16 - Pandas

December 1, 2022

1 Pandas

1.1 Ciencia de Datos

Hoy en día, una de las áreas del conocimiento que se encuentra en crecimiento a nivel exponencial es la *Data Science* (ciencia de datos). Se basa en combinar distintas áreas, como estádistica, matemática, programación y visualización de datos, con la finalidad de interpretar información de manera más clara y eficiente. La idea es poder encontrar patrones en los datos que a simple vista no son detectables, de manera de poder analizar y comprender fenómenos para poder tomar las respectivas decisiones.

Básicamente trabaja con el análisis de grandes cantidades de datos para poder ser analizadas en busca de patrones para la toma de decisiones.

1.1.1 Python y la Ciencia de Datos

Como se ha mencionado antes, Python posee diversos módulos que permiten realizar una serie de tareas en las distintas áreas del conocimiento.

Para poder aportar en la Ciencia de Datos, Python provee de funciones que apoyan la labor, las cuales se encuentran agrupadas en el módulo `pandas`.

El módulo `pandas` es una biblioteca de código abierto que facilita el análisis de grandes cantidad de datos, como la manipulación, limpieza y transformaciones en cortos periodos de tiempo.

Este módulo fue diseñado para trabajar datos tabulados, por lo que es de gran utilidad considerando la similitud de esto con programas de uso masivo como Microsoft Excel.

1.2 Instalación

Como se mencionó previamente, para poder utilizar las funcionalidades que ofrece `pandas` es necesario instalarlo. Para ello, se sugiere revisar el apunte de `numpy` en donde se explica de manera detallada cómo instalar distintos módulos en el sistema.

1.3 Importación

Para poder trabajar con la estructura y tipos de datos que ofrece este módulo, se requiere importarlo, lo que se realizará de la siguiente manera:

```
import pandas as pd
```

1.4 Estructura de Datos

El módulo `pandas` ofrece la posibilidad de trabajar con 2 nuevos tipos de datos:

- `Series`: son lista de datos con un largo determinado, es decir, no se pueden quitar ni agregar elementos, solo modificarlos. Para poder visualizarlo de manera más sencilla, se puede considerar como si fuera una columna en una tabla de datos. Cada elemento tiene un nombre asociado, que corresponde a su índice (como en una lista, salvo que puede ser otro tipo de dato inmutable, como string o número flotante).
- `DataFrame`: son estructuras de datos de dos dimensiones que emulan tablas de datos, en donde se pueden encontrar filas y columnas. Tanto las columas como las filas tienen índices con las mismas características que los índices en una `Series`.

1.4.1 `Series`

Para poder crear este tipo de dato, el mecanismo más directo es definir una lista de datos y utilizar el método que `pandas` ofrece, el cual es `.Series(<lista>)`.

Lo anterior se muestra en el siguiente ejemplo:

[1]:
```
# Ejemplo_01: creación de una serie de pandas
# Importación de Funciones
import pandas as pd

lista_numeros = [2, 4, 6, 8, 10]
serie = pd.Series(lista_numeros)

print(serie)
```

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

La primera columna muestra los índices y la segunda, los valores. Podemos especificar los índices con el parámetro opcional `index`:

[2]:
```
# Ejemplo_01b: creación de una serie de pandas con índices
# Importación de Funciones
import pandas as pd

lista_numeros = [2, 4, 6, 8, 10]
# Nótese que los índices son tantos como elementos de la lista
serie = pd.Series(lista_numeros, index=list("abcde"))

print(serie)
```

```
a     2
b     4
```

```
c     6
d     8
e    10
dtype: int64
```

1.4.2 DataFrame

Para poder crear este tipo de datos, el mecanismo más sencillo es definir una lista de listas de datos y utilizar el método que pandas ofrece, el cual es .DataFrame(<lista>).

Lo anterior se muestra en el siguiente ejemplo:

[3]:
```
# Ejemplo_02: creación de un DataFrame de pandas
# Importación de Funciones
import pandas as pd

lista_notas = [[4.2, 5.3, 4.8, 6.9, 7], [3.9, 6.8, 5.6, 6.2, 6.0], [4.0, 2.0, 6.
↪1, 4.1, 3.1]]
df = pd.DataFrame(lista_notas)

print(df)
```

```
     0    1    2    3    4
0  4.2  5.3  4.8  6.9  7.0
1  3.9  6.8  5.6  6.2  6.0
2  4.0  2.0  6.1  4.1  3.1
```

Como se aprecia en el **Ejemplo_02**, el DataFrame es una tabla que tiene posiciones determinadas.

Por lo general, cuando se busca en bibliografía información sobre DataFrame, se les llama con el alias df. Por ello, cada vez que se genere un DataFrame, se le llamará con el mismo alias.

Si se considera que los datos ingresados en el **Ejemplo_02** al DataFrame son las notas de 3 personas en 5 asignaturas, se podría indicar el nombre de dichas personas.

Para lo anterior, pandas permite el uso del tipo de dato diccionario. Por temas de considerar que este curso es solo una introducción a la programación, no se ahondará sobre las características de este tipo de dato, solo se utilizará para ejemplificar las ventajas que presenta al utilizarlo con DataFrame.

[4]:
```
# Ejemplo_03: creación de un DataFrame de pandas
# Importación de Funciones
import pandas as pd

dic_notas = {"Juan":[4.2, 5.3, 4.8, 6.9, 7],
             "Daniela":[3.9, 6.8, 5.6, 6.2, 6.0],
             "Miguel":[4.0, 2.0, 6.1, 4.1, 3.1]}
df = pd.DataFrame(dic_notas)

print(df)
```

```
   Juan  Daniela  Miguel
0   4.2      3.9     4.0
1   5.3      6.8     2.0
2   4.8      5.6     6.1
3   6.9      6.2     4.1
4   7.0      6.0     3.1
```

Si ahora se quiere que las filas muestren a qué asignaturas corresponden las notas ingresadas, se utiliza un parámetro opcional que poseen los `DataFrame`, al igual que las `Series`, que es `index`:manera:

`pd.DataFrame(<diccionario>, index = [<nombres de cada elemento>]`

El **Ejemplo_04** muestra lo indicado anteriormente:

[6]:
```
# Ejemplo_04: creación de un DataFrame de pandas
# Importación de Funciones
import pandas as pd

dic_notas = {"Juan":[4.2, 5.3, 4.8, 6.9, 7],
             "Daniela":[3.9, 6.8, 5.6, 6.2, 6.0],
             "Miguel":[4.0, 2.0, 6.1, 4.1, 3.1]}

df = pd.DataFrame(dic_notas, index = ["Matemáticas", "Lenguaje", "Artes",␣
 ↪"Física", "Programación"])

print(df)
```

```
              Juan  Daniela  Miguel
Matemáticas    4.2      3.9     4.0
Lenguaje       5.3      6.8     2.0
Artes          4.8      5.6     6.1
Física         6.9      6.2     4.1
Programación   7.0      6.0     3.1
```

1.5 Indexación

1.5.1 Indexación de Filas

Para poder acceder a determinadas posiciones, tanto para `Series` como para `DataFrame`, se utilizan 2 métodos:

- `<Serie/DataFrame>.iloc[<posición>]`: devuelve el valor que se encuentra en la fila mencionada o la fila completa, según sea el caso.
- `<DataFrame>.loc[<nombre>]`: devuelve el valor que se encuentra en la fila mencionada o la fila completa, según sea el caso.

Los ejemplos siguientes mostrarán el uso de los métodos mencionados:

[7]:
```
# Ejemplo_05: creación de una serie de pandas y la posterior obtención de una␣
 ↪fila en particular
```

```
# Importación de Funciones
import pandas as pd

lista_numeros = [2, 4, 6, 8, 10]
serie = pd.Series(lista_numeros)

posicion = serie.iloc[2]

print(serie)
print("\n")
print("El valor de la fila obtenida es", posicion)
```

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


El valor de la fila obtenida es 6
```

[9]:
```
# Ejemplo_06: creación de un DataFrame de pandas y la posterior obtención de una␣
 ↪fila en particular

# Importación de Funciones
import pandas as pd

dic_notas = {"Juan":[4.2, 5.3, 4.8, 6.9, 7],
             "Daniela":[3.9, 6.8, 5.6, 6.2, 6.0],
             "Miguel":[4.0, 2.0, 6.1, 4.1, 3.1]}

df = pd.DataFrame(dic_notas, index = ["Matemáticas", "Lenguaje", "Artes",␣
 ↪"Física", "Programación"])

# Como esta es una fila completa, el tipo de dato es una Series
posicion_df = df.iloc[1]

print(df)
print("\n")
print(posicion_df)
```

```
             Juan  Daniela  Miguel
Matemáticas   4.2      3.9     4.0
Lenguaje      5.3      6.8     2.0
Artes         4.8      5.6     6.1
Física        6.9      6.2     4.1
Programación  7.0      6.0     3.1
```

```
Juan       5.3
Daniela    6.8
Miguel     2.0
Name: Lenguaje, dtype: float64
```

Si se quiere mostrar un rango de filas, tanto en las `Series` como en los `DataFrames`, se deben seguir las siguientes instrucciones:

- **Uso de** `.iloc[]`: se debe colocar una posición inicial y una posición final dentro del corchete, separados por **:** (como cortes en listas y strings). Se debe tener en cuenta, que al igual que en las listas, la posición final que se escribe dentro del corchete **NO** es considerada.

```
Serie/DataFrame.iloc[<posición_inicial>:<posición_final>]
```

- **Uso de** `.loc[]`: se debe colocar dentro de un doble corchete los nombres de todas las filas que se desean filtrar:

```
Serie/DataFrame.loc[[<nombre_fila_1>, <nombre_fila_2>, ...]]
```

O, como alternativa, para obtener un rango de filas, se dan los nombres de la primera y última fila deseada, separados por dos puntos:

```
Serie/DataFrame.loc[<fila_inicial>:<fila_final>]
```

[10]:
```python
# Ejemplo_07: creación de una serie de pandas y la posterior obtención de un␣
 ↪rango de fila
# Importación de Funciones
import pandas as pd

lista_numeros = [2, 4, 6, 8, 10]
serie = pd.Series(lista_numeros)

posicion = serie.iloc[2:5]

print(serie)
print("\n")
print(posicion)
```

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


2     6
3     8
4    10
```

```
dtype: int64
```

[11]:
```
# Ejemplo_08: creación de un DataFrame de pandas y la posterior obtención de␣
 ↪rango de fila
# Importación de Funciones
import pandas as pd

dic_notas = {"Juan":[4.2, 5.3, 4.8, 6.9, 7],
             "Daniela":[3.9, 6.8, 5.6, 6.2, 6.0],
             "Miguel":[4.0, 2.0, 6.1, 4.1, 3.1]}
df = pd.DataFrame(dic_notas, index = ["Matemáticas", "Lenguaje", "Artes",␣
 ↪"Física", "Programación"])

posicion_df = df.iloc[1:3]

print(df)
print("\n")
print(posicion_df)
```

```
             Juan  Daniela  Miguel
Matemáticas   4.2      3.9     4.0
Lenguaje      5.3      6.8     2.0
Artes         4.8      5.6     6.1
Física        6.9      6.2     4.1
Programación  7.0      6.0     3.1


          Juan  Daniela  Miguel
Lenguaje   5.3      6.8     2.0
Artes      4.8      5.6     6.1
```

[12]:
```
#Ejemplo_09: creación de un DataFrame de pandas y la posterior obtención de una␣
 ↪fila en particular
# Importación de Funciones
import pandas as pd

dic_notas = {"Juan":[4.2, 5.3, 4.8, 6.9, 7],
             "Daniela":[3.9, 6.8, 5.6, 6.2, 6.0],
             "Miguel":[4.0, 2.0, 6.1, 4.1, 3.1]}
df = pd.DataFrame(dic_notas, index = ["Matemáticas", "Lenguaje", "Artes",␣
 ↪"Física", "Programación"])

# Un subconjunto de filas
posicion_df_2 = df.loc[["Lenguaje", "Física"]]
# Un rango de filas
posicion_df_3 = df.loc["Lenguaje":"Física"]

print(df)
```

```
print("\n")
print(posicion_df_2)
print("\n")
print(posicion_df_3)
```

```
              Juan  Daniela  Miguel
Matemáticas    4.2      3.9     4.0
Lenguaje       5.3      6.8     2.0
Artes          4.8      5.6     6.1
Física         6.9      6.2     4.1
Programación   7.0      6.0     3.1


          Juan  Daniela  Miguel
Lenguaje   5.3      6.8     2.0
Física     6.9      6.2     4.1


          Juan  Daniela  Miguel
Lenguaje   5.3      6.8     2.0
Artes      4.8      5.6     6.1
Física     6.9      6.2     4.1
```

1.5.2 Indexación de Columnas

Para acceder a las distintas columnas que tienen los `DataFrame`, se utilizan los corchetes. En otras palabras solo se debe indicar el número de la columna o, en caso de haber colocado nombres a las columnas, se debe indicar el nombre particular de la columna a mostrar.

`df.[<nombre_columna>]`

El ejemplo siguiente mostrará cómo se accede a columnas determinadas:

[13]:
```
#Ejemplo_10: creación de un DataFrame de pandas y la posterior obtención de una␣
 ↪columna en particular
# Importación de Funciones
import pandas as pd

dic_notas = {"Juan":[4.2, 5.3, 4.8, 6.9, 7],
             "Daniela":[3.9, 6.8, 5.6, 6.2, 6.0],
             "Miguel":[4.0, 2.0, 6.1, 4.1, 3.1]}
df = pd.DataFrame(dic_notas, index = ["Matemáticas", "Lenguaje", "Artes",␣
 ↪"Física", "Programación"])

columna = df["Juan"]

print(df)
print("\n")
print(columna)
```

```
              Juan  Daniela  Miguel
Matemáticas    4.2      3.9     4.0
Lenguaje       5.3      6.8     2.0
Artes          4.8      5.6     6.1
Física         6.9      6.2     4.1
Programación   7.0      6.0     3.1


Matemáticas     4.2
Lenguaje        5.3
Artes           4.8
Física          6.9
Programación    7.0
Name: Juan, dtype: float64
```

Si se quiere acceder a más de una columna, en lugar de un nombre de columna, se usa como "índice" una lista con los nombres de columnas que queremos acceder:

```
df[[<nombre_columna_inicial>, <nombre_columna_final>]]
```

[14]:
```
# Ejemplo_11: creación de un DataFrame de pandas y la posterior obtención de un␣
 ↪conjunto de columnas
# Importación de Funciones
import pandas as pd

dic_notas = {"Juan":[4.2, 5.3, 4.8, 6.9, 7],
             "Daniela":[3.9, 6.8, 5.6, 6.2, 6.0],
             "Miguel":[4.0, 2.0, 6.1, 4.1, 3.1]}

df = pd.DataFrame(dic_notas, index = ["Matemáticas", "Lenguaje", "Artes",␣
 ↪"Física", "Programación"])

columna = df[["Juan", "Miguel"]]

print(df)
print("\n")
print(columna)
```

```
              Juan  Daniela  Miguel
Matemáticas    4.2      3.9     4.0
Lenguaje       5.3      6.8     2.0
Artes          4.8      5.6     6.1
Física         6.9      6.2     4.1
Programación   7.0      6.0     3.1


              Juan  Miguel
Matemáticas    4.2     4.0
Lenguaje       5.3     2.0
```

```
Artes          4.8     6.1
Física         6.9     4.1
Programación   7.0     3.1
```

1.5.3 Indexación Combinada

Es posible acceder a una combinación de filas y columnas determinadas utilizando los métodos vistos anteriormente. Para ello se debe realizar lo siguiente:

- **Uso de** `.loc[]`: separamos por coma las filas y las columnas que queremos, utilizando cualquiera de las notaciones vistas (nombres únicos, listas de nombres o rangos de nombres), por ejemplo:

```
DataFrame.loc[[<nombre_fila_1>, <nombre_fila:2>, ...], [<nombre_column_1>, <nombre_columna_2>, .
```

- **Uso de** `.iloc[]`: separamos por coma las posiciones de filas y columnas que queremos, utilizando cualquiera de las notaciones vistas (posiciones o rangos de posiciones), por ejemplo:

```
DataFrame.iloc[<posición_inicial_fila>:<posición_final_fila>, <posición_inicial_columna>:<posici
```

[15]:
```
# Ejemplo_12: creación de un DataFrame de pandas y la posterior obtención de un␣
 ↪conjunto de columnas
# Importación de Funciones
import pandas as pd

dic_notas = {"Juan":[4.2, 5.3, 4.8, 6.9, 7],
             "Daniela":[3.9, 6.8, 5.6, 6.2, 6.0],
             "Miguel":[4.0, 2.0, 6.1, 4.1, 3.1]}
df = pd.DataFrame(dic_notas, index = ["Matemáticas", "Lenguaje", "Artes",␣
 ↪"Física", "Programación"])

segmento = df.loc[["Lenguaje", "Física"], ["Juan", "Miguel"]]

print(df)
print("\n")
print(segmento)
```

```
              Juan  Daniela  Miguel
Matemáticas    4.2      3.9     4.0
Lenguaje       5.3      6.8     2.0
Artes          4.8      5.6     6.1
Física         6.9      6.2     4.1
Programación   7.0      6.0     3.1


          Juan  Miguel
Lenguaje   5.3     2.0
Física     6.9     4.1
```

[16]:
```
# Ejemplo_13: creación de un DataFrame de pandas y la posterior obtención de un␣
 ↪conjunto de columnas
# Importación de Funciones
import pandas as pd

dic_notas = {"Juan":[4.2, 5.3, 4.8, 6.9, 7],
             "Daniela":[3.9, 6.8, 5.6, 6.2, 6.0],
             "Miguel":[4.0, 2.0, 6.1, 4.1, 3.1]}
df = pd.DataFrame(dic_notas, index = ["Matemáticas", "Lenguaje", "Artes",␣
 ↪"Física", "Programación"])

segmento = df.iloc[0:3, 0:2]

print(df)
print("\n")
print(segmento)
```

```
              Juan  Daniela  Miguel
Matemáticas    4.2      3.9     4.0
Lenguaje       5.3      6.8     2.0
Artes          4.8      5.6     6.1
Física         6.9      6.2     4.1
Programación   7.0      6.0     3.1


             Juan  Daniela
Matemáticas   4.2      3.9
Lenguaje      5.3      6.8
Artes         4.8      5.6
```

1.6 Operaciones en Series/DataFrame

Trabajar con el módulo pandas facilita considerablemente el trabajo, especialmente porque no se requiere realizar ciclos. Esto se debe a que este módulo está orientado al trabajo con filas y columnas.

Como pandas está basado en numpy, se pueden encontrar operaciones como cálculos estadísticos y otros.

1.6.1 Cálculo del promedio

[18]:
```
# Ejemplo_14: creación de un DataFrame de pandas y la posterior obtención del␣
 ↪promedio de cada estudiante
# Importación de Funciones
import pandas as pd

dic_notas = {"Juan":[4.2, 5.3, 4.8, 6.9, 7],
             "Daniela":[3.9, 6.8, 5.6, 6.2, 6.0],
```

```
            "Miguel":[4.0, 2.0, 6.1, 4.1, 3.1]}
df = pd.DataFrame(dic_notas, index = ["Matemáticas", "Lenguaje", "Artes",␣
 ↪"Física", "Programación"])

# Obtiene el promedio aritmético por columna y el resultado es una serie cuyos␣
 ↪índices son los nombres de cada
# columna original
promedio = df.mean()

print(df)
print("\n")
print(promedio)
```

```
              Juan  Daniela  Miguel
Matemáticas    4.2      3.9     4.0
Lenguaje       5.3      6.8     2.0
Artes          4.8      5.6     6.1
Física         6.9      6.2     4.1
Programación   7.0      6.0     3.1


Juan       5.64
Daniela    5.70
Miguel     3.86
dtype: float64
```

1.6.2 Cálculo de la mediana

[19]:
```
# Ejemplo_15: creación de un DataFrame de pandas y la posterior obtención de la␣
 ↪mediana de cada estudiante
# Importación de Funciones
import pandas as pd

dic_notas = {"Juan":[4.2, 5.3, 4.8, 6.9, 7],
            "Daniela":[3.9, 6.8, 5.6, 6.2, 6.0],
            "Miguel":[4.0, 2.0, 6.1, 4.1, 3.1]}
df = pd.DataFrame(dic_notas, index = ["Matemáticas", "Lenguaje", "Artes",␣
 ↪"Física", "Programación"])

mediana = df.median()

print(df)
print("\n")
print(mediana)
```

```
              Juan  Daniela  Miguel
Matemáticas    4.2      3.9     4.0
Lenguaje       5.3      6.8     2.0
```

```
Artes          4.8      5.6     6.1
Física         6.9      6.2     4.1
Programación   7.0      6.0     3.1


Juan       5.3
Daniela    6.0
Miguel     4.0
dtype: float64
```

1.6.3 Cálculo de la desviación estándar

[20]:
```
# Ejemplo_16: creación de un DataFrame de pandas y la posterior obtención de la
# desviación estándar de cada estudiante
# Importación de Funciones
import pandas as pd

dic_notas = {"Juan":[4.2, 5.3, 4.8, 6.9, 7],
             "Daniela":[3.9, 6.8, 5.6, 6.2, 6.0],
             "Miguel":[4.0, 2.0, 6.1, 4.1, 3.1]}
df = pd.DataFrame(dic_notas, index = ["Matemáticas", "Lenguaje", "Artes",␣
 ↪"Física", "Programación"])

desviacion = df.std()

print(df)
print("\n")
print(desviacion)
```

```
              Juan  Daniela  Miguel
Matemáticas    4.2      3.9     4.0
Lenguaje       5.3      6.8     2.0
Artes          4.8      5.6     6.1
Física         6.9      6.2     4.1
Programación   7.0      6.0     3.1


Juan       1.258173
Daniela    1.095445
Miguel     1.510960
dtype: float64
```

1.6.4 Filtrar Tabla

El módulo `pandas` ofrece la opción de filtrar los DataFrame mediante operaciones lógicas aplicadas a las columnas, conocidas como **máscaras**.

Las operaciones aritméticas y de comparación funcionan exactamente igual a como funcionan en `numpy`. En particular, al comparar un vector, matriz, `Series` o `DataFrame` con un valor, el resultado

es el mismo dato, pero con sus valores reemplazados con `True` o `False`, según si cumple o no la condición:

[21]:
```
import numpy as np
v = np.random.random(10)

print(v > 0.5)

lista = [3,4,2]
```

```
[ True False False  True False False False False  True False]
```

Podemos usar este resultado para **enmascarar** los valores que cumplen la condición:

[23]:
```
import numpy as np
v = np.random.random(10)
print(v)
print(v[v > 0.5])
```

```
[0.15499279 0.98537039 0.69457572 0.55658684 0.96666472 0.48275747
 0.61405709 0.41052536 0.3901588  0.35908873]
[0.98537039 0.69457572 0.55658684 0.96666472 0.61405709]
```

En el **Ejemplo_17**, usaremos este principio para filtrar las notas de Miguel que son mayores a 4.0 y quedarnos solo no esas filas:

[28]:
```
# Ejemplo_17: creación de un DataFrame de pandas y la posterioraplicación de␣
 ↪filtros a toda la tabla
# Importación de Funciones
import pandas as pd

dic_notas = {"Juan":[4.2, 5.3, 4.8, 6.9, 7],
             "Daniela":[3.9, 6.8, 5.6, 6.2, 6.0],
             "Miguel":[4.0, 2.0, 6.1, 4.1, 3.1]}

df = pd.DataFrame(dic_notas, index = ["Matemáticas", "Lenguaje", "Artes",␣
 ↪"Física", "Programación"])

# Compara las notas de Miguel con 4 y se queda solo con las filas que cumplen
tabla_filtrada = df[df["Miguel"] >= 4.0]

print(df)
print("\n")
print(tabla_filtrada)
```

```
             Juan  Daniela  Miguel
Matemáticas   4.2      3.9     4.0
Lenguaje      5.3      6.8     2.0
Artes         4.8      5.6     6.1
Física        6.9      6.2     4.1
```

```
Programación   7.0      6.0     3.1
```

```
             Juan  Daniela  Miguel
Matemáticas   4.2      3.9     4.0
Artes         4.8      5.6     6.1
Física        6.9      6.2     4.1
```

Se puede aplicar filtros a columna en específico, según se requiera.

En el **Ejemplo_18** se filtran los datos, mostrando solo las notas que Miguel tiene sobre 4.0.

[27]:
```
# Ejemplo_18: creación de un DataFrame de pandas y la posterior aplicación de␣
 ↪filtros a una columna en particular
# Importación de Funciones
import pandas as pd

dic_notas = {"Juan":[4.2, 5.3, 4.8, 6.9, 7],
             "Daniela":[3.9, 6.8, 5.6, 6.2, 6.0],
             "Miguel":[4.0, 2.0, 6.1, 4.1, 3.1]}
df = pd.DataFrame(dic_notas, index = ["Matemáticas", "Lenguaje", "Artes",␣
 ↪"Física", "Programación"])

tabla_filtrada = df[df["Miguel"] > 4.0]["Miguel"]

print(df)
print("\n")
print(tabla_filtrada)
```

```
             Juan  Daniela  Miguel
Matemáticas   4.2      3.9     4.0
Lenguaje      5.3      6.8     2.0
Artes         4.8      5.6     6.1
Física        6.9      6.2     4.1
Programación  7.0      6.0     3.1


Artes     6.1
Física    4.1
Name: Miguel, dtype: float64
```

1.7 Importación y Exportación de `DataFrame`

Una de las ventajas de trabajar con el módulo `pandas` es que se pueden crear tablas (`DataFrame`), filtrarlas o realizar muchos cálculos para la toma de decisiones de manera más simple. Más aún, este módulo permite exportar las tablas que se vayan generando para ser analizada por otras personas o en otro momento.

Para ello, `pandas` ofrece el método `.to_csv(<nombre_archivo_a_crear)`, lo que genera un archivo de tipo CSV.

El **Ejemplo_19** muestra el proceso descrito anteriormente:

[29]:
```
# Ejemplo_19: creación de un DataFrame de pandas y la posterior exportación para␣
 ↪futuros trabajos
# Importación de Funciones
import pandas as pd

dic_notas = {"Juan":[4.2, 5.3, 4.8, 6.9, 7],
             "Daniela":[3.9, 6.8, 5.6, 6.2, 6.0],
             "Miguel":[4.0, 2.0, 6.1, 4.1, 3.1]}
df = pd.DataFrame(dic_notas, index = ["Matemáticas", "Lenguaje", "Artes",␣
 ↪"Física", "Programación"])

columnas_filtradas = df.loc[["Lenguaje", "Física"], ["Juan", "Miguel"]]

columnas_filtradas.to_csv("columnas_seleccionadas.csv", index=False)
```

Una vez exportados los datos en forma de tablas (`DataFrame`), es posible volver a trabajar con los datos filtrados.

Para ello, `pandas` cuanta con el método `.read_csv(<nombre_archivo>)`, que permite cargar datos en formato CSV.

El **Ejemplo_20** muestra el proceso descrito anteriormente:

[30]:
```
# Ejemplo_20: Importación de un DataFrame creado con anterioridad

# Importación de Funciones
import pandas as pd

df_filtrado = pd.read_csv('columnas_seleccionadas.csv')

print(df_filtrado)
```

```
   Juan  Miguel
0   5.3     2.0
1   6.9     4.1
```

Lo mostrado hasta el momento es solo una pequeña parte de lo que el módulo `pandas` ofrece para facilitar el trabajo en análisis de datos.

Se recomienda que, en caso de querer ahondar en otras funcionalidades, lean la documetación que pueden encontrar en el siguiente enlace:

Documentación de `pandas`

2 Bibliografía

GeeksforGeeks. (2020, 29 febrero). Pandas Tutorial. Recuperado 10 de agosto de 2022, de https://www.geeksforgeeks.org/pandas-tutorial/

McKinney, W. (2017). Getting Started with pandas. En Python for Data Analysis: Data Wrangling with Pandas, Numpy, and Ipython (2.a ed., p. 123). O'Reilly Media.