

Manipulacion de datos - Bases

Curso: Bases para Data Science - Estadística, R y Python

Katherine Morales / Néstor Montaña

Sociedad Ecuatoriana de Estadística

Octubre-2020



Nota:

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

Ejemplo: Data de transacciones bancarias

El Banco del Pacífico requiere mejorar los tiempos de atención al cliente en ventanilla, para ello ha recolectado esta información anónimamente para cada cajero y transacción realizada.

Le suministran un excel con dos hojas:

1. Tiene los datos de las transacciones, columnas: Sucursal, Cajero, ID_Transaccion, Transaccion, Tiempo_Servicio_seg, Nivel de satisfacción, Monto de la transaccion.
2. Otra hoja que indica si en la sucursal se ha puesto o no el nuevo sistema.

Ejemplo - Caso Banco: Importar

- Abrir Jupyter Notebook y crear un nuevo notebook dentro de la carpeta scripts dentro de la carpeta del proyecto creado anteriormente
- Importar paquete os
- Cambiar el directorio de trabajo al directorio del proyecto (recuerden que para importar requerimos que la carpeta Data se pueda acceder directamente)

```
import os
os.getcwd()
os.chdir('ruta/al/proyecto/')
```

Ejemplo - Caso Banco: Importar

Importar las hojas de excel que estamos utilizando

```
import pandas as pd
data_banco_xlsx = pd.read_excel('Data//Data_Banco.xlsx', sheet_name = 'Data')
data_banco_xlsx.head(3)
```

```
##      Sucursal  Cajero  ID_Transaccion  ... Tiempo_Servicio_seg  Satisfaccion  Monto
## 0           62    4820           2  ...           311.0      Muy Bueno  2889,3
## 1           62    4820           2  ...           156.0           Malo  1670,69
## 2           62    4820           2  ...           248.0      Regular  3172,49
##
## [3 rows x 7 columns]
```

```
data_sucursal = pd.read_excel('Data//Data_Banco.xlsx', sheet_name = 'Data_Sucursal')
data_sucursal.head(3)
```

```
##      ID_Sucursal  Sucursal  Nuevo_Sistema
## 0           62  Riocentro Sur           No
## 1           85      Centro           Si
## 2          267    Alborada           Si
```

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

Entender los datos - Ejemplo

¿Están bien nuestros tipos de datos?

...

```
# Ver la estructura del data.frame  
data_banco_xlsx.info()
```

```
## <class 'pandas.core.frame.DataFrame'>  
## RangeIndex: 24299 entries, 0 to 24298  
## Data columns (total 7 columns):  
## Sucursal                24299 non-null int64  
## Cajero                  24299 non-null int64  
## ID_Transaccion          24299 non-null int64  
## Transaccion             24299 non-null object  
## Tiempo_Servicio_seg     24299 non-null float64  
## Satisfaccion            24299 non-null object  
## Monto                   24299 non-null object  
## dtypes: float64(1), int64(3), object(3)  
## memory usage: 1.3+ MB
```


Entender los datos - Ejemplo

¿Están bien nuestros tipos de datos?

...

```
# Ver la estructura del data.frame  
data_sucursal.info()
```

```
## <class 'pandas.core.frame.DataFrame'>  
## RangeIndex: 5 entries, 0 to 4  
## Data columns (total 3 columns):  
## ID_Sucursal      5 non-null int64  
## Sucursal         5 non-null object  
## Nuevo_Sistema    5 non-null object  
## dtypes: int64(1), object(2)  
## memory usage: 248.0+ bytes
```

Entender los datos - Ejemplo

¿Está bien nuestros tipos de datos?

Si no lo están entonces debemos transformarlos, para esto aprenderemos sobre manipulación de datos.

Manipulacion de datos - Basico

Curso: Bases para Data Science - Estadística, R y Python

Manipulación de datos

Vamos a aprender a manipular Dataframes, esto se realiza con la librería *Pandas*, así que si no tenemos ya cargadas las librerías, directamente cargamos las librerías más usadas que ya vimos

```
import os
import math as mt
import numpy as np
import pandas as pd
import scipy
```

Seleccionar columnas: `[[,]]`

Seleccionar una columna

```
# Seleccionar una columna  
data_banco_xlsx['Monto']
```

```
## 0          2889,3  
## 1          1670,69  
## 2          3172,49  
## 3          1764.92  
## 4          1835.69  
##          ...  
## 24294         657.38  
## 24295         763.65  
## 24296        3326.79  
## 24297        1237.91  
## 24298        1643.14  
## Name: Monto, Length: 24299, dtype: object
```

Seleccionar columnas: `[[,]]`

Seleccionar una columna

```
# Seleccionar una columna  
data_banco_xlsx.Monto
```

```
## 0          2889,3  
## 1          1670,69  
## 2          3172,49  
## 3          1764.92  
## 4          1835.69  
##          ...  
## 24294         657.38  
## 24295         763.65  
## 24296        3326.79  
## 24297        1237.91  
## 24298        1643.14  
## Name: Monto, Length: 24299, dtype: object
```

Seleccionar columnas: `[[,]]`

Seleccionar varias columnas

```
# Seleccionar varias columnas  
data_banco_xlsx[ ['Tiempo_Servicio_seg', 'Sucursal'] ]
```

```
##           Tiempo_Servicio_seg  Sucursal  
## 0                311.0           62  
## 1                156.0           62  
## 2                248.0           62  
## 3                 99.0           62  
## 4                123.0           62  
## ...                ...           ...  
## 24294             184.0          586  
## 24295             124.0          586  
## 24296             141.0          586  
## 24297              54.0          586  
## 24298             105.0          586  
##  
## [24299 rows x 2 columns]
```

Seleccionar columnas: `[[,]]`

Seleccionar varias columnas

```
# Seleccionar varias columnas
lista_nombres = ['Tiempo_Servicio_seg', 'Sucursal']
data_banco_xlsx[ lista_nombres ]
```

```
##           Tiempo_Servicio_seg  Sucursal
## 0                311.0           62
## 1                156.0           62
## 2                248.0           62
## 3                 99.0           62
## 4                123.0           62
## ...                ...           ...
## 24294             184.0          586
## 24295             124.0          586
## 24296             141.0          586
## 24297              54.0          586
## 24298             105.0          586
##
## [24299 rows x 2 columns]
```


Seleccionar columnas: `[[,]]`

Para seleccionar un columna por número usando *iloc*

```
# Seleccionar un columna por número  
data_banco_xlsx.iloc[ : , 1]
```

```
## 0          4820  
## 1          4820  
## 2          4820  
## 3          4820  
## 4          4820  
## ...  
## 24294       4424  
## 24295       4424  
## 24296       4424  
## 24297       4424  
## 24298       4424  
## Name: Cajero, Length: 24299, dtype: int64
```

Seleccionar columnas: `[[,]]`

Para seleccionar dos columnas por número usando *iloc*

```
# Seleccionar varias columnas  
data_banco_xlsx.iloc[ :, [1 , 2]]
```

```
##          Cajero  ID_Transaccion  
## 0          4820             2  
## 1          4820             2  
## 2          4820             2  
## 3          4820             2  
## 4          4820             2  
## ...          ...             ...  
## 24294        4424             10  
## 24295        4424             10  
## 24296        4424             10  
## 24297        4424             10  
## 24298        4424             10  
##  
## [24299 rows x 2 columns]
```

Seleccionar columnas: `[[,]]`

Para seleccionar dos columnas por número usando *iloc*

```
# Seleccionar varias columnas
data_banco_xlsx.iloc[ :, 0:4]
```

```
##          Sucursal  Cajero  ID_Transaccion  Transaccion
## 0           62    4820           2    Cobro/Pago (Cta externa)
## 1           62    4820           2    Cobro/Pago (Cta externa)
## 2           62    4820           2    Cobro/Pago (Cta externa)
## 3           62    4820           2    Cobro/Pago (Cta externa)
## 4           62    4820           2    Cobro/Pago (Cta externa)
## ...         ...      ...         ...         ...
## 24294        586    4424          10    Cobrar cheque (Cta del Bco)
## 24295        586    4424          10    Cobrar cheque (Cta del Bco)
## 24296        586    4424          10    Cobrar cheque (Cta del Bco)
## 24297        586    4424          10    Cobrar cheque (Cta del Bco)
## 24298        586    4424          10    Cobrar cheque (Cta del Bco)
##
## [24299 rows x 4 columns]
```

Seleccionar columnas: `[:,]`

Seleccionar varias columnas usando un slice

```
# Seleccionar varias columnas
data_banco_xlsx.loc[:, 'Cajero':'Transaccion']
```

```
##           Cajero  ID_Transaccion           Transaccion
##  0             4820             2      Cobro/Pago (Cta externa)
##  1             4820             2      Cobro/Pago (Cta externa)
##  2             4820             2      Cobro/Pago (Cta externa)
##  3             4820             2      Cobro/Pago (Cta externa)
##  4             4820             2      Cobro/Pago (Cta externa)
##  ...           ...           ...           ...
## 24294          4424            10  Cobrar cheque (Cta del Bco)
## 24295          4424            10  Cobrar cheque (Cta del Bco)
## 24296          4424            10  Cobrar cheque (Cta del Bco)
## 24297          4424            10  Cobrar cheque (Cta del Bco)
## 24298          4424            10  Cobrar cheque (Cta del Bco)
##
## [24299 rows x 3 columns]
```

Filtrar Filas usando un slice

Filtrar filas usando un slice

```
# Filtrar filas usando un slice  
data_banco_xlsx[2:7]
```

```
##      Sucursal  Cajero  ID_Transaccion  ...  Tiempo_Servicio_seg  Satisfaccion  Monto  
## 2          62    4820          2  ...          248.0        Regular  3172,49  
## 3          62    4820          2  ...          99.0        Regular  1764.92  
## 4          62    4820          2  ...         123.0      Muy Bueno  1835.69  
## 5          62    4820          2  ...         172.0        Bueno  2165.42  
## 6          62    4820          2  ...         140.0        Regular  1304.9  
##  
## [5 rows x 7 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.

Operadores de relación en Python

```
3 == 3 # Igualdad
```

```
## True
```

```
3 == 3.0 # Igualdad
```

```
## True
```

```
3 >= 1 # Mayor igual
```

```
## True
```

```
3 < 2 # Menor
```

```
## False
```

```
3 != 7 # Diferente
```

```
## True
```

Operadores de relación en Python

Para aprender a filtrar por condiciones, debemos aprender a "preguntar"

```
lista= ['a', 'b', 'c', 'd']  
'c' in lista
```

```
## True
```

```
'z' in lista
```

```
## False
```

```
'c' not in lista
```

```
## False
```

```
'z' not in lista
```

```
## True
```


Operadores de relación en Python

Para aprender a filtrar por condiciones, debemos aprender a "preguntar"

```
a = 'x'  
a in lista
```

```
## False
```

```
a = 'c'  
a in lista
```

```
## True
```

```
type(a) is str
```

```
## True
```

```
type(a) is not str
```

```
## False
```

Operadores de relación en Python

Para aprender a filtrar por condiciones, debemos aprender a "preguntar"

```
True & True
```

```
## True
```

```
True & False
```

```
## False
```

```
False | False
```

```
## False
```

```
False | True
```

```
## True
```

Filtrar Filas por número

Filtrar Filas por número

```
# Filtrar filas por numero de fila  
data_banco_xlsx.iloc[[5,6,7]]
```

```
##      Sucursal  Cajero  ID_Transaccion  ...  Tiempo_Servicio_seg  Satisfaccion  Monto  
## 5          62    4820          2  ...          172.0          Bueno  2165.42  
## 6          62    4820          2  ...          140.0          Regular  1304.9  
## 7          62    4820          2  ...          247.0          Bueno  4080.05  
##  
## [3 rows x 7 columns]
```

```
data_banco_xlsx.loc[[5,6,7]]
```

```
##      Sucursal  Cajero  ID_Transaccion  ...  Tiempo_Servicio_seg  Satisfaccion  Monto  
## 5          62    4820          2  ...          172.0          Bueno  2165.42  
## 6          62    4820          2  ...          140.0          Regular  1304.9  
## 7          62    4820          2  ...          247.0          Bueno  4080.05  
##  
## [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)
data_banco_xlsx.iloc[ 0:4]
```

```
##      Sucursal  Cajero  ID_Transaccion  ...  Tiempo_Servicio_seg  Satisfaccion  Monto
## 0           62    4820           2  ...           311.0      Muy Bueno  2889,3
## 1           62    4820           2  ...           156.0           Malo  1670,69
## 2           62    4820           2  ...           248.0      Regular  3172,49
## 3           62    4820           2  ...            99.0      Regular  1764.92
##
## [4 rows x 7 columns]
```

Filtrar filas según una condición

Filtrar filas según una condición *loc*, filtrar transacciones cuyo tiempo de servicio sea mayor a 100

```
# Filtrar filas según una condición loc
# Filtrar transacciones cuyo tiempo de servicio sea mayor a 100
data_banco_xlsx.loc[data_banco_xlsx["Tiempo_Servicio_seg"] > 100]
```

```
##          Sucursal  Cajero  ...  Satisfaccion  Monto
## 0             62    4820  ...      Muy Bueno  2889,3
## 1             62    4820  ...           Malo  1670,69
## 2             62    4820  ...      Regular  3172,49
## 4             62    4820  ...      Muy Bueno  1835.69
## 5             62    4820  ...          Bueno  2165.42
## ...          ...      ...  ...          ...      ...
## 24292         586    4424  ...           Malo   2582.1
## 24294         586    4424  ...      Muy Malo   657.38
## 24295         586    4424  ...          Bueno   763.65
## 24296         586    4424  ...          Bueno  3326.79
## 24298         586    4424  ...           Malo  1643.14
##
## [14809 rows x 7 columns]
```

Filtrar filas según una condición

Filtrar filas según una condición *loc*, filtrar transacciones cuyo tiempo de servicio sea mayor a 100 y que se hayan realizado en la sucursal 62

```
# Filtrar filas según una condición loc
data_banco_xlsx.loc[ (data_banco_xlsx["Tiempo_Servicio_seg"] > 100) &
                     (data_banco_xlsx["Sucursal"] == 62) ]
```

```
##          Sucursal  Cajero  ...  Satisfaccion  Monto
## 0             62    4820  ...      Muy Bueno  2889,3
## 1             62    4820  ...           Malo  1670,69
## 2             62    4820  ...      Regular  3172,49
## 4             62    4820  ...      Muy Bueno  1835.69
## 5             62    4820  ...           Bueno  2165.42
## ...          ...      ...  ...          ...      ...
## 2831          62    5286  ...           Malo  3265.79
## 2832          62    5286  ...           Bueno  1144.88
## 2833          62    5286  ...      Regular  1779.14
## 2834          62    5286  ...           Malo    847.3
## 2837          62    5286  ...      Regular  1231.13
##
## [886 rows x 7 columns]
```

Filtrar filas y Seleccionar columnas

filtrar por numero de fila y Seleccionar según número de Columna

```
# Filtrar por numero de fila y Seleccionar según número de Columna  
data_banco_xlsx.iloc[[5,6,7], [4, 1]]
```

##	Tiempo_Servicio_seg	Cajero
## 5	172.0	4820
## 6	140.0	4820
## 7	247.0	4820

Filtrar filas y Seleccionar columnas

Filtrar por numero de fila y Seleccionar según nombre de Columna

```
# Filtrar por numero de fila y Seleccionar según nombre de Columna  
data_banco_xlsx.loc[[5,6,7], ['Tiempo_Servicio_seg', 'Sucursal']]
```

##	Tiempo_Servicio_seg	Sucursal
## 5	172.0	62
## 6	140.0	62
## 7	247.0	62

Filtrar filas y Seleccionar columnas

Filtrar por numero de fila y Seleccionar según nombre de Columna

```
# Filtrar filas según una condición, Seleccionar según número de Columna  
data_banco_xlsx.loc[ (data_banco_xlsx["Tiempo_Servicio_seg"] > 100) &  
    (data_banco_xlsx["Sucursal"] == 62), ['Tiempo_Servicio_seg', 'Sucursal']]
```

```
##      Tiempo_Servicio_seg  Sucursal  
## 0          311.0          62  
## 1          156.0          62  
## 2          248.0          62  
## 4          123.0          62  
## 5          172.0          62  
## ...          ...          ...  
## 2831         220.0          62  
## 2832         142.0          62  
## 2833         113.0          62  
## 2834         134.0          62  
## 2837         156.0          62  
##  
## [886 rows x 2 columns]
```

Filtrar usando .apply

Se puede utilizar las bondades de .apply para hacer filtros complicados.
Supongamos que deseamos filtrar las filas que tengan más de 2 palabras en la columna Transaccion

```
# Filtrar las filas que tengan más de 2 palabras en la Transaccion (primer intento fallido)
# Para saber que tenemos más de dos palabras, vamos a cortar la frase usando los espacios
# con el metodo split(" ") y luego contaremos las palabras que queden.
new_df = data_banco_xlsx[len(data_banco_xlsx['Transaccion'].split(" "))>2] ## ERROR
# Incluso sólo la parte del split() también da error
data_banco_xlsx['Transaccion'].split(" ") ## ERROR
```

¡Pero en el string independiente sí se puede! ¿Por qué? Pues porque split es un método de los string, no de las listas ni Series

```
a= "Esto es un string"
a.split(" ") # Probar split en un string
```

```
## ['Esto', 'es', 'un', 'string']
```

```
len( a.split(" ") ) # Contar cantidad de palabras
```

Filtrar usando .apply

Se puede utilizar las bondades de `.apply` para hacer filtros complicados.

Supongamos que deseamos filtrar las filas que tengan más de 2 palabras en la columna Transaccion; para saber que tenemos más de dos palabras, vamos a cortar la frase usando los espacios con el metodo `split(" ")` y luego contaremos las palabras que queden, pero ya vimos que `split()` es un método de los string, así que **debo aplicarlo a cada elemento de mi columna, para eso se usa `.apply`**

```
# Con .apply se genera un vector booleano (true - false)
data_banco_xlsx.apply( lambda x: len(x['Transaccion'].split(" "))>2, axis= 1 )#Vector Booleano
```

```
## 0          True
## 1          True
## 2          True
## 3          True
## 4          True
##          ...
## 24294      True
## 24295      True
## 24296      True
## 24297      True
## 24298      True
## Length: 24299, dtype: bool
```

Filtrar usando .apply

Se puede utilizar las bondades de `.apply` para hacer filtros complicados.

Supongamos que deseamos filtrar las filas que tengan más de 2 palabras en la columna Transaccion; para saber que tenemos más de dos palabras, vamos a cortar la frase usando los espacios con el metodo `split(" ")` y luego contaremos las palabras que queden, pero ya vimos que `split()` es un método de los string, así que **debo aplicarlo a cada elemento de mi columna, para eso se usa `.apply`**

```
# Con .apply se genera un vector booleano (true - false)
# Y dicho vector se usa para filtrar
data_banco_xlsx.loc[ data_banco_xlsx.apply(
    lambda x: len(x['Transaccion'].split(" "))>2, axis= 1 ) ]
```

##	Sucursal	Cajero	...	Satisfaccion	Monto
## 0	62	4820	...	Muy Bueno	2889,3
## 1	62	4820	...	Malo	1670,69
## 2	62	4820	...	Regular	3172,49
## 3	62	4820	...	Regular	1764.92
## 4	62	4820	...	Muy Bueno	1835.69
##
## 24294	586	4424	...	Muy Malo	657.38
## 24295	586	4424	...	Bueno	763.65
## 24296	586	4424	...	Bueno	3326.79
## 24297	586	4424	...	Muy Bueno	1237.91

Filtrar usando .apply

Se puede utilizar las bondades de .apply para hacer filtros complicados.

Supongamos que deseamos filtrar las filas que tengan más de 2 palabras en la columna Transaccion; para saber que tenemos más de dos palabras, vamos a cortar la frase usando los espacios con el metodo split(" ") y luego contaremos las palabras que queden, pero ya vimos que split() es un método de los string, así que **debo aplicarlo a cada elemento de mi columna, para eso se usa .apply**

```
# Para verificar el resultado del filtro, podría obtener los valores unicos  
# de las transacciones que quedaaron luego del filtro  
data_banco_xlsx.loc[ data_banco_xlsx.apply(  
    lambda x: len(x['Transaccion'].split(" "))>2,  
    axis= 1 )].Transaccion.unique()
```

```
## array(['Cobro/Pago (Cta externa)', 'Cobrar cheque (Cta del Bco)'],  
##      dtype=object)
```

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
data_banco_xlsx['Tiempo_Servicio_Min'] = data_banco_xlsx['Tiempo_Servicio_seg']/60
data_banco_xlsx.head(5)
```

```
##      Sucursal  Cajero  ID_Transaccion  ... Satisfaccion    Monto  Tiempo_Servicio_Min
## 0           62    4820             2  ...      Muy Bueno  2889,3         5.183333
## 1           62    4820             2  ...           Malo  1670,69         2.600000
## 2           62    4820             2  ...      Regular  3172,49         4.133333
## 3           62    4820             2  ...      Regular  1764.92         1.650000
## 4           62    4820             2  ...      Muy Bueno  1835.69         2.050000
##
## [5 rows x 8 columns]
```

Crear o modificar columnas/variables

nuevo_df = dataframe.assign(nueva_Var= lambda x: ...), a un nuevo DF

dataframe = dataframe.assign(nueva_Var= lambda x: ...), a mismo DF

Crear una nueva columna con el tiempo en minutos, opc

```
# Crear una nueva columna con el tiempo en minutos
data_banco_xlsx = data_banco_xlsx.assign(
    Tiempo_Servicio_Min2= lambda x: x.Tiempo_Servicio_seg/60)
data_banco_xlsx.head(5)
```

```
##      Sucursal  Cajero  ...  Tiempo_Servicio_Min  Tiempo_Servicio_Min2
## 0           62    4820  ...           5.183333           5.183333
## 1           62    4820  ...           2.600000           2.600000
## 2           62    4820  ...           4.133333           4.133333
## 3           62    4820  ...           1.650000           1.650000
## 4           62    4820  ...           2.050000           2.050000
##
## [5 rows x 9 columns]
```

Modificar o crear columnas

Crear una nueva columna y sólo quedarse con dicha columna.

```
# Crear una columna pero sólo mantener dicha columna  
data_banco_xlsx.apply(lambda x: x.Tiempo_Servicio_seg/60, axis= 1)
```

```
## 0          5.183333  
## 1          2.600000  
## 2          4.133333  
## 3          1.650000  
## 4          2.050000  
##          ...  
## 24294       3.066667  
## 24295       2.066667  
## 24296       2.350000  
## 24297       0.900000  
## 24298       1.750000  
## Length: 24299, dtype: float64
```


Modificar o crear columnas

Crear una nueva columna usando `.apply` y asignarla a una nueva columna

```
# Crear una nueva columna usando .apply y asignarla a una nueva columna
data_banco_xlsx['Tiempo_Servicio_Min3'] = data_banco_xlsx.apply(
    lambda x: x.Tiempo_Servicio_seg/60, axis= 1)
data_banco_xlsx.head(5)
```

```
##      Sucursal  Cajero  ...  Tiempo_Servicio_Min2  Tiempo_Servicio_Min3
## 0          62    4820  ...                5.183333                5.183333
## 1          62    4820  ...                2.600000                2.600000
## 2          62    4820  ...                4.133333                4.133333
## 3          62    4820  ...                1.650000                1.650000
## 4          62    4820  ...                2.050000                2.050000
##
## [5 rows x 10 columns]
```

Eliminar columnas

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

```
# borrar una columna "in-place"
del data_banco_xlsx['Tiempo_Servicio_Min3']
# borrar varias columnas (se debe asignar)
data_banco_xlsx.drop(columns= ['Tiempo_Servicio_Min', 'Tiempo_Servicio_Min2'])
```

```
##          Sucursal  Cajero  ...  Satisfaccion  Monto
## 0             62    4820  ...      Muy Bueno  2889,3
## 1             62    4820  ...           Malo  1670,69
## 2             62    4820  ...      Regular  3172,49
## 3             62    4820  ...      Regular  1764.92
## 4             62    4820  ...      Muy Bueno  1835.69
## ...          ...      ...  ...          ...      ...
## 24294         586    4424  ...      Muy Malo   657.38
## 24295         586    4424  ...          Bueno   763.65
## 24296         586    4424  ...          Bueno  3326.79
## 24297         586    4424  ...      Muy Bueno  1237.91
## 24298         586    4424  ...           Malo  1643.14
##
## [24299 rows x 7 columns]
```

Ordenar los datos

Para ordenar los datos usamos `.sort_values()` así:

```
df.sort_values( "columna")
```

```
df.sort_values( "columna", ascenascending=False) <- descendente
```

```
data_banco_xlsx.sort_values( "Tiempo_Servicio_seg")
```

```
##          Sucursal  Cajero  ...  Tiempo_Servicio_Min  Tiempo_Servicio_Min2
## 10425          85    3983  ...           0.302196           0.302196
##  7162          85     472  ...           0.302490           0.302490
## 11871          85    3983  ...           0.316781           0.316781
## 12021          85    3983  ...           0.332725           0.332725
##  1211          62   5211  ...           0.333333           0.333333
## ...          ...     ...  ...           ...           ...
## 21032         443   4208  ...          20.220172          20.220172
## 10368          85    3983  ...          20.800597          20.800597
##  5735          85     472  ...          21.095559          21.095559
##  8325          85    3678  ...          22.276138          22.276138
## 10330          85    3983  ...          26.711639          26.711639
##
## [24299 rows x 9 columns]
```

Ordenar los datos

Para ordenar los datos usamos `.sort_values()` así:

`df.sort_values("columna")`

`df.sort_values("columna", ascenascending=False) <- descendente`

```
data_banco_xlsx.sort_values( ["Transaccion", "Tiempo_Servicio_seg"], ascending= [True, False])
```

```
##          Sucursal  Cajero  ...  Tiempo_Servicio_Min  Tiempo_Servicio_Min2
## 16916         267    2556  ...           15.218604           15.218604
## 20634         443    3732  ...           13.814274           13.814274
## 20644         443    3732  ...           13.492129           13.492129
## 17960         267    4796  ...           13.226732           13.226732
##  9924          85    3678  ...           12.964907           12.964907
## ...          ...      ...  ...              ...              ...
## 24245         586    4424  ...            0.333333            0.333333
## 12021          85    3983  ...            0.332725            0.332725
## 11871          85    3983  ...            0.316781            0.316781
##  7162          85     472  ...            0.302490            0.302490
## 10425          85    3983  ...            0.302196            0.302196
##
## [24299 rows x 9 columns]
```

Entender los datos - Ejemplo

¿Está bien nuestros tipos de datos?
Si no lo están entonces debemos transformarlos.

```
# Ver la estructura del data.frame  
data_banco_xlsx.info()
```

```
## <class 'pandas.core.frame.DataFrame'>  
## RangeIndex: 24299 entries, 0 to 24298  
## Data columns (total 9 columns):  
## Sucursal                24299 non-null int64  
## Cajero                  24299 non-null int64  
## ID_Transaccion          24299 non-null int64  
## Transaccion             24299 non-null object  
## Tiempo_Servicio_seg     24299 non-null float64  
## Satisfaccion            24299 non-null object  
## Monto                   24299 non-null object  
## Tiempo_Servicio_Min     24299 non-null float64  
## Tiempo_Servicio_Min2    24299 non-null float64  
## dtypes: float64(3), int64(3), object(3)  
## memory usage: 1.7+ MB
```

Ejemplo - Manipulación de datos

Lo primero que necesitamos es corregir los tipos de datos, nótese que

- **Monto** tiene una mezcla de "," y "."
- **Sucursal** y **Cajero** deberían ser de tipo character
- **Satisfaccion** debe ser factor ordenado

```
data_banco_xlsx['Monto'].head(4)
```

```
## 0      2889,3
## 1      1670,69
## 2      3172,49
## 3      1764.92
## Name: Monto, dtype: object
```

```
# Modificar la coma por punto en Monto luego transformar a numérico
data_banco_xlsx['Monto'] = data_banco_xlsx['Monto'].replace(',', '.', regex=True)
data_banco_xlsx["Monto"] = pd.to_numeric(data_banco_xlsx.Monto, errors='coerce')
data_banco_xlsx['Monto'].head(4)
```

```
## 0      2889.30
## 1      1670.69
## 2      3172.49
## 3      1764.92
## Name: Monto, dtype: float64
```

Ejemplo - Manipulación de datos

Lo primero que necesitamos es corregir los tipos de datos, nótese que

- **Monto** tiene una mezcla de "," y "."
- **Sucursal** y **Cajero** deberían ser de tipo character
- **Satisfaccion** debe ser factor ordenado

```
# Modificar a String  
data_banco_xlsx['Sucursal'] = data_banco_xlsx['Sucursal'].astype(str)  
data_banco_xlsx['Cajero'] = data_banco_xlsx['Cajero'].astype(str)  
data_banco_xlsx['ID_Transaccion'] = data_banco_xlsx['ID_Transaccion'].astype(str)
```

Ejemplo - Manipulación de datos

Lo primero que necesitamos es corregir los tipos de datos, nótese que

- **Monto** tiene una mezcla de "," y "."
- **Sucursal** y **Cajero** deberían ser de tipo character
- **Satisfaccion** debe ser factor ordenado

```
## Dato Categorical
data_banco_xlsx['Satisfaccion'] = pd.Categorical(
    data_banco_xlsx['Satisfaccion'],
    categories= ['Muy Malo', 'Malo', 'Regular', 'Bueno', 'Muy Bueno'],
    ordered=True)
data_banco_xlsx.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 24299 entries, 0 to 24298
## Data columns (total 9 columns):
## Sucursal                24299 non-null object
## Cajero                  24299 non-null object
## ID_Transaccion          24299 non-null object
## Transaccion             24299 non-null object
## Tiempo_Servicio_seg     24299 non-null float64
## Satisfaccion            24299 non-null category
```


Ejemplo - Manipulación de datos

Lo primero que necesitamos es corregir los tipos de datos, nótese que

- **Monto** tiene una mezcla de "," y "."
- **Sucursal** y **Cajero** deberían ser de tipo character
- **Satisfaccion** debe ser factor ordenado

```
## Dato Categorical -- Otra opción
data_banco_xlsx2 = pd.read_excel('Data//Data_Banco.xlsx', sheet_name = 'Data')
## Objeto que controla los niveles y ordered de la categoria
cat_Satisf = pd.CategoricalDtype(
    categories= ['Muy Malo', 'Malo', 'Regular', 'Bueno', 'Muy Bueno'],
    ordered= True)
## Se hace que Satisfaccion se comporte segun el objeto anterior
data_banco_xlsx2.Satisfaccion = data_banco_xlsx2.Satisfaccion.astype(cat_Satisf)
data_banco_xlsx2.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 24299 entries, 0 to 24298
## Data columns (total 7 columns):
## Sucursal          24299 non-null int64
## Cajero            24299 non-null int64
## ID_Transaccion    24299 non-null int64
```

Fin

Curso: Bases para Data Science - Estadística, R y Python

Katherine Morales / Néstor Montaña