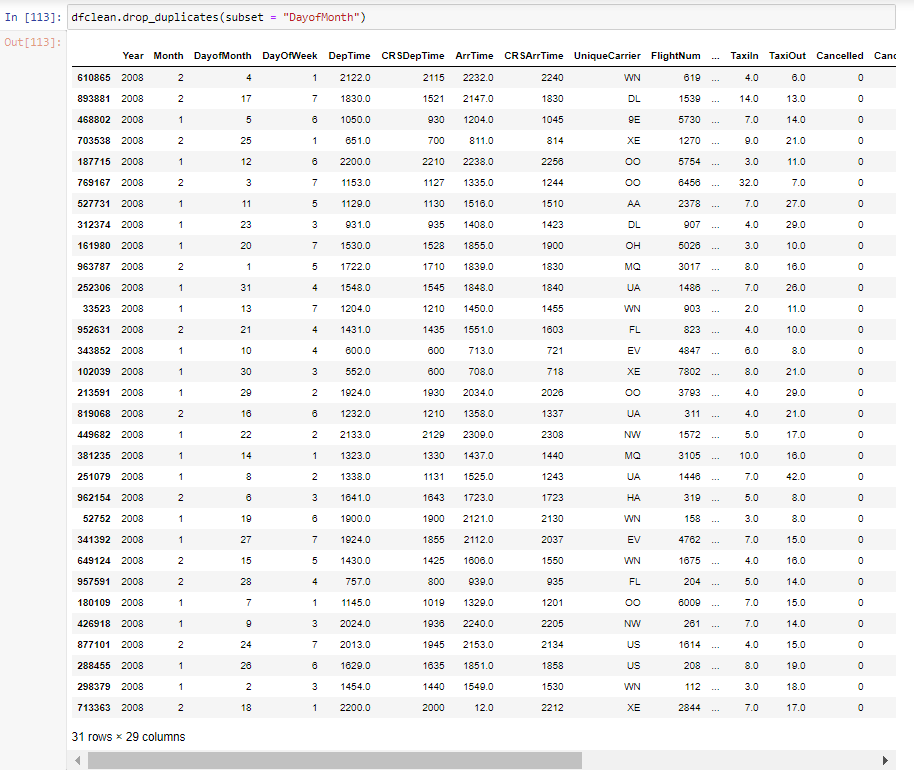
Esto podemos comprobarlo evaluando cuántas filas tiene nuestro nuevo "data frame", y vemos que tiene cuatro filas menos de las que debería tener si el "data frame" original no tuviese filas repetidas.



Lo que acabamos de ver aquí es que el "data frame" que hemos leído, que nosotros suponíamos que no tenía filas repetidas, en realidad sí que tenía cuatro.

Para terminar este primer tema, vamos a ver un ejemplo de eliminación de datos un poco más específico, aprovechando las opciones de la **función 'drop\_duplicates**'.

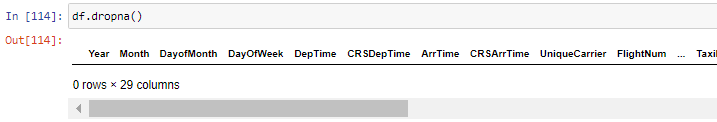
Vamos a aplicar esta función sobre nuestro nuevo "data frame" y vamos a usar el **parámetro 'subset'**, que lo que va a hacer es buscar duplicados sobre la columna que le especifiquemos, por ejemplo, 'DayofMonth'.



Si ejecutamos, podemos ver que ha mantenido una sola fila para cada distinto valor de 'DayofMonth', así que nos hemos quedado con una base de datos con tan solo 31 días.

A continuación, vamos a ver **cómo gestionamos los datos faltantes**. La gestión de datos faltantes es muy sencilla, solo depende de una sola **función dropna()**

Si ejecutamos esta función sobre nuestro "data frame", vamos a ver que obtenemos un "data frame" vacío. ¿Por qué ha ocurrido eso?

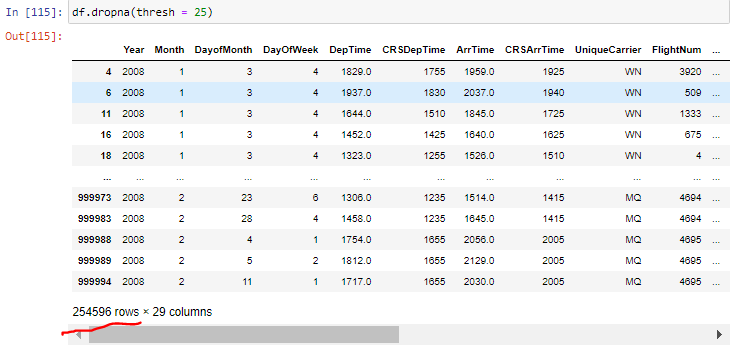


Básicamente porque ninguna de las filas de nuestro "data frame" no tiene ningún dato faltante.

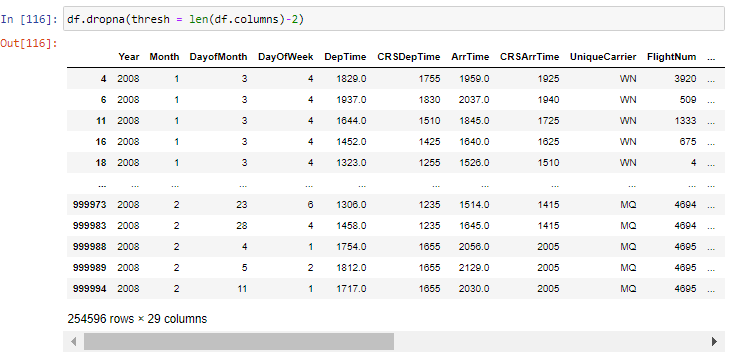
Esto es debido a que, si el vuelo está cancelado, no tiene hora de llegada. Y si tiene hora de llegada, el vuelo no está cancelado. Así que por fuerza alguna de estas dos columnas va a tener un dato faltante.

Afortunadamente tenemos varias opciones dentro de esta función que permiten no ser tan radicales a la hora de gestionar nuestros datos faltantes.

Podemos usar el **parámetro 'threshold'** y especificarle cuántas entradas de todas las columnas que tenemos queremos que estén llenas de información relevante. Por ejemplo, podríamos decirle "quiero 25 entradas llenas de las 29 que hay".



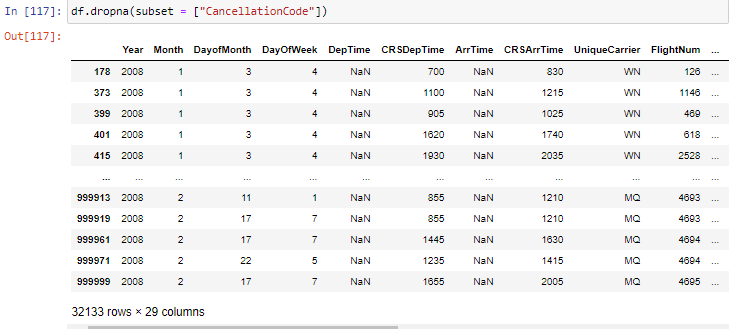
Si ejecutamos esta instrucción, vemos que ha mantenido la cuarta parte de información de nuestro "data frame". Una buena manera de usar este código, por ejemplo, sería en vez de poner 25, ponerle el número de columnas que tenemos en nuestro "data frame" menos una, por ejemplo, o menos dos, lo que nos interese. Y así si nuestro "data frame" incrementa en tamaño, podemos decirle: "vale, solo quiero que tenga como máximo dos valores faltantes".



Vemos que el resultado es el mismo, pero básicamente porque las columnas que tienen datos faltantes van en bloque.

**Si lo que nos interesa es filtrar por una columna en concreto, podemos usar la misma función y usar el parámetro 'subset', y dentro de una lista ponerle el nombre de columna por el que queremos filtrar**.

Así, nos va a devolver un "data frame" donde no tenga NaN en esta columna(“CancellationCode”).



Tan solo 32 000 filas.

En resumen, hemos visto cómo funcionan las instrucciones principales para la gestión de datos duplicados y datos perdidos. Esto no implica que Python no ofrezca métodos más avanzados para la gestión de estos, como la imputación de valores cuando nos disponemos de ellos, pero aquí te he ofrecido una primera aproximación a su tratamiento.

## Introducción a la librería Numpy

Voy a usar este vídeo para presentarte los fundamentos del paquete Numpy.

**Numpy es un paquete orientado al análisis matemático y científico de datos, que es uno de los más populares que hay disponibles para Python**.

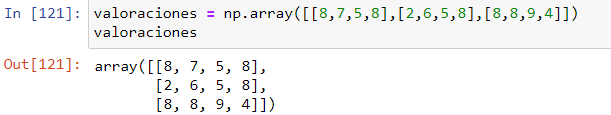
Este paquete viene incorporado dentro de la plataforma Anaconda, así que no tendremos que instalarlo.

Vamos a importarlo utilizando la sintaxis estándar.



Lo primero que te voy a presentar es como crear el objeto más popular de este paquete, que son los **"arrays".** ¿Qué es un "array"? Ahora lo veremos. **Es muy parecido a una matriz**.

Vamos a usar estos valores y os voy a enseñar cómo se visualiza.



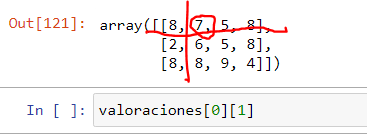
Esto es un "numpy array". ¿Qué tenemos aquí?

Tenemos básicamente una matriz con dos ejes: un eje vertical, donde tenemos cuatro columnas, y un eje horizontal, donde tenemos tres filas.

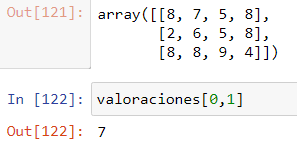
¿Cómo vamos a interpretar los datos que aquí tenemos? Voy a presentaros un ejemplo donde tenemos tres clientes, cada una de las filas, que han valorado cuatro productos, cada una de las columnas. ¿Cómo se accede a la información? Vamos a verlo.

Hay dos maneras de hacerlo, os las presento ambas.

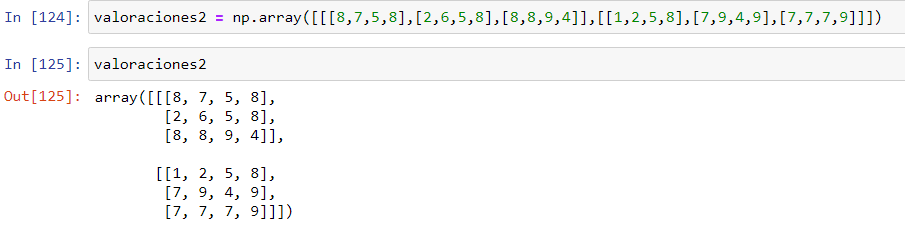
La primera es esta de aquí. ¿Qué le estamos diciendo? Que queremos acceder al primer elemento de la primera dimensión y al segundo elemento de la segunda dimensión. Como podéis ver, la primera dimensión son las filas, así que vamos a coger la primera fila, y la segunda dimensión son las columnas. El segundo elemento es esta de aquí, así que estamos cogiendo el primero y el segundo, que es este siete, como aquí vemos.

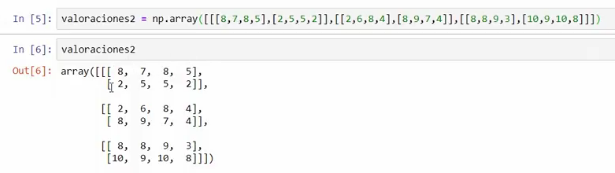


Esto tiene una notación alternativa que podemos hacer separando por comas. Es exactamente lo mismo.



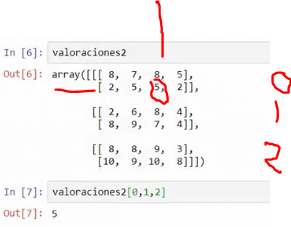
Vamos a ver que, si vamos añadiendo nuevas dimensiones, podemos ir añadiendo aquí nuevos corchetes y abajo nuevas comas. Vamos a ver un ejemplo de cómo creamos una red con más dimensiones. Os he pegado el código. Vamos a llamarle 'valoraciones2' y aquí vamos a tener tres dimensiones. Ahora vamos a verlas. Lo mostramos por pantalla, ejecutamos y ahora os explico qué es lo que vamos a tener aquí.





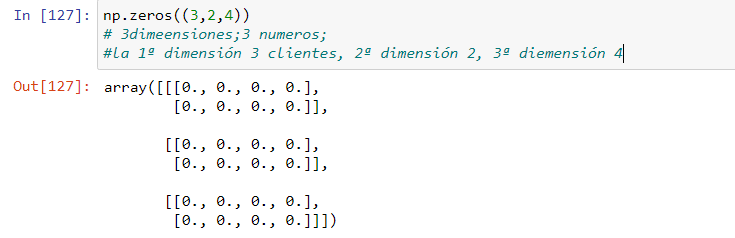
Aquí vamos a tener lo mismo, tres clientes –uno, dos y tres–, cuatro productos, uno para cada columna, pero cada cliente habrá valorado estos productos hace un año y este año.

Así, con este tipo de estructura, podemos ir incrementando la dimensionalidad de nuestros datos y podemos mantener el tipo de análisis que queremos. Como te he comentado, ahora podemos acceder a los elementos con esta sintaxis, por ejemplo, '[0,1,2]'. Nos va a mostrar el primer elemento de la primera dimensión, el segundo elemento de la segunda dimensión y el tercer elemento de la tercera dimensión.

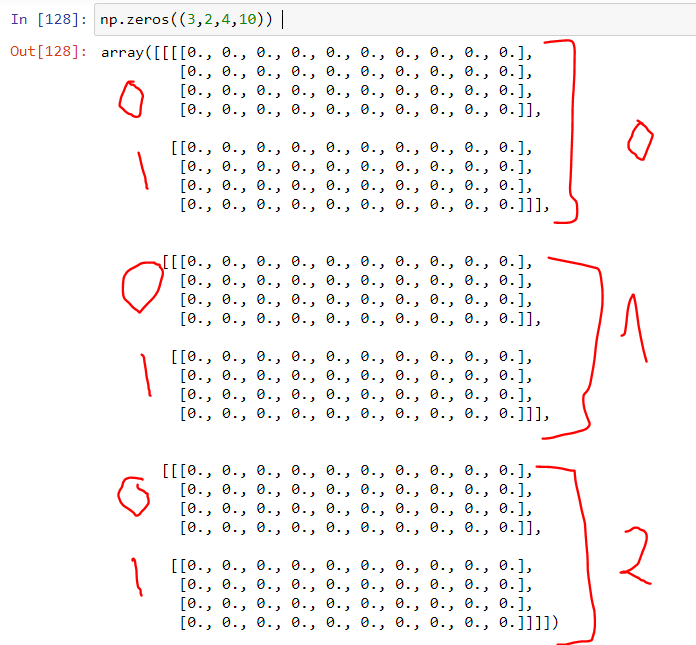


**Recuerdo que el cero es el primer elemento siempre en Python**.

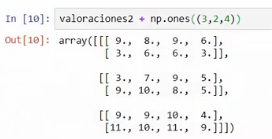
Otra funcionalidad interesante que podemos utilizar es crear "arrays" vacíos. Por ejemplo, podemos usar esta **función, 'zeros'**, y especificarle las dimensiones que queremos crear. Por ejemplo, vamos a crear tres dimensiones, es decir, tres números. La primera dimensión vamos a querer tres clientes, dos y cuatro, por ejemplo. Vemos que acabamos de generar la misma estructura que teníamos antes pero vacía.



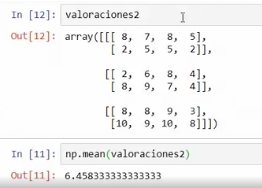
Esto, como te puedes imaginar, es escalable a cualquier tipo de dimensiones. Podemos ponerle 10, 5... Y si ejecutamos, vamos a ver que hemos creado un objeto difícil de interpretar, pero que tiene las dimensiones que le hemos pedido.



Continuamos viendo algunas de las funcionalidades de este paquete. Vamos a ver ahora que **podemos realizar operaciones entre "arrays" de los mismos tamaños**. Vamos a coger 'valoraciones2' y vamos a sumarle –y aprovecho para mostrarte una nueva función–, vamos a sumarle a cada uno de los elementos un uno. Un uno porque hemos creado aquí un "array" con estas tres dimensiones y lleno de unos, así que, cuando lo ejecutemos, vamos a ver que tenemos el mismo "array" que antes, pero le hemos sumado uno a cada puntuación.

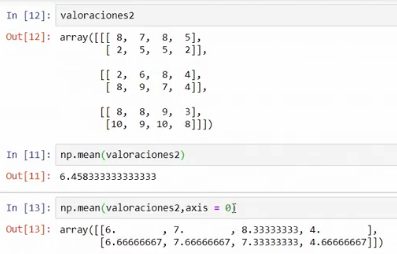


Los "arrays" son una estructura muy práctica con la que trabajar, ya que podemos aplicarle muchas funciones. Por ejemplo, la media al "array" entero, y nos devuelve la media de todos los valores que podíamos encontrar aquí, el original. El resultado anterior no lo hemos guardado.

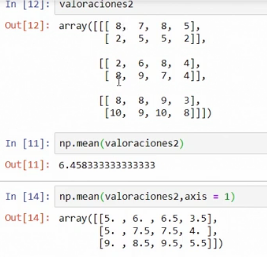


Así que todas las valoraciones de todos los productos clientes y los dos años que hemos comentado antes son 6,45.

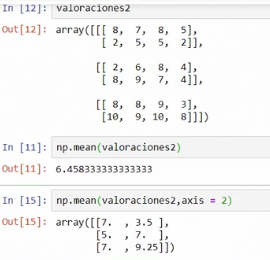
Pero esto no se termina aquí, ya que podemos hacer, por ejemplo, 'valoraciones2' y especificarle en qué eje queremos que nos calcule la media. Por ejemplo, ejecutamos para primer eje. ¿Qué ha pasado aquí? Nos ha calculado, por ejemplo, este 6 es la media entre 8, 2 y 8. Este 6,66 es la media entre 2, 8 y 10, y así sucesivamente.



Podemos cambiar el eje. Si hemos puesto el segundo eje, este 5 va a ser la media entre 8 y 2. El siguiente 5, entre este 8 y 2. El siguiente 9 entre el 8 y el 10.

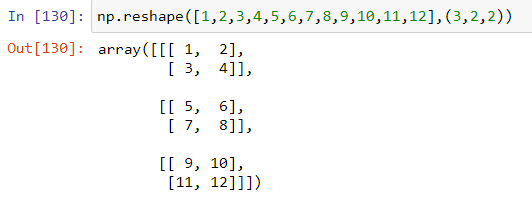


Y ya para verlos todos ejecutamos el último eje y podemos ver que las medias ahora se realizan de la siguiente manera. Este 7 es la media de todos estos valores. Este 3 y medio es la media entre todos estos valores y así vamos haciendo.



Os comento algunas funciones más. Una muy interesante es este que vemos aquí, el **'reshape**', que **nos permite convertir una lista, por ejemplo, esta lista del uno al 12 a las dimensiones que nosotros queramos**.

Ejecutamos y podemos ver que lo ha partido en tres dimensiones: una, dos, tres. Tres filas, dos columnas.



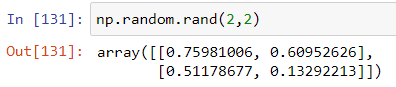
Tenemos que ir con cuidado de que estas dimensiones multiplicadas se ajusten al tamaño general de la lista que le hemos dado.

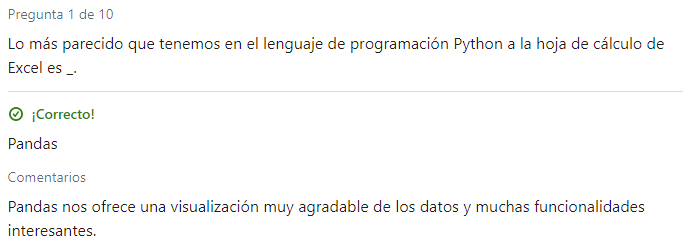
3\*2\*2 = 12

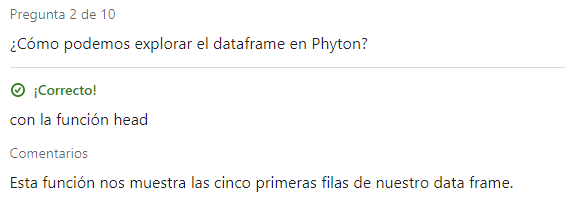
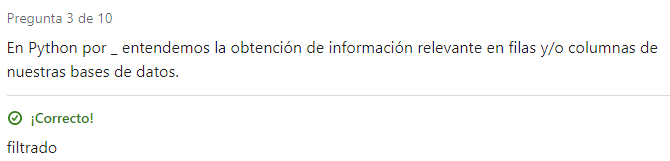
Para terminar, comentar que podemos usar este paquete, por ejemplo, la **función 'median'** sobre columnas de "data frames". Ahora no lo tenemos cargado, pero podemos poner 'Columna1'. Y si ejecutásemos esto, obtendríamos la mediana.

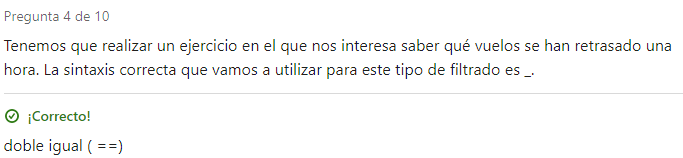
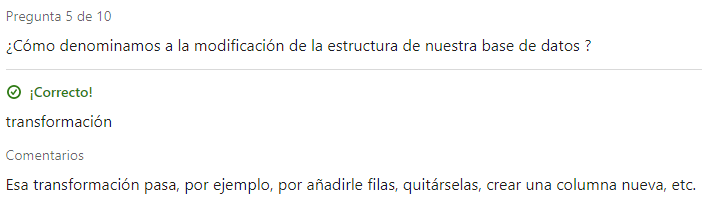


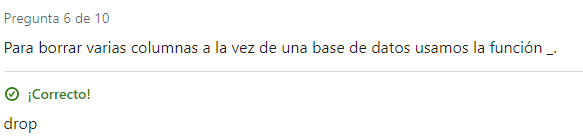
Dentro del paquete Numpy podemos encontrar **el módulo 'random'**, que nos permite **generar objetos aleatorios**. Aquí lo que hemos hecho es generar un objeto con dos dimensiones, cada una dos filas y dos columnas, y nos ha generado cuatro números aleatorios entre 0 y el 1.

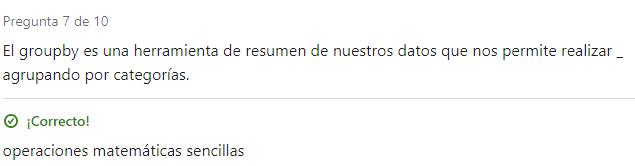


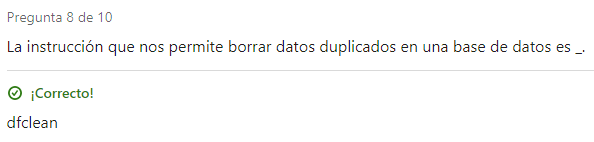
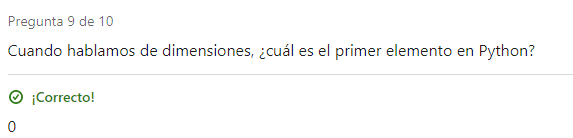
TEST

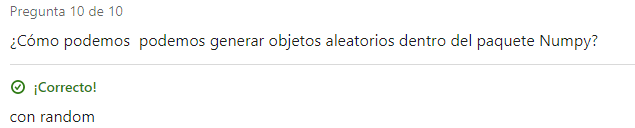
 







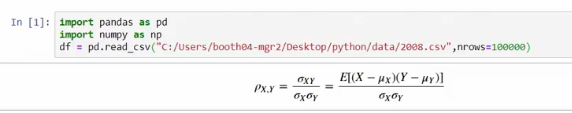
**GESTIÓN DE DATOS AVANZADA**

## Correlaciones. Entender las relaciones entre las variables

Si alguna vez has asistido una clase de estadística o has hablado con alguien que lo haya hecho, es muy probable que hayas escuchado la frase **"Correlación no implica causalidad"**.

Aquí voy a enseñarte **cómo detectar correlaciones en bases de datos**. Pero, como acabo de comentar, **esto no va a servirte para encontrar explicaciones, sino frases como "'a' está correlacionado positivamente con 'b'" y poder cuantificar esta relación.**

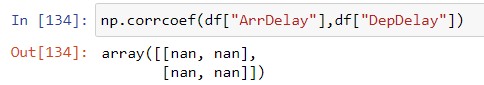
Pero ¿qué es una correlación? Una correlación es una relación lineal entre dos variables cuantitativas que toma la expresión que vemos por pantalla.



Y ¿cómo se interpreta? Se interpreta como el cociente entre la covarianza entre dos variables y el producto de sus desviaciones estándar. Esto puede tomar valores entre -1 y 1. Y como más cerca de estos extremos se encuentra el valor, más fuerte será la relación.

Normalmente los valores entre 0,3 y -0,3 son considerados muy bajos, y ya sea a partir de 0,6 o 0,7 en cualquiera de los dos signos cuando estamos hablando de correlaciones fuertes.

**Vamos a ver el** **código para encontrar relaciones entre dos variables**. Vamos a utilizar el paquete 'numpy' y vamos a coger dos columnas de nuestro "data frame". Ejecutamos y obtenemos un "array" de datos perdidos. ¿Por qué ha pasado esto?

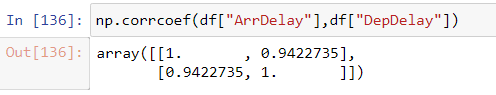


Ha pasado esto porque el cociente de correlación no admite valores faltantes. ¿Qué tenemos que hacer? O imputarlos o quitarlos.

Para hacerlo más sencillo, vamos a quitarlos. Vamos a hacerlo utilizando las opciones del paquete 'pandas', usando 'df. dropna (implace)=True' de un 'subset', que son las dos columnas que hemos cogido antes.

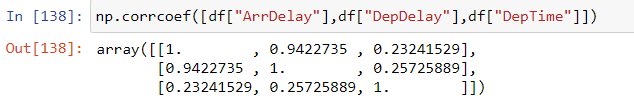


Ya los hemos quitado y ahora podemos volver a ejecutar la instrucción anterior y vemos una **matriz de correlaciones,** una matriz entre las dos columnas que hemos seleccionado. Estas matrices siempre van a ser simétricas y un elemento consigo mismo siempre va a tener una correlación de uno.



Lo que estamos viendo aquí con un 0,94 es que estas dos variables están altísimamente correlacionadas, lo que tiene mucho sentido, ya que es retraso de un avión al salir con el retraso de un avión al llegar.

Podemos complicar nuestros análisis añadiendo nuevas columnas, por ejemplo, 'DepTime', y encontraríamos las mismas relaciones entre las variables que ya teníamos y unas nuevas correlaciones.



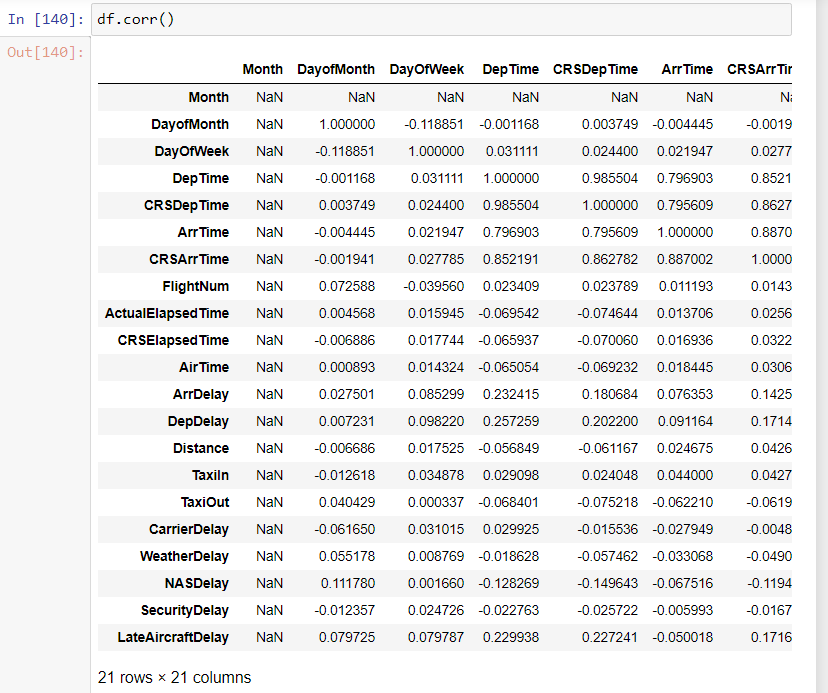
Esas correlaciones son mucho más flojas.

El paquete 'pandas' también ofrece funciones para calcular nuestras correlaciones. Esta es **la función, corr()**

**lo que primero tendríamos que eliminar aquellas columnas que no sean numéricas**. Así que vamos a hacerlo con la instrucción 'drop (impliace = True' de aquellas columnas que claramente o son constantes o son, por ejemplo, variables "dummy", como 'Cancelled' o 'Diverted'. La primera es 'Year'...



Ejecutamos. Y, **ahora sí, ya vamos a poder mostrar la matriz de correlaciones**.



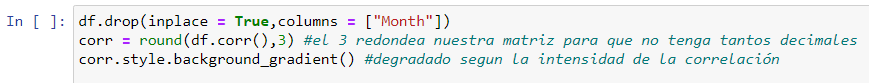
Esto es difícil de interpretar.

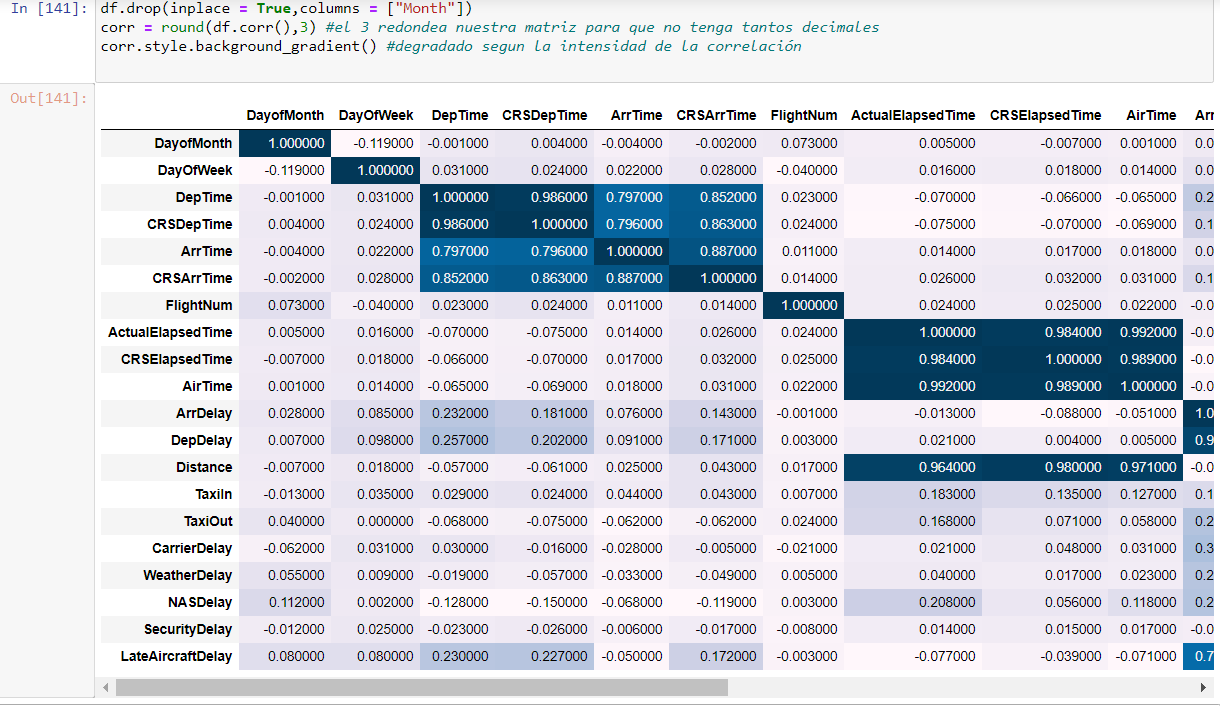
En este caso, vemos que 'Month', aunque sea numérica del 1 al 12, no está categorizada como tal, así que cuando ejecutamos el método correlación sobre nuestro "data frame", nos devuelve una columna y una fila de 'Not a Number'.

Aparte de eso, nuestra matriz tiene unos en toda la diagonal, como vemos aquí, y podemos estudiar una a una las correlaciones entre cada una de las columnas que tenemos. Esto puede ser muy costoso, pero tenemos toda esta información a nuestro alcance.

(Estaría guay poder añadir colores a eta matriz en función de una mayor o menor correlación)

Por último, vamos a visualizar gráficamente está matriz. Para hacerlo, vamos a quitar primero la columna 'Month', que antes la hemos dejado y nos ha generado este error, que para un gráfico no es nada recomendable. Vamos a redondear nuestra matriz para que no tenga tantos decimales y vamos a aplicarle este estilo. Este estilo sencillamente lo que va a hacer es mostrarnos un degradado según la intensidad de la correlación. Ejecutamos y podemos ver de manera gráfica cuáles son las relaciones que son más intensas.





Aquí vemos que hay unas cuantas variables que están muy correlacionadas entre ellas. Esta función es especialmente útil si lo que queremos es hacer una exploración inicial de nuestra base de datos porque nos permite encontrar relaciones interesantes que quizás no habíamos considerado.

## Test de la Chi-Cuadrado

Vamos a ver en este vídeo un **ejemplo práctico de cómo realizar el famosísimo test de 'x' cuadrado**, que supuso una revolución al permitir un criterio sólido para **encontrar relaciones entre variables cualitativas** usando las tablas de contingencia.

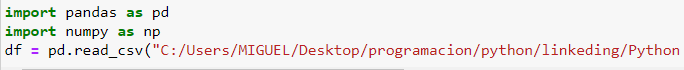


El fundamento de este test es comparar la suma de diferencias al cuadrado entre lo que estamos observando, es decir, nuestros datos reales, y aquello que cabría esperar en una situación donde no existe ninguna relación entre variables.

Esto permite cuantificar la magnitud de todas estas distancias, compararlas con una distribución, que es la de 'x' cuadrado, y decidir si podemos afirmar que estas variables están relacionadas significativamente.

Una de las **debilidades** de este test es que **no permite cuantificar cada una de las relaciones entre categorías, pero permite afirmaciones globales del tipo: "fumar está relacionado significativamente con el cáncer de pulmón".** Y, aun así, una exploración visual de las tablas permite determinar en qué dirección se da esta relación.

después de esta breve introducción, vamos a ver un ejemplo de código sobre datos de aerolíneas estadounidenses. Lo primero que vamos a hacer es importar dos paquetes fundamentales: el 'pandas' y el 'numpy'. Después vamos a leer toda nuestra base de datos.



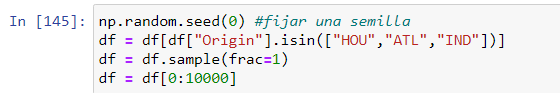
Lo siguiente que vamos a hacer es **fijar una semilla**.

**np.random.seed(0)** #fijar una semilla

¿Qué estamos haciendo cuando fijamos una semilla? Estamos haciendo que, cuando quiera replicar este código, vas a obtener exactamente los mismos resultados que yo cuando apliques funciones aleatorias.

Para simplificar la interpretación de los resultados, vamos a crear un "subset" de nuestros datos. ¿Qué "subset"? Vamos a seleccionar solamente aquellos casos, aquellos vuelos, que tengan un origen en uno de estos tres aeropuertos que vemos aquí: el de Houston, el de Atlanta y el de Indiana.

Una vez aplicado este filtro, lo que vamos a hacer es reordenar nuestra base de datos con esta instrucción de aquí, 'sample'. Vamos a aplicarle 'frac=1', aunque luego vamos a seleccionar las primeras 10 000 filas.

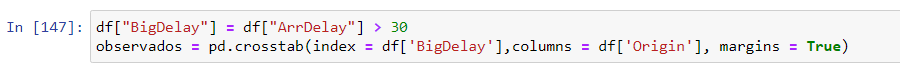


¿Por qué estamos seleccionando solo 10 000 filas? Porque un test con millones de filas no tiene demasiado sentido, ya que siempre vamos a encontrar algún tipo de relación, por pequeña que sea, y aquí hay millones de casos.

Vamos a ejecutar estas líneas y, a continuación, vamos a generar una nueva variable.

Esta nueva variable va a distinguir entre dos casos: va a tener valor 'true' cuando el 'ArrDelay' sea más grande que 30, que implicará que tenemos un gran retraso de este vuelo. Por otro lado, va a tener un valor 'false' cuando sea más pequeño o igual que 30.

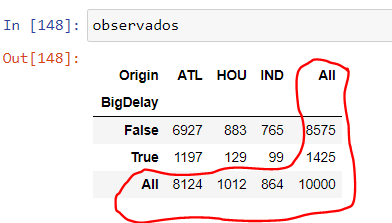
La siguiente instrucción es un poco más compleja y lo que estamos haciendo es **crear una tabla de contingencia.** Vamos a ver qué estamos haciendo concretamente.



Estamos seleccionando dos columnas de nuestros "data frame". ¿Qué columnas? Las dos que queremos estudiar. Primero, 'BigDelay', que es la que acabamos de crear anteriormente. Y otra columna, que es 'Origin'.

Como podemos ver, el parámetro del que depende es 'columns', así que podemos crear una tabla de contingencia de múltiples dimensiones, aunque aquí por simplicidad solo vamos a usar una dimensión, que es 'Origin', la que hemos filtrado anteriormente.

Con el valor 'margins=True' lo que le estamos diciendo es que queremos que guarde también los márgenes de la tabla. Ahora vamos a ver lo que significa esto. Ejecutamos estas líneas y vamos a ver su resultado.

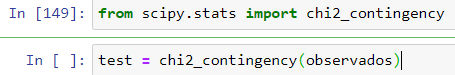


**Esto es una tabla de contingencia clásica**. ¿Cómo se interpreta? Este es 6 900, es el número de vuelos que se han retrasado poco y que han salido de Atlanta. El resto de celdas se interpreta de la misma manera excepto los márgenes.

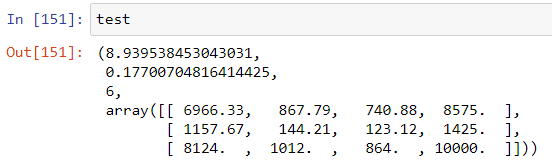
¿Qué son los márgenes? Los márgenes son la columna de más a la derecha, esta, y la fila inferior. El margen inferior es la suma por cada una de las columnas y el margen derecho es la suma para cada una de las filas. El valor que encontramos abajo a la derecha es la suma de todas las observaciones.

A continuación**, vamos a importar el paquete que va a realizar el test**. Es este. Depende del paquete, 'scipy' se llama 'chi2', "square", 'contingency'. Ya está importado.

Y vamos a pedirle que nos cree un objeto a partir de la tabla de observados que hemos generado nosotros mismos.



Esto no genera ningún "output", pero podemos ver que es el objeto que hemos creado. Esto nos devuelve varia información.

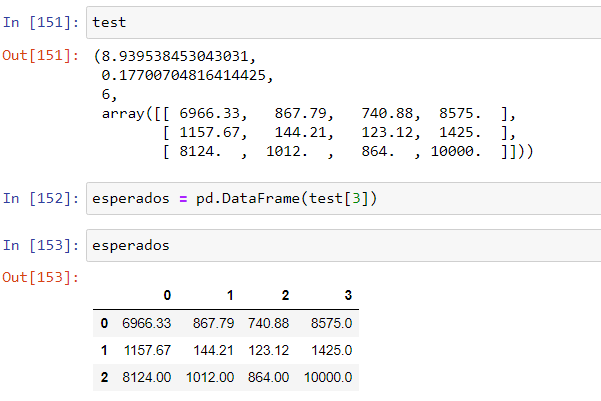


El primero de todos es el estadístico, es decir, la suma de las diferencias al cuadrado.

El segundo de ellos es **el p-valor**, luego vamos a hablar de él. El p-valor es una medida que permite tomar decisiones basadas en algún criterio.

Y finalmente una tabla. Esta tabla es la tabla de valores esperados, que es lo que vamos a comparar con nuestra tabla de valores observados. Esto el test lo va a hacer por nosotros, pero podríamos realizarlo manualmente.

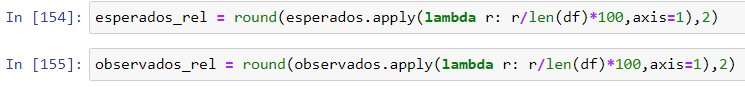
Vamos a convertir esta tabla en un "data frame" para que la visualización sea más atractiva y vamos a mostrarla.



Lo que vemos aquí son los valores teóricos esperados si no hubiese ningún tipo de relación. Lo que está claro es que estos valores no son posibles, ya que son decimales. Esto solo es una aproximación teórica, pero va a servir para comparar con los valores que hemos observado.

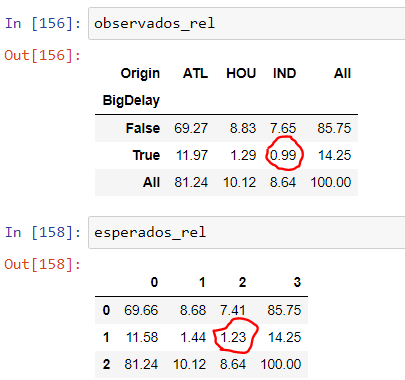
A continuación, lo que vamos a realizar es una tabla con los valores esperados relativos y una tabla con los valores observados relativos para que sea más fácil de comparar.

Esto vamos a conseguirlo usando la **función 'round'**, es decir, un redondeo para que sea más fácil de interpretar sobre cada una de las tablas aplicándole **una función 'lambda'**, es decir, una función sin nombre, que va a ser sencillamente cada uno de los valores dividido por el total de valores multiplicado por 100.



Así, estamos generando tablas con valores relativos y no valores absolutos, que son más difíciles de comparar.

Vamos a ejecutar estas celdas y vamos a mostrar sus resultados.

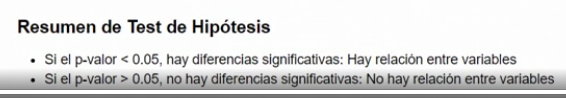


Aunque en la segunda tabla no hayamos mantenido los nombres, sí que se ha mantenido el orden, así que compararlo es relativamente sencillo. Una primera cosa que vemos es que los márgenes se han mantenido. Esto tiene que ser así. Si no, implica que algo hemos hecho mal ya que lo que no queríamos era modificar la cantidad de "trues" y "falses" que había en 'BigDelay' o la cantidad de vuelos que han salido de cada uno de los aeropuertos.

Entonces lo que hace el test por nosotros es comparar si este 1.23, por ejemplo, es suficientemente mayor a este 0,99 que vemos aquí. Nosotros podemos suponer que es poco o es mucho, pero no tenemos una medida. **Lo que nos ofrece el test de 'x' cuadrado es una herramienta precisa de decisión basada en la estadística, de si esta diferencia es muy grande o muy pequeña**.

De hecho, lo que va a hacer el test es comparar este elemento con este, este elemento con este, este con este, etc., sumarlos, elevarlos al cuadrado, dividirlos y luego va a generar un estadístico que va a comparar con una distribución.

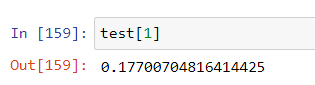
Aquí, al final de todo, vemos un criterio de decisión.



Voy a explicarlo muy brevemente, ya que el p-valor es un concepto complejo y no es el objetivo de este vídeo profundizar en el tema,

pero en este caso puede resumirse como la probabilidad de que las relaciones sean mucho más extremas de lo que hemos visto en nuestros datos si comparamos con la situación donde no hay ninguna relación. Dicho de otra forma, es un indicador de cómo de extraño sería ver nuestros datos si no hubiese relación entre variables. Un valor muy bajo, como vemos aquí, implica que hay relación significativa. Y un valor muy alto, que no podemos afirmar que la haya.

Vamos a ver cuál es el resultado concreto en nuestro test y en qué situación nos encontramos.



Como podemos ver, hemos obtenido un 0,17, que es bastante mayor que el límite que nos fijamos para decir que las diferencias son significativas, así que no podemos afirmar que haya relación entre variables.

## Análisis de datos extremos

En este vídeo vamos a ver **cómo detectar aquellos datos que toman valores extremadamente grandes o pequeños, los llamados "outliers"**.

Vamos a hacerlo explorando dos maneras de seleccionarlos.

Por un lado, un criterio para una sola variable en la manera más clásica y, en un segundo lugar, una aproximación multidimensional, que, aunque no vayamos a entrar en detalle, te va a permitir hacerte una idea de cómo gestionar este tipo de funcionalidades.

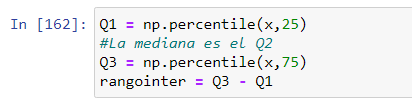
Para hacerlo, vamos a seleccionar 100 000 filas de nuestra base de datos.



Después vamos a extraer una columna, por ejemplo, 'ArrDelay', que son los retrasos de los vuelos, y vamos a **eliminar todos los datos faltantes de esta columna con esta instrucción. Dropna()**

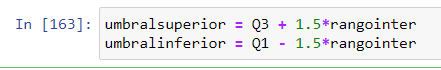


A continuación, lo que vamos a necesitar son los cuartiles. ¿Qué son los cuartiles? Los cuartiles es una medida estadística que lo que nos separan es el 25 % inferior de los datos del 75 % superior o el 75 % inferior del 25 % superior de nuestros datos ordenados. La sintaxis para obtenerlos es esta. Aquí estamos especificando que queremos que separe el 25 % inferior del 75% superior, el primer cuartil y aquí el tercero. Si estás familiarizado con la mediana, la mediana es el segundo cuartil. Entonces vamos a necesitar calcular el rango intercuartílico. ¿Qué es el rango intercuartílico? El rango intercuartílico es la diferencia entre el tercer cuartil y el primer cuartil.

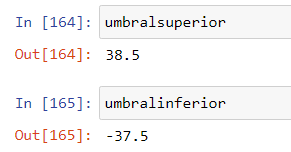


Pero ¿para qué necesitamos estas medidas? Un criterio clásico de selección de "outliers" es definir los umbrales.

Un umbral superior, que va a ser igual al tercer cuartil más 1,5 veces el rango, y un umbral inferior, que va a ser igual al primer cuartil menos 1,5 veces el rango.

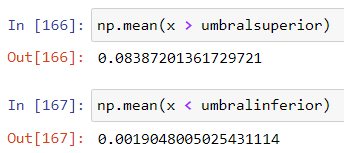


Ejecutamos esto y vamos a ver los resultados.



Lo que nos está diciendo este criterio es que cualquier valor que esté por encima de 38,5 va a ser un "outlier" en nuestra distribución. Lo mismo por cualquier valor que esté por debajo de menos 37,5.

Si queremos comprobar cuántos casos hay por debajo o por encima de estos umbrales, vamos a usar esta **función np.mean()** y vamos a ver, por ejemplo, por encima y nos da que el 8 % de los casos están por encima. Si comprobamos los que están por debajo, vemos que no llega al 2 %.



¿Qué estamos viendo aquí? Que estas medidas no son simétricas, es decir, nos da un umbral, un "threshold" que parte nuestros datos, pero no nos asegura que estén igualmente distribuidos por encima o por debajo.

Voy a mostrarte ahora una manera de estudiar varias variables a la vez. ¿Cómo vamos a hacerlo? Con **este paquete**. Cargamos aquí. Esto puede tardar un poco.



Vamos a definir un objeto, que le vamos a llamar "outliers", y vamos a usar esta **función, 'EllipticEnvelope'**.



Lo que estamos haciendo aquí es crear un modelo que nos va a seleccionar el 1 % de datos que considere que están muy alejados de nuestros datos más centrados.

Ejecutamos este modelo y vamos a definir una lista con un grupo de variables de nuestra base de datos, todas ellas son cuantitativas, y con este modelo estamos asumiendo que están distribuidas gausianamente. Es decir, son normales. Esto no tiene por qué ser así, pero es solo un modelo, entonces nos va a servir para hacernos una idea de cuáles son las más alejadas.



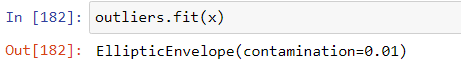
Ejecutamos, vamos a definir nuevamente nuestra variable 'x', que van a ser los valores que nos interesan en nuestra base de datos,



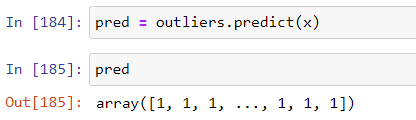
y vamos a entrenar nuestro modelo. ¿Cómo lo hacemos?

'Outliers', que es el objeto, 'fit' y con nuestros datos. Ya está.

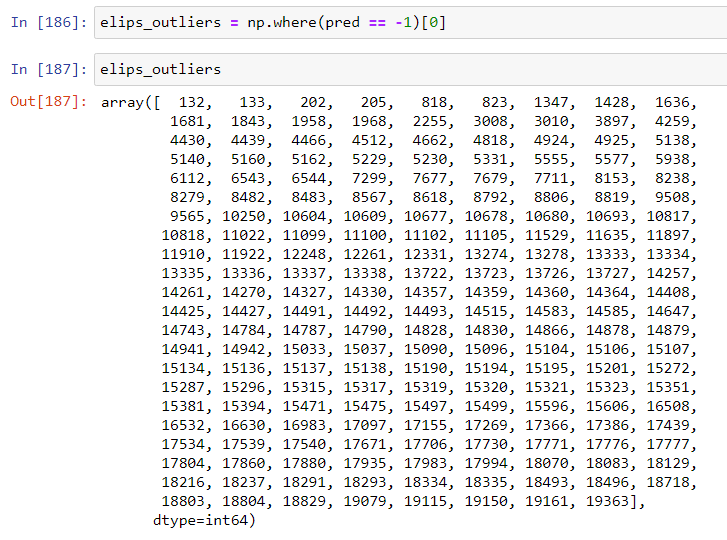




Lo que vamos a hacer ahora es pedirle que nos prediga con este modelo qué valores forman parte de este 1 % que queremos detectar. Me he dejado una 's'.



Podemos explorar este objeto y vemos que es un "array" de unos. No solo es un "array" de unos, es una array de unos y menos unos, y lo que nos va a interesar es detectar los menos uno. ¿Cómo podemos hacerlo? Usando este código.



Y lo que hemos conseguido es una lista con todos los "index" de aquellos casos que están muy alejados de los valores centrales de todas las variables estudiadas, es decir, nuestros "outliers".

No hemos profundizado en el porqué de este método, pero puede servir para detectar aquellas observaciones en nuestras bases de datos que sean demasiado extremas o que requieran especial atención. Por otro lado, siempre podremos explorar las variables una a una con el primer método que hemos comentado.

## Principios de las bases de datos relacionales

En este vídeo vamos a explicar brevemente**uno de los tipos de estructuras más famoso para gestionar y almacenar bases de datos**. **El más utilizado cuando se trata de bases de datos ya planificadas.**

Las principales características de este **modelo de almacenaje de datos** son las siguientes.

La información se divide en tablas que deben presentar un nombre y unos registros únicos.



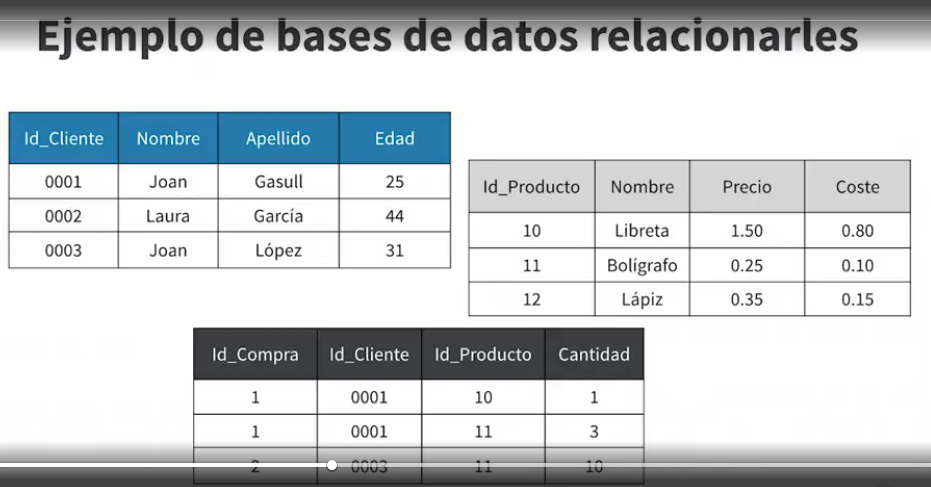
Esto implica que **no vamos a encontrar la misma información repetida nunca en ningún sitio**. Conceptualmente cada una de las tablas va a hacer referencia a una dimensión de lo que estamos estudiando, por ejemplo, clientes, productos y compras realizadas.

Las tablas se estructuran en **filas que hacen referencia a los registros o casos** y **columnas, que hacen referencia a los campos o variables**.

En este ejemplo podemos ver que las columnas 'Nombre' y 'Apellido' son columnas de texto y, por el otro lado, la columna 'Edad' son valores numéricos. Esto debe estar especificado cuando generemos nuestra tabla.

Las tablas están relacionadas entre ellas mediante las **claves primarias**, que identifican únicamente los registros de cada tabla. En esta tabla que vemos aquí, 'Id\_Cliente' es la clave primaria.

La unión entre tablas se debe realizar explícitamente y, en términos de bases de datos relacionales, se dice que estamos uniendo la clave primaria de nuestra tabla con la clave foránea de una tabla externa.



En el ejemplo que vemos aquí, la tabla inferior se une con las dos tablas superiores ya que 'Id\_Cliente' se enlaza con la superior izquierda e 'Id\_Producto' con la superior derecha.

Así, **si leemos la primera fila de la tabla inferior, podemos obtener la siguiente información**: el cliente Joan Gasull, de 25 años, ha comprado una libreta a precio de 1,50. Esta libreta, además, se produjo a 80 céntimos.

**Una parte muy importante de esta información sería redundante si un mismo cliente hubiese comprado cientos de productos cientos de veces, ya que en un formato de estilo de hoja de cálculo de Excel o un "data frame" clásico estaríamos guardando múltiples veces información como la edad o el precio, por ejemplo, cuando aquí solo la almacenamos una vez.**

**Las claves primarias** son un tipo de columna o columnas que deben identificar únicamente las filas de cada tabla –a las que llamamos **registros**, recordemos–, por lo tanto, **no pueden repetirse**.

Como podemos comprobar, fijémonos en la tabla inferior. **Las claves primarias no necesariamente deben ser únicas en una sola tabla**. Aquí aparece repetido el cliente número 1.

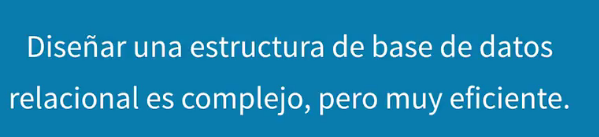
**Aunque sí que debemos encontrar en cada tabla una combinación única de claves primarias**. En nuestro ejemplo, la unión de 'Id\_Compra', 'Id\_Cliente' e 'Id\_Producto' generan un registro único, puedes comprobarlo.

Una de las principales virtudes de este tipo de estructuras de datos es que mantienen la integridad referencial de los datos. Es decir, si en algún momento eliminamos un cliente, por ejemplo, automáticamente vamos a eliminar todas las compras que este ha realizado.

Si eliminásemos al cliente número 1, estaríamos borrando también los dos primeros registros de la tabla de compras. Esto, por otro lado, no implicaría borrar los productos 10 y 11 de la tabla de productos, ya que, si no, estaríamos eliminando información que es relevante para otros clientes.

**Uno de los principales retos con los que podemos encontrarnos a la hora de gestionar un gran volumen de datos es qué estructura relacional construiremos para albergar la información**.

Aquí hemos visto un ejemplo con solo tres tablas y dos relaciones entre ellas para entender las principales características del modelo relacional. Pero diseñar e implementar una base de datos relacional real no es un proceso simple y requiere considerar muchas relaciones, a la vez que generar muchas claves primarias para mantener los principios de no duplicidad y la integridad de datos correctamente.



## Transformar un dataframe en una base de datos relacional

Vamos a ver un **ejemplo práctico de cómo transformar una base de datos clásica, de las que podemos encontrar en cualquier hoja de cálculo de Excel, por ejemplo, en una estructura de datos relacional**.

El método que voy a mostrar aquí no es único en absoluto, pero te va a permitir entender cuál es la idea detrás de las transformaciones que realizamos.

Vamos a usar una base de datos de juguete para ilustrar gráficamente los ejemplos que mostramos. Lo primero que vamos a hacer es transformar esto en un "dataframe" clásico. Vamos a usar esta **función, que permite la importación con este tipo de estructura con el que trabajamos**. Va a depender de "data", y los nombres de las columnas van a ser los "labels" que hemos especificado anteriormente.



Si mostramos los datos que acabamos de generar, vemos esta estructura.



Lo que aquí vemos es que tenemos mucha información repetida, por ejemplo, nombre y apellido, nombre y apellido, nombre y apellido... Esto no lo necesitamos. Edad tampoco. Tampoco necesitamos guardar todos los nombres múltiples veces ni los costes ni los márgenes.

Aquí hay muchísima información repetida, así que lo que vamos a realizar aquí es **partir esta tabla en tres subtablas más pequeñas que contengan solo la información necesaria**.

Lo primero que vamos a necesitar es transformar este "dataframe" para tener solo una sola fila para cada comprador. Esto podemos hacerlo de la siguiente forma: 'dataframe.drop\_duplicates'. Vamos a coger un 'subset' de la columna 'Comprador\_id'. Y vamos a mantener solo el primero, aunque aquí sería indiferente. Y vamos a ver lo que obtenemos.



Acabamos de obtener una tabla con toda la información que queremos, pero aun así estamos guardando muchísima más información de la que queremos si solo queremos guardar información de los compradores.

Así pues, lo que vamos a hacer es eliminar todas las columnas que no necesitemos en la tabla de compradores.

Vamos a hacer un "subset" de columnas sobre el mismo objeto, aunque podríamos cambiarle el nombre, y vamos a coger todas las columnas que nos interesen.

¿Qué columnas nos interesan? en este caso está muy claro: es toda la información que refiere a los compradores, que es su nombre, su apellido y su edad.

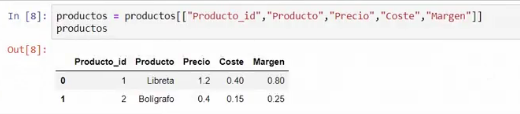


Vamos a ver los resultados. Como podemos ver aquí, acabamos de generar una tabla que contiene solo y exclusivamente la información que queremos, que son nuestros ID, el nombre, el apellido y la edad. Nada más.

Vamos a replicar este proceso para crear la tabla de productos y la tabla de compras.

En primer lugar, vamos a crear la tabla 'Productos' con un "subset" manteniendo solo una sola fila por cada "product id" que tengamos. No vamos a mostrar los resultados porque ya hemos visto lo que hace esta función.

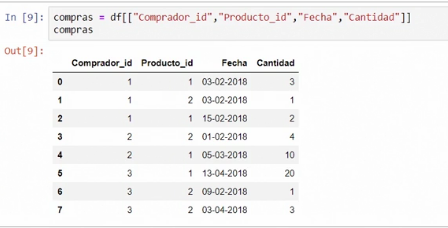
En segundo lugar, vamos a quedarnos solo con las columnas que necesitemos. ¿Cuáles son las columnas que necesitamos? Todas aquellas que hagan referencia a las dimensiones asociadas a cada uno de los productos. Vamos a ver los resultados.



Y podemos ver aquí que tenemos dos productos distintos, sus nombres, precios, costes y márgenes de beneficio. Aquí no tenemos ninguna información duplicada.

Para generar nuestra tabla de compras no necesitamos un proceso tan elaborado, ya que nuestro "dataframe" original ya respondía a cada una de las compras que se habían realizado.

Así pues, lo único que vamos a necesitar es seleccionar las columnas que queremos del "dataframe" original. ¿Qué columnas necesitamos? Necesitamos el ID del comprador, el ID del producto, la fecha, que conjuntamente con el 'Comprador\_id' y el 'Producto\_id' va a configurar la clave primaria de esta tabla, y finalmente la cantidad de producto que se ha comprado. Veamos finalmente el resultado.



Y aquí tenemos la información que va a unir la tabla de compradores con la tabla de productos.

Para resumir, hemos visto un ejemplo muy reducido de cómo crear nuestra propia base de datos relacional. Tenemos que ser conscientes de que, si tenemos muchísimas columnas, este proceso puede ser muy complicado, y controlar las tablas primarias y qué columna va a qué tabla puede ser una verdadera pesadilla.

## Joins. Trabajar con bases de datos relacionales

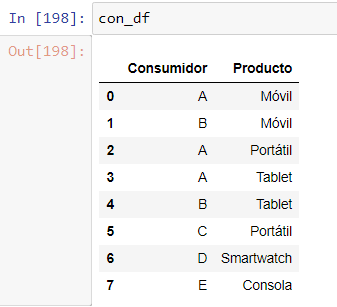
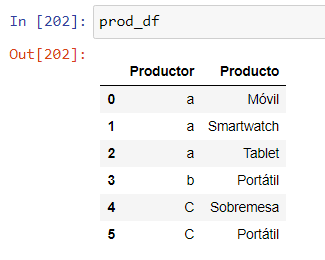
Vamos a ver cómo operar con las principales funciones para **unir distintas bases de datos en una misma estructura**. **Estas funciones se llaman "joins" y hay de distintos tipos**.

La teoría detrás de ellas es muy interesante, aunque aquí vamos a ver su funcionamiento con un ejemplo de juguete para entender en qué casos vamos a necesitar cuál de ellas.

El código que muestro aquí es la generación de dos pequeñas tablas, dos "dataframes" con dos columnas cada uno que vamos a unir de distintas maneras.



Lo primero que vamos a hacer es visualizar las dos tablas: los consumidores y, en segundo lugar, los productos.

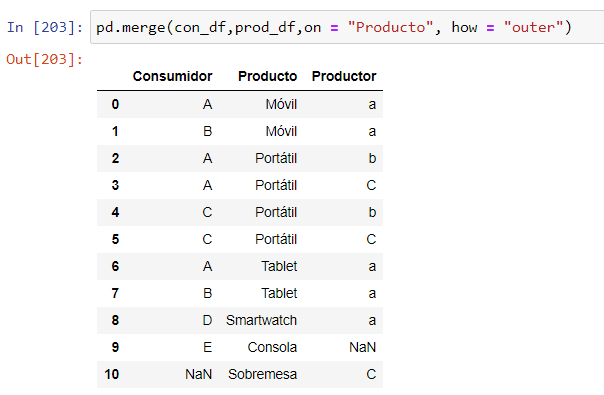
 

Como puedes ver, los resultados no están ordenados siguiendo ningún criterio. Ya lo he hecho a propósito para que quede claro que el orden no es importante aquí, sino la información contenida.

Vamos a ver cómo podemos unirlas. **La instrucción** principal que vamos a usar aquí es el '**merge**'. Es una instrucción que depende del paquete Pandas que nos va a permitir **unir las dos bases de datos**.

Vamos a usar primero los consumidores y después los productos. Esto va a depender de dos parámetros: el primero es 'on', que vamos a tener que especificarle un nombre de columna, y el segundo es 'how'. 'How' es lo que va a especificar qué tipo de "join" vamos a usar.

Vamos a usar primero uno de los más populares, que es el **"outer**". Ejecutamos y os muestro los resultados.

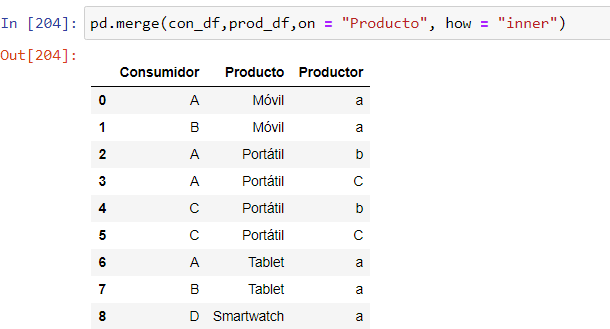


¿Qué ha hecho esta función? **Ha creado un "dataframe" con tres columnas**. Ha cogido la primera del "dataframe" de consumidores. La central que podemos ver aquí es la que ha unido entre las dos tablas distintas. Finalmente Productor la ha cogido de la tabla de productores.

Pero ¿qué ha pasado cuando no ha encontrado un cierto valor en las dos tablas? Por ejemplo, 'consola' es un valor que aparece en la tabla de consumidores –aquí lo vemos–, pero que no aparece en la tabla de productores. ¿Qué ha pasado? Lo ha añadido y, como no ha encontrado nada en la tabla de productores, ha puesto un "not a number". Lo mismo ha pasado con el producto 'sobremesa', donde ha puesto un "not a number" en la columna Consumidor, ya que 'sobremesa' aparece en la tabla de productores –aquí lo encontramos–, pero no aparece en la tabla de consumidores.

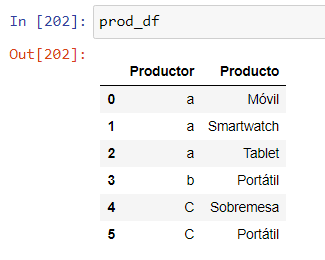
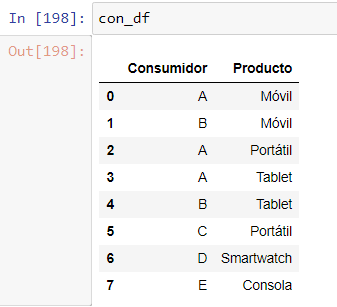
A continuación, vamos a ver otra vez la misma función, pero con un argumento distinto en el 'how'.

Vamos a usar **"inner**", que también es muy popular. Vamos a ejecutarlo y exploramos los resultados.



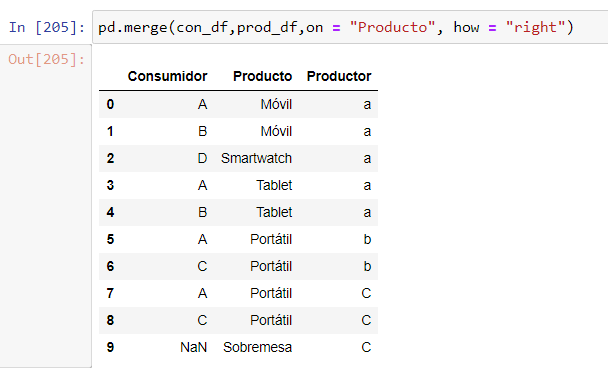
Lo que ha pasado aquí es que ha escogido todos los valores que aparecían en la columna Producto de las dos tablas y los ha adjuntado. Pero **si encontraba algún valor en la primera tabla que no estuviese en la segunda o en la segunda que no estuviese en la primera, lo ha omitido completamente**. Así, en esta tabla no vamos a encontrar ni sobremesa' ni 'consola'.

Podemos comprobarlo también al ver que esta tabla tiene nueve filas y, a su vez, esta tiene 11. Vamos a volver a mostrar las tablas iniciales, porque para el siguiente ejemplo necesitamos tenerlas a mano.



Y ya, para ir terminando, vamos a ver cómo funcionan el "left" y el "right join".

Vamos a usar la misma estructura que hasta ahora, pero ahora el orden va a importar. Por ejemplo, empecemos por el "**right"**. Ejecutamos y vamos a interpretar estos resultados.

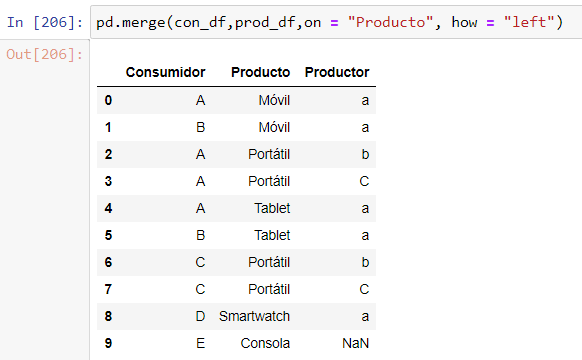


Lo que acabamos de hacer aquí es decirle **que nos junte los dos "dataframes", pero solo si encuentra la clave en el "dataframe" de la derecha**. Aquí vemos que nos ha puesto 'sobremesa', pero no nos ha puesto 'consola'. El "dataframe" de la derecha, aquí, es el de los productores.

Productores tiene 'sobremesa' y no tiene 'consola'. Y en los consumidores tiene 'consola' y no tiene 'sobremesa'.

Como podemos ver aquí, ha aparecido 'sobremesa' y no 'consola' porque le hemos dicho que nos **junte el de la izquierda con el de la derecha solo si sus valores aparecen en el de la derecha**.

Vamos a volver a ejecutar esto mismo, pero, en vez de "right", vamos a poner "left" y vamos a ver las diferencias.



¿Qué ha pasado? Exactamente lo mismo, pero en el sentido contrario. Ha cogido y ha buscado todas las claves de producto que había en este "dataframe", el primero que hemos puesto aquí, el de la izquierda (con\_df) , y ha mirado si estaban en el de la derecha (prod\_df). Así, no vamos a encontrar 'sobremesa', pero sí vamos a encontrar 'consola'. No lo he comentado antes, pero, como te puedes imaginar, si no encuentra un valor en la tabla que está juntando con la que queremos, pone un "not a number" como ha hecho con las versiones anteriores.

Aquí solo hemos visto unos ejemplos muy básicos de los distintos tipos de "join" principales. El objetivo es ofrecerte las herramientas para ser capaz de decidir qué tipo de unión de datos quieres realizar y planificar cómo vas a gestionar la información faltante que se genere.