

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación



IIC2115 - Programación como Herramienta para la Ingeniería

Análisis de datos - Exploración, limpieza y depuración

Profesora: Francesca Lucchini
Prof. Coordinador: Hans Löbel

¿Qué es el análisis de datos en Python?

- Esencialmente, buscamos responder preguntas relevantes, o descubrir aspectos desconocidos, en base a la evidencia dada por los datos
- Desde un punto de vista práctico, consiste principalmente en utilizar herramientas para:
 - Limpiar y transformar los datos
 - Explorar distintas dimensiones de los datos
 - Calcular estadísticas de los datos
 - Visualizar los datos
 - Construir modelos predictivos

Librerías Open-Source, permiten uso comercial



En esta primera parte nos centraremos en **Pandas**

- Permite manipular, analizar y visualizar datos.
- Puede ser vista como una herramienta para trabajar datos almacenados en una estructura de tabla o de serie de tiempo.
- Se basa en, y generaliza a, la librería **Numpy**.
- 2 Estructuras principales
 - **Series**
 - **DataFrame**



En un DataFrame, cada columna es un Series

	Comuna	Manzana	Predial	Línea de construcción	Material estructural	Calidad construcción	Año construcción
0	9201	1	1	1	E	4	1940
1	9201	1	1	2	E	4	1960
2	9201	1	2	1	E	4	1930
3	9201	1	3	1	E	4	1960
4	9201	1	4	1	E	3	1925

El primero paso siempre es explorar los datos

- Abrirlos y describirlos
- Identificar la presencia de valores nulos y outliers
- Corregir lo que corresponda

Vamos a Jupyter Lab...



```

1 import pandas as pd
2 import numpy as np
3
4
5 df = pd.read_csv("data.csv")

```

```

1 display(df.describe())

```

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	592.000000	600.000000	564.000000
mean	5403.459283	1621.245798	146.412162	342.000000	0.842199
std	6109.041673	2926.248369	85.587325	65.12041	0.364878
min	150.000000	0.000000	9.000000	12.000000	0.000000
25%	2877.500000	0.000000	100.000000	360.000000	1.000000
50%	3812.500000	1188.500000	128.000000	360.000000	1.000000
75%	5795.000000	2297.250000	168.000000	360.000000	1.000000
max	81000.000000	41667.000000	700.000000	480.000000	1.000000

```

1 df['Property_Area'].value_counts()

```

```

Semiurban    233
Urban         202
Rural         179
Name: Property_Area, dtype: int64

```



```

1 def conteo_nulo(x):
2     return sum(x.isnull())
3
4 df.apply(conteo_nulo, axis = 0)

```

```

Loan_ID          0
Gender           13
Married          3
Dependents       15
Education        0
Self_Employed    32
ApplicantIncome  0
CoapplicantIncome 0
LoanAmount       22
Loan_Amount_Term 14
Credit_History   50
Property_Area     0
Loan_Status      0
dtype: int64

```

```

1 df['LoanAmount'].fillna(df['LoanAmount'].mean(), inplace=True)

```

```

1 df.apply(conteo_nulo, axis = 0)

```

```

Loan_ID          0
Gender           13
Married          3
Dependents       15
Education        0
Self_Employed    32
ApplicantIncome  0
CoapplicantIncome 0
LoanAmount       0
Loan_Amount_Term 14
Credit_History   50
Property_Area     0
Loan_Status      0
dtype: int64

```

Muchas veces exploraremos múltiples fuentes

- Cuando todo está en un (1) DataFrame, la cosa fluye...
- Pero la mayoría de las veces, tenemos más de uno
- **Pandas** entrega varios mecanismos para enfrentar esto

Vamos a Jupyter Lab...



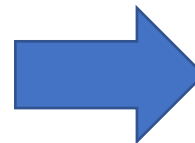
```

1 def make_df(cols, ind):
2     data = {c: [str(c) + str(i) for i in ind] for c in cols}
3     return pd.DataFrame(data, ind)

```

```
1 make_df('ABC', range(3))
```

	A	B	C
0	A0	B0	C0
1	A1	B1	C1
2	A2	B2	C2



```

1 df1 = make_df('AB', [1, 2])
2 df2 = make_df('AB', [3, 4])
3 dfc = pd.concat([df1, df2], axis=1,)
4 display(df1, df2, dfc)

```

	A	B
1	A1	B1
2	A2	B2

	A	B
3	A3	B3
4	A4	B4

	A	B	A	B
1	A1	B1	NaN	NaN
2	A2	B2	NaN	NaN
3	NaN	NaN	A3	B3
4	NaN	NaN	A4	B4

```

1 df1 = pd.DataFrame({'employee': ['Bob', 'Jake', 'Lisa', 'Sue'],
2                       'group': ['Accounting', 'Engineering', 'Engineering', 'HR']})
3 df2 = pd.DataFrame({'employee': ['Lisa', 'Bob', 'Jake', 'Sue'],
4                       'hire_date': [2004, 2008, 2012, 2014]})
5 display(df1, df2)

```

	employee	group
0	Bob	Accounting
1	Jake	Engineering
2	Lisa	Engineering
3	Sue	HR

	employee	hire_date
0	Lisa	2004
1	Bob	2008
2	Jake	2012
3	Sue	2014

```

1 df3 = pd.merge(df1, df2)
2 df3

```

	employee	group	hire_date
0	Bob	Accounting	2008
1	Jake	Engineering	2012
2	Lisa	Engineering	2004
3	Sue	HR	2014

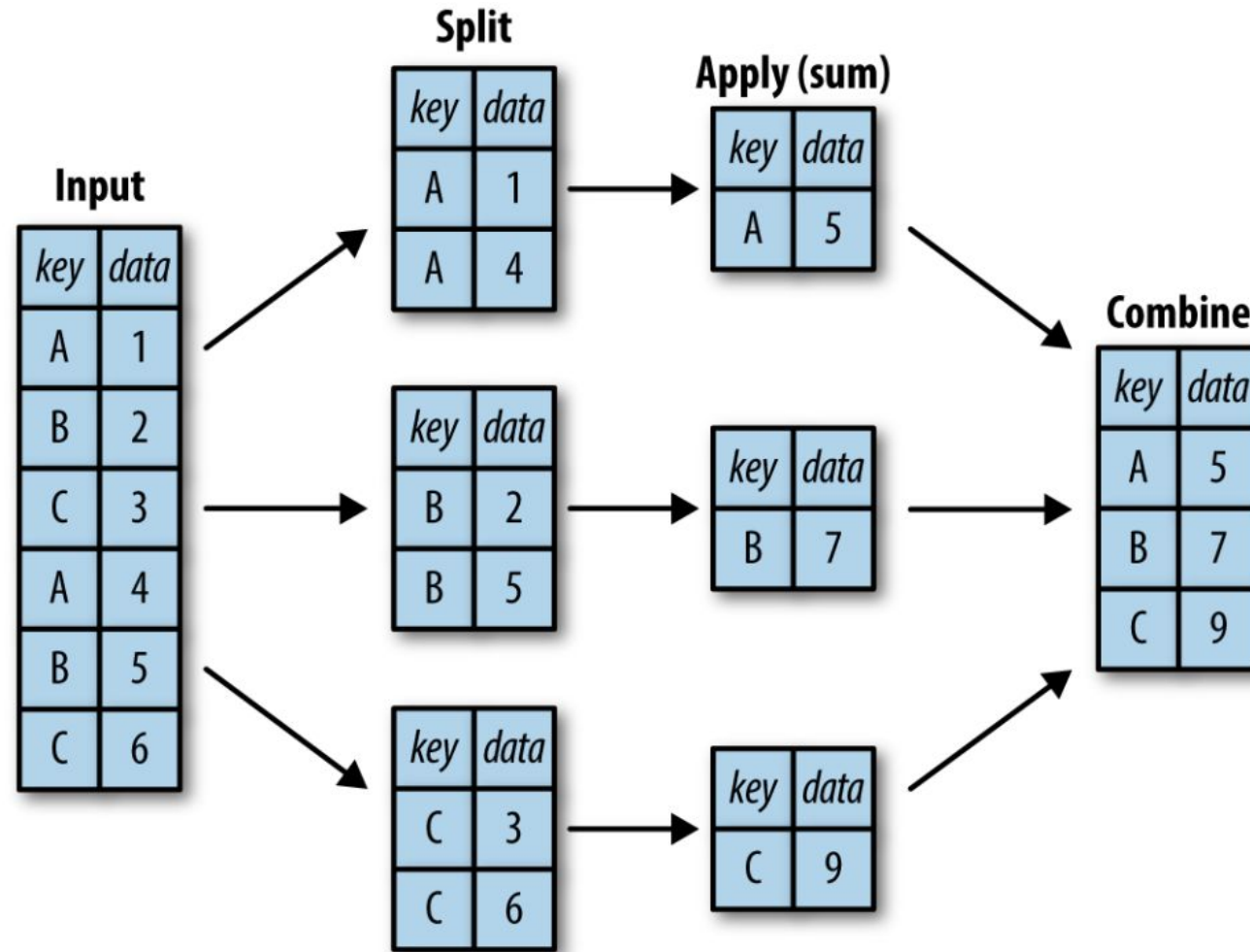
Agregación es la más común de las tareas exploratorias

- Analizar tendencias o buscar patrones se hace difícil si el análisis es individual
- Para evitar esto, datos generalmente se analizan de manera agregada
- Además de esto, la agregación suele ser a nivel grupal y no global
- Pandas permite enfrentar estos problemas con una serie de mecanismos que facilitan la exploración

Agregación es la más común de las tareas exploratorias

Aggregation	Description
<code>count()</code>	Total number of items
<code>first(), last()</code>	First and last item
<code>mean(), median()</code>	Mean and median
<code>min(), max()</code>	Minimum and maximum
<code>std(), var()</code>	Standard deviation and variance
<code>mad()</code>	Mean absolute deviation
<code>prod()</code>	Product of all items
<code>sum()</code>	Sum of all items

Función **groupby** permite combinar todo



Vamos a Jupyter Lab...



```

1 import seaborn as sns
2 planets = sns.load_dataset('planets')
3 planets.head()

```



	method	number	orbital_period	mass	distance	year
0	Radial Velocity	1	269.300	7.10	77.40	2006
1	Radial Velocity	1	874.774	2.21	56.95	2008
2	Radial Velocity	1	763.000	2.60	19.84	2011
3	Radial Velocity	1	326.030	19.40	110.62	2007
4	Radial Velocity	1	516.220	10.50	119.47	2009

```

1 planets.dropna().describe()

```



	number	orbital_period	mass	distance	year
count	498.00000	498.000000	498.000000	498.000000	498.000000
mean	1.73494	835.778671	2.509320	52.068213	2007.377510
std	1.17572	1469.128259	3.636274	46.596041	4.167284
min	1.00000	1.328300	0.003600	1.350000	1989.000000
25%	1.00000	38.272250	0.212500	24.497500	2005.000000
50%	1.00000	357.000000	1.245000	39.940000	2009.000000
75%	2.00000	999.600000	2.867500	59.332500	2011.000000
max	6.00000	17337.500000	25.000000	354.000000	2014.000000

```
1 planets.groupby('method')['orbital_period'].median()
```

```
method
Astrometry          631.180000
Eclipse Timing Variations  4343.500000
Imaging             27500.000000
Microlensing         3300.000000
Orbital Brightness Modulation    0.342887
Pulsar Timing        66.541900
Pulsation Timing Variations  1170.000000
Radial Velocity      360.200000
Transit              5.714932
Transit Timing Variations   57.011000
Name: orbital_period, dtype: float64
```

```
1 planets.groupby('method')['year'].describe()
```

```
count      mean      std      min      25%      50%      75%      max
method
Astrometry      2.0  2011.500000  2.121320  2010.0  2010.75  2011.5  2012.25  2013.0
Eclipse Timing Variations  9.0  2010.000000  1.414214  2008.0  2009.00  2010.0  2011.00  2012.0
Imaging         38.0  2009.131579  2.781901  2004.0  2008.00  2009.0  2011.00  2013.0
Microlensing    23.0  2009.782609  2.859697  2004.0  2008.00  2010.0  2012.00  2013.0
Orbital Brightness Modulation  3.0  2011.666667  1.154701  2011.0  2011.00  2011.0  2012.00  2013.0
Pulsar Timing   5.0  1998.400000  8.384510  1992.0  1992.00  1994.0  2003.00  2011.0
Pulsation Timing Variations  1.0  2007.000000      NaN  2007.0  2007.00  2007.0  2007.00  2007.0
Radial Velocity 553.0  2007.518987  4.249052  1989.0  2005.00  2009.0  2011.00  2014.0
Transit        397.0  2011.236776  2.077867  2002.0  2010.00  2012.0  2013.00  2014.0
Transit Timing Variations   4.0  2012.500000  1.290994  2011.0  2011.75  2012.5  2013.25  2014.0
```

Tablas dinámicas son otra forma de agrupar

- Nos permiten agregar valores utilizando múltiples dimensiones y funciones
- Además, pandas permite crear rangos para las variables a analizar, permitiendo un análisis más fino

Vamos a Jupyter Lab...



```
1 import numpy as np
2 import pandas as pd
3 import seaborn as sns
4 titanic = sns.load_dataset('titanic')
5 titanic.head()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

```
1 titanic.groupby('sex')[['survived']].mean()
```

```
survived
sex
female  0.742038
male    0.188908
```

```
1 titanic.groupby(['sex', 'class'])['survived'].aggregate('mean')
```

```
sex  class  survived
female First    0.968085
      Second   0.921053
      Third    0.500000
male  First    0.368852
      Second   0.157407
      Third    0.135447
Name: survived, dtype: float64
```

```
1 titanic.pivot_table('survived', index='sex', columns='class')
```

```
↳
```

	class	First	Second	Third
sex				
female		0.968085	0.921053	0.500000
male		0.368852	0.157407	0.135447

```
1 age = pd.cut(titanic['age'], [0, 18, 80])  
2 titanic.pivot_table('survived', ['sex', age], 'class')
```

```
↳
```

		class	First	Second	Third
sex	age				
female	(0, 18]		0.909091	1.000000	0.511628
	(18, 80]		0.972973	0.900000	0.423729
male	(0, 18]		0.800000	0.600000	0.215686
	(18, 80]		0.375000	0.071429	0.133663


```

1 fare = pd.qcut(titanic['fare'], 2)
2 titanic.pivot_table('survived', ['sex', age], [fare, 'class'])

```

↳

		fare (-0.001, 14.454]			(14.454, 512.329]		
		class			class		
		First	Second	Third	First	Second	Third
sex		age					
female	(0, 18]	NaN	1.000000	0.714286	0.909091	1.000000	0.318182
	(18, 80]	NaN	0.880000	0.444444	0.972973	0.914286	0.391304
male	(0, 18]	NaN	0.000000	0.260870	0.800000	0.818182	0.178571
	(18, 80]	0.0	0.098039	0.125000	0.391304	0.030303	0.192308

```

1 titanic.pivot_table(index='sex', columns='class',
2                       aggfunc={'survived':sum, 'fare':'mean'})

```

↳

		fare			survived		
		First	Second	Third	First	Second	Third
sex							
female		106.125798	21.970121	16.118810	91	70	72
male		67.226127	19.741782	12.661633	45	17	47

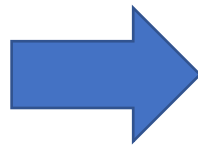
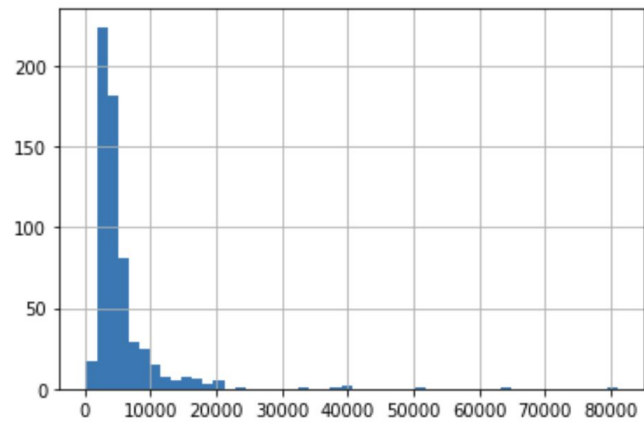
Cómo podemos presentar todo esto en Python

- Existen varias maneras en Python de presentar resultados gráficamente. Todas comparten la facilidad de uso y gran calidad de la presentación
- Con el fin de facilitar su uso, Pandas incorpora varias visualizaciones adecuadas a Series y DataFrame

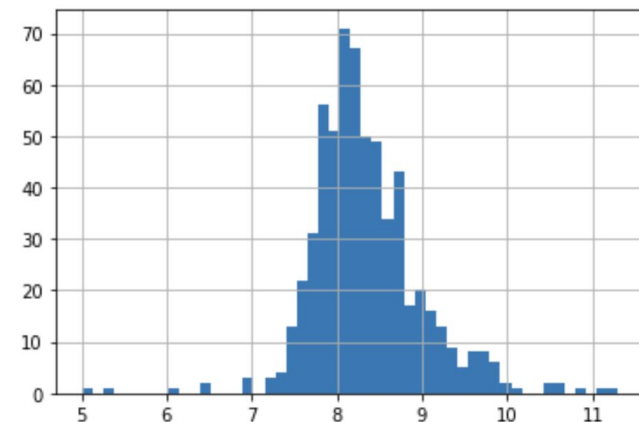
Vamos a Jupyter Lab...



```
1 df['ApplicantIncome'].hist(bins=50)
2 plt.show()
```



```
1 df['ApplicantIncome_log'] = np.log(df['ApplicantIncome'])
2 df['ApplicantIncome_log'].hist(bins=50)
3 plt.show()
```



Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación



IIC2115 - Programación como Herramienta para la Ingeniería

Análisis de datos - Exploración, limpieza y depuración

Profesora: Francesca Lucchini
Prof. Coordinador: Hans Löbel