

IA PUCP - Diplomado de Desarrollo de Aplicaciones de Inteligencia Artificial
Python para Ciencia de Datos



Lectura y procesamiento de datos tabulares: Pandas

Contenido

- Series, DataFrame e Índices
- Slicing e Slicing avanzado
- Alineamiento de Índices
- Creación de DataFrame
- Indexing
- Alineamiento de índices
- Valores faltantes
- Lectura de CSV
- Escritura de CSV
- Plots y exploración de un dataset

Motivación

En **Numpy** podríamos tener un vector que represente las edades de un grupo de personas

$$\begin{bmatrix} 15 \\ 19 \\ 20 \\ 25 \\ 16 \\ 10 \end{bmatrix}$$

Motivación

Sin embargo, puede ser útil asociar las edades a los nombres de las personas

Juan	15
María	19
Pedro	20
Daniela	25
Gustavo	16
Rocío	10

Motivación

Sin embargo, puede ser útil asociar las edades a los nombres de las personas

Juan	15
María	19
Pedro	20
Daniela	25
Gustavo	16
Rocío	10

← En **pandas** este objeto se llama una **Serie**

Motivación

Podemos pensar
también en un matriz
para representar edad
y talla

15	170
19	180
20	170
25	170
16	160
10	150

Motivación

Podemos ponerle
nombre a las
columnas

Edad	Talla
15	170
19	180
20	170
25	170
16	160
10	150

Motivación

...e igualmente a
las filas

En **pandas**
este objeto
se llama una
DataFrame



	Edad	Talla
Juan	15	170
María	19	180
Pedro	20	170
Daniela	25	170
Gustavo	16	160
Rocío	10	150

Motivación

Y para acceder eficientemente a los datos, tenemos a los **Index**

Índices		Edad	Talla
Juan	15	170	
María	19	180	
Pedro	20	170	
Daniela	25	170	
Gustavo	16	160	
Rocío	10	150	

Motivación

Y para acceder
eficientemente a
los datos,
tenemos a los
Index

	Tipos de dato diferente	
	Edad	Talla
Juan	15	170.0
María	19	180.0
Pedro	20	170.0
Daniela	25	170.0
Gustavo	16	160.0
Rocío	10	150.0

Pandas

Pandas provee estructura de datos y funciones para facilitar el trabajo con **datos estructurados**

- Mezcla de procesamiento de arreglos de alta performance (Numpy) e ideas de hojas de cálculo/base de datos relacionales
- Índices sofisticados: reshape / slice / aleatorizar
- Agregar datos (en el sentido de: sumarizar)
- Seleccionar subconjuntos de los datos

Datos Estructurados

Un término “*vago*” para referirse a varios tipos de datos comunes

- Datos Tabulares
- Arreglos n-dimensionales
- Tablas relacionales
- Series temporales

Objetos importantes en Pandas (a revisar en esta sesión)

- Series (datos en una dimensión)
- DataFrame (datos en múltiples dimensiones)
- Index (para acceder de manera optimizada a los datos)

Importando pandas

```
import pandas as pd  
from pandas import Series, DataFrame
```

Pandas: Series

- Un objeto similar a una arreglo unidimensional que contiene una secuencia de valores, y una secuencia de etiquetas llamado index (índice)

obj = pd.Series([1,2,3,4,5])

- Sino especificamos índices, serán usados los números 0..n-1

Pandas: DataFrame

- Un **DataFrame** representa un tabla rectangular de datos que contiene colecciones ordenadas de columnas (que pueden ser de un tipo diferente)
- Las columnas y las filas tienen índices

Índices

- Son inmutables
- Están en la frontera de los sets y de las listas
 - Son hashables (lo que permite un lookup rápido)
 - Permiten obtener slices
- Definen una relación de orden entre los elementos (piense en un índice)
- Puede contener elementos duplicados

Índices + Series

- Permite a la serie (un arreglo unidimensional) tener una interfaz de **lista** y de **conjunto**
- En general, no escribimos los datos ni los índices... ¡los importamos!

Accediendo a las filas con `.loc[]` e `.iloc[]`

- Podemos usar el nombre (índice) con **`.loc[indice]`**
- Si queremos usar el número de fila usaremos **`.iloc[idx]`**
- Con ambas es posible usar ***slicing*** -- que es una operación que ya vimos en Numpy
- Ambas funcionan tanto con **Series** y **DataFrames**

Slices en Series: `.loc` versus `.iloc` versus `numpy`

```
1 arreglo = np.array([15,19,20,25,16,10])
2 serie = pd.Series(arreglo, index=['Juan',
3                                   'Maria',
4                                   'Pedro',
5                                   'Daniela',
6                                   'Gustavo',
7                                   'Rocio'])
```

```
1 serie.loc['Juan':'Pedro']
```

```
Juan      15
Maria     19
Pedro     20
dtype: int64
```

De Juan hasta Pedro
(inclusive)

```
1 arreglo[0:2]
```

```
array([15, 19])
```

De Juan hasta Pedro
(¡sin incluir a Pedro!)

```
1 serie.iloc[0:2]
```

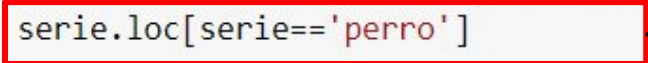
```
Juan      15
Maria     19
dtype: int64
```

Indexación avanzada

Pandas admite
indexación

avanzada similar
a numpy

```
1  mascota = ['perro', 'gato', 'gato', 'perro', 'gato', 'goldfish']
2  dueño = ['Rocio', 'Maria', 'Pedro', 'Daniela', 'Rocio', 'Gustavo']
3  serie = pd.Series(mascota, index=dueño)
4  serie.loc[serie=='perro']
```



```
Rocio      perro
Daniela    perro
dtype: object
```

```
1  serie=='perro'
```

```
Rocio      True
Maria      False
Pedro      False
Daniela    True
Rocio      False
Gustavo    False
dtype: bool
```

Usamos una
operación booleana
sobre la Serie para
realizar un query

Indexación avanzada

Pandas admite
indexación

avanzada similar
a numpy

```
1  mascota = ['perro', 'gato', 'gato', 'perro', 'gato', 'goldfish']
2  dueño = ['Rocio', 'Maria', 'Pedro', 'Daniela', 'Rocio', 'Gustavo']
3  serie = pd.Series(mascota, index=dueño)
4  serie.loc[serie=='perro']
```

```
Rocio      perro
Daniela    perro
dtype: object
```

```
1  serie=='perro'
```

```
Rocio      True
Maria     False
Pedro     False
Daniela    True
Rocio     False
Gustavo   False
dtype: bool
```

Compara los
elementos uno a
uno...

Indexación avanzada, un poco más compleja

```
1  mascota = ['perro', 'gato', 'gato', 'perro', 'gato', 'goldfish']
2  dueño = ['Rocio', 'Maria', 'Pedro', 'Daniela', 'Rocio', 'Gustavo']
3  serie = pd.Series(mascota, index=dueño)
4  serie.loc[(serie=='perro') | (serie=='goldfish')]
```

```
Rocio      perro
Daniela    perro
Gustavo    goldfish
dtype: object
```



La notación de la barra vertical | viene de los conjuntos

Alineamiento de índices

S1 =

a	b	c
1	2	3

S2 =

a	c	d
5	6	7

¿Qué ocurre si ejecuto $S1 + S2$? ¿por qué?

Alineamiento de índices

```
1 s1 = pd.Series([1,2,3], index=['a', 'b', 'c'])
2 s2 = pd.Series([5,6,7], index=['a', 'c', 'd'])
3 s1+s2
```

a 6.0

b NaN

c 9.0

d NaN

dtype: float64

NaN indique data
faltante



Alineamiento de índices

```
1 s1 = pd.Series([1,2,3], index=['a',  
2 s2 = pd.Series([5,6,7], index=['a',  
3 s1+s2
```

```
a    6.0  
b    NaN  
c    9.0  
d    NaN
```

dtype: float64

¿por qué es
float64?

1 s1

a 1
b 2
c 3

dtype: int64

1 s2

a 5
c 6
d 7

dtype: int64

Alineamiento de índices

```
1 s1 = pd.Series([1,2,3], index=['a',  
2 s2 = pd.Series([5,6,7], index=['a',  
3 s1+s2
```

```
a    6.0  
b    NaN  
c    9.0  
d    NaN
```

`dtype: float64`

En Numpy, NaN solo pertenece a los números de punto flotante

1 s1

a 1
b 2
c 3

`dtype: int64`

1 s2

a 5
c 6
d 7

`dtype: int64`

Alineamiento de índices: fill value

```
1 s1.add(s2)
```

```
a    6.0  
b    NaN  
c    9.0  
d    NaN  
dtype: float64
```

```
1 s1.add(s2, fill_value=50)
```

```
a    6.0  
b   52.0  
c    9.0  
d   57.0  
dtype: float64
```

Series → DataFrames

- Las **Series** de pandas nos permiten lidiar con datos unidimensionales
- Los **DataFrames** de pandas nos permiten lidiar con tablas bidimensionales (matrices!)
 - Con indexación de **columnas** e
 - **indexación** de filas
- Las funcionalidades de Series que hemos visto
 - `.loc[]`
 - `.iloc[]`
 - slicing
 - alineamiento de índices

también aplican para **DataFrames**

...recordando

Y para acceder
eficientemente a
los datos,
tenemos a los
Index

	Tipos de dato diferente	
	Edad	Talla
Juan	15	170.0
María	19	180.0
Pedro	20	170.0
Daniela	25	170.0
Gustavo	16	160.0
Rocío	10	150.0

Pandas: DataFrame

- Un **DataFrame** representa un tabla rectangular de datos que contiene colecciones ordenadas de columnas (que pueden ser de un tipo diferente)
- Las columnas y las filas tienen índices

Creando un DataFrame en base a varias Series

```
1 nombres = ['Juan', 'Maria', 'Pedro', 'Daniela']
2 edad = pd.Series([25, 30, 35, 40], index=nombres)
3 estatura = pd.Series([160, 165, 170, 175], index=nombres)
4 sexo = pd.Series(['m', 'f', 'm', 'f'], index=nombres)
5 df = pd.DataFrame({ 'edad' : edad,
6                      'estatura': estatura,
7                      'sexo': sexo
8                      })
9 df
```

	edad	estatura	sexo
Juan	25	160	m
Maria	30	165	f
Pedro	35	170	m
Daniela	40	175	f

Creando un DataFrame en base a varias Series

```
1 nombres = ['Juan', 'Maria', 'Pedro', 'Daniela']
2 edad = pd.Series([25, 30, 35, 40], index=nombres)
3 estatura = pd.Series([160, 165, 170, 175], index=nombres)
4 sexo = pd.Series(['m', 'f', 'm', 'f'], index=nombres)
5 df = pd.DataFrame({'edad': edad,
6                    'estatura': estatura,
7                    'sexo': sexo
8                    })
9 df
```

	edad	estatura	sexo
Juan	25	160	m
Maria	30	165	f
Pedro	35	170	m
Daniela	40	175	f

Creamos un DataFrame a partir de un diccionario de Series

Creando un DataFrame en base a varias Series

```
1 nombres = ['Juan', 'Maria', 'Pedro', 'Daniela']
2 edad = pd.Series([25, 30, 35, 40], index=nombres)
3 estatura = pd.Series([160, 165, 170, 175], index=nombres)
4 sexo = pd.Series(['m', 'f', 'm', 'f'], index=nombres)
5 df = pd.DataFrame({ 'edad' : edad,
6                     'estatura': estatura,
7                     'sexo': sexo
8                     })
9 df
```

	edad	estatura	sexo
Juan	25	160	m
Maria	30	165	f
Pedro	35	170	m
Daniela	40	175	f

Los índices de fila son los nombres, y son compartidos por todas las Series

Creando un DataFrame en base a varias Series

```
1 nombres = ['Juan', 'Maria', 'Pedro', 'Daniela']
2 edad = pd.Series([25, 30, 35, 40], index=nombres)
3 estatura = pd.Series([160, 165, 170, 175], index=nombres)
4 sexo = pd.Series(['m', 'f', 'm', 'f'], index=nombres)
5 df = pd.DataFrame({'edad': edad,
6                    'estatura': estatura,
7                    'sexo': sexo
8                    })
9 df
```

	edad	estatura	sexo
Juan	25	160	m
Maria	30	165	f
Pedro	35	170	m
Daniela	40	175	f

Los índices de columna son las llaves del diccionario

Otras formas de construir un DataFrame

- `ndarray` de numpy de 2 dimensiones
- diccionario de arrays, listas o tuplas
- Array estructurado de Numpy
- Diccionario de Series
- Diccionario de diccionarios
- Lista de diccionarios o Series
- Lista de listas o tuplas
- Otro DataFrame
- Un arreglo “enmascarado” `MaskedArray` de Numpy

Atributos del DataFrame

```
1 df.index
```



Índices de fila

```
Index(['Juan', 'Maria', 'Pedro', 'Daniela'], dtype='object')
```

Atributos del DataFrame

```
1 df.index
```



Índices de fila

```
Index(['Juan', 'Maria', 'Pedro', 'Daniela'], dtype='object')
```

```
1 df.columns
```



Índices de columna

```
Index(['edad', 'estatura', 'sexo'], dtype='object')
```

Atributos del DataFrame

```
1 df.index
```

→ Índices de fila

```
Index(['Juan', 'Maria', 'Pedro', 'Daniela'], dtype='object')
```

```
1 df.columns
```

→ Índices de columna

```
Index(['edad', 'estatura', 'sexo'], dtype='object')
```

```
1 df.values
```

→ ndarray de Numpy

```
array([[25, 160, 'm'],  
       [30, 165, 'f'],  
       [35, 170, 'm'],  
       [40, 175, 'f']], dtype=object)
```

Head y Tail de un DataFrame

- Permiten obtener los **n** primeras o **n** últimas filas de un **DataFrame**

```
1 df.head(2)
```

→ 2 primeras filas

	edad	estatura	sexo
Juan	25	160	m
Maria	30	165	f

Head y Tail de un DataFrame

- Permiten obtener los **n** primeras o **n** últimas filas de un **DataFrame**

```
1 df.head(2)
```

	edad	estatura	sexo
Juan	25	160	m
Maria	30	165	f

→ 2 primeras filas

```
1 df.tail(2)
```

	edad	estatura	sexo
Pedro	35	170	m
Daniela	40	175	f

→ 2 últimas filas

Propiedades estadísticas (rápidas) de un DataFrame

- Usaremos el método `.describe()`

```
1 df.describe()
```

	edad	estatura
count	4.000000	4.000000
mean	32.500000	167.500000
std	6.454972	6.454972
min	25.000000	160.000000
25%	28.750000	163.750000
50%	32.500000	167.500000
75%	36.250000	171.250000
max	40.000000	175.000000

Propiedades estadísticas (rápidas) de un DataFrame

- Usaremos el método `.describe()`

```
1 df.describe()
```

	edad	estatura
count	4.000000	4.000000
mean	32.500000	167.500000
std	6.454972	6.454972
min	25.000000	160.000000
25%	28.750000	163.750000
50%	32.500000	167.500000
75%	36.250000	171.250000
max	40.000000	175.000000



tendencia central y
dispersión

Propiedades estadísticas (rápidas) de un DataFrame

- Usaremos el método `.describe()`

```
1 df.describe()
```

	edad	estatura
count	4.000000	4.000000
mean	32.500000	167.500000
std	6.454972	6.454972
min	25.000000	160.000000
25%	28.750000	163.750000
50%	32.500000	167.500000
75%	36.250000	171.250000
max	40.000000	175.000000

→ tendencia central y dispersión

→ rangos

Accediendo a los datos con .loc[] e .iloc[]

- Sigue las mismas reglas que la Serie

	edad	estatura	sexo
Juan	25	160	m
Maria	30	165	f
Pedro	35	170	m
Daniela	40	175	f

```
1 df.loc['Maria']
```

```
edad      30
estatura  165
sexo       f
Name: Maria, dtype: object
```

```
1 df.iloc[0]
```

```
edad      25
estatura  160
sexo       m
Name: Juan, dtype: object
```

Accediendo a los datos con .loc[] e .iloc[]

- Sigue las mismas reglas que la Serie

	edad	estatura	sexo
Juan	25	160	m
Maria	30	165	f
Pedro	35	170	m
Daniela	40	175	f

```
1 df.loc['Maria']
```

edad	30
estatura	165
sexo	f

Name: Maria, dtype: object

```
1 df.iloc[0]
```

edad	25
estatura	160
sexo	m

Name: Juan, dtype: object

Accediendo a los datos con .loc[] e .iloc[]

- Sigue las mismas reglas que la Serie

	edad	estatura	sexo
Juan	25	160	m
Maria	30	165	f
Pedro	35	170	m
Daniela	40	175	f

```
1 df.loc['Maria']
```

```
edad      30
estatura  165
sexo       f
Name: Maria, dtype: object
```

```
1 df.iloc[0]
```

```
edad      25
estatura  160
sexo       m
Name: Juan, dtype: object
```

Accediendo a los datos con `.loc[]` e `.iloc[]`

- Pero se puede acceder usando los dos índices (filas y columnas)

	edad	estatura	sexo
Juan	25	160	m
Maria	30	165	f
Pedro	35	170	m
Daniela	40	175	f

```
1 df.loc['Maria', 'edad']
```


Accediendo a los datos con `.loc[]` e `.iloc[]`

- Pero se puede acceder usando los dos índices (filas y columnas)

	edad	estatura	sexo
Juan	25	160	m
Maria	30	165	f
Pedro	35	170	m
Daniela	40	175	f

```
1 df.loc['Maria', 'edad']
```

30

Seleccionando “todas” las filas

- Usando los dos puntos (:) se puede seleccionar “toda” una dimensión

```
1 df.loc[:, 'estatura']
```

```
Juan      160  
Maria     165  
Pedro     170  
Daniela   175  
Name: estatura, dtype: int64
```

Indexing Avanzado

- Permite usar condiciones lógicas para filtrar un DataFrame

```
1 df.loc[df.loc[:, 'edad']<32]
```

	edad	estatura	sexo
Juan	25	160	m
Maria	30	165	f

Notas sobre el indexing

- La columna retorna de un dataframe a través del indexing es una vista de los datos, no una copia. Por lo tanto, cualquier modificación in-place de la Serie será reflejada en el DataFrame
- Se puede copiar explícitamente con el método **.copy()** de la clase **Series**

Otras opciones de indexing (I)

Type	Notes
<code>df[val]</code>	Select single column or sequence of columns from the DataFrame; special case conveniences: boolean array (filter rows), slice (slice rows), or boolean DataFrame (set values based on some criterion)
<code>df.loc[val]</code>	Selects single row or subset of rows from the DataFrame by label
<code>df.loc[:, val]</code>	Selects single column or subset of columns by label
<code>df.loc[val1, val2]</code>	Select both rows and columns by label
<code>df.iloc[where]</code>	Selects single row or subset of rows from the DataFrame by integer position

Otras opciones de indexing (II)

Type	Notes
<code>df.iloc[:, where]</code>	Selects single column or subset of columns by integer position
<code>df.iloc[where_i, where_j]</code>	Select both rows and columns by integer position
<code>df.at[label_i, label_j]</code>	Select a single scalar value by row and column label
<code>df.iat[i, j]</code>	Select a single scalar value by row and column position (integers)
reindex method	Select either rows or columns by labels
get_value, set_value methods	Select single value by row and column label

Alineamiento de Índices

- Usando los dos puntos (:) se puede seleccionar “toda” una dimensión

Reset Index

- Podemos resetear lo índices (y convertirlos en columna!) usando el método

`.reset_index()`

```
1 df
```

	edad	estatura	sexo
Juan	25	160	m
Maria	30	165	f
Pedro	35	170	m
Daniela	40	175	f

```
1 df = df.reset_index()  
2 df
```



¿por qué
asignación?

	index	edad	estatura	sexo
0	Juan	25	160	m
1	Maria	30	165	f
2	Pedro	35	170	m
3	Daniela	40	175	f

Renombrando una columna

- Podemos usar el método `rename` para renombrar las columnas usando un diccionario donde
- Las llaves son los nombres antiguos
- Los valores son los nuevos nombres

	index	edad	estatura	sexo
0	Juan	25	160	m
1	Maria	30	165	f
2	Pedro	35	170	m
3	Daniela	40	175	f

```
1 df = df.rename(columns={'index': 'nombre'})
```

```
1 df
```

	nombre	edad	estatura	sexo
0	Juan	25	160	m
1	Maria	30	165	f
2	Pedro	35	170	m
3	Daniela	40	175	f

Cambiar el índice

Podemos usar el método `set_index` para especificar un nuevo índice (una columna ya existente)

	nombre	edad	estatura	sexo
0	Juan	25	160	m
1	Maria	30	165	f
2	Pedro	35	170	m
3	Daniela	40	175	f

```
1 df.set_index('edad')
```

	nombre	estatura	sexo
edad			
25	Juan	160	m
30	Maria	165	f
35	Pedro	170	m
40	Daniela	175	f

Formas de acceder a las Columnas (series)

- **df.nombre_columna** (en caso sea un nombre amigable con Python)
 - Recordar las reglas de los identificadores
- **df['nombre_columna']**

Cada una de las **Series** “seleccionada” de esta forma tendrá el mismo índice que el **DataFrame**

Agregando Columnas

- Asignar una columna que no existe creará una nueva columna
 - Nota: No se puede crear usando la sintaxis de `df.nombre_columna`, solo usando `['nombre_columna']`
- El keyword **del**, eliminará las columnas como funciona con un diccionario

Alineamiento de Índices

- Al igual que con las Series, los índices se alinean
- Dados un valor faltante, la suma es también un valor faltante

```
1 df1 = pd.DataFrame(np.ones((2,2)), index=['a','b'], columns=['x','y'])
2 df1
```

	x	y
a	1.0	1.0
b	1.0	1.0

```
1 df2 = pd.DataFrame(np.ones((2,2)), index=['a','c'], columns=['x','z'])
2 df2
```

	x	z
a	1.0	1.0
c	1.0	1.0

```
1 df1 + df2
```

	x	y	z
a	2.0	NaN	NaN
b	NaN	NaN	NaN
c	NaN	NaN	NaN

Fill Value

¿Por qué hay valores NaN en esta versión de la tabla?

```
1 df1.add(df2, fill_value=0)
```

	x	y	z
a	2.0	1.0	1.0
b	1.0	1.0	NaN
c	1.0	NaN	1.0

fillna(*value*)

Permite llenar o “completar” valores faltantes

```
1 df1.add(df2, fill_value=0)
```

	x	y	z
a	2.0	1.0	1.0
b	1.0	1.0	NaN
c	1.0	NaN	1.0

```
1 df3 = df1.add(df2, fill_value=0)  
2 df3.fillna(-1)
```

	x	y	z
a	2.0	1.0	1.0
b	1.0	1.0	-1.0
c	1.0	-1.0	1.0

dropna()

Permite eliminar filas que tienen datos faltantes

```
1 df1.add(df2, fill_value=0)
```

	x	y	z
a	2.0	1.0	1.0
b	1.0	1.0	NaN
c	1.0	NaN	1.0

```
1 df3.dropna()
```

	x	y	z
a	2.0	1.0	1.0

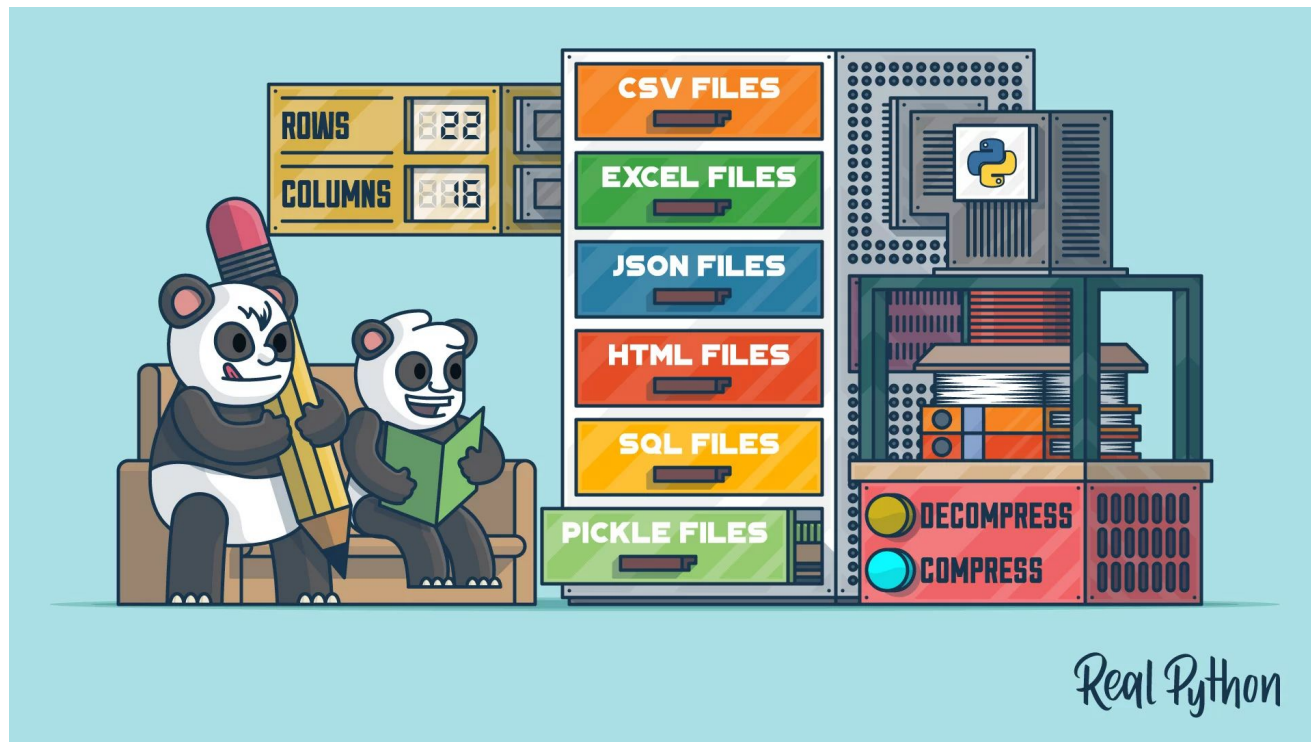
Métodos de los índices

<code>append</code>	Concatenar con otros objetos Index, produciendo un nuevo índice
<code>difference</code>	Calcula la diferencia de conjuntos como Index
<code>intersection</code>	Calcula la intersección
<code>union</code>	Calcula la unión
<code>isin</code>	Calcula un arreglo booleano indicando si cada valor está contenido en la colección pasada.

Métodos de los índices

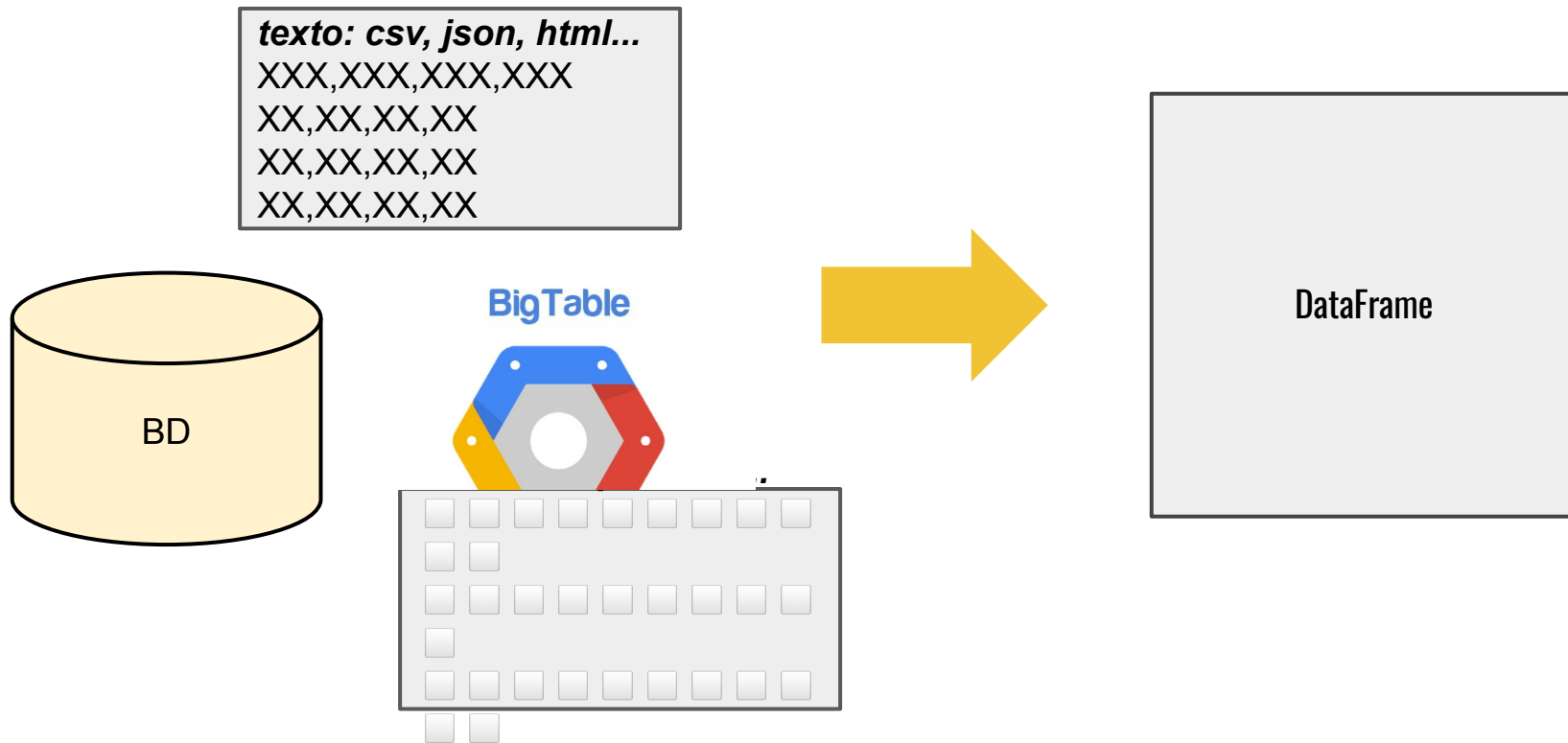
<code>delete</code>	Genera un nuevo índice con el elemento en la posición i eliminado
<code>drop</code>	Genera un nuevo índice al eliminar los valores pasados
<code>insert</code>	Genera un nuevo índice al insertar el elemento en el índice i
<code>is_monotonic</code>	Retorna True si cada elemento es mayor o igual que los elementos previos
<code>is_unique</code>	Retorna True si el índice no tiene valores duplicados
<code>unique</code>	Calcula el arreglo de valores únicos en el Index

Lectura y Escritura de Archivos en Pandas



<https://realpython.com/pandas-read-write-files/>

Lectura de Datos



Funciones I/O: Texto

Format Type	Data Description	Reader	Writer
text	CSV	<code>read_csv</code>	<code>to_csv</code>
text	Fixed-Width Text File	<code>read_fwf</code>	
text	JSON	<code>read_json</code>	<code>to_json</code>
text	HTML	<code>read_html</code>	<code>to_html</code>
text	Local clipboard	<code>read_clipboard</code>	<code>to_clipboard</code>
	MS Excel	<code>read_excel</code>	<code>to_excel</code>

https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html

Funciones I/O: binarios

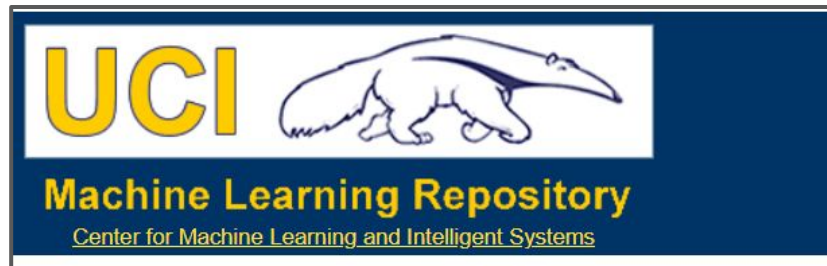
Format Type	Data Description	Reader	Writer
binary	OpenDocument	read_excel	
binary	HDF5 Format	read_hdf	to_hdf
binary	Feather Format	read_feather	to_feather
binary	Parquet Format	read_parquet	to_parquet
binary	ORC Format	read_orc	
binary	Msgpack	read_msgpack	to_msgpack
binary	Stata	read_stata	to_stata
binary	SAS	read_sas	
binary	SPSS	read_spss	
binary	Python Pickle Format	read_pickle	to_pickle

https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html

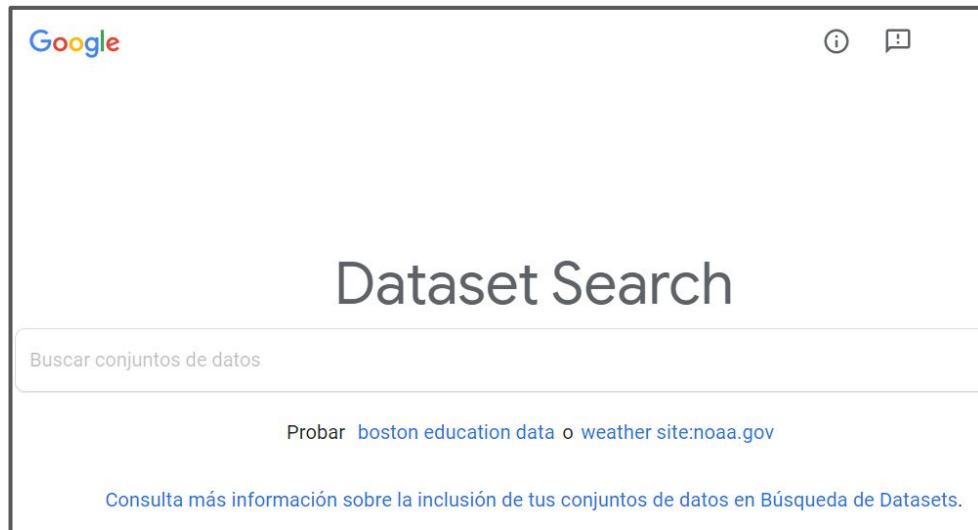
Funciones I/O: SQL

Format Type	Data Description	Reader	Writer
SQL	SQL	<code>read_sql</code>	<code>to_sql</code>
SQL	Google BigQuery	<code>read_gbq</code>	<code>to_gbq</code>

Datasets online



<https://archive.ics.uci.edu/ml/datasets.php>



<https://datasetsearch.research.google.com/>

Abalone



Abalone Data Set

Download: [Data Folder](#), [Data Set Description](#)

Abstract: Predict the age of abalone from physical measurements



Data Set Characteristics:	Multivariate	Number of Instances:	4177	Area:	Life
Attribute Characteristics:	Categorical, Integer, Real	Number of Attributes:	8	Date Donated	1995-12-01
Associated Tasks:	Classification	Missing Values?	No	Number of Web Hits:	899930

Source:

Data comes from an original (non-machine-learning) study:

Warwick J Nash, Tracy L Sellers, Simon R Talbot, Andrew J Cawthorn and Wes B Ford (1994)

"The Population Biology of Abalone (_Haliotis_ species) in Tasmania. I. Blacklip Abalone (_H. rubra_) from the North Coast and Islands of Bass Strait", Sea Fisheries Division, Technical Report No. 48 (ISSN 1034-3288)



<https://spanish.alibaba.com/product-detail/polished-natural-abalone-shell-flashy-large-abalone-shell-s-wholesale-price-60771662084.html>

Abalone

- En el notebook Abalone corrija los errores

```
1 import pandas as pd
```

```
1 abalone_link = 'https://archive.ics.uci.edu/ml/machine-learning-databases/abalone/abalone.data'  
2 df = pd.read_csv(abalone_link)  
3 df
```

	M	0.455	0.365	0.095	0.514	0.2245	0.101	0.15	15
0	M	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.0700	7
1	F	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.2100	9
2	M	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.1550	10
3	I	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.0550	7
4	I	0.425	0.300	0.095	0.3515	0.1410	0.0775	0.1200	8
...
4171	F	0.565	0.450	0.165	0.8870	0.3700	0.2390	0.2490	11
4172	M	0.590	0.440	0.135	0.9660	0.4390	0.2145	0.2605	10
4173	M	0.600	0.475	0.205	1.1760	0.5255	0.2875	0.3080	9
4174	F	0.625	0.485	0.150	1.0945	0.5310	0.2610	0.2960	10
4175	M	0.710	0.555	0.195	1.9485	0.9455	0.3765	0.4950	12

4176 rows × 9 columns

Nombres de columnas equivocadas.
Hint: header

Abalone

- Verifique las estadísticas en <https://archive.ics.uci.edu/ml/machine-learning-databases/abalone/abalone.data> se cumplen:

Statistics for numeric domains:

	Length	Diam	Height	Whole	Shucked	Viscera	Shell	Rings
Min	0.075	0.055	0.000	0.002	0.001	0.001	0.002	1
Max	0.815	0.650	1.130	2.826	1.488	0.760	1.005	29
Mean	0.524	0.408	0.140	0.829	0.359	0.181	0.239	9.934
SD	0.120	0.099	0.042	0.490	0.222	0.110	0.139	3.224
Correl	0.557	0.575	0.557	0.540	0.421	0.504	0.628	1.0

La variable a
Predecir

Abalone

- Transformaremos las variables categóricas en 1-hot-encoding. Ver `pandas.get_dummies()`

M



1	0	0
---	---	---

F



0	1	0
---	---	---

I



0	0	1
---	---	---

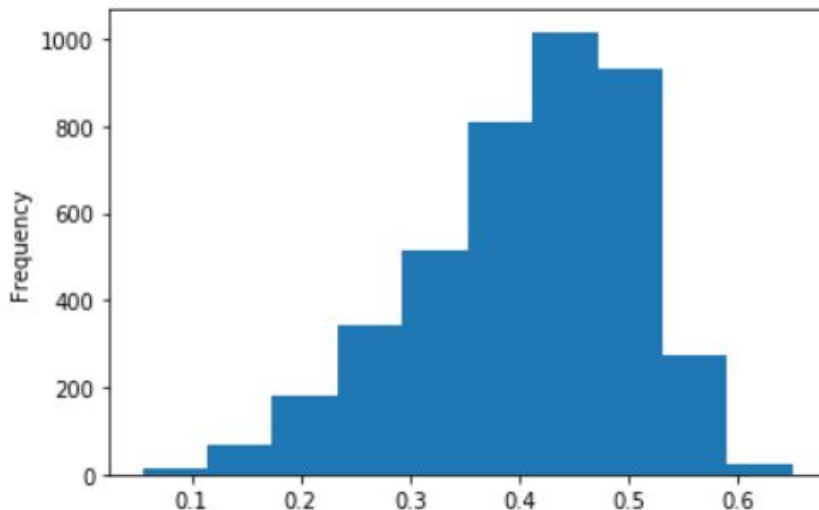
https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.get_dummies.html

Histograma

Esto es una serie,
obtenida desde el
DataFrame

```
1 df['Diam'].plot.hist()
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f7c6c719898>



**Para más resolución
aumente la cantidad de
bins**

Otros gráficos

- 'line' : line plot (default)
- 'bar' : vertical bar plot
- 'barh' : horizontal bar plot
- 'hist' : histogram
- 'box' : boxplot
- 'kde' : Kernel Density Estimation plot
- 'density' : same as 'kde'
- 'area' : area plot
- 'pie' : pie plot

Grabando un csv

```
DataFrame.to_csv('new-data.csv')
```

Algunos parámetros interesantes:

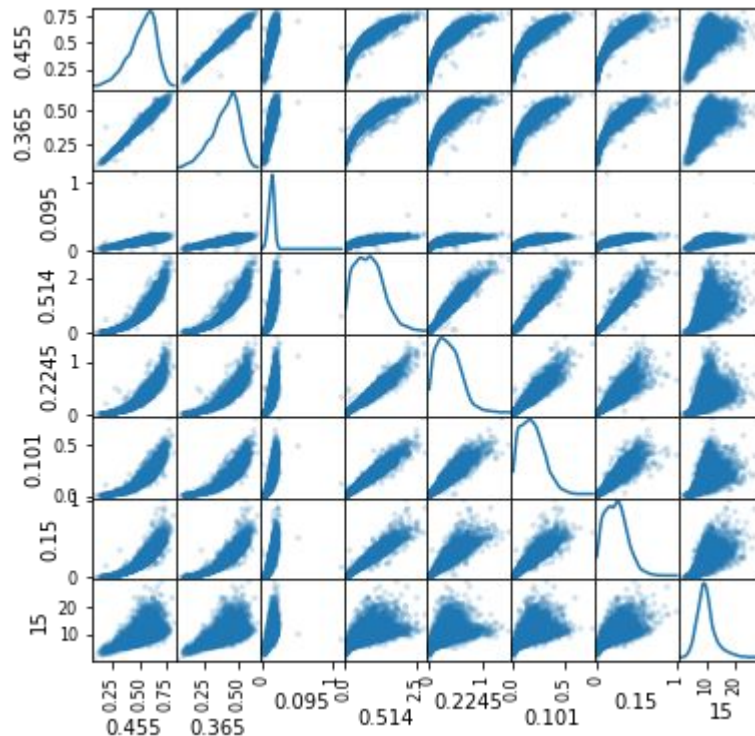
- **sep** : str, default ','
- **na_rep** : str, default "Missing data representation."
- **header** : bool or list of str, default True

Ejercicio

Usando el DataFrame de Abalone grabe un archivo .csv donde la columna sexo haya sido reemplazada por los indicadores

Relación entre features

```
from pandas.plotting import scatter_matrix  
  
scatter_matrix(df, alpha=0.2, figsize=(6, 6),  
diagonal='kde')
```

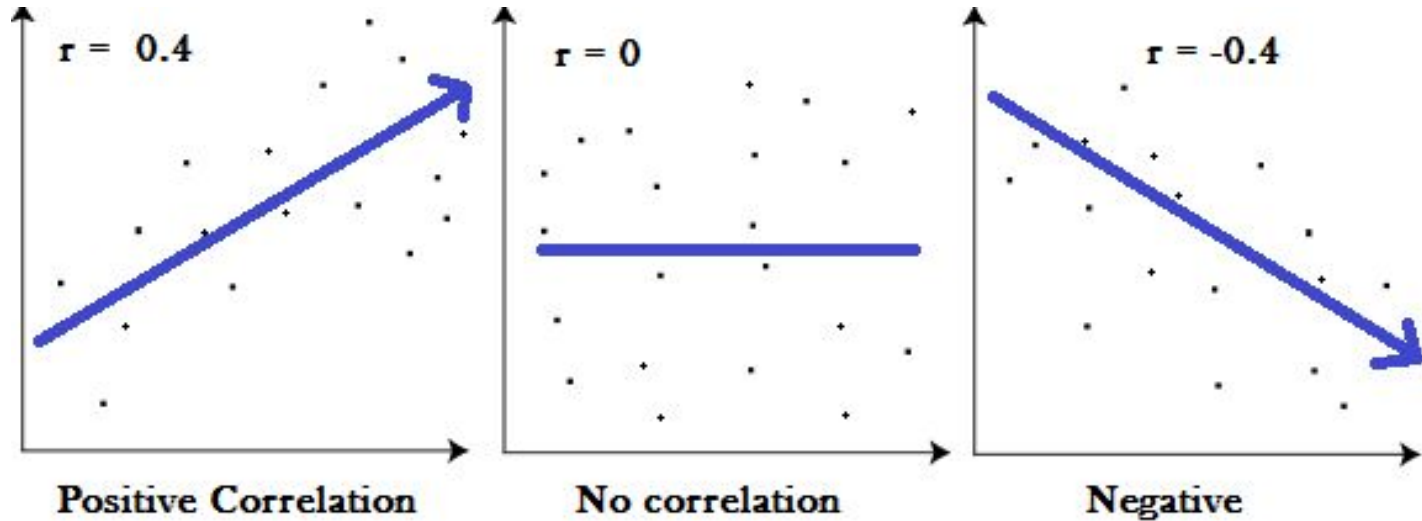


Coeficiente de Correlación de Pearson

Es una medida de correlación lineal entre dos variables

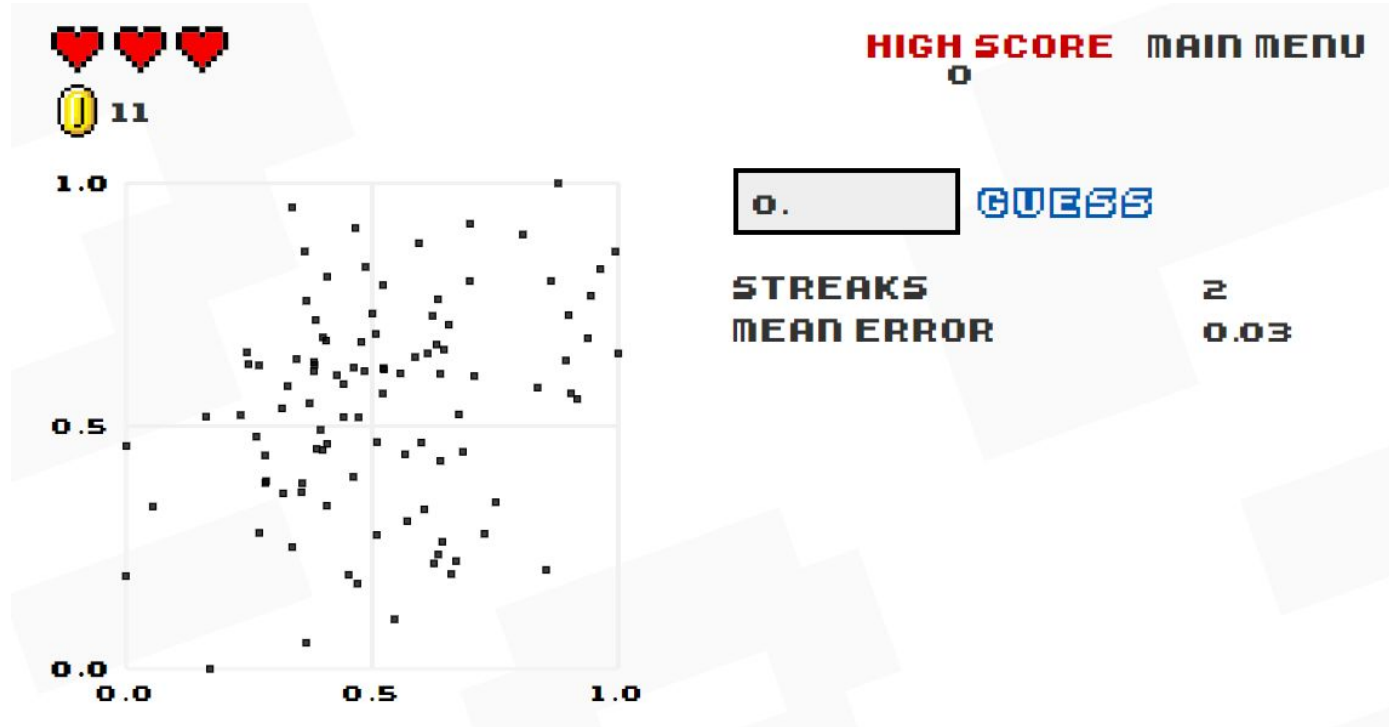
$$r_{xy} = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}}$$

Correlación positiva y negativa



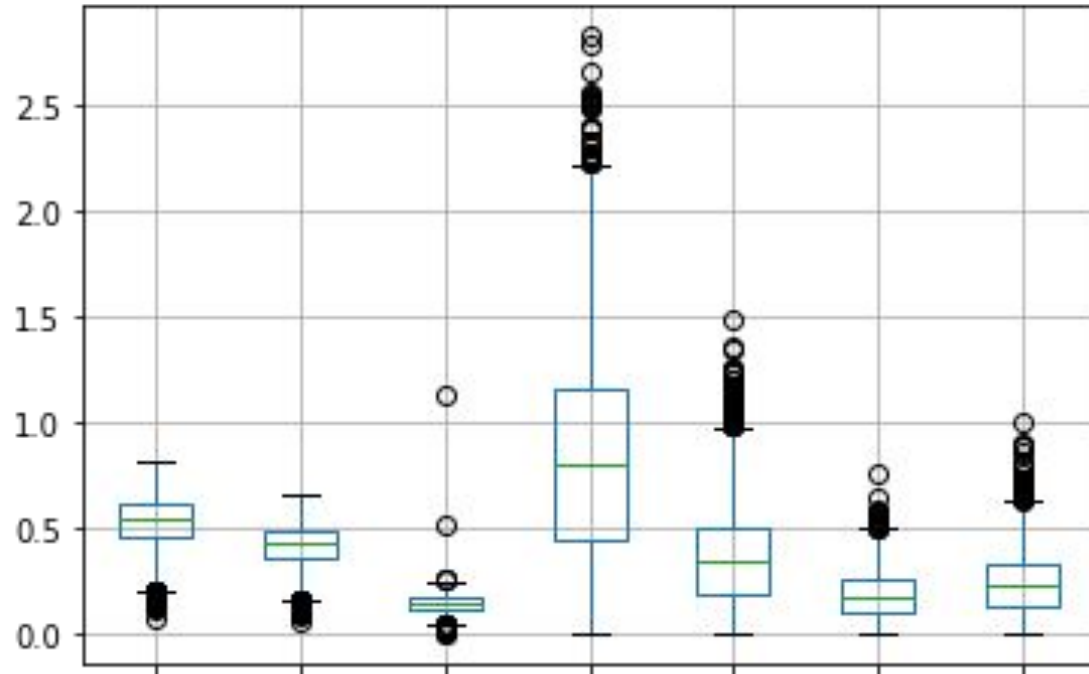
<https://www.statisticshowto.datasciencecentral.com/probability-and-statistics/correlation-coefficient-formula/#Pearson>

Juguemos <http://guessthecorrelation.com/>



Ejercicio

df.boxplot() nos ayudará a tener una idea del rango de las variables



Juguemos <http://guessthecorrelation.com/>



11

HIGH SCORE MAIN MENU

0

NEXT

TRUE R	0.29
GUESSED R	0.50
DIFFERENCE	0.21

STREAKS	0
MEAN ERROR	0.08



-1

