Bases de Python para Ciencia de datos

Data Science Python Weekend

Néstor Montaño

Sociedad Ecuatoriana de Estadística

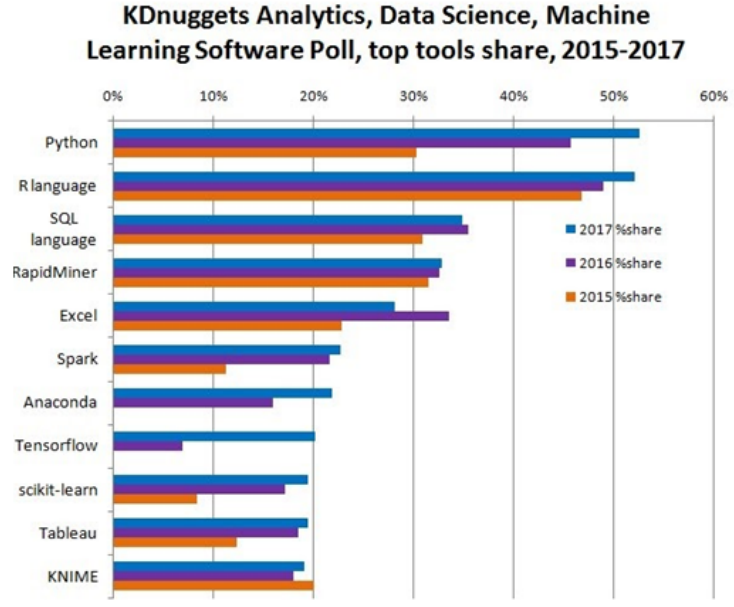
Noviembre-2020


SOCIEDAD
ECUATORIANA
ESTADÍSTICA

Nota:
Con *Alt* + *F* o *Option* + *F* puede hacer que estas dapositivas ocupen todo el navegador (es decir que se ignore el aspecto de diapositiva que tiene por default la presentación)

¿Por qué Python?

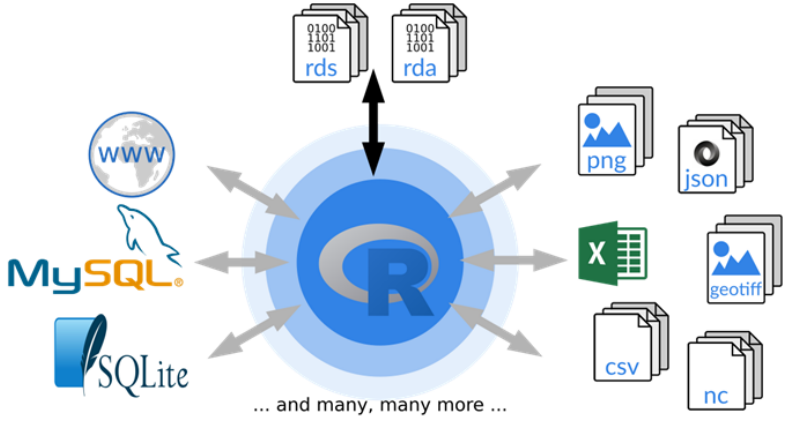
- Software Libre (Open Source), gratuito y de desarrollo independiente,
- Es un lenguaje de objetivo general, es decir que sirve para hacer sitios web, aplicaciones móviles, sistemas de escritorio y para hacer ciencia de datos
- Hoy es el lenguaje más usado para Ciencia de datos y uno de los más usados en general,
- Enorme comunidad de usuarios,
- La mayoría de Universidades enseñan Python para carreras de computación o sistemas.



Popularidad Python

¿Por qué Python?

- Rico ecosistema de librerías, integraciones, frameworks, etc,
- Las integraciones de un modelo a producción suelen ser sencillo con Python,
- Hay distribuciones de Python enfocadas a Ciencia de Datos como Anaconda,
- Para programar en Python se pueden usar algunas IDEs (Interfaz de desarrollo) como Jupyter, PyCharm, Spyder, Rstudio, algunas de ellas integradas con Anaconda,
- Los Frameworks más usados para DeepLearning usan Python como base.



Algunas de las integraciones de Python

Google Colab

- Abrir https://colab.research.google.com y poner **"Nuevo cuaderno"**,
- Notar que Google Colab en realidad es una versión de Jupyter notebook,
- Notebook: Combinacion de código, gráficos y texto (notas/análisis/etc.),
- Equivalente a usar rmarkdown como notebook en R,
- Permite compartir los resultados.

Generalidades

Python, aparte de objetos, tiene:

- Expresión.- Se evalúa, se imprime y el valor se pierde (iPython)

```
5+5 # Expresión con output
```

```
## 10
```

```
5+5; # Expresión sin salida
```

```
## 10
```

- Asignación.- Evalúa la expresión y guarda el resultado en una variable (no lo imprime)

```
a = 5+5 # Asigna el valor a la variable "a"
```

Generalidades

- Comandos se separan por `;` o `enter`
- Para comentar se usa #
- Case sensitivity (`Abc` es diferente de `abc`)

```
a= 2; b= 1; a + b
```

```
## 3
```

Python como calculadora

```
2 + 3*5 # operaciones básicas
```

```
## 17
```

```
7 // 3 # division entera
```

```
## 2
```

```
7 % 3 # Modular
```

```
## 1
```

```
2 ** 3 # 2 elevado al cubo
```

```
## 8
```

```
pow(2, 3) # 2 elevado al cubo
```

```
## 8
```

Python como calculadora

A diferencia de R, para usar operaciones más "complicadas" debemos ya importar una libreria (import en Python es el equivalente de library en R).

Una biblioteca/librería es una colección de funciones y objetos que aumentan las capacidades del lenguaje, en este caso `math` permite cargar funciones enfocadas en cálculos matemáticos básicos.

Una biblioteca se instala (que es descargar los archivos ordenados a nuestro disco duro) y luego se activa (que es cargarla a RAM para poder usar sus funciones módulos), esto último se hace con `import`

```
import math
math.floor(2.3) # Funcion piso
```

```
## 2
```

```
math.fabs(-5) # valor absoluto
```

```
## 5.0
```

```
math.factorial(3) # Factorial
```

```
## 6
```

Python como calculadora

Otros ejemplos

```
math.exp(3) # e elevado a la x
```

```
## 20.085536923187668
```

```
math.sqrt(9) # raiz cuadrada
```

```
## 3.0
```

```
math.log(math.exp(2)) # Logaritmo natural
```

```
## 2.0
```

```
math.floor(2.3) # funcion piso
```

```
## 2
```

Caso a desarrollar: Retail online

Bases de Python para Data Science

Caso: Retail online

Publicado en la UCI Machine Learning Repository, en https://archive.ics.uci.edu/ml/datasets/online+retail.

Este es un conjunto de datos transnacionales que contiene las transacciones ocurridas entre el 01/12/2010 y el 09/12/2011 para una tienda minorista en línea registrada y con sede en el Reino Unido.

Nosotros trabajaremos con una muetra de estos datos, los mismos que están en el repositorio de github: https://github.com/see-ecuador/2020_11_PythonWeekend, no se preocupen, desde el mismo colab descargaremos la información.

1ro: Cargar paquetes a usar

Los paquetes más usados para ciencia de datos son:

```
import os
import math
import numpy as np
import pandas as pd
import scipy
import matplotlib.pyplot as plt
import seaborn as sns
```

2do: Directorios o Proyecto

Técnicamente, si trabajamos en local, deberíamos crear un proyecto o setear los directorios de trabajo, pero como estamos en google colab sólo procederemos a crear una carpeta y descargar el archivo de datos dentro de la misma.

```
import os
os.mkdir('Data') # Crear carpeta Data
```

```
# Obtener el archivo de excel
!wget -O Data/retail_online_sample.xlsx https://github.com/ \
see-ecuador/2020_11_PythonWeekend/blob/main/ \
01_Bases_Python_DataScience/Data/retail_online_sample.xlsx
```

3ro: Importar Datos

Panda permite importar desde una gran cantidad de fuentes, incluso conectarse a Bases de Datos.

- `read_csv` para importar desde csv
- `ExcelFile` & `xl.parse` o `read_excel` para importar desde excel
- `read_json` para importar desde json
- más en: https://pandas.pydata.org/docs/reference/io.html

```
transacciones = pd.read_excel("Data/retail_online_sample.xlsx",
sheet_name = 'transacciones')
transacciones.head(5)
```

```
##   InvoiceNo StockCode  Quantity  ... UnitPrice  CustomerID         Country
## 0    536378     21931        10  ...       195     14688.0  United Kingdom
## 1    536382     22379        10  ...        21     16098.0  United Kingdom
## 2    536394     22866        96  ...       185     13408.0  United Kingdom
## 3    536395     22866        48  ...        21     13767.0  United Kingdom
## 4    536398     22866        12  ...        21     13448.0  United Kingdom
##
## [5 rows x 7 columns]
```

3ro: Importar Datos

Debemos importar la otra hoja del archivo de excel suministrado.

```
productos = pd.read_excel("Data/retail_online_sample.xlsx",
sheet_name = 'productos')
productos.head(5)
```

```
##   StockCode                         Description
## 0     23064              CINDERELLA CHANDELIER
## 1     21002          ROSE DU SUD DRAWSTRING BAG
## 2     20831                    GOLD PHOTO FRAME
## 3     22964  3 PIECE SPACEBOY COOKIE CUTTER SET
## 4     22609              PENS ASSORTED SPACEBALL
```

¿Qué es Pandas?

Pandas es la gran navaja suiza de Python para Data Science

- Soporta lectura desde una variedad de datos, integrarlos y transformarlos
- Permite realizar estadística descriptiva
- Tiene opciones de gráficos integrados
- Soporta series de tiempo
- Métodos integrados para manejar valores perdidos
- Soporte para procesamiento de imágenes
- Internamente maneja dos tipos de objetos, pandas Series, panda DataFrame

pandas Series, pandas DataFrame ¿Qué es eso?

pandas Series, pandas DataFrame son **Estructuras de datos**, las cuales no son más que tipos de Objetos dentro de Python. Un entero es un objeto, un númeor también; además existe estructuras como:

- Listas
- Matrices
- Tuples
- ndarrays
- pandas Series
- pandas DataFrames
- Ver https://docs.python.org/3/tutorial/datastructures.html

Estructuras de datos | Objetos

Un string

```
a= 'Hola Mundo'
a
```

```
## 'Hola Mundo'
```

```
a.__class__
```

```
## <class 'str'>
```

Estructuras de datos | Objetos

Un entero

```
a= 5
a
```

```
## 5
```

```
a.__class__
```

```
## <class 'int'>
```

Estructuras de datos | Objetos

Una lista

```
a= [2, 4]
a
```

```
## [2, 4]
```

```
a.__class__
```

```
## <class 'list'>
```

Estructuras de datos | Objetos

Un pandasSeries

```
a = pd.Series([2, 4])
a
```

```
## 0    2
## 1    4
## dtype: int64
```

```
a.__class__
```

```
## <class 'pandas.core.series.Series'>
```

```
a.values
```

```
## array([2, 4], dtype=int64)
```

¿Qué importamos entonces?

```
transacciones.__class__
```

```
## <class 'pandas.core.frame.DataFrame'>
```

```
type(productos)
```

```
## <class 'pandas.core.frame.DataFrame'>
```

¿Qué importamos entonces?

Nótese que se preguntó lo mismo usando un método y una función, no se va a entrar mucho en esto pero podemos resumir que un método es parte de una clase y por tanto está asociado a un objeto; las funciones en cambio están definidas por si mismas y no pertenecen a ninguna clase.

```
transacciones.__class__
```

```
## <class 'pandas.core.frame.DataFrame'>
```

```
type(productos)
```

```
## <class 'pandas.core.frame.DataFrame'>
```

pandas.DataFrame

- DataFrame es un objeto que cumple:
  - Las columnas son vectores de tipo pandas Series
  - Cada columnas puede ser de un tipo de dato distinto
  - Cada elemento, columna es una variable
  - Las columnas tienen el mismo largo
- Se podría decir que un data.frame es como una tabla en una hoja de excel

4to: Entender los datos

Luego de importar se debe entender los datos

- ¿Qué representa cada columna?
- ¿Qué tipo de dato debería tener cada columna?
- ¿Qué granularidad o atomicidad tiene la data?
- Si es que se tiene varios conjuntos de datos ¿Cómo se relacionan los datos?
- A qué periodo de tiempo corresponde la data
- Muchas veces se obtiene la información desde una base de datos y por tanto toca entender la base y el query que genera los datos

Tipos de datos

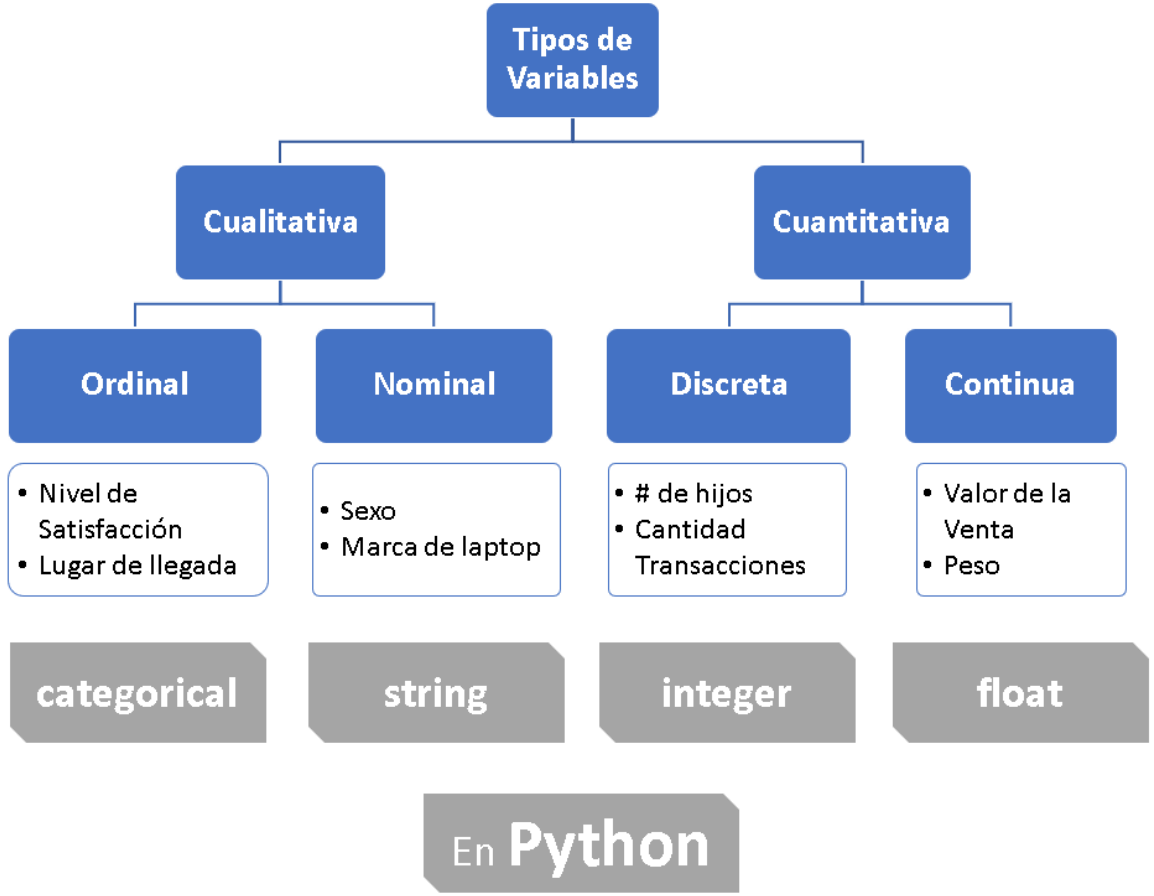
Nosotros como analistas debemos analizar que tipo de dato debe tener cada columna y luego eso reflejarlo en Python, para esto Python soporta una variedad de tipos de datos como entero (int64), numérico (float64), fecha y hora (datetime64), caracter (objet64), categórico (categorical), entre otros. Veamos lo que tenemos en nuestro dataframe

```
transacciones.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 15677 entries, 0 to 15676
## Data columns (total 7 columns):
## InvoiceNo      15677 non-null object
## StockCode      15677 non-null object
## Quantity       15677 non-null int64
## InvoiceDate    15677 non-null object
## UnitPrice      15677 non-null int64
## CustomerID     11300 non-null float64
## Country        15677 non-null object
## dtypes: float64(1), int64(2), object(4)
## memory usage: 857.5+ KB
```

Tipos de variables

Tipos de variables y su correspondencia en Python



Tipos de datos

Revisar que todas las columnas tengan el tipo correcto y además la relación que existe entre los dos dataframes importados.

```
transacciones.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 15677 entries, 0 to 15676
## Data columns (total 7 columns):
## InvoiceNo      15677 non-null object
## StockCode      15677 non-null object
## Quantity       15677 non-null int64
## InvoiceDate    15677 non-null object
## UnitPrice      15677 non-null int64
## CustomerID     11300 non-null float64
## Country        15677 non-null object
## dtypes: float64(1), int64(2), object(4)
## memory usage: 857.5+ KB
```

Tipos de datos

Revisar que todas las columnas tengan el tipo correcto y además la relación que existe entre los dos dataframes importados.

```
productos.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 100 entries, 0 to 99
## Data columns (total 2 columns):
## StockCode      100 non-null object
## Description    100 non-null object
## dtypes: object(2)
## memory usage: 1.7+ KB
```

5to. Corregir y ordenar los datos

Del punto anterior podemos ver que:

- La columna 'InvoiceDate' debe modificarse a tipo fecha,
- La columna 'CustomerID' debe modificarse a tipo string,
- La data de productos se puede agregar a la de transacciones por el código del producto.

Para poder realizar eso debemos aprender a manejar dataFrames, esto es: seleccionar columnas, filtrar filas, modificar columnas, unir dos conjuntos de datos. Vamos a dar un par de ejemplos de cómo hacerlo, sin embargo cabe indicar que hay muchas más opciones y que si quieren aprender más detalles (alerta de spam) les recomendamos los cursos de la Sociedad Ecuatoriana de Estadística.

Seleccionar columnas: [[,]]

Seleccionar una columna

```
# Seleccionar una columna
productos['Description']
```

```
## 0                  CINDERELLA CHANDELIER
## 1               ROSE DU SUD DRAWSTRING BAG
## 2                        GOLD PHOTO FRAME
## 3      3 PIECE SPACEBOY COOKIE CUTTER SET
## 4                  PENS ASSORTED SPACEBALL
##                       ...
## 95         FUSCHIA FLOWER PURSE WITH BEADS
## 96    ANTIQUE TALL SWIRLGLASS TRINKET POT
## 97          DROP EARRINGS W FLOWER & LEAF
## 98       HANGING CHICK  YELLOW DECORATION
## 99     WHITE/PINK CHICK EASTER DECORATION
## Name: Description, Length: 100, dtype: object
```

Seleccionar columnas: [[,]]

Seleccionar una columna

```
# Seleccionar una columna
productos.Description
```

```
## 0                  CINDERELLA CHANDELIER
## 1               ROSE DU SUD DRAWSTRING BAG
## 2                        GOLD PHOTO FRAME
## 3      3 PIECE SPACEBOY COOKIE CUTTER SET
## 4                  PENS ASSORTED SPACEBALL
##                       ...
## 95         FUSCHIA FLOWER PURSE WITH BEADS
## 96    ANTIQUE TALL SWIRLGLASS TRINKET POT
## 97          DROP EARRINGS W FLOWER & LEAF
## 98       HANGING CHICK  YELLOW DECORATION
## 99     WHITE/PINK CHICK EASTER DECORATION
## Name: Description, Length: 100, dtype: object
```

Seleccionar columnas: [[,]]

Seleccionar varias columnas

```
# Seleccionar varias columnas
transacciones[ ['StockCode', 'Quantity', 'Country'] ]
```

```
##        StockCode  Quantity         Country
## 0          21931        10  United Kingdom
## 1          22379        10  United Kingdom
## 2          22866        96  United Kingdom
## 3          22866        48  United Kingdom
## 4          22866        12  United Kingdom
## ...          ...       ...             ...
## 15672      22992         1  United Kingdom
## 15673      20979        -5  United Kingdom
## 15674     84997D         8         Germany
## 15675      22992        12         Germany
## 15676      21931        10  United Kingdom
##
## [15677 rows x 3 columns]
```

Filtrar filas por posición o condición

Para Filtrar filas según número de fila o según el cumplimiento de condiciones se usa `.iloc` y `.loc` respectivamente, ojo que para usar `.loc` debemos saber los operadores de relación y lógicos en Python.

Filtrar Filas por número

```
# Filtrar filas por numero de fila
transacciones.iloc[[5,6,7]]
# transacciones.loc[[5,6,7]] Tambien es válido
```

```
##   InvoiceNo StockCode  Quantity  ... UnitPrice  CustomerID         Country
## 5    536401     22767         2  ...       995     15862.0  United Kingdom
## 6    536401     85150         1  ...       255     15862.0  United Kingdom
## 7    536401     22068         2  ...       165     15862.0  United Kingdom
##
## [3 rows x 7 columns]
```

Filtrar Filas por número

Filtrar Filas por número usando un slice (sólo iloc)

```
# Filtrar Filas por número usando un slice (sólo iloc)
transacciones.iloc[ 0:4]
```

```
##   InvoiceNo StockCode  Quantity  ... UnitPrice  CustomerID         Country
## 0    536378     21931        10  ...       195     14688.0  United Kingdom
## 1    536382     22379        10  ...        21     16098.0  United Kingdom
## 2    536394     22866        96  ...       185     13408.0  United Kingdom
## 3    536395     22866        48  ...        21     13767.0  United Kingdom
##
## [4 rows x 7 columns]
```

Filtrar filas según una condición

Filtrar filas según una condición *loc*, filtrar transacciones realizadas en United Kingdom y que sean de más de 10 unidades

```
# Filtrar filas según una condición loc
# Filtrar transacciones cuyo tiempo de servicio sea mayor a 100
transacciones.loc[ (transacciones["Country"] == "United Kingdom") &
(transacciones["Quantity"] > 10) ]
```

```
##       InvoiceNo StockCode  Quantity  ... UnitPrice  CustomerID         Country
## 2        536394     22866        96  ...       185     13408.0  United Kingdom
## 3        536395     22866        48  ...        21     13767.0  United Kingdom
## 4        536398     22866        12  ...        21     13448.0  United Kingdom
## 8        536404     22773        12  ...       125     16218.0  United Kingdom
## 9        536404     22964        12  ...        21     16218.0  United Kingdom
## ...         ...       ...       ...  ...       ...         ...             ...
## 15627    581475     22380        20  ...        79     13069.0  United Kingdom
## 15631    581488     22179        24  ...       595     17428.0  United Kingdom
## 15657    581496     20831        12  ...        79     16558.0  United Kingdom
## 15665    581514    84031B        14  ...       125     17754.0  United Kingdom
## 15666    581514     22068        14  ...        39     17754.0  United Kingdom
##
## [3007 rows x 7 columns]
```

Filtrar filas y Seleccionar columnas

Filtrar filas según una condición *loc*, filtrar transacciones realizadas en United Kingdom y que sean de más de 10 unidades. Además que sólo queden las columnas de País y Unidades

```
# Filtrar filas según una condición loc
# Filtrar transacciones cuyo tiempo de servicio sea mayor a 100
transacciones.loc[ (transacciones["Country"] == "United Kingdom") &
(transacciones["Quantity"] > 10) ,  ['Country', 'Quantity']]
```

```
##                Country  Quantity
## 2       United Kingdom        96
## 3       United Kingdom        48
## 4       United Kingdom        12
## 8       United Kingdom        12
## 9       United Kingdom        12
## ...                ...       ...
## 15627   United Kingdom        20
## 15631   United Kingdom        24
## 15657   United Kingdom        12
## 15665   United Kingdom        14
## 15666   United Kingdom        14
##
## [3007 rows x 2 columns]
```

Ordenar los datos

Para ordenar los datos usamos `.sort_values()` así:
df.sort_values("columna")
df.sort_values("columna", ascenascending=False) <- descendente

```
transacciones.sort_values( "Quantity")
```

```
##        InvoiceNo StockCode  Quantity  ... UnitPrice  CustomerID         Country
## 5784      552733     23059     -2376  ...         0         NaN  United Kingdom
## 2295     C542693     15036      -600  ...        65     12908.0  United Kingdom
## 13845     576765     22219      -550  ...         0         NaN  United Kingdom
## 8553      561086     15036      -530  ...         0         NaN  United Kingdom
## 11209     569341     82600      -458  ...         0         NaN  United Kingdom
## ...          ...       ...       ...  ...       ...         ...             ...
## 7169      557092     15036       600  ...        72     12908.0  United Kingdom
## 11395     569815     22379       700  ...       185     15838.0  United Kingdom
## 8906      562285     15036       720  ...        72     17404.0          Sweden
## 7817      559047     15036      1200  ...        72     13082.0  United Kingdom
## 2478      543192     15036      1200  ...        65     17381.0  United Kingdom
##
## [15677 rows x 7 columns]
```

Crear o modificar columnas/variables

dataFrame['nueva_Var'] = ...

Crear una nueva columna con el tiempo en minutos, opc

```
# Crear una nueva columna con el tiempo en minutos
transacciones['Cantidad_10'] = transacciones['Quantity']*10
transacciones.head(5)
```

```
##   InvoiceNo StockCode  Quantity  ... CustomerID         Country  Cantidad_10
## 0    536378     21931        10  ...    14688.0  United Kingdom          100
## 1    536382     22379        10  ...    16098.0  United Kingdom          100
## 2    536394     22866        96  ...    13408.0  United Kingdom          960
## 3    536395     22866        48  ...    13767.0  United Kingdom          480
## 4    536398     22866        12  ...    13448.0  United Kingdom          120
##
## [5 rows x 8 columns]
```

Eliminar columnas

Para eliminar se usa `del` y `.drop`

```
# borrar una columna "in-place"
del transacciones['Cantidad_10']
```

Crear o modificar columnas/variables

**Con esto ya podemos modificar la columna 'InvoiceDate' y pasarla a tipo fecha

```
# Cambiar la columna data a fecha
transacciones["InvoiceDate"] = pd.to_datetime(transacciones["InvoiceDate"])
transacciones["CustomerID"] = transacciones["CustomerID"].astype(str)
transacciones.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 15677 entries, 0 to 15676
## Data columns (total 7 columns):
## InvoiceNo      15677 non-null object
## StockCode      15677 non-null object
## Quantity       15677 non-null int64
## InvoiceDate    15677 non-null datetime64[ns]
## UnitPrice      15677 non-null int64
## CustomerID     15677 non-null object
## Country        15677 non-null object
## dtypes: datetime64[ns](1), int64(2), object(4)
## memory usage: 857.5+ KB
```

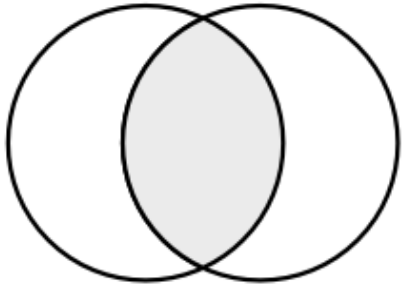
Nos queda unir los datos.

Unir datos - Merge|Join|Buscarv

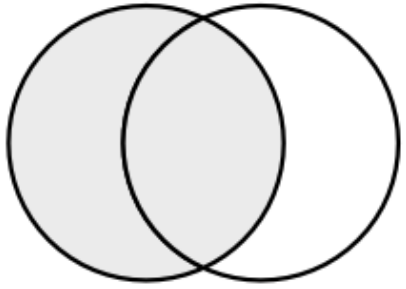
- Se tienen dos data.frames con columnas o variables que hacen las veces de “key” o “id” de los mismos
- Se desea agregar al primer conjunto el contenido del segundo conjunto de datos si y sólo si el “key” o “id” del segundo conjunto corresponde con el “key” o “id” del primer conjunto de datos.
- Parecido al Buscarv y Vlookup de excel
- Equivalente al Join de Bases de datos

Unir datos - Merge|Join|Buscarv

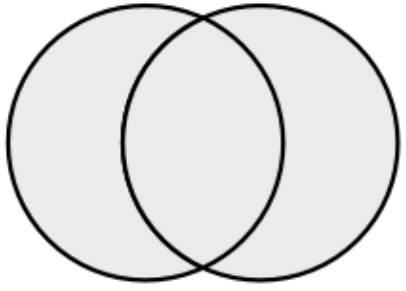
Entendiendo los tipos de Join



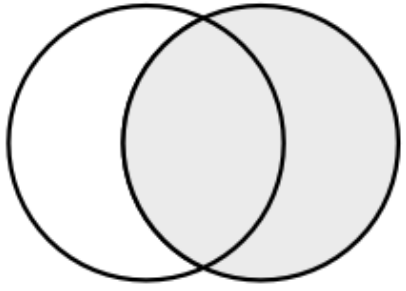
inner_join(x, y)



left_join(x, y)



full_join(x, y)



right_join(x, y)

Unir datos - Merge|Join|Buscarv

Para nuestro caso sólo necesitamos el left join que se realiza de esta manera:

```
# Left join
transacciones= transacciones.merge(productos,
how='left', left_on='StockCode', right_on= 'StockCode')
transacciones.head(5)
```

```
##   InvoiceNo StockCode  ...         Country                  Description
## 0    536378     21931  ...  United Kingdom         JUMBO STORAGE BAG SUKI
## 1    536382     22379  ...  United Kingdom       RECYCLING BAG RETROSPOT
## 2    536394     22866  ...  United Kingdom  HAND WARMER SCOTTY DOG DESIGN
## 3    536395     22866  ...  United Kingdom  HAND WARMER SCOTTY DOG DESIGN
## 4    536398     22866  ...  United Kingdom  HAND WARMER SCOTTY DOG DESIGN
##
## [5 rows x 8 columns]
```

6to: Explorar los datos

Con los datos corregidos, podemos empezar a explorar; para ello podemos seleccionar columnas o filtrar filas pero esto nos deja aún con muchas filas como para poder obtener información, de ahí la utilidad de la estadística descriptiva como herramienta para resumir los datos y empezar a obtener información inicial *"insights"*

Pandas - Descriptivas

- Comando describe() presenta varias estadísticas descriptivas, como summary en R.
- Existen funciones como min(), max(), mode(), median(), mean(), std(), corr(), count(), rank()
- Existen funciones como mean() que con el parámetro 1 permite obtiener media por filas
- Más información en https://pandas.pydata.org/pandas-docs/stable/reference/frame.html#computations-descriptive-stats

Descriptivas

Con .describe() se obtinen algunas estadísticas descriptivas

```
transacciones.describe()
```

```
##             Quantity     UnitPrice
## count  15677.000000  15677.000000
## mean       9.307521    316.796453
## std       40.151242    565.844724
## min    -2376.000000      0.000000
## 25%        1.000000     85.000000
## 50%        3.000000    195.000000
## 75%       10.000000    375.000000
## max     1200.000000   9996.000000
```

Medidas de Tendencia Central

Algunas medidas de tendencia central por separado

```
# Media de la cantidad
transacciones['Quantity'].mean()
```

```
## 9.307520571537921
```

```
# Media acotada de la cantidad
scipy.stats.trim_mean(transacciones.Quantity , 0.05)
```

```
## 5.603642548366523
```

Medidas de Posición

Calcular las medidas de Posición para el tiempo de servicio data de Banco

```
# Minimo y Máximo
transacciones.Quantity.min()
```

```
## -2376
```

```
transacciones.Quantity.max()

# Cuartiles
```

```
## 1200
```

```
transacciones.Quantity.quantile( [0.25, 0.50, 0.75] )
```

```
## 0.25     1.0
## 0.50     3.0
## 0.75    10.0
## Name: Quantity, dtype: float64
```

Cálculos con agrupamiento

Pandas permite obtener resúmenes o cálculos agrupando por los valores de una variable del dataFrame, como summarise + group_by en R o un Select, from, group by en SQL.

```
## Obtiene las descriptivas por Pais
transacciones.groupby('Country').describe()
```

```
##                      Quantity                                  ... UnitPrice
##                         count       mean          std  min  ...       50%     75%     max
## Country                                                      ...
## Australia                40.0  69.125000  106.699960  1.0  ...     165.0   249.0  1495.0
## Austria                  11.0  12.909091    8.312094  1.0  ...     195.0   332.0   415.0
## Bahrain                   1.0   8.000000         NaN  8.0  ...     145.0   145.0   145.0
## Belgium                  48.0   8.187500    4.823066  1.0  ...     231.5   315.0  1695.0
## Canada                    3.0   9.333333    4.618802  4.0  ...     195.0   310.0   425.0
## Channel Islands          20.0   9.200000    6.228965  1.0  ...     255.0   295.0   995.0
## Cyprus                   16.0   7.125000    6.130525  1.0  ...     201.5   387.5  3995.0
## Czech Republic            1.0  24.000000         NaN  24.0 ...     125.0   125.0   125.0
## Denmark                  14.0  17.428571   18.780982  1.0  ...     201.5   383.5   595.0
## EIRE                    163.0  16.668712   28.689279  1.0  ...     195.0   375.0  4995.0
## Finland                  27.0  27.000000   27.929857  1.0  ...     295.0   375.0  1275.0
## France                  231.0  13.709957   24.161555  1.0  ...     195.0   255.0  1695.0
## Germany                 221.0  14.316742   45.901566  1.0  ...     195.0   289.0  3995.0
```

Cálculos con agrupamiento

Group By se puede usar también para funciones simples, aquí por ejemplo obtenemos el promedio aritmético de la cantidad por Pais

```
## Obtiene las Media de la cantidad por Pais
transacciones[['Country', 'Quantity']].groupby('Country').mean()
```

```
##                       Quantity
## Country
## Australia            69.125000
## Austria              12.909091
## Bahrain               8.000000
## Belgium               8.187500
## Canada                9.333333
## Channel Islands       9.200000
## Cyprus                7.125000
## Czech Republic       24.000000
## Denmark              17.428571
## EIRE                 16.668712
## Finland              27.000000
## France               13.709957
## Germany              14.316742
## Greece               12.000000
```

Ejemplos más avanzados: Venta por País

Se quiere ver la evolución de la venta mensual de cada país, de tal forma que cada fila representa un país-año y en las columnas se tiene los meses de venta.

```
# Como vamos a necesitar columnas para año, mes, se crean
# Además una columna periodo
transacciones['anio']= transacciones.InvoiceDate.dt.year
transacciones['mes']= transacciones.InvoiceDate.dt.month
transacciones['periodo']= transacciones.InvoiceDate.dt.year * 100 + \
transacciones.InvoiceDate.dt.month
```

Ejemplos más avanzados: Venta por País

Ahora tenemos esto:

```
transacciones.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## Int64Index: 15677 entries, 0 to 15676
## Data columns (total 11 columns):
## InvoiceNo      15677 non-null object
## StockCode      15677 non-null object
## Quantity       15677 non-null int64
## InvoiceDate    15677 non-null datetime64[ns]
## UnitPrice      15677 non-null int64
## CustomerID     15677 non-null object
## Country        15677 non-null object
## Description    15677 non-null object
## anio           15677 non-null int64
## mes            15677 non-null int64
## periodo        15677 non-null int64
## dtypes: datetime64[ns](1), int64(5), object(5)
## memory usage: 1.4+ MB
```

Ejemplos más avanzados: Venta por País

Para lo que se requiere calcular, una opción para calcular la venta total es crear una columna con la venta en dólares y aplicar groupby + sum, así:

```
## Primero obtenemos la venta resumida por país, año y mes
transacciones['venta_usd']= transacciones.Quantity * transacciones.UnitPrice
transacciones.groupby(['Country', 'periodo'])['venta_usd'].sum().reset_index(name='venta_usd')
```

```
##           Country  periodo  venta_usd
## 0       Australia   201012       1020
## 1       Australia   201101      18605
## 2       Australia   201102      14267
## 3       Australia   201104        420
## 4       Australia   201105      66960
## ..            ...      ...        ...
## 219   Unspecified   201105       1500
## 220   Unspecified   201107       2782
## 221   Unspecified   201108      13872
## 222   Unspecified   201109         42
## 223   Unspecified   201111       1660
##
## [224 rows x 3 columns]
```

Ejemplos más avanzados: Venta por País

La opción anterior modifica el objeto transacciones, si no quisiéramos eso se podría realizar el cálculo directo usando '.apply'

```
transacciones.groupby(['Country', 'periodo']).apply(
    lambda x: (x.Quantity * x.UnitPrice).sum()
    ).reset_index(name='venta_usd')
```

```
##           Country  periodo  venta_usd
## 0       Australia   201012       1020
## 1       Australia   201101      18605
## 2       Australia   201102      14267
## 3       Australia   201104        420
## 4       Australia   201105      66960
## ..            ...      ...        ...
## 219   Unspecified   201105       1500
## 220   Unspecified   201107       2782
## 221   Unspecified   201108      13872
## 222   Unspecified   201109         42
## 223   Unspecified   201111       1660
##
## [224 rows x 3 columns]
```

Fin

Bases de Python para Ciencia de datos

Néstor Montaño