# Manejo de Librerías

## Manejo de Librerías

Jonatan Gómez Perdomo, Ph. D.
jgomezpe@unal.edu.co

Arles Rodríguez, Ph.D.
aerodriguezp@unal.edu.co

Camilo Cubides, Ph.D. (c)
eccubidesg@unal.edu.co

Carlos Andrés Sierra, M.Sc.
casierrav@unal.edu.co

Research Group on Artificial Life – Grupo de investigación en vida artificial – (Alife)
Computer and System Department
Engineering School
Universidad Nacional de Colombia

# Agenda



# Numpy

Es una de las librerías más conocidas para realizar computación científica en Python. Provee una librería de alto rendimiento para la manipulación de arreglos en varias dimensiones, además de ofrecer herramientas sofisticadas para manipularlos.

Para los códigos que veremos hoy se recomienda trabajarlos en

https://colab.research.google.com/

# Arreglos en Numpy

- Un arreglo en `numpy` es una retícula de valores del mismo tipo indexadas por enteros no negativos
- El número de dimensiones (`rank`) y la forma (`shape`) del arreglo es una tupla de enteros que da el tamaño del arreglo para cada dimensión.
- Se pueden crear arreglos de `numpy` desde listas de Python y acceder a los elementos con el operador *`subscript`* `[]`.

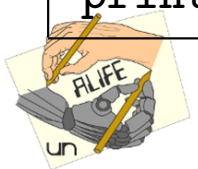
# Utilizando la librería numpy

Para utilizar la librería numpy utilizamos `import numpy as np`

```
import numpy as np

a = np.array(list(range(1,5)))   # Crea un arreglo lineal
print(type(a))     # Imprime "<class 'numpy.ndarray'>"
print(a)
print(a.shape)     # Imprime "(4,)" es de tamaño 4, de 1 dimensión
print(a[0], a[1], a[2])
a[0] = -4
print(a)

b = np.array([[1,2,3,5,6],[4,5,6,7,8]]) # Crea un arreglo 2-dimensional
print(b.shape)
print(b)
b[0,0] = 1590
print(b)
print(b[0,0], b[0,1], b[1,0])
```




# Algunas funcionalidades de numpy

Es posible crear arreglos de diferente tipo de forma muy rápida:

```
a = np.zeros((2,3,4))
                # Crea una matriz 3-dimensional (2x3x4) de ceros (0's)
print(a.shape)
print(a)
print("---------")

b = np.ones((2,3))     # Crea una matriz de 2x3 de unos (1's)
print(b.shape)
print(b)
print("---------")
```

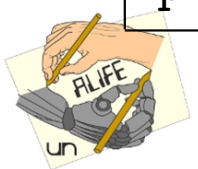
# Operador *subscript*

El operador *subscript* funciona en numpy:

```
import numpy as np

# Crea un arreglo 2-dimensional con forma (3, 4)
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
print(a.shape)
print(a)
b = a[:2, 1:3]
# El primer argumento indica las filas y el segundo las columnas
print(b)
print("------------------------")

# Si se modifica algo de b, se cambia algo de a
b[0, 0] = -11     # b[0, 0] es el mismo a[0, 1]
print(b)
print(a)
print(a[0,1])   # Imprime "-11"
```




# Determinación del tipo de dato

Numpy como Python determinan el tipo de dato basado en el valor. Sin embargo, dicho tipo también puede especificarse

```
import numpy as np

x = np.array([5, -4])
print(x.dtype)

x = np.array([1.0, 2.0])
print(x.dtype)

x = np.array([5, -4], dtype=np.int32)
print(x.dtype)
```

La salida del código anterior es:

```
int64
float64
int32
```

# Operaciones elemento a elemento

Es posible realizar operaciones elemento a elemento:

```
import numpy as np

x = np.array([[1,2,5], [3,4,6]], dtype=np.float128)
y = np.array([[5,6,-1], [7,8,-6]], dtype=np.float128)

print("Suma:")
print(x + y)
print("-----")
print(np.add(x, y))

print("raiz cuadrada:")
print(np.sqrt(x))
```


# Operaciones elemento a elemento

La salida del código es la siguiente:

```
Suma:
[[ 6.  8.  4.]
 [10. 12.  0.]]
-----
[[ 6.  8.  4.]
 [10. 12.  0.]]
raiz cuadrada:
[[1.         1.41421356 2.23606798]
 [1.73205081 2.         2.44948974]]
```

# np.linspace

Retorna datos en el intervalo `start` hasta `stop` que incluyen `num` datos:

```
import numpy as np

np.linspace(2, 3, num=10, endpoint=True, retstep=False)
```

La salida del programa es la siguiente:

```
array([2.        , 2.11111111, 2.22222222, 2.33333333, 2.44444444,
       2.55555556, 2.66666667, 2.77777778, 2.88888889, 3.        ])
```

# Agenda



# Matplotlib

- `Matplotlib` es una librería para crear visualizaciones estáticas o animadas en Python.
- Es posible graficar en un área con uno o más ejes (en términos de coordenadas `x-y`, `theta-r`, coordenadas polares, `x-y-z`, etc.
- La forma más simple de crear una figura con ejes es usar el módulo `pyplot`.

La documentación oficial de la librería se puede encontrar en

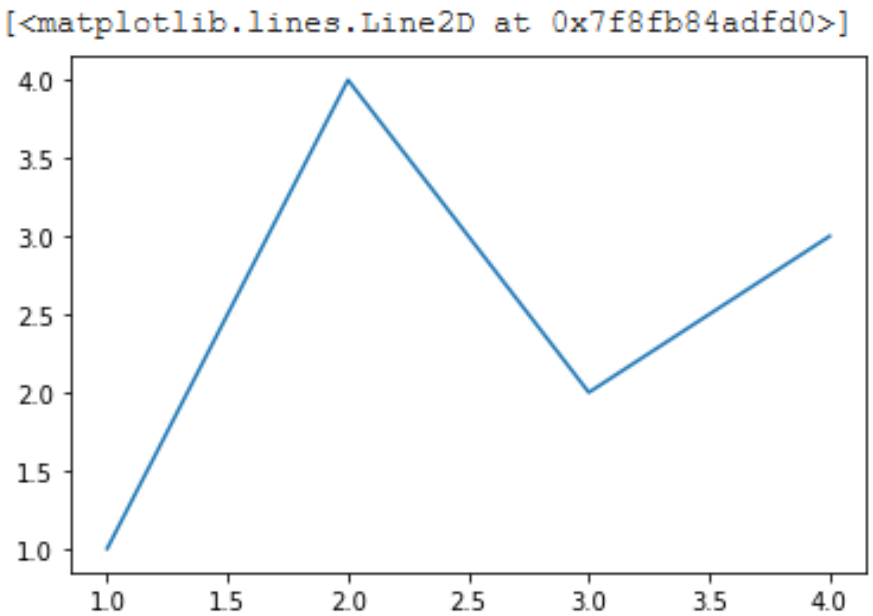
https://matplotlib.org/stable/tutorials/introductory/usage.html

# Generación de una gráfica sencilla (I)

Con el siguiente código, es posible generar una gráfica sencilla:

```
import matplotlib.pyplot as plt
# Matplotlib plot.
plt.plot([1, 2, 3, 4], [1, 4, 2, 3])
```

La salida del código anterior en colab es como la siguiente:



# Generación de una gráfica sencilla (II)

Es posible combinar librerías. En el siguiente código, se generan 50 puntos en el intervalo `[0,2]` y se grafican algunas funciones conocidas.

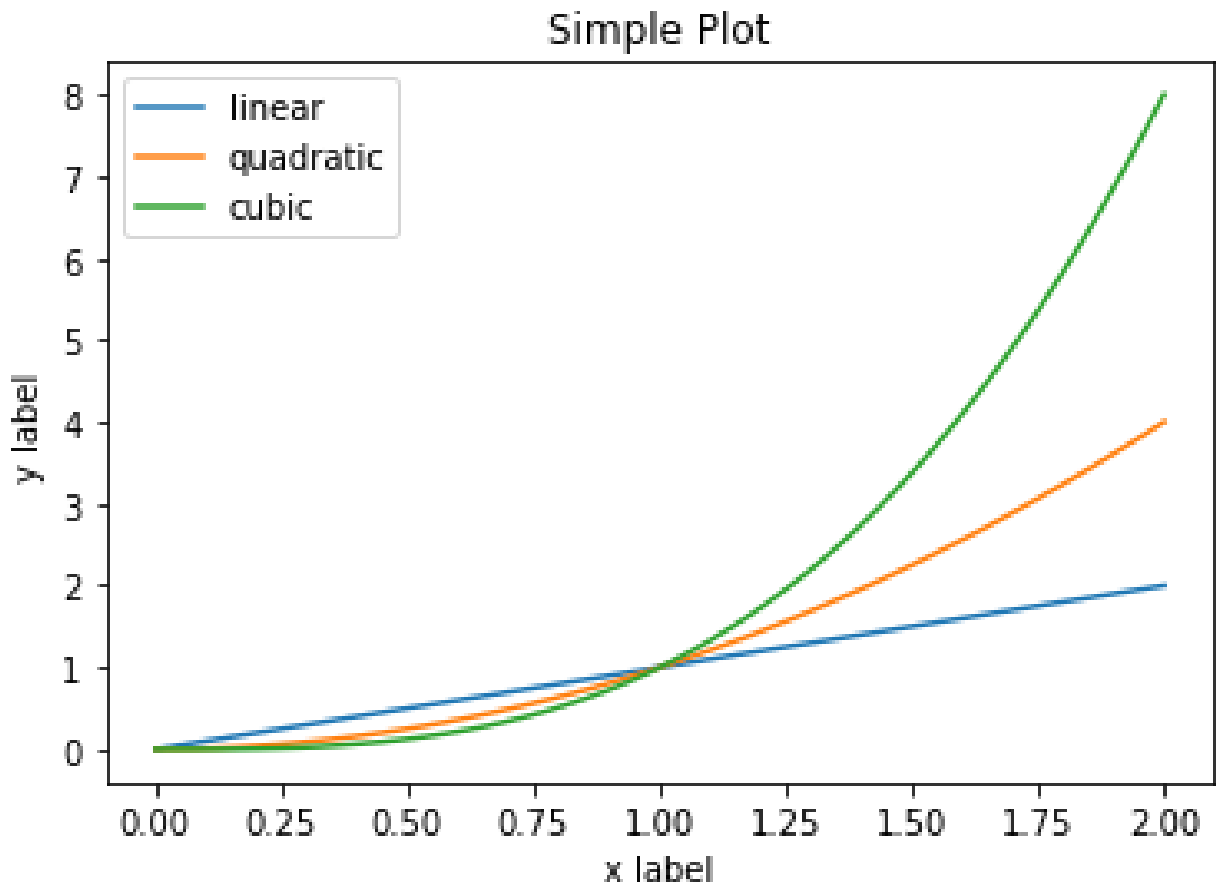
```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 2, 50)
#print(x)
# Aún con el OO-style, usamos ".pyplot.figure" para crear la figura.
fig, ax = plt.subplots()  # Crea la figura y los ejes.
ax.plot(x, x, label="linear")  # Dibuja algunos datos en los ejes.
ax.plot(x, x**2, label="quadratic")  # Dibuja mas datos en los ejes.
ax.plot(x, x**3, label="cubic")  # ... y algunos más.
ax.set_xlabel("x label")  # Agrega un x-label a los ejes.
ax.set_ylabel("y label")  # Agrega un y-label a los ejes.
ax.set_title("Simple Plot")  # Agrega título a los ejes.
ax.legend()  # Agrega una leyenda.
```


# Generación de una gráfica sencilla (III)

La salida del código anterior en colab es como la siguiente:

```
<matplotlib.legend.Legend at 0x7f8fb7f8c410>
```



# Graficas a partir de grupos (I)

Es posible generar distintos tipos de gráfica y agruparlos:

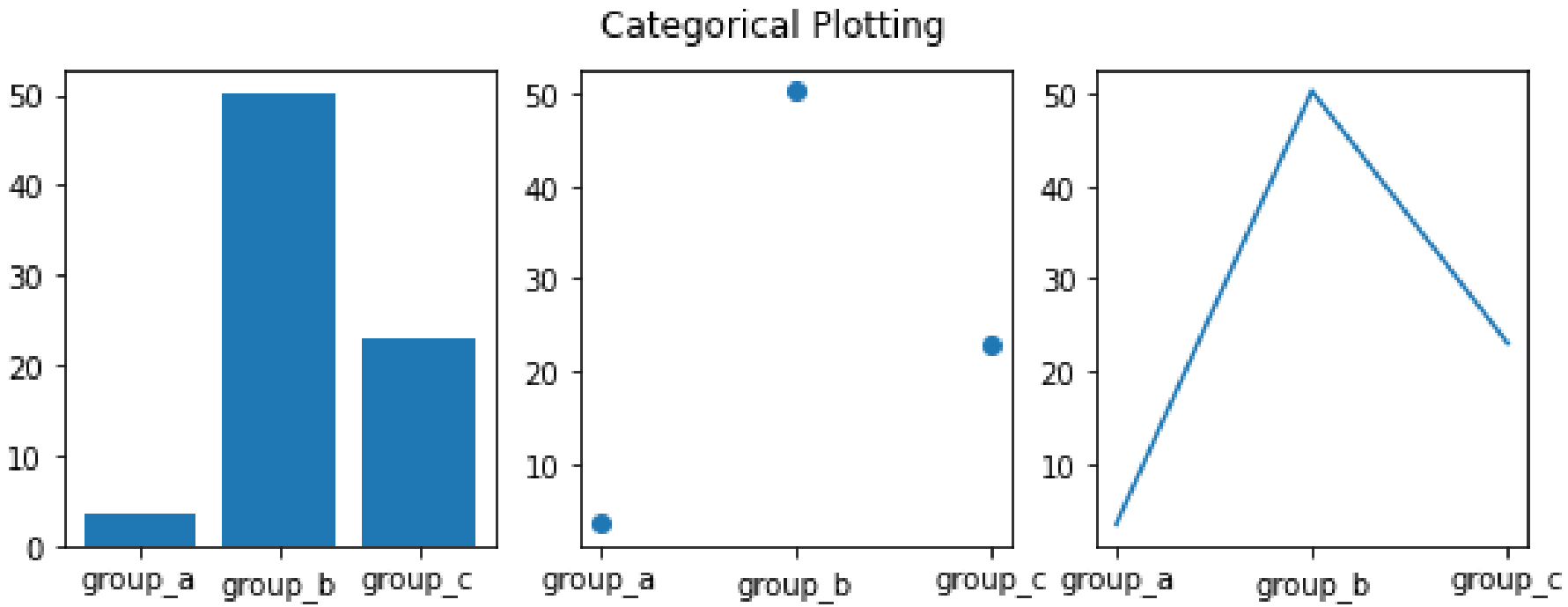
```
names = ["group_a", "group_b", "group_c"]
values = [3.4, 50.3, 23]

plt.figure(figsize=(9, 3))

plt.subplot(131)
plt.bar(names, values)
plt.subplot(132)
plt.scatter(names, values)
plt.subplot(133)
plt.plot(names, values)
plt.suptitle("Categorical Plotting")
plt.show()
```

# Graficas a partir de grupos (II)

La salida del código anterior en `colab` es como la siguiente:



# Agenda



# Pandas

Permite manipular datos de alto nivel. Es una librería fácil de usar para análisis y manipulación de datos. Está construida sobre Numpy.

```
dictc = {"country": ["Brazil", "Russia", "India",
                     "China", "South Africa", "Colombia"],
        "capital": ["Brasilia", "Moscow", "New Dehli",
                    "Beijing", "Pretoria", "Bogotá"],
        "area": [8.516, 17.10, 3.286, 9.597, 1.221, 1.142],
        "population": [200.4, 143.5, 1252, 1357, 52.98, 49.65] }

import pandas as pd

brics = pd.DataFrame(dictc)
print(brics)
```

# Pandas

La salida del código anterior es la siguiente:

| | country | capital | area | population |
|---|---|---|---|---|
| 0 | Brazil | Brasilia | 8.516 | 200.40 |
| 1 | Russia | Moscow | 17.100 | 143.50 |
| 2 | India | New Dehli | 3.286 | 1252.00 |
| 3 | China | Beijing | 9.597 | 1357.00 |
| 4 | South Africa | Pretoria | 1.221 | 52.98 |
| 5 | Colombia | Bogotá | 1.142 | 49.65 |

# Carga de archivos

- Pandas permite manipular archivos de tipo `*.csv`. Para el siguiente código puede descargar este archivo: click aquí para descargar.
- Para cargar el archivo a colab agregue el siguiente código a una celda y suba el archivo descargado.

```
from google.colab import files
uploaded = files.upload()  #Upload files/SalesJan2009.csv
```

```
# to use in case of Google Colab

from google.colab import files
uploaded = files.upload()  #Upload files/SalesJan2009.csv
```

Elegir archivos SalesJan2009.csv

- **SalesJan2009.csv**(application/vnd.ms-excel) - 129482 bytes, last modified: 17/5/2021 - 100% done

```
Saving SalesJan2009.csv to SalesJan2009.csv
```

# Procesamiento de archivos csv (I)

Es posible cargar los datos y visualizar como se ven para pandas. El archivo contiene una lista de transacciones realizadas con diferentes medios de pago en diferentes países:

```
# Import pandas as pd
import pandas as pd
from collections import Counter

ventasdf = pd.read_csv("SalesJan2009.csv")
ventasdf.head(3)
```

| | Transaction_date | Product | Price | Payment_Type | Name | City | State | Country |
|---|---|---|---|---|---|---|---|---|
| 0 | 1/2/2009 6:17 | Product1 | 1200 | Mastercard | carolina | Basildon | England | United Kingdom |
| 1 | 1/2/2009 4:53 | Product1 | 1200 | Visa | Betina | Parkville | MO | United States |
| 2 | 1/2/2009 13:08 | Product1 | 1200 | Mastercard | Federica e Andrea | Astoria | OR | United States |

# Procesamiento de archivos csv (II)

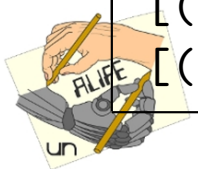
Es posible cargar los datos, listar los 3 países y las 3 franquicias de tarjetas en las que se realizan más transacciones:

```
import pandas as pd
from collections import Counter

ventasdf = pd.read_csv("SalesJan2009.csv")
#print(ventas)
cp = Counter(ventasdf["Country"])
print(cp.most_common(3))
cv = Counter(ventasdf["Payment_Type"])
print(cv.most_common(3))
```

La salida del código anterior es:

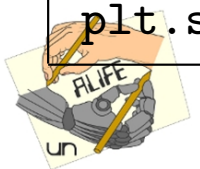
```
[('United States', 462), ('United Kingdom', 100), ('Canada', 76)]
[('Visa', 521), ('Mastercard', 277), ('Amex', 110)]
```



# Procesamiento de archivos csv (III)

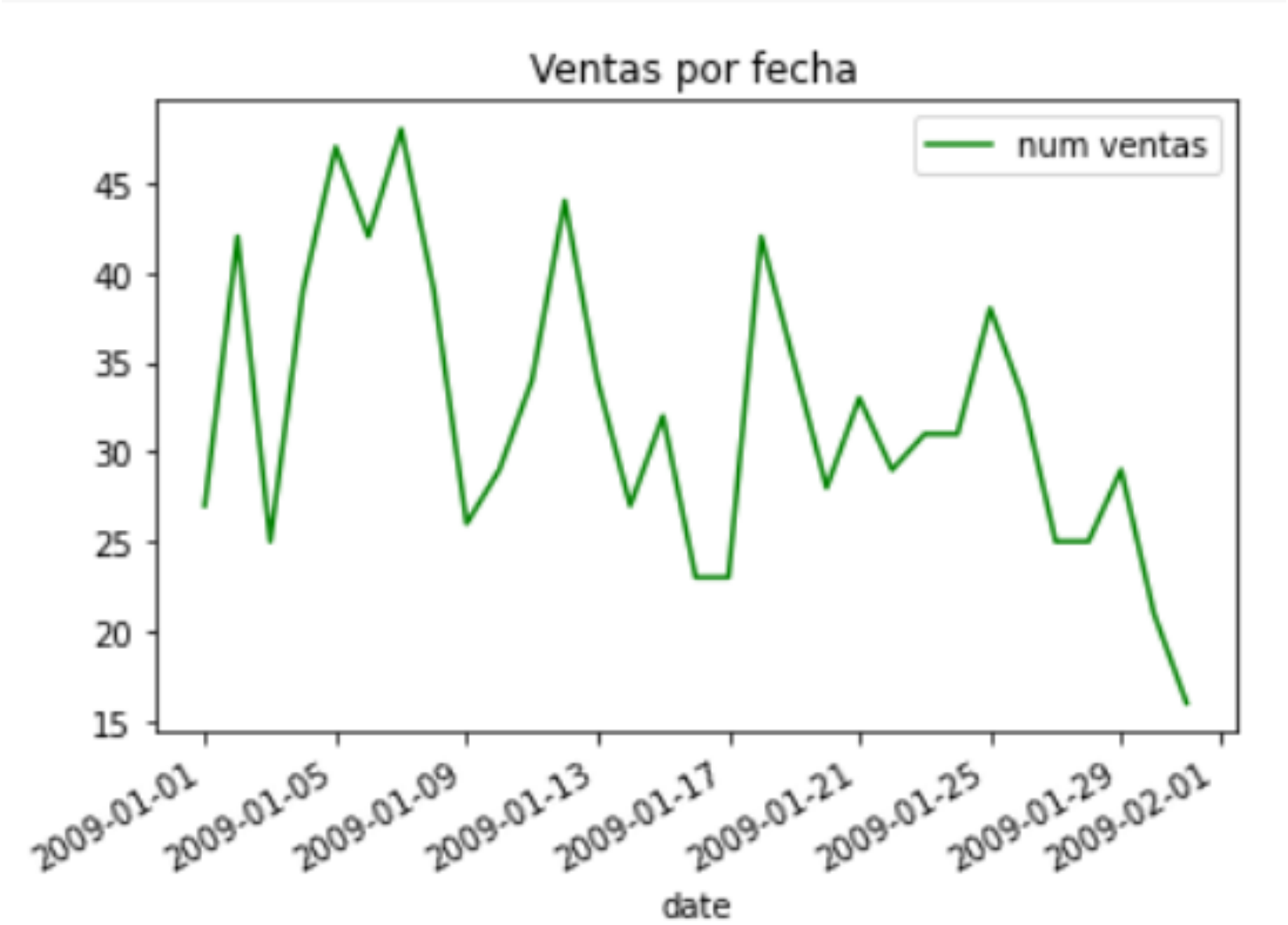
- Pandas puede ser utilizado en conjunto con otras librerías como `Matplotlib`.
- Adicionalmente es posible realizar consultas y operaciones sobre los `*.csv`. A continuación se graficarán las ventas por fecha.

```
import pandas as pd
import datetime
import matplotlib.pyplot as plt
#Reporte por fecha
ventasdf["Transaction_date"]=pd.to_datetime(ventasdf["Transaction_date"])
A = (ventasdf["Transaction_date']
        .dt.floor("d")
        .value_counts()
        .rename_axis("date")
        .reset_index(name="num ventas"))
G=A.plot(x="date",y="num ventas",color="green",title="Ventas por fecha")
plt.show()
```




# Procesamiento de archivos csv (IV)

La salida del código anterior es la siguiente:



# Referencias


- https://cs231n.github.io/python-numpy-tutorial/
- https://matplotlib.org/Matplotlib.pdf
- https://pandas.pydata.org/
- https://www.learnpython.org/es/Pandas_Basics
- Rodríguez, A (2020). Curso de Programación en Python. https://github.com/arleserp/cursopython