Introducción a la programación con el lenguaje Python 3 y su aplicación básica en el Análisis de Datos

Miguel A. Astor y Adelis Nieves

EVI - CoNCISa 2018

Agenda

Introducción

Parte II del tutorial práctico que introduce al manejo básico de datos con Python 3, Jupyter y Pandas

¿Qué es Análisis de Datos?

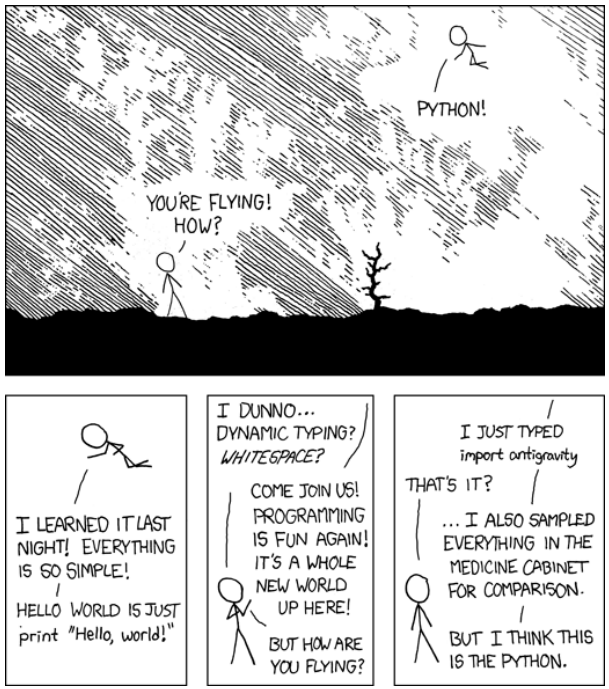
TRABAJO PRODUCTIVO CON LOS DATOS

- Interactuar con el mundo exterior
- Preparación
- Transformación
- Modelación y cáculo
- Presentación

¿Por qué Python para Análisis de Datos?

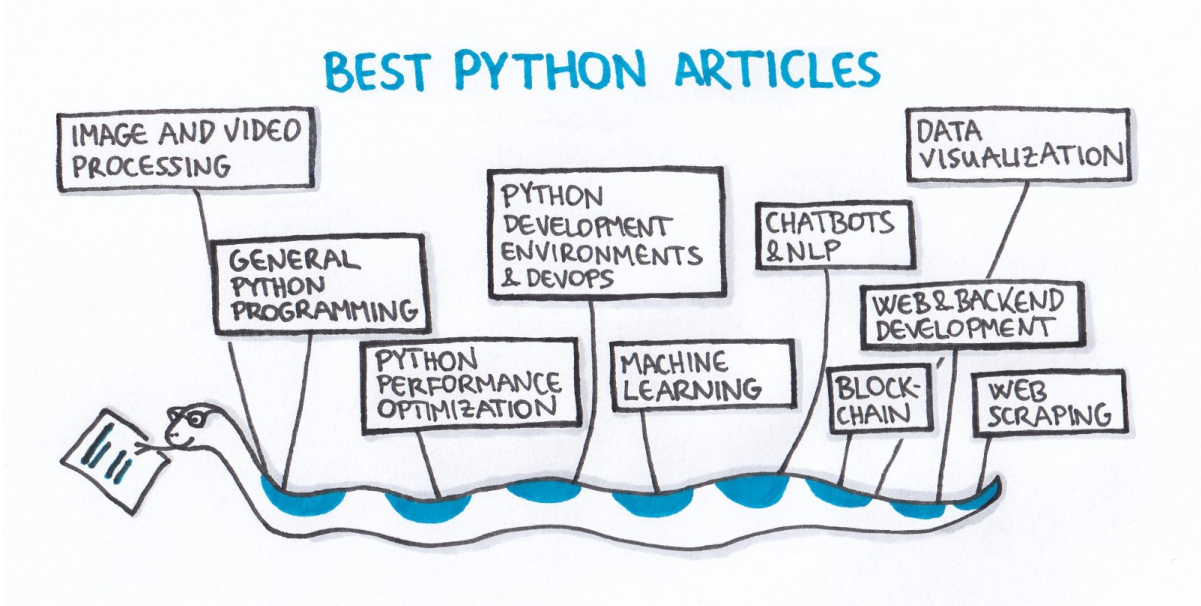
FÁCIL ENAMORARSE DE ÉL

- Fácil, intuitivo y legible.
- Análisis de datos + Apps web | Estadística + Producción de base de datos
- Escribir y buscar errores más sencillo por su tipo de sintáxis
- Múltiples paquetes e IDE's para el análisis de datos
- Comunidad de cómputo científico activa (industria + academia)
- Soporte de bibliotecas para manipulación de datos



¿QÚE BIBLIOTECAS USAREMOS?

- NumPy
- Pandas
- Matplotlib



¿Qué es el cuaderno de Jupyter?

Aplicación web libre que permite crear y compartir documentos con código interactivo, ecuaciones, visualización y texto narrativo.

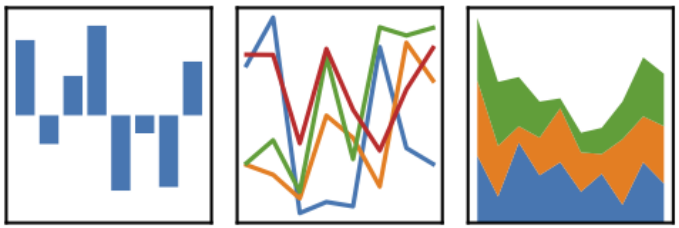
En él podemos limpiar y transformar datos, hacer simulación numérica, modelación estadística, visualización de datos, machine learnig, entre otras cosas.

¿Qué es Pandas?

Biblioteca de software libre hecha como extensión de NumpPy para manipular y analizar datos con el lenguaje de programación Python.

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



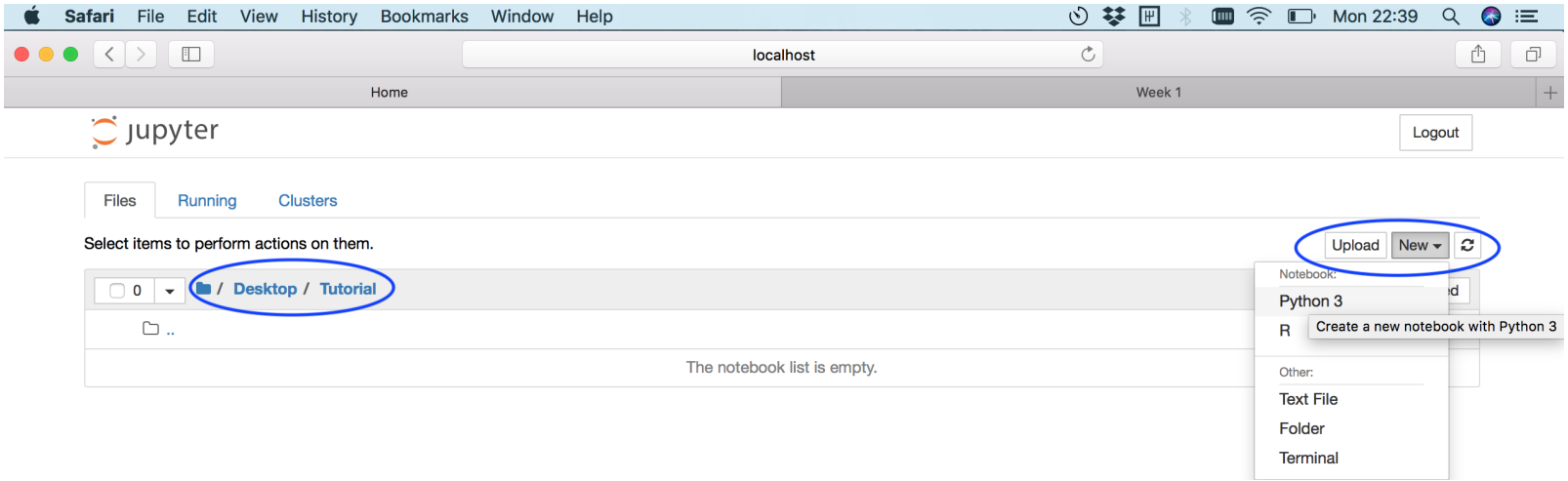
Con Pandas podemos tener un flujo de trabajo con estructuras de datos y operaciones para el manejo de tablas numéricas y series temporales.

Fundamentos de Python en Jupyter

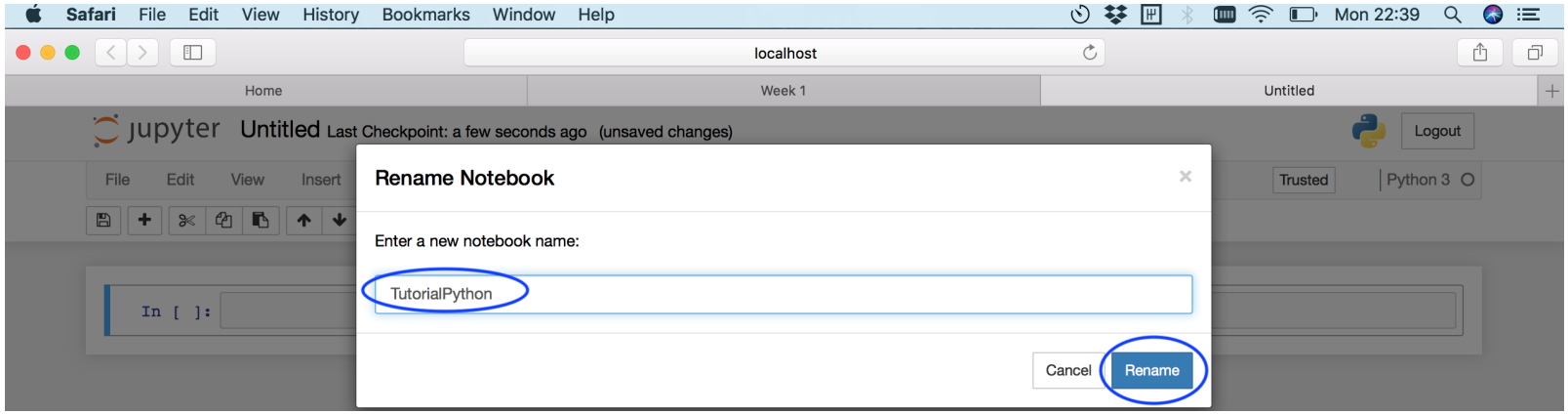
Aquí veremos cómo crear un cuaderno en Jupyter, trabajar con funciones de Python, tipos y secuencias y un poco de la biblioteca NumPy.

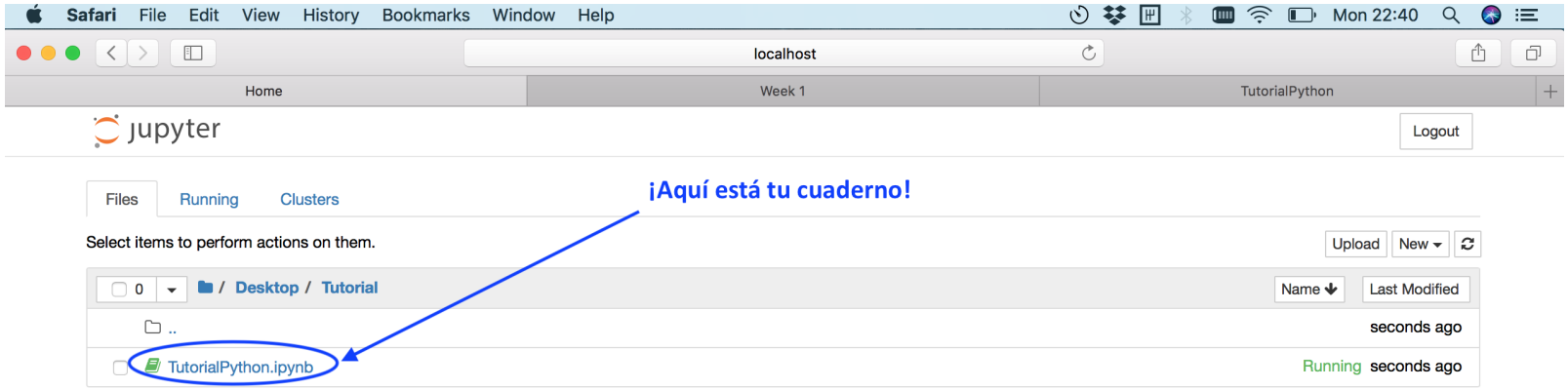
Abriendo un cuaderno en Jupyter

Primero, abre la **Terminal** y escribe **jupyter notebook**. Ahora, vamos a crear un cuaderno de Jupyter con el nombre **TutorialPython.ipynb** siguiendo los siguientes pasos:



Abriendo un cuaderno en Jupyter





En este cuaderno podremos luego hacer nuestro ejercicio práctico de final de curso ☺

Trabajando con funciones

En Python las **funciones universales** ejecutan operaciones elementales entre datos y arreglos. Por ejemplo:

- `add_numbers` ejecuta la suma de dos o más números

```
def add_numbers(x,y):
    return x + y
add_numbers(1,2)
3
```

- Si queremos agregar un tercer parámetro

```
def add_numbers(x,y,z= None):
    if(z==None)
        return x + y
    else:
        return x + y + z
print(add_numbers(1,2))
print(add_numbers(1,2,3))
3
6
```

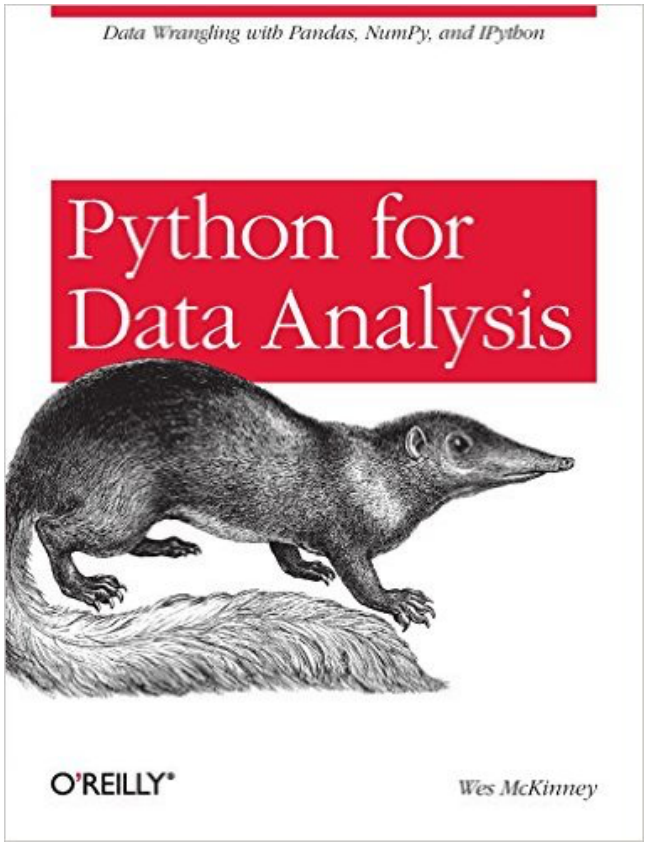
Más funciones

En Python tenemos una lista amplia de **funciones binarias**, entre las más utilizadas tenemos:

- `min( )` devuelve el elemento más pequeño de un iterable o el más pequeño entre dos o más argumentos.
- `max( )` devuelve el elemento más grande de un iterable o el más grande entre dos o más argumentos.
- `pow( )` devuelve `x` a la potencia de `y`.
- `x %y` devuelve el resto de la división de `x` y `y`.
- `x//y` devuelve la parte entera del cociente entre `x` y `y`.

Bibliografía para la búsqueda

En el libro **Python for Data Analysis** de Wes MacKinney podemos encontrar mucha información sobre funciones y muchos ejemplos! Es una buena bibliografía para comenzar en el lenguaje Python.



Tipos y secuencias

Con la función `type( )` de Python en Jupyter podemos devolver el tipo de algún objeto. Veamos que devuelven en el cuaderno **Modulo1.ipynb** las siguientes instrucciones:

- `type('Esto es una cadena de caracteres')`
- `type(None)`
- `type(10000)`
- `type(10000.0)`
- `type(add_numbers)`

Tipos y secuencias

En Python se puede hacer un montón con las secuencias. Las listas son secuencias muy utilizadas, y a este tipo de objeto le podemos:

- Agregar elementos (`append`)
- Iterar sobre cada elemento de la misma (`for, while`)
- Concatenar otras listas (`'+'`)
- Repetir listas (`'*'`)
- Picar o rebanar (`'[ ]'`)

Veamos en el cuaderno de Jupyter **Modulo1.ipynb** algunos ejemplos ☺.

Leer y escribir archivos CSV

En los cuadernos de Jupyter también podemos leer/escribir archivos `CSV`, usando:

- `import csv`
- `with open( ) ... as`
- `reader( )`
- `print( )`

¡Vamos a leer un archivo en formato `txt` como un archivo `CSV` en nuestro cuaderno de Jupyter!

Numerical Python (NumPy)

`NumPy` constituye una biblioteca de funciones matemáticas de alto nivel para operar con vectores o matrices. Con ella podemos:

- Crear arreglos (`np.array`)
- Hallar dimensiones del arreglo (`shape`)
- Ordenar (`np.arange`)
- Cambiar forma (`reshape`)
- Operar (`'+'`, `'*'`, `'-'`, `'/'`, `'**'`)

Revisemos esta lista y otras funciones que podemos utilizar de `NumPy` en nuestro cuaderno **Modulo1.ipynb** ☺.

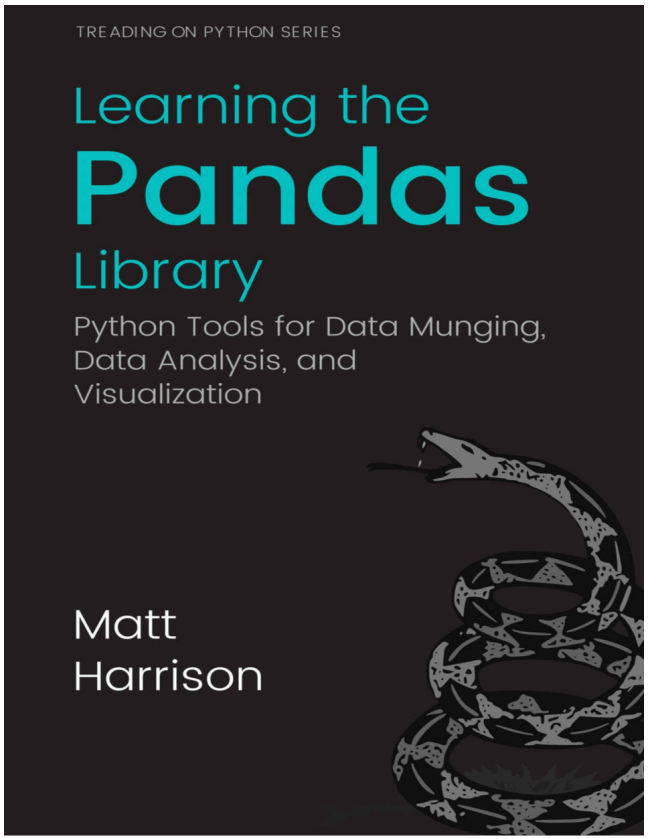
Introducción a Pandas en Jupyter

Ahora que ya sabemos un poco más del uso del cuaderno de Jupyter a través de los fundamentos de Python, estudiaremos el uso de la biblioteca **Pandas** para manejar tablas de datos.

Abre tu cuaderno de Jupyter **Modulo2.ipynb** para empezar a trabajar con Pandas ☺.

Bibliografía para la búsqueda

En el libro **Learning the Pandas Library** de Matt Harrison encontraremos mucha información y ejemplos sobre el uso de la librería Pandas. Es una excelente bibliografía para comenzar a aprender.



Estructura de datos `Serie`

Una `Serie` es un objeto similar a una matriz unidimensional que contiene un arreglo de datos. También puede pensarse como un cruce entre una `lista` y un `diccionario`, es decir, datos etiquetados.

```
animales = ['Tigre', 'Oso', 'Camello']
pd.Series(animales)
```

```
0     Tigre
1       Oso
2   Camello
dtype: object
```

```
numeros = [1, 2, 3]
pd.Series(numeros)
```

```
0    1
1    2
2    3
dtype: int64
```

Estructura de datos `Serie`

`NumPy` y `Pandas` también manejan datos faltantes. En Python tenemos el tipo `None` para indicarlo. Si tenemos una lista de números, `Pandas` automáticamente convierte este valor `None` en un valor designado como `NaN`, que significa **Not a Number**.

```
numeros = [1, 2, None]
pd.Series(numeros)
0    1.0
1    2.0
2    NaN
dtype: float64
```

Note que `NaN` no es `None`, veamos qué significa esto en nuestro cuaderno de Jupyter.

Estructura de datos `Serie`

Para crear una `Serie` en Pandas podemos utilizar un `diccionario` con sus claves. En este caso, los índices de la serie son las claves del diccionario.

```
1  deportes = {'Capoeira': 'Brasil',
2              'Rayuela': 'Chile',
3              'Pelota Vasca': 'País Vasco',
4              'Béisbol': 'Cuba',
5               'Rugby': 'Gales',
6               'Golf': 'Escocia',
7               'Corrida de Toros': 'España',
8               'Sumo': 'Japón'}
9  s = pd.Series(deportes)
10 s
```

```
Béisbol                 Cuba
Capoeira               Brasil
Corrida de Toros       España
Golf                  Escocia
Pelota Vasca        País Vasco
Rayuela                 Chile
Rugby                   Gales
Sumo                    Japón
dtype: object
```

En el cuaderno **Modulo2.ipynb** podemos ver otro ejemplo y cómo hacer una modificación de la `Serie` ☺.

Búsqueda en una `Serie`

Se pueden hacer búsquedas en una `Serie` con los siguientes atributos:

`iloc`

Se utiliza para hacer búsqueda por posición de índice (ubicación numérica) empezando desde 0.

`loc`

Se utiliza para hacer búsqueda por etiqueta de índice

Usemos el ejemplo de deportes nacionales y sus respectivos países (la `Serie` s) para trabajar con los atributos `iloc` y `loc`.

Trabajando con la `Serie`

Una forma de trabajar con una `Serie` en Pandas es iterar sobre el conjunto de datos e invocar una operación de interés.

```
0     105.0
1     223.0
2       5.0
3     102.0
4      27.0
5    -126.0
dtype: float64
```

```
total= 0
for elemento in s:
    total+= elemento
print(total)
```

```
336.0
```

Trabajando con la `Serie`

- NumPy ofrece las **funciones universales** (binarias y unarias) para trabajar con las series haciendo cálculos vectorizados.
- NumPy y Pandas tienen el `broadcasting`, se puede aplicar una operación a cada valor de la serie y modificarla.
- Los cuadernos de Jupyter tienen funciones mágicas que pueden ser útiles. Basta con tipear el símbolo % y la tecla `Tab` para obtener una lista de estas funciones.

Veamos algunos usos de estas bondades en el cuaderno de Jupyter **Modulo2.ipynb**.

Estructura de datos `DataFrame`

DataFrame

Tabla de datos o arreglo bidimensional con etiquetas en los ejes.

¿Qué podemos hacer con el DataFrame?

- Crearlos a través de diccionarios o archivos.
- Extraer información con `iloc` y `loc`.
- Chequear el tipo de dato usando `type`.
- Transponer la tabla de datos con el atributo `T`.
- Combinar `T` y `loc` para seleccionar columnas y la notación `[ ]` para rebanarlo o picarlo.
- Eliminar datos con la función `drop`.

En **Modulo2.ipynb** podemos empezar a poner en práctica esto ☺.

Lectura de `DataFrame`

Pandas tiene un conjunto de funciones para leer datos en tablas como un objeto DataFrame.

Función	Acción
`read_csv`	Carga datos delimitados de un archivo, usa (,) como delimitador.
`read_table`	Carga datos delimitados de un archivo, usa (\t)*comodelimitador*.
`read_fwf`	Lee datos en formato de columna de ancho fijo.
`read_clipboard`	Versión de `read_table` para datos en el porta papeles.

Los más utilizados son `read_csv` y `read_table`. Veamos algunos ejemplos en el cuaderno **Modulo2.ipynb**.

Búsqueda en un `DataFrame`

Se pueden hacer búsquedas en un `DataFrame` con los siguientes métodos:

Máscara Booleana

Se utiliza para hacer búsqueda dando una condición (**si** ··· **entonces**) sobre el `DataFrame` que devuelve un objeto de la misma forma con entradas de `True` si se cumple la condición y `False` si no.

`DataFrame.where()`

Este método toma una máscara booleana como condición en el argumento, la aplica al `DataFrame`, y devuelve un `DataFrame` de la misma forma.

Búsqueda en un `DataFrame`

`count()`

Es una función que devuelve el número de ocurrencias para cada columna o fila y excluye `NaN` y `None`.

Usemos un archivo `CSV` para leerlo y emplear este tipo de lecturas en el cuaderno Jupyter.

Manipulación y Análisis de Datos con Python

Pues ya somos **Masters** en Jupyter y Pandas ☺.
Ahora aprenderemos a manipular los datos y sacar alguna información de ellos 😄.

Abre tu cuaderno de Jupyter **Modulo3.ipynb** para empezar a jugar con los datos.

Pre-procesamiento de datos

Técnicas de pre-procesamiento

- Identificar y manejar datos faltantes.
- Transformar los datos.
- Normalización de datos (centrar/escalar).

Manejo de datos faltantes

Hay muchas maneras de manejar los datos faltantes, entre ellas pordemos considerar:

- Chequear con la fuente de recolección de datos.
- Remover los datos faltantes encontrados (si no son muchos).
  - Remover la variable completa.
  - Remover solo la entrada faltante.
- Reemplazar los valores faltantes.
  - Reemplazar por el promedio.
  - Reemplazar por su frecuencia.
  - Reemplazar basado en otras funciones.
- **En algunos casos dejarlos como valores faltantes.**

Remover datos faltantes

`dropna()`

Esta función remueve valores faltantes. Se pueden seleccionar remover filas o columnas que contienen valores faltantes, como `NaN`. Es necesario especificar el argumento `axis=0` para

remover filas o `axis=1` para remover columnas. El argumento

`inplace` es muy importante pues cuando tiene valor `True` aplica los cambios de forma inmediata sobre el `DataFrame`.

Reemplazar datos faltantes

`replace()`

Esta función permite reemplazar valores faltantes en el `DataFrame` por valores nuevos. Normalmente se reemplazan los valores `NaN` con el promedio de la variable.

Ejemplo:

- `media = df['<etiqueta>'].mean()`
- `df['<etiqueta>'].replace(np.nan, media), inplace = True`

Veamos algunos ejemplos en el cuaderno de Jupyter **Modulo3.ipynb** para entender el funcionamiento de estos métodos ☺.

Transformando los datos

Hasta ahora, nos hemos preocupado limpiar y filtrar datos. La reorganización y combinación de los datos son otra clase de operaciones importantes. Para ello, contamos con algunos métodos de NumPy:

`merge()`

Conecta filas en el `DataFrame` basado en una o más teclas. Para los conocedores de SQL esta función hace unión de bases de datos por columnas o índices.

`concat()`

Pega o apila objetos a lo largo de un eje.

Normalización de los datos

Escala de característica simple

$$x_{nuevo} = \frac{x_{viejo}}{x_{maximo}}$$

Mínimo - Máximo

$$x_{nuevo} = \frac{x_{viejo} - x_{minimo}}{x_{maximo} - x_{minimo}}$$

Puntaje estándar

$$x_{nuevo} = \frac{x_{viejo} - \mu}{\sigma}$$

Estadística descriptiva

La **Estadística descriptiva** sirve para explorar las características básicas de los datos. NumPy ofrece funciones y gráficos para describir los datos.

describe()

Calcula estadística básica para todas las variables numéricas: valores extremos, media, varianza, desviación estándar, cuantiles y número total de datos para cada variable.

value_counts()

Esta función resume la cantidad de variables categóricas (o categorías) en el conjunto de datos.

En nuestro cuaderno de Jupyter **Modulo3.ipynb** tenemos un resumen estadístico de los datos `'Automobile_data.csv'`.

Estadística descriptiva

Con la librería `Matplotlib` de Python tenemos acceso a herramientas para graficar:

Gráfico de caja (o Boxplot)

Muestra la media, percentil 75, percentil 25, extremos superior e inferior, y finalmente los datos atípicos como puntos aislados fuera de los valores extremos.

Histograma

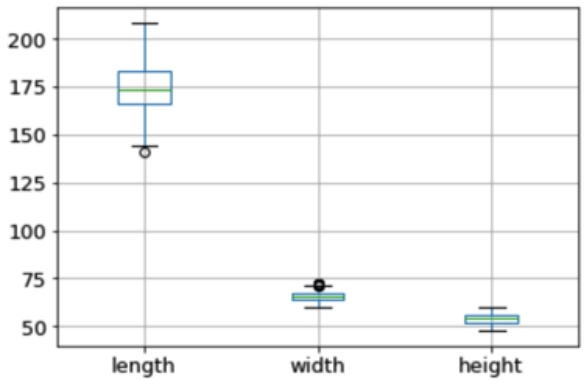
Gráfico de barras que da una visualización discretizada de la frecuencia de los datos.

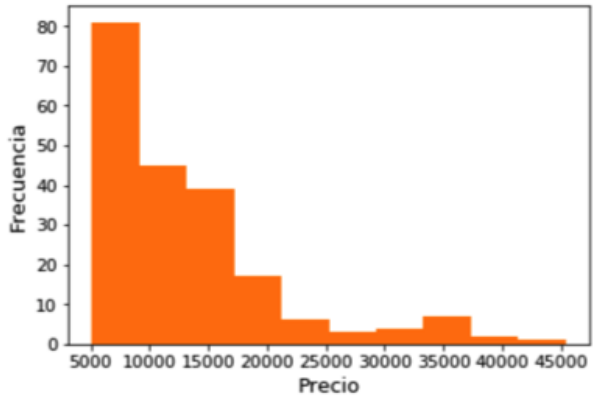
Gráfico de dispersión

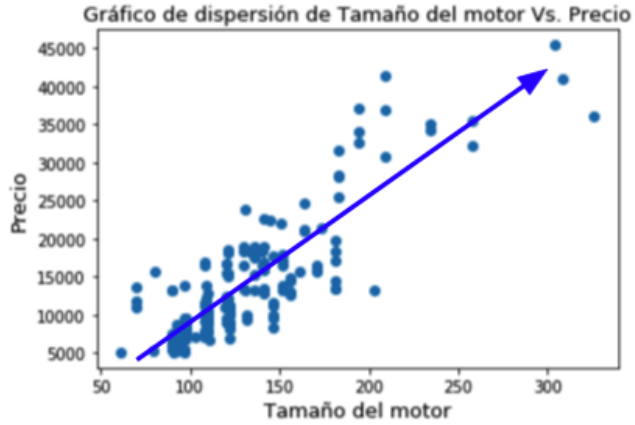
Gráfico que muestra la relación entre dos variables: la variable predictora (la que predice un resultado, eje-x) y la variable objetivo (la que deseamos predecir, eje-y)

Estadística descriptiva

Gráficos asociados a los ejemplos del cuaderno de Jupyter **Modulo3.ipynb**.







Correlación entre variables

Correlación

Es una métrica estadística para medir la interdependencia de las diferentes variables. En otras palabras, cuando miramos dos variables en el tiempo, si una variable cambia, ¿Cómo este efecto cambia la otra variable?

Correlación entre variables

Correlación

Es una métrica estadística para medir la interdependencia de las diferentes variables. En otras palabras, cuando miramos dos variables en el tiempo, si una variable cambia, ¿Cómo este efecto cambia la otra variable?

Ojo! Correlación no implica una relación causa-efecto! 😕

Correlación entre variables

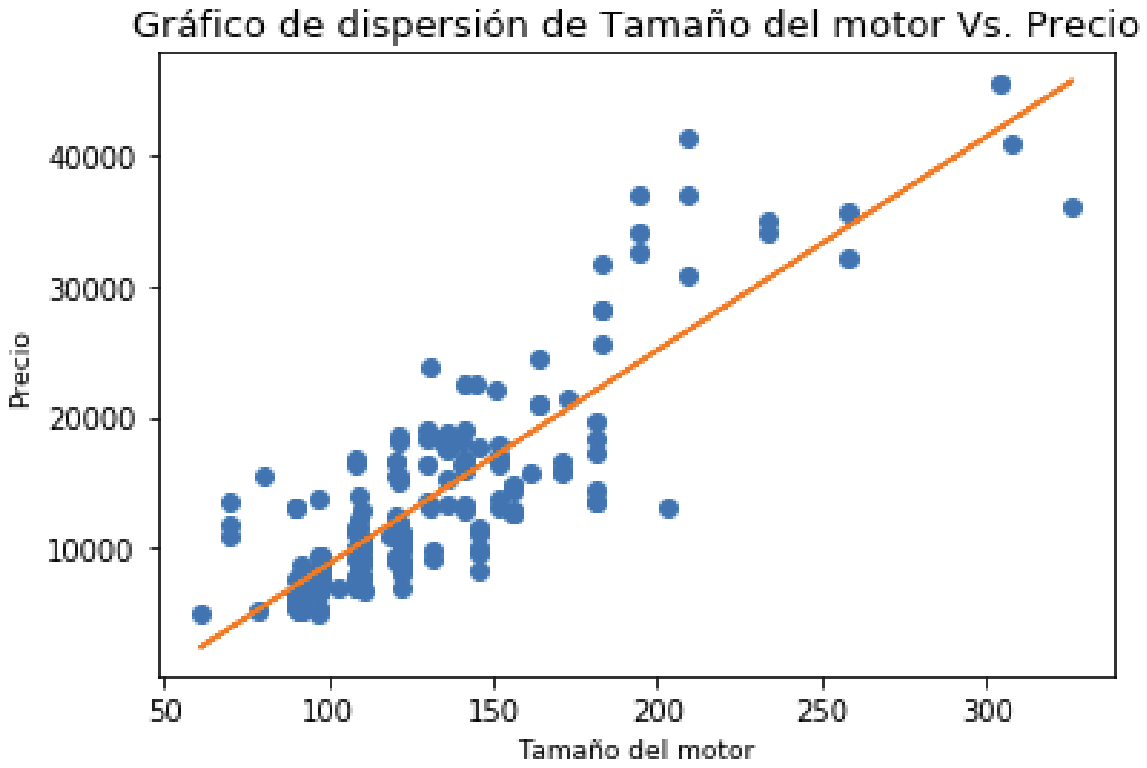
Correlación

Es una métrica estadística para medir la interdependencia de las diferentes variables. En otras palabras, cuando miramos dos variables en el tiempo, si una variable cambia, ¿Cómo este efecto cambia la otra variable?

Ojo! Correlación no implica una relación causa-efecto! 😕

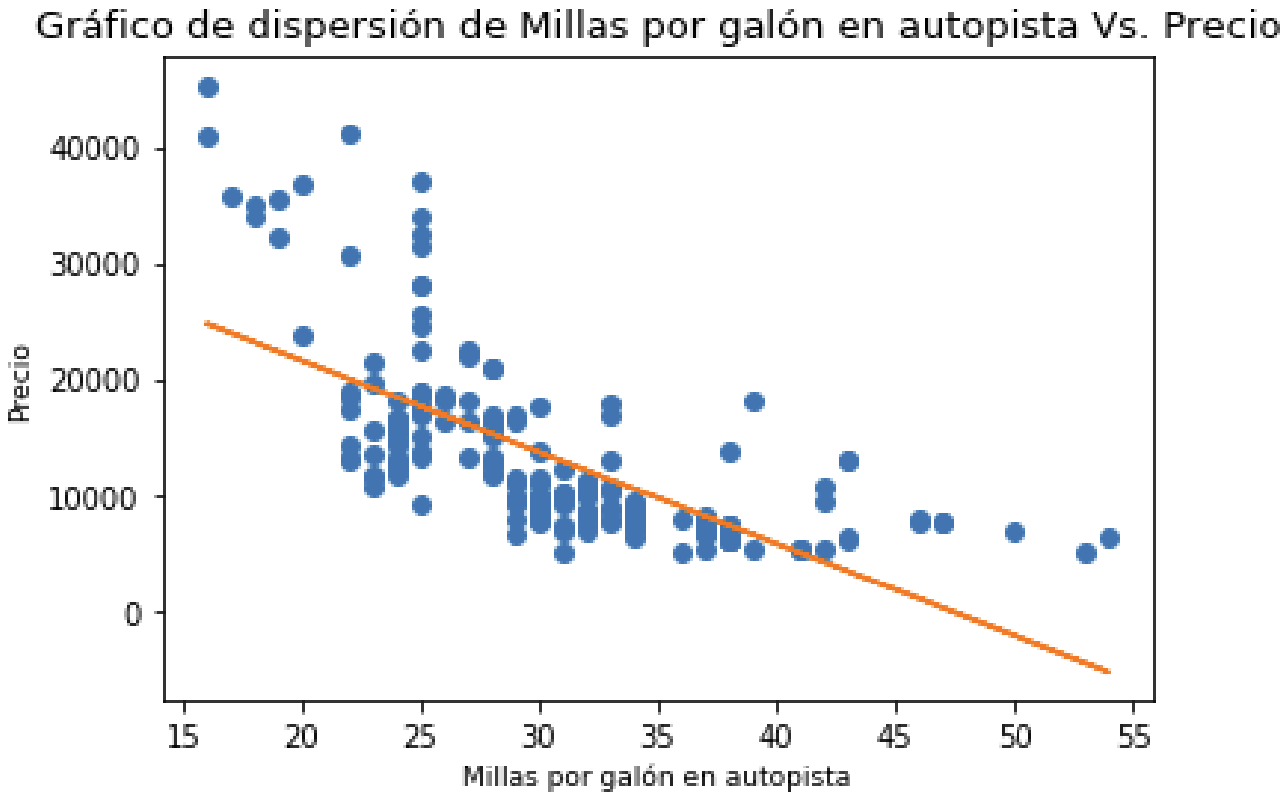
Vamos a ver en nuestro cuaderno de Jupyter **Modulo3.ipynb** un ejemplo de correlación entre dos variables a través de un gráfico.

Ejemplos de correlación



El gráfico de dispersión anterior revela que hay una relación lineal positiva entre el tamaño del motor y el precio del auto. Es decir, a medida que aumenta el tamaño del motor aumenta el precio.

Ejemplos de correlación



El gráfico de dispersión anterior revela que hay una relación lineal negativa entre las millas que recorre el auto por combustible que usa y el precio del mismo. Es decir, mientras más millas por galón el auto es más económico.

Correlación de Pearson

Mide la fuerza de la correlación entre dos variables:

- Coeficiente de correllación
- P-valor

Correlación de Pearson

Mide la fuerza de la correlación entre dos variables:

- Coeficiente de correllación
- P-valor

- Coeficiente de correlación
  - Cercano a +1: Relación positiva grande.
  - Cercano a −1: Relación negativa grande.
  - Cercano a 0: No hay relación.

Correlación de Pearson

Mide la fuerza de la correlación entre dos variables:

- Coeficiente de correllación
- P-valor

- Coeficiente de correlación
  - Cercano a +1: Relación positiva grande.
  - Cercano a −1: Relación negativa grande.
  - Cercano a 0: No hay relación.

- P-valor
  - $p - valor < 0{,}001$: Fuerte certeza en los resultados.
  - $p - valor < 0{,}05$: Certeza moderada.
  - $p - valor < 0{,}1$: Certeza debil.
  - $p - valor > 0{,}1$: No hay certeza en los resultados.

Correlación de Pearson

Mide la fuerza de la correlación entre dos variables:

- Coeficiente de correllación
- P-valor

- Coeficiente de correlación
  - Cercano a +1: Relación positiva grande.
  - Cercano a −1: Relación negativa grande.
  - Cercano a 0: No hay relación.

- P-valor
  - $p - valor < 0{,}001$: Fuerte certeza en los resultados.
  - $p - valor < 0{,}05$: Certeza moderada.
  - $p - valor < 0{,}1$: Certeza debil.
  - $p - valor > 0{,}1$: No hay certeza en los resultados.

- Correlación fuerte
  - Coeficiente de correlación cercano a 1 o −1.
  - P-valor menor a 0.001.

Coeficiente de Correlación y p-valor

```
from scipy import stats
c, p = stats.pearsonr(df['horsepower'],
                      df['price'])
```

Coeficiente de Correlación y p-valor

```
from scipy import stats
c, p = stats.pearsonr(df['horsepower'],
                      df['price'])
```

Coeficiente de Correlación: $c = 0{,}757916953745141$

P-Valor: $p = 1{,}6076704005409566e - 39$

Coeficiente de Correlación y p-valor

```
from scipy import stats
c, p = stats.pearsonr(df['horsepower'],
                      df['price'])
```

Coeficiente de Correlación: $c = 0{,}757916953745141$
P-Valor: $p = 1{,}6076704005409566e - 39$
Existe una fuerte correlación positiva entre las variables ya que el coeficiente de correlación es cercano a 1 y el p-valor es mucho menor que 0.001.

Conclusiones

- El cuaderno de Jupyter es una herramienta interactiva de fácil de apremder que permite la ejecución sencilla de código Python
- La biblioteca Pandas de Python permite el manejo de secuencias y tablas de datos de una forma amigable
- Cuaderno de Jupyter + Pandas + NumPy + Matplotlib son una buena combinación para comenzar a analizar datos y sacar información relevante de ellos

Ejercicio Práctico

1. Descargue el archivo `Ejercicio.csv` de la página en Github https://github.com/AdNAdN/Presentations
2. Lea este archivo como un `DataFrame` de Pandas
3. Haga una limpieza de los datos
4. Resumen estadístico
5. Grafique las variables de su preferencia con el gráfico que más le sea útil
6. Haga análisis de correlación de algún par de variables
7. Si algo le da error, pida ayuda a los cuadernos de Jupyter de este tutorial, documentación de Python o a la bibliografía recomendada, o si no encomiende a Google y StackOverflow ☺

Contactos

Prof. Miguel A. Astor

- miguel.astor@ciens.ucv.ve
- miguel.a.astor@ucv.ve

Profa. Adelis Nieves

- adelis.nieves@ciens.ucv.ve

?'Donde conseguir esta presentación y ejemplos?

- `https://github.com/miky-kr5/Presentations`
- `https://github.com/AdNAdN/Presentations`

?'Preguntas?