Python Cheat Sheet 3

Pandas

Series: estructuras en una dimension

Crear series

`serie = pd.Series()` crear serie vacía
`serie = pd.Series(array)` crear serie a partir de un array con el indice por defecto
`serie = pd.Series(array, index = [‘a’, ‘b’, ‘c’...])` crear una serie con indice definida; debe ser lista de la misma longitude del array
`serie = pd.Series(lista)` crear una seria a partir de una lista
`serie = pd.Series(número, indice)` crear una serie a partir de un escalar con la longitude igual al número de indices
`serie = pd.Series(diccionario)` crear una serie a partir de un diccionario

Acceder a informacion de una serie

`serie.index` devuelve los indices
`serie.values` devuelve los valores
`serie.shape` devuelve la forma (no. filas)
`serie.size` devuelve el tamaño
`serie.dtypes` devuelve el tipo de dato

`serie[i]` devuelve el valor del elemento en indice i
`serie[[i,j]]` devuelve el valor de los dos elementos
`serie[i:m]` devuelve el valor de un rango

`serie[“etiqueta”]` devuelve el valor de los elementos en indices i y j

Operaciones con series

`serie1 +-*/ serie2` suma/resta/multiplica/divide las filas con indices comunes entre las dos series
`serie1.add(serie2, fill_value = número)` suma las filas con indices comunes, y suma el fill value a los valores sin indice comun
`serie1.sub(serie2, fill_value = número)` restan las filas de la seria2 de la serie1 cuando tienen indices comunes, y resta el fill value de las otras indices de serie1
`serie1.mul(serie2, fill_value = número)` multiplica las filas con indices comunes y multiplica el fill value con las otras *usar 1 para conservar el valor*
`serie1.mul(serie2, fill_value = número)` divida las filas de la serie1 entre las de la serie2 cuando tienen indices comunes, y divide las otras por el fill value
`serie1.mod(serie2, fill_value = número)` devuelve el modulo (division sin resta)
`serie1.pow(serie2, fill_value = número)` calcula el exponencial
`serie1.ge(serie2)` compara si serie1 es mayor que serie2 y devuelve True o False
`serie1.le(serie2)` compara si serie1 es menor que serie2 y devuelve True o False

Filtrado booleanos

`serie < > >= <= == valor` devuelve True o False segun si cada condición cumple la condición
`serie1[serie1 < > >= <= == valor]` devuelve solo los valores que cumplen la condición
`np.nan` crear valor nulo (NaN)
`serie.isnull()` devuelve True o False segun si los valores existen o son nulos (“” no cuenta como nulo)
`serie.notnull()` devuelve True o False segun si los valores existen o son nulos (“” no cuenta como nulo)

DataFrames

Crear DataFrames

`df = pd.DataFrame(data, index, columns)`
data: NumPy Array, diccionario, lista de diccionarios
index: indice que por defecto se asigna como 0-(n-1), n siendo el número de filas;
`index = [lista]` para asignar “etiquetas” (nombres de filas)
column: nombre de las columnas; por defecto 0-(n-1);
`columns =[lista]` para poner mas nombres

`df = pd.DataFrame(array)` crear un dataframe a partir de un array con indices y columnas por defecto
`df = pd.DataFrame(diccionario)` crear un dataframe a partir de un diccionario – los keys son los nombres de las columnas

Acceder a informacion de un DataFrame

`df.loc[“etiqueta_fila”, “etiqueta_columna”]` devuelve el contenido de un campo en una columna de una fila
`df.loc[“etiqueta_fila”,:]` devuelve los valores de todas las columnas de una fila
`df.loc[:,“etiqueta_columna”]` devuelve los valores de todas las filas de una columna
`df.iloc[indice_fila, indice_columna]` devuelve el contenido de un campo en una columna de una fila
`df.iloc[indice_fila, :]` devuelve los valores de todas las columnas de una fila
`df.iloc[:,indice_columna]` devuelve el contenido de un campo en una columna de una fila
`df.loc[[lista_etiquetas_filas], [lista_etiquetas_columnas]]` devuelve el contenido de varias filas / varias columnas
`df.loc[[lista_indices_filas], [lista_indices_columnas]]` devuelve el contenido de varias filas / varias columnas
- se puede usar los indices/rangos de las listas [start:stop:step] dentro de los loc/iloc
`df.loc[df.etiqueta > x]` seleccionar datos basado en una condición usando operadores comparativos
`df.loc[(df.etiqueta > x) & (df.etiqueta == y)]` seleccionar datos que tienen que cumplir las dos condiciónes (and)
`df.loc[(df.etiqueta > x) | (df.etiqueta == y)]` seleccionar datos que tienen que deben cumplir una de las dos condiciones (or)
`df.iloc[list(df.etiqueta > x), :]` iloc no acepta una Serie booleana; hay que convertirla en lista
`variable_df.head(n)` devuelve las n primeras filas del df, o 5 por defecto

Crear columnas

`df[“nueva_columna”] = (df[“etiqueta_columna”] + x)` crea una nueva columna basada en otra
`df = df.assign(nueva_columna= df[“etiqueta_columna] + x)` crea una nueva basada en otra
`df = df.assign(nueva_columna= [lista_valores])` crea una nueva columna de una lista de valores *tiene que ser de la misma longitud como el número de filas del dataframe*
`df.insert(indice_nueva_columna, “nombre_columna”, valores)` crea una nueva columna en la indice indicada
`allow_duplicates = True` parametro cuando queremos permitir columnas duplicadas (por defecto es False)

Eliminar columnas

`df = df.drop(columns = [“column1”, “column2”])` eliminar columnas

DataFrames: carga de datos

Carga de datos

`df = pd.read_csv(“ruta/nombre_archivo.csv”)` crear un dataframe de un archivo de Comma Separated Values
`df = pd.read_csv(“ruta/nombre_archivo”, sep= “;”)` crear un dataframe de un csv si el separador es ;
`df = pd.read_csv(“ruta/nombre_archivo”, index_col= 0)` crear un dataframe de un csv si el archivo ya tiene una columna indice

`df = pd.read_excel(“ruta/nombre_archivo.xlsx”)` crear un dataframe de un archivo de Excel
- si sale “ImportError:... openpyxl...”, en el terminal:
pip3 install openpyxl o **pip install openpyxl**

`df = pd.read_json(“ruta/nombre_archivo.json”)` crear un dataframe de un archivo de JavaScript Object Notation (formato crudo)
`df = df[‘data’].apply(pd.Series)` convertir el dataframe de json en un formato legible

`df = pd.read_clipboard(sep=‘\t’)` crear un dataframe de datos en forma de dataframe en el clipboard; el separador podria ser \n ; , etc.

Pickle: modulo que serializa objetos (convertir objetos complejos en una serie de bytes, en este caso en formato binario) para guardarlos en un archivo
`with open(‘ruta/nombre_archivo.pkl’, ‘wb’) as f: pickle.dump(df,f)` pone los datos de un dataframe en el archivo pkl

`pd.read_pickle(‘ruta/nombre_archivo.csv’).head(n)` leer n filas y 5 columnas del archivo pickle

`pd.read_parquet(‘ruta/nombre_archivo.parquet’)` leer un archivo parquet

`pd.read_sas(‘ruta/nombre_archivo.sas7bdat’, format = ‘sas7bdat’)` leer un archivo SAS de formato SAS7BDAT

`pd.read_spss(‘ruta/nombre_archivo.sav’` leer un archivo SAS de formato SAS7BDAT

Guardado de datos

`df.to_csv(‘ruta/nombre_archivo.csv’)` guardar dataframe como archivo csv
`df.to_excel(‘ruta/nombre_archivo.xlsx’)` guardar dataframe como archivo de Excel
`df.to_json(‘ruta/nombre_archivo.json’)` guardar dataframe como archivo de JSON
`df.to_parquet(‘ruta/nombre_archivo.parquet’)` guardar dataframe como archivo de parquet
`df.to_pickle(‘ruta/nombre_archivo.pkl’)` guardar dataframe como archivo de pickle

Librería PyDataset

`pip install pydataset` o `pip3 install pydataset`
from pydataset import data
`data()` para ver los datasets listados en un dataframe por su id y título
`df = data(‘nombre_dataset’)` guardar un dataset en un dataframe

Metodos para explorar un dataframe

`df.head(n)` devuelve las primeras n lineas del dataframe, o por defecto 5
`df.tail(n)` devuelve las últimas n lineas del dataframe, o por defecto 5
`df.sample(n)` devuelve n filas aleatorias de nuestro dataframe, o uno por defecto

Metodos de DataFrames

Metodos para explorar un dataframe

`df.shape` devuelve el número de filas y columnas
`df.dtypes` devuelve el tipo de datos que hay en cada columna
`df.columns` devuelve los nombres de las columnas
`df.describe` devuelve un dataframe con un resumen de los principales estadísticos (media, mediana, desviación estándar etc.) de las columnas numéricas
`df.describe(include = object)` devuelve un dataframe con un resumen de los principales estadísticosde las columnas con variables tipo string
`df.info()` devuelve un resumen sobre el no. de columnas, nombres de columnas, numero de valores no nulos y los tipos de datos de las columnas
`df[“nombre_columna”].unique()` o `df.nombre_columna.unique()` devuelve un array con los valores únicos de la columna
`df[“nombre_columna”].value_counts()` o `df.nombre_columna.value_counts()` devuelve una serie con el recuento de valores únicos en orden descendente
`df.isnull()` o `df.isna()` devuelve True o False según si cada valor es nulo o no
`df.isnull().sum()` o `df.isna().sum()` devuelve el número de valores nulos por columnas
`df.corr()` devuelve la correlación por pares de columnas, excluyendo valores NA/nulos
`df.set_index([“nombre_columna”], inplace = True)` establece el índice utilizando uno o mas columnas; puede sustituir o ampliar un índice existente
`inplace = True` los cambios sobreescriben sobre el df
*** cuando una columna se cambia a índice ya no es columna ***

`df.reset_index(inplace = True)` quitar una columna como indice para que vuelva a ser columna
`df.rename(columns = {“nombre_columna”: “nombre_nueva”}, inplace = True)` cambia los nombres de una o mas columnas
ejemplo de dict comprehension para crear diccionario sobre las columnas existentes de un dataframe:
`diccionario = {col : col.upper() for col in df.columns}`
`df.rename(columns = diccionario, inplace = True)` cambia los nombres de las columnas según el diccionario
`df.drop([“columna1”, “columna2”], axis = b)` eliminar una o mas columnas o filas segun lo que especificamos
`axis = 1` columnas
`axis = 0` filas
`df.rename(columns = diccionario, inplace = True)` cambia los nombres de las columnas según el diccionario
`df[“columna_nueva”] = pd.cut(x=df[“nombre_columna”], bins=[n,m,l..])` separa los elementos de un dataframe en diferentes intervalos (n-m, m-l, etc); con este sintaxis se crea una columna nueva que indica en cual intervalo cae el valor
`df.replace(to_replace = valor, value = valor_nuevo, inplace = True)` reemplaza cierto valor por otro que especificamos
`df[“nombre_columna”].replace(to_replace = valor, value = valor_nuevo, inplace = True)` reemplaza cierto valor en una columna por otro que especificamos
`df[“nombre_columna”] = df[“nombre_columna”] + x` reemplaza los valores de la columna por el valor + x (o otro valor que indicamos)

Filtrados de datos

`pd.options.display.max_columns = None` ejecutar antes del df.head() para poder ver todas las columnas

Filtrado por una columna con operadores de comparación

`variable_filtro = df[df[“nombre_columna”] == valor]` extrae las filas donde el valor de la columna igual al valor dado
*** se puede usar con cualquier operador de comparación ***

Filtrado por multiples columnas con operadores logicos

& **and**
| **or**
~ **not**

`variable_filtro = df[(df[“columna1”] == valor) & (df[“columna2”] == valor) & (df[“columna3”] > n valor)]` extrae las filas donde los valores de las columnas cumplan las condiciónes en parentesis

`variable_filtro = df[(df[“columna1”] == valor) | (df[“columna1”] == valor)` extrae las filas donde los valores de las columnas cumplan con una condición u otra

`variable_filtro = ~(df[df[“columna1”] == valor])` extrae las filas donde los valores de las columnas NO cumplan con la condición

Metodos de pandas de filtrar

`variable_filtro = df[df[“nombre_columna”].isin(iterable)]` extrae las filas cuyas valores de la columna nombrada están en el iterable (una lista, serie, dataframe o diccionario)

`variable_filtro = df[df[“nombre_columna”].str.contains (patron, regex = True)]` extrae las filas cuyas valores de la columna nombrada contenienen el patron de regex

`variable_filtro = df[df[“nombre_columna”].str.contains (“substring”, case = False, regex = False)]` extrae las filas cuyas valores de la columna nombrada contienen el substring, no siendo case sensitive

`variable_filtro = df[df[“nombre_columna”].str.contains (“substring”, case = False, regex = False)]` extrae las filas cuyas valores de la columna nombrada contienen el substring, no siendo case sensitive

`df[pd.notnull(df[“nombre_columna”])]` devuelve las filas que no tiene valores nulos en la columna especificada

Reemplazar valores basados en indices y condiciones:

```
indices_filtrados = df.index[df[“columna”] == “valor”]
for indice in indices_filtrados:
    df[“nombre_columna”].iloc[indice] = “valor_nuevo”
```

Reemplazar valores basados en metodos NumPy:

`df[“nueva_columna”] = np.where(df[“nombre_columna”] > n, “categoria_if_true”, “categoria_if_false”)` crea una nueva columna con los valores basados en una condición

`df[“nueva_columna”] = np.select(lista_de_condiciones, lista_de_opciones)` crea una nueva columna con los valores basados en multiples condiciones