KIBERNUM
IT Academy

Módulo 2 – Obtención y Preparación de Datos

Data Wrangling

Ciencia de Datos

Contenidos

Data Wrangling I

- Qué es Data Wrangling.
- Muestreos Aleatorios.
- Duplicados.
- Transformación de datos.
- Expresiones regulares.
- Conversión de tipos de datos.
- Ordenamiento.
- Columnas e Índices.

KIBERNUM

Data Wrangling

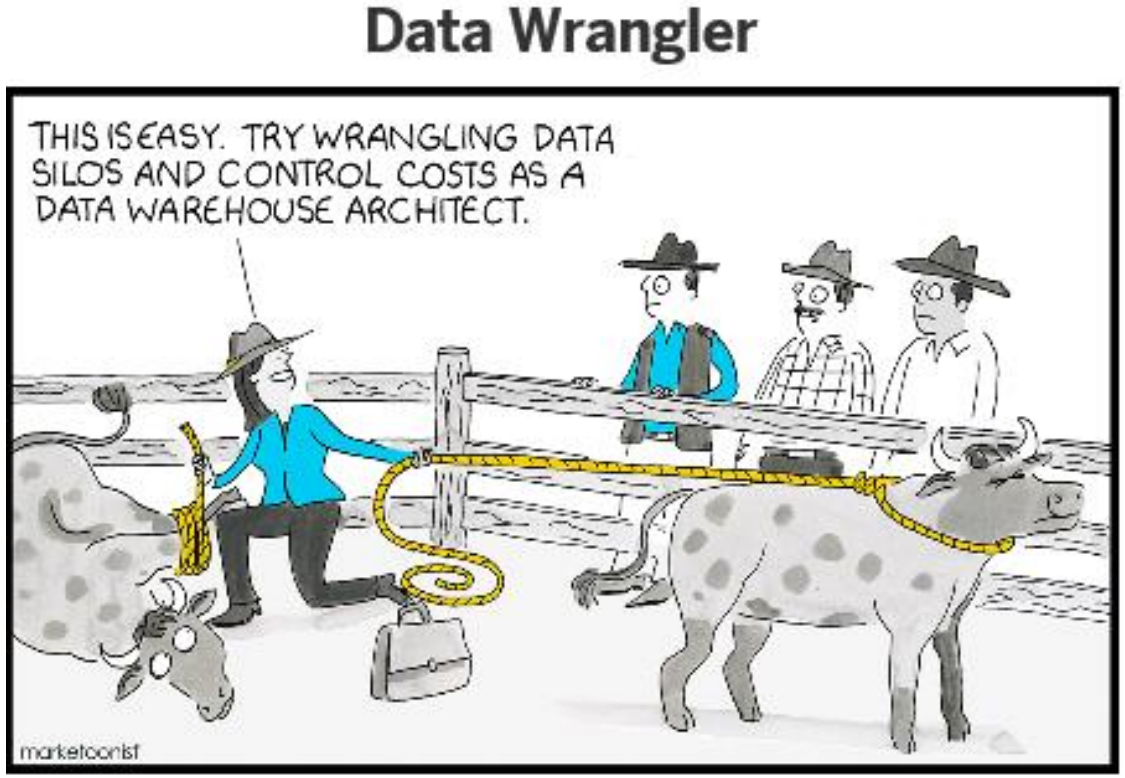
- Uno de los errores más comunes es pensar que los proyectos relacionados con analítica concentran su esfuerzo en la implementación y utilización de herramientas de análisis.

- Lo anterior no es correcto. La mayor parte del tiempo será destinado al trabajo de los datos, debido a que la forma natural de éstos (Raw Data) suelen tener un porcentaje considerable de errores que imposibilitan el análisis.

- Un analista puede dedicar en promedio el 80% de su tiempo a realizar Data Wranging.

KIBERNUM

Data Wrangling



"Esto es fácil. Intenta manejar silos de datos y controlar los costos como arquitecto de almacén de datos"

"¿Dónde está el nuevo empleado que contrataste para gestionar las migraciones de datos?

KIBERNUM

Data Wrangling

Descubrimiento
Entendimiento de los datos

Publicación
Preparación de la información para su posterior uso

Estructuración
Organización de las estructuras de datos

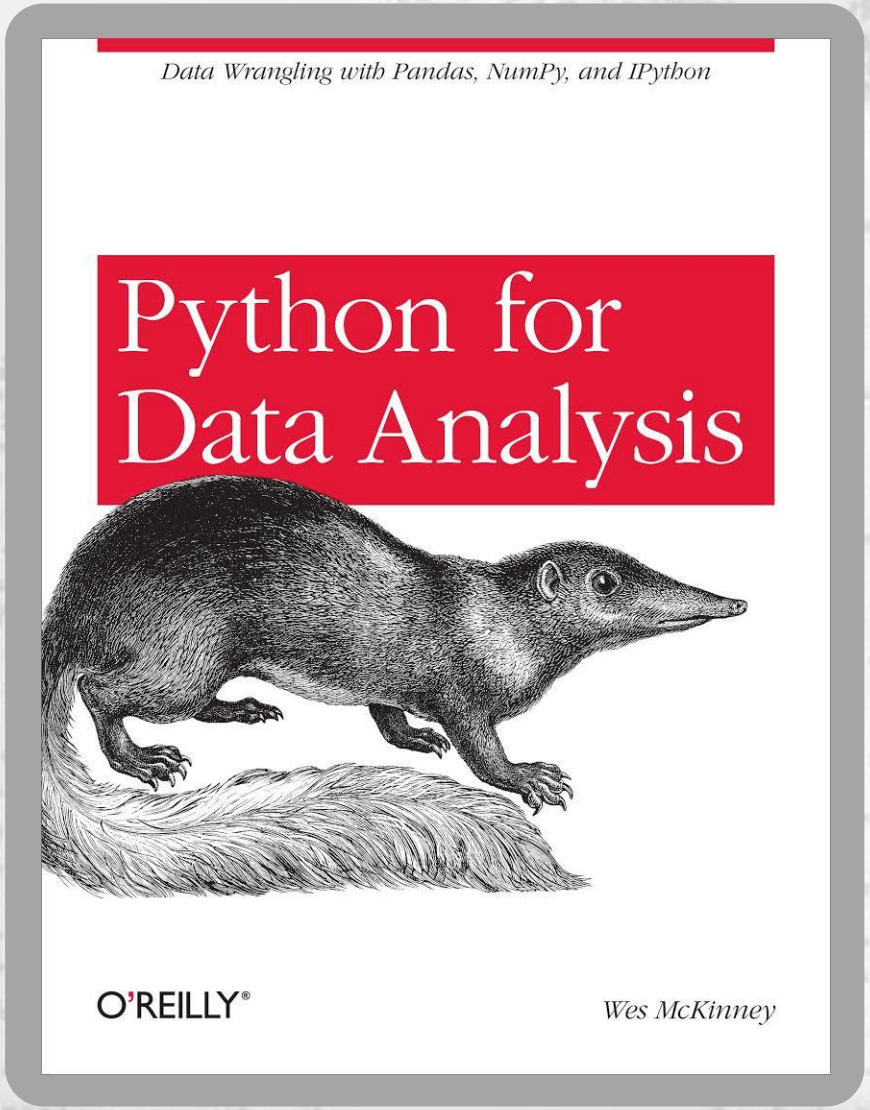
Validación
Verificación de la consistencia y calidad de los datos

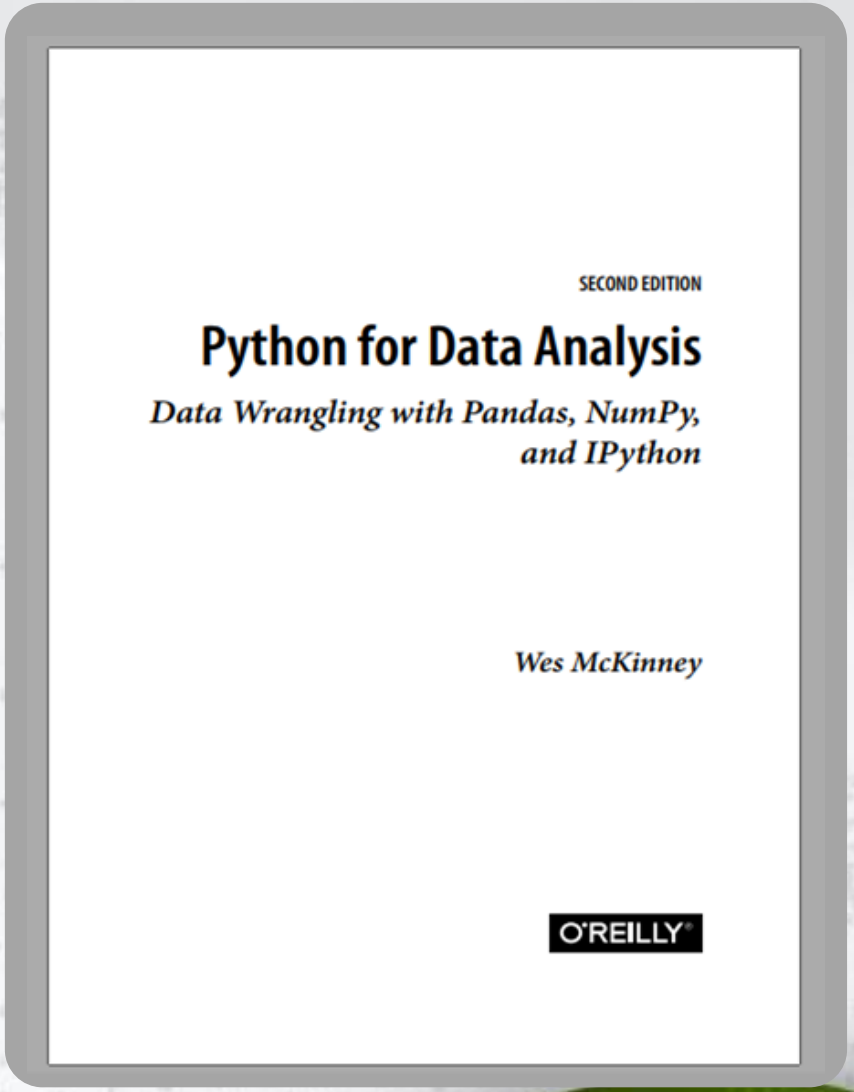
Limpieza
Null values, outliers

Enriquecimiento
Incorporación de nuevos datos, columnas, entre otros.

KIBERNUM

“Explorando a fondo”





KIBERNUM

Muestreo aleatorio

KIBERNUM

Muestreos Aleatorios

Como se ha visto anteriormente, es frecuente la utilización tanto, del método head() como del método tail() para la visualización de algunos registros del dataset. También, es posible utilizar el método sample() para obtener un conjunto de datos obtenidos de forma aleatoria.

```
df = pd.read_excel('resumen_sueldos.xlsx')
df.head(2)
```

	NOMBRE	SUELDO LIQUIDO
0	Cecilia Del Carmen Ayala Cabrera	1.803.344
1	Jesús Ignacio Contreras Vivar	236.489

```
df.sample(n=5)
```

	NOMBRE	SUELDO LIQUIDO
66	Debora Andrea Carvallo González	335.374
171	Jocelyn Trinidad Calderón Varas	1.409.140
103	Sergio Ernesto Lillo Rivillo	455.346
119	María Francisca Olivares Duarte	139.767
110	Carolina Lorena Marín Marín	769.306

Muestreos Aleatorios

Con el parámetro frac, se puede indicar qué fracción de la data se quiere obtener de forma aleatoria. Por defecto, retorna una proporción de filas del dataset, pero con el parámetro axis=1, puede retornar un dataframe con columnas seleccionadas de forma aleatoria.

```
df.sample(frac=.05)
```

	NOMBRE	SUELDO LIQUIDO
71	Hernán Roberto De Jesús Cataldo Olivares	2.030.574
5	Eugenio Leonardo Hidalgo Araya	315.293
107	Mariela Maldonado León	2.115.372
87	Javier Nicolás Fraser Hernández	823.149
115	Paola Denyts Núñez Palma	1.161.606
132	Fabián Rojas Andrade	197.280
33	Eugenio Eliecer Pereira Lazcano	669.360
172	Valeria Andrea Céspedes Vilches	448.028
110	Carolina Lorena Marín Marín	769.306

KIBERNUM

Duplicados

KIBERNUM

Identificación de valores duplicados

Los sets de datos pueden contener filas duplicadas en la información, lo cual puede provocar la doble contabilización en los análisis. Es conveniente identificarlas con el objeto de tomar una decisión sobre su tratamiento. Para esto, tomaremos el siguiente dataset de pasajeros que volaron en Estados Unidos entre 1949 y 1960.

```
df = pd.read_csv('flights-dups.csv')
df.head(2)
```

	year	month	passengers
0	1949	January	112
1	1949	February	118

El método duplicated retorna una serie de valores booleanos en donde se indican las filas duplicadas con valor True.

```
# señalar duplicados
df.duplicated()
```

```
0      False
1      False
2      False
3      False
4      False
       ...
143    False
144    False
145    False
146    False
147    False
Length: 148, dtype: bool
```

Si sumamos, obtenemos la cantidad total de registros duplicados en el dataset.

```
# contabilizar duplicados
df.duplicated().sum()
```

```
4
```

KIBERNUM

Identificación de valores duplicados

También, podemos contabilizar duplicados en ciertas columnas y los no duplicados, utilizando el operador negación (virgulilla alt-126).

```
# contabilizar duplicados en ciertas columnas
df.duplicated(subset=['year','month']).sum()
```

```
4
```

```
# contabilizar no duplicados en ciertas columnas
(~df.duplicated(subset=['year','month'])).sum()
```

```
144
```

```
# contabilizar no duplicados en todo el dataset
(~df.duplicated()).sum()
```

```
144
```

Identificación de las filas duplicados

Con esta expresión podemos visualizar los registros que están marcados como duplicados.

```
# identificación de las filas duplicados
df[df.duplicated()]
```

	year	month	passengers
6	1949	June	135
37	1951	December	166
123	1959	January	360
124	1959	January	360

```
df[(df['year']==1949) & (df['month']=='June')]
```

	year	month	passengers
5	1949	June	135
6	1949	June	135

```
df[(df['year']==1959) & (df['month']=='January')]
```

	year	month	passengers
122	1959	January	360
123	1959	January	360
124	1959	January	360

```
df[(df['year']==1951) & (df['month']=='December')]
```

	year	month	passengers
36	1951	December	166
37	1951	December	166

KIBERNUM

Eliminar duplicados de un dataset

- Después de haber identificado y analizado los registros duplicados de un dataset, el próximo paso podría ser su eliminación. El método drop_duplicates() nos ayuda en la eliminación de duplicados en un dataset. Este método también acepta el parámetro subset, con lo cual se puede acotar el rango de columnas donde localizar los registros duplicados.

- Por defecto, drop_duplicates() retorna una copia del dataframe con la operación realizada, pero no afecta el dataframe original a no ser que especifiquemos el parámetro inplace=True.

```
df.shape
```

```
(148, 3)
```

```
# eliminación de duplicados
df.drop_duplicates()
```

	year	month	passengers
0	1949	January	112
1	1949	February	118
2	1949	March	132
3	1949	April	129
4	1949	May	121
...	...	...	...
143	1960	August	606
144	1960	September	508
145	1960	October	461
146	1960	November	390
147	1960	December	432

144 rows × 3 columns

Retorna una copia del dataframe sin duplicados. Nótese que tiene 4 registros menos.

KIBERNUM

Columnas categóricas

Chequeo de columnas categóricas

Las variables categóricas pueden tener datos erróneos, o bien, sin estandarizar. Nótese cómo en la primera figura aparece “Téc. Enfermería” y “T. Enfermería”, los cuales, para efectos de análisis éstos serían representados como categorías distintas. Idem para “Administrativo” y “Administrativa”. Sin embargo, al ser un dataframe con demasiados registros, se puede perder la visualización.

```
# valores únicos de una serie
df['TITULO Y/O ESPECIALIDAD']
```

```
0                Enfermera
1           Tec. Enfermería
2             T. Enfermería
3                Odontólogo
4                 Conductor
                   ...
180    Químico Farmacéutico
181          Administrativo
182          Administrativa
183           T. Enfermería
184             Kinesiólogo
Name: TITULO Y/O ESPECIALIDAD, Length: 185, dtype: object
```

KIBERNUM

Chequeo de columnas categóricas

```
# valores únicos de una serie
df['TITULO Y/O ESPECIALIDAD'].unique()
```

```
array(['Enfermera', 'Tec. Enfermería', 'T. Enfermería',
'Odontólogo',
       'Conductor', 'Chofer', 'A. Social', 'Kinesiólogo',
       'Técnico Odontológico', 'Médico', 'Matrona', 'Admi
nistrativa',
       'Tec. Administración De Empresa', 'Psicólogo', 'As
istente Dental',
       'Profesor Educación Física', 'Administrativo',
       'Químico Farmacéutico', 'Medico Cirujano', 'Nutric
ionista',
       'Ed. De Párvulos', 'Med.Cirujano', 'Aux. Paramédic
o',
       'Nutrición Y Dietética', 'Tecnólogo Médico', 'Fono
audióloga',
       'Secretaria Ejecutiva', 'Ing. Comercial', 'Auxilia
r De Servicio',
       'Psicopedagoga', 'Matron  ', 'Técnico Jurídico',
       'Tec. En Adm De Empresa',
       'Ingeniero En Adm De Empresa Mención Rrhh',
       'Ingeniero Informático', 'Contador General',
       'Tec. Nivel Superior De Analista Programa',
       'Terapeuta  Ocupacional', 'Contador Auditor',
       'Ing. En Administración De Empresas', 'Ingeniera e
n Finanzas',
       'Técnico Administración', 'Técnico En Prevención D
e Riesgos'],
      dtype=object)
```

El método **unique()** permite visualizar los valores únicos de una serie de datos. De esta forma, se pueden identificar de forma fácil los problemas de estandarización y errores.

```
# cantidad de valores únicos
df['TITULO Y/O ESPECIALIDAD'].nunique()
```

```
43
```

Por otra parte, si lo que buscamos es conocer cuántos valores distintos hay en la serie de datos, podemos utilizar el método **nunique()**.

KIBERNUM

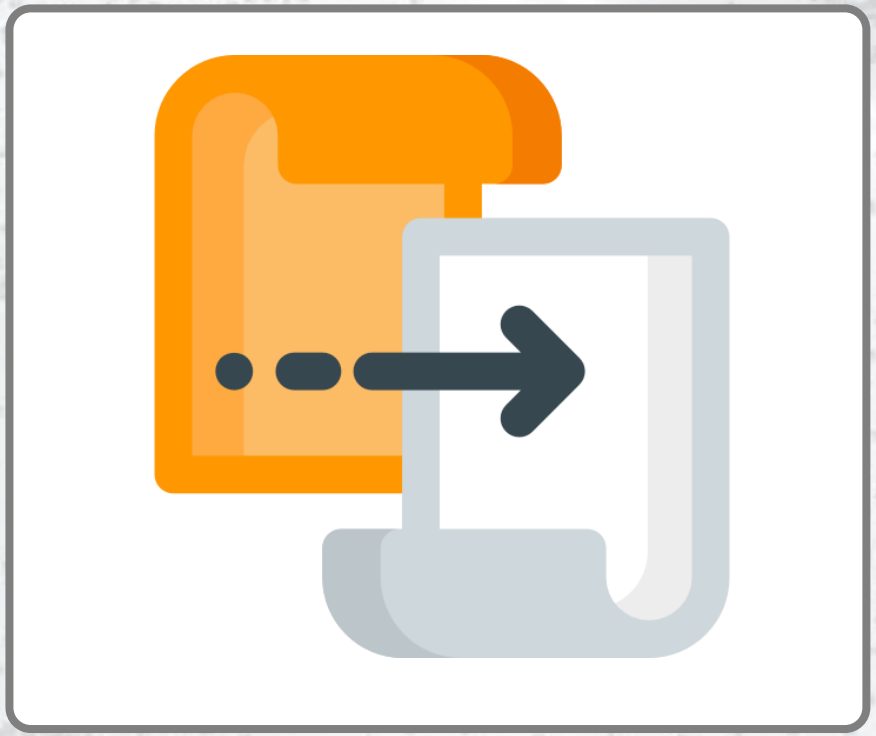
Chequeo de columnas categóricas

Otro método muy útil es **value_counts()**, que muestra de forma ordenada las clases y su frecuencia.

```
# contabilizar cuántos elementos de cada categoría
df['TITULO Y/O ESPECIALIDAD'].value_counts()
```

T. Enfermería	30
Tec. Enfermería	18
Enfermera	14
Administrativo	11
Odontólogo	10
Kinesiólogo	10
Matrona	8
A. Social	7
Medico Cirujano	6
Técnico Odontológico	6
Psicólogo	6
Nutricionista	5
Conductor	5
Aux. Paramédico	4
Med.Cirujano	4
Químico Farmacéutico	4
Asistente Dental	3
Administrativa	3
Ingeniero Informático	2
Ed. De Párvulos	2
Técnico Administración	2
Contador Auditor	2
Chofer	2
Ing. Comercial	2
Nutrición Y Dietética	1
Tec. Nivel Superior De Analista Programa	1
Contador General	1
Tec. En Adm De Empresa	1
Ing. En Administración De Empresas	1

Transformación de Datos



KIBERNUM

Transformando una serie

- A continuación, vamos a definir una serie de datos. El objetivo es transformar cada uno de los elementos que compone la serie con la función cubo(), que también ha sido definida.
- Como se puede apreciar, se creó una nueva serie vacía y se realizó una iteración, elemento a elemento, para calcular el cubo de cada elemento y agregarlo a la serie de salida.

```python
# Sea la siguiente serie de datos
serie = pd.Series(range(10,20,2))
serie
```

✓ 0.0s Python

```
0    10
1    12
2    14
3    16
4    18
dtype: int64
```

```python
# sea la siguiente función
def cubo(x):
    return x**3
```

✓ 0.0s Python

```python
# forma tradicional de transformar una serie
out = pd.Series(dtype=int)
for i,e in serie.iteritems():
    out.at[i] = cubo(e)
print(out)
```

✓ 0.0s Python

```
0    1000
1    1728
2    2744
3    4096
4    5832
dtype: int64
```

KIBERNUM

Transformando una serie

- El mismo resultado se puede obtener utilizando el método apply() de una serie, que retorna una nueva serie de datos con los elementos en el cual se le ha aplicado la función especificada. De esta forma, se produce el efecto de transformación de la serie original.
- El mismo resultado se puede obtener utilizando directamente una expresión lambda.

Una **expresión lambda** es una función anónima (función literal), no está enlazada a un identificador.

```
# aplicamos la función cubo a cada uno de los elementos
# de la serie
serie.apply(cubo)
```

```
0    1000
1    1728
2    2744
3    4096
4    5832
dtype: int64
```

```
# utilizando expresiones lambda
serie.apply(lambda x : x**3)
```

```
0    1000
1    1728
2    2744
3    4096
4    5832
dtype: int64
```

KIBERNUM

Transformando una serie

Un caso de uso muy práctico es la limpieza de columnas en un dataframe. En este caso, la columna **OWN_OCCUPIED** del dataframe de **real estate** tiene valores que fueron detectados como erróneos.

```
# limpienza de columnas
df = pd.read_csv('real-estate.csv')
df['OWN_OCCUPIED']
```

```
0      Y
1      N
2      N
3     12
4      Y
5      Y
6    NaN
7      Y
8      Y
Name: OWN_OCCUPIED, dtype: object
```

```
df['OWN_OCCUPIED'].unique()
```

```
array(['Y', 'N', '12', nan], dtype=object)
```

KIBERNUM

Transformando una serie

Un caso de uso muy práctico es la limpieza de columnas en un DataFrame. En este caso, definimos previamente una función que sólo retorna 'Y' o 'N' de acuerdo con una definición razonable.

Finalmente, aplicamos la función y reescribimos la serie.

```
# limpieza de la serie de datos
def limpiar(valor):
    if valor == 'N':
        return valor
    else:
        return 'Y'
```

```
df['OWN_OCCUPIED'] = df['OWN_OCCUPIED'].apply(limpiar)
```

```
df['OWN_OCCUPIED']
```

```
0    Y
1    N
2    N
3    Y
4    Y
5    Y
6    Y
7    Y
8    Y
Name: OWN_OCCUPIED, dtype: object
```

Lo mismo en una sola línea.

```
# lo mismo en una sola linea
df['OWN_OCCUPIED'].apply(lambda x : x if x=='N' else 'Y')
```

```
0    Y
1    N
2    N
3    Y
4    Y
5    Y
6    Y
7    Y
8    Y
Name: OWN_OCCUPIED, dtype: object
```

KIBERNUM

Transformando un dataframe

Los dataframes también, cuentan con el método **apply()**, en ese caso, lo que se itera es una fila del dataframe, o bien, una columna (si se indica el parámetro **axis=1**).

```
# transformando un dataframe
df = pd.read_csv('titanic.csv')
df.head(2)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C

```
df.isnull().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

Nótese que la columna Age tiene valores nulos. Realizaremos una imputación

KIBERNUM

Transformando un dataframe

Vamos a realizar una imputación estratificada en la columna Age. Nótese que si se analiza la edad promedio de los pasajeros del Titanic de acuerdo a la clase de pasajero (Pclass), se observa que en promedio los pasajeros que viajaron en clases superiores tenían un promedio de edad mayor. Por tanto, resulta razonable realizar la imputación estratificada.

```
# imputación estratificada
df[['Age','Pclass']].groupby('Pclass').mean()
```

	Age
Pclass	
1	38.233441
2	29.877630
3	25.140620

KIBERNUM

Transformando un dataframe

Para realizar la imputación, hemos definido una función que toma dos parámetros de entrada: pclass y age. También, se realizan algunos tests para verificar el correcto funcionamiento.

```
import numpy as np
```

```
def imputar_edad(pclass, age):
    # si la edad es nula, realizamos la imputacion
    if pd.isnull(age):
        if pclass == 1:
            return 38.23
        elif pclass == 2:
            return 29.67
        elif pclass == 3:
            return 25.14
    else:
    # caso contrario, retornamos el mismo valor
        return age
```

```
# algunas pruebas
print(imputar_edad(1, np.nan))
print(imputar_edad(2, np.nan))
print(imputar_edad(3, np.nan))
print(imputar_edad(1, 35))
```

```
38.23
29.67
25.14
35
```

KIBERNUM

Transformando un dataframe

Al utilizar el método **apply()** en un dataframe, se debe especificar si se desea iterar filas o columnas. En este caso, iteraremos filas del dataframe, por esto, debemos especificar el parámetro **axis=1**. Cada valor de la variable row corresponde a la serie de datos correspondiente a la columna iterada.

```
# aplicamos la función
df['Age']=df[['Age','Pclass']].apply(lambda row : imputar_edad(row['Pclass'],row['Age']), axis=1)
```

```
df.isnull().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

Expresiones Regulares

KIBERNUM

Expresiones Regulares

Hay oportunidades en que es necesario aplicar patrones de búsqueda para la limpieza o transformación de datos. Por ejemplo, para verificar que una columna cumple con el formato adecuado, o bien, para la creación de nuevas columnas.

```
df = pd.read_csv('titanic.csv')
df['Name'][:10]
```

```
0                              Braund, Mr. Owen Harris
1    Cumings, Mrs. John Bradley (Florence Briggs Th...
2                               Heikkinen, Miss. Laina
3         Futrelle, Mrs. Jacques Heath (Lily May Peel)
4                             Allen, Mr. William Henry
5                                     Moran, Mr. James
6                              McCarthy, Mr. Timothy J
7                       Palsson, Master. Gosta Leonard
8    Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)
9                  Nasser, Mrs. Nicholas (Adele Achem)
Name: Name, dtype: object
```

Supongamos que necesitamos verificar que los valores de la columna Name cumplen con el siguiente formato:

Braund, Mr. Owen Harris

LastName, Title. Name MiddleName

KIBERNUM

Expresiones Regulares

Una Expresión Regular es una secuencia de caracteres que definen un patrón de búsqueda. En este caso, utilizaremos una expresión regular para verificar si una cadena de caracteres respeta el patrón definido.

Importamos la librería

Definimos el patrón

```
import re

# creamos una expresión regular
p = re.compile('^([A-Za-z]*), (([A-Za-z]*))\. (([A-Za-z]*)) .*$')

# hacemos una prueba
s = 'Braund, Mr. Owen Harris'
print(bool(p.match(s)))
```

```
True
```

Utilizamos el patrón

En este curso, no abordaremos en profundidad la creación de expresiones regulares, pero se recomienda revisar algunos tutoriales. Puede comenzar por estos links:

https://regexone.com/
https://www.w3schools.com/python/python_regex.asp

KIBERNUM

Expresiones Regulares

Ahora, mediante el método apply(), vamos a aplicar el patrón a todos los elementos de la serie de datos correspondiente a la columna Name. Note que en el resultado se aprecia que hubo algunos registros que no hicieron match.

```
# ahora verificamos si la columna cumple el patrón
df['Name'].apply(lambda x : bool(p.match(x)))
```

```
0       True
1       True
2      False
3       True
4       True
       ...
886    False
887     True
888     True
889     True
890    False
Name: Name, Length: 891, dtype: bool
```

KIBERNUM

Expresiones Regulares

Exploremos, entonces, cuáles fueron los registros donde la expresión regular no hizo match. Como se puede apreciar, estos registros no hicieron match debido a que no poseen un **middle name**.

```
df[~df['Name'].apply(lambda x : bool(p.match(x)))]['Name']
```

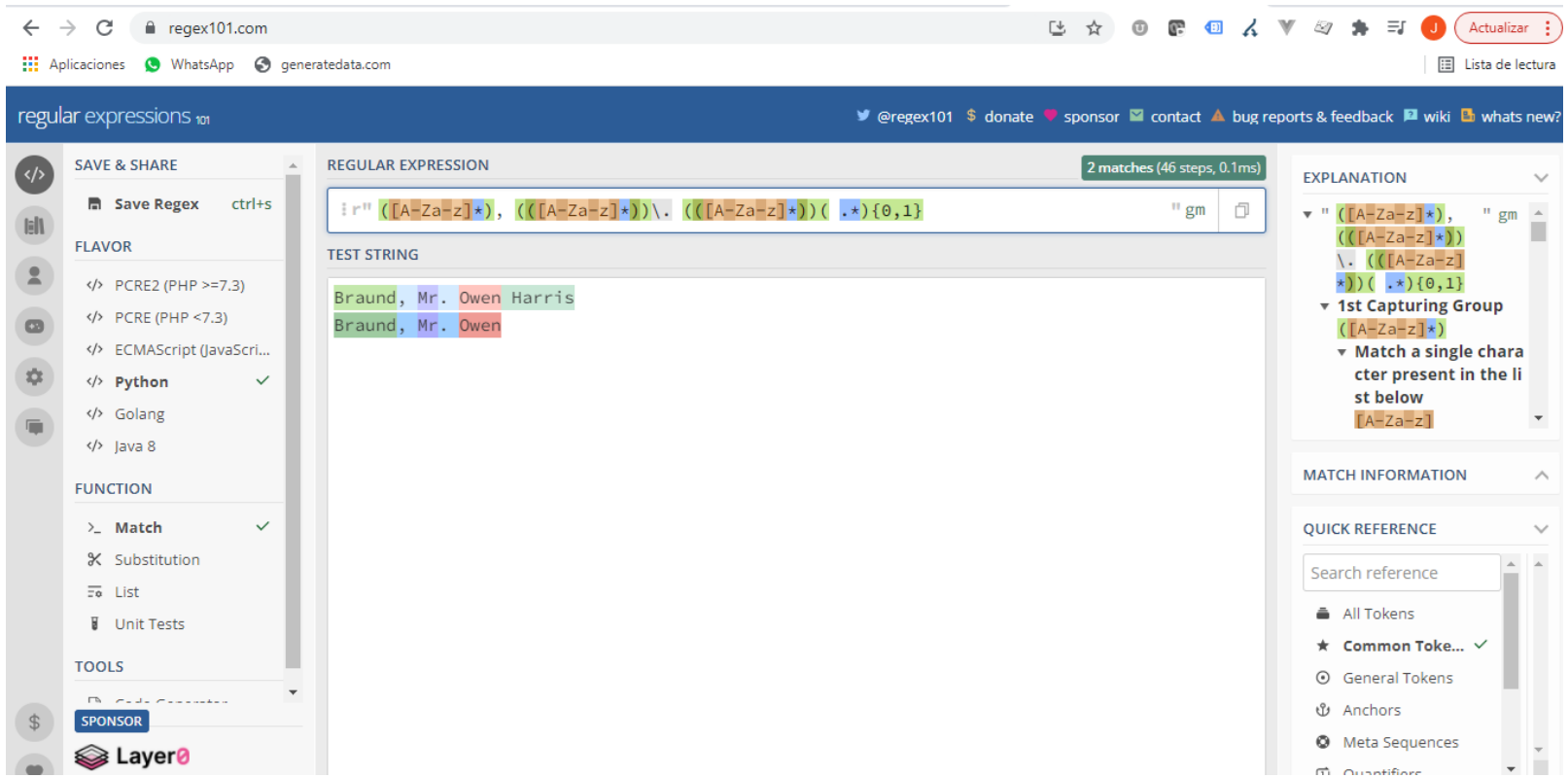
```
2                  Heikkinen, Miss. Laina
5                        Moran, Mr. James
11                Bonnell, Miss. Elizabeth
15     Hewlett, Mrs. (Mary D Kingcome)
16                   Rice, Master. Eugene
                        ...
877                  Petroff, Mr. Nedelio
878                    Laleff, Mr. Kristo
881                   Markun, Mr. Johann
886                 Montvila, Rev. Juozas
890                   Dooley, Mr. Patrick
Name: Name, Length: 352, dtype: object
```

KIBERNUM

Expresiones Regulares

Podemos mejorar la expresión regular para que acepte como válidos los nombres que no presentan un Middle Name. Se recomienda probar los patrones con un testeador de expresiones regulares, tal como:

http://regex101.com



http://regex101.com

KIBERNUM

Conversión de tipos de datos

KIBERNUM

Conversión de tipos de datos

La librería Pandas, al momento de crear un dataframe, durante la lectura de los datos, realiza una inferencia de los tipos de dato de cada columna a partir de los valores que contiene. En la mayoría de los casos, hace una asignación adecuada, sin embargo, en oportunidades será necesario realizar ajustes en la estructura. Esto, considerando que durante los procesos de limpieza y wrangling podrían verse afectados los tipos de dato.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9 entries, 0 to 8
Data columns (total 7 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   PID            8 non-null      float64
 1   ST_NUM         7 non-null      float64
 2   ST_NAME        9 non-null      object
 3   OWN_OCCUPIED   8 non-null      object
 4   NUM_BEDROOMS   5 non-null      float64
 5   NUM_BATH       8 non-null      object
 6   SQ_FT          7 non-null      float64
dtypes: float64(4), object(3)
memory usage: 632.0+ bytes
```

Estos valores podrían se int

Lo reconoció como object , este campo podría ser boolean

Lo reconoció como object , debería haberlo identificado como dato numérico. Eso indica que hay una posible inconsistencia.

KIBERNUM

Conversiones de datos

El método **astype()** recibe como parámetro el tipo de dato para la conversión. El parámetro puede ser un tipo de dato **Pandas**, **Numpy** o **Python**.

```
df['NUM_BATH'] = df['NUM_BATH'].astype(float)
```

```
df.dtypes
```

```
PID              float64
ST_NUM           float64
ST_NAME           object
OWN_OCCUPIED      object
NUM_BEDROOMS     float64
NUM_BATH         float64
SQ_FT            float64
dtype: object
```

```
df['OWN_OCCUPIED'] = df['OWN_OCCUPIED'].astype(bool)
```

```
df.dtypes
```

```
PID              float64
ST_NUM           float64
ST_NAME           object
OWN_OCCUPIED        bool
NUM_BEDROOMS     float64
NUM_BATH         float64
SQ_FT            float64
dtype: object
```

KIBERNUM

Conversiones de datos

Los siguientes, son los tipos de datos que podríamos utilizar para realizar conversión. Nótese la comparativa de los distintos tipos de datos utilizados tanto por: la librería estándar de Python, la librería Pandas y la librería Numpy.

Pandas dtype	Python type	NumPy type	Usage
object	str or mixed	string_, unicode_, mixed types	Text or mixed numeric and non-numeric values
int64	int	int_, int8, int16, int32, int64, uint8, uint16, uint32, uint64	Integer numbers
float64	float	float_, float16, float32, float64	Floating point numbers
bool	bool	bool_	True/False values
datetime64	NA	datetime64[ns]	Date and time values
timedelta[ns]	NA	NA	Differences between two datetimes
category	NA	NA	Finite list of text values

KIBERNUM

Conversiones de datos

```
df = pd.read_csv('real-estate.csv')
df
```

	PID	ST_NUM	ST_NAME	OWN_OCCUPIED	NUM_BEDROOMS	NUM_BATH	SQ_FT
0	100001000.0	104.0	PUTNAM	Y	3	1	1000
1	100002000.0	197.0	LEXINGTON	N	3	1.5	--
2	100003000.0	NaN	LEXINGTON	N	NaN	1	850
3	100004000.0	201.0	BERKELEY	12	1	NaN	700
4	NaN	203.0	BERKELEY	Y	3	2	1600
5	100006000.0	207.0	BERKELEY	Y	NaN	1	800
6	100007000.0	NaN	WASHINGTON	NaN	2	HURLEY	950
7	100008000.0	213.0	TREMONT	Y	1	1	NaN
8	100009000.0	215.0	TREMONT	Y	na	2	1800

Con lo aprendido, vamos a tomar entonces el set de datos real-estate para realizar la conversión de la columna NUM_BATROOMS. (En este caso, se ha tomado el set de datos sin limpiar aún).

Conversiones de datos

Al intentar hacer la conversión, se lanza una excepción donde se indica que hubo un valor que no se pudo convertir, por lo tanto, la conversión no fue exitosa. Esto quiere decir, que la columna tiene valores erróneos, por lo tanto, debemos utilizar otra técnica.

```
: df['NUM_BATH'].astype(float)

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-154-1619a4971e6d> in <module>
----> 1 df['NUM_BATH'].astype(float)

~\Anaconda3\envs\dataanalysis\lib\site-packages\pandas\core\generic.py in astyp
e(self, dtype, copy, errors)
   5696         else:
   5697             # else, only a single dtype is given
-> 5698             new_data = self._data.astype(dtype=dtype, copy=copy, errors
=errors)
    898
    899     return arr.view(dtype)

ValueError: could not convert string to float: 'HURLEY'
```

KIBERNUM

Conversiones de datos

La nueva estrategia consiste, entonces, en crear una función personalizada para la conversión de dicha columna. En esta función, se capturan las excepciones que son lanzadas cuando no es posible realizar la conversión, en cuyo caso se asigna un valor nan.

```python
# otra técnica de conversión
def convertir(valor):
    try:
        return float(valor)
    except:
        return np.nan
```

```python
df['NUM_BATH'].apply(convertir)
```

```
0    1.0
1    1.5
2    1.0
3    NaN
4    2.0
5    1.0
6    NaN
7    1.0
8    2.0
Name: NUM_BATH, dtype: float64
```

Nótese que ahora la serie de datos es de tipo float64

Ordenamiento

Ordenar un dataframe

- Tomemos el dataset de sueldos, vamos a ordenar los registros de acuerdo con su sueldo, de forma descendente.
- Pero antes de proceder, revisemos su estructura para verificar que la columna es numérica.

```
df = pd.read_excel('sueldos.xlsx')
df.head(2)
```

	_id	NOMBRE	TITULO Y/O ESPECIALIDAD	LABOR	LUGAR DE SU FUNCION	SUELDO LIQUIDO
0	1	Cecilia Del Carmen Ayala Cabrera	Enfermera	Encargada Del Cecosf	Cecosf Padre Hugo Cornelissen	1.803.344
1	2	Jesús Ignacio Contreras Vivar	Tec. Enfermería	Despacho De Medicamentos, Pnac Pacam	Farmacia/Coordinación	236.489

Ordenar un dataframe

Tomemos el dataset de sueldos, vamos a ordenar los registros de acuerdo con su sueldo, de forma descendente.

Pero antes de proceder, revisemos su estructura para verificar que la columna es numérica.

Como se puede apreciar, la columna está definida como object, esto puede deberse a que la librería Pandas interpreta los puntos como separadores decimales. Realizaremos un pequeño wrangling antes de proceder.

```
df = pd.read_excel('sueldos.xlsx')
df.head(2)
```

	_id	NOMBRE	TITULO Y/O ESPECIALIDAD	LABOR	LUGAR DE SU FUNCION	SUELDO LIQUIDO
0	1	Cecilia Del Carmen Ayala Cabrera	Enfermera	Encargada Del Cecosf	Cecosf Padre Hugo Cornelissen	1.803.344
1	2	Jesús Ignacio Contreras Vivar	Tec. Enfermería	Despacho De Medicamentos, Pnac Pacam	Farmacia/Coordinación	236.489

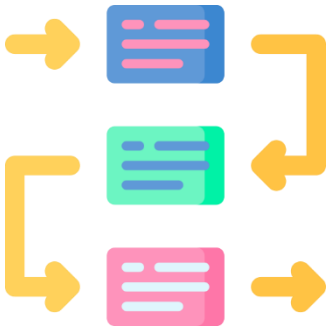
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 185 entries, 0 to 184
Data columns (total 6 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   _id                     185 non-null    int64
 1   NOMBRE                  185 non-null    object
 2   TITULO Y/O ESPECIALIDAD 185 non-null    object
 3   LABOR                   185 non-null    object
 4   LUGAR DE SU FUNCION     185 non-null    object
 5   SUELDO LIQUIDO          185 non-null    object
dtypes: int64(1), object(5)
memory usage: 8.8+ KB
```

KIBERNUM

Ordenamiento: Ordenar un dataframe

Ahora el dataframe cuenta con una estructura en donde la columna **sueldo líquido** es de tipo entero.



```
: df['SUELDO LIQUIDO'] = df['SUELDO LIQUIDO'].apply(lambda x : x.replace('.',''))
  df['SUELDO LIQUIDO'] = df['SUELDO LIQUIDO'].astype(int)
```

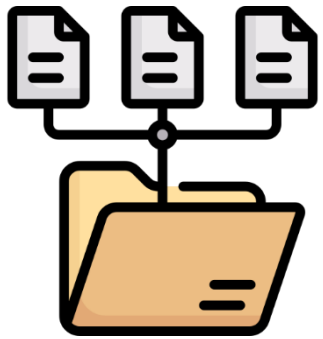
```
: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 185 entries, 0 to 184
Data columns (total 6 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   _id                     185 non-null    int64
 1   NOMBRE                  185 non-null    object
 2   TITULO Y/O ESPECIALIDAD 185 non-null    object
 3   LABOR                   185 non-null    object
 4   LUGAR DE SU FUNCION     185 non-null    object
 5   SUELDO LIQUIDO          185 non-null    int32
dtypes: int32(1), int64(1), object(4)
memory usage: 8.1+ KB
```

KIBERNUM

Ordenar un dataframe

Y, por último, realizamos el ordenamiento del dataframe de forma descendente de acuerdo con la columna sueldo líquido utilizando el método sort_values().



```
df.sort_values(by='SUELDO LIQUIDO', ascending=False).head()
```

	_id	NOMBRE	TITULO Y/O ESPECIALIDAD	LABOR	LUGAR DE SU FUNCION	SUELDO LIQUIDO
169	170	Marcela Guida Brito Báez	Enfermera	Directora De Salud Municipal	Dirección	3158100
155	156	Patricia Jeaneth Valle Moran	Med.Cirujano	Med.Cirujano	Atención Morbilidad/Sector Verde	2411245
45	46	Aylin Acevedo Vera	Med.Cirujano	Med.Cirujano	Box Morbilidad/Sector Verde	2372045
76	77	Ada Evelyn Cortes Contreras	Odontólogo	Odontólogo	Unidad Dental/ Sector Amarillo	2177717
107	108	Mariela Maldonado León	Ing. Comercial	Directora	Dirección	2115372

Columnas e Índices

KIBERNUM

Columnas e Índices: Renombrar Columnas

El método **rename()** permite, dentro de otras cosas, renombrar las columnas. Para esto, debe proporcionarse un diccionario con los nombres actuales y nuevos nombres de las columnas que se desea modificar. Recuerde que este método utiliza el parámetro inplace. Si **inplace=False**, el método retorna una copia del datafram.

```
df = pd.read_excel('sueldos.xlsx')
df.head(2)
```

	_id	NOMBRE	TITULO Y/O ESPECIALIDAD	LABOR	LUGAR DE SU FUNCION	SUELDO LIQUIDO
0	1	Cecilia Del Carmen Ayala Cabrera	Enfermera	Encargada Del Cecosf	Cecosf Padre Hugo Cornelissen	1.803.344
1	2	Jesús Ignacio Contreras Vivar	Tec. Enfermería	Despacho De Medicamentos, Pnac Pacam	Farmacia/Coordinación	236.489

```
# renombrar columnas
df.rename(columns={'_id':'PID', 'LUGAR DE SU FUNCION':'UBICACION'}, inplace=True)
```

```
# entrega listado de columnas del df
df.columns
```

```
Index(['PID', 'NOMBRE', 'TITULO Y/O ESPECIALIDAD', 'LABOR', 'UBICACION',
       'SUELDO LIQUIDO'],
      dtype='object')
```

KIBERNUM

Setear un índice

A continuación, definiremos que la columna PID (anteriormente llamada _id) ahora será el índice de la serie, sobrescribiendo el índice por defecto que partía en cero. Este método también posee el parámetro **inplace**.

```
df.set_index('PID', inplace=True)
df.head()
```

Nuevo índice

PID	NOMBRE	TITULO Y/O ESPECIALIDAD	LABOR	UBICACION	SUELDO LIQUIDO
1	Cecilia Del Carmen Ayala Cabrera	Enfermera	Encargada Del Cecosf	Cecosf Padre Hugo Cornelissen	1.803.344
2	Jesús Ignacio Contreras Vivar	Tec. Enfermería	Despacho De Medicamentos, Pnac Pacam	Farmacia/Coordinación	236.489
3	Carolina Andrea Estay Pangue	T. Enfermería	Preparación De Pacientes/Coordinación	Farmacia/Coordinación	664.647
4	Jorge Eduardo García Lagos	Odontólogo	Encargado De Reas	Unidad Dental	1.279.353
5	Carolina Lissett Gómez Morales	Conductor	Estafeta Y Conductor	Cecosf Padre Hugo Cornelissen	255.036

Resetear un índice

De forma análoga, al resetear un índice, éste es promovido a columna y por lo tanto, la información no se pierde. Por otra parte, se asigna el índice por defecto sin nombre. Al igual que los otros casos, este método también utiliza el parámetro inplace.

```
# resetear el índice
df.reset_index(inplace=True)
df.head()
```

	PID	NOMBRE	TITULO Y/O ESPECIALIDAD	LABOR	UBICACION	SUELDO LIQUIDO
0	1	Cecilia Del Carmen Ayala Cabrera	Enfermera	Encargada Del Cecosf	Cecosf Padre Hugo Cornelissen	1.803.344
1	2	Jesús Ignacio Contreras Vivar	Tec. Enfermería	Despacho De Medicamentos, Pnac Pacam	Farmacia/Coordinación	236.489
2	3	Carolina Andrea Estay Pangue	T. Enfermería	Preparación De Pacientes/Coordinación	Farmacia/Coordinación	664.647
3	4	Jorge Eduardo García Lagos	Odontólogo	Encargado De Reas	Unidad Dental	1.279.353
4	5	Carolina Lissett Gómez Morales	Conductor	Estafeta Y Conductor	Cecosf Padre Hugo Cornelissen	255.036

Remover columnas

Para remover una o varias columnas de un dataframe, aplicamos la función drop.

El parámetro **axis=1** indica que se realizará la operación en una columna, inplace=False devuelve una copia del dataframe con la operación aplicada.

Si se opta por **inplace=True**, se modifica el dataframe original.

```
# eliminación de una columna
df.drop('UBICACION',axis=1).head()
```

	PID	NOMBRE	TITULO Y/O ESPECIALIDAD	LABOR	SUELDO LIQUIDO
0	1	Cecilia Del Carmen Ayala Cabrera	Enfermera	Encargada Del Cecosf	1.803.344
1	2	Jesús Ignacio Contreras Vivar	Tec. Enfermería	Despacho De Medicamentos, Pnac Pacam	236.489
2	3	Carolina Andrea Estay Pangue	T. Enfermería	Preparación De Pacientes/Coordinación	664.647
3	4	Jorge Eduardo García Lagos	Odontólogo	Encargado De Reas	1.279.353
4	5	Carolina Lissett Gómez Morales	Conductor	Estafeta Y Conductor	255.036

```
# eliminación de varias columnas
df.drop(['UBICACION','TITULO Y/O ESPECIALIDAD'],axis=1).head()
```

	PID	NOMBRE	LABOR	SUELDO LIQUIDO
0	1	Cecilia Del Carmen Ayala Cabrera	Encargada Del Cecosf	1.803.344
1	2	Jesús Ignacio Contreras Vivar	Despacho De Medicamentos, Pnac Pacam	236.489
2	3	Carolina Andrea Estay Pangue	Preparación De Pacientes/Coordinación	664.647
3	4	Jorge Eduardo García Lagos	Encargado De Reas	1.279.353
4	5	Carolina Lissett Gómez Morales	Estafeta Y Conductor	255.036

Dudas y consultas

Fin de la Presentación

KIBERNUM