

Universidad Especializada de las Américas
Licenciatura en Biomédica con Especialización en Electrónica Médica
Programación Avanzada

Taller de Módulo Pandas de Python para manejo de datos.

Ver:1.0 03/junio/2017

Importante: los programas para instalar o ejecutar se deben hacer desde el modo Administrador, de lo contrario puede aparecer un mensaje de error por acceso denegado, esto ocurre mucho cuando los programas se instalan en Archivos de Programas o Programs Files y se intenta modificar algún archivo.

Prerrequisitos:

Haber leído , instalado y configurado los programas explicados en los documentos:

- Herramientas de Programación usadas en clase. Parte I.
- Herramientas de Programación usadas en clase. Parte II.

Haber leído los siguientes documentos:

- Taller de base de datos I.
- Taller de base de datos II.

Haber culminado y aprendido el contenido del curso:

- Curso I de Python (saber y entender las instrucciones fundamentales del lenguaje de programación Python).

En el curso usaremos la base de datos FILM.FBD descargada de :

<https://github.com/damianquijano/PythonCurso3/tree/master/Data>

junto con otros archivos en formato CSV.

Utilizaremos los softwares:

- Firebird.
- SQL Manager Lite .
- Pyzo
- Python.
- Tener Instalado el módulo Pandas y el conector Firebird para Python(FBD) en Pyzo , explicado en el documento “Herramientas de Programación usadas en clase. Parte I.”
- Todos los scripts python usados en el documento están en:

Objetivos .

1. Saber la función y utilidad del módulo Pandas y FBD..
2. Aprender el uso de una Dataframe.
3. Aprender a leer archivos externos y convertirlos en un Dataframe.
4. Aprender las operaciones de mayor uso con Dataframe.
5. Aprender conectar con Firebird y convertir los datos en Dataframe.

6. Aprender construir y ejecutar queries desde Pandas en Python a la base de datos Firebird.

1. Saber la función y utilidad del módulo Pandas y FBD

El módulo Pandas permite a los programadores de Python recibir datos externos o internos y convertirlos en una tabla muy parecida a Excell (hoja de cálculo); la presentación es mucho más entendible y familiar en vez de usar el formato más tradicional de una matriz.

Además Pandas ofrece muchas funciones que permiten manipular o seleccionar registros y columnas de la data de forma sencilla, además nos permite utilizar funciones de sumas, promedios y otras funciones (para ser más precisos son métodos) de gran utilidad. Pandas tiene una potencia similar a un gestor de base de datos pero en el contexto de programación Python. Las modificaciones o las operaciones que afectan a los datos, no afectan al origen de dichos datos, a menos que lo indiquemos.

2. Aprender el uso de una Dataframe.

La técnica de Pandas permite que se copien los datos de diversas fuentes de información y se conviertan en un contenedor llamado Dataframe que a su vez ofrece todo un kit de métodos para operar sobre la data recogida.

Veamos un ejemplo: archivo llamado Pandas1.py.

```
import numpy as np

import pandas as pd # importamos o añadimos el módulo pandas al programa y le asignamos
# el alias pd

# A continuación tenemos una colección de datos llamada diccionario, la cual inicia y
# termina con llaves, y dentro tenemos filas conformadas cada una por dos partes, el
# nombre de la fila a la izquierda de dos puntos y el valor de la fila a la derecha de
# los dos puntos y que es una lista con elementos dentro de ella.

data = {'regiment': ['Nighthawks', 'Nighthawks', 'Nighthawks', 'Nighthawks', 'Dragoons',
'Dragoons', 'Dragoons', 'Dragoons', 'Scouts', 'Scouts', 'Scouts', 'Scouts'],

        'company': ['1st', '1st', '2nd', '2nd', '1st', '1st', '2nd', '2nd', '1st', '1st',
'2nd', '2nd'],

        'name': ['Miller', 'Jacobson', 'Ali', 'Milner', 'Cooze', 'Jacon', 'Ryaner',
'Sone', 'Sloan', 'Piger', 'Riani', 'Ali'],

        'preTestScore': [4, 24, 31, 2, 3, 4, 24, 31, 2, 3, 2, 3],

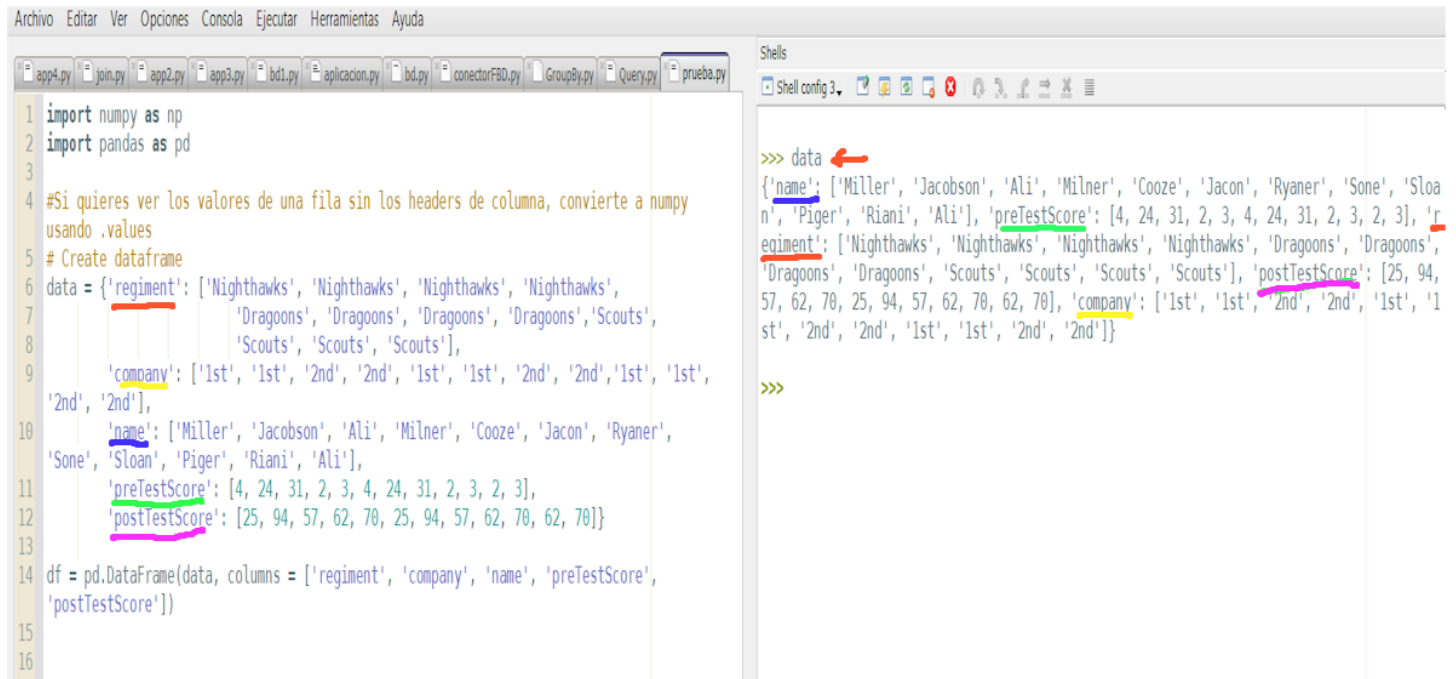
        'postTestScore': [25, 94, 57, 62, 70, 25, 94, 57, 62, 70, 62, 70]}

# A continuación convertimos el diccionario data en un dataframe:

df = pd.DataFrame(raw_data, columns = ['regiment', 'company', 'name', 'preTestScore',
'postTestScore'])
```

Ahora tenemos que ejecutar el script para que las variables se carguen en memoria y podamos usar la consola.

Escribamos en la consola : data , vean lo que aparece:



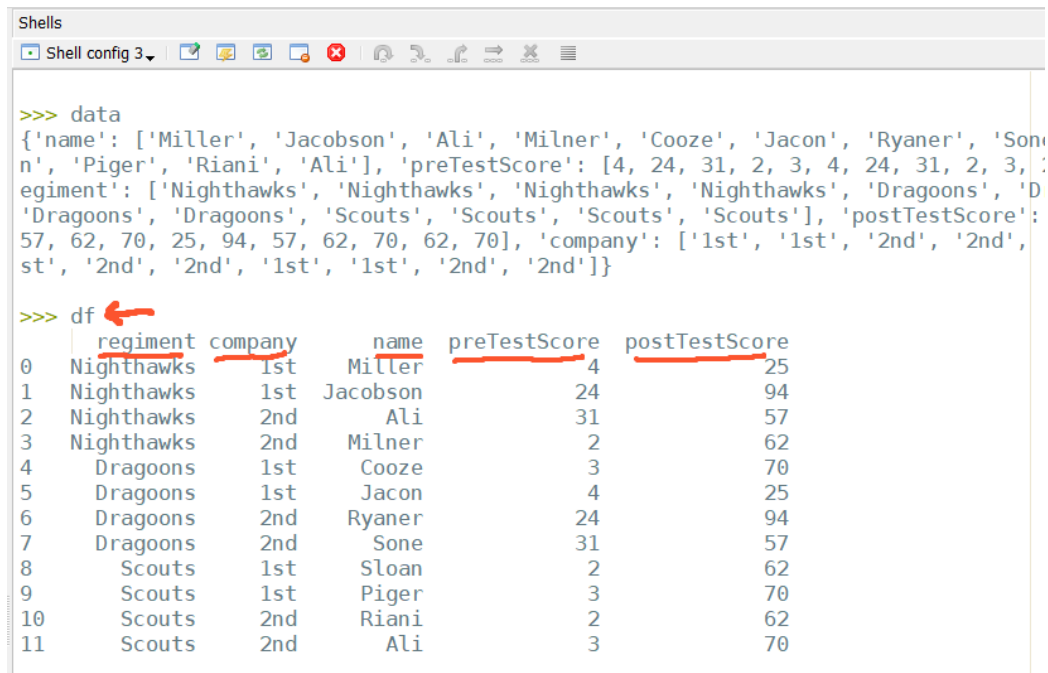
The screenshot shows an IDE with a Python script on the left and a shell window on the right. The script defines a dictionary 'data' with keys 'regiment', 'company', 'name', 'preTestScore', and 'postTestScore'. The shell window shows the command '>>> data' followed by the dictionary's representation, with some values highlighted in red and green in the original image.

```
1 import numpy as np
2 import pandas as pd
3
4 #Si quieres ver los valores de una fila sin los headers de columna, convierte a numpy
  usando .values
5 # Create dataframe
6 data = {'regiment': ['Nighthawks', 'Nighthawks', 'Nighthawks', 'Nighthawks',
7                    'Dragoons', 'Dragoons', 'Dragoons', 'Dragoons', 'Scouts',
8                    'Scouts', 'Scouts'],
9         'company': ['1st', '1st', '2nd', '2nd', '1st', '1st', '2nd', '2nd', '1st', '1st',
10                   '2nd', '2nd'],
11         'name': ['Miller', 'Jacobson', 'Ali', 'Milner', 'Cooze', 'Jacon', 'Ryaner',
12                 'Sone', 'Sloan', 'Piger', 'Riani', 'Ali'],
13         'preTestScore': [4, 24, 31, 2, 3, 4, 24, 31, 2, 3, 2, 3],
14         'postTestScore': [25, 94, 57, 62, 70, 25, 94, 57, 62, 70, 62, 70]}
15
16 df = pd.DataFrame(data, columns = ['regiment', 'company', 'name', 'preTestScore',
17                                  'postTestScore'])
```

```
>>> data
{'name': ['Miller', 'Jacobson', 'Ali', 'Milner', 'Cooze', 'Jacon', 'Ryaner', 'Sone', 'Sloan', 'Piger', 'Riani', 'Ali'], 'preTestScore': [4, 24, 31, 2, 3, 4, 24, 31, 2, 3, 2, 3], 'regiment': ['Nighthawks', 'Nighthawks', 'Nighthawks', 'Nighthawks', 'Dragoons', 'Dragoons', 'Dragoons', 'Dragoons', 'Scouts', 'Scouts', 'Scouts'], 'postTestScore': [25, 94, 57, 62, 70, 25, 94, 57, 62, 70, 62, 70], 'company': ['1st', '1st', '2nd', '2nd', '1st', '1st', '2nd', '2nd', '1st', '1st', '2nd', '2nd']}
```

A la derecha vemos el formato de presentación de la data, es el formato convencional en matriz.

Ahora escribamos en la consola: df , y vean la diferencia:



The screenshot shows a shell window with the command '>>> df' followed by the output of the DataFrame, which is displayed as a table. The table has columns 'regiment', 'company', 'name', 'preTestScore', and 'postTestScore'. The first few rows are highlighted in red in the original image.

```
>>> data
{'name': ['Miller', 'Jacobson', 'Ali', 'Milner', 'Cooze', 'Jacon', 'Ryaner', 'Sone', 'Sloan', 'Piger', 'Riani', 'Ali'], 'preTestScore': [4, 24, 31, 2, 3, 4, 24, 31, 2, 3, 2, 3], 'regiment': ['Nighthawks', 'Nighthawks', 'Nighthawks', 'Nighthawks', 'Dragoons', 'Dragoons', 'Dragoons', 'Dragoons', 'Scouts', 'Scouts', 'Scouts'], 'postTestScore': [25, 94, 57, 62, 70, 25, 94, 57, 62, 70, 62, 70], 'company': ['1st', '1st', '2nd', '2nd', '1st', '1st', '2nd', '2nd', '1st', '1st', '2nd', '2nd']}
```

```
>>> df
   regiment company  name  preTestScore  postTestScore
0  Nighthawks    1st  Miller             4             25
1  Nighthawks    1st Jacobson            24             94
2  Nighthawks    2nd   Ali             31             57
3  Nighthawks    2nd  Milner             2             62
4  Dragoons     1st   Cooze             3             70
5  Dragoons     1st   Jacon             4             25
6  Dragoons     2nd  Ryaner            24             94
7  Dragoons     2nd   Sone            31             57
8    Scouts     1st  Sloan             2             62
9    Scouts     1st   Piger             3             70
10   Scouts     2nd   Riani             2             62
11   Scouts     2nd    Ali             3             70
```

3. Aprender a leer archivos externos y convertirlos en un Dataframe.

Cuatro tipos de fuentes de datos son los que más usaremos para nuestro curso:

1.Base de datos.

2.Archivos csv.

3.Diccionarios o matrices creadas en nuestro programa de python tal como hemos visto en nuestro ejemplo en el punto 2.

Los casos 1 y 2 requieren que previamente se incorporen dichos datos antes de convertirlos en dataframe.

Veamos el caso de los archivos csv (recuerde que tenemos los datos descargados en c:\data).Usaremos el método read_csv para leer documentos del tipo delimitado .csv , los cuales a su vez se abren fácilmente en un bloque de notas o en excell.

Archivo: Pandas2.py

```
import numpy as np
```

```
import pandas as pd
```

```
data = pd.read_csv('c:\data\Film20SD.csv')
```

Ejecute y luego en la consola escriba y ejecute: data

```
>>> data
   film_id  category_id      title  release_year  language_id  \
0         1           11  AFFAIR PREJUDICE         2010          1
1         2           11  AIRPORT POLLOCK         2016          6
2         3           11  ALABAMA DEVIL         2011          3
3         4           11  ALI FOREVER         2008          1
4         5           16  BASIC EASY         2007          3
5         6            1  CAMPUS REMEMBER         2014          3
6         7           15  CARIBBEAN LIBERTY         2015          5
7         8            2  CAROL TEXAS         2006          2
8         9            1  CASUALTIES ENCINO         2013          1
9        10            9  CATCH AMISTAD         2016          5
10        11            6  DELIVERANCE MULHOLLAND         2006          4
11        12           16  DESPERATE TRAINSPOTTING         2013          3
12        13            4  DETECTIVE VISION         2013          5
13        14            7  DIARY PANIC         2016          2
14        15            8  EFFECT GLADIATOR         2013          3
15        16            6  EGG IGBY         2012          1
16        17           12  ELF MURDER         2007          2
17        18           16  ENOUGH RAGING         2006          1
18        19           15  EVOLUTION ALTER         2008          3

   rental_duration  rental_rate  length  replacement_cost  rating
0                5          2.99     117             26.99      G
1                6          4.99      54             15.99      R
2                3          2.99     114             21.99  PG-13
3                4          4.99     150             21.99    PG
4                4          2.99      90             18.99  PG-13
5                5          2.99     167             27.99      R
6                3          4.99      92             16.99  NC-17
7                4          2.99     151             15.99    PG
8                3          4.99     179             16.99      G
9                7          0.99     183             10.99      G
10               4          0.99     100              9.99      R
11               7          4.99      81             29.99      G
12               4          0.99     143             16.99  PG-13
13               7          2.99     107             20.99      G
14               6          0.99     107             14.99    PG
15               4          2.99      67             20.99    PG
16               4          4.99     155             19.99  NC-17
17               7          2.99     158             16.99  NC-17
18               5          0.99     174             10.99  PG-13
```

Enseguida notamos que la instrucción `read_csv` carga la información del archivo y la convierte a formato Dataframe. Arriba vemos los datos de Film20SD. Dado que son muchas columnas, Pandas realiza una separación de columnas mediante la barra `\`, esto produce que primeros veamos los registros de una cantidad de columnas y más abajo vemos los registros del resto de la columnas.

Para evitar la separación mencionada arriba, usamos la instrucción `set_option` que permite configurar el módulo Pandas para variar la presentación de los resultados. La instrucción `pd.set_option` permite modificar la presentación.

pandas3.py

```
import numpy as np

import pandas as pd

pd.set_option('expand_frame_repr', False)

data = pd.read_csv('c:\data\Film20SD.csv')
```

```
>>> data
   film_id  category_id      title  release_year  language_id  rental_duration  rental_rate  length  replacement_cost  rating
0         1           11  AFFAIR PREJUDICE        2010           1             5           2.99      117           26.99         G
1         2           11  AIRPORT POLLOCK        2016           6             6           4.99       54           15.99         R
2         3           11    ALABAMA DEVIL        2011           3             3           2.99      114           21.99  PG-13
3         4           11    ALI FOREVER        2008           1             4           4.99      150           21.99         PG
4         5           16    BASIC EASY        2007           3             4           2.99       90           18.99  PG-13
5         6            1  CAMPUS REMEMBER        2014           3             5           2.99      167           27.99         R
6         7           15  CARIBBEAN LIBERTY        2015           5             3           4.99       92           16.99  NC-17
7         8            2    CAROL TEXAS        2006           2             4           2.99      151           15.99         PG
8         9            1  CASUALTIES ENCINO        2013           1             3           4.99      179           16.99         G
9        10            9    CATCH AMISTAD        2016           5             7           0.99     183           10.99         G
10       11            6  DELIVERANCE MULHOLLAND        2006           4             4           0.99      100           9.99         R
11       12           16  DESPERATE TRAINSPOTTING        2013           3             7           4.99       81           29.99         G
12       13            4    DETECTIVE VISION        2013           5             4           0.99     143           16.99  PG-13
13       14            7    DIARY PANIC        2016           2             7           2.99     107           20.99         G
14       15            8    EFFECT GLADIATOR        2013           3             6           0.99     107           14.99         PG
15       16            6    EGG IGBY        2012           1             4           2.99       67           20.99         PG
16       17           12    ELF MURDER        2007           2             4           4.99     155           19.99  NC-17
17       18           16    ENOUGH RAGING        2006           1             7           2.99     158           16.99  NC-17
18       19           15  EVOLUTION ALTER        2008           3             5           0.99     174           10.99  PG-13
```

Arriba vemos que aparecen todas las columnas seguidas, esto es importante a la hora de copiar y pegar contenidos posteriormente.

Vamos a cambiar el archivo que leemos y lo haremos por otro similar pero con 100 registros:

```
import numpy as np

import pandas as pd

pd.set_option('expand_frame_repr', False)

data = pd.read_csv('c:\data\Film100SD.csv')
```

Ejecute el script y escriba y ejecute la variable `data` en la consola para ver los resultados.

Los resultados son:

	film_id	category_id	title	release_year	language_id	rental_duration	rental_rate	length	replacement_cost	rating
0	1	11	AFFAIR PREJUDICE	2010	1	5	2.99	117	26.99	G
1	2	11	AIRPORT POLLOCK	2016	6	6	4.99	54	15.99	R
2	3	11	ALABAMA DEVIL	2011	3	3	2.99	114	21.99	PG-13
3	4	11	ALI FOREVER	2008	1	4	4.99	150	21.99	PG
4	5	16	BASIC EASY	2007	3	4	2.99	90	18.99	PG-13
5	6	1	CAMPUS REMEMBER	2014	3	5	2.99	167	27.99	R
6	7	15	CARIBBEAN LIBERTY	2015	5	3	4.99	92	16.99	NC-17
7	8	2	CAROL TEXAS	2006	2	4	2.99	151	15.99	PG
8	9	1	CASUALTIES ENCINO	2013	1	3	4.99	179	16.99	G
9	10	9	CATCH AMISTAD	2016	5	7	0.99	183	10.99	G
10	11	6	DELIVERANCE MULHOLLAND	2006	4	4	0.99	100	9.99	R
11	12	16	DESPERATE TRAINSPOTTING	2013	3	7	4.99	81	29.99	G
12	13	4	DETECTIVE VISION	2013	5	4	0.99	143	16.99	PG-13
13	14	7	DIARY PANIC	2016	2	7	2.99	107	20.99	G
14	15	8	EFFECT GLADIATOR	2013	3	6	0.99	107	14.99	PG
15	16	6	EGG IGBY	2012	1	4	2.99	67	20.99	PG
16	17	12	ELF MURDER	2007	2	4	4.99	155	19.99	NC-17
17	18	16	ENOUGH RAGING	2006	1	7	2.99	158	16.99	NC-17
18	19	15	EVOLUTION ALTER	2008	3	5	0.99	174	10.99	PG-13
19	20	10	FEATHERS METAL	2008	1	3	0.99	104	12.99	PG-13
20	21	8	GABLES METROPOLIS	2013	5	3	0.99	161	17.99	PG
21	22	16	GAMES BOWFINGER	2007	2	7	4.99	119	17.99	PG-13
22	23	9	GENTLEMAN STAGE	2009	2	6	2.99	125	22.99	NC-17
23	24	7	GOLDFINGER SENSIBILITY	2009	4	3	0.99	93	29.99	G
24	25	3	HALL CASSIDY	2010	2	5	4.99	51	13.99	NC-17
25	26	7	HANGING DEEP	2014	1	5	4.99	62	18.99	G
26	27	12	IMPACT ALADDIN	2011	4	6	0.99	180	20.99	PG-13
27	28	9	IMPOSSIBLE PREJUDICE	2013	5	7	4.99	103	11.99	NC-17
28	29	9	INNOCENT USUAL	2012	6	3	4.99	178	26.99	PG-13
29	30	15	INSTINCT AIRPORT	2011	6	4	2.99	116	21.99	PG
...
69	70	6	VIRGINIAN PLUTO	2016	6	5	0.99	164	22.99	R
70	71	2	WAIT CIDER	2015	3	3	0.99	112	9.99	PG-13
71	72	13	WAKE JAWS	2012	4	7	4.99	73	18.99	G
72	73	12	YOUTH KICK	2015	6	4	0.99	179	14.99	NC-17
73	74	8	AFRICAN EGG	2007	1	6	2.99	130	22.99	G
74	75	4	ALICE FANTASIA	2006	3	6	0.99	94	23.99	NC-17
75	76	1	ANTITRUST TOMATOES	2008	2	5	2.99	168	11.99	NC-17
76	77	1	BAREFOOT MANCHURIAN	2015	6	6	2.99	129	15.99	G
77	78	4	CANDIDATE PERDITION	2013	5	4	2.99	70	10.99	R
78	79	16	CASABLANCA SUPER	2010	4	6	4.99	85	22.99	G

Observe que hay un salto de 29 a 69. Esto lo hace Pandas para evitar saturar la memoria de la computadora y a su vez permite al usuario visualizar los primeros y los últimos registros de la data. Se entiende mejor si nos imaginamos una tablas con 1 millón de registros; no es razonable que el programe cargue el millón de registros siendo que solo queremos saber por encima el contenido de la tabla.

Para evitar esto, y que se muestren todos los registros de la tabla, nuevamente usaremos la instrucción `set_options("display.max_rows",None)`:

pandas4.py

```
import numpy as np

import pandas as pd

pd.set_option('expand_frame_repr', False)

pd.set_option("display.max_rows",None)

data = pd.read_csv('c:\data\Film100SD.csv')
```

Ejecute el script y escriba y ejecute la variable data en la consola para ver los resultados; veremos que no hay saltos y aparecen todos los registros.

También podemos leer los archivos desde el internet, pruebe lo siguiente:

pandas5.py

```
import numpy as np
import pandas as pd
pd.set_option('expand_frame_repr', False)
pd.set_option("display.max_rows",None)

data =
pd.read_csv('https://raw.githubusercontent.com/damianquijano/PythonCurso3/master/
Data/Film20SD.csv')
```

Arriba descargamos directamente del internet un archivo csv, en nuestro caso lo hemos descargado del repositorio Github utilizado por el curso:

The screenshot shows the GitHub repository page for 'damianquijano / PythonCurso3'. The 'Data' directory is selected, showing a list of files. A red arrow points to 'Film20SD.csv'.

File	Description	Time
FILM.FDB	Se ha subido Film.fdb	3 days ago
Film100SD.csv	Se han subido archivos de datos	3 days ago
Film20SD.csv	Se han subido archivos de datos	3 days ago
acerca	Creacion de carpeta Data	3 days ago
category.csv	Se han subido archivos de datos	3 days ago
language.csv	Se han subido archivos de datos	3 days ago

damianquijano / PythonCurso3

Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Wiki Settings Insights

Branch: master PythonCurso3 / Data / Film20SD.csv Find file Copy path

damianquijano Se han subido archivos de datos 7fddfc2 3 days ago

1 contributor

21 lines (20 sloc) | 1.01 KB

Raw Blame History

Search this file...

	film_id	category_id	title	release_year	language_id	rental_duration	rental_rate	length	replacement_cost	rating
1	1	11	AFFAIR PREJUDICE	2010	1	5	2.99	117	26.99	G
2	2	11	AIRPORT POLLOCK	2016	6	6	4.99	54	15.99	R
3	3	11	ALABAMA DEVIL	2011	3	3	2.99	114	21.99	PG-13
4	4	11	ALI FOREVER	2008	1	4	4.99	150	21.99	PG

Pyth x Pyth x arch x git .git x git git - x git git - x git x insta x Insta x

Seguro <https://raw.githubusercontent.com/damianquijano/PythonCurso3/master/Data/Film20SD.csv>

Aplicaciones Google Traductor de Google UDELAS - Universidad Inicio Wix SimpleSite.com Courser

```
film_id,category_id,title,release_year,language_id,rental_duration,rental_rate,length,replacement_cost,rating
1,11,AFFAIR PREJUDICE,2010,1,5,2.99,117,26.99,G
2,11,AIRPORT POLLOCK,2016,6,6,4.99,54,15.99,R
3,11,ALABAMA DEVIL,2011,3,3,2.99,114,21.99,PG-13
4,11,ALI FOREVER,2008,1,4,4.99,150,21.99,PG
5,16,BASIC EASY,2007,3,4,2.99,90,18.99,PG-13
6,1,CAMPUS REMEMBER,2014,3,5,2.99,167,27.99,R
7,15,CARIBBEAN LIBERTY,2015,5,3,4.99,92,16.99,NC-17
8,2,CAROL TEXAS,2006,2,4,2.99,151,15.99,PG
9,1,CASUALTIES ENCINO,2013,1,3,4.99,179,16.99,G
10,9,CATCH AMISTAD,2016,5,7,0.99,183,10.99,G
11,6,DELIVERANCE MULHOLLAND,2006,4,4,0.99,100,9.99,R
12,16,DESPERATE TRAINSPOTTING,2013,3,7,4.99,81,29.99,G
13,4,DETECTIVE VISION,2013,5,4,0.99,143,16.99,PG-13
14,7,DIARY DANCE,2016,2,7,2.99,107,20.99,G
```

Arriba vemos el link que usamos para descargar el archivo csv.

Si no desea cargar todas las filas, podemos limitar la carga:

```
data = pd.read_csv('c:\data\Film100SD.csv',nrows=3) esto nos carga solamente las
3 primeras filas.
```

4. Aprender las operaciones de mayor uso con Dataframe.

Utilizaremos el siguiente Dataframe (llamado **df**) para la práctica escribiendo el siguiente programa: pandas6.py

```
import numpy as np
import pandas as pd
```



```
pd.set_option('expand_frame_repr', False)
pd.set_option("display.max_rows",None)

data = {'regiment': ['Nighthawks', 'Nighthawks', 'Nighthawks',
'Nighthawks','Dragoons', 'Dragoons', 'Dragoons', 'Dragoons','Scouts',
'Scouts', 'Scouts', 'Scouts'],'company': ['1st', '1st', '2nd', '2nd', '1st',
'1st', '2nd', '2nd','1st', '1st', '2nd', '2nd'],'name': ['Miller', 'Jacobson',
'Ali', 'Milner', 'Cooze', 'Jacon', 'Ryaner', 'Sone', 'Sloan', 'Piger', 'Riani',
'Ali'],'preTestScore': [4, 24, 31, 2, 3, 4, 24, 31, 2, 3, 2, 3],'postTestScore':
[25, 94, 57, 62, 70, 25, 94, 57, 62, 70, 62, 70]}
```

A continuación convertimos el diccionario data en un dataframe:

```
df = pd.DataFrame(data, columns = ['regiment', 'company', 'name', 'preTestScore',
'postTestScore'])
```

Por tanto nuestro Dataframe es la variable **df** .

Para verificar nuestros valores, ejecutamos el script [pandas6.py](#) , esto permite cargar las variables **data** y **df** en memoria, y posibilita que podamos usar dichas variables en la consola(del lado derecho de la pantalla) , de lo contrario mostrará un mensaje de error en rojo que nos dirá que python no encuentra las variables **data** y **df**.

Después de ejecutar el script, vamos a ver los datos de la variable data y luego df. En la consola escriba data y luego haga Enter, después escriba df y luego de nuevo Enter. Veremos lo siguiente:

```
>>> data
{'preTestScore': [4, 24, 31, 2, 3, 4, 24, 31, 2, 3, 2, 3], 'regiment': ['Nighthawks', 'Nighthawks', 'Nighthawks', 'Nighthawks', 'Dragoons', 'Dragoons', 'Dragoons', 'Dragoons', 'Scouts', 'Scouts', 'Scouts', 'Scouts'], 'company': ['1st', '1st', '2nd', '2nd', '1st', '1st', '2nd', '2nd', '1st', '1st', '2nd', '2nd'], 'postTestScore': [25, 94, 57, 62, 70, 25, 94, 57, 62, 70, 62, 70], 'name': ['Miller', 'Jacobson', 'Ali', 'Milner', 'Cooze', 'Jacon', 'Ryaner', 'Sone', 'Sloan', 'Piger', 'Riani', 'Ali']}
```

```
>>> df
  company  name  postTestScore  preTestScore  regiment
0     1st  Miller             25             4  Nighthawks
1     1st Jacobson             94            24  Nighthawks
2     2nd   Ali              57            31  Nighthawks
3     2nd Milner             62             2  Nighthawks
4     1st  Cooze             70             3   Dragoons
5     1st  Jacon             25             4   Dragoons
6     2nd Ryaner             94            24   Dragoons
7     2nd  Sone              57            31   Dragoons
8     1st  Sloan             62             2     Scouts
9     1st  Piger             70             3     Scouts
10    2nd  Riani             62             2     Scouts
11    2nd   Ali             70             3     Scouts
```

Al momento que convertimos los datos de una tabla excell, csv o de otra fuente(como en nuestro caso , la hemos construido nosotros en el programa) en un dataframe de la librería pandas, se agrega a esa nueva estructura (dataframe) una columna que enumera desde 0 hasta el final cada fila de dicha tabla, son los índices. Esto no impide que se definan otros índices o la columna que se usa como índice en la base de datos origen de la tabla, pero es evidente que debe ser numérico.

Por general, los corchetes [] en python se utilizan para identificar una lista, por ejemplo: [1,3,6,'hola'] , pero también se usan para identificar valores de una matriz o de un arreglo de la librería numpy o -en nuestro caso- de un dataframe de la librería de pandas, por ejemplo:
df.iloc[5][1] , imprime el valor de la fila 5 y columna 1, por tanto los [] no representan en este caso una lista.

Una manera de visualizar porciones de la tabla dataframe, es mediante el uso de los índices(el número único y secuencial que identifica un registro de otro) y esto lo haremos mediante el método **iloc**. Existen muchas maneras de seleccionar, particionar o buscar los datos, por ejemplo con la instrucción loc (sin la i) o mediante la instrucción query, pero con el objeto de unificar criterios, utilizaremos iloc. Se menciona esto para evitar que se generen confusiones a la hora de consultar en internet y en otras fuentes que pueden usar otras técnicas para lograr lo mismo.

La **metodología** que usaremos es el aprendizaje con la práctica.

Como hemos mencionado antes, después de ejecutar el programa de arriba(pandas6.py) que ha creado el dataframe **df**, dicha variable dataframe (df) se carga en memoria y la puede llamar y usar en la consola, de lo contrario la consola no reconocerá la variable **df**.

Entonces, lea a continuación las instrucciones que debe escribir y ejecutar (una a una, no a la vez) en la consola(la pantalla a la derecha) y vea los resultados.

Primero veremos el contenido de **df**, por tanto escriba (después de ejecutar el programa) **df** en la consola y luego enter, entonces usted verá los datos que antes hemos mostrado, esto garantiza que ha ejecutado el programa correctamente, de lo contrario debe ejecutar el programa de nuevo. Si no se acuerda el contenido de toda la tabla, ejecute de nuevo **df** en la consola y copie los datos a un bloq de notas o a excell para poder consultar y comparar los resultados.

Acto seguido siga las instrucciones siguientes(escriba en la consola y luego pulse enter para ver los resultados):

Imprimir el valor de una celda de la tabla:

df.iloc[5][1] imprime el valor de la celda de la fila 5 y columna 1 , vemos que a la izquierda son filas, y a la derecha son columnas.

Otra opción más general es: df.iloc[5,1]

El formato universal de iloc:

df.iloc[1:3,2:3] ,en este formato notamos que se usa coma para separar filas y columnas, estos números separados por comas y dos puntos, representan intervalos secuenciales y se le llama Slicing(partir).

El valor 1:3 representa un intervalo, y se refiere a las filas que inician en 1 y termina en 2, o sea, no incluye el 3, por tanto veremos las filas 1 y 2.En en la parte derecha de la expresión, tenemos 2:3 que representa columnas y en este caso solo a la columna 2, pues no se incluye el 3.

Es necesario colocar en la parte derecha de los dos puntos (:) un número posterior al que se intenta incluir como el final, por tanto si deseamos ver las filas : 4,5,6 y 7, el intervalo es 4:8 , y si queremos visualizar solamente la fila 100, será 100:101.

Cuando dejas una parte (izquierda o derecha de los :) sin escribir nada, implica el inicio o el final de todo el conjunto de filas o columnas. Por ejemplo, si escribimos [:5] es como decir [desde el inicio: 5] o [0:5] y si escribimos [5:] es como escribir [5: hasta el final], y si escribimos [:] se refiere a todas las filas o todas las columnas, lo mismo que decir [el primero : el último] o sea el intervalo es desde el primer elemento (fila o columna) hasta el último elemento (hasta el final). Por ejemplo [:, 1:3], significa que veremos todas las filas para las columnas 1 y 2. Y [5:9, :] significa que veremos las filas de la 5 hasta la 8 y para todas las columnas. Qué ocurre si usamos números negativos? pues las cosas empiezan a funcionar de al revés: de último hacia el primero. Por ejemplo [-1:, :] nos muestra la última fila, pues -1 es el último elemento, -2 es el penúltimo...

Veamos lo que podemos visualizar usando el formato universal de iloc(escriba y pruebe cada una):

```
df.iloc[5:6,:] imprime el valor de la fila 5 y todas las columnas.
df.iloc[:5,:] imprime hasta la fila 4 y todas las columnas.
df.iloc[3:5,:] imprime fila de 3 a 4 y todas las columnas.
df.iloc[-1:,:] imprime la última fila y todas las columnas.
df.iloc[-5:,:] imprime las últimos 5 filas y todas las columnas
```

```
df.iloc[:1,:] imprime la primera fila (índice=0) y todas las columnas
df.iloc[:5,:] imprime las 5 primeras filas y todas las columnas.
```

Ahora podemos agregar columnas si queremos:

```
df.iloc[1:5,2:4] imprime las filas de la 1 a la 4, y las columnas de la 2 a la 3.
```

Pruebe las siguientes instrucciones:

```
df.iloc[-5:-1,:]
df.iloc[-5:-1,-3:-1]
```

Forma abreviada solo para Filas:

```
df.iloc[5] imprime el valor de la fila 5
a=df.iloc[5].values-->array(['Dragoons', '1st', 'Jacon', 4, 25], dtype=object)
df.iloc[:5] imprime hasta la fila 4
df.iloc[3:5] imprime de la fila 3 a la 4
df.iloc[-1:] imprime la última fila
df.iloc[-5:] imprime las últimas 5 filas
```

Forma abreviada para una solo fila y varias columnas:

```
df.iloc[5][1:3] imprime las columnas 1 y 3 para la fila 5, esto aplica para una fila.
```

Te lo muestra como un registro de datos en forma vertical.

Forma que permite ver intervalos de filas y columnas:

```
df.iloc[1:5, 1:3] imprime las filas de 1 a 4, y las columnas de 1 a 2.
```

Formas usando nombres de columnas e intervalos de columnas

```
df.iloc[1]['regiment']
df.iloc[[1,3,4]]['regiment']
df.iloc[1:5]['regiment']
```

```
df.iloc[1][['regiment','name']]
df.iloc[[1,4,8]][['regiment','name']]
df.iloc[1:5][['regiment','name']]
```

```
df.iloc[1:2][df.columns[1:3]]      o    df.iloc[1:2,1:3]  , una fila
df.iloc[[1,5,7]][df.columns[1:3]]  o    df.iloc[[1, 5, 7], 1: 3]
df.iloc[1:5][df.columns[1:3]]      o    df.iloc[1:5, 1: 3]
```

df1.iloc[[1, 5, 7], [1, 3]] nos imprime filas 1,5 7 y columnas 1 y 3.

Ver:<http://pandas.pydata.org/pandas-docs/stable/indexing.html>

Hasta ahora hemos usado los índices para extraer información del dataframe, pero también **podemos usar los símbolos de comparación**: ==, < , > , >=, etc a partir de las filas, lo cual no impide que limitemos la extracción a la columna o columnas que nos interesan. Para lograr esto, no se usa la instrucción iloc, usaremos los [] del dataframe directamente:

```
df[df.preTestScore == 24].iloc[:, :2]
df[(df.preTestScore == 24) & (df.postTestScore > 24)].iloc[:, :2]
df[df.preTestScore == 24][['company','name']]
df[(df.preTestScore == 24) & (df.postTestScore > 24)][['company','name']]
df[df.preTestScore == 24][df.columns[[1,3]]] en este caso seleccionamos que se
vean las columnas 1 y 3 solamente.
df[df.preTestScore.isin([4,31])] imprime las filas que contenga los valores 4 y
3 en la columna preTestScore, se ven todas las columnas.
df[df.preTestScore.isin([4,31]).iloc[:, :3] solo las columnas de 0 a 2.
df[df.preTestScore.isin([4,31]).iloc[:, [0,3]] lo mismo de arriba pero solo las
columnas 0 y 3
```

Otros ejemplos(el dataframe se llama surveys df):

```
surveys_df[surveys_df.year == 2002]
surveys_df[surveys_df.year != 2002]
surveys_df[(surveys_df.year >= 1980) & (surveys_df.year <= 1985)]
Otra forma diferida:
listabool=df.preTestScore > 4 al ser operaciones de comparación, devuelve bool
df[listabool]
```

Símbolos de comparación y lógicos:

```
# * Equals: ==
# * Not equals: !=
# * Greater than, less than: > or <
# * Greater than or equal to >=
# * Less than or equal to <=
# * Or: |
# * And: &
# You can use the isin command in Python to query a DataFrame based upon a list
of values as follows:
# surveys_df[surveys_df['species_id'].isin([listGoesHere])]
# Use the isin function to find all plots that contain particular species in the
"surveys" DataFrame. How many records contain these values?
# The ~ symbol in Python can be used to return the OPPOSITE of the selection that
you specify in Python. It is equivalent to is not in.
| (or), & (and), and ~ (not)
```

<http://pandas.pydata.org/pandas-docs/stable/generated/pandas.eval.html#pandas.eval>

Si queremos copiar el contenido de un dataframe a otro:

```
df2 = df.copy()
```

Si queremos asignar valores a un grupo de filas que deben cumplir una condición:

```
df2[ df2[1:4] > 0 ] = 3
```

Ordenar los datos de un dataframe:

```
dat.sort_values(by=['CURRENCY','COUNTRY'], ascending=[True,True])
```

<http://pandas.pydata.org/pandas-docs/stable/indexing.html>

Usar el método de query para simular un where de sql:

El uso del **query** es más lento, pues usa evaluación de expresiones.

```
df.query('preTestScore > 4')
```

```
df.query('preTestScore > 4 & postTestScore<94')
```

```
df.query("regiment=='Nighthawks' & postTestScore<94")
```

```
var='Nighthawks'
```

```
df.query("regiment==@var & postTestScore<94")
```

```
df.query('preTestScore > 4').head(2) #escoge los 2 primeros de los resultados
```

```
df.query('preTestScore > 4').iloc[:,[0, 1]] limita las columnas ,
```

```
y -si quiero- también las filas.
```

```
df.query('preTestScore > 4').iloc[:, :3] limita las columnas , y -si quiero-  
también las filas.
```

También se pueden construir agrupamientos, joins y otras acciones similares a un manejador de base de datos como SQL Managment Lite para Firebird dado que podemos crear datos dataframe como queramos, por tanto podemos cargar varias tablas diferentes y luego mediante el método merge podemos construir joins.

Pero la mejor estrategia es realizar dichas operaciones desde la base de datos, y traer a python mediante Pandas una sola tabla debidamente procesada de tal modo que limitemos las acciones de consultas mediante Pandas a operaciones simples y sencillas.

Para mayor información:

<http://pandas.pydata.org/pandas-docs/stable/indexing.html>

Métodos para lograr información de dataframes: se aplica a todo el dataframe o a una columna, según lo que se desea en el momento.

Previo es necesario conocer un vocabulario que aparece constantemente a la hora de usar las instrucciones.

axis=0 se refiere a filas y axis 1 es columnas

inplace: Modifica el dataframe sobre el que se trabaja, no crea copia. False por defecto. Si pones True, los efectos serán duraderos en el dataframe, de lo contrario solo será una vista, pero cuando corres de nuevo el dataframe, no se mantienen los efectos. Eso pasa al ordenar por ejemplo, después de ordenar y ver los resultados, no se mantienen al ejecutar al ver de nuevo el contenido del dataframe.

Los métodos para lograr información general sobre dataframes, se aplican a todo el dataframe de la siguiente manera: `df.metodo()` o a una o varias columnas: `df.columna.metodo()` .

Veamos los métodos más comunes:

mean() aplica solo para columnas numéricas:

df.mean() verás los promedios de aquellas columnas numéricas solamente.

df.salarios.mean() te dará el promedio de la columna salarios.

Lo mismo aplica para las funciones siguientes:

min() solo columna numérica

max() solo columna numérica

count() numérica y de otro tipo

sum() solo columna numérica

Ejemplo sumando las columnas:

dataframe.sum()

Out[21]:

col1 0.395993

col2 -3.059013

col3 1.153082

col4 -1.447322

dtype: float64

Donde col1 es la columna 1, col2 es la columna 2...

In [22]:

Si queremos sumar filas

dataframe.sum(axis=1), axis=1 nos dice que sumemos en vez los valores de columnas.

Out[22]:

a -1.889150

b -1.616652

c 2.847987

d -1.352268

e -0.947177

dtype: float64

corr() calcula la correlacion , solo columna numérica

corrwith() , solo columna numérica

Lea el siguiente link sobre las funciones de agregación(parecidas a las estudiadas en los estudios de base de datos).

<http://www.shanelynn.ie/summarising-aggregation-and-grouping-data-in-python-pandas/>

EL método: **describe** , es muy interesante, veamos:

Si aplicamos el método a todo el dataframe de este modo:

df.describe(include=['object']), te arrojará valores para columnas no numéricas.

	regiment	company	name
count	12	12	12
unique	3	2	11
top	Nighthawks	1st	Ali
freq	4	6	2

Si aplicamos el método a todo el dataframe sin parámetros dentro del (), de este modo: df.describe(), genera estadísticos (min,max,count,std,mean,percentiles) de aquellas columnas numéricas.

	preTestScore	postTestScore
count	12.000000	12.000000
mean	11.083333	62.333333

```
std          12.324833      21.376850
min          2.000000      25.000000
25%          2.750000      57.000000
50%          3.500000      62.000000
75%          24.000000      70.000000
max          31.000000      94.000000
Ojo, excluye los valores nulos o NaN.
```

Si queremos que aparezcan los cálculos para todas las columnas:

```
df.describe(include='all')
ejecutará los estadísticos en las columnas que aplique, y donde no aplique pondrá
el valor Nan (nulo).
```

Podemos indicar a describe los percentiles que deseamos:

```
df.describe(percentiles=[0.25, 0.5, 0.75])
```

Otro método que nos muestra información sobre la tabla de datos es **info**:

```
df.info()
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId 891 non-null int64
Survived 891 non-null int64
Pclass 891 non-null int64
Name 891 non-null object
Sex 891 non-null object
Age 714 non-null float64
SibSp 891 non-null int64
Parch 891 non-null int64
Ticket 891 non-null object
Fare 891 non-null float64
Cabin 204 non-null object
Embarked 889 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.6+ KB
```

Los métodos head y tail nos permiten ver los primeros o últimos registros:

```
df.head(5) # las primeras 5 filas
df.tail(5) # las últimas 5 filas.
```

df.index sin () te indica los valores que se deben usar en el range (filas) cuando uses un for para leer el contenido de un dataframe.

Por ejemplo:

```
df.index --> RangeIndex(start=0, stop=20, step=1) me dice en qué índice inicia y
en cuál termina el rango del dataframe, por tanto el stop me dice la cantidad de
filas o rows de la tabla. Recuerda que al crear un dataframe sin definir todas
sus opciones, por default crea una columna de índice que es un autoincrementado.
```

Otras instrucciones de interés:

len(df.index) , sirve para saber la cantidad de filas o registros, no es lo mismo que la cantidad de elementos o celdas. Se usa el index porque esa columna siempre está llena, pero también puedes hacer len(df) sin más y te dará las rows, pero debemos tener en cuenta que pueden existir datos que tienen columnas con

cantidades asimétricas, o sea, no tienen la misma cantidad de filas y se llenan con Nan y eso puede alterar la cifra real.

```
len(df.columns) , cantidad de columnas
```

df.shape[0] , cantidad de registros de la columna 0 o columna de índices creada por dataframe, que suele ser la totalidad real de las filas de la data, dado que la columna índice no tiene nulos.

```
data.shape[0] # devuelve número de filas.
```

```
data.shape[1] #devuelve número de columnas sin incluir la columna índice creada por python al crear el dataframe.
```

```
data.shape # ojo, sin paréntesis ni corchetes, devuelve una tupla que contiene número de filas y número de columnas.
```

```
df.dtypes sin paréntesis ni corchetes, presenta el tipo de datos de cada columna.
```

Si deseamos medir la velocidad de una instrucción o bloque de instrucciones:

```
import time
```

```
ini_t1=time.time()
```

```
df.query("preTestScore > 4")
```

```
fin_t1=time.time()
```

```
print(fin_t1-ini_t1 )
```

```
ini_t2=time.time()
```

```
df[(df.preTestScore > 4)]
```

```
fin_t2=time.time()
```

```
print(fin_t2-ini_t2 )
```


5. Aprender conectar con Firebird y convertir los datos en Dataframe.

Veamos el siguiente script: pandas7.py

```
import fdb # tenemos que importar el módulo fdb que permite conectar con
Firebird, esto ya se explicó en los documentos anteriores durante la
instalación de las herramientas de software para el curso.

import numpy as np

import pandas as pd

# A continuación procedemos a escribir el código para establecer la
conexión desde python con la base de datos EMPLOYEE.FDB mediante el método
connect del módulo fdb . Para nosotros debe ser familiar los valores que
agregamos en la conexión. La variable con contendrá la conexión a la base
de datos.

con = fdb.connect(
    dsn='localhost:C:/data/EMPLOYEE.FDB',
    user='SYSDBA', password='udelas'
)

cursor = con.cursor() # el método cursor() de una conexión fdb-en nuestro
caso la variable con - permite crear un contenedor de métodos que
permitirán ejecutar queries a la base de datos.

cursor.execute('SELECT * from COUNTRY') #mediante el método execute y las
instrucciones sql, lanzamos la acción a la base de datos.

rows = cursor.fetchall()# dado que es un selec o consulta, mandamos la
instrucción que descargue los resultados de la consulta (una colección de
registros) y los guardamos en una variable llamada rows.

#Vamos a imprimir cada fila del grupo de registros guardado en la variable
rows.

for row in rows :
    print (row[0])
```

```
print ("registros: ", len(rows)) # imprime la cantidad de registros que
hemos recibido de la consulta.

# Al terminar de usar el cursor, y no volverlo a utilizar más, hay que
cerrarlo, lo mismo ocurre para la conexión abierta a la base de datos.

cursor.close

con.close

Recuerden que tenemos la base de datos EMPLOYEE.FDB en la carpeta C:\data.

http://codehero.co/python-desde-cero-bases-de-datos/
```

6. Aprender construir y ejecutar queries desde Pandas en Python a la base de datos Firebird.

Usaremos el módulo fdb para conectar y operar con las bases de datos de Firebird.

pandas8.py

```
import fdb

import numpy as np

import pandas as pd

pd.set_option('expand_frame_repr', False)
pd.set_option("display.max_rows",None)

con = fdb.connect(
    dsn='localhost:C:/data/FILM.FDB',
    user='SYSDBA', password='udelas'
)
```

Arriba, creamos nuestro objeto conexión mediante la instrucción **fdb.connect**, cuyos parámetros apuntan a la base de datos FILM.FDB y el usuario y clave que permite acceder. El resultado se asigna a la variable llamada **con** (puede ser otro nombre cualquiera) la cual contendrá todos los métodos que permiten posteriormente seleccionar, insertar, actualizar, eliminar y otras muchas operaciones, por eso se le dice que es un objeto, pues contiene muchos métodos (anteriormente dimos el curso de Clases y Objetos).

Creado el objeto conexión, vamos a realizar unas pruebas con alguno de sus métodos que nos permitirán conocer información sobre la base de datos.

```
print("Version de Firebird: ",con.firebird_version)
print("-----")
print("Versión (corta): ", con.version)
print("-----")
print("Nombre de la computadora en la que instaló firebird: ",con.site_name)
print("-----")
print("Nombre de la base de datos y su ubicación: ", con.database_name)
print("-----")
print("Ods: ",con.ods_version)
print("-----")
print("Nombres de los usuarios conectados y cantidad de conexiones:
",con.db_info(fdb.isc_info_user_names))
print("-----")
```

Ejecute todo el script (pandas9.py).

No se olvide de cerrar la conexión:

con.close() al final del programa.

Continuamos con las operaciones de buscar datos, insertar, actualizar y eliminar, justamente parecido a lo que hicimos previamente con el SQL Manager Lite al operar con las bases de datos directamente.

Empecemos con Selec:

pandas9.py

```
import fdb

import numpy as np

import pandas as pd

pd.set_option('expand_frame_repr', False)

pd.set_option("display.max_rows",None)


con = fdb.connect(

    dsn='localhost:C:/data/FILM.FDB',

    user='SYSDBA', password='udelas'

)


cursor = con.cursor()

query="""SELECT CATEGORY_ID,NAME

        FROM CATEGORY ;"""

resultados=cursor.execute(query).fetchall() # execute ejecuta la acción y con fetchall se
pide que se envíen los resultados que se guardan en la variable resultados (escriba en la
consola resultados y enter para ver el contenido.

print("Impresión de Resultados en formato crudo, tal como lo recibimos")

print(resultados)

print("-----")

data=pd.DataFrame(resultados,columns = ['CATEGORY_ID', 'NAME']) # convertimos los
resultados en un formato dataframe de panda para que se vea mejor

print("Impresión de Resultados convertido a formato Dataframe")

print(data) # se manda imprimir

print ("Ok")

cursor.close

con.close
```

Observamos instrucciones nuevas en el código.

El objeto de conexión **con** contiene muchos métodos, uno de ellos es cursor(), este método permite ejecutar acciones contra la base de datos que maneja la conexión- en nuestro caso FILM.FDB- como : select, update, insert y delete. En programación primero se crean los

objetos y luego se usan. El cursor se asigna a una variable, que la hemos llamado también cursor (pero pudo ser cualquier otro nombre) y que es un objeto que contiene otros métodos diferentes.

Mediante el método `execute` del cursor, podemos traer datos que consultamos pero en dos tiempos, uno mediante `execute` y luego mediante `fetchall`.

El método `execute` lanza la instrucción `query` o `sql`, en este caso un `select`, que previamente hemos escrito y asignado a una variable:

```
query="""SELECT CATEGORY_ID,NAME  
  
FROM CATEGORY ;"""
```

Fijémonos que la sentencia `string` la tenemos contenida dentro de un `docstring`, o sea, entre comillas triples al inicio y final para que permita que podamos escribir sin tener que agregar saltos de línea, es como si escribimos en un notepad, esto lo hemos aprendido en el curso uno. La cadena o `string` del `select` la hemos asignado a la variable llamada **query** (pudo ser cualquier otro nombre), y posteriormente la incluimos en el método **execute** del cursor:

```
resultados=cursor.execute(query).fetchall()
```

Arriba el cursor primero ejecuta la instrucción `sql` (un `select`) contenida en la variable llamada `query` contra la tabla de la base de datos y luego ordena que los resultados de dicha consulta los carguemos en la variable llamada `resultados` (pudo ser otro nombre cualquiera).

El resto de la codificación es fácil de entender. Convertimos los resultados en un `dataframe` de `panda` y luego los visualizamos, su aspecto es mucho más entendible, y nos permite aplicar los métodos del módulo `pandas` que previamente hemos estudiado.

Seguimos con la acción **Insert** o insertar un nuevo registro a la tabla Category de la base de datos FILM.FDB.

pandas10.py

```
import fdb

import numpy as np

import pandas as pd

pd.set_option('expand_frame_repr', False)
pd.set_option("display.max_rows",None)


con = fdb.connect(
    dsn='localhost:C:/data/FILM.FDB',
    user='SYSDBA', password='udelas'
)

cursor = con.cursor()

query="""INSERT INTO CATEGORY

(CATEGORY_ID,NAME)

VALUES (17, 'Aventuras') """

cursor.execute(query) #los queries como insert, delete o update, son de acción, no
devuelven ningún grupo de registros o resultados

# por eso no asignamos el resultado de la acción a ninguna variable como es el caso de un
query del tipo select

con.commit() #para que sea efectiva la acción de arriba, se confirma con la instrucción
commit desde el objeto conexión.

# Ahora vamos a visualizar los datos con un selec para verificar la inserción

query="""SELECT CATEGORY_ID,NAME

FROM CATEGORY ;"""

resultados=cursor.execute(query).fetchall()

data=pd.DataFrame(resultados,columns = ['CATEGORY_ID', 'NAME'])

print(data)

print ("Ok")

cursor.close

con.close
```

Tenemos la primera diferencia en la construcción de la instrucción sql que asignamos a la variable llamada query. Utilizamos la instrucción Insert que ya hemos estudiado en los talleres de las bases de datos.

Luego tenemos la instrucción: **cursor.execute(query)** sin el método fetchall. Esto es porque estamos ordenando insertar, no estamos pidiendo que se nos envíe los resultados de una consulta (como al usar un select) que implica una cantidad de registros que deseamos ver.

Pero no es suficiente execute, es necesario confirmar y por tanto ordenar definitivamente la orden de ejecutar, esto es con la instrucción:

con.commit()

la instrucción es necesaria después de cada execute del tipo insert, update y delete. Las del tipo select requieren de la instrucción posterior fetchall. Tenga mucho esto en cuenta, de lo contrario sentirá que no se reflejan los resultados esperados.

Después de ejecutar el Insert, ejecutamos la acción de un Select para ver todos los registros de la tabla Category y observar si aparece el nuevo registro al final de la tabla.

```
query="""SELECT CATEGORY_ID,NAME
        FROM CATEGORY ;"""
resultados=cursor.execute(query).fetchall()
data=pd.DataFrame(resultados,columns = ['CATEGORY_ID', 'NAME'])
print(data)
```

Y para terminar debemos cerrar el cursor y la conexión , esto es para que no quede en memoria y sostenida en el software de la base de datos a pesar de finalizar el programa python. Recuerde que en este escenario estamos trabajando en dos componentes, uno es el de la base de datos que sigue viva con o sin python, y el programa que hacemos en python.

cursor.close

con.close

Continuamos con **Update y Delete**, que son explicadas de la misma manera que Insert, y cuya diferencia es el query o sql usado para lanzar contra la base de datos. Los programas fuentes de python son:

pandas11.py para el caso de Update

pandas12.py para el caso de Delete.

Referencias de estudio:

<http://pandas.pydata.org/pandas-docs/stable/indexing.html>

<http://www.shanelynn.ie/summarising-aggregation-and-grouping-data-in-python-pandas/>