Unir datos y Pivot de tablas

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

Katherine Morales / Néstor Montaño

Sociedad Ecuatoriana de Estadística

Octubre-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)

Caso a desarrollar

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

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: Preeliminares

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

```
import os
import math as mt
import numpy as np
import pandas as pd
import scipy.stats
# os.getcwd() # obtener el directorio de trabajo
# os.chdir('ruta/al/proyecto/') # Definir el directorio de trabajo
```

Ejemplo - Caso Banco: Importar

Importar las hojas de excel que estamos utilizando

```
data_banco_xlsx = pd.read_excel('Data//Data_Banco.xlsx', sheet_name = 'Data')
data_banco_xlsx.head(5)
```

```
##    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        62    4820               2  ...               123.0     Muy Bueno  1835.69
##
## [5 rows x 7 columns]
```

Ejemplo - Caso Banco: Importar

Importar las hojas de excel que estamos utilizando

```
data_sucursal = pd.read_excel('Data//Data_Banco.xlsx', sheet_name = 'Data_Sucursal')
data_sucursal.head(5)
```

```
##    ID_Sucursal       Sucursal Nuevo_Sistema
## 0           62  Riocentro Sur            No
## 1           85         Centro            Si
## 2          267       Alborada            Si
## 3          443   Mall del Sol            Si
## 4          586      Via Daule            No
```

Entender los datos - Ejemplo

Entender los datos y modificar los que tienen mal el tipo de columna.

```
# 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

Entender los datos y modificar los que tienen mal el tipo de columna.

```
# 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
```

Ejemplo - Manipulacion 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
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 - Manipulacion 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 - Manipulacion 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 7 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
## Monto                  24299 non-null float64
## dtypes: category(1), float64(2), object(4)
## memory usage: 1.1+ MB
```

Ejemplo - Manipulacion de datos

Corregir tipo de dato en data_sucursal

```
# Modificar a String
data_sucursal['ID_Sucursal'] = data_sucursal['ID_Sucursal'].astype(str)
```

Unir Datos

Introducción al análisis de datos

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

Katherine Morales / Néstor Montaño

Unir datos

```
df_1 = pd.DataFrame({
  'Nombre': ['Ana', 'Berni', 'Carlos', 'Daniel', 'Ericka'],
  'Edad' : [20,19,20,19,18],
  'Ciudad': ['Gye', 'Uio', 'Cue', 'Gye', 'Cue']
    })
df_1
```

```
##     Nombre  Edad Ciudad
## 0      Ana    20    Gye
## 1    Berni    19    Uio
## 2  Carlos    20    Cue
## 3  Daniel    19    Gye
## 4  Ericka    18    Cue
```

Unir datos

```
df_2 = pd.DataFrame({
  'Nombre': ['Fulton', 'Gilda'],
  'Ciudad': ['Mach', 'Gye'] ,
  'Edad' : [21,18]
    })
df_2
```

```
##    Nombre Ciudad  Edad
## 0  Fulton   Mach    21
## 1   Gilda    Gye    18
```

Unir datos

```
pd.concat([df_1, df_2]) #notese el indice
```

```
##   Ciudad  Edad  Nombre
## 0    Gye    20     Ana
## 1    Uio    19   Berni
## 2    Cue    20  Carlos
## 3    Gye    19  Daniel
## 4    Cue    18  Ericka
## 0   Mach    21  Fulton
## 1    Gye    18   Gilda
##
## C:/ProgramData/Anaconda3/python.exe:1: FutureWarning: Sorting because non-concatenation axis is not
## of pandas will change to not sort by default.
##
## To accept the future behavior, pass 'sort=False'.
##
## To retain the current behavior and silence the warning, pass 'sort=True'.
```

Unir datos

```
pd.concat([df_1, df_2], ignore_index=True)
```

```
##   Ciudad  Edad  Nombre
## 0    Gye    20     Ana
## 1    Uio    19   Berni
## 2    Cue    20  Carlos
## 3    Gye    19  Daniel
## 4    Cue    18  Ericka
## 5   Mach    21  Fulton
## 6    Gye    18   Gilda
```

Unir datos

Aumentar columna que indica origen de la fila

```
pd.concat([df_1, df_2] , keys=['df1', 'df2'])
```

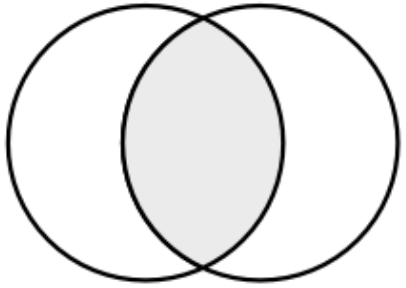
```
##        Ciudad  Edad  Nombre
## df1 0     Gye    20     Ana
##     1     Uio    19   Berni
##     2     Cue    20  Carlos
##     3     Gye    19  Daniel
##     4     Cue    18  Ericka
## df2 0    Mach    21  Fulton
##     1     Gye    18   Gilda
```

Unir datos - Merge|Join|Buscarv

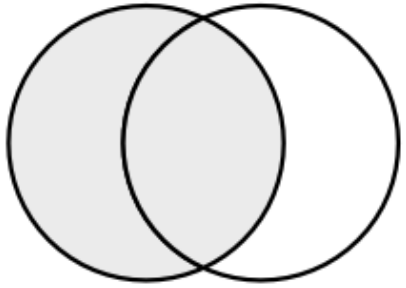
- Se tienen dos data.frames con columnas o variables que hacen las veces de “key” o “id” de los mismos
- Se desea agregar al primer conjunto el contenido del segundo conjunto de datos si y sólo si el “key” o “id” del segundo conjunto corresponde con el “key” o “id” del primer conjunto de datos.
- Parecido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

Unir datos - Merge|Join|Buscarv

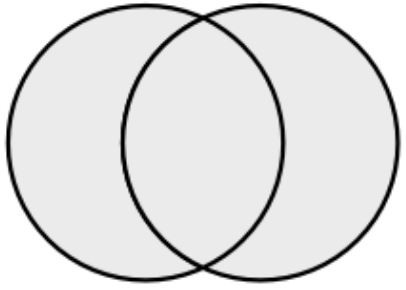
Entendiendo los tipos de Join



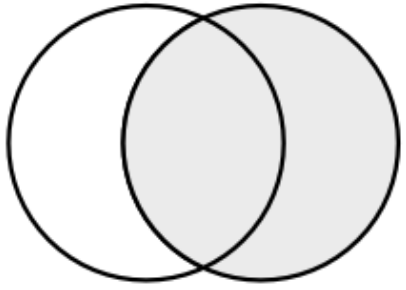
inner_join(x, y)



left_join(x, y)



full_join(x, y)



right_join(x, y)

Unir datos - Merge|Join|Buscarv

Creamos el DF para los ejemplos

```
df_6 = pd.DataFrame({
  'A': ['Ana', 'Daniel','Jose'],
  'B' : [100,200,300]
    })
df_6
```

```
##          A    B
## 0      Ana  100
## 1   Daniel  200
## 2     Jose  300
```

Unir datos - Inner Join

```
pd.merge(df_1, df_6, left_on='Nombre', right_on= 'A')
```

```
##     Nombre  Edad Ciudad       A    B
## 0      Ana    20    Gye     Ana  100
## 1   Daniel    19    Gye  Daniel  200
```

```
df_1.merge(df_6, left_on='Nombre', right_on= 'A')
```

```
##     Nombre  Edad Ciudad       A    B
## 0      Ana    20    Gye     Ana  100
## 1   Daniel    19    Gye  Daniel  200
```

Unir datos - Inner Join

```
pd.merge(df_1, df_6, how='inner', left_on='Nombre', right_on= 'A')
# df_1.merge(df_6, how='inner', left_on='Nombre', right_on= 'A')
```

```
##     Nombre  Edad Ciudad       A    B
## 0      Ana    20    Gye     Ana  100
## 1   Daniel    19    Gye  Daniel  200
```

Unir datos - Left Join

```
pd.merge(df_1, df_6, how='left', left_on='Nombre', right_on= 'A')
# df_1.merge(df_6, how='left', left_on='Nombre', right_on= 'A')
```

```
##     Nombre  Edad Ciudad       A      B
## 0      Ana    20    Gye     Ana  100.0
## 1    Berni    19    Uio     NaN    NaN
## 2   Carlos    20    Cue     NaN    NaN
## 3   Daniel    19    Gye  Daniel  200.0
## 4   Ericka    18    Cue     NaN    NaN
```

Unir datos - Right Join

```
pd.merge(df_1, df_6, how='right', left_on='Nombre', right_on= 'A')
# df_1.merge(df_6, how='right', left_on='Nombre', right_on= 'A')
```

```
##     Nombre  Edad Ciudad       A    B
## 0      Ana  20.0    Gye     Ana  100
## 1   Daniel  19.0    Gye  Daniel  200
## 2      NaN   NaN    NaN    Jose  300
```

Unir datos - Full Join

```
pd.merge(df_1, df_6, how='outer', left_on='Nombre', right_on= 'A')
# df_1.merge(df_6, how='outer', left_on='Nombre', right_on= 'A')
```

```
##     Nombre  Edad Ciudad       A      B
## 0      Ana  20.0    Gye     Ana  100.0
## 1    Berni  19.0    Uio     NaN    NaN
## 2   Carlos  20.0    Cue     NaN    NaN
## 3   Daniel  19.0    Gye  Daniel  200.0
## 4   Ericka  18.0    Cue     NaN    NaN
## 5      NaN   NaN    NaN    Jose  300.0
```

Unir datos - Merge|Join|Buscarv

Vamos a duplicar un Valor en df_6 y a replicar los Joins para revisar qué sucede cuando se tiene "key" no únicos

```
df_6 = pd.DataFrame({
  'A': ['Ana', 'Daniel', 'Jose', 'Ana'],
  'B' : [100,200,300, 110]
    })
df_6
```

```
##          A    B
## 0      Ana  100
## 1   Daniel  200
## 2     Jose  300
## 3      Ana  110
```

Unir datos - Inner Join

```
pd.merge(df_1, df_6, how='inner', left_on='Nombre', right_on= 'A')
# df_1.merge(df_6, how='inner', left_on='Nombre', right_on= 'A')
```

```
##     Nombre  Edad Ciudad       A    B
## 0      Ana    20    Gye     Ana  100
## 1      Ana    20    Gye     Ana  110
## 2   Daniel    19    Gye  Daniel  200
```

Unir datos - Join

Existe también el método .join que concatena usando los indices (index) de los dataframes

```
df_1
```

```
##    Nombre  Edad Ciudad
## 0     Ana    20    Gye
## 1   Berni    19    Uio
## 2  Carlos    20    Cue
## 3  Daniel    19    Gye
## 4  Ericka    18    Cue
```

```
df_6
```

```
##          A    B
## 0     Ana  100
## 1  Daniel  200
## 2    Jose  300
## 3     Ana  110
```

Unir datos - Join

Existe también el método .join que concatena usando los indices (index) de los dataframes

```
#Esto va a dar un resultado erróneo porque
# .join hace la union usando los índices numéricos
df_1.join(df_6, how='inner')
```

```
##    Nombre  Edad Ciudad       A    B
## 0     Ana    20    Gye     Ana  100
## 1   Berni    19    Uio  Daniel  200
## 2  Carlos    20    Cue    Jose  300
## 3  Daniel    19    Gye     Ana  110
```

Unir datos - Join

Existe también el método .join que concatena usando los indices (index) de los dataframes

```
# Para usar .join se debe definir los
# indices en cada dataframe
df_1.set_index('Nombre').join(df_6.set_index('A'), how='inner')
```

```
##         Edad Ciudad    B
## Ana       20    Gye  100
## Ana       20    Gye  110
## Daniel    19    Gye  200
```

Unir datos - Join

Existe también el método .join que concatena usando los indices (index) de los dataframes

```
# O definir el indice en el segundo DF y usar el
# argumento "on" para definir la columna del 1er DF
df_1.join(df_6.set_index('A'), on='Nombre', how='inner')
```

```
##    Nombre  Edad Ciudad    B
## 0     Ana    20    Gye  100
## 0     Ana    20    Gye  110
## 3  Daniel    19    Gye  200
```

Unir datos - Caso de estudio

Vamos a agregar la información de las sucursales a las transacciones

```
data_banco_xlsx= pd.merge( data_banco_xlsx.rename(columns={'Sucursal':'ID_Sucursal'}),
         data_sucursal, on='ID_Sucursal')
#Verificar
data_banco_xlsx[ ['ID_Sucursal', 'Sucursal']].drop_duplicates()
```

```
##       ID_Sucursal       Sucursal
## 0              62  Riocentro Sur
## 2838           85         Centro
## 14882         267       Alborada
## 18211         443   Mall del Sol
## 22401         586      Via Daule
```

Pivotear los datos

Introducción al análisis de datos

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

Katherine Morales / Néstor Montaño

Pivot

pivot_table permite un funcionamiento parecido a una tabla dinámica de excel, se define lo que va a nivel de filas (index), niveles en las columnas (columns), variables a operar (values) y las funciones para resumir la data a usar (aggfunc)

```
pd.pivot_table(data_banco_xlsx,
               index=["Sucursal"],
               columns=["Satisfaccion"],
               values= ['Tiempo_Servicio_seg'],
               aggfunc=np.mean)
```

```
##                Tiempo_Servicio_seg                ...
## Satisfaccion              Muy Malo        Malo  ...       Bueno   Muy Bueno
## Sucursal                                        ...
## Alborada                178.288403  176.860435  ...  182.631086  202.188738
## Centro                  149.943312  154.821875  ...  173.950992  171.561675
## Mall del Sol            150.047746  162.763379  ...  185.257154  191.550564
## Riocentro Sur            85.751740   86.350993  ...   92.529240   92.495826
## Via Daule                80.352239   77.351706  ...   82.795337   89.141907
##
## [5 rows x 5 columns]
```

Pivot

pivot_table permite un funcionamiento parecido a una tabla dinámica de excel, se define lo que va a nivel de filas (index), niveles en las columnas (columns), variables a operar (values) y las funciones para resumir la data a usar (aggfunc)

```
pd.pivot_table(data_banco_xlsx,
               index=["Sucursal"],
               columns=["Satisfaccion"],
               values= ['Tiempo_Servicio_seg'],
               aggfunc=np.count_nonzero)
```

```
##                Tiempo_Servicio_seg
## Satisfaccion             Muy Malo    Malo Regular   Bueno Muy Bueno
## Sucursal
## Alborada                    539.0   848.0   642.0   707.0     593.0
## Centro                     1243.0  1968.0  2309.0  3094.0    3430.0
## Mall del Sol                461.0   673.0   823.0  1044.0    1189.0
## Riocentro Sur               431.0   604.0   520.0   684.0     599.0
## Via Daule                   335.0   381.0   345.0   386.0     451.0
```

Pivot

pivot_table permite un funcionamiento parecido a una tabla dinámica de excel, se define lo que va a nivel de filas (index), niveles en las columnas (columns), variables a operar (values) y las funciones para resumir la data a usar (aggfunc)

```
pd.pivot_table(
        data_banco_xlsx,
        index=["Sucursal", "Transaccion"],
        columns=["Satisfaccion"],
        values= ['Tiempo_Servicio_seg'],
        aggfunc=np.count_nonzero)
```

```
##                                          Tiempo_Servicio_seg  ...
## Satisfaccion                                        Muy Malo  ... Muy Bueno
## Sucursal      Transaccion                                     ...
## Alborada      Cobrar cheque (Cta del Bco)               144.0  ...     232.0
##               Cobro/Pago (Cta externa)                   70.0  ...     100.0
##               Deposito                                  325.0  ...     261.0
## Centro        Cobrar cheque (Cta del Bco)               154.0  ...     639.0
##               Cobro/Pago (Cta externa)                   59.0  ...     516.0
##               Deposito                                 1030.0  ...    2275.0
## Mall del Sol  Cobrar cheque (Cta del Bco)                77.0  ...     423.0
##               Cobro/Pago (Cta externa)                   37.0  ...     204.0
```

Pivot

pivot_table permite un funcionamiento parecido a una tabla dinámica de excel, se define lo que va a nivel de filas (index), niveles en las columnas (columns), variables a operar (values) y las funciones para resumir la data a usar (aggfunc)

```
pd.pivot_table(
        data_banco_xlsx,
        index=["Sucursal"],
        columns=["Transaccion", "Satisfaccion"],
        values= ['Tiempo_Servicio_seg'],
        aggfunc=np.count_nonzero)
```

```
##                       Tiempo_Servicio_seg          ...
## Transaccion   Cobrar cheque (Cta del Bco)          ... Deposito
## Satisfaccion                       Muy Malo   Malo  ...    Bueno Muy Bueno
## Sucursal                                         ...
## Alborada                              144.0  277.0  ...    362.0     261.0
## Centro                                154.0  285.0  ...   2059.0    2275.0
## Mall del Sol                           77.0   95.0  ...    574.0     562.0
## Riocentro Sur                          85.0  122.0  ...    390.0     335.0
## Via Daule                              45.0   61.0  ...    256.0     279.0
##
## [5 rows x 15 columns]
```

Pivot

pivot_table permite un funcionamiento parecido a una tabla dinámica de excel, se define lo que va a nivel de filas (index), niveles en las columnas (columns), variables a operar (values) y las funciones para resumir la data a usar (aggfunc)

```
pd.pivot_table(
        data_banco_xlsx,
        index=["Sucursal"],
        columns=["Transaccion"],
        values= ['Tiempo_Servicio_seg', 'Monto'],
        aggfunc=np.mean)
```

```
##                                      Monto  ... Tiempo_Servicio_seg
## Transaccion   Cobrar cheque (Cta del Bco)  ...             Deposito
## Sucursal                                   ...
## Alborada                      2133.384149  ...           129.699062
## Centro                        2153.427773  ...           130.383375
## Mall del Sol                  2149.999323  ...           129.725190
## Riocentro Sur                 1832.281773  ...            65.481672
## Via Daule                     1789.757928  ...            67.269174
##
## [5 rows x 6 columns]
```

Pivot

pivot_table permite un funcionamiento parecido a una tabla dinámica de excel, se define lo que va a nivel de filas (index), niveles en las columnas (columns), variables a operar (values) y las funciones para resumir la data a usar (aggfunc)

```
pd.pivot_table(
        data_banco_xlsx,
        index=["Sucursal"],
        columns=["Transaccion"],
        values= ['Tiempo_Servicio_seg', 'Monto'],
        aggfunc=[np.mean, np.count_nonzero],
        margins=True)
```

```
##                                      mean  ...       count_nonzero
##                                     Monto  ... Tiempo_Servicio_seg
## Transaccion   Cobrar cheque (Cta del Bco)  ...                 All
## Sucursal                                  ...
## Alborada                      2133.384149  ...              3329.0
## Centro                        2153.427773  ...             12044.0
## Mall del Sol                  2149.999323  ...              4190.0
## Riocentro Sur                 1832.281773  ...              2838.0
## Via Daule                     1789.757928  ...              1898.0
## All                           2079.749432  ...             24299.0
```

Pivot

pivot_table permite un funcionamiento parecido a una tabla dinámica de excel, se define lo que va a nivel de filas (index), niveles en las columnas (columns), variables a operar (values) y las funciones para resumir la data a usar (aggfunc)

```
pd.pivot_table(
        data_banco_xlsx,
        index=["Sucursal"],
        columns=["Transaccion"],
        values= ['Tiempo_Servicio_seg', 'Monto'],
        aggfunc= {'Tiempo_Servicio_seg': np.mean,
                  'Monto': [np.mean, np.std]})
```

```
##                                    Monto  ... Tiempo_Servicio_seg
##                                     mean  ...                mean
## Transaccion   Cobrar cheque (Cta del Bco)  ...            Deposito
## Sucursal                                  ...
## Alborada                      2133.384149  ...          129.699062
## Centro                        2153.427773  ...          130.383375
## Mall del Sol                  2149.999323  ...          129.725190
## Riocentro Sur                 1832.281773  ...           65.481672
## Via Daule                     1789.757928  ...           67.269174
##
```

Melt and pivot

Melt y pivot son funciones que nos permiten pasar valores de filas a columnas

```
df= pd.DataFrame({
        'Sucursal': ["Norte", 'Centro', 'Sur'],
        '2017': [100, 101, 102],
        '2018': [200, 201, 202],
        '2019': [300, 301, 302]
        })
df
```

```
##   Sucursal  2017  2018  2019
## 0    Norte   100   200   300
## 1   Centro   101   201   301
## 2      Sur   102   202   302
```

Melt and pivot

En Melt se definen las columnas que van a ser ahora filas (value_vars), el resultado es un dataframe "más largo"

```
pd.melt(df, id_vars=['Sucursal'],
        value_vars=['2017','2018','2019'],
        var_name='Anio',
        value_name='TotalTransacciones')
```

```
##   Sucursal  Anio  TotalTransacciones
## 0    Norte  2017                 100
## 1   Centro  2017                 101
## 2      Sur  2017                 102
## 3    Norte  2018                 200
## 4   Centro  2018                 201
## 5      Sur  2018                 202
## 6    Norte  2019                 300
## 7   Centro  2019                 301
## 8      Sur  2019                 302
```

Melt and pivot

Pivot es lo contrario de melt, permite convertir los niveles de una columna en nuevas columnas, el resultado es un dataframe "más ancho"

```
dfmelt =pd.melt(df, id_vars=['Sucursal'],
        value_vars=['2017','2018','2019'],
        var_name='Anio',
        value_name='TotalTransacciones')

pd.pivot( dfmelt,
         index='Sucursal',
         columns='Anio',
         values='TotalTransacciones' )
```

```
## Anio       2017  2018  2019
## Sucursal
## Centro      101   201   301
## Norte       100   200   300
## Sur         102   202   302
```

Fin

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

Katherine Morales / Néstor Montaño