Datos con Pandas

(Teórico)

tiimiit by

Introducción

Bienvenidos al módulo denominado, datos con Pandas. Los objetivos de aprendizaje estarán orientados a incorporar conocimientos vinculados a las funcionalidades principales de la librería de código abierto Pandas

Una vez finalizado este módulo serás capaz de:

- Obtener conocimientos básicos sobre Pandas.
- Realizar su instalación
- Acceder a datos y crear un dataframe.
- Conocer funcionalidades básicas como: creación y carga de datos. Creación desde escalar. Posicionamiento.
- Realizar operaciones esenciales como: Merge, grouping, Resharping, Plotting.

Tema 1. Introducción a Pandas y Dataframes

Objetivo

El objetivo de este tema es brindar una descripción conceptual de las variables de la librería Pandas.

Una vez finalizado este tema serás capaz de:

- Conocer Pandas y sus características principales
- Conocer el concepto de Dataframe
- Instalar la librería

¿Qué es Pandas?

Pandas es una biblioteca de software enfocada en la manipulación y el análisis de datos rápidos y fáciles en Python. En particular, ofrece estructuras de datos de alto nivel (como DataFrame y Series) y métodos de datos para manipular y visualizar tablas numéricas y datos de series temporales. Está construido sobre NumPy y está altamente optimizado para el rendimiento (alrededor de 15 veces más rápido), con rutas de código críticas escritas en Python o C. La estructura de datos ndarray y las capacidades de transmisión de NumPy se usan mucho.

El creador de Pandas, Wes McKinney, comenzó a desarrollar la biblioteca en 2008 durante su mandato en AQR, una empresa de gestión de inversiones cuantitativas. Estaba motivado por un conjunto distinto de requisitos de análisis de datos que no estaban bien abordados por ninguna herramienta única a su disposición en ese momento.

Características de pandas

La librería de código abierto pandas posee las siguientes particularidades:

- Estructuras de datos con ejes etiquetados que admiten la alineación automática o explícita de datos capaces de manejar datos de series temporales y no temporales.
- Capacidad para agregar y eliminar columnas sobre la marcha.
- Manejo flexible de datos faltantes.
- Combinación similar a SQL y otras operaciones relacionales.
- Herramientas para leer y escribir datos entre estructuras de datos en memoria y diferentes formatos de archivo (csv, xls, HDF5, bases de datos SQL).
- Remodelación y pivoteo de conjuntos de datos.
- Segmentación basada en etiquetas, indexación elegante y creación de subconjuntos de grandes conjuntos de datos.
- Agrupar por motor que permite operaciones de división, aplicación y combinación en conjuntos de datos.
- Funcionalidad de serie temporal: generación de rango de fechas y conversión de frecuencia, estadísticas de ventana móvil, regresiones lineales de ventana móvil, cambio de fecha y retraso.
- Indexación de ejes jerárquicos para trabajar con datos de alta dimensión en una estructura de datos de menor dimensión.

¿Qué es un Dataframe?

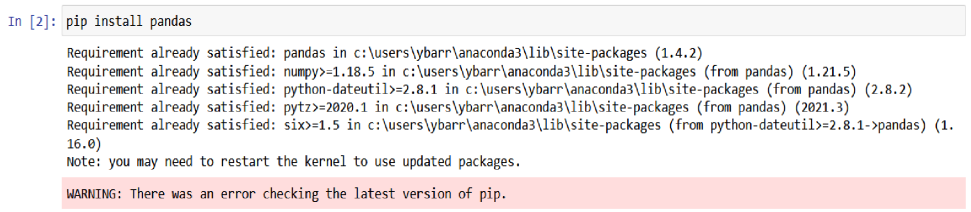
Los dataframes son estructuras contenedoras de datos, del tipo dataset, a través de los cuales se pueden manipular los datos. En distintos lenguajes los contenedores de datos pueden recibir distintos nombres como RecordSet, DataSet, DataTable, File, etc. En nuestro caso, y teniendo en cuenta que estamos utilizando Python como lenguaje de programación de modelos, utilizaremos el nombre de Dataframe al referirnos a los contenedores de datos debido a que la librería más utilizada en Python, Pandas, denomina dataframe a sus contenedores.

instalación de Pandas

Durante este recorrido utilizaremos Jupyter Notebook. El mismo lo hemos instalado en la Unidad “1-Entornos Virtuales”. Entonces, abrimos un notebook y ejecutamos el siguiente código:

```
In [ ]: pip install pandas
```

El sistema procederá a la instalación de pandas si no se encuentra instalado o informará lo siguiente para el caso que el mismo ya se encuentre instalado.



Una vez instalado Pandas, avanzaremos al siguiente tema.

Tema 2. Acceder a datos y crear un dataframe

Objetivo

El objetivo de este tema es brindar conocimientos que permitan crear y cargar datos en dataframes utilizando la librería de código abierto, Pandas.

Una vez finalizado este tema serás capaz de:

- Acceder a archivos CSV y Excel para cargar un data Frame
- Crear un dataframe desde un conjunto de datos.

Accediendo a Datos

En este módulo, accederemos a datos contenidos en archivos en formato Texto y en formato Excel. A continuación detallamos la sintaxis a utilizar para cada uno de los casos:

- **Lectura de un archivo CSV**

  Leer un archivo separado por comas es tan simple como llamar a la función read_csv. De forma predeterminada, la función read_csv

espera que el separador de columnas sea una coma, pero puede cambiarlo con el parámetro sep.

Sintaxis:

pd.read_csv(filepath, sep=, header=, names=, skiprows=, na_values= ...)

Archivo de ayuda: para obtener una explicación detallada de todos los parámetros, ejecute

pd.read_csv?

Entonces para leer un CSV y mostrar su contenido por pantalla, ejecutamos el siguiente código:

import pandas as pd

df=pd.read_csv("C:/Users/ybarr/Desktop/salud/vira.csv",encoding = "utf-8")

print(df)

Ahora bien, analicemos el código utilizado.

En la primera línea, importamos la librería pandas anteriormente instalada y la instanciamos en pd, para luego utilizar sus métodos.

En la línea 2, utilizamos el método read_csv, utilizando la ruta como parámetro y un segundo modificador para establecer la codificación de los caracteres, como “utf-8”. Asignamos el resultado de esa lectura, a una variable que por defecto será un dataframe.

Finalmente, en la tercera línea, solicitamos a Python, la impresión por pantalla de los datos existentes en el dataframe.

Entonces, al ejecutar el código, el notebook nos dará el siguiente resultado:

```
In [15]: import pandas as pd
df=pd.read_csv("C:/Users/ybarr/Desktop/salud/vira.csv",encoding = "utf-8")
print(df)
```

```
      departamento_id     departamento_nombre  provincia_id provincia_nombre  \
0                 760              SAN MIGUEL             6     Buenos Aires
1                 760              SAN MIGUEL             6     Buenos Aires
2                  14                 CAPITAL            14          Córdoba
3                 427              LA MATANZA             6     Buenos Aires
4                 427              LA MATANZA             6     Buenos Aires
...               ...                     ...           ...              ...
3458              274        FLORENCIO VARELA             6     Buenos Aires
3459                0  CIUDAD DE BUENOS AIRES             2             CABA
3460                0  CIUDAD DE BUENOS AIRES             2             CABA
3461                0  CIUDAD DE BUENOS AIRES             2             CABA
3462                0  CIUDAD DE BUENOS AIRES             2             CABA

       año  semanas_epidemiologicas                      evento_nombre  \
0     2018                       18  Bronquiolitis en menores de 2 años
1     2018                       18  Bronquiolitis en menores de 2 años
2     2018                       21     Enfermedad tipo influenza (ETI)
3     2018                       19                            Neumonía
4     2018                       19  Bronquiolitis en menores de 2 años
...    ...                      ...                                 ...
```

- **Leer un archivo Excel**

  Pandas le permite leer y escribir en archivos de Excel, por lo que puede leer fácilmente desde Excel, escribir su código en Python y luego volver a escribir en Excel, sin necesidad de VBA. La lectura de archivos de Excel requiere la biblioteca xlrd. El mismo podemos instalarlo a través de pip, ejecutando el siguiente comando:

  pip install xlrd

  La sintaxis de read_excel es:

  pd.read_excel('mi-excel.xlsx', 'hoja1')

  Entonces, para leer un archivo Excel, ejecutamos el siguiente código:

import pandas as pd
df=pd.read_excel("C:/Users/ybarr/Desktop/salud/vdz.xls")
print(df)

Ahora bien, analicemos el código utilizado.

En la primera línea, importamos la librería pandas anteriormente instalada y la instanciamos en pd, para luego utilizar sus métodos.

En la línea 2, utilizamos el método read_excel, utilizando solamente la ruta como parámetro. Asignamos el resultado de esa lectura, a una variable “df” que por defecto será un dataframe.

Finalmente, en la tercera línea, solicitamos a Python, la impresión por pantalla de los datos existentes en el dataframe.

Entonces, al ejecutar el código, el notebook nos dará el siguiente resultado:

```
In [2]: import pandas as pd
        df=pd.read_excel("C:/Users/ybarr/Desktop/salud/vdz.xls")
        print(df)

     departamento_id departamento_nombre  provincia_id provincia_nombre  anio  \
0                  0          *sin dato*             6     Buenos Aires  2018
1                 35          Avellaneda             6     Buenos Aires  2018
2                 35          Avellaneda             6     Buenos Aires  2018
3                 35          Avellaneda             6     Buenos Aires  2018
4                 35          Avellaneda             6     Buenos Aires  2018
..               ...                 ...           ...              ...   ...
655              126               Orán             66            Salta  2018
656              126               Orán             66            Salta  2018
657              126               Orán             66            Salta  2018
658              126               Orán             66            Salta  2018
659              126               Orán             66            Salta  2018

     semanas_epidemiologicas evento_nombre  grupo_edad_id  \
0                         11        Dengue             10
1                          7        Dengue              8
2                          7        Dengue             10
3                          9        Dengue              6
4                          9        Dengue              8
..                       ...           ...            ...
```

- **Crear df desde un conjunto de Datos**

  Pandas le permite crear un dataframe desde una lista de valores. El mismo lo creamos con el metido dataframe, enviando como parámetros, el conjunto de claves y sus respectivos valores para esas claves. Veamos un ejemplo:

  Creamos un nuevo dataframe con el siguiente código:

  import pandas as pd
  df1 = pd.DataFrame({'lkey': ['foo', 'bar', 'baz', 'foo'], 'value': [1, 2, 3, 5]})
  print(df1)

  Ejecutamos en notebook y vemos el siguiente resultado:

```
In [2]: import pandas as pd
        df1 = pd.DataFrame({'lkey': ['foo', 'bar', 'baz', 'foo'], 'value': [1, 2, 3, 5]})
        print(df1)

     lkey  value
0   foo      1
1   bar      2
2   baz      3
3   foo      5
```

- **Creación a partir de un escalar**

Si los datos se reducen a un escalar (no a una lista con un único elemento, sino a un sencillo escalar como 7 o 15.4) será necesario añadir el índice explícitamente. El número de elementos de la serie coincidirá con el número de elementos del índice, y el escalar será asignado como valor a todos ellos. La sintaxis es la siguiente:

Df1 = pd.series (7, index = [“Enero”,”Febrero”,”Marzo”])

```
In [15]: df1 = pd.Series (7, index = ["Enero","Febrero","Marzo"])
         print (df1)

Enero      7
Febrero    7
Marzo      7
dtype: int64
```

- **Creación desde un array**

Suponiendo que contamos con el siguiente array:
mi_array = {'col1': 1.0, 'col2':2.0, 'col3': 3.0}
La sintaxis para crear un dataframe a partir del mismo e imprimir sus datos, es la siguiente:
mi_array = {'col1': 1.0, 'col2':2.0, 'col3': 3.0}
df = pd.DataFrame([mi_array])
print (df)

Veamos el resultado de su ejecución:

```
In [16]: mi_array = {'col1': 1.0, 'col2':2.0, 'col3': 3.0}
         df = pd.DataFrame([mi_array])
         print (df)

   col1  col2  col3
0   1.0   2.0   3.0
```

- **Acciones simples sobre un Dataframe**

Siendo el dataframe la variable df1, a continuación profundizamos sobre algunas operaciones básicas:

Mostrar las primeras 10 filas del CSV:
print(df1.head(10))

Contar la cantidad de filas:
print(len(df1)

Contar la cantidad de filas 2:
print(«Cant. filas: %i» % len(df1))
#El %i indica que ahí va una variable de tipo integer.
#Al cerrar comillas dobles, el % indica qué es lo que debe reemplazar a %i

Splicing de la data:
print(df1[:10]) #muestra las primeras 10 filas
print(df1[5:]) #muestra todo salvo las primeras 5 filas
print(df1[-3:]) #muestra las últimas 3 filas
print(df1[:-2]) #muestra todo salvo las últimas 2 filas
print(df1[-5:-2]) #muestra desde la 5ta desde el final hasta 2 desde el final

Pasar el campo "keyword" a minúscula:
Df1 = df1['Keyword'].str.lower()

Cómo ver las columnas de un DataFrame en Pandas:
print(df1.columns)

Cómo ver el contenido de una columna en Pandas:
print(df1['CTR']) #la columna se llama CTR

Cómo sacar el promedio de los valores de una columna en Pandas:
print(df1['CTR'].mean())

Posicionamiento:
Supongamos que deseamos mostrar el primer row completo, mostrando todas las columnas, del dataframe llamado df1. La sintaxis es la siguiente:

Df1.loc[0,:]

Usaremos la siguiente sintaxis para mostrar lo mismo que el caso anterior pero para solamente una columna:

Df1.loc[0,:'nombre_columna']

Tema 3. Operaciones esenciales

Objetivo

El objetivo de este tema es brindar conocimientos básicos que permitan realizar operaciones esenciales sobre dataframes utilizando la librería de código abierto, Pandas.

Una vez finalizado este tema serás capaz de:

- Realizar operaciones esenciales como: Merge, Group By, Resharping, Plotting.

Operaciones esenciales:

Una vez generado un dataframe con datos desde un archivo csv o Excel, podremos trabajar sobre el conjunto de datos. A continuación, nos enfocaremos en las siguientes operaciones: Merge, grouping, Resharping, Plotting.

Merge

En la operación de Merge necesitamos contar con dos dataframes. En este caso vamos a trabajar con 2 df generados a través de una lista de valores que generamos para entender su funcionamiento. Entonces generamos los mismos con el siguiente código:

```
import pandas as pd

df1 = pd.DataFrame({'lkey': ['foo', 'bar', 'baz', 'foo'], 'value': [1, 2, 3, 5]})

df2 = pd.DataFrame({'rkey': ['foo', 'bar', 'baz', 'foo'], 'value': [5, 6, 7, 8]})
```

Ahora bien, primero debemos entender las opciones de Merge que brinda pandas. La sintaxis de Merge es la siguiente:

```
DataFrame.merge (right, how='inner', on=None, left_on=None, right_on=None, left_index=False, right_index=False, sort=False, suffixes=('_x', '_y'), copy=True, indicator=False, validate=None)
```

En este sentido, solo repasaremos los parámetros más importantes del método. Nos enfocaremos puntualmente en el parámetro “how”. El mismo refiere a cómo hacer el merge. Sus posibles valores son:

- left: Usa solamente las claves del df izquierdo. Funciona de forma similar al SQL Left join. Se preserva el orden de las claves del df izquierdo. El resultado de la ejecución de Merge es el siguiente:

```
In [9]: df1.merge(df2, how='left',left_on='lkey', right_on='rkey',suffixes=('_left', '_right'))
Out[9]:
```

	lkey	value_left	rkey	value_right
0	foo	1	foo	5
1	foo	1	foo	8
2	bar	2	bar	6
3	baz	3	baz	7
4	foo	5	foo	5
5	foo	5	foo	8

- right: Usa solamente las claves del df derecho. Funciona de forma similar al SQL Right join. Se preserva el orden de las claves del dataframe derecho. El resultado de la ejecución de Merge es el siguiente:

```
In [10]: df1.merge(df2, how='right',left_on='lkey', right_on='rkey',suffixes=('_left', '_right'))
Out[10]:
```

	lkey	value_left	rkey	value_right
0	foo	1	foo	5
1	foo	5	foo	5
2	bar	2	bar	6
3	baz	3	baz	7
4	foo	1	foo	8
5	foo	5	foo	8

- outer: Usa la union de las claves de ambos df. Funciona de forma similar a un Full Outer Join. El resultado de la ejecución de Merge es el siguiente:

```
In [11]: df1.merge(df2, how='outer',left_on='lkey', right_on='rkey',suffixes=('_left', '_right'))
Out[11]:
```

	lkey	value_left	rkey	value_right
0	foo	1	foo	5
1	foo	1	foo	8
2	foo	5	foo	5
3	foo	5	foo	8
4	bar	2	bar	6
5	baz	3	baz	7

- inner: Usa la intersección de las claves de ambos df. Funciona de forma similar al SQL inner join. Se preserva el orden de las

claves del df izquierdo. El resultado de la ejecución de Merge es el siguiente:

```
In [12]: df1.merge(df2, how='inner',left_on='lkey', right_on='rkey',suffixes=('_left', '_right'))
Out[12]:
```

	lkey	value_left	rkey	value_right
0	foo	1	foo	5
1	foo	1	foo	8
2	foo	5	foo	5
3	foo	5	foo	8
4	bar	2	bar	6
5	baz	3	baz	7

- cross: Funciona de forma similar a un producto cartesiano. Se preserva el orden de las claves del df izquierdo. El resultado de la ejecución de Merge es el siguiente:

```
In [15]: df1.merge(df2, how='cross', suffixes=('_left', '_right'))
Out[15]:
```

	lkey	value_left	rkey	value_right
0	foo	1	foo	5
1	foo	1	bar	6
2	foo	1	baz	7
3	foo	1	foo	8
4	bar	2	foo	5
5	bar	2	bar	6
6	bar	2	baz	7
7	bar	2	foo	8
8	baz	3	foo	5
9	baz	3	bar	6
10	baz	3	baz	7
11	baz	3	foo	8
12	foo	5	foo	5
13	foo	5	bar	6
14	foo	5	baz	7
15	foo	5	foo	8

Nota: Los modificadores ***left_on='lkey', right_on='rkey',suffixes=('_left', '_right')*** nos permiten:

- Colocar un el nombre “lkey” a la columna resultante de la clave del df izquierdo
- Colocar un el nombre “rkey” a la columna resultante de la clave del df derecho
- Agregar el prefijo “_left” y “_right” a las columnas de valores resultantes

Grouping

La sintaxis de Group By Pandas para un dataframe es la siguiente:

DataFrame.groupby(*by=None, axis=0, level=None, as_index=True, sort=True, group_keys=True, squeeze=NoDefault.no_default, observed=False, dropna=True*)

Una operación de agrupamiento, con Group By, implica combinar los valores iguales de uno o más campos. Esta operación se suele utilizar para agrupar grandes cantidades de datos y calcular operaciones agregadas en estos grupos, como por ejemplo, contar, sumar o realizar un promedio. Veamos un ejemplo.

Primero cargaremos en un dataframe un conjunto de datos que podamos agrupar para sumar los valores de ese campo en el dataframe. Entonces, ejecutamos el siguiente código para cargar un archivo CSV con un conjunto de valores:

```
In [36]: import pandas as pd
         df=pd.read_csv("C:/Users/ybarr/Desktop/salud/vira.csv",encoding = "utf-8")
         df2=df.drop(columns=["departamento_id", "departamento_nombre","provincia_id","año","semanas_epidemiologicas","evento_nombre",
                              "grupo_edad_id","grupo_edad_desc"])
         print(df2)

              provincia_nombre  cantidad_casos
         0        Buenos Aires               2
         1        Buenos Aires               3
         2             Córdoba               1
         3        Buenos Aires               2
         4        Buenos Aires               2
         ...               ...             ...
         3458     Buenos Aires               1
         3459             CABA               1
         3460             CABA               2
         3461             CABA               1
         3462             CABA               1

         [3463 rows x 2 columns]
```

El dataframe contiene la cantidad de casos de una enfermedad por localidad en el tiempo.

En la línea 3 de nuestro código, eliminamos todas las columnas (con df.drop(columna=”X”)) que no queremos en nuestro dataframe y asignamos el dataframe resultante, a un nuevo dataframe, df2.

Luego, ejecutaremos el group by sobre el dataframe 2, para sumar la cantidad de casos existentes en el mismo por cada provincia. Para ello ejecutamos lo siguiente:

```
In [35]: df2.groupby(['provincia_nombre'])['cantidad_casos'].sum()

Out[35]: provincia_nombre
         Buenos Aires          1525
         CABA                  1140
         Catamarca              211
         Chaco                  728
         Chubut                 136
         Corrientes             212
         Córdoba                454
         Entre Ríos             444
         Formosa                102
         Jujuy                  476
         La Pampa                49
         La Rioja               297
         Mendoza                357
         Misiones                64
         Neuquén                184
         Río Negro              202
         Salta                  587
         San Juan               325
         San Luis               217
         Santa Cruz             135
         Santa Fe                41
         Santiago del Estero    369
         Tierra del Fuego       150
         Tucumán                638
         Name: cantidad_casos, dtype: int64
```

Vemos como resultado la suma de la cantidad de casos existentes, agrupados por provincia

Cabe resaltar que el agrupamiento podemos hacerlo por uno o más campos del dataframe. Su funcionamiento es similar al group by de SQL.

Por otro lado, las funciones que pueden aplicarse luego del agrupamiento pueden ser, count(), sum(), min(), max(), etc. Más funciones y detalles en el siguiente link (en la entrada “Aggregation”):

https://pandas.pydata.org/pandas-docs/stable/user_guide/groupby.html#dataframe-column-selection-in-groupby

Resharping

Utilizaremos esta funcionalidad para trasponer un dataframe. Para este casos nos enfocaremos solamente en el método melt(), sabiendo que existen además otros métodos para poder realizar acciones similares. Entenderemos más claramente su funcionamiento, con el siguiente ejemplo. Supongamos que tenemos el siguiente dataframe df, creado con la siguiente sintaxis:

```
df = pd.DataFrame({'equipo': ['A', 'B', 'C', 'D'],
                   'goles': [100, 200, 300, 400],
                   'faltas': [1000, 190, 999, 888],
                   'amonestaciones': [232, 218, 310, 311]})
```

Al mostrar los datos por pantalla, vemos lo siguiente:

```
In [6]: df = pd.DataFrame({'equipo': ['A', 'B', 'C', 'D'],
                           'goles': [100, 200, 300, 400],
                           'faltas': [1000, 190, 999, 888],
                           'amonestaciones': [232, 218, 310, 311]})
        df
```

Out[6]:

	equipo	goles	faltas	amonestaciones
0	A	100	1000	232
1	B	200	190	218
2	C	300	999	310
3	D	400	888	311

Aplicaremos el método melt() para hacer un resharp del dataframe df1 y trasponerlo, con la siguiente sintaxis:

df = pd.melt(df, id_vars='equipo', value_vars=['goles', 'faltas', 'amonestaciones'])

Veamos el resultado en pantalla luego de ejecutar melt() sobre el dataframe df1:

```
In [7]: df = pd.melt(df, id_vars='equipo', value_vars=['goles', 'faltas', 'amonestaciones'])
        df
```

Out[7]:

	equipo	variable	value
0	A	goles	100
1	B	goles	200
2	C	goles	300
3	D	goles	400
4	A	faltas	1000
5	B	faltas	190
6	C	faltas	999
7	D	faltas	888
8	A	amonestaciones	232
9	B	amonestaciones	218
10	C	amonestaciones	310
11	D	amonestaciones	311

De esta forma, vemos claramente como hemos modificado el dataframe df1.

Plotting

El metod plot(), lo usaremos para realizar algunos tipos de gráficos en pantalla en base a los datos contenidos en nuestro dataframe. Para trabajar con estos gráficos utilizaremos la librería matplotlib. La página oficial describe a la misma como:

“una biblioteca completa para crear visualizaciones estáticas, animadas e interactivas en Python.”

Para instalar la misma, la sintaxis es la siguiente:

pip install -U matplotlib

Una vez instalada la librería, avanzamos con plot(). La sintaxis de plot() es la siguiente:

dataFrame.plot (*args, **kwargs)

En las siguientes líneas mostraremos algunos ejemplos básicos para graficar dataframes.

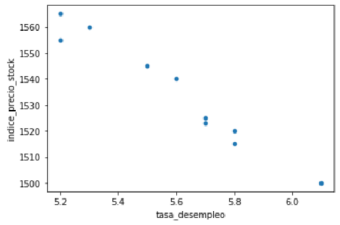
- **Diagrama de Dispersión (Scatter Plot)**

Los diagramas de dispersión se utilizan para representar una relación entre dos variables. Este es el código para realizar un diagrama de dispersión usando Pandas.

```
import pandas as pd

import matplotlib.pyplot as plt


data = {'tasa_desempleo': [6.1,5.8,5.7,5.7,5.8,5.6,5.5,5.3,5.2,5.2],

                                    'indice_precio_stock':
[1500,1520,1525,1523,1515,1540,1545,1560,1555,1565]

    }


df                                                  =
pd.DataFrame(data,columns=['tasa_desempleo','indice_precio_stoc
k'])

df.plot(x ='tasa_desempleo', y='indice_precio_stock', kind = 'scatter')

plt.show()
```

Analicemos el código. En las primeras líneas importamos pandas y matplotlib para graficar. Luego definimos dos dataframes. Posteriormente, en el método plot(), definimos que nuestro gráfico será del tipo “scatter” y enviamos los valores para los ejes X e Y. Vemos el resultado en Notebook:

```
In [2]: import pandas as pd
        import matplotlib.pyplot as plt

        data = {'tasa_desempleo': [6.1,5.8,5.7,5.7,5.8,5.6,5.5,5.3,5.2,5.2],
                'indice_precio_stock': [1500,1520,1525,1523,1515,1540,1545,1560,1555,1565]
               }

        df = pd.DataFrame(data,columns=['tasa_desempleo','indice_precio_stock'])
        df.plot(x ='tasa_desempleo', y='indice_precio_stock', kind = 'scatter')
        plt.show()
```



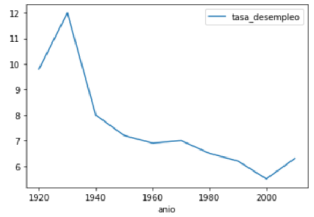
- **Diagrama de líneas**

Los gráficos de líneas se utilizan a menudo para mostrar tendencias a lo largo del tiempo. Este es el código para realizar un diagrama de líneas usando Pandas.

```
import pandas as pd
import matplotlib.pyplot as plt

data = {'anio': [1920,1930,1940,1950,1960,1970,1980,1990,2000,2010],
        'tasa_desempleo': [9.8,12,8,7.2,6.9,7,6.5,6.2,5.5,6.3]
       }

df = pd.DataFrame(data,columns=['anio','tasa_desempleo'])
df.plot(x ='anio', y='tasa_desempleo', kind = 'line')
plt.show()
```

Analicemos un poco el código anterior. En las primeras líneas importamos pandas y matplotlib para graficar. Luego definimos un data con dos columnas, tasa de desempleo y año. Posteriormente, en el método plot(), definimos que nuestro gráfico será del tipo “linea” y enviamos los valores para los ejes X e Y con el fin de graficar la evolución de la tasa de desempleo por año. Vemos el resultado en Notebook:

```
In [3]: import pandas as pd
        import matplotlib.pyplot as plt

        data = {'anio': [1920,1930,1940,1950,1960,1970,1980,1990,2000,2010],
                'tasa_desempleo': [9.8,12,8,7.2,6.9,7,6.5,6.2,5.5,6.3]
                }

        df = pd.DataFrame(data,columns=['anio','tasa_desempleo'])
        df.plot(x ='anio', y='tasa_desempleo', kind = 'line')
        plt.show()
```



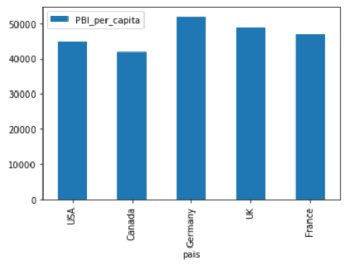
- **Diagrama de líneas**

Los gráficos de barras se utilizan para mostrar datos categóricos. Veamos ahora cómo trazar un gráfico de barras usando Pandas. Este es el código para realizar un diagrama de barras usando Pandas.

```
import pandas as pd

import matplotlib.pyplot as plt


data = {'pais': ['USA','Canada','Germany','UK','France'],

    'PBI_per_capita': [45000,42000,52000,49000,47000]

    }


df = pd.DataFrame(data,columns=['pais','PBI_per_capita'])

df.plot(x ='pais', y='PBI_per_capita', kind = 'bar')

plt.show()
```

Analicemos un poco el código anterior. En las primeras líneas importamos pandas y matplotlib para graficar. Luego definimos un data con dos columnas, País y PBI per cápita. Posteriormente, en el método plot(), definimos que nuestro gráfico será del tipo “Barras” y enviamos los valores para los ejes X e Y con el fin de graficar la evolución el PBI per cápita de acuerdo a cada país. Vemos el resultado en Notebook:

```
In [5]: import pandas as pd
        import matplotlib.pyplot as plt

        data = {'pais': ['USA','Canada','Germany','UK','France'],
                'PBI_per_capita': [45000,42000,52000,49000,47000]
                }

        df = pd.DataFrame(data,columns=['pais','PBI_per_capita'])
        df.plot(x ='pais', y='PBI_per_capita', kind = 'bar')
        plt.show()
```



De esta forma cerramos el tema 3, Operaciones esenciales sobre dataframes.

Cierre

A lo largo de este módulo repasamos el concepto y las funcionalidades básicas de la librería Pandas. Aprendimos a acceder a datos y crear un dataframe. Nos enfocamos en las funcionalidades básicas para la creación y carga de datos de un dataframe. Por otro lado, también aprendimos a realizar operaciones esenciales como: Merge, grouping, Resharping, Plotting.

Referencias

- https://www.w3schools.com/python/pandas/pandas_dataframes.asp
- https://www.geeksforgeeks.org/python-pandas-dataframe/
- https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html
- https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.merge.html
- https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.groupby.html
- https://pandas.pydata.org/docs/reference/api/pandas.melt.html
- https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.plot.html


tiiMiit
by