Introducción al análisis de datos usando Python

Curso: Bases para Data Science - Estadística, R y Python

Katherine Morales / Néstor Montaño

Sociedad Ecuatoriana de Estadística

Octubre-2020


SOCIEDAD
ECUATORIANA
ESTADÍSTICA

Nota:
Con *Alt* + *F* o *Option* + *F* puede hacer que estas dapositivas ocupen todo el navegador (es decir que se ignore el aspecto de diapositiva que tiene por default la presentación)

Consideraciones

Este parte del curso va a seguir la misma estructura de la parte de R, es decir mientras ustedes empiezan a familiarizarse con el software (en este caso python) en paralelo se van dando conceptos claves de análisis de datos. Sin embargo, ads a este punto ya vieron los conceptos estadísticos; por lo que si bien se los va a recordar esto se los pasará rápidamente.

En otras palabras, nos centraremos en aprender a manejar proyectos, importar a python, conocer los objetos dentro de python, manipular datos y en particular los dataFrames, esto es: seleccionar columnas, filtrar filas, modificar columnas, unir dos conjuntos de datos, realizar pivots, calculos y demás operaciones que se vieron ya con R.

¿Por qué aprender dos lenguajes
Pues, ¿por qué no?. Mientras más instrumentos tengan en su caja de herramientas es mejor. Además, cada lenguaje tiene puntos fuertes y se acomodan mejor a diferentes tareas, esto lo irán descubriendo por ustedes mismos.

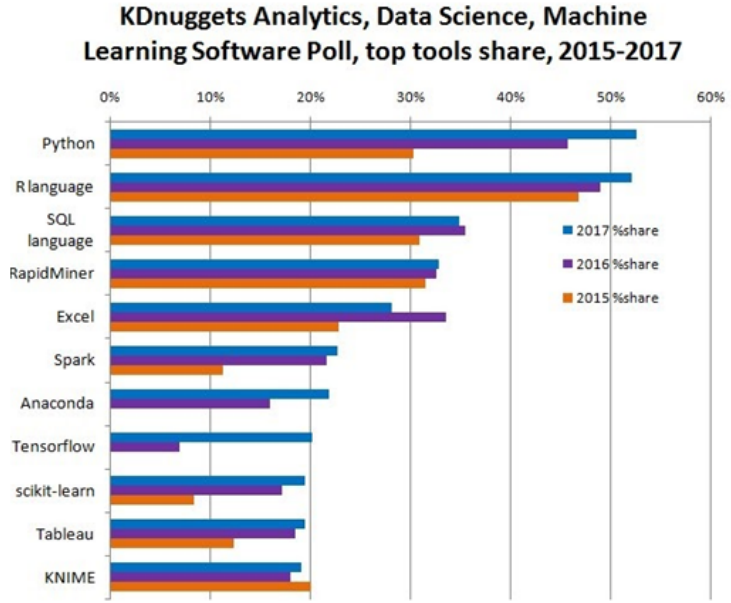
Introducción a Python y Anaconda

Curso: Bases para Data Science - Estadística, R y Python

Zulemma Bazurto / Néstor Montaño

¿Por qué Python?

- Software Libre (Open Source), gratuito y de desarrollo independiente,
- Es un lenguaje de objetivo general, es decir que sirve para hacer sitios web, aplicaciones móviles, sistemas de escritorio y para hacer ciencia de datos
- Hoy es el lenguaje más usado para Ciencia de datos y uno de los más usados en general,
- Enorme comunidad de usuarios,
- La mayoría de Universidades enseñan Python para carreras de computación o sistemas.



Popularidad Python

Instalar Python

En windows y Mac

- Visitar sitio web de Python
- Elegir la versión que se desea instalar
- Descargar y ejecutar el instalador marcando la casilla "Añadir Python #.# al PATH"
- Ojo: Mac viene con python 2.7, pero se aconseja actualizar a nuevas versiones

En Linux (Distribuciones)

- Python también viene instalado en linux
- Se lo instala usando la consola, ejemplo:
  - sudo yum install python (en Fedora, Red Hat o derivadas)
  - sudo apt-get install Python (en Debian, Ubuntu y derivadas)

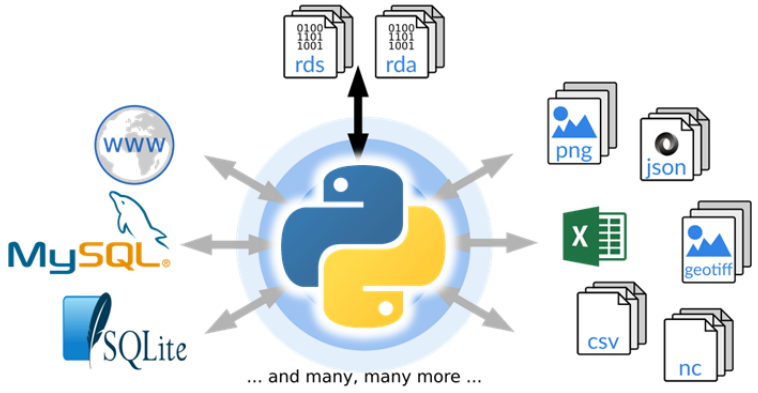
Instalar Anaconda

Anaconda es un distribución libre y abierta de Python (y R), viene con utilidades muy usadas como Spyder o Jupyter en python. Es probablemente la distribución más usada para ciencia de datos, ojo que anaconda instala python por default (no se requiere el paso de la diapositiva anterior)

- Visitar sitio web de Anaconda
- Elegir la versión que se desea instalar
- Descargar y ejecutar el instalador

¿Por qué Python?

- Rico ecosistema de librerías, integraciones, frameworks, etc,
- Las integraciones de un modelo a producción suelen ser sencillo con Python,
- Hay distribuciones de Python enfocadas a Ciencia de Datos como Anaconda,
- Para programar en Python se pueden usar algunas IDEs (Interfaz de desarrollo) como Jupyter, PyCharm, Spyder, Rstudio, algunas de ellas integradas con Anaconda,
- Los Frameworks más usados para DeepLearning usan Python como base.



Algunas de las integraciones de Python

¿Por qué Anaconda?

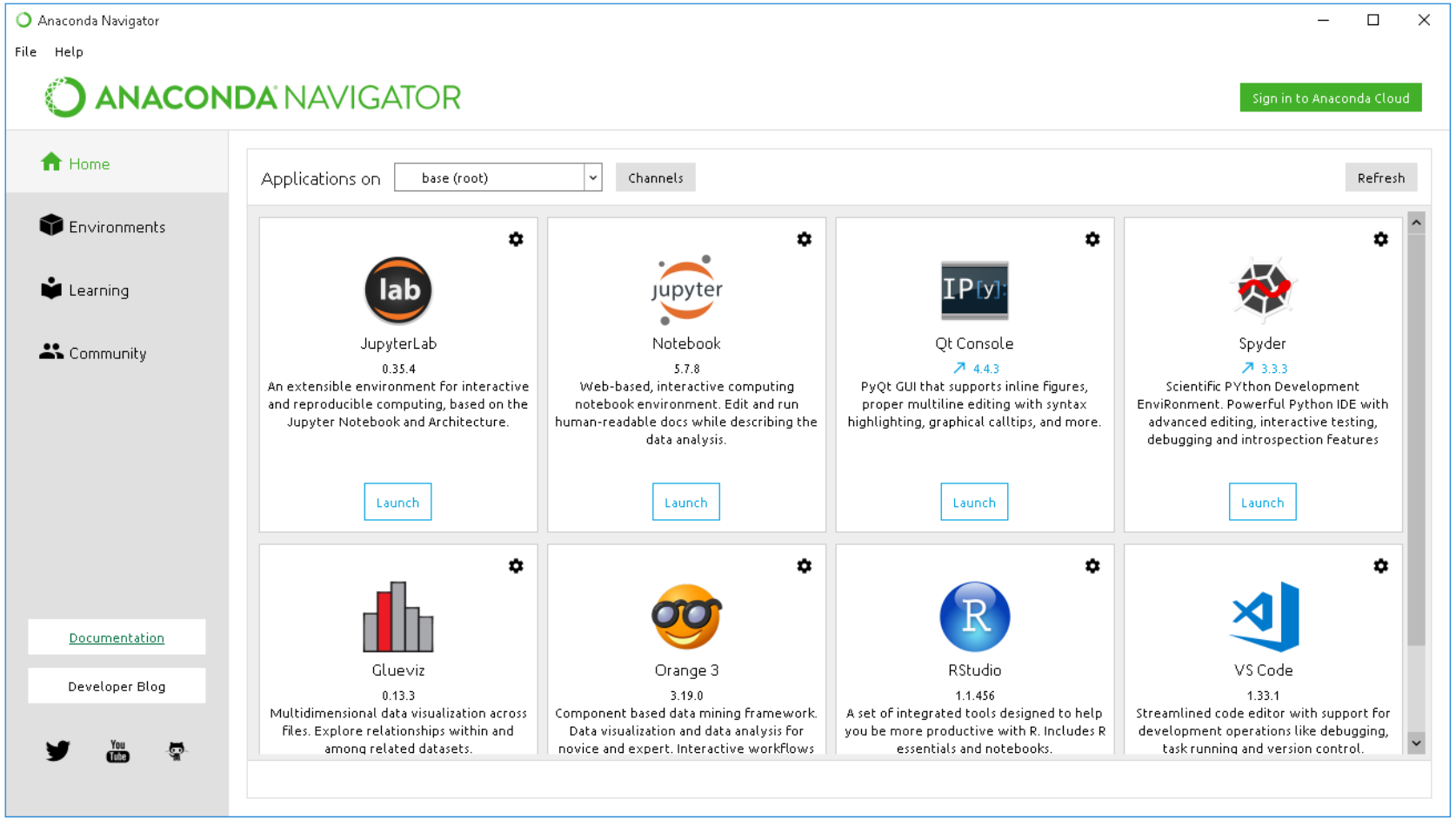
Anaconda

- Gratuito
- Software Libre (Open Source)
- Amplia documentación y gran comunidad.
- Permite instalar y administrar paquetes, dependencias y entornos para la ciencias de datos con Python de una manera muy sencilla.
- Integra diversos IDE como Jupyter, JupyterLab, Spyder y RStudio.
- Cuenta con herramientas como Dask, numpy, pandas y Numba para analizar Datos.
- Para visualizar datos integra Bokeh , Datashader , Holoviews o Matplotlib.
- Aplicaciones relacionadas con el aprendizaje de máquina y los modelos de aprendizaje como Orange
- Elimina problemas de dependencia de paquetes y control de versiones.

Estructura de Anaconda

Anaconda

Abrir Anaconda Navigator

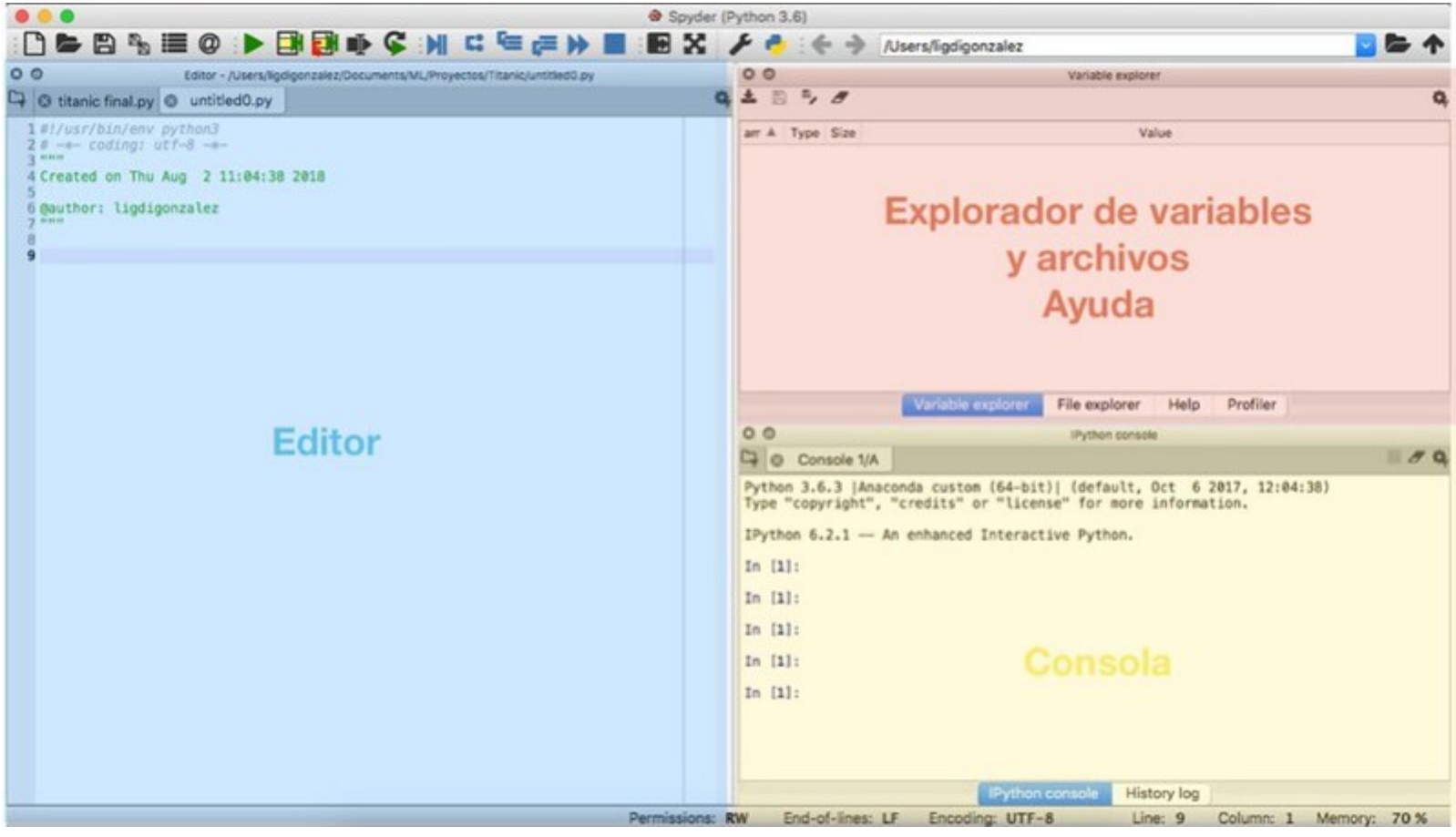


Spyder

- The Scientific (S) Python (PY) Development Environment (DER)
- Jupyter, PyCharm, Spyder son de las 3 IDE más usadas para ciencia de datos en Python

Spyder

Abrir Spyder



Spyder - Primera impresion

1. Editor: Pantalla donde se escriben las líneas de código
2. Consola: Donde se muestra el código ejecutado y el resultado
3. Explorador de Archivos, variables, Ayuda: Esta pantalla esta particionada en varias pestañas como:
    - Explorador de carpetas y archivos
    - Variables.- Donde se van a mostrar los objetos que vamos a ir creando
    - Help.- las ayudas internas del sistema

Proyectos en Spyder

- Un **Proyecto** es una carpeta que contiene todos los scripts y archivos el proyecto
- Permite tener nuestros análisis ordenados,

- Se sugiere tener una estructura interior, por ejemplo:
    - Scripts, Data, Exports, Info

- Nuevo Proyecto.- Proyectos > Nuevo Proyecto

- Elegir ubicacion y nombre del proyecto
- En la carpeta del proyecto crear las carpetas: Data, Exports, Scripts, Info (Recomendado)

Realizar un script en Spyder

- Nuevo script: `ctrl + n`
- Completado de comando: `tab`
- Ejecutar selección: `ctrl + enter`, `shift + enter`
- Comentarios: `ctrl + 1`,en bloque: `ctrl + 4`

```
import os
print(f'Hola, {os.getlogin()}!, ¡¿Listo para empezar?!')
```

```
## Hola, Nestor!, ¡¿Listo para empezar?!
```

Generalidades

Python, aparte de objetos, tiene:

- Expresión.- Se evalúa, se imprime y el valor se pierde (iPython)

```
5+5 # Expresión con output
```

```
## 10
```

```
5+5; # Expresión sin salida
```

```
## 10
```

- Asignación.- Evalúa la expresión y guarda el resultado en una variable (no lo imprime)

```
a = 5+5 # Asigna el valor a la variable "a"
```

Asignaciones

- El resultado de una función de un objeto X puede ser asignada al mismo objeto X en la misma sentencia, es decir

```
a = 5 # Asignación
a
```

```
## 5
```

```
a = 2*a  # Asignación a mismo objeto
a
```

```
## 10
```

Generalidades

- Comandos se separan por `;` o `enter`
- Para comentar se usa #
- Case sensitivity (`Abc` es diferente de `abc`)

```
a= 2; b= 1; a + b
```

```
## 3
```

Python como calculadora

```
2 + 3*5 # operaciones básicas
```

```
## 17
```

```
7 // 3 # division entera
```

```
## 2
```

```
7 % 3 # Modular
```

```
## 1
```

```
2 ** 3 # 2 elevado al cubo
```

```
## 8
```

```
pow(2, 3) # 2 elevado al cubo
```

```
## 8
```

Python como calculadora

A diferencia de R, para usar operaciones más "complicadas" debemos ya importar una libreria (import en Python es el equivalente de library en R).

Una biblioteca/librería es una colección de funciones y objetos que aumentan las capacidades del lenguaje, en este caso `math` permite cargar funciones enfocadas en cálculos matemáticos básicos.

Una biblioteca se instala (que es descargar los archivos ordenados a nuestro disco duro) y luego se activa (que es cargarla a RAM para poder usar sus funciones módulos), esto último se hace con `import`

```
import math
math.floor(2.3) # Funcion piso
```

```
## 2
```

```
math.fabs(-5) # valor absoluto
```

```
## 5.0
```

```
math.factorial(3) # Factorial
```

```
## 6
```

Python como calculadora

Otros ejemplos

```
math.exp(3) # e elevado a la x
```

```
## 20.085536923187668
```

```
math.sqrt(9) # raiz cuadrada
```

```
## 3.0
```

```
math.log(math.exp(2)) # Logaritmo natural
```

```
## 2.0
```

```
math.floor(2.3) # funcion piso
```

```
## 2
```

Manejo de paquetes y environments

Como se dijo:

- Una biblioteca/librería es una colección de funciones y objetos que aumentan las capacidades del lenguaje,
- Es simplemente un directorio que contiene otros paquetes, módulos o scripts,
- Instalación: `conda install`, `pip install`, `pip3 install`.

Python tiene además **environments**, que permiten tener encapsulados versiones de Python con un conjunto de paquetes específico

- En el navegador de Anaconda, se puede ver los environments creados,
- El environments **base** es el predefinido,
- Además el navegador permite instalar bibliotecas de forma visual.

Bibliotecas a usar

Los paquetes o bibliotecas más usadas para ciencia de datos son:

```
import os
import math
import numpy as np
import pandas as pd
import scipy
import matplotlib.pyplot as plt
import seaborn as sns
```

Directorio de trabajo

- El directorio de trabajo es la ruta en la cual se va a empezar todas las rutas que no estén completas (recordar al momento de importar data)
- Tecnicamente es un conjunto de objetos y un puntero

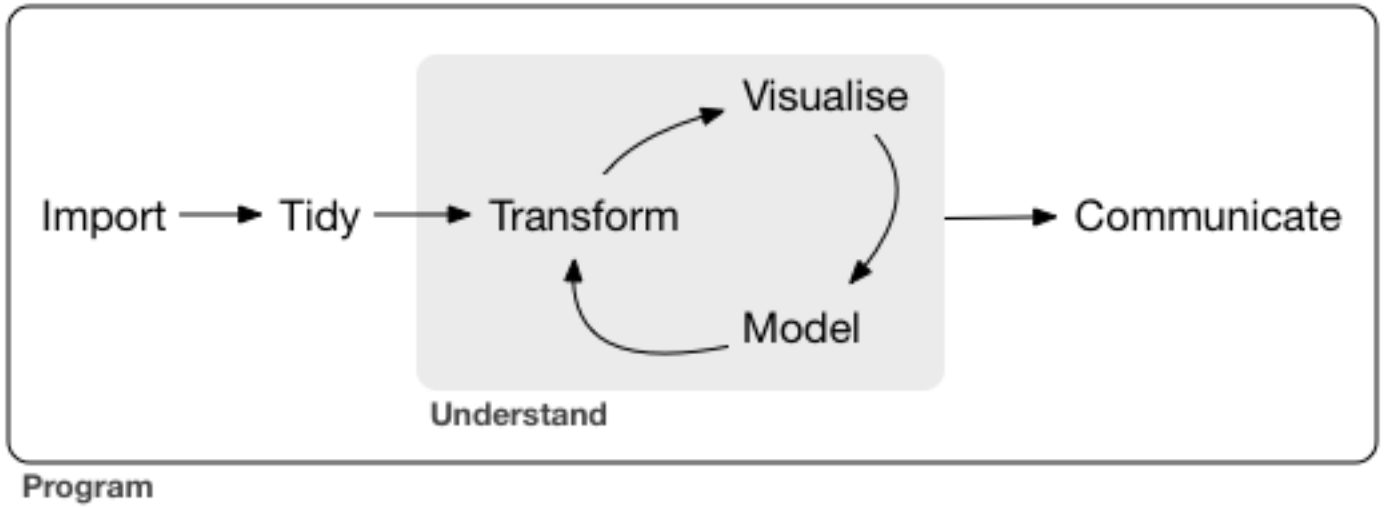
```
import os
os.chdir("D:\SEE\...\Proyeto")
os.getcwd() # Verificar el directorio de trabajo
os.listdir("./")
```

Introducción a los análisis estadísticos

Curso: Bases para Data Science - Estadística, R y Python

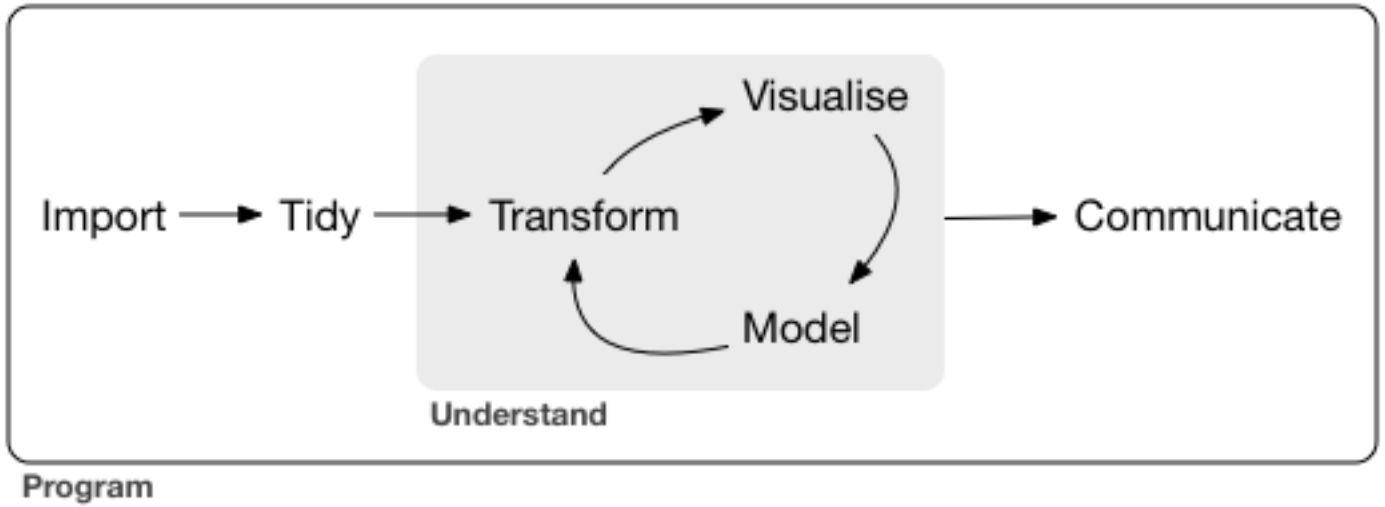
Katherine Morales / Néstor Montaño

Workflow de un análisis estadístico



- Import: Obtener y entender los datos
- Tidy: Ordenar los datos de tal manera que sea sencillo transformarlos, sumarizarlo, visualizarlos o realizar un modelo con ellos
- Transform: Manipular los datos hasta obtener el input que el análisis o técnica estadística necesita
- Visualise: Realizar el análisis exploratorio de datos
- Model: Aplicar técnicas estadísticas para el entendimiento del problema o tomar decisiones
- Comunicate: Tratar de mostrar los resultados de tal forma que el resto del mundo los entienda, usando reportes, gráficos, visualizaciones interactivas, integración con herramientas de BI, web apps, etc.

Workflow de un análisis estadístico



- Import
- Tidy
  Repetir mientras sea necesario
- **Transform:** Manipular los datos, obtener el input del modelo
- **Visualise:** Realizar el análisis exploratorio de datos
- **Model:** Aplicar técnicas estadísticas
- Comunicate

Caso: Data de transacciones bancarias

Caso: Data de transacciones bancarias

El Banco del Pacífico requiere mejorar los tiempos de atención al cliente en ventanilla, para ello ha recolectado esta información anónimamente para cada cajero y transacción realizada.

Le suministran un excel con dos hojas:

1. Tiene los datos de las transacciones, columnas: Sucursal, Cajero, ID_Transaccion, Transaccion, Tiempo_Servicio_seg, Nivel de satisfacción, Monto de la transaccion.
2. Otra hoja que indica si en la sucursal se ha puesto o no el nuevo sistema.

Caso: Data de transacciones bancarias

Revisar archivo de excel: Data_Banco.xlsx

Crear un proyecto en Spyder, con las carpetas Data, Exports, etc

Poner en la carpeta Data, el excel suministrado

Importar datos

Introducción al análisis de datos

Importar desde csv

- Opción 1: Desde Spyder, explorador de archivos
- Opción 2: Usando Pandas:

```
import pandas as pd
archivo_csv= "Data/Data_Banco.csv" # Ruta al archivo
data_banco_csv = pd.read_csv(archivo_csv, sep = ";") # Notese el sep= ";"
data_banco_csv.head(5) # Mostrar 5 primeras filas del DataFrame
```

```
##    Sucursal  Cajero  ...  Tiempo_Servicio_seg Satisfaccion
## 0        62    4820  ...                   41    Muy Bueno
## 1        62    4820  ...                   41         Malo
## 2        62    4820  ...                   41      Regular
## 3        62    4820  ...                   41      Regular
## 4        62    4820  ...                   41    Muy Bueno
##
## [5 rows x 6 columns]
```

Importar desde excel

Importar desde excel, opción 1

```
# Debe estar seteado el chdir
archivo_xlsx = 'Data//Data_Banco.xlsx' # Ruta al archivo
xlsx = pd.ExcelFile(archivo_xlsx) # Carga todo el spreadsheet
print(xlsx.sheet_names) # Ver hojas del archivo
```

```
## ['Data', 'Data_Sucursal', 'Data_Cajero']
```

```
data_banco = xlsx.parse('Data')
data_banco.head(5)
```

```
##    Sucursal  Cajero  ID_Transaccion  ... Tiempo_Servicio_seg  Satisfaccion    Monto
## 0        62    4820               2  ...               311.0     Muy Bueno   2889,3
## 1        62    4820               2  ...               156.0          Malo  1670,69
## 2        62    4820               2  ...               248.0       Regular  3172,49
## 3        62    4820               2  ...                99.0       Regular  1764.92
## 4        62    4820               2  ...               123.0     Muy Bueno  1835.69
##
## [5 rows x 7 columns]
```

Importar desde excel

Importar desde excel, opción 2

```
# OPCION 2
data_banco_xlsx = pd.read_excel(archivo_xlsx, sheet_name = 'Data')
data_banco_xlsx.head(5)
```

```
##    Sucursal  Cajero  ID_Transaccion  ... Tiempo_Servicio_seg  Satisfaccion    Monto
## 0        62    4820               2  ...               311.0     Muy Bueno   2889,3
## 1        62    4820               2  ...               156.0          Malo  1670,69
## 2        62    4820               2  ...               248.0       Regular  3172,49
## 3        62    4820               2  ...                99.0       Regular  1764.92
## 4        62    4820               2  ...               123.0     Muy Bueno  1835.69
##
## [5 rows x 7 columns]
```

Importar desde excel

Importar la otra hoja de excel

```
data_sucursal = pd.read_excel('Data//Data_Banco.xlsx', sheet_name = 'Data_Sucursal')
data_sucursal.head()
```

```
##    ID_Sucursal       Sucursal Nuevo_Sistema
## 0           62  Riocentro Sur            No
## 1           85         Centro            Si
## 2          267       Alborada            Si
## 3          443   Mall del Sol            Si
## 4          586      Via Daule            No
```

Importar desde otras fuentes

Panda permite importar desde una gran cantidad de fuentes, incluso conectarse a Bases de Datos.

- `read_csv` para importar desde csv
- `ExcelFile` & `xl.parse` o `read_excel` para importar desde excel
- `openpyxl` también permite manipular excels
- `read_json` para importar desde json
- `read_sql_table` para importar toda una tabla
- Más información en: https://pandas.pydata.org/docs/reference/io.html

¿Qué es Pandas?

Pandas es la gran navaja suiza de Python para Data Science

- Soporta lectura desde una variedad de datos, integrarlos y transformarlos
- Permite realizar estadística descriptiva
- Tiene opciones de gráficos integrados
- Soporta series de tiempo
- Métodos integrados para manejar valores perdidos
- Soporte para procesamiento de imágenes
- Internamente maneja dos tipos de objetos, pandas Series, panda DataFrame

pandas Series, pandas DataFrame ¿Qué es eso?

pandas Series, pandas DataFrame son **Estructuras de datos**, las cuales no son más que tipos de Objetos dentro de Python. Un entero es un objeto, un número también es un objeto. Ambos se pueden "ordenar" en estructuras como:

- Listas
- Matrices
- Tuples
- ndarrays
- pandas Series
- pandas DataFrames
- Ver https://docs.python.org/3/tutorial/datastructures.html

Procederemos ahora a entender un poco los objetos y estructuras de datos que tiene Python base y los objetos que crean algunas bibliotecas.

Estructuras de datos y Objetos

Introducción a Python

Estructuras de datos | Objetos

Un string

```
a= 'Hola Mundo'
a
```

```
## 'Hola Mundo'
```

```
a.__class__
```

```
## <class 'str'>
```

Métodos en Python

Existen lenguajes que a pesar de ser orientados a objetos, se comportan de forma funcionales como **R**, Python es como java/c++, se usan métodos de los objetos creados.

> En python para aplicar un método se usa nombre_del_objeto.metodo(parametros)

Para entender esto vamos a ver algunos métodos asociados a un objeto de tipo string. Ir a https://docs.python.org/3/library/stdtypes.html#string-methods

Métodos en Python

En python para aplicar un método se usa nombre_del_objeto.metodo(parametros)

```
z = "hola mundo" # Crear objeto
z.capitalize() # Usar metodo para letra capital (primera mayuscula)
```

```
## 'Hola mundo'
```

```
z # Pero z no ha cambiado
```

```
## 'hola mundo'
```

```
z= z.capitalize() # Cambiar z
z # mostar z
```

```
## 'Hola mundo'
```

Métodos en Python

En python para aplicar un método se usa nombre_del_objeto.metodo(parametros)

```
z # mostar z
```

```
## 'Hola mundo'
```

```
z.isupper() # TRUE si todos son mayuscula
```

```
## False
```

```
z.upper() # transforma todo a mayuscula
```

```
## 'HOLA MUNDO'
```

```
z.upper().isupper() # Concatenar metodos
```

```
## True
```

Estructuras de datos | Objetos

Un entero

```
a= 5
a
```

```
## 5
```

```
a.__class__
```

```
## <class 'int'>
```

Estructuras de datos | Objetos

Una lista

```
a= [2, 4]
a
```

```
## [2, 4]
```

```
a.__class__
```

```
## <class 'list'>
```

Estructuras de datos | Objetos

Un pandasSeries

```
a = pd.Series([2, 4])
a
```

```
## 0    2
## 1    4
## dtype: int64
```

```
a.__class__
```

```
## <class 'pandas.core.series.Series'>
```

```
a.values
```

```
## array([2, 4], dtype=int64)
```

Estructuras de datos | Objetos

En primera instancia vamos a centrarnos en

- **Listas**
- Matrices
- Tuples
- ndarrays
- **pandas Series**
- **pandas DataFrames**

Listas

Las listas en Python se crean con []

```
lista = [10, 20, 30]
lista[1] # Nótese la forma de indexar
```

```
## 20
```

```
lista[0]
# lista[3] # Error
```

```
## 10
```

Listas

Las listas en Python se crean con []

```
lista[1:3] # Mostrar 2do y 3er elemento
```

```
## [20, 30]
```

```
lista[0:2] # Mostrar 1ro y 2do elemento
```

```
## [10, 20]
```

```
len(lista) # largo del vector
```

```
## 3
```

Listas

Las listas en Python se crean con []

```
lista2 = ['a', 'b', 'c'] # Un lista de sólo texto
lista2[1:3] # Mostrar 2do y 3er elemento
```

```
## ['b', 'c']
```

```
lista3 = ['a', 12, 'c']  # Las listas en Python permiten mezclar tipos de datos
lista3[0]
```

```
## 'a'
```

```
lista3[1]
```

```
## 12
```

Listas

Cambiar 2do elemento por 200, notese indexado

```
# Python
lista[1] = 200
lista
```

```
## [10, 200, 30]
```

Listas

Agregar, borrar elementos

```
lista
```

```
## [10, 200, 30]
```

```
lista.append(400)
lista
```

```
## [10, 200, 30, 400]
```

```
lista.append(400)
lista
```

```
## [10, 200, 30, 400, 400]
```

Listas

Agregar, borrar elementos

```
lista
```

```
## [10, 200, 30, 400, 400]
```

```
lista.pop(1) #Elimina segundo elemento
```

```
## 200
```

```
lista
```

```
## [10, 30, 400, 400]
```

```
lista.remove(400) # Elimina el primer 400 que encuentra
lista
```

```
## [10, 30, 400]
```

Listas

Agregar, borrar elementos

```
lista
```

```
## [10, 30, 400]
```

```
del lista[1] #Elimina segundo elemento
lista
```

```
## [10, 400]
```

Listas

Agregar, borrar elementos

```
lista= [10, 2, 30]
lista.sort() # Ordenar valores internos
lista
```

```
## [2, 10, 30]
```

```
lista.reverse() # Cambia la lista de ultimo a primer valor
lista
```

```
## [30, 10, 2]
```

Listas

Concatenar dos listas, en python los tipos de datos pueden ser distintos

```
lista1 = [10, 20, 30]
lista2 = ['a', 'b', 'c']
lista1 + lista2 # Concatenar: No cambia lista1 ni lista2
```

```
## [10, 20, 30, 'a', 'b', 'c']
```

```
lista1.extend(lista2) # Agregar lista2 a lista1
lista1
```

```
## [10, 20, 30, 'a', 'b', 'c']
```

Listas

Concatenar dos listas, en python los tipos de datos pueden ser distintos

```
for i in lista1:
    print( i, type(i) )
```

```
## 10 <class 'int'>
## 20 <class 'int'>
## 30 <class 'int'>
## a <class 'str'>
## b <class 'str'>
## c <class 'str'>
```

Listas

Trabajar con dos listas en paralelo

```
lista1 = [10, 20, 30]
lista2 = ['a', 'b', 'c']
zip(lista1, lista2)
```

```
## <zip object at 0x000000002A80DC88>
```

```
for i, j in zip(lista1, lista2) :
    print( i, j )
## Intenten con listas de diferente largo
## Python coge el menor tamaño
```

```
## 10 a
## 20 b
## 30 c
```

Listas y las referencias (punteros) en Python

```
lista1 = [10, 20, 30]
lista2 = lista1 # nueva variable pero apuntando al mismo objeto
lista1[1] = 200 # Cambiamos la lista1
lista1
```

```
## [10, 200, 30]
```

```
lista2 # Pero tb se ha cambiado la lista2
## Esto es por el manejo de las referencias en python
```

```
## [10, 200, 30]
```

Listas y las referencias (punteros) en Python

```
lista1 = [10, 20, 30]
lista2 = list(lista1) # Nuevo objeto
lista1[1] = 200 # Cambiamos la lista1
lista1
```

```
## [10, 200, 30]
```

```
lista2
```

```
## [10, 20, 30]
```

Listas y las referencias (punteros) en Python

Es importante saber cuándo hacer copias o simplemente usar la misma referencia, usar la misma referencia es mejor para la memoria pero puede llevar a cometer errores incluso a programadores experimentados, se requiere un análisis detrás de la decisión.

pandas.DataFrame

- DataFrame es un objeto que cumple:
  - Las columnas son vectores de tipo pandas Series
  - Cada columnas puede ser de un tipo de dato distinto
  - Cada elemento, columna es una variable
  - Las columnas tienen el mismo largo
- Se podría decir que un data.frame es como una tabla en una hoja de excel

pandas.DataFrame

Crear un data.frame

```
Nombre = ['Ana', 'Berni', 'Carlos']
Edad = [20,19,20]
Ciudad = ['Gye', 'Uio', 'Cue']
df_1= pd.DataFrame({'Nombre': Nombre, 'Edad': Edad, 'Ciudad': Ciudad})
df_1
```

```
##    Nombre  Edad Ciudad
## 0     Ana    20    Gye
## 1   Berni    19    Uio
## 2  Carlos    20    Cue
```

pandas.DataFrame

Crear un data.frame

```
df_2= pd.DataFrame({
        'Nombre' : ['Ana', 'Berni', 'Carlos'],
        'Edad' : [20,19,20],
        'Ciudad' : ['Gye', 'Uio', 'Cue']
        })
df_2
```

```
##     Nombre  Edad Ciudad
## 0      Ana    20    Gye
## 1    Berni    19    Uio
## 2  Carlos    20    Cue
```

pandas.DataFrame

Index en un dataframe

```
df_3= pd.DataFrame({
        'Nombre' : ['Ana', 'Berni', 'Carlos'],
        'Edad' : [20,19,20],
        'Ciudad' : ['Gye', 'Uio', 'Cue']
        }, index= ['a', 'b', 'c'])
df_3
```

```
##     Nombre  Edad Ciudad
## a      Ana    20    Gye
## b    Berni    19    Uio
## c   Carlos    20    Cue
```

pandas.DataFrame

Visualizar primeras filas

```
df_3.head(2)
```

```
##   Nombre  Edad Ciudad
## a    Ana    20    Gye
## b  Berni    19    Uio
```

pandas.DataFrame

Visualizar últimas filas

```
df_3.head(2)
```

```
##   Nombre  Edad Ciudad
## a    Ana    20    Gye
## b  Berni    19    Uio
```

pandas.DataFrame

`.info()` permite ver la estructura de cualquier objeto en python.

```
## Visualizar la estructura
df_3.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## Index: 3 entries, a to c
## Data columns (total 3 columns):
## Nombre    3 non-null object
## Edad      3 non-null int64
## Ciudad    3 non-null object
## dtypes: int64(1), object(2)
## memory usage: 96.0+ bytes
```

pandas.DataFrame

Modificar nombre de las variables

```
df_3.rename( columns= {'Nombre':'Name', 'Edad':'Age', 'Ciudad':'City'}) # No cambia el objeto
```

```
##       Name  Age City
## a      Ana   20  Gye
## b    Berni   19  Uio
## c   Carlos   20  Cue
```

```
df_3
```

```
##     Nombre  Edad Ciudad
## a      Ana    20    Gye
## b    Berni    19    Uio
## c   Carlos    20    Cue
```

pandas.DataFrame

Modificar nombre de las variables

```
df_3.rename( columns= {'Nombre':'Name', 'Edad':'Age', 'Ciudad':'City'},
inplace= True) # Cambia el objeto
df_3
```

```
##       Name  Age City
## a      Ana   20  Gye
## b    Berni   19  Uio
## c   Carlos   20  Cue
```

¿Qué importamos entonces?

```
data_banco_xlsx.__class__
```

```
## <class 'pandas.core.frame.DataFrame'>
```

```
type(data_sucursal)
```

```
## <class 'pandas.core.frame.DataFrame'>
```

¿Qué importamos entonces?

Nótese que se preguntó lo mismo usando un método y una función, recuerden que un método es parte de una clase y por tanto está asociado a un objeto; las funciones en cambio están definidas por si mismas y no pertenecen a ninguna clase.

```
data_banco_xlsx.__class__
```

```
## <class 'pandas.core.frame.DataFrame'>
```

```
type(data_sucursal)
```

```
## <class 'pandas.core.frame.DataFrame'>
```

¿Qué importamos entonces?

Usando .info:

```
data_banco_xlsx.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 24299 entries, 0 to 24298
## Data columns (total 7 columns):
## Sucursal                24299 non-null int64
## Cajero                  24299 non-null int64
## ID_Transaccion          24299 non-null int64
## Transaccion             24299 non-null object
## Tiempo_Servicio_seg     24299 non-null float64
## Satisfaccion            24299 non-null object
## Monto                   24299 non-null object
## dtypes: float64(1), int64(3), object(3)
## memory usage: 1.3+ MB
```

Entender los datos

Introducción a los análisis estadísticos

Entender los datos

Luego de importar se debe entender los datos

- ¿Qué representa cada columna?
- ¿Qué tipo de dato debería tener cada columna?
- ¿Qué granularidad o atomicidad tiene la data?
- Si es que se tiene varios conjuntos de datos ¿Cómo se relacionan los datos?
- A qué periodo de tiempo corresponde la data
- Muchas veces se obtiene la información desde una base de datos y por tanto toca entender la base y el query que genera los datos

Ejemplo - Entender los datos

Podríamos ver las primeras filas

```
# ver las primeras 5 filas
data_sucursal.head( 5)
```

```
##    ID_Sucursal       Sucursal Nuevo_Sistema
## 0           62  Riocentro Sur            No
## 1           85         Centro            Si
## 2          267       Alborada            Si
## 3          443   Mall del Sol            Si
## 4          586      Via Daule            No
```

Ejemplo - Entender los datos

O ver la estructura de los dataFrames

```
data_banco_xlsx.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 24299 entries, 0 to 24298
## Data columns (total 7 columns):
## Sucursal                24299 non-null int64
## Cajero                  24299 non-null int64
## ID_Transaccion          24299 non-null int64
## Transaccion             24299 non-null object
## Tiempo_Servicio_seg     24299 non-null float64
## Satisfaccion            24299 non-null object
## Monto                   24299 non-null object
## dtypes: float64(1), int64(3), object(3)
## memory usage: 1.3+ MB
```

Tipos de datos

Para saber qué tipo de dato debería tener cada columna, debemos también conocer los tipos datos en Python

```
a= 1
a.__class__
```

```
## <class 'int'>
```

```
type(a)
```

```
## <class 'int'>
```

```
a= 1.3
a.__class__
```

```
## <class 'float'>
```

```
type(a)
```

```
## <class 'float'>
```

Tipos de datos

Tipos datos en Python

```
a= 1 + 2j
a.__class__
```

```
## <class 'complex'>
```

```
type(a)
```

```
## <class 'complex'>
```

```
a= 'texto'
a.__class__
```

```
## <class 'str'>
```

```
type(a)
```

```
## <class 'str'>
```

Tipos de datos

```
from datetime import date
a= date.fromisoformat('2019-12-04')
a.__class__
```

```
## <class 'datetime.date'>
```

```
type(a)
```

```
## <class 'datetime.date'>
```

```
b= date(2019, 12, 4)
b.__class__
```

```
## <class 'datetime.date'>
```

```
type(b)
```

```
## <class 'datetime.date'>
```

Tipos de datos - pd.Categorical

Util para tipos de datos ordinales

- Primero se agrega el vector de información
- Categories: los niveles del factor labels: nombre de los niveles
- El factor puede tener un orden específico

```
# Crear un factor ordenado
a = pd.Categorical( ['alto', 'bajo', 'alto', 'alto'],
                    categories= ['bajo', 'mediano', 'alto'],
                    ordered=True)
a.__class__
```

```
## <class 'pandas.core.arrays.categorical.Categorical'>
```

```
a # Mostrar el factor
```

```
## [alto, bajo, alto, alto]
## Categories (3, object): [bajo < mediano < alto]
```

Tipos de datos

Datos lógicos

```
b= True
b.__class__
```

```
## <class 'bool'>
```

```
type(b)
```

```
## <class 'bool'>
```

```
b
```

```
## True
```

```
b= False
```

Tipos de datos

Casos especiales

```
1.797e308 # maximo float posible
```

```
## 1.797e+308
```

```
1.798e308 # Resulta en infinito
```

```
## inf
```

```
pos_inf = math.inf     # Constante desde la libreria math
neg_inf = float('-inf')    # Declarar un float Inf, tb permite nan
```

Tipos de datos

Casos especiales: Not a Number

```
c= math.nan
type(c)
```

```
## <class 'float'>
```

```
d= float('nan')
type(d)
```

```
## <class 'float'>
```

Tipos de datos

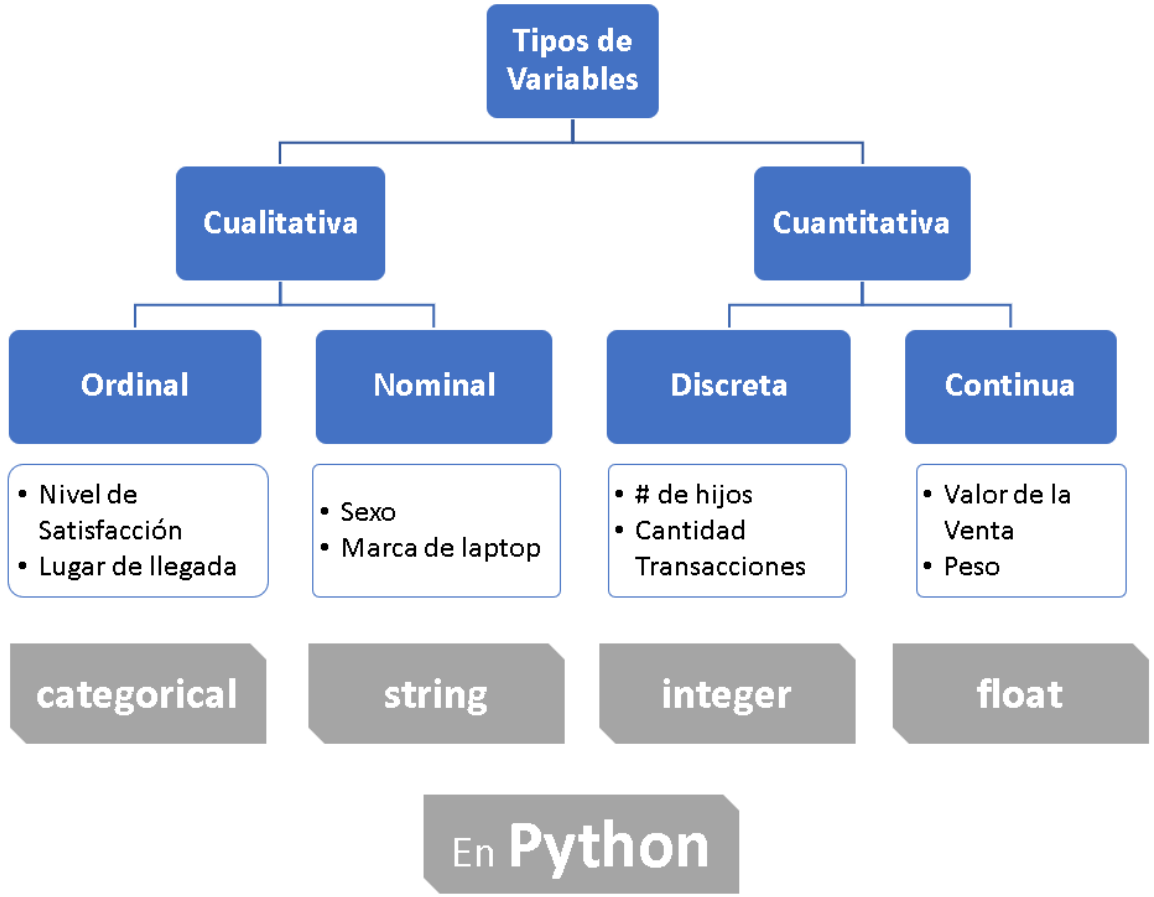
Casos especiales: Not a Number

```
0.0 * neg_inf
```

```
## nan
```

Tipos de variables

Tipos de variables y su correspondencia en Python



Tipos de datos

Revisar que todas las columnas tengan el tipo correcto y además la relación que existe entre los dos dataframes importados.

```
data_banco_xlsx.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 24299 entries, 0 to 24298
## Data columns (total 7 columns):
## Sucursal                24299 non-null int64
## Cajero                  24299 non-null int64
## ID_Transaccion          24299 non-null int64
## Transaccion             24299 non-null object
## Tiempo_Servicio_seg     24299 non-null float64
## Satisfaccion            24299 non-null object
## Monto                   24299 non-null object
## dtypes: float64(1), int64(3), object(3)
## memory usage: 1.3+ MB
```

Tipos de datos

Revisar que todas las columnas tengan el tipo correcto y además la relación que existe entre los dos dataframes importados.

```
data_sucursal.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 5 entries, 0 to 4
## Data columns (total 3 columns):
## ID_Sucursal      5 non-null int64
## Sucursal         5 non-null object
## Nuevo_Sistema    5 non-null object
## dtypes: int64(1), object(2)
## memory usage: 248.0+ bytes
```

5to. Corregir y ordenar los datos

Del punto anterior podemos ver que:

- Hay columnas numéricas que deben ser texto,
- Satisfacción debería ser categórica,
- El Monto debemos convertirno a numérico,
- La data de sucursales se puede agregar a la de transacciones por el código de la sucursal.

Para poder realizar eso debemos aprender a manejar dataFrames, esto es: seleccionar columnas, filtrar filas, modificar columnas, unir dos conjuntos de datos. Pero antes vamos a aprender Jupyter, otra IDE muy usada para ciencia de datos usando Python.

Fin

Curso: Bases para Data Science - Estadística, R y Python

Katherine Morales / Néstor Montaño