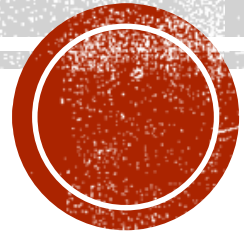
PYTHON - PANDAS

Christian Camilo Urcuqui López, MSc



UNIVERSIDAD ICESI | A OTRO NIVEL

PRESENTACIÓN

Christian Camilo Urcuqui López

Ing. Sistemas, Magister en Informática y Telecomunicaciones

Big Data Professional

Big Data Scientist

Deep Learning Specialization

Grupo de investigación i2t

Líder de investigación y desarrollo

Ciberseguridad y ciencia de datos aplicada

ccurcuqui@icesi.edu.co

COMPETENCIAS

- Utilizar las librerías de Python para proyectos de analítica de datos.
- Numpy
- Pandas
- Matplotlib

OBTENIENDO INFORMACIÓN DE NUESTRO DATAFRAME

Ya hemos visto algunas funciones (head y tail) que nos permiten obtener una primera vista de los datos, ahora veamos otros métodos para conocer la estructura del DataFrame.

```
data = pd.read_csv("../../../Datasets/parks.csv")
```

Observemos la salida de data.info()

OBTENIENDO INFORMACIÓN DE NUESTRO DATAFRAME

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 56 entries, 0 to 55
Data columns (total 6 columns):
Park Code    56 non-null object
Park Name    56 non-null object
State        56 non-null object
Acres        56 non-null int64
Latitude     56 non-null float64
Longitude    56 non-null float64
dtypes: float64(2), int64(1), object(3)
memory usage: 2.7+ KB
```

Cantidad de registros y variables

Tipos de variables

Memoria utilizada por el objeto

Pandas

TIPOS DE VARIABLES

- **Variables cuantitativas**, sus valores son numéricos y pueden ser contados o medidos, por ejemplo, ventas netas de una compañía.
  - **Variables discretas**, es una variable numérica que usualmente se obtiene a través del conteo y solamente puede tomar valores específicos de un conjunto, por ejemplo, el número de personas en una ciudad o el número de quejas de los clientes.
  - **Variables continuas**, son variables numéricas que pueden tomar un valor (infinito/decimal) entre dos valores numéricos cualquiera. Usualmente, esta variable se obtiene a partir de mediciones, por ejemplo, la temperatura de un paciente.

TIPOS DE VARIABLES

- **Variables cualitativas**, conocidos también como variables categóricas, sus valores pueden ser contados pero no medidos.
  - **Variables nominales**, son valores que presentan a una categoría y no cuentan con un orden. Estos valores pueden ser contados pero no pueden ser ni medidos y ni ordenados, por ejemplo, género de música y categorías de productos.
  - **Variables ordinales**, son valores numéricos que pueden ser discretos o continuos y que están ya sea ordenadas o jerarquizadas.
  - **Variables binarias**, sus valores hacen parte únicamente a dos categorías que generalmente son opuestos, por ejemplo, 1/0 y verdadero/falso.

TIPOS DE VARIABLES

- **Variables independientes**, sus valores no dependen de otra variable pero posiblemente si puedan influenciar a otras.
- **Variables dependientes**, este tipo de variable si depende de otras variables.
- **Variable aleatoria**, es una variable que puede asumir un valor de un rango de valores basado en la probabilidad.

CAMBIANDO TIPOS DE LAS VARIABLES

Obtengamos los tipos de cada variable en nuestro DataFrame.

```
data = pd.read_csv("../../../Datasets/parks.csv“, index_col=['Park Code'],
encoding='utf-8')

data.dtypes
```

```
Park Name    object
State        object
Acres         int64
Latitude    float64
Longitude   float64
dtype: object
```

CAMBIANDO TIPOS DE LAS VARIABLES

Pandas type	Python type	Numpy type	Usage
object	str	string_, unicode_	Text
int64	int	int_, int8, int16, int32, int64, unit8, unit16, unit32, unit64	Integer numbers
float64	float	float_, float16, float32, float64	Floating point numbers
bool	bool	bool_	True / False values
datetime64[ns]	NA	datetime64[ns]	Date and time values
timedelta[ns]	NA	NA	Differences between two datetimes
category	NA	NA	Finite list of text values

Pandas

https://pandas.pydata.org/pandas-docs/stable/getting_started/basics.html

CAMBIANDO TIPOS DE LAS VARIABLES

Biodiversity in National Parks

The National Park Service publishes a database of animal and plant species identified in individual national parks and verified by evidence — observations, vouchers, or reports that document the presence of a species in a park. All park species records are available to the public on the National Park Species portal; exceptions are made for sensitive, threatened, or endangered species when widespread distribution of information could pose a risk to the species in the park.

- **Park Code National Parks Service**: park code.
- **Park Name Office**: park name.
- **State**: US state(s) in which the park is located. Comma-separated.
- **Acres**: Size of the park in acres.
- **Latitude**: Latitude of the park (centroid).
- **Longitude**: Longitude of the park (centroid).

https://www.kaggle.com/nationalparkservice/park-biodiversity

CAMBIANDO TIPOS DE LAS VARIABLES

- **Park Code National Parks Service**: park code. → cuantitativa
- **Park Name Office**: park name. → cualitativa
- **State**: US state(s) in which the park is located. Comma-separated → cualitativa
- **Acres**: Size of the park in acres. → cuantitativa
- **Latitude**: Latitude of the park (centroid). → cuantitativa
- **Longitude**: Longitude of the park (centroid). → cuantitativa

Pandas

https://www.kaggle.com/nationalparkservice/park-biodiversity

CAMBIANDO TIPOS DE LAS VARIABLES

1. Podemos utilizar el método astype("el tipo a cambiar") de un objeto tipo DataFrame con el fin de cambiar el tipo de una columna en especifico. También, podemos utilizar el método sobre el DataFrame para afectar a todas las variables, en el siguiente ejemplo cambiaremos solo la columna **State**.

```
#data.State = data.State.astype("category")
data['State'] = data['State'].astype("category")
data.dtypes
```

Pandas

CAMBIANDO TIPOS DE LAS VARIABLES

2. Pandas también provee dos métodos que nos ayudarán a cambiar los tipos de una variable
   - to_numeric()
   - to_datetime()

```
serie_dates = pd.Series(["1970-01-01", "1980-01-01", "2019-03-01", "2019-02-16"])
pd.to_datetime(serie_dates, format='%Y-%m-%d')
```

```
0   1970-01-01
1   1980-01-01
2   2019-03-01
3   2019-02-16
dtype: datetime64[ns]
```

CAMBIANDO TIPOS DE LAS VARIABLES

```
serie_dates = pd.Series(["1970-01-01", "1980-01-01", "2019-03-01", "2019-02-16"])
pd.to_datetime(serie_dates, format='%Y-%m-%d')
```

Directive	Meaning	Example
%a	Weekday as locale's abbreviated name.	Sun, Mon, …, Sat (en_US);
%A	Weekday as locale's full name.	Sunday, Monday, …, Saturday (en_US);
%w	Weekday as a decimal number, where 0 is Sunday and 6 is Saturday.	0, 1, …, 6
%d	Day of the month as a zero-padded decimal number.	01, 02, …, 31
%b	Month as locale's abbreviated name.	Jan, Feb, …, Dec (en_US);
%B	Month as locale's full name.	January, February, …, December (en_US);
%m	Month as a zero-padded decimal number.	01, 02, …, 12
%Y	Year with century as a decimal number.	0001, 0002, …, 2013, 2014, …, 9998, 9999

https://docs.python.org/3/library/datetime.html#strftime-and-strptime-behavior

RENOMBRANDO COLUMNAS

En algunas ocasiones podemos encontrarnos nombres con espacios y mayúsculas, es decir, con particularidades que deben ser tratadas.

Para nuestro DataFrame de parks.csv contamos con columnas que tienen espacios, y tanto mayúsculas y minúsculas. Llevémoslos a un mismo estándar aplicando los métodos de **str.**

RENOMBRANDO COLUMNAS

Una forma fácil de realizar los cambios en los nombres de las columnas es con el método **rename()**

```
data = data.rename({'Park Name': 'park_name', 'State': 'state', 'Acres': 'acres', 'Latitude': 'latitude',
                    'Longitude': 'longitude'}, axis='columns')
```

RENOMBRANDO COLUMNAS

Para nuestro DataFrame de parks.csv contamos con columnas que tienen espacios, tanto mayúsculas y minúsculas. Llevémoslos a un mismo estándar aplicando los métodos de **str**.

```
data.columns.str.replace(' ', '_')
```

```
Index(['Park_Name', 'State', 'Acres', 'Latitude', 'Longitude'], dtype='object')
```

```
df_park.columns = df_park.columns.str.replace(' ', '_').str.lower()
```

```
Index(['park_name', 'state', 'acres', 'latitude', 'longitude'], dtype='object')
```

Pandas

OBTENIENDO INFORMACIÓN DE NUESTRO DATAFRAME

Estadísticos descriptivos

```
data = pd.read_csv("../../../Datasets/parks.csv")
data.describe()
```

	acres	latitude	longitude
count	5.600000e+01	56.000000	56.000000
mean	9.279291e+05	41.233929	-113.234821
std	1.709258e+06	10.908831	22.440287
min	5.550000e+03	19.380000	-159.280000
25%	6.901050e+04	35.527500	-121.570000
50%	2.387645e+05	38.550000	-110.985000
75%	8.173602e+05	46.880000	-103.400000
max	8.323148e+06	67.780000	-68.210000

Pandas

OBSERVACIÓN

Note que el método describe() por defecto nos trae los estadísticos descriptivos solo para las variables cuantitativas.

OBTENIENDO INFORMACIÓN DE NUESTRO DATAFRAME

Para obtener información acerca de las variables categóricas debemos tener el método con el siguiente parámetro:

data.describe(include="all")

	park_name	state	acres	latitude	longitude
count	56	56	5.600000e+01	56.000000	56.000000
unique	56	27	NaN	NaN	NaN
top	Cuyahoga Valley National Park AK	NaN	NaN	NaN	NaN
freq	1	8	NaN	NaN	NaN
mean	NaN	NaN	9.279291e+05	41.233929	-113.234821
std	NaN	NaN	1.709258e+06	10.908831	22.440287
min	NaN	NaN	5.550000e+03	19.380000	-159.280000
25%	NaN	NaN	6.901050e+04	35.527500	-121.570000
50%	NaN	NaN	2.387645e+05	38.550000	-110.985000
75%	NaN	NaN	8.173602e+05	46.880000	-103.400000
max	NaN	NaN	8.323148e+06	67.780000	-68.210000

OBTENIENDO INFORMACIÓN DE NUESTRO DATAFRAME

Para obtener información acerca de las variables categóricas debemos tener el método con el siguiente parámetro:

```
data.describe(include=["category"])
```

	state
count	56
unique	27
top	AK
freq	8

OBTENIENDO INFORMACIÓN DE NUESTRO DATAFRAME

Para variables categóricas podemos también obtener información sobre la frecuencia de los valores

data.state.value_counts()

```
AK          8
CA          7
UT          5
CO          4
WA          3
AZ          3
FL          3
TX          2
SD          2
HI          2
MI          1
AR          1
CA, NV      1
KY          1
ME          1
WY, MT, ID  1
MN          1
MT          1
WY          1
NM          1
NV          1
OH          1
OR          1
SC          1
TN, NC      1
VA          1
ND          1
Name: state, dtype: int64
```

INDEXACIÓN, SELECCIÓN Y ASIGNACIÓN

Podemos acceder a una variable como si fuera un atributo del objeto. Un parque nacional en US podría tener una propiedad *nombre*, podemos obtener los datos así:

```
data.park_name
```

Otra manera que podemos acceder a los datos es como si fuera un directorio en Python, es decir, la llave seria el nombre de la variable

```
data['park_name‘]
```

Pandas

INDEXACIÓN, SELECCIÓN Y ASIGNACIÓN

Note que el tipo de objeto que obtenemos del anterior proceso es de tipo Serie.

```
type(data.park_name)
```

```
pandas.core.series.Series
```

Ahora si deseamos obtener uno o más valores de la serie solo le agregamos unos índices asociados a los registros, es decir…

```
data.park_name[0] # primer registro
```

```
data['park_name'][0:8]  # slice – los primeros ocho registros
```

INDEXACIÓN, SELECCIÓN Y ASIGNACIÓN

Otra forma de acceder a los datos es a través de los métodos indexación que Pandas nos provee, entre estos encontramos dos:

- **iloc**, obtenemos los datos a partir de su posición numérica.
- **loc**, obtenemos los datos con los identificadores.

Ambos métodos utilizan una convención de [fila, columna] totalmente opuesto al enfoque nativo en Python

INDEXACIÓN, SELECCIÓN Y ASIGNACIÓN

- **iloc**, obtenemos los datos a partir de su posición numérica.

Ambos métodos utilizan una convención de [fila, columna] totalmente opuesto al enfoque nativo en Python

```
data.iloc[0] # obtenemos el primer registro

data.iloc[1, 2:4] # obtenemos del segundo registro las columnas 2 y 3

data.iloc[1:3, [1,2]] # podemos pasar una lista
```

INDEXACIÓN, SELECCIÓN Y ASIGNACIÓN

- **loc**, obtenemos los datos con los identificadores.

Ambos métodos utilizan una convención de [fila, columna] totalmente opuesto al enfoque nativo en Python.

data.index # obtenemos los índices, es decir, la variable "Park Code"

data.loc['ACAD', 'latitude'] # del registro 'ACAD' obtenemos la latitude

data.loc[["ACAD","ARCH"]] # obtenemos dos registros

data.loc[:, ["latitude", "acres"]]# la latitude y acres de todos los registros

Pandas

INDEXACIÓN, SELECCIÓN Y ASIGNACIÓN

Finalmente ya es cuestión del programador el proceso de indexación y selección de los datos.

```
data.iloc[1:3].loc[: ,['state','park_name']]
```

SELECCIÓN CONDICIONAL

Podemos realizar filtros en nuestro conjunto de datos a partir de condicionales en el DataFrame.

Supongamos que estamos interesados en conocer solo los datos que pertenecen al estado de 'CA'

```
data.loc[data.state == "CA"]
```

Supongamos que estamos interesados en conocer solo los datos que pertenecen al estado de 'CA' y tienen un tamaño (acres) mayor o igual a 700000

```
data.loc[(data.state == "CA“) & (data.acres >= 700000)]
```

Pandas

SELECCIÓN CONDICIONAL

Pandas provee otros métodos para selección condicional, un ejemplo de estos es **isin()**. isin selecciona los datos cuyo valor pertenece a una lista.

Supongamos que estamos interesados obtener los datos de los estados de 'CA' y 'AK'

```
data[data.state.isin(['AK', 'CA'])]
```

Pandas

ASIGNACIÓN DE DATOS

La asignación de datos a una DataFrame es relativamente sencilla, podemos aplicar los pasos vistos y asignar los registros a un valor constante.

Supongamos que deseamos cambiar los registros que tienen un estado 'AK' por NaN

```
data[data.state =='AK'] = np.nan
# el valor NaN lo encontramos en Numpy
```

Pandas

SELECCIÓN CONDICIONAL NAN

Para la selección de registros con o sin NaN podemos encontrar los siguientes:

- **isnull()**.Nos entrega los datos con NaN
- **notnull()**. Nos entrega los datos que no son NaN

```
# obtenemos las dimensiones de los registros que no son NaN
print(data.info())
print(data[data.state.notnull()].shape)
```

```
# obtenemos las dimensiones de los registros que son NaN
print(data[data.state.isnull()].shape)
```

Pandas

Python For Data Science *Cheat Sheet*

Python Basics

Learn More Python for Data Science Interactively at www.datacamp.com

Variables and Data Types

Variable Assignment

```
>>> x=5
>>> x
5
```

Calculations With Variables

```
>>> x+2
7
>>> x-2
3
>>> x*2
10
>>> x**2
25
>>> x%2
1
>>> x/float(2)
2.5
```

Sum of two variables
Subtraction of two variables
Multiplication of two variables
Exponentiation of a variable
Remainder of a variable
Division of a variable

Types and Type Conversion

`str()`	`'5', '3.45', 'True'`	Variables to strings
`int()`	`5, 3, 1`	Variables to integers
`float()`	`5.0, 1.0`	Variables to floats
`bool()`	`True, True, True`	Variables to booleans

Asking For Help

```
>>> help(str)
```

Lists

Also see NumPy Arrays

```
>>> a = 'is'
>>> b = 'nice'
>>> my_list = ['my', 'list', a, b]
>>> my_list2 = [[4,5,6,7], [3,4,5,6]]
```

Selecting List Elements

Index starts at 0

Subset

```
>>> my_list[1]
>>> my_list[-3]
```

Select item at index 1
Select 3rd last item

Slice

```
>>> my_list[1:3]
>>> my_list[1:]
>>> my_list[:3]
>>> my_list[:]
```

Select items at index 1 and 2
Select items after index 0
Select items before index 3
Copy my_list

Subset Lists of Lists

```
>>> my_list2[1][0]
>>> my_list2[1][:2]
```

my_list[list][itemOfList]

List Operations

```
>>> my_list + my_list
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list * 2
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list2 > 4
True
```

List Methods

`>>> my_list.index(a)`	Get the index of an item
`>>> my_list.count(a)`	Count an item
`>>> my_list.append('!')`	Append an item at a time
`>>> my_list.remove('!')`	Remove an item
`>>> del(my_list[0:1])`	Remove an item
`>>> my_list.reverse()`	Reverse the list
`>>> my_list.extend('!')`	Append an item
`>>> my_list.pop(-1)`	Remove an item
`>>> my_list.insert(0,'!')`	Insert an item
`>>> my_list.sort()`	Sort the list

Libraries

Import libraries

```
>>> import numpy
>>> import numpy as np
```

Selective import

```
>>> from math import pi
```

pandas — Data analysis
scikit-learn — Machine learning
NumPy — Scientific computing
matplotlib — 2D plotting

Install Python

ANACONDA — Leading open data science platform powered by Python
spyder — Free IDE that is included with Anaconda
jupyter — Create and share documents with live code, visualizations, text, ...

Numpy Arrays

Also see Lists

```
>>> my_list = [1, 2, 3, 4]
>>> my_array = np.array(my_list)
>>> my_2darray = np.array([[1,2,3],[4,5,6]])
```

Selecting Numpy Array Elements

Index starts at 0

Subset

```
>>> my_array[1]
2
```

Select item at index 1

Slice

```
>>> my_array[0:2]
array([1, 2])
```

Select items at index 0 and 1

Subset 2D Numpy arrays

```
>>> my_2darray[:,0]
array([1, 4])
```

my_2darray[rows, columns]

Numpy Array Operations

```
>>> my_array > 3
array([False, False, False,  True], dtype=bool)
>>> my_array * 2
array([2, 4, 6, 8])
>>> my_array + np.array([5, 6, 7, 8])
array([6, 8, 10, 12])
```

Numpy Array Functions

`>>> my_array.shape`	Get the dimensions of the array
`>>> np.append(other_array)`	Append items to an array
`>>> np.insert(my_array, 1, 5)`	Insert items in an array
`>>> np.delete(my_array,[1])`	Delete items in an array
`>>> np.mean(my_array)`	Mean of the array
`>>> np.median(my_array)`	Median of the array
`>>> my_array.corrcoef()`	Correlation coefficient
`>>> np.std(my_array)`	Standard deviation

DataCamp
Learn Python for Data Science Interactively

Strings

```
>>> my_string = 'thisStringIsAwesome'
>>> my_string
'thisStringIsAwesome'
```

String Operations

```
>>> my_string * 2
'thisStringIsAwesomethisStringIsAwesome'
>>> my_string + 'Innit'
'thisStringIsAwesomeInnit'
>>> 'm' in my_string
True
```

String Operations

Index starts at 0

```
>>> my_string[3]
>>> my_string[4:9]
```

String Methods

`>>> my_string.upper()`	String to uppercase
`>>> my_string.lower()`	String to lowercase
`>>> my_string.count('w')`	Count String elements
`>>> my_string.replace('e', 'i')`	Replace String elements
`>>> my_string.strip()`	Strip whitespace from ends

BIBLIOGRAFÍA

- Lutz, M. (2013). *Learning Python: Powerful Object-Oriented Programming*. " O'Reilly Media, Inc.".