Computación Blanda

Soft Computing

Analisis de datos con numpy sobre Covid_19

Autores: Juan José León Tabares, Luis Aníbal Loaiza Cardona
IS&C, Universidad Tecnológica de Pereira, Pereira, Colombia
Correo-e: juanjose.leon@utp.edu.co Luis.loaiza@utp.edu.co

***Resumen*— Este documento tiene como finalidad fomentar la programación en Python con la librería numpy para aplicarla en problemas complejos como el Covid 19.**
Palabras clave*— *numpy, vector, arreglo, atributo, variable, función, datos.

***Abstract*— This document is intended to promote Python programming with the numpy library to apply it to complex problems as the Covid 19.**

Keywords: numpy, vector, array, attribute, variable, function, data.

I. INTRODUCCIÓN

Se realiza un algoritmo junto con la librería numpy de Python para filtrar los datos de un archivo .tsv el cual contiene en las columnas los días totales, los casos positivos y las muertes de Covid 19 en Colombia en el transcurso de la pandemia, además podemos realizar las diferentes operaciones con estos datos, para mostrar visualmente la propagación de este virus los meses correspondientes a su inicio.

II. ANALISIS

El primer paso es extraer los datos que tiene un archivo .tsv, principalmente se basó en la información relativa en los casos positivos y las muertes de Covid 19 cada día durante el transcurso de la pandemia. Dichos datos están almacenado en Covid_19.tsv.

```
data = np.genfromtxt("Covid_19.tsv", delimiter="\t")
```

Con la función anterior se convierte un archivo de texto a una matriz para poder operar los datos más fácilmente, que tiene como valores de entrada el nombre del archivo y un delimitador, dentro de este archivo se pudieron encontrar tres tipos de datos, la primera columna los días transcurridos de la pandemia, la segunda columna los casos positivos y la tercer columna con las muertes, estos datos son tomados diariamente.

Con los siguientes arreglos se logra dividir la información de los datos:

```
x = data[:,0]
y = data[:,1]
z = data[:,2]
```

En la posición cero (0) se encuentran cada uno delos días transcurridos y se almacenan en la variable X, mientras que en la posición uno (1) se encuentran los casos positivos diarios y se almacena en la variable Y, la posición dos (2) se encuentra las muertes causadas diariamente por la pandemia y se almacenan en la variable Z.

Ya con los datos debidamente separados en las variables se da a la tarea de saber las dimensiones y la cantidad de datos que compone cada variable, para esto se toma la siguiente línea de código:

- Con el atributo ndim se puede retornar el número de dimensiones del vector.

  Dimensiones de los datos de las variables:
  ```
  print(x.ndim, '\n')
  print(y.ndim, '\n')
  print(z.ndim, '\n')
  ```

- Con el atributo shape se puede retornar la cantidad de datos del vector.

  Cantidad de datos en las variables:
  ```
  print(x.shape)
  print(y.shape)
  print(z.shape)
  ```

Analizando la cantidad de datos en los vectores, se llegó a que hay unos valores nulos (como ejemplo: nan). Por lo tanto, se debe hallar alguna forma para eliminar estos valores, ya que si no se hacen pueden afectar a la entrega de los resultados solicitados.

```
print(np.sum(np.isnan(x)))
print(np.sum(np.isnan(y)))
print(np.sum(np.isnan(z)))
```

En la función anterior np.isnan que como valor de entrada recibe un vector, lo que realiza es convertir los datos nulos (nan) en valores booleanos en este caso será True para identificar que existe un valor nulo. La función np.sum lo que realiza es la suma de cada uno de los valores booleanos que son True para retornar el total de valores nulos que contenga el vector, como valor de entrada recibe la vector con valores booleanos y retorna un numero entero indicando la cantidad de datos nulos.

Al eliminar los datos nulos (nan) se presente un problema con las tres columnas debido a que cada columna tiene diferentes cantidades de valores erróneos (nan); para resolver este problema se tiene en cuenta el siguiente código de línea:

Se eliminan los elementos nan de x:

```
z = z[~np.isnan(x)]
y = y[~np.isnan(x)]
x = x[~np.isnan(x)]
```

Se eliminan los elementos nan de Y:

```
x = x[~np.isnan(y)]
z = z[~np.isnan(y)]
y = y[~np.isnan(y)]
```

Se eliminan los elementos nan de Z:

```
y = y[~np.isnan(z)]
x = x[~np.isnan(z)]
z = z[~np.isnan(z)]
```

Basado en el código anterior lo primero que se realiza es eliminar los valores nan consecutivamente, desde la X hasta la Z. Se le asigna a X un vector X con todas las posiciones nulas (nan) eliminadas por esta razón el comando es ~np.isnan. En tanto a la variable Y y Z se le realiza el mismo procedimiento pero con sus respectivos vectores.

Con estos nuevos vectores se utilizaran para realizar tres (3) graficas distintas.

Con la función de matplotlib plt.scatter permite dibujar los puntos en el plano cartesiano con un tamaño determinado, la función plat.scatter recibe un arreglo de posiciones tanto en X como en Y, por ultimo recibe el tamaños de los puntos.

```
plt.scatter(x, y, s=10)
```

La función plt.xticks realiza la demarcación de cada mes multiplicando la cantidad de días que tiene un mes, aplicándolo en un rango de diez (10). Recibe como valor de entrada cada cuanto debe de hacer la demarcación con un valor de W que funciona como un puntero recorriendo la gráfica contando los puntos, este proceso de demarcación lo realiza en un rango de diez (10) veces, además recibe una cadena en este caso MESES con una variable %i que funciona como contador.

```
plt.xticks([w*30 for w in range(10)],
['meses %i' % w for w in range(10)])
```

Para realizar la cuadricula se utiliza plt.grid que recibe como entrada un valor booleano que indica si queremos ver o no la cuadricula, como segundo valor se debe indicar el tipo de línea que se quiere utilizar, y como tercer valor entre 0 y 1 recibe la tonalidad de la línea de división y por ultimo con la función plt.show muestra la gráfica en pantalla.

```
plt.grid(True,linestyle='-',color='0.75')

plt.show()
```

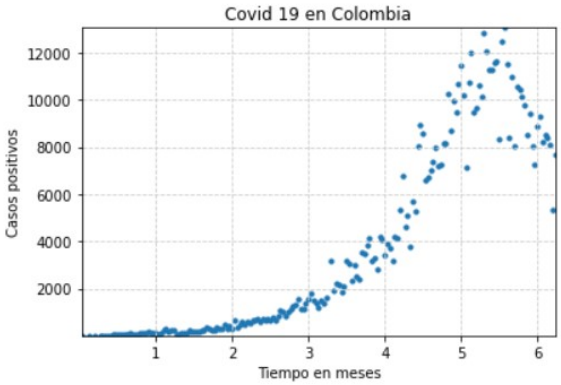

Figura 1. Casos positivos.

Se realiza el mismo procedimiento para las muertes y se representa en la siguiente grafica:

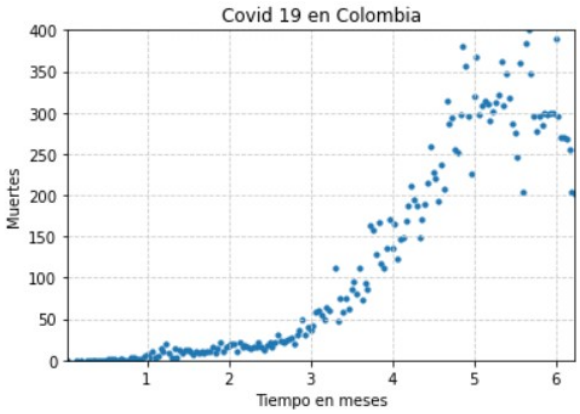

Figura 2. Muertes registradas.

Por ultimo se realiza la comparacion de los casos positivos y las muertes en una misma grafica, donde los puntos azules son los casos positivos y los puntos naranjas son las muertes registradas.

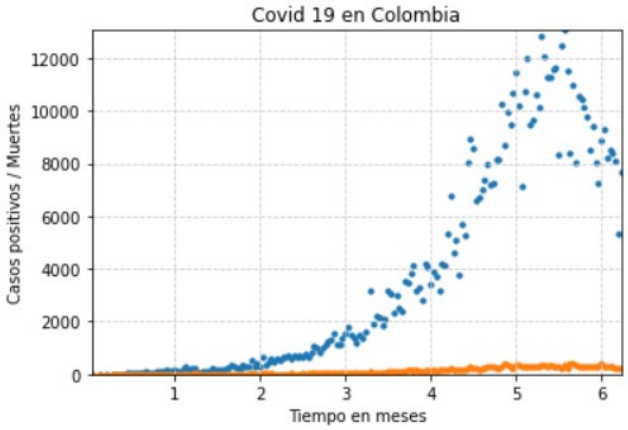

Figura 3. Casos positivos y muertes.

A continuación se importan las librerías *Scipy y matplotlib.* SciPy contiene módulos para optimización, álgebra lineal, integración, interpolación, funciones especiales, FFT, procesamiento de señales y de imagen, resolución de ODEs y otras tareas para la ciencia e ingeniería. SciPy se basa en el objeto de matriz NumPy y es parte del conjunto NumPy, que incluye herramientas como Matplotlib, pandas y SymPy, y un conjunto en expansión de bibliotecas de computación científica. Este conjunto está dirigido al mismo tipo de usuarios que los de aplicaciones como MATLAB, GNU Octave, y Scilab.

```
import scipy as sp
```

La función plt_models se encarga de graficar y guardar los modelos en formato .png y como valores de entrada X, Y (Vectores de Covid_19) fname Nombre con el cual se va a guardar la gráfica models modelo implementado.

```
def plot_models(x, y, models, fname, mx=None, ymax=None, xmin=None):
```

La función linspace devuelve números espaciados uniformemente durante un intervalo especificado. En este caso, sobre el conjunto de valores x establecido. En el caso de x [-1], lo realiza hasta el último dato del arreglo X.

```
mx = np.linspace(0, x[-1], 1000)
```

La zip () función devuelve un objeto zip, que es un iterador de tuplas donde el primer elemento de cada iterador pasado se empareja, y luego el segundo elemento de cada iterador pasado se empareja, etc.

Es un emparejamiento por cada elemento según su orden en el arreglo por ejemplo model será un vector dentro de otro que contiene la lista de los 4 elementos:

```
    for model, style, color in zip(models, linestyles, colors):
```

plt.plot recibe los valores de los vectores, que se van a graficar, además también recibe las características del grafico como el tipo de línea, el tamaño y el color.

mx se refiere a los puntos de entrada que serán evaluados, model (mx), son los puntos evaluados en el modelo producido.

```
            plt.plot(mx,        model(mx), linestyle=style, linewidth=2, c=color)
```

La siguiente línea de código indica el orden de la ecuación que se digito al lado izquierdo.

```
        plt.legend(["d=%i" % m.order for m in models], loc="upper left")
```

plt.savefig guarda la imagen con el respectivo nombre (Fname) al inicio de la función.

```
plt.savefig(fname)
```

Polyfit es el ajuste de un polinomio de grado deg a los puntos (x, y). Devuelve un vector de coeficientes P que minimiza el error al cuadrado en el orden deg , `deg-1 ,… 0 .p(x) = p[0] * x**deg + ... + p[deg].`

Ya que full es verdadero devuelve cuatro datos demás (residuos (res1), rango (rank1), valores individuales (sv1), segundo (rcond)). Presente solo si full = Verdadero. Los residuos son la suma de los residuos cuadrados del ajuste por mínimos cuadrados, el rango efectivo de la matriz de coeficientes de Vandermonde escalada, sus valores singulares y el valor especificado de rcond.

```
fp1,    res1,    rank1,    sv1,    rcond1    = np.polyfit(x, y, 1, full=True)
```

Básicamente lo que realiza Polyfit es encontrar un modelo o una ecuación que representa todos los datos, el modelo encontrado en este caso es fp1.

f1 es el modelo fp1 más preciso, ya que poly1d permite reducir los errores.

```
f1 = sp.poly1d(fp1)
```

numpy.poly1d (arr, root, var): esta función ayuda a definir una función polinomial. Facilita la aplicación de "operaciones naturales" en polinomios. Pero a simple vista descompone un polinomio en factores de un grado más bajo, en resumen

vuelve más exacto el modelo descomponiéndolo en factores más simples.

```
f3 = sp.poly1d(np.polyfit(x, y, 3))
f10 = sp.poly1d(np.polyfit(x, y, 10))
f100 = sp.poly1d(np.polyfit(x, y, 100))
```

Esta es la versión traducida para obtener modelos más precisos, además con el último número podemos variar el grado del polinomio. Los siguientes modelos se graficaran y se guardan con la función `plot_models.`

```
plot_models(x, y, [f1], os.path.join(CHART_DIR, "1400_01_02.png"))
plot_models(x, y, [f1, f2], os.path.join(CHART_DIR, "1400_01_03.png"))
plot_models(x, y, [f1, f2, f3, f10, f100], os.path.join(CHART_DIR, "1400_01_04.png"))
```

Ajusta y dibuja un modelo utilizando el conocimiento del punto de inflexión; Un punto de inflexión, en una función matemática, es un punto donde los valores de una función continua en X pasan de un tipo de concavidad a otra, 5.7 especifica el punto donde el crecimiento en la curva empezó a cambiar y 30 indica la cantidad de días que tiene un mes.

```
inflexión = 5.7 * 30
```

Toma todos los puntos desde el inicio x hasta el punto inflexión.

```
xa = x[:int(inflexión)]
```

Toma todos los puntos desde el inicio y hasta el punto inflexión.

```
ya = y[:int(inflexión)]
```

Toma desde el punto inflexión hasta el último valor en X.

```
xb = x[int(inflexión):]
```

Toma desde el punto inflexión hasta el último valor en Y

```
yb = y[int(inflexión):]
```

```
fa = sp.poly1d(np.polyfit(xa, ya, 1))
fb = sp.poly1d(np.polyfit(xb, yb, 1))
```

Teniendo los dos modelos se agrupan en una gráfica, ya que esto es una forma de reducir la complejidad en el modelo matemático ya que el problema se divide en dos no se sobrecarga una función haciendo que acapare todos los puntos si no que se le da una zona en específico.

La función de error:

```
def error(f, x, y):
    return np.sum((f(x) - y) ** 2)
```

Se resta el modelo evaluado con el valor original, para determinar el error o que tanto están separados de si, se eleva a una potencia de dos por la razón de que algunos errores son negativos ocasionando que el resultado en la sumatoria sea igual a cero.

```
fb1 = fb
fb2 = sp.poly1d(np.polyfit(xb, yb, 2))
fb3 = sp.poly1d(np.polyfit(xb, yb, 3))
fb10 = sp.poly1d(np.polyfit(xb, yb, 10))
fb100 = sp.poly1d(np.polyfit(xb, yb, 100))
```

Graficas con punto de inflexión con distinto grado polinomial.

> Nota: Entre mayor grado polinomial se asemeja mas al comportamiento de los puntos, recordar que entre mayor grado polinomial la complejidad aumenta porque en el modelo solo existe por los puntos por lo cual su futuro o predicción es casi inexistente ya que a la hora de evaluarlo será casi imposible

Separa el entrenamiento de los datos de prueba. Se extrae el 30%de los datos se mezclan y luego se ordenan.

30% en decimal.

```
frac = 0.3
```

Se obtiene el 30% de longitud del vector.

```
split_idx = int(frac * len(xb))
```

Se mezclan los índices de los datos.

```
shuffled = sp.random.permutation(list(range(len(xb))))
```

Se toma datos del 30% de la información, luego se ordenan.

```
test = sorted(shuffled[:split_idx])
```

Se toma datos del 70% de la información, luego se ordenan.

```
train = sorted(shuffled[split_idx:])
```

6500 es el punto de análisis al que se desea saber el día en que se va aproximar a ese dato. Encuentra las raíces de una función restando el punto en Y para obtener el corte en el eje X, desde el punto de inflexión la función va recorriendo los puntos en X desde x0 hasta la posición restada.

```
alcanzado_max = fsolve(fbt2 - 6500, x0=182) / (30)
```

```
alcanzado_max2 = fsolve(fbt2 - 6500, x0=182)
```

67 son los días del año donde comenzó la pandemia y C desfase de los nan en todo el archivo.

```
días=int(alcanzado_max2[0])+67+c

print(ftb2(186))
```

Se realiza el proceso inverso de según los días determinados cuantos casos positivos pueden ocurrir, en este caso fueron 6437 en comparación con los 6500 casos que se reportan para ese día.

```
6530 casos esperados en el mes 6.197164

6530 casos esperados en el dia 185.914907
6437.326828108402
```

Nota: para el proceso de las muertes se realizó el mismo procedimiento, por lo tanto no se explicara.

III. RESULTADOS

A partir de los datos analizados desde el 7 de marzo del año 2020 a los 67 días del presente año, hasta el 9 de septiembre del año 2020, se realizó una estimación con los casos del 18 de septiembre que fueron 6530 casos positivos esperados, mientras que nuestro modelo nos reportó 6530 el 15 de septiembre, es decir el día 185.64 en comparación con la fecha real el modelo tiene un desfase de tres (3) días aproximadamente, aunque se le adiciono el valor de desfase de los valores nulos (nan), además de tomar en cuenta que el día que empezó la pandemia fue 7 marzo 2020, a los 67 días del año. Nunca se va a tener un modelo exacto en la predicción de los datos, a continuación se pueden evidenciar la evolución de los distintos modelos.

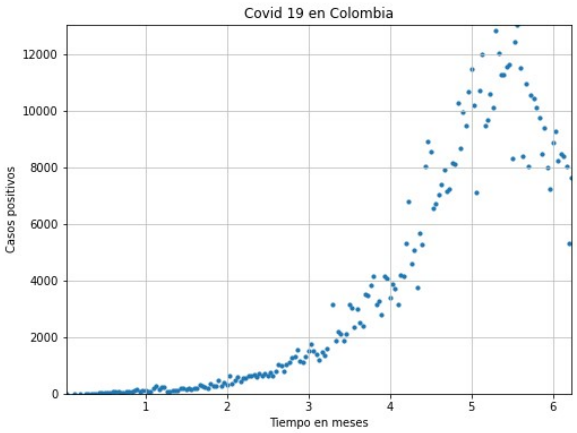

Figura 1. Casos positivos

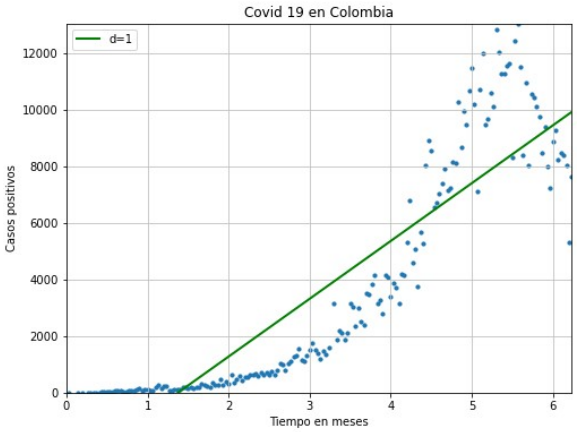

Figura 4. Modelo lineal

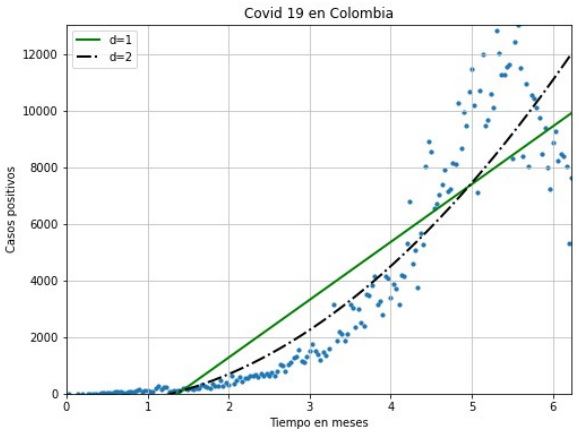

Figura 5. Modelo lineal y cuadrático

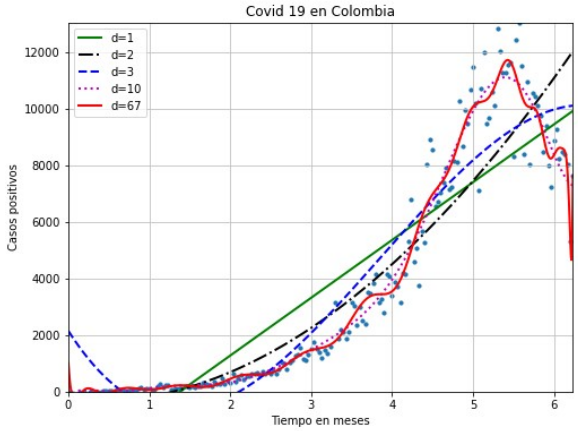

Figura 6. Modelo lineal, cuadrático, cubico.

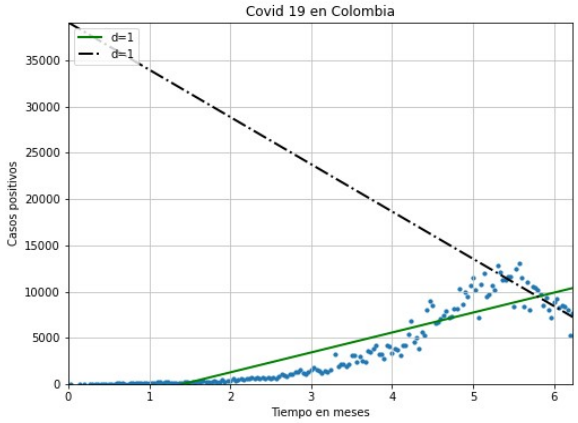

Figura 7. Punto de inflexión.

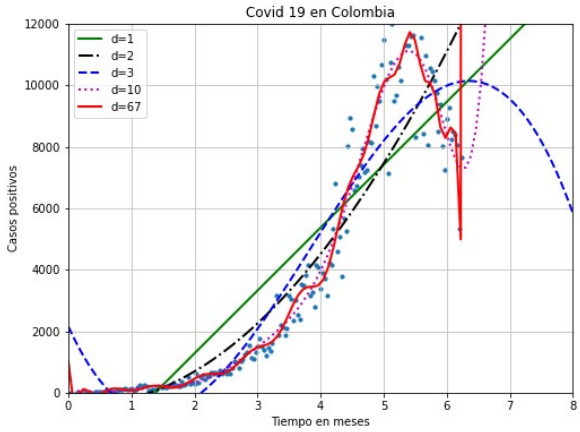

Figura 8. Divididas por el punto de inflexión.

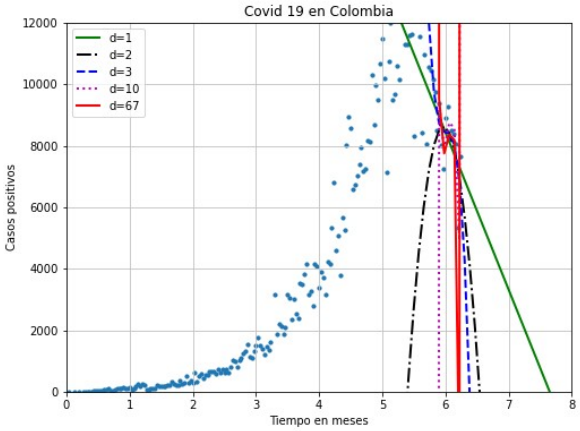

Figura 9. Entrenamiento de datos después del punto de inflexión.

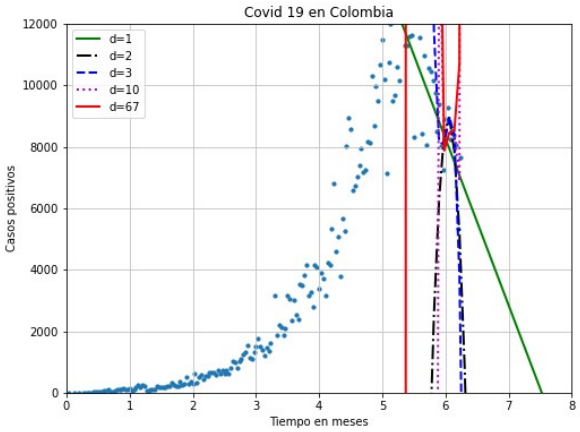

Figura 10. Entrenamiento de datos por muestra.

IV. CONCLUSIONES

A pesar de tomar las precauciones los modelos nunca va a ser exacto, ya que los datos nunca van a tener un comportamiento predecible.

Si se tienen en cuenta todas funciones mencionadas anteriormente se puede ver su comportamiento a gran escala, esto con el fin de predecir su valor a futuro.

Asumiendo que en cualquier recopilación de datos los errores o datos nulos estarán presentes, siempre se debe tener en cuenta realizar un filtrado para manipular correctamente los datos.

Con la gráfica se puede visualizar la dispersión de los datos separados por cada semana, para su futuro análisis ya sea en una empresa, una investigación o simplemente para toma de estadísticas.

Nos podemos acercar a la predicción mediante el método de polyfit, que tiene un comportamiento similar a la transformada rápida de Fourier, ya que con un grupo de puntos construye una función, la única diferencia es que la realiza sin funciones trigonométricas.

V. BIBLIOGRAFIA

1- https://www.tutorialspoint.com/matplotlib/matplotlib_grids.htm
2- https://www.tutorialspoint.com/matplotlib/matplotlib_jupyter_notebook.htm