# Casos de uso de Python

Daniel Molina Cabrera

Curso de Python (Abril 2018)

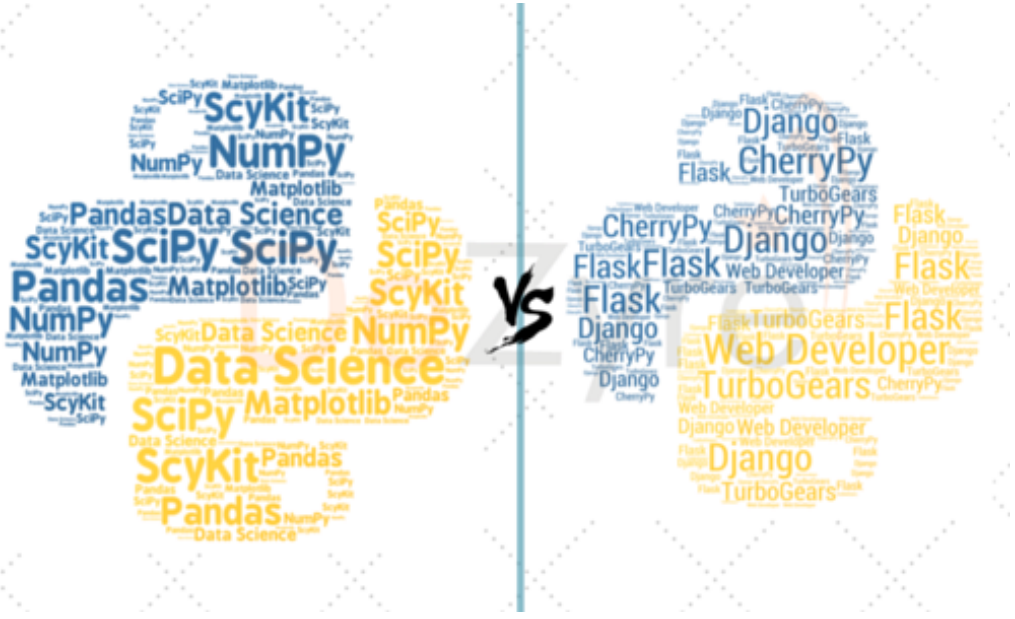
# Casos de uso de Python



## Qué veremos

- Áreas donde Python aporta.
- Librerías recomendadas.

# Áreas que veremos




## Ciencia

- Cálculo: Numpy, Pandas.
- Visualización.

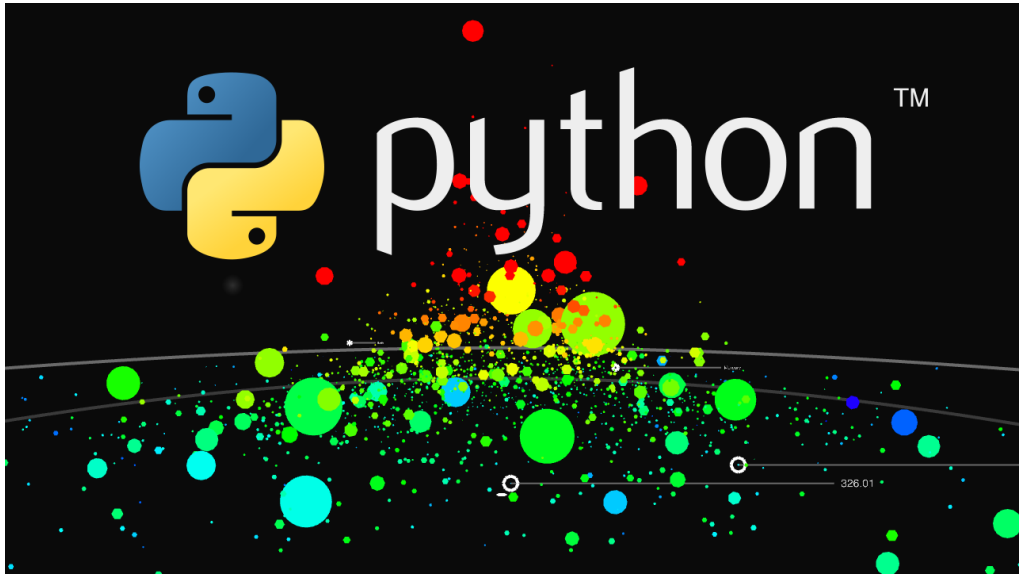
## Programación web

- Backend: Django, Flask.

# Índice

1. Python para la ciencia
2. Python y la programación web
3. Comunicaciones con Python

# Python para la ciencia




## Python para la ciencia

- Python se usa mucho para la ciencia de datos[a].
- Gran parte de su comunidad son científicos, no informáticos.

[a] http://awahid.net/blog/data-science-with-python-or-java/

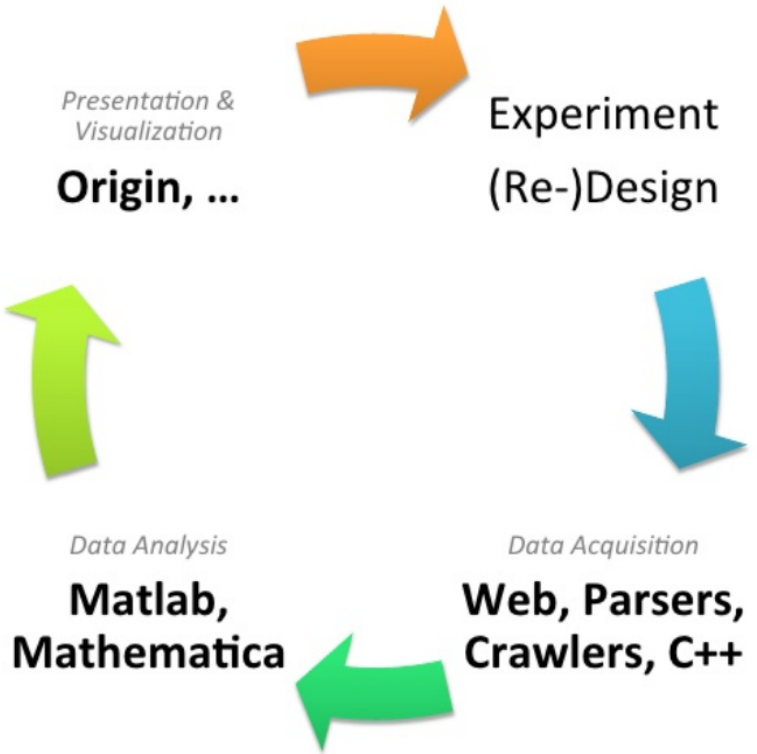
# ¿Qué aporta Python en la ciencia?

## Ventajas

- Lenguaje fácil de usar para no informáticos.
- Comunidad amigable.
- Librerías científicas avanzadas fáciles de usar.
- Entorno desarrollo estable.
- Paralelismo.

# Utilidad de Python

Utilidad en proceso científico [1]



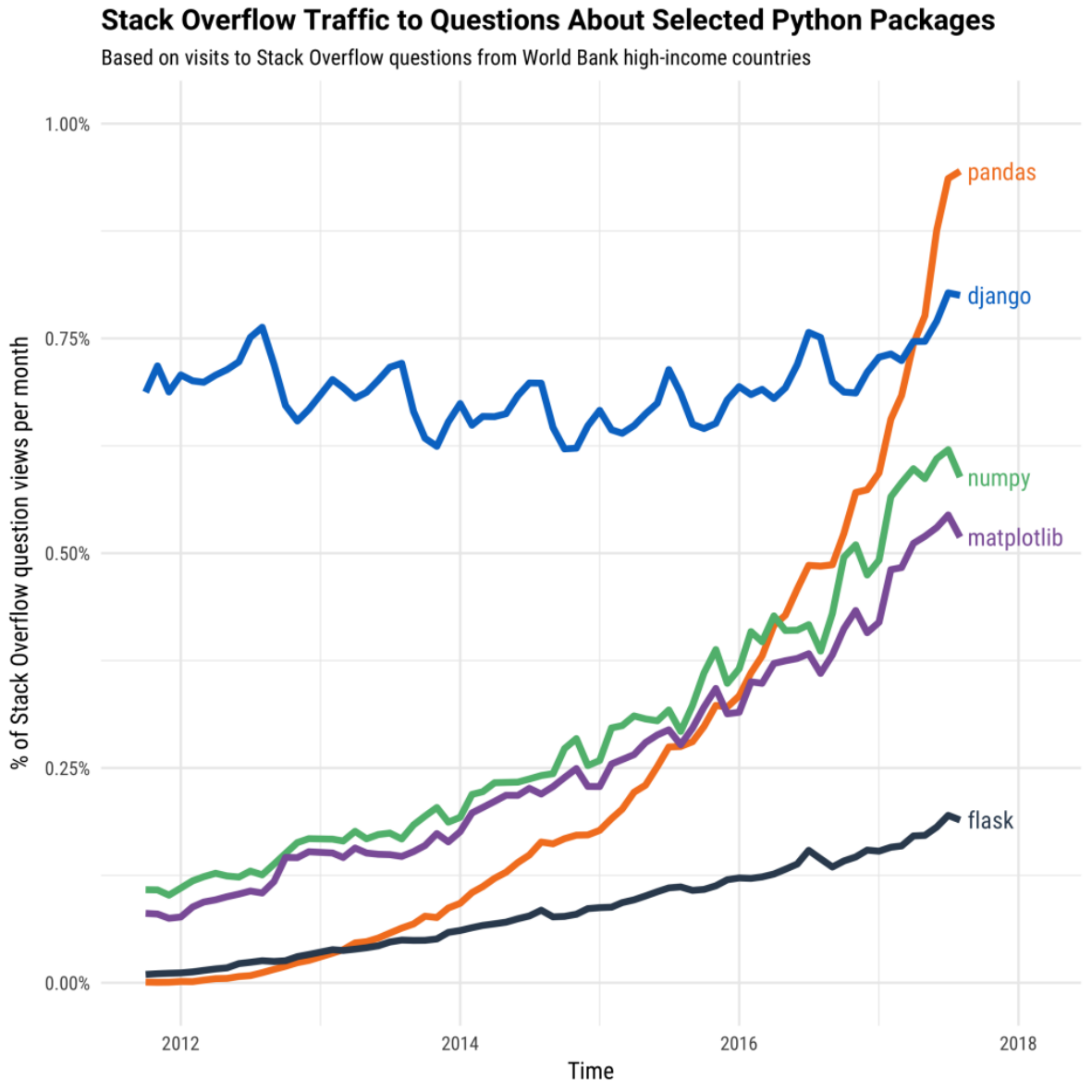

[1] https://www.slideshare.net/marcelcaraciolo/computao-cientfica-com-python-numpy-e-scipy

# Utilidad de Python

Utilidad en proceso científico [1]


[1] https://www.slideshare.net/marcelcaraciolo/computao-cientfica-com-python-numpy-e-scipy

# Uso de Python



# Rendimiento de Python

## ¿Pero Python no era lento?

Python por defecto sí[a]

Table: Tiempo de un benchmark (s)

| Versión | Tiempo (ms) |
|---|---|
| Python puro | 183 |
| Numpy | 5.97 |
| Cython normal | 7.76 |
| Cython optimizado | 2.18 |
| Cython llamando a C | 2.22 |

[a] https://arogozhnikov.github.io/2015/01/06/benchmarks-of-speed-numpy-vs-all.html

# Rendimiento de Python

## ¿Pero Python no era lento?[1]

Table: tiempo matriz de distancias

| Versión | Tiempo |
|---|---|
| Python | 9.51 seg |
| Naive numpy | 64.7 ms |
| Numba | 6.72 ms |
| Cython | 6.57 ms |
| Parakeet | 12.3 ms |
| Cython | 6.57 ms |

# Rendimiento en Python

- Procesar 1 GB de datos de Datos (1936 variables y 145232 filas).
- Comparando Python vs Scala (Java)[2].

## Usando Spark

| Nodos | Versión | Tiempo |
|---|---|---|
| 3 | Scala | 250 s |
| 3 | Python | 246 s |

[2] https://stackoverflow.com/questions/32464122/spark-performance-for-scala-vs-python

# Rendimiento en Python

## Hay distintas alternativas

**Librería numpy** Equivalente a Matlab, optimizable con BLAS.

**PyPy** Intérprete JIT, en migración a Python3.

**Cython** Python compilado a C (Python + tipos).

**Numba** Compilación JIT.

## Y en paralelo

**Paralelismo fácil** Clusters, snakemake, Luigi.

**Librerías paralelas** Dask.

**Big Data** pyspark.

**Librerías en GPU** PyCUDA, PyTorch.

# Caso de ejemplo: Numpy + Pandas


## Numpy

- Librería matricial potente.
- Pandas, leer tablas de datos.

# Tutorial de Pandas

# Visualizando [3]

## Múltiples librerías

| | |
|---|---|
| Matplotlib | Librería por defecto, basada en Matlab. |
| Seaborn | Sobre matplotlib, estilos. |
| Pandas | Directamente. |
| Bokeh | Gráficas webs. |
| Holoviews | Sobre Bokeh, mayor nivel abstracción. |
| Altair | Enfoque declarativo, web, en desarrollo. |

[3] https://bit.ly/2sUHcJu

# Visualizando

## ¿Qué aporta?

- Excel ya me permite hacer gráficas.
- Excel ya gestiona tablas.

## Qué ofrece Python

- Obtener datos de fuentes distintas
- Análisis de datos visualmente
- Diagramas interactivos

# Qué ofrece Python

## Obtener datos de fuentes distintas

- Redes sociales.
- Páginas webs (*web scraping*).
- Otros recursos (Bases de Datos, . . . ).

## Análisis de datos visualmente

- Interactivo: Notebook.
- Forma declarativa.

## Diagramas interactivos

- Explorar datos.
- Panel de control interactivo.

# Ejemplo: Caso de estudio

## Vamos a ver un ejemplo

**Altair** Librería en contrucción declarativa.

**Objetivo** Explorar tiempo de Seattle.

## Datos

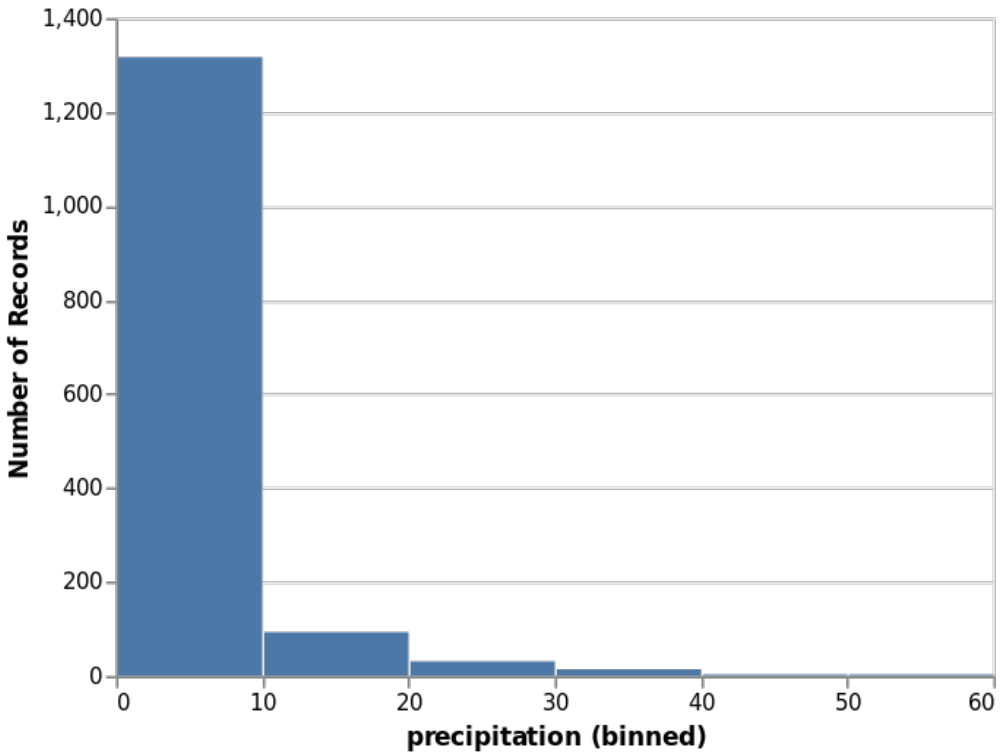
```
df.head()
```

| date | precipitation | temp_max | temp_min | wind | weather |
|---|---|---|---|---|---|
| 2012-01-01 | 0.0 | 12.8 | 5.0 | 4.7 | drizzle |
| 2012-01-02 | 10.9 | 10.6 | 2.8 | 4.5 | rain |
| 2012-01-03 | 0.8 | 11.7 | 7.2 | 2.3 | rain |
| 2012-01-04 | 20.3 | 12.2 | 5.6 | 4.7 | rain |
| 2012-01-05 | 1.3 | 8.9 | 2.8 | 6.1 | rain |

# Contar las precipitaciones

## Código

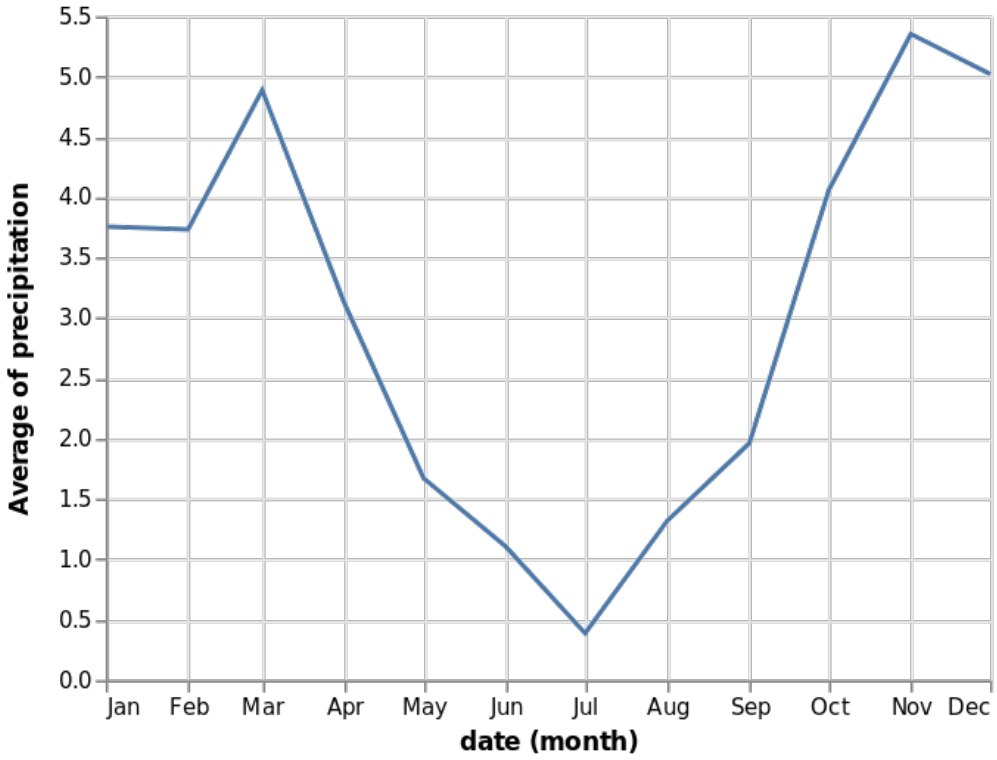
```
alt.Chart(df).mark_bar().encode(
    alt.X("precipitation", bin=True),
    alt.Y("count()")
)
```



# Precipitaciones por mes

## Código

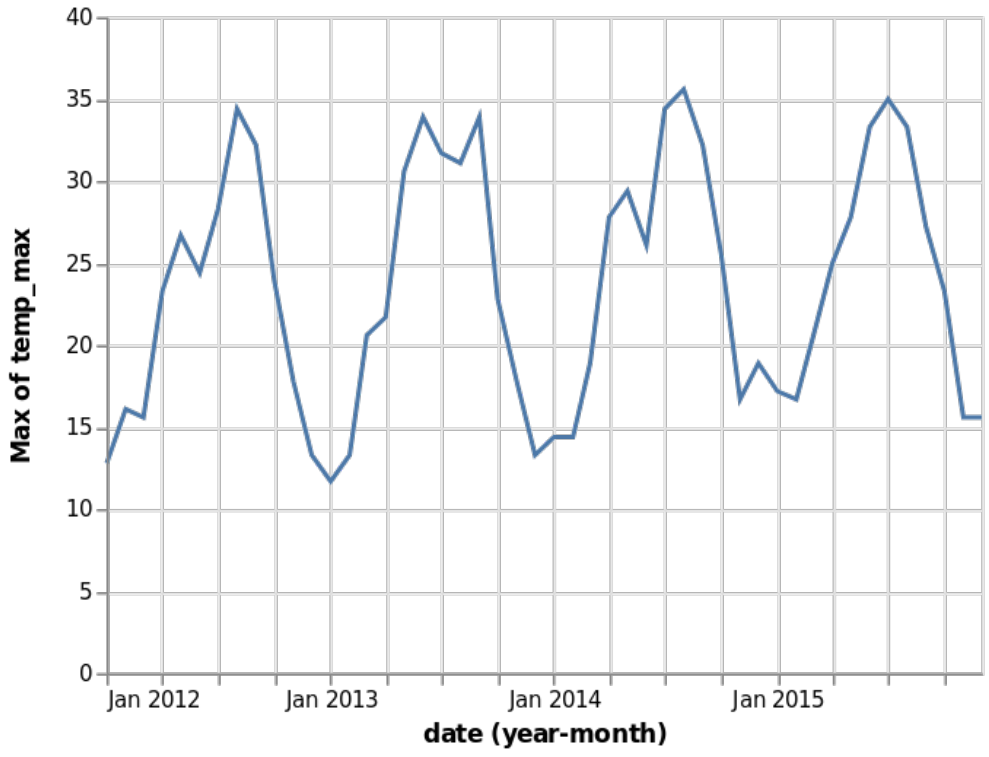
```
alt.Chart(df).mark_line().encode(
    alt.X("date:T", timeUnit="month"),
    alt.Y("average(precipitation)")
)
```



# Precipitaciones por año y mes

## Código

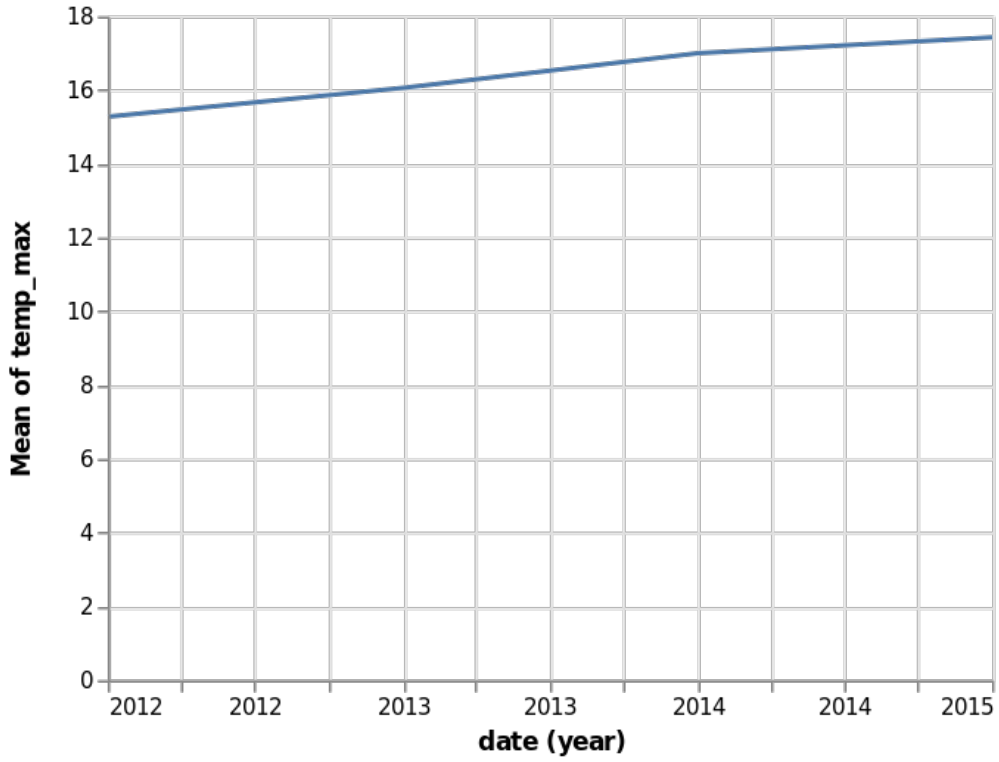
```
alt.Chart(df).mark_line().encode(
    alt.X("date:T", timeUnit="yearmonth"),
    alt.Y("max(temp_max)"),
)
```



# Calentamiento por año

## Código

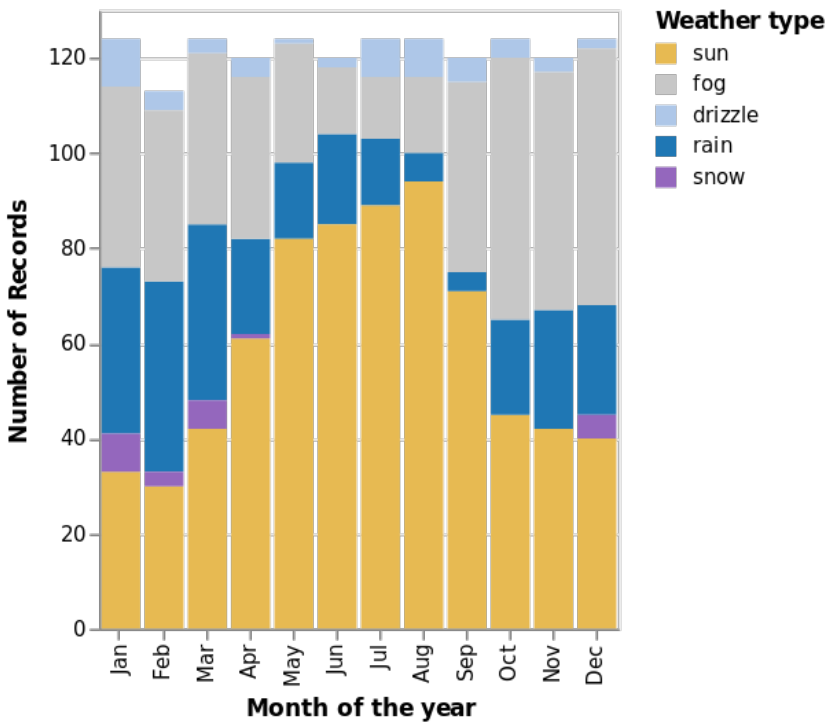
```
alt.Chart(df).mark_line().encode(
    alt.X("date:T", timeUnit="year"),
    alt.Y("mean(temp_max)"),
)
```



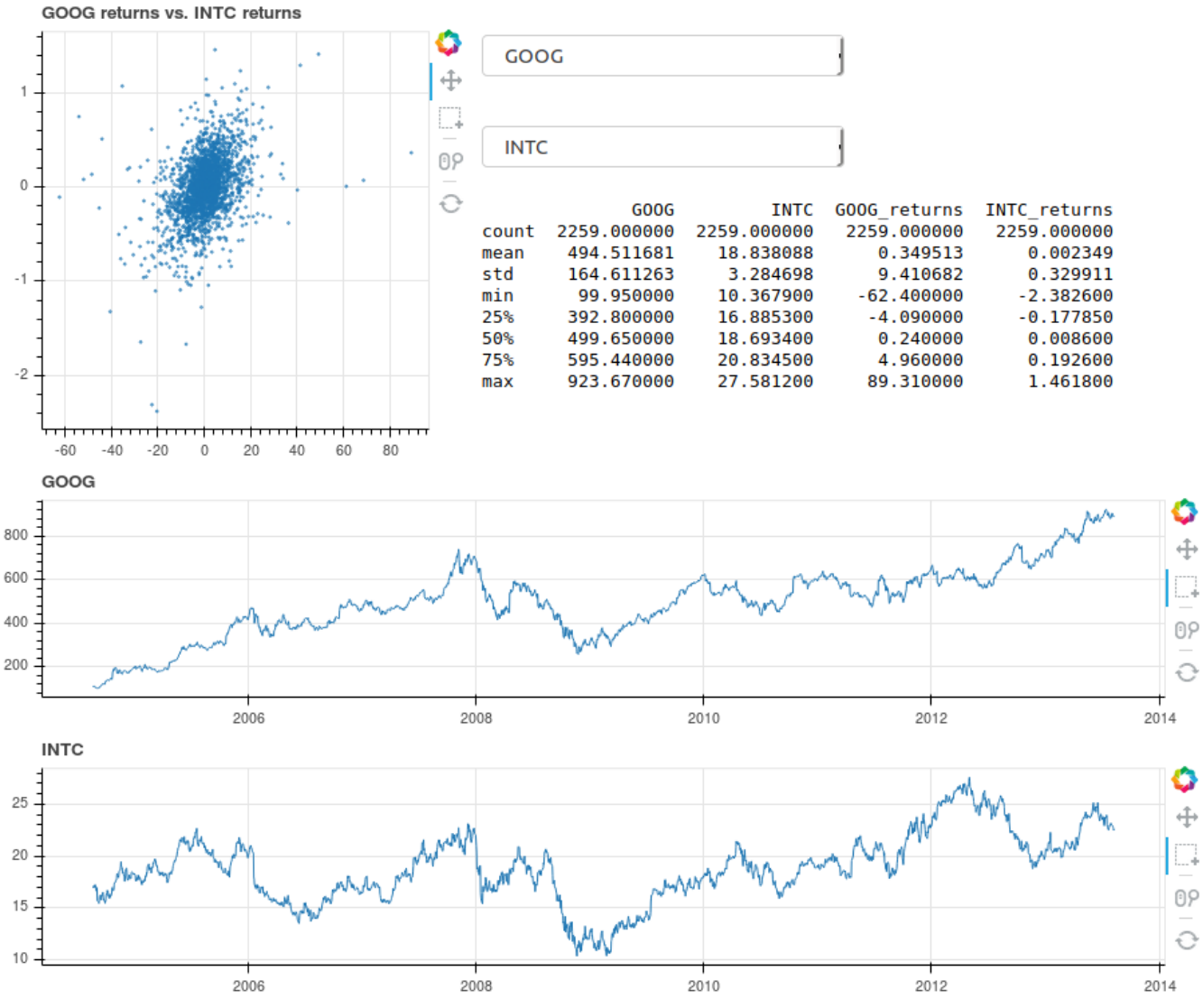
# Tipo de día

## Código

```
alt.Chart(df).mark_bar().encode(
    x=alt.X("date:N", timeUnit="month"),
    y="count()", color=alt.Color("weather",
    legend=alt.Legend(title="Weather type"), scale=scale),
)
```

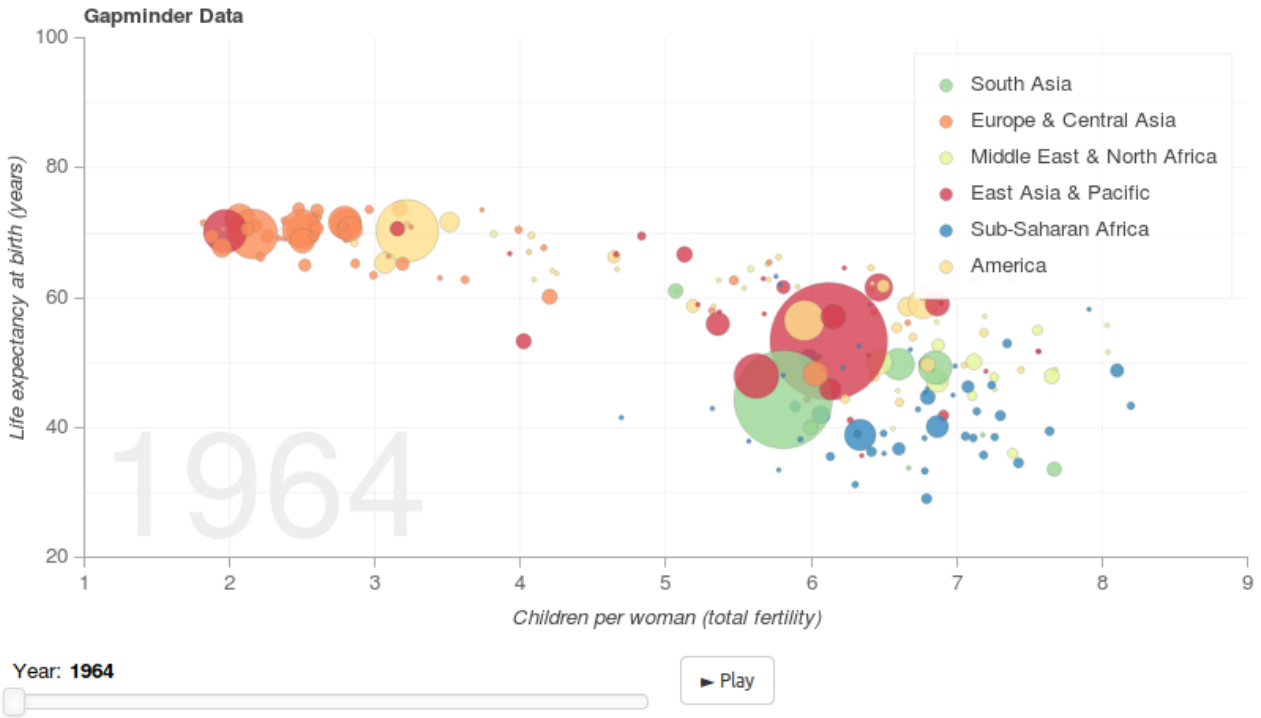


# Demo de diagrama interactivo




| | GOOG | INTC | GOOG_returns | INTC_returns |
|---|---|---|---|---|
| count | 2259.000000 | 2259.000000 | 2259.000000 | 2259.000000 |
| mean | 494.511681 | 18.838088 | 0.349513 | 0.002349 |
| std | 164.611263 | 3.284698 | 9.410682 | 0.329911 |
| min | 99.950000 | 10.367900 | -62.400000 | -2.382600 |
| 25% | 392.800000 | 16.885300 | -4.090000 | -0.177850 |
| 50% | 499.650000 | 18.693400 | 0.240000 | 0.008600 |
| 75% | 595.440000 | 20.834500 | 4.960000 | 0.192600 |
| max | 923.670000 | 27.581200 | 89.310000 | 1.461800 |

# Demo de diagrama interactivo



# Otras librerías imprescindibles

# Dask

# Índice

1 Python para la ciencia

2 Python y la programación web

3 Comunicaciones con Python

# Web Scraping

# Índice

1 Python para la ciencia

2 Python y la programación web

3 Comunicaciones con Python