Numpy y Pandas

En este laboratorio realizarás prácticas utilizando las librerías de NumPy y Pandas. Asimismo, se espera que al final del mismo cuentes con las herramientas necesarias para trabajar con datos empleando estas librerías.

Series

Para iniciar es necesario importar Pandas a nuestro Jupyter Notebook, para lo cual introducirás la siguiente instrucción:

```
In [1]: import pandas as pd
```

Del mismo modo, lo harás para importar Numpy:

```
In [2]: import numpy as np
```

Para crear una serie utilizarás el método de Pandas Series:

```
In [3]: data=pd.Series([2, 4, 6, 8, 10])

In [4]: data

Out[4]: 0     2
        1     4
        2     6
        3     8
        4    10
        dtype: int64
```

También podrás crear una serie con indexado, por ejemplo, si deseas tener una variable que tenga las calificaciones que obtuvieron los estudiantes de una clase, lo realizarás de la siguiente forma:

```
In [6]: calif=pd.Series(data=[80, 90, 75, 100, 83, 97, 79], index=['Ana', 'María', 'Pedro', 'Juan', 'José', 'Bryan', 'Laura'])

In [7]: calif

Out[7]: Ana       80
        María     90
        Pedro     75
        Juan     100
        José      83
        Bryan     97
        Laura     79
        dtype: int64
```

Podrás revisar los valores de cada índice utilizando el dato de texto, tal y como lo harías con los índices numéricos. En este caso, por ejemplo, si deseas verificar cuál es la calificación de Juan solo tendrás que hacer lo siguiente:

```
In [10]: calif['Juan']

Out[10]: 100
```

Podrás utilizar también los índices numéricos, por ejemplo, si quisieras extraer los datos de todas las mujeres podrías hacerlo de esta manera:

```
In [13]: calif[[0, 1, 6]]
Out[13]: Ana      80
         María    90
         Laura    79
         dtype: int64
```

Por otro lado, si lo que te interesara es saber quiénes obtuvieron una calificación mayor a 85, podrías ejecutar la siguiente instrucción:

```
In [14]: calif>85
Out[14]: Ana      False
         María     True
         Pedro    False
         Juan      True
         José     False
         Bryan     True
         Laura    False
         dtype: bool
```

Es posible realizar operaciones con la serie, por ejemplo, para obtener calificaciones en una escala del 0 y 10, basta con que realices la división de la serie entre 10:

```
In [15]: calif/10
Out[15]: Ana       8.0
         María     9.0
         Pedro     7.5
         Juan     10.0
         José      8.3
         Bryan     9.7
         Laura     7.9
         dtype: float64
```

Dataframes

Para crear *dataframes* puedes seguir utilizando el ejemplo anterior, pero ahora agregando las calificaciones de la segunda y la tercera evaluación:

```
In [24]: tabla=pd.DataFrame({
         'primera':pd.Series(data=[80, 90, 75, 100, 83, 97, 79], index=['Ana', 'María', 'Pedro', 'Juan', 'José', 'Bryan', 'Laura']),
         'segunda':pd.Series(data=[60, 94, 79, 96, 77, 50, 89], index=['Ana', 'María', 'Pedro', 'Juan', 'José', 'Bryan', 'Laura']),
         'tercera':pd.Series(data=[90, 100, 100, 57, 97, 99, 100], index=['Ana', 'María', 'Pedro', 'Juan', 'José', 'Bryan', 'Laura'])})
In [25]: tabla
Out[25]:
```

	primera	segunda	tercera
Ana	80	60	90
María	90	94	100
Pedro	75	79	100
Juan	100	96	57
José	83	77	97
Bryan	97	50	99
Laura	79	89	100

Asimismo, desde Numpy se puede generar una matriz de números aleatorios de 4 x 4. Por lo tanto, para el dataframe agregarás adicionalmente como índices los meses donde se recolectaron, mientras que para los encabezados de las columnas utilizarás una lista, tal y como se muestra a continuación:

```
In [28]: datos = pd.DataFrame(np.random.randn(4, 4), index=['enero','febrero', 'marzo', 'abril'], columns=list('ABCD'))

In [29]: datos
```

Out[29]:

	A	B	C	D
enero	-2.356033	-0.167330	0.350056	-1.338982
febrero	0.745043	-1.392040	0.983806	1.436896
marzo	0.743091	1.473013	0.593900	-1.698344
abril	-1.122313	-0.543166	0.598551	-0.037300

En el siguiente ejemplo utilizarás el método de Numpy para generar un vector de 25 elementos (0 a 25) que utilizarás para formar una matriz de datos de cinco filas por cinco columnas con la finalidad de crear una tabla, en la cual las filas tendrán municipios o localidades del área metropolitana de Monterrey, México y en las columnas los datos obtenidos de cada una de las cinco estaciones de monitoreo:

```
In [31]: temp=pd.DataFrame(np.arange(25).reshape((5,5)),
                           index=['San Pedro', 'Apodaca', 'Monterrey', 'San Nicolás', 'Juárez'],
                           columns=['Estación 1', 'Estación 2', 'Estación 3', 'Estación 4', 'Estación 5'])

In [32]: temp
```

Out[32]:

	Estación 1	Estación 2	Estación 3	Estación 4	Estación 5
San Pedro	0	1	2	3	4
Apodaca	5	6	7	8	9
Monterrey	10	11	12	13	14
San Nicolás	15	16	17	18	19
Juárez	20	21	22	23	24

Reading Data

Para leer datos en Pandas debes colocar el archivo de datos deseado dentro de un folder donde se encuentre el programa. En ese crearás un folder llamado Datos, en donde insertarás un pequeño archivo con temperaturas promedio en el año de las tres ciudades más grandes de México. Este archivo tendrá por nombre *Temp.csv* (es un archivo donde los datos están separados por comas). Asimismo, con el comando ls podrás verificar que el archivo esté disponible para ser leído:

```
In [41]: !ls ./datos
         Temp.csv
```

A continuación, utilizarás el comando *cat* para cargar el archivo:

```
In [42]: !cat ./datos/Temp.csv
```

```
,Mexico,Guadalajara,Monterrey
Enero,14.6,13.5,14.4
Febrero,15.9,16.7,16.6
Marzo,18.1,18.1,20
Abril,19.6,20.5,23.4
Mayo,21,22.5,26.2
Junio,19.4,23,27.9
Julio,18.2,21.7,28.6
Agosto,18.3,21.7,28.5
Septiembre,18,20.5,26.2
Octubre,17.1,18.9,22.4
Noviembre,16.3,16,18.4
Diciembre,15,14,15.1
```

Si utilizas la función *read* podrás leer la información más ordenadamente:

```
In [44]: info=pd.read_csv('datos/Temp.csv')
```

```
In [45]: info
```

Out[45]:

	Unnamed: 0	Mexico	Guadalajara	Monterrey
0	Enero	14.6	13.5	14.4
1	Febrero	15.9	16.7	16.6
2	Marzo	18.1	18.1	20.0
3	Abril	19.6	20.5	23.4
4	Mayo	21.0	22.5	26.2
5	Junio	19.4	23.0	27.9
6	Julio	18.2	21.7	28.6
7	Agosto	18.3	21.7	28.5
8	Septiembre	18.0	20.5	26.2
9	Octubre	17.1	18.9	22.4
10	Noviembre	16.3	16.0	18.4
11	Diciembre	15.0	14.0	15.1

Como podrás observar, no se tiene título para la primera columna de datos (donde se encuentran los meses). Para modificar esto, basta con corregir el archivo de datos que fue descargado desde la red.
A continuación, cargarás un archivo llamado TempCorregido.csv:

```
In [46]: info=pd.read_csv('datos/TempCorregido.csv')
         info
```

Out[46]:

	Meses	Mexico	Guadalajara	Monterrey
0	Enero	14.6	13.5	14.4
1	Febrero	15.9	16.7	16.6
2	Marzo	18.1	18.1	20.0
3	Abril	19.6	20.5	23.4
4	Mayo	21.0	22.5	26.2
5	Junio	19.4	23.0	27.9
6	Julio	18.2	21.7	28.6
7	Agosto	18.3	21.7	28.5
8	Septiembre	18.0	20.5	26.2
9	Octubre	17.1	18.9	22.4
10	Noviembre	16.3	16.0	18.4
11	Diciembre	15.0	14.0	15.1

Con el comando *head* podrás extraer elementos de las primeras filas. En este caso, imagina que deseas extraer solo las primeras tres filas de datos. La instrucción quedaría como la siguiente:

```
In [47]: !head -3 ./datos/Temp.csv

         ,Mexico,Guadalajara,Monterrey
         Enero,14.6,13.5,14.4
         Febrero,15.9,16.7,16.6
```

En el caso del dataframe, que previamente definiste como *info*, podrás utilizar la función de head para hacer exactamente lo mismo:

```
In [49]: info.head(3)
```

Out[49]:

	Meses	Mexico	Guadalajara	Monterrey
0	Enero	14.6	13.5	14.4
1	Febrero	15.9	16.7	16.6
2	Marzo	18.1	18.1	20.0

Con el comando *iloc* puedes extraer cualquier fila de la tabla, siempre y cuando especifiques los índices de los elementos que deseas:

```
In [53]: info.iloc[[1, 3, 7, 10]]
```

Out[53]:

	Meses	Mexico	Guadalajara	Monterrey
1	Febrero	15.9	16.7	16.6
3	Abril	19.6	20.5	23.4
7	Agosto	18.3	21.7	28.5
10	Noviembre	16.3	16.0	18.4

Operaciones con datos

Pandas ofrece una gran variedad de funciones para realizar operaciones con datos, ya que dentro de la ciencia de datos las operaciones estadísticas son muy importantes para el análisis de la información.

Tomando como referencia el ejemplo anterior, obtén la media de cada una de las columnas. Para esto, basta con utilizar la siguiente instrucción:

```
In [54]: info.mean()
Out[54]: Mexico         17.625000
         Guadalajara    18.925000
         Monterrey      22.308333
         dtype: float64

In [56]: info['Mexico'].mean()
Out[56]: 17.625
```

Como se puede apreciar en la imagen, cuando se utiliza *info.mean()* se obtienen los promedios de cada una de las columnas. Sin embargo, si deseas la media de alguna columna en particular, por ejemplo, México, entonces basta con indicar que te interesa solo la columna de ese país con *info['Mexico'].mean()*.

Lo mismo sucede si deseas sacar la desviación estándar:

```
In [57]: info.std()
Out[57]: Mexico         1.922179
         Guadalajara    3.270008
         Monterrey      5.294673
         dtype: float64

In [58]: info['Guadalajara'].std()
Out[58]: 3.270008340272933
```

La mediana:

```
In [59]: info.median()
Out[59]: Mexico         18.05
         Guadalajara    19.70
         Monterrey      22.90
         dtype: float64

In [60]: info['Monterrey'].median()
Out[60]: 22.9
```

Los máximos y mínimos:

```
In [61]: info.max()
Out[61]: Meses          Septiembre
         Mexico                 21
         Guadalajara            23
         Monterrey            28.6
         dtype: object

In [62]: info.min()
Out[62]: Meses          Abril
         Mexico          14.6
         Guadalajara     13.5
         Monterrey       14.4
         dtype: object
```

Observa que, en el caso de los máximos y mínimos, se toman en cuenta los meses.

Asimismo, se pueden conseguir las sumatorias de las columnas:

```
In [63]: info.sum()
Out[63]: Meses          EneroFebreroMarzoAbrilMayoJunioJulioAgostoSept...
         Mexico                                                     211.5
         Guadalajara                                                227.1
         Monterrey                                                  267.7
         dtype: object
```

Por otro lado, el cálculo de la correlación entre vectores de datos es de suma importancia en estadística, lo cual es fácil de obtener utilizando *corr*:

```
In [64]: info.corr()
```

Out[64]:

	Mexico	Guadalajara	Monterrey
Mexico	1.000000	0.913243	0.814624
Guadalajara	0.913243	1.000000	0.964178
Monterrey	0.814624	0.964178	1.000000

Con la función *describe* se retornan los principales cálculos de estadística descriptiva de cada una de las columnas:

```
In [65]: info.describe()
```

Out[65]:

	Mexico	Guadalajara	Monterrey
count	12.000000	12.000000	12.000000
mean	17.625000	18.925000	22.308333
std	1.922179	3.270008	5.294673
min	14.600000	13.500000	14.400000
25%	16.200000	16.525000	17.950000
50%	18.050000	19.700000	22.900000
75%	18.575000	21.700000	26.625000
max	21.000000	23.000000	28.600000

Lo cual también se puede hacer por columna:

```
In [66]: info['Mexico'].describe()
```

```
Out[66]: count    12.000000
         mean     17.625000
         std       1.922179
         min      14.600000
         25%      16.200000
         50%      18.050000
         75%      18.575000
         max      21.000000
         Name: Mexico, dtype: float64
```

Por otro lado, se puede trabajar con datos redondeados mediante la función *round*:

```
In [67]: info.round()
```

Out[67]:

	Meses	Mexico	Guadalajara	Monterrey
0	Enero	15.0	14.0	14.0
1	Febrero	16.0	17.0	17.0
2	Marzo	18.0	18.0	20.0
3	Abril	20.0	20.0	23.0
4	Mayo	21.0	22.0	26.0
5	Junio	19.0	23.0	28.0
6	Julio	18.0	22.0	29.0
7	Agosto	18.0	22.0	28.0
8	Septiembre	18.0	20.0	26.0
9	Octubre	17.0	19.0	22.0
10	Noviembre	16.0	16.0	18.0
11	Diciembre	15.0	14.0	15.0

Ordenar es otra opción muy útil en el procesamiento de datos. Para este fin Pandas cuenta con la función *sort_values,* que ordena de múltiples formas sin perder la información, por ejemplo, imagina que deseas ordenar la información de los meses del año por orden alfabético, por lo cual el reordenamiento de la columna de los meses se realizaría llevando consigo los datos de las filas, tal y como se muestra en el siguiente ejemplo:

```
In [69]: info.sort_values(by=['Meses'])
```

Out[69]:

	Meses	Mexico	Guadalajara	Monterrey
3	Abril	19.6	20.5	23.4
7	Agosto	18.3	21.7	28.5
11	Diciembre	15.0	14.0	15.1
0	Enero	14.6	13.5	14.4
1	Febrero	15.9	16.7	16.6
6	Julio	18.2	21.7	28.6
5	Junio	19.4	23.0	27.9
2	Marzo	18.1	18.1	20.0
4	Mayo	21.0	22.5	26.2
10	Noviembre	16.3	16.0	18.4
9	Octubre	17.1	18.9	22.4
8	Septiembre	18.0	20.5	26.2

En caso de organizar en orden inverso:

```
In [70]: info.sort_values(by=['Meses'], ascending=False)
```

Out[70]:

	Meses	Mexico	Guadalajara	Monterrey
8	Septiembre	18.0	20.5	26.2
9	Octubre	17.1	18.9	22.4
10	Noviembre	16.3	16.0	18.4
4	Mayo	21.0	22.5	26.2
2	Marzo	18.1	18.1	20.0
5	Junio	19.4	23.0	27.9
6	Julio	18.2	21.7	28.6
1	Febrero	15.9	16.7	16.6
0	Enero	14.6	13.5	14.4
11	Diciembre	15.0	14.0	15.1
7	Agosto	18.3	21.7	28.5
3	Abril	19.6	20.5	23.4

Plotting

Para graficar puedes utilizar los comandos básicos de graficado, como es el caso de *plot*, por ejemplo, genera 15 datos aleatorios en una serie, en donde los índices vayan de dos en dos. Para esto, escribe la siguiente instrucción:

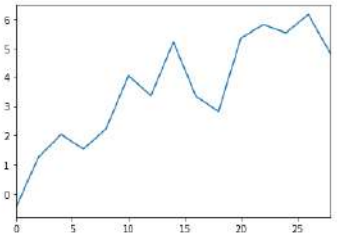
```
In [76]: data = pd.Series(np.random.randn(15).cumsum(), index=np.arange(0, 30, 2))

In [77]: data

Out[77]: 0     1.879868
         2     0.913626
         4     1.066782
         6     0.700783
         8     2.318898
         10    0.564755
         12   -0.458379
         14    0.013433
         16   -0.065902
         18    0.338526
         20   -1.699668
         22   -1.620616
         24    0.114931
         26    0.510410
         28   -2.424731
         dtype: float64
```

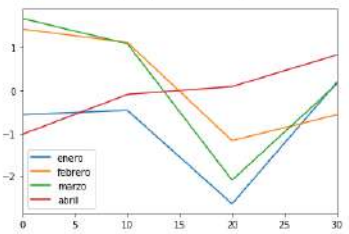
Después de eso utiliza la función *plot*:

```
In [83]: data.plot()
Out[83]: <matplotlib.axes._subplots.AxesSubplot at 0x1869b0a4bc8>
```



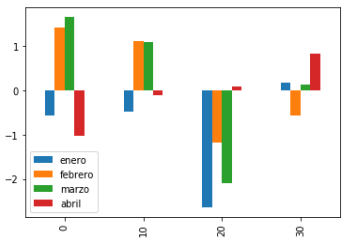
En el siguiente ejemplo utilizarás un dataframe para graficar los datos correspondientes a cada una de las columnas (enero, febrero, marzo y abril) contra los valores del eje x (que en este caso se trata de valores desde 0 de 10 en 10):

```
In [86]: datos = pd.DataFrame(np.random.randn(4, 4), columns=['enero','febrero', 'marzo', 'abril'], index=np.arange(0,40,10))
         datos.plot()
Out[86]: <matplotlib.axes._subplots.AxesSubplot at 0x1869b147e48>
```



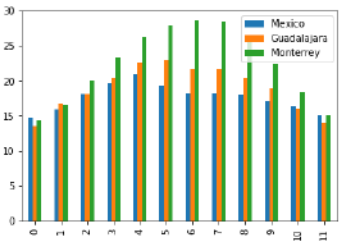
Esos datos también los puedes graficar utilizando barras, para lo cual debes especificar dentro de la función el tipo de graficado que deseas. En este caso sería algo como lo siguiente:

```
In [88]: datos.plot(kind='bar')
Out[88]: <matplotlib.axes._subplots.AxesSubplot at 0x1869b1cfa08>
```



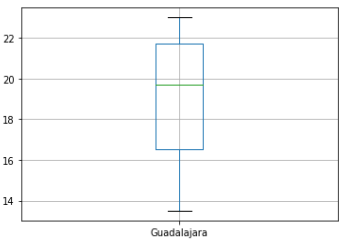
Si desearas que el dataframe se grafique en barras (en este caso el que realizaste con la información de las temperaturas y al cual llamamos Info) solo deberás usar la función plot:

```
In [89]: info.plot(kind='bar')
Out[89]: <matplotlib.axes._subplots.AxesSubplot at 0x1869b259a08>
```



También podrás hacer gráficas de caja con *boxplot*:

```
In [91]: info.boxplot(column='Guadalajara')
Out[91]: <matplotlib.axes._subplots.AxesSubplot at 0x1869b329ac8>
```



Los siguientes enlaces son externos a la Universidad Tecmilenio, al acceder a estos considera que debes apegarte a sus términos y condiciones.

Para mayor información, consulta la siguiente página: https://pandas.pydata.org