

USM Numérica

Librería Pandas

Objetivos

1. Conocer los principales comandos de la librería pandas
2. Utilizar pandas para limpieza y manipulación de datos.



Sobre el autor

Sebastián Flores

ICM UTFSM

sebastian.flores@usm.cl

Sobre la presentación

Contenido creada en ipython notebook (jupyter)

Versión en Slides gracias a RISE de Damián Avila

Software:

- python 2.7 o python 3.1
- pandas 0.16.1

Opcional:

- numpy 1.9.2
- matplotlib 1.3.1

Aprender haciendo

Consideraremos el siguiente archivo `data.csv` que contiene datos incompletos:

In []:

```
%%bash  
cat data/data.csv
```

1.- ¿Porqué utilizar pandas?

Razón oficial: Porque en numpy no es posible mezclar tipos de datos, lo cual complica cargar, usar, limpiar y guardar datos mixtos.

Razón personal: Porque habían cosas que en R eran más fáciles pero no pythonísticas. La librería pandas es un excelente compromiso.

In []:

```
import numpy as np
df = np.loadtxt("data/data.csv", delimiter=";", dtype=str)
print( df )
```

```
In [ ]: import pandas as pd
df = pd.read_csv("data/data.csv", sep=";")
print( df )
#df
```

```
In [ ]: inch2m = 0.0254
        feet2m = 0.3048
        df.diametro = df.diametro * inch2m
        df.altura = df.altura * feet2m
        df.volumen = df.volumen * (feet2m**3)
        df.tipo_de_arbol = "Cherry Tree"
        df
```



```
In [ ]: print( df.columns )
```

```
In [ ]: print( df.index )
```

```
In [ ]: print( df["altura"]*2 )
```

```
In [ ]: print( df["diаметro"]**2 * df["altura"] / df.volumen )
```

2. Lo básico de pandas

- Pandas imita los dataframes de R, pero en python. Todo lo que no tiene sentido es porque se parece demasiado a R.
- Pandas permite tener datos como en tablas de excel: datos en una columna pueden ser mixtos.
- La idea central es que la indexación es "a medida": las columnas y las filas (index) pueden ser enteros o floats, pero también pueden ser strings. Depende de lo que tenga sentido.
- Los elementos básicos de pandas son:
 - **Series**: Conjunto de valores con indexación variable.
 - **DataFrames**: Conjunto de Series.

2.1 Series

Una serie es un conveniente conjunto de datos, como una columna de datos de excel, pero con indexación más genérica.

```
pd.Series(self, data=None, index=None, dtype=None, name=None, copy=False,
fastpath=False)
```

```
In [ ]: import pandas as pd
s1 = pd.Series([False, 1, 2., "3", 4 + 0j])
print( s1 )
```

```
In [ ]: # Casting a otros tipos
print( list(s1) )
print( set(s1) )
print( np.array(s1) )
```

```
In [ ]: # Ejemplo de operatoria  
s0 = pd.Series(range(6), index=range(6))  
s1 = pd.Series([1,2,3], index=[1,2,3])  
s2 = pd.Series([4,5,6], index=[4,5,6])  
s3 = pd.Series([10,10,10], index=[1,4,6])
```

```
In [ ]: print( s0 )
```

```
In [ ]: print( s0 + s1 )
```

```
In [ ]: print( s0 + s1 + s2 )
```

```
In [ ]: print( s0.add(s1, fill_value=0) )
```

2.2 DataFrames

Un Dataframe es una colección de Series con una indexación común. Como una planilla de excel.

```
pd.DataFrame(self, data=None, index=None,  
             columns=None, dtype=None, copy=False)
```

In []:

```
# dict  
df = pd.DataFrame({"col1": [1, 2, 3, 4],  
                  "col2": [1., 2., 3., 4.],  
                  "col3": ["uno", "dos", "tres", "cuatro"]})  
  
df
```

3.1 Obteniendo datos

1. **Archivo csv**
2. **Archivo json**
3. **Archivo de excel:** convertir a csv cuidadosamente (elegir un separador apropiado, no comentar strings).


```
In [ ]: # csv
df = pd.read_csv("data/data.csv", sep=";")
df
```

```
In [ ]: df = pd.read_json("data/data.json")
df
```

4.- Inspeccionando datos

1. Accesando las columnas
2. shape
3. head, tail, describe
4. histogram
5. pd.scatter_matrix
6. count_values

```
In [ ]: df = pd.read_csv("data/data.csv", sep=";")  
df.columns
```

```
In [ ]: df['altura']
```

```
In [ ]: df.shape
```

```
In [ ]: df.head()
```

```
In [ ]: df.tail()
```

```
In [ ]: df.describe()
```

```
In [ ]: df.describe(include="all")
```


In []:

```
from matplotlib import pyplot as plt
df.hist(figsize=(10,10), layout=(3,1))
#df.hist(figsize=(8,8), layout=(3,1), by="tipo_de_arbol")
plt.show()
```

```
In [ ]: from matplotlib import pyplot as plt
pd.scatter_matrix(df, figsize=(10,10), range_padding=0.2)
plt.show()
```

```
In [ ]: df.tipo_de_arbol.value_counts()
```

5.- Manipulando DataFrames

1. Agregando columnas
2. Borrando columnas
3. Agregando filas
4. Borrando filas
5. Mask
6. Grouping
7. Imputación de datos
8. Apply
9. Merge (a la SQL)
10. Accesamiento

5.1 Agregando columnas

```
In [ ]: df = pd.read_csv("data/data.csv", sep=";")  
df["radio"] = .5 * df.diametro  
df
```

```
In [ ]: df.area = np.pi * df.radio **2  
df.columns
```

5.2 Renombrando columnas

```
In [ ]: df = pd.read_csv("data/data.csv", sep=";")  
        print( df.columns )  
        df.columns = ["RaDiO", "AlTuRa", "VoLuMeN", "TiPo_De_ArBoL"]
```

```
In [ ]: print( df.columns )
```

```
In [ ]: df.columns = [col.lower() for col in df.columns]  
        print( df.columns )
```

5.3 Borrando columnas

```
In [ ]: df = pd.read_csv("data/data.csv", sep=";")  
        print( df.columns )
```

```
In [ ]: df = df[["tipo_de_arbol", "volumen", "diametro"]]  
        df
```

```
In [ ]: df = df.drop("tipo_de_arbol", axis=1)  
        df
```

```
In [ ]: df.drop("diametro", axis=1, inplace=True)  
        df
```


5.4 Agregando filas (indices)

```
In [ ]: df = pd.read_csv("data/data.csv", sep=";")  
        print( df.index )  
        df
```

```
In [ ]: df = df.reindex( range(20) )  
        df
```

```
In [ ]: # Usando loc para acceder con notación de índices tradicional  
        df.loc[20, :] = [10, 20, 30, "CT"]  
        df
```

5.5 Renombrando filas (índices)

```
In [ ]: df = pd.read_csv("data/data.csv", sep=";")  
print df.index
```

```
In [ ]: df.index = df.index + 10  
print df.index
```

```
In [ ]: df.index = ["i_%d"%idx for idx in df.index]  
print df.index
```

5.6 Borrando indices

```
In [ ]: print df.index  
df
```

```
In [ ]: df = df.drop(["i_11","i_13","i_19"], axis=0)  
print( df.index )  
df
```

```
In [ ]: df.drop(["i_24", "i_25", "i_26"], axis=0, inplace=True)
df
```

In []:

```
df = df[-5:]  
df
```

Observación

```
# seleccionar la columna col  
# regresa una serie  
df[col]  
  
# seleccionar las columnas col1, col2, ..., coln  
# regresa dataframe  
df[[col1,col2,..., coln]]  
  
# selecciona solo el indice inicio  
# regresa un dataframe  
df[inicio:(inicio+1)]  
  
# selecciona los indices en notacion  
#regresa un dataframe  
df[inicio:fin:salto]  
  
# seleccion mixta  
# regresa un dataframe  
df.loc[inicio:fin:salto, col1:col2]
```

5.7 Masking

```
In [ ]: df = pd.read_csv("data/data.csv", sep=";")
        vol_mean = df.volumen.mean()
        vol_std = df.volumen.std()
```

```
In [ ]: mask_1 = df.altura < 80
        mask_2 = df.volumen <= vol_mean + vol_std
        df1 = df[ mask_1 & mask_2 ]
        df1
```

```
In [ ]: # Si se hace dinamicamente, utilizar suficientes parentesis
        #df2 = df[ ((vol_mean - vol_std) <= df.volumen) & (df.volumen <= (vol_mean + vol_std) ) ]
        df2 = df[ (df.volumen >=(vol_mean - vol_std)) & (df.volumen <= (vol_mean + vol_std) ) ]
        df2
```

```
In [ ]: # A veces para simplificar numpy ayuda  
mask_1 = df.volumen >= (vol_mean - vol_std)  
mask_2 = df.volumen <= (vol_mean + vol_std)  
mask = np.logical_and(mask_1, mask_2)  
df3 = df[np.logical_not(mask)]  
df3
```


5.8.- Grouping

```
In [ ]: df = pd.read_csv("data/data.csv", sep=";")  
df.columns
```

```
In [ ]: g = df.groupby("tipo_de_arbol")  
print( g )
```

```
In [ ]: print( g.count() )
```

```
In [ ]: print( g.sum() ) # .mean(), .std()
```

```
In [ ]: # Ejemplo real  
df[["tipo_de_arbol", "diametro", "altura"]].groupby("tipo_de_arbol").mean()
```

5.9.- Imputación de datos

```
In [ ]: # Antes de imputar datos, siempre explorar  
df.describe(include="all")
```

In []:

```
# Imputación manual de datos (incorrecto)  
df["tipo_de_arbol"][df.tipo_de_arbol=="Cherrie Tree"] = "Cherry Tree"  
df
```

```
In [ ]: # Imputación manual de datos  
df = pd.read_csv("data/data.csv", sep=";")  
index_mask = (df.tipo_de_arbol=="Cherrie Tree")  
df.loc[index_mask, "tipo_de_arbol"] = "Cherry Tree" # .loc es esencial  
df
```

```
In [ ]: # Imputación de datos: llenar NaNs con promedio  
df = pd.read_csv("data/data.csv", sep=";")  
df1 = df.fillna(df.mean())  
df1
```

```
In [ ]: # Imputación de datos: llenar NaNs con valor  
df2 = df.fillna(0)  
df2
```

```
In [ ]: # Imputación de datos: desechar filas con NaN  
df3 = df.dropna()  
df3
```


5.10 Apply

```
In [ ]: df = pd.read_csv("data/data.csv", sep=";")  
df1 = df.diametro.apply(lambda x: x*2)  
df1
```

```
In [ ]: # Aplicación incorrecta  
df2 = df["tipo_de_arbol"].apply(str.upper) # Error  
df2
```

```
In [ ]: # Aplicación correcta  
df2 = df["tipo_de_arbol"].apply(lambda s: str(s).upper() )  
df2
```

In []:

```
# Error (o no?)  
df3 = df.apply(lambda x: x*2)  
df3
```

Atajo

Para usar las operaciones de string en una columna de strings, es posible utilizar la siguiente notación para ahorrar espacio.

```
In [ ]: df.tipo_de_arbol.str.upper()
```

```
In [ ]: df.tipo_de_arbol.str.len()
```

```
In [ ]: df.tipo_de_arbol.str[3:-3]
```

5.11 Merge

```
In [ ]: df1 = pd.read_csv("data/data.csv", sep=";")  
df1
```

```
In [ ]: df2 = pd.DataFrame(data={"tipo_de_arbol":["Cherry Tree", "Apple Tree", "Pear Tree"],
                                "fruto":["guinda", "manzana", "pera"],
                                "precio_pesos_por_kg":[500, 2000, np.nan]})

df2
```


In []:

```
df3 = df1.merge(df2, how="left", on="tipo_de_arbol")  
df3
```

In []:

```
df3 = df1.merge(df2, how="right", on="tipo_de_arbol")  
df3
```

In []:

```
df3 = df1.merge(df2, how="inner", on="tipo_de_arbol")  
df3
```

In []:

```
df3 = df1.merge(df2, how="outer", on="tipo_de_arbol")  
df3
```

Guardando datos

1. **csv**
2. **json**
3. **excel**

Lo más importante es tener cuidado de cómo se guardan los nombres de las columnas (header), y el índice (index).

Depende de la utilización, pero mi recomendación es guardar el header explícitamente y guardar el index como una columna.

In []:

```
# guardar un csv
df = pd.read_csv("data/data.csv", sep=";")
df = df[df.tipo_de_arbol=="Cherry Tree"]
df.to_csv("data/output.csv", sep="|", index=True) # header=True by default
df
```

```
In [ ]: # Leer el csv anterior  
df2 = pd.read_csv("data/output.csv", sep="|", index_col=0) # get index from first column  
df2
```

In []:

```
%%bash  
cat data/output.csv
```



```
In [ ]: # guardar un json  
df = pd.read_csv("data/data.csv", sep=";")  
df = df[df.tipo_de_arbol=="Cherry Tree"]  
df.to_json("data/output.json")  
df
```

```
In [ ]: # Leyendo el json anterior  
df2 = pd.read_json("data/output.json")  
df2
```

In []:

```
%%bash  
cat data/output.json
```

Desafío para la casa

Descargar algún archivo de interés:

- Abrir el archivo.
- Explorar los datos
- Visualizar los datos
- Completar los datos incompletos
- Guardar el archivo

¡Gracias por la atención!