Introducción a Pandas

Definitivamente es el sinónimo de "Python para el análisis de datos".

Pandas es una poderosa librería de Python de análisis de datos que se construye encima de numpy que es otra librería que le permite crear arreglos de datos 2d e incluso 3d en Python. El objeto principal de pandas se llama un `DataFrame`. Un `DataFrame` es básicamente una matriz numpy de 2d con filas y columnas, que también tiene etiquetas para columnas y filas.

Pandas ofrece las siguientes estructuras de datos:

Series: Son arrays unidimensionales con indexación (arrays con índice o etiquetados), similar a los diccionarios. Pueden generarse a partir de diccionarios o de listas.

DataFrame: Son estructuras de datos similares a las tablas de bases de datos relacionales como SQL.

Puede crear `DataFrames` de varios formatos de datos de entrada, como CSV, JSON, diccionarios Python, etc. Una vez que haya cargado el marco de datos en Python, puede aplicar varias funciones de análisis y visualización de datos al `DataFrame` y convertir básicamente los datos del `DataFrame` en información valiosa.

El primer ejemplo que vamos a poner va a ser el de definir una estructura de datos "Series" que como ya comentamos es un array de datos unidimensional con idexación. Las "Series" se definen de la siguiente manera:

```
serie = pd.Series(data, index=index)
```

Es decir, que en el primer parámetro le indicamos los datos del array y en el segundo parámetro los índices. Veamos un ejemplo de como crear una estructura "Series" con los integrantes de la selección Española de fútbol que ganó el mudial del año 2010, en el que tenemos como 'data' sus nombres y como índice su dorsal:

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

In [ ]:
```
seleccionEspaña = pd.Series(
    ['Casillas', 'Ramos', 'Pique', 'Puyol', 'Capdevila', 'Xabi Alonso', 'Busquets', 'Xavi Hernand
     'Iniesta', 'Villa'], index=[1, 15, 3, 5, 11, 14, 16, 8, 18, 6, 7])
print ("Jugadores de la Selección Española: \n%s" % seleccionEspaña)
```

En el siguiente caso; en el que no le indiquemos los índices de forma explícita, no generará los índices de forma automática empezando desde el valor cero:

In [ ]:
```
seleccionEspaña = pd.Series(
    ['Casillas', 'Ramos', 'Pique', 'Puyol', 'Capdevila', 'Xabi Alonso', 'Busquets', 'Xavi Hernand
     'Iniesta', 'Villa'])
print ("Jugadores de la Selección Española: \n%s" % seleccionEspaña)
```

También podemos crearnos una estructura de datos "Series" a partir de una lista o de un diccionario. Si la construimos a partir de una lista nos pondrá los índices por defecto y si lo creamos a partir de un diccionario, pondrá como índices las claves. Vamos a ver a continuación un ejemplo de como crear una Serie a partir de un diccionario y además vamos a ver como insertar en esta serie un nuevo elemento:

In [ ]:
```
dictJugadores = {1: 'Casillas', 15: 'Ramos', 3: 'Pique', 5: 'Puyol', 11: 'Capdevila', 14: 'Xabi A
                 16: 'Busquets', 8: 'Xavi Hernandez', 18: 'Pedrito', 6: 'Iniesta', 7: 'Villa'}
jugadores = pd.Series(dictJugadores)
# Insertar un nuevo jugador
jugadores[10] = 'Cesc'
print ("Jugadores de la Selección de Futbol Española: \n\n%s" % jugadores)
```

Vamos a pasar a continuación a ver un ejemplo con la estructura de datos `DataFrame`. Como ya se ha comentado es una estructura de datos similar a una tabla de una base de datos relacionar, una tabla de excel, etc. y como tal se pueden hacer muchas operaciones como las que se harían con consultas a tablas de bases de datos o en excel.

Para construir un `DataFrame` se puede hacer de diferentes formas, como por ejemplo a partir de una lista, de un diccionario, de una Serie, de otro `DataFrame`, leyendo una tabla excel, csv, etc. Vamos a ver a continuación como construiríamos un `DataFrame` con datos de los integrantes de la selección Española de Fútbol:

In [ ]:
```
selecciónEspañola = pd.DataFrame(
    {
        'nombre': ['Casillas', 'Ramos', 'Pique', 'Puyol', 'Capdevila', 'Xabi Alonso', 'Busquets',
                   'Pedrito', 'Iniesta', 'Villa'],
        'posición': ['Portero', 'Lateral Derecho', 'Defensa central', 'Defensa central', 'Lateral
                     'Mediocampista defensivo', 'Mediocampista', 'Extremo izquierdo', 'Extremo
        'equipo': ['Real Madrid', 'Real Madrid', 'FC Barcelona', 'FC Barcelona', 'Villareal', 'Re
                   'FC Barcelona', 'FC Barcelona', 'FC Barcelona', 'FC Barcelona', 'FC Barcelona']
    }, columns=['nombre', 'posición', 'equipo'], index=[1, 15, 3, 5, 11, 14, 16, 8, 18, 6, 7]
)
selecciónEspañola
```

Como resultado tenemos una estructura de dato similar a la de una tabla de una base de datos relacional o de un documento excel o csv. Con esta estructura de datos se pueden hacer muchas operaciones como las que haríamos en una base de datos o en un documento excel.

Por último vamos a ver como insertar un nuevo elemento en este `DataFrame`, que lo haríamos de la siguiente manera con el método `loc()`:

In [ ]:
```
# Insertar un nuevo jugador
selecciónEspañola.loc[10] = ['Cesc', 'Delantero', 'Arsenal']
selecciónEspañola
```

Crear un `DataFrame` desde un archivo `.csv`

In [ ]:
```
# Leer el archivo Income.csv y almacernarlo en un dataframe
df1 = pd.read_csv("../datos/Income_data.csv")
print(df1)
```

In [ ]:
```
# Visualizar el dataframe
df1.head()
```

In [ ]:
```
df1.info()
```

In [ ]:
```
df1.shape
```

Crear un `DataFrame` desde una `url`

In [ ]:
```
# Leer un archivo desde internet
df2 = pd.read_csv("https://github.com/owid/covid-19-data/raw/master/public/data/owid-covid-data.c
```

Almacenar un `DataFrame` como un archivo `.csv`

In [ ]:
```
df2.to_csv('../datos/covid-19.csv')
```

In [ ]:
```
covid_19 = pd.read_csv('../datos/covid-19.csv')
covid_19.head()
```

In [ ]:
```
covid_19.info()
```

In [ ]:
```
covid_19.shape
```

Antes de extraer datos del dataframe, sería una buena práctica asignar una columna con valores únicos como el índice del dataframe. La columna `State` sería una buena opción:

In [ ]:
```
df2 = df1.set_index('State')
df2
```

Ahora, vamos a extraer un subconjunto del dataframe. Aquí está la regla de sintaxis general para subconjunto de porciones de un dataframe:

```
df2.loc[startrow:endrow,startcolumn:endcolumn]
```

Extraer los valores para las filas desde Alaska hasta Arkansas para los años 2005 a 2007:

In [ ]:
```
df2.loc["Alaska":"Arkansas","2005":"2007"]
```

Extraer una columna:

In [ ]:
```
df2.loc[: , "2005"]
```

Tenga en cuenta que cuando extrae una sola fila o columna, obtiene un objeto unidimensional como salida. Eso se llama una Series en pandas. Los valores de la izquierda son sólo etiquetas tomadas del índice del dataframe. Por otro lado, cuando extrajimos porciones de un pandas dataframe como lo hicimos antes, obtuvimos un tipo bidimensional de objeto DataFrame.

Por lo tanto, la fórmula para extraer una columna sigue siendo la misma, pero esta vez no pasamos ningún nombre de índice antes y después del primer colon. Eso le dice a Python que incluya todas las filas. Y pasamos sólo un nombre de columna.

Para extraer sólo una fila se haría a la inversa:

In [ ]:
```
df2.loc["California", : ]
```

Para extraer una celda:

In [ ]:
```
df2.loc["California","2013"]
```

Se pueden aplicar métodos a los subconjuntos. Por ejemplo la media de la columna 2005:

In [ ]:
```
df2.loc[:,"2005"].mean()
```

Indexación basada en la posición

A veces, no se tienen etiquetas de fila o columna. En tal caso usted tendrá que confiar en indexación basada en la posición, la cual que se implementa con `iloc` en lugar de `loc`

In [ ]:
```
df2.iloc[0:3,0:4]
```

Tenga en cuenta que cuando se utilizó indexación basada en etiquetas tanto el inicio y el final de las etiquetas se incluyeron en el subconjunto. Con el rebanado basado en posición, sólo se incluye el índice de inicio. Así, en este caso Alabama tenía un índice de 0, Alaska 1, y Arizona 2. Lo mismo ocurre con las columnas.

Funciones de análisis de datos Pandas

Veamos ahora qué métodos de análisis de datos podemos aplicar a los dataframes de pandas.

Usted sabe que el dataframe es el principal objeto Pandas. Por lo tanto, si tiene algunos datos cargados en un dataframe, podría aplicar métodos para analizar esos datos. Por ejemplo, aquí es cómo se aplica el método `mean` al dataframe en el que hemos estado trabajando

In [ ]:
```
df2.iloc[:,1:].mean()
```

Puede obtener una lista de los métodos disponibles para los dataframe utilizando la función `dir` de Python:

In [ ]:
```
dir(pd.DataFrame)
```

Y puede obtener la descripción de cada método mediante `help`:

In [ ]:
```
help(pd.DataFrame.max)
```

También puede aplicar métodos a las columnas del dataframe

In [ ]:
```
df2.loc[:,"2005"].mean()
```

Tenga en cuenta que en este caso no está aplicando el método de media a un pandas dataframe, sino a un objeto series de pandas:

In [ ]:
```
type(df2.loc[:,"2005"])
```

Añadir columnas a un DataFrame es bastante sencillo:

In [ ]:
```
# Añadir una nueva columna con etiqueta "2014" con los valores de la lista
df2["2014"] = [41325,65789,49002,40489,58934]
df2
```

In [ ]:
```
# Anadir una nueva columna con el ingreso medio de cada estado
df2["Mean"] = df2.iloc[:, 1:].mean(axis=1)
df2
```

El parámetro `axis` le indica a Python que calcule la media a lo largo del eje 1, que significa a lo largo de las columnas. El eje puesto a 0 iría a lo largo de las filas. Vamos a ver cómo calcular la media de cada año y agregarlos como una nueva fila

In [ ]:
```
# Añadir nueva fila con el ingreso medio por año
df2.loc["Mean"] = df2.iloc[:,1:].mean(axis=0)
df2
```

Eliminación de Columnas

In [ ]:
```
covid_19.head()
```

In [ ]:
```
covid_19.drop(columns=['Unnamed: 0'], inplace=True)
```

In [ ]:
```
covid_19
```

Crear un nuevo `DataFrame` con columnas de `covid_19`

In [ ]:
```
columnas = ['iso_code', 'continent', 'location', 'date', 'total_cases', 'new_cases', 'total_deatl
covid_19_casos_muertes = covid_19[columnas].copy()
```

In [ ]:
```
covid_19_casos_muertes.head()
```

In [ ]:
```
covid_19_casos_muertes.tail()
```

In [ ]:
```
covid_19_casos_muertes.info()
```

Convertir columna `date` del tipo `object` a tipo `datetime`

In [ ]:
```
covid_19_casos_muertes['date'] = pd.to_datetime(covid_19_casos_muertes['date'], infer_datetime_fo
```

In [ ]:
```
covid_19_casos_muertes.info()
```

Tratamiento de los valores `NaN`

In [ ]:
```
covid_19_casos_muertes[pd.isna(covid_19_casos_muertes.continent)]
```

In [ ]:
```
covid_19_casos_muertes = covid_19_casos_muertes[covid_19_casos_muertes['continent'].notna()]
```

In [ ]:
```
covid_19_casos_muertes.info()
```

In [ ]:
```
covid_19_casos_muertes.fillna(0, inplace=True)
```

In [ ]:
```
covid_19_casos_muertes.info()
```

Agrupar datos

In [ ]:
```
covid_19_casos_muertes.groupby(['continent'])['new_cases'].sum()
```

In [ ]:
```
covid_19_casos_muertes.groupby(['continent', 'location'])['new_cases'].agg(['mean', 'sum'])
```

In [ ]:
```
casos_venezuela = covid_19_casos_muertes[covid_19_casos_muertes.location == 'Venezuela']['total_
muertes_venezuela = covid_19_casos_muertes[covid_19_casos_muertes.location == 'Venezuela']['tota
plt.plot(casos_venezuela)
plt.plot(muertes_venezuela)
```

In [ ]:
```
new_df = covid_19_casos_muertes.groupby(['continent', 'location'])['new_cases'].agg(['mean', 'su
```

In [ ]:
```
sur_america = new_df[new_df.continent == 'South America']
```

In [ ]:
```
sur_america.sort_values('sum', ascending=False)
```

In [ ]:
```

```