Computación Blanda

Soft Computing

Autor: Angie Paola Villada Ortiz, Luz Adriana Quitumbo Santa
IS&C, Universidad Tecnológica de Pereira, Pereira Colombia
Correo-e:

paola.villada@utp.edu.co,Adriana.quitumbo@utp.edu.co

Resumen— Este documento se mostrará la implementación y el funcionamiento de las diferentes funciones de la librería Numpy, scipy y matplotlib entre otras. Este trabajo se enfoca en el análisis de datos en cuanto al número de contagios por Covid-19 que se registró en las localidades de Ciudad bolívar y de Kennedy estas pertenecen a la ciudad de Bogotá; durante los meses de julio, agosto y la primera semana del mes de septiembre; cabe resaltar que solo se tendrá en cuenta el rango de edad entre 50 a 60 años.

El objetivo principal de este análisis es determinar si por medio de datos suministrados permite enseñarle al modelo realizar predicciones a futuro, Unos de sus métodos de enseñanza es aplicar parámetros de referencia y funciones como shuffled que sirve para barajar los datos, fresolve para determinar la semana. Dichas predicciones son generadas por ajustes polinomiales, el cual determinara que grado (1,2,3,10 y 100) se ajusta a los puntos de dispersión. En este caso se quiere saber en qué semana se alcanzará el número de contagio de cuarenta y ocho personas.

Palabras clave— Numpy, Predicción, Contagio, Vector.

Abstract— This document shows the implementation and operation of the different functions of the Numpy, scipy and matplotlib library, among others. This work focuses on data analysis regarding the number of infections by Covid-19 that was registered in the towns of Ciudad Bolívar and Kennedy, these belong to the city of Bogotá; during the months of July, August and the first week of September; It should be noted that only the age range between 50 to 60 years will be taken into account.

The main objective of this analysis is to determine if through the supplied data it allows the model to be taught to make future predictions. One of its teaching methods is to apply reference parameters and functions such as shuffled that serves to shuffle the data, fresolve to determine the week. . These predictions are generated by polynomial adjustments, which will determine what degree (1,2,3,10 and 100) fits the dispersion points. In this case, you want to know in which week the contagion number of forty-eight people will be reached.

Key Word— Numpy, Prediction, Contagion, Vector.

I. INTRODUCCIÓN

Este articulo hace un análisis mediante el uso de ajustes polinomiales respecto al 33,5 % de los casos reportados en Colombia de Covid-19, cuyos casos se encuentran en la ciudad de Bogotá. La cual reporta 242.007 casos confirmados de los cuales 1.720 son casos nuevos; del total de casos 51% son mujeres .

Cuando se desea hacer un ajuste polinomial en un estudio relacionado al número de contagios reportados en Bogotá en el año 2020; este abre campo a la explicación de cómo se importar archivos de trabajos , se crean variables, se generan modelos y se grafican funciones gracias a la herramienta jupyter mediante el uso de numpy.

Numpy es un paquete de programa Python que significa "Python numérico". Es la biblioteca principal para la computación científica.

Algunos de los beneficios son: acceso más compacto, más rápido para leer y escribir artículos, más conveniente y eficiente.

Se debe tener en cuenta que el análisis detallado sobre ajustes polinomiales de grado uno, dos, tres, decimo y centésimo; se hizo respecto a los datos de la localidad de Kennedy y bolívar , la cual registra el 14% y 8,1% de casos reportados en la ciudad de Bogotá.

Para ello se hace un estudio respecto a las edades entre los rangos 50 y 60 años de sexo femenino y masculino respecto a los meses julio, agosto y septiembre ,donde se podrá evidenciar que cada grado polinomial proporciona un gráfico diferente donde se muestra ajustes que se hacen sobre los puntos establecidos ,los cuales son generados acorde a la base de datos suministrada , esta base contiene dos columnas (x) y (y) ;La primera columna hace referencia al número de días (77) suministrando así un total de once semanas y la segunda columna (y) hace referencia al número de contagios durante este periodo de tiempo(meses).

El objetivo de este estudio es que por medio del ajuste de grados polinomiales se pueda dar una predicción; la cual consiste en determinar en qué semana se presenta cuarenta y ocho contagios.

1.2 METODOLOGIA

Para empezar, se deben importar las librerías necesarias la primera de ellas es **os**; nos permite acceder a funcionalidades dependientes del Sistema Operativo. Sobre todo, aquellas que nos refieren información sobre el entorno de este y nos permiten manipular la estructura de directorios (para leer y escribir archivos). Este módulo proporciona una forma portátil de utilizar la funcionalidad dependiente del sistema operativo. Si solo se desea leer o escribir un archivo se daría la instrucción open(), si se van a manipular rutas, se utiliza él os.path, dado el caso que se quieran leer todas las líneas en todos los archivos en la línea de comando, se utilizaría la instrucción fileinput y para el manejo de archivos y directorios de alto nivel, consulte se utiliza shutil.

Luego se agregan los directorios de trabajo que son: DATA_DIR, CHART_DIR; El directorio DATA_DIR es el directorio para los datos y CHART_DIR es el directorio de los gráficos generados. Seguidamente se importa la librería Numpy, la cual se le pondrá el alias de np.

```
from utils import DATA_DIR, CHART_DIR
import numpy as np
```

Se utiliza seterr para eliminar las advertencias por el uso de funciones, también sirve para manejar los errores de punto flotante.

```
np.seterr(all='ignore')
```

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. Y Matplotlib es una librería para generar gráficas a partir de datos contenidos en listas, vectores, en el lenguaje de programación Python y en su extensión matemática Numpy.

```
import scipy as sp
import matplotlib.pyplot as plt
```

Se agregan los datos con los que se va a trabajar, con la función **np.genfromtxt** se puede cargar datos de un archivo de texto, con valores perdidos manejados como se especifica. Función mucho más sofisticada que tiene muchos parámetros para controlar su importación.

```
data = np.genfromtxt(os.path.join(DATA_DIR,
"casos_covid.txt"),
                     delimiter="\t")
```

Después de agregar los datos, se definen los colores y los tipos de líneas que llevara las gráficas; los colores que se definieron para este programa fueron: g = verde, k = negro, b = azul, m = magenta, r = rojo , y=amarillo.

```
colors = ['k', 'y', 'g', 'm', 'r'] linestyles = ['-', '-.', '--', ':', '-']
```

Se deben de crear los vectores que contendrá la información de las columnas de los datos que se utilizaran; **x** que serían el número de horas y corresponden para la primera columna de los datos y la segunda columna es **y** que es el número de solicitudes.

x = data[:, 0] y = **data[:, 1].**

Se utiliza la función isnan para observar los datos que son incorrectos y poder eliminarlos; la función isnan(vector) devuelve un vector en el cual los TRUE son valores de tipo nan, y los valores FALSE son valores diferentes a nan. Con esta información, este vector permite realizar transformaciones a otros vectores (o al mismo vector), y realizar operaciones como sumar el número de posiciones TRUE, con lo cual se calcula el total de valores tipo nan.

```
print("\n Número de entradas incorrectas:",
np.sum(np.isnan(y)))
```

Luego se deben de encontrar los puntos que son NAN y se deben de eliminar. Para lograr esto primero se crea un vector TRUE y FALSE se niegan estos valores (~), los valores que son FALSE se vuelven TRUE los cuales corresponden con aquellos valores que no son nan. Si el vector **x**, que contiene los valores en el eje **x**, se afectan a partir de dicho valores lógicos, se genera un nuevo vector en el que solos se toman aquellos que son TRUE. Por tanto, se crea un nuevo vector **x**, en el cual han desaparecido los correspondientes valores de **y** que son NAN.

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

Con la implementación de la función que define un **modelo (def plot_models)** nos permite obtener un ajuste de la columna (x) y columna (y) con base a un grado polinomial, en la lección número seis al ejecutar dicho programa podemos observar que en la parte inferior se despliegan una serie de imágenes que contienen unos gráficos en los cuales se puede evidenciar una serie de puntos dando referencia a los datos de la columna (x) y la columna (y) que tenemos en nuestra base de datos con él nombre"CasosCovid2020_Localidades_kennedy_bolivar_rangoEdad_50y60_mes_julio_Agosto_Sept.tsv" en la primera imagen podemos observar que se genera una serie de puntos los cuales no están definidos por un ajuste de grado polinomial pero podemos observar que en la segunda imagen esta se encuentra con una recta, esta recta se crea mediante la función def plot_models(x, y, models, fname, mx=None, ymax=None, xmin=None) ya que gracias a esta función se establece dentro de ella un ciclo donde podemos trazar líneas que corresponden bien sea a un grado polinomial ,el cual hace referencia una recta y en la tercera grafica se implementa un grado dos dibujando así una curva, estos grados polinomiales proporcionan dentro de la gráfica un ajuste de puntos y por consiguiente se puede observar que no solo se implementa uno o dos grados polinomiales si no que se generan muchos más.

La función plt.figure(num=None, figsize=(10, 6)) es la que crea o activa las imágenes existentes al momento de ejecutar el programa proporcionando así, un numero identificador ,el cual determina el ancho y el largo de la figura en este caso sus

valores son diez y seis. La función plt.clf() se encarga de borrar el espacio de la figura.

El método encargado de dibujar los datos de dispersión de la columna (x) y la columna (y) es plt.scatter(x, y, s=10); para establecer la dimensión de los puntos es decir su grosor se hace mediante la variable (s) la cual está definida por un tamaño de diez.

El diseño de la figura se establece por plt.title("Casos de covid durante los meses julio/agosto en \n las localidades Kennedy y Bolívar\n"") donde plt.title es la función que es proporciona el titulo superior de los gráficos y por consiguiente para generar los títulos bases del eje (x); se implementa la función plt.xlabel("Tiempo") y para el titulo lateral del eje (Y); es generado por el método plt.ylabel("\n Numero Contagios\n Rango de 50 a 60 años\n").

Para el estabelecimiento de las ubicaciones de las marcas y las etiquetas del eje (x) se establece mediante el método:

```
plt.xticks(
      [w * 7 for w in range(20)],
      ['sem %i' % w for w in range(20)])
```

El primer corchete [w * 7 for w in range(20)], hace referencia a las marcas en (x) definiendo con el número siete los días de una semana dando como resultado un total de once semanas ;este corchete funciona acorde al tamaño del vector x es decir hasta que se complete el total de días para este caso setenta y siete días. Además de esto calcula los valores de w ya que indica los valores de un poco más de 2 meses para este informe julio, agosto y sept del periodo 2020.

En el segundo corchete ['semana %i' % w for w in range(20)]),se utilizan los valores del primero ítem para escribir las etiquetas basadas de los valores de w.

Si se desea evaluar el tipo de modelo recibido se determina mediante un if models gracias a este método nos damos cuenta si se envía un modelo o si no se dibuja ninguna línea de ajuste.

En caso de que no se defina ningún valor para mx, este será calculado por la función mx = np.linspace(0, x[-1], 80)) se debe tener en cuenta que esta función se encarga de devolver en un intervalo los números espaciados uniformemente.

Para establecer un modelo y un estilo se utiliza una paleta de colores la cual está definida por la siguiente función:

```
for model, style, color in zip(models, linestyles, colors):
        # print "Modelo:",model
        # print "Coeffs:",model.coeffs
        plt.plot(mx, model(mx), linestyle=style, linewidth=3,
c=color)
```

Si deseamos conocer el funcionamiento del for primero explicaremos que es o para que se utiliza el:

For:

Es un bucle que repite el bloque de instrucciones un número prederminado de veces.

El bloque de instrucciones que se repite se suele llamar cuerpo del bucle y cada repetición se suele llamar iteración.

zip():
Devuelve un iterador de tuplas basado en los objetos iterables se debe tener en cuenta:

- Si no pasamos ningún parámetro, zip()devuelve un iterador vacío.
- Si se pasa un único iterable, zip()devuelve un iterador de tuplas con cada tupla que tiene solo un elemento.
- Si se pasan varios iterables, zip()devuelve un iterador de tuplas y cada tupla tiene elementos de todos los iterables.

Linestyles:
Los estilos de línea simples se pueden definir utilizando las cadenas "sólido", "punteado", "discontinuo" o "dashdot".

Se puede lograr un control más refinado proporcionando una tupla de guiones.

En este for lo que se hace es recorrer todos los modelos y estilos y se le asigna los colores a las listas que están comprimidas con la función zip, esta función lo que hace es que me comprime tres listas en una sola y con el for lo que se realiza es recorrerlas. Con esto se obvia hacer un bucle for para cada uno de los arreglos.

Legend es decir la función plt.legend(["d=%i" % m.order for m in models], loc="upper left") crea una leyenda con etiquetas descriptivas para cada serie de datos trazada.

Para las etiquetas, la leyenda utiliza el texto de las propiedades DisplayName de la serie de datos. Si la propiedad DisplayName está vacía, la leyenda utiliza una etiqueta con la forma 'dataN'.

La leyenda se actualiza automáticamente al agregar o eliminar series de datos de los ejes.

Este comando crea una leyenda para los ejes actuales o el gráfico devuelto. Si los ejes actuales están vacíos, entonces la leyenda está vacía. Si los ejes no existen, este comando los crea.

Por consiguiente, se crea un método el cual hace un ajuste de escala automático ,si se activa un ajuste de escala automático en un eje o en los ejes se realiza un ajuste con la función:

```
plt.autoscale(tight=True)
   plt.ylim(ymin=0)
   if ymax:
      plt.ylim(ymax=ymax)
   if xmin:
      plt.xlim(xmin=xmin)
   plt.grid(True, linestyle='-', color='0.75')
   plt.savefig(fname)
```

Donde plt.ylim(ymin=0) Obtiene o establece los límites y de los ejes actuales.

```
if ymax:
  plt.ylim(ymax=ymax)
```

Establece los límites de vista del eje (y).

```
if xmin:
     plt.xlim(xmin=xmin)
```

Obtiene o establece los límites x de los ejes actuales.

plt.grid(True, linestyle='-', color='0.75') Configura las líneas de la cuadrícula.

plt.savefig(fname) Guarda la figura actual.

plot_models(x,y,None,os.path.join(CHART_DIR,"1400_01_01.png"))crea gráficos a partir de modelos de regresión, ya sean estimaciones (como los llamados gráficos de bosque o de puntos) o efectos marginales.

La siguiente celda ejecuta código que genera cuatro modelos diferentes, llamados f1, f2, f3 , f10 y f100.

Explicaremos los nombres a continuación, la f significa función.

Cada uno de estos modelos es un tipo de función matemática ligeramente diferente.

A continuación, utilizaremos nuestra función de trazado para mostrar imágenes de lo que hacen los diferentes modelos.

En esta función se ajusta el polinomio de grado uno:

```
fp1, res1, rank1, sv1, rcond1 = np.polyfit(x, y, 1, full=True)
print("Parámetros del modelo fp1: %s" % fp1)
print("Error del modelo fp1:", res1)
f1 = sp.poly1d(fp1)
```

En esta función se ajusta el polinomio de grado dos:

```
fp2, res2, rank2, sv2, rcond2 = np.polyfit(x, y, 2, full=True)
print("Parámetros del modelo fp2: %s" % fp2)
print("Error del modelo fp2:", res2)
f2 = sp.poly1d(fp2)
```

ajuste de un polinomio de tercer(f3),decimo(f10) y centésimo(f100) orden:

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

Luego de haber hecho el ajuste de los polinomios grado uno, dos, tres, decimo y centésimo se procede a graficas los modelos con la función plot_models la cual se encarga de crear graficas a partir de modelos de regresión ya sean estimaciones (como los llamados gráficos de bosque o de puntos) o efectos marginales.

Se agrega los modelos de las imágenes (f1,f2,f3,f10 y f100) con las siguientes instrucciones:

```
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"))
```

Un modelo para ser ajustado y dibujado acorde al conocimiento del punto de inflexión opera de la siguiente forma:

```
inflexión = 5.3 * 7

xa = x[:int(inflexion)]
ya = y[:int(inflexion)]
xb = x[int(inflexion):]
yb = y[int(inflexion):]
```

Primero se saca los puntos de inflexion al modelo A, en sus respectivos ejes y por consiguiente se saca los puntos de inflexion al modelo B, de acuerdo con sus ejes.

Luego de esto se grafican dos líneas rectas que se ajustan a dos conjuntos de datos tales como:

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

Estos ajusten se evidencia al momento de ejecutar la imagen número cinco dado que la función:

```
plot_models(x, y, [fa, fb], os.path.join(CHART_DIR, "1400_01_05.png"))
```

Representa los datos de la columna (x),(y) con sus respectivas estimaciones.

Como se necesita evaluar si el modelo está bien entonces se reúnen las diferencias entre lo que predice el modelo para una día determinado y lo que realmente sucedió a ese día. Cuanto mayor sea esta puntuación, peor será el modelo. Y esto se logra con las siguientes instrucciones: donde definimos a error el cual contendrá **f** que es la función; **x**,**y** que son las columnas de datos que se están analizando.

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

Seguido de la anterior función se deben de mostrar los errores del conjunto de los datos y se logra con la instrucción:

```
print("Errores para el conjunto completo de datos:")
for f in [f1, f2, f3, f10, f100]:
   print("Error d=%i: %f" % (f.order, error(f, x, y)))
```

Se debe de mostrar los errores para **f1**, **f2**,**f3**, **f10**, **f100,** pero solo cuando se haya pasado el punto de inflexion.

```
print("Errores solamente después del punto de inflexión")
for f in [f1, f2, f3, f10, f100]:
   print("Error d=%i: %f" % (f.order, error(f, xb, yb)))
```

ahora se mostrará solo los errores de inflexión de **fa**, **xa**, **ya**, **fb**, **xb** y de **yb**.

```
print("Error de inflexión=%f" % (error(fa, xa, ya) + error(fb, xb, yb)))
```

luego de evidenciar los tipos de errores, se debe crear los gráficos a partir de modelos de regresión, ya sean estimaciones (como los llamados gráficos de bosque o de puntos) o efectos marginales. **os.path** se puede usar para analizar cadenas que representan nombres de archivo en sus partes componentes. Estas funciones operan únicamente en las cadenas.

```
plot_models(
   x, y, [f1, f2, f3, f10, f100],
   os.path.join(CHART_DIR, "1400_01_06.png"),
```

Se utiliza la función linspace, esto sirve para generar un array Numpy formado por n números equiespaciados entre dos dados uniformemente durante un intervalo especificado. Devuelve el número de muestras espaciadas uniformemente, calculadas sobre el intervalo (inicio, parada). Su sintaxis es: np.linspace(valor-inicial, valor-final, número de valores).

```
mx=np.linspace(0 * 7, 11 *7, 80), ymax=0, xmin=0 *7)
```

el número **(7);** está indicando los días que tiene la semana, dado que para este caso se tiene un total de 77 días que sería igual a **11** semanas.

Se deben se construir funciones para los modelos de orden superior que serían (órdenes 2, 3, 10 y 100). Se les asignan a las variables fb2,fb3,fb10,fb100 un ajuste de grado polinomial donde a la función polyfit se le manda con los puntos x, y de la matriz y también recibe el grado polinomial. en las siguientes líneas de código.

```
print("Entrenamiento de datos únicamente después del punto de inflexión")

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))
```

sp.poly1d es una función de clase polinomial unidimensional. **(np.polyfit())** se encarga de ajustar 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 además este método de clase se recomienda para código nuevo ya que es más estable numéricamente.

En el siguiente for se realiza es un análisis de cada uno de los vectores que se devuelven con los ajustes polinomiales y con el análisis de la reducción de errores, de cada uno de los polinomios. for f in [fb1, fb2, fb3, fb10, fb100]:

Luego se muestra en pantalla cada uno de los errores encontrados en los en cada vector; los errores se muestran en el orden que fueron hallados.

```
   print("Error d=%i: %f" % (f.order, error(f, xb, yb)))
```

Se debe de volver a graficar, estos gráficos son después de haber alcanzado el punto de inflexión.

```
plot_models(
   x, y, [fb1, fb2, fb3, fb10, fb100],
   os.path.join(CHART_DIR, "1400_01_07.png"),
   mx=np.linspace(0 * 7, 11 * 7, 100),
   ymax=0, xmin=0 * 7)
```

Esta parte es muy importante dado a que se separaran los datos de prueba que se van a entrenar; además se desea evaluar el rendimiento de predicción de un modelo, se debe probar con algunos datos que nunca ha visto. Entonces se le pasa un conjunto de puntos para estimar el modelo: llamamos a este conjunto de puntos datos de entrenamiento . Pero no dejamos que el modelo vea todos nuestros datos. Tenemos algunos para la prueba, y ese conjunto se llama datos de prueba .Esta es una de las ideas clave del paradigma del aprendizaje automático. Entrenamiento y prueba separados; no permita que el modelo en entrenamiento tenga acceso a información sobre los datos de prueba mientras se está entrenando.

frac = 0.3
Esta línea indica que solo se van a utilizar la tercera parte de los datos.

split_idx = int(frac * len(xb))
Aquí se saca el muestreo o el número de elementos que se va a utilizar, entonces se multiplica la fracción **0.3** por la longitud que sería de **40**, solo se tiene en cuanta la parte entera.

```
print("\n\nxb = ", xb, "\n\n")
print("\n\nlen(xb) = ", len(xb), "\n\n")
print("\n\nsplit_idx = ", split_idx, "\n\n")
```

Las anteriores instrucciones sirven para imprimir en pantalla los datos que tiene xb, la longitud y el resultado del muestreo.

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

Permutación aleatoriamente una secuencia o devolver un rango permutado. Si **x** es una matriz multidimensional, solo se baraja a lo largo de su primer índice.

```
print("\n\nrange(len(xb)) = ", range(len(xb)), "\n\n")
```

Para mostrar el rango de xb, que queda convertido en un vector el cual tiene una fila y 40 columnas.

```
print("\n\nlist(range(len(xb))) = ", list(range(len(xb))), "\n\n")
```

Para convertir ese rango de 40, en una lista.

```
print("\n\nshuffled=sp.random.permutation(list(range(len(xb))
)) = ", sp.random.permutation( list( range( len(xb) ) ) ), "\n\n")
```

Sirve para ver esa lista barajada de forma aleatoria con la función shuffled.

```
print("\n\nshuffledordenado=sp.random.permutation(list(range
(len(xb)))) = ",sorted( sp.random.permutation( list( range(
len(xb) ) ) ) ), "\n\n")
```

Se baraja la lista, pero de una forma más ordenada.

```
print("\n\nshuffled                                    =
sp.random.permutation(list(range(len(xb)))) = ",
    sp.random.permutation( list( range( len(xb) ) ) ), "\n\n")
```

En esta parte se vuelve y se baraja la lista.

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

Se toma el vector barajado de índices(lista) y seguidamente se toman los últimos 12 de esa lista de forma ordenada.

```
print("\n\ntest = sorted(shuffled[:split_idx]) = ",
    sorted(shuffled[:split_idx]), "\n\n")
```

Se verifica la anterior instrucción.

```
train = sorted(shuffled[split_idx:])
print("\n\n train = sorted(shuffled[split_idx:]) : ", train, "\n\n")
```

Train sirve para mostrar el vector barajado de índices(lista) y seguidamente se toman los primeros 12 de esa lista de forma ordenada.

```
print("\n\nxb[train] = ", xb[train], "\n\n")
```

Para mostrar los primeros 12 índices, pero de xb.

Se vuelve y se realiza una ajuste polinomial con los valores que se guardaron en las variables test, train y se le da nuevamente el grado de ajuste polinomial. y se crean los modelos en base a los datos que fueron guardados de manera aleatoria en las listas anteriormente mencionadas.

```
fbt1 = sp.poly1d(np.polyfit(xb[train], yb[train], 1))
fbt2 = sp.poly1d(np.polyfit(xb[train], yb[train], 2))
```

Se toman todos los xb y las yb y usando un polinomio de segundo grado, se debe encontrar una función tal que el error sea mínimo y después se debe construir la ecuación matemática y esta seria fbt2; esta función optimiza los valores.

```
print("fbt2(x)= \n%s" % fbt2)
print("fbt2(x)-48= \n%s" % (fbt2-48))
```

se imprime el modelo polinomial de grado 2 que se guardó en la variable fbt2 y se resta 48 al valor independiente del polinomio que nos dio en **fbt2.**

```
fbt3 = sp.poly1d(np.polyfit(xb[train], yb[train], 3))
fbt10 = sp.poly1d(np.polyfit(xb[train], yb[train], 10))
fbt100 = sp.poly1d(np.polyfit(xb[train], yb[train], 100))
```

Se vuelve a evaluar el modelo donde se evidenciarán los errores después del punto de inflexión.

```
print("Prueba de error para después del punto de inflexión")
```

se realiza una iteración para determinar cuáles son los errores que se pueden presentar en los modelos después de hacer el ajuste y analizarlos en la nueva lista de los valores aleatorios que se guardaron en train.

```
for f in [fbt1, fbt2, fbt3, fbt10, fbt100]:
```

se imprimen los errores en orden, analizándolos en cada modelo en un punto de la lista. que se guardó y de manera organizada en test.

```
print("Error d=%i: %f" % (f.order, error(f, xb[test], yb[test])))
```

Nuevamente se crea una imagen donde están las funciones dibujadas que el modelo debió aprender.

```
plot_models(
   x, y, [fbt1, fbt2, fbt3, fbt10, fbt100],
   os.path.join(CHART_DIR, "1400_01_08.png"),
   mx=np.linspace(0 * 7, 11 * 7, 100), ymax=0, xmin=0 * 7)
```

Finalmente se debe Scipy.optimize import fsolve encuentra las raíces de una función. Devuelve las raíces de las ecuaciones (no lineales) definidas por una estimación inicial dada.

```
from scipy.optimize import fsolve
```

Se llama el método fsolve, este utiliza la ecuación cuadrática y devuelve estimado de cuando se alcancen las 48 contagios

```
print(fbt2)   print(fbt2 - 48)
```

Se muestra en pantalla a **fbt2** es la función matemática original y el resultado de fbt2-48; para observar que funciones dan.

```
alcanzado_max = fsolve(fbt2 - 48, x0=77) / (7)
```

fsolve devuelve el día en el que se va a alcanzar ese número de contagio y lo hace de la siguiente manera: primero se le resta **48** a la función **fbt2**; para saber cuál es la raíz ósea **y=0** y así de esta forma saber el **x** correspondiente. Pero se le debe dar una referencia al sistema en este caso sería **x0=77**. Como se desea saber en qué semana se presenta ese número, entonces se divide el resultado que da fsolve con 7 que es el número de días que tiene una semana. Todo esto queda en la variable alcanzado_max.

```
print("\n48 contagios esperados en el día 77 en la semana
%f" % alcanzado_max[0])
```

Finalmente se imprime en pantalla el resultado de la variable alcanzado_max.

1.3 RESULTADOS

Se imprimen los primeros 10 datos, además se observa que son el número de días (77) que corresponden a los meses de julio, agosto y septiembre; en la siguiente columna es el acumulado de los casos de infectados por covid-19 que se presentaron en las localidades de Ciudad Bolívar y Kennedy en el rango de edades de 50 a 60 años.

```
[[ 1. 34.]
 [ 2. 62.]
 [ 3. 41.]
 [ 4. 43.]
 [ 5. 36.]
 [ 6. 34.]
 [ 7. 84.]
 [ 8. 66.]
 [ 9. 52.]
 [10. 74.]]

(77, 2)
```

Al comprobar el número de valores incorrectos se evidenció que no hay ningún valor NAN.

Número de entradas incorrectas: 0

La base de datos suministrada para el análisis de ajustes polinomiales genera como principal grafica los puntos de dispe rsión de las localidades Kennedy y bolívar tales puntos determ inan el número de contagios que se presentan entre las edades 50 y 60 años tal y como lo muestra la siguiente imagen:

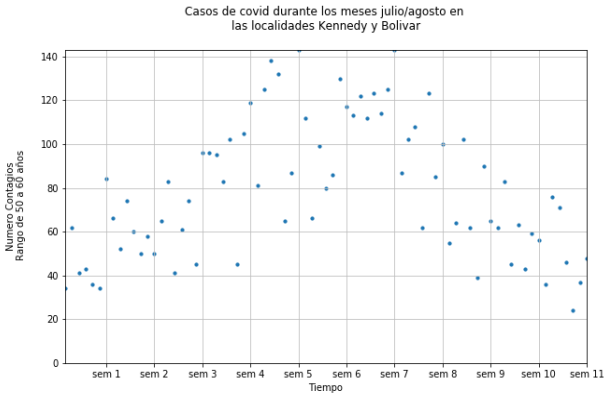

Figura 0.

Los ajustes de grados polinomiales mediante la función def plot_models(x, y, models, fname, mx=None, ymax=None, xmin=None) genera como resultado ajuste de puntos mediante diversos tipos de líneas tales como recta, curva etc.

La función def plot_models en su primer ciclo opera con un grado uno, generando una recta la cual representa un ajuste de puntos , esto se puede evidenciar en el siguiente gráfico:

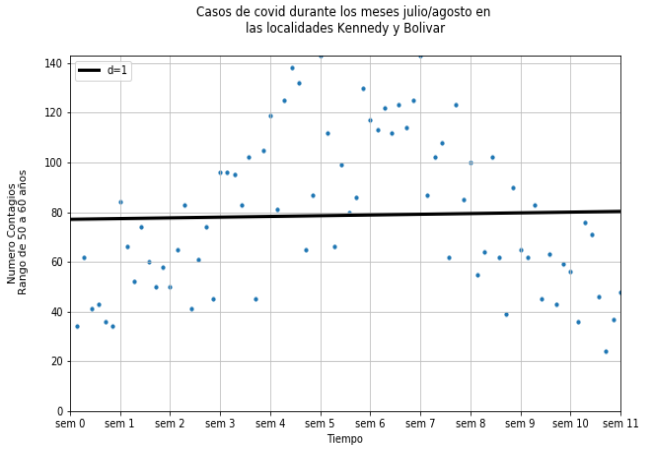

Figura 1.

Se debe tener en cuenta que este no es el único ajuste de puntos ,ya que se implementan dentro del programa diversos grados polinomiales tal como el grado dos que dibuja una gráfica:

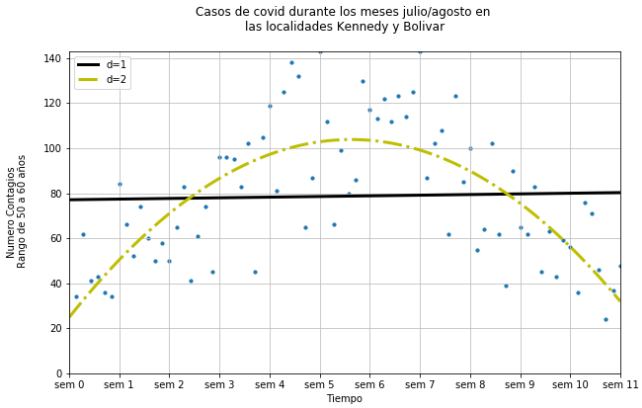

Figura 2.

Ajuste polinomial tercer grado:

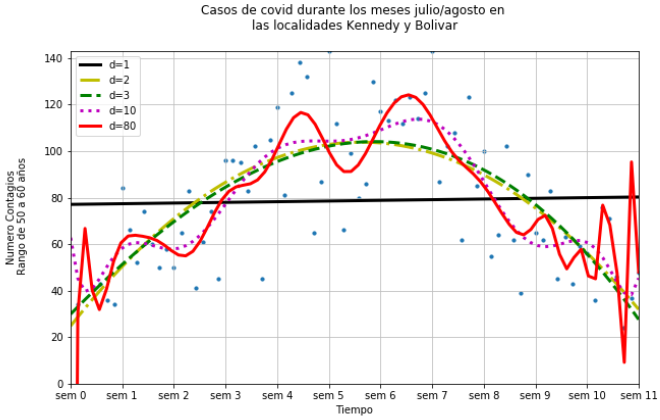

Figura 3.

En las anteriores figuras (0,1,2,3) se puede observar el tamaño del grafico mediante la función plt.figure(num=None, figsize=(10, 6))donde figsize es el ancho y el largo de la imagen y también se puede evidenciar el tamaño de los puntos dispersos dentro de la gráfica en este caso de diez mediante la función plt.scatter(x, y, s=10).

Además, se puede evidenciar los títulos en la parte superior, base y lateral, los cuales son generados por las siguientes funciones:

- plt.title("Casos de covid durante los meses julio/agosto en \n las localidades Kennedy y Bolívar \n ")
- plt.xlabel("Tiempo")
- plt.ylabel("\n Numero Contagios\n Rango de 50 a 60 años\n")

Mediante el método plt.xticks([w * 7 for w in range(20)], ['sem %i' % w for w in range(20)]); nos permite tener como resultado las etiquetas y las marcas del grafico tales como son mes 1,mes 2, y mes 3 ,los cuales hacen referencia a julio ,agosto y septiembre del año 2020 estos meses indican la cantidad de personas que son contagiadas por dicha enfermedad acorde a su edad(50 y 60).

El método autoscale es el que permite obtener cuadriculas dentro de las gráficas para este caso tienen un color de 0.75 y se encarga de guardarlas.

```
plt.autoscale(tight=True)
    plt.ylim(ymin=0)
    if ymax:
        plt.ylim(ymax=ymax)
    if xmin:
        plt.xlim(xmin=xmin)
    plt.grid(True, linestyle='-', color='0.75')
    plt.savefig(fname)
```

En las imágenes anteriores podemos evidenciar como se muestra la cuadricula en la figura(0,1,2,3).

Después de haber ajustado el modelo acorde podemos evidenciar en la siguiente figura que se dibujan dos líneas, las cuales representan en su unión el punto de inflexión:

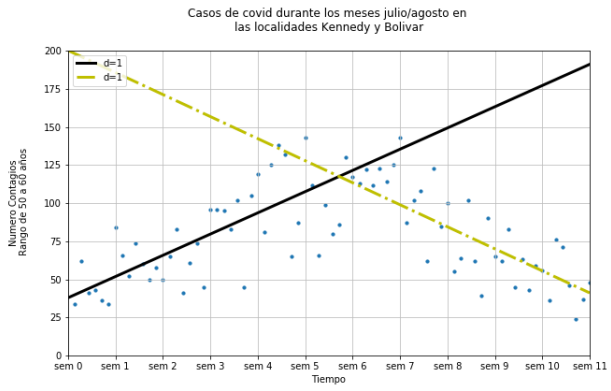

Figura 4.

El cual esta determinado en la semana cinco.

Como se necesita saber si el modelo está bien anteriormente se implementó la función error; la cual arroja los resultados de los diferentes tipos de errores.

- Errores para el conjunto completo de datos:

Error d=1: 72621.163678
Error d=2: 33803.983148
Error d=3: 33562.175171
Error d=10: 27888.599805
Error d=80: 22759.458740

Se muestran los errores para f1, f2,f3, f10, f100, pero solo después del punto de inflexion.

- Errores solamente después del punto de inflexión:

Error d=1: 39175.216842
Error d=2: 15438.144976
Error d=3: 15757.502484
Error d=10: 12147.150576
Error d=80: 8921.698568

- `Error de inflexión: 32569.977512`

Después de haber verificado los diferentes tipos de errores se grafica los datos. Para hacer proyecciones de casos covid-19 a futuro.:

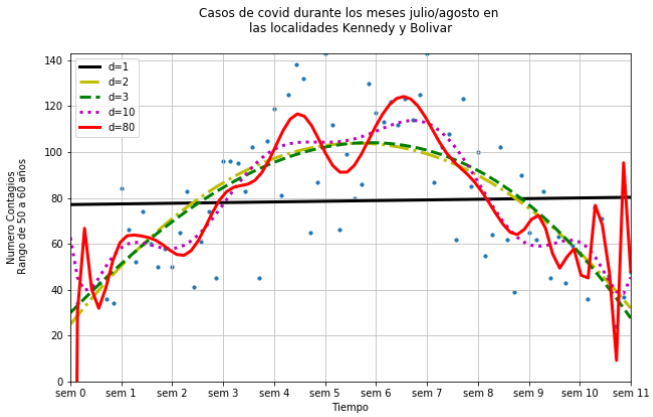

Figura 5.

En este apartado es muy importante dado a que se entrenara el modelo.

Se muestran en pantalla los errores, pero después de alcanzar el punto de inflexión.

Entrenamiento de datos únicamente después del punto de infle xión.

- Errores después del punto de inflexión:

Error d=1: 15319.527298
Error d=2: 14846.247436
Error d=3: 12591.700300
Error d=10: 9934.863049
Error d=81: 7734.988146

Grafica después del punto de inflexión:

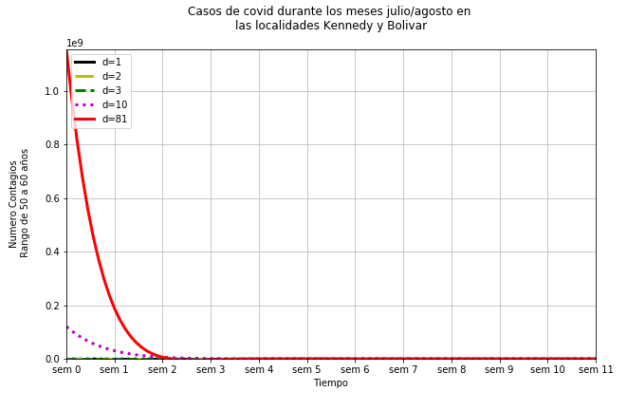

Figura 6.

- A continuación, se mostrará a xb, la longitud(40), el mues treo (12) y el rango de xb que va de (0,40).

xb = [38. 39. 40. 41. 42. 43. 44. 45. 46. 47. 48. 49. 50. 51. 52. 53. 54. 55.56. 57. 58. 59. 60. 61. 62. 63. 64. 65. 66. 67. 68. 69. 70. 71. 72. 73.74. 75. 76. 77.]

len(xb) = 40

split_idx = 12

range(len(xb)) = range(0, 40)

- Aquí se observa el resultado de haber trasformado el rang o len (0,40) en una lista.

list(range(len(xb))) = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30 , 31, 32, 33, 34, 35, 36, 37, 38, 39]

- Se ve el resultado después de haber utilizado shuffled de forma aleatoria en la lista que anteriormente se había crea do.

shuffled = sp.random.permutation(list(range(len(xb)))) = [30 8 19 27 23 7 24 39 15 25 4 2 10 34 20 28 18 13 35 38 3 37 21 33 26 6 31 0 32 9 22 5 12 14 11 16 36 1 17 29]

- Aquí se observa que se ordenaron los valores después de haber utilizado shuffled, pero de forma ordenada.

shuffled ordenado = sp.random.permutation(list(range(len(xb)))) = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 1 8, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39]

- Nuevamente se imprime la lista de forma barajada.

shuffled = sp.random.permutation(list(range(len(xb)))) = [33 14 12 31 17 1 19 0 6 27 11 22 15 8 35 2 30 9 20 18 32 4 2 9 25
7 36 37 16 10 34 24 28 21 5 39 13 3 38 23 26]

- Se ven los últimos 12 índices de la lista, pero de forma ordenada.

test = sorted(shuffled[:split_idx]) = [0, 1, 2, 5, 6, 10, 22, 26, 3 1, 32, 35, 37]

- Train tiene los primeros 12 índices de la lista, pero de for ma ordenada.

train = sorted(shuffled[split_idx:]) : [3, 4, 7, 8, 9, 11, 12, 13, 1 4, 15, 16, 17, 18, 19, 20, 21, 23, 24, 25, 27, 28, 29, 30, 33, 34, 36, 38, 39]

Se muestran los índices train, pero de xb.

xb[train] = [41. 42. 45. 46. 47. 49. 50. 51. 52. 53. 54. 55. 56. 5 7. 58. 59. 61. 62.
63. 65. 66. 67. 68. 71. 72. 74. 76. 77.]

- Se muestra en pantalla a fbt2 menos 48.

fbt2(x)=
2
0.02903 x - 5.947 x + 327.5
fbt2(x)-48=

- Cuando se le resta 48 se convierte en:
2
0.02903 x - 5.947 x + 279.5

- Prueba de error para después del punto de inflexión.

Error d=1: 7176.864569
Error d=2: 9631.962562
Error d=3: 6991.094253
Error d=10: 499949.163724
Error d=81: 27088634998.618504

Se grafican las funciones 1, 2, 3,10 y 81

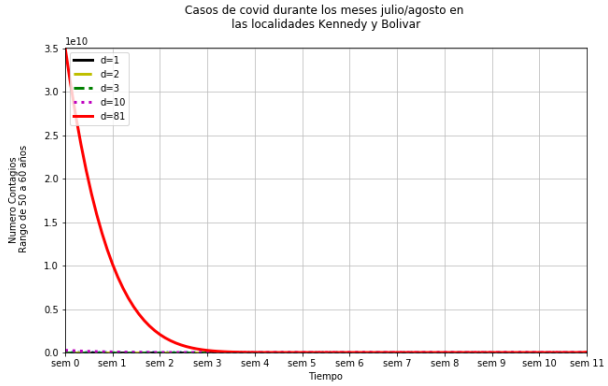

Figura 7.

Se muestra en pantalla a fbt2 que es la función original y el resultado de fbt2 menos 48.

```
          2
-0.02616 x + 0.8528 x + 124.1
          2
-0.02616 x + 0.8528 x + 76.14
```

Por último, se imprime en pantalla el resultado de la fsolve y d e la variable alcanzado_max en la cual se evidencia la semana 10 donde se presentará ese número de contagio.

48 contagios esperados en el día **77** en la semana 10.379898

1.4 CONCLUSIONES

- Se puede determinar que para las localidades Kennedy y bolívar el número de contagios entre los rangos de edad cincuenta y sesenta comenzó a descender a partir de la semana cinco.

- Se observa que la predicción para este estudio es casi exacta a los datos arrojados por la base de datos, esta predicción consiste en determinar en qué semana se presenta un contagio de cuarenta y ocho personas en las localidades Kennedy y bolívar; La base de datos muestra en la columna (x) que en el día setenta y siete se genera un contagio de cuarenta y ocho personas ,el cual hace referencia a la semana once.

- Finalmente se puede concluir que este modelo, aprendió y realizo una buena predicción de la semana en la que se presentaría el contagio 48 ; de forma adecuada con los datos que se le dio.

1.5 INFORMACION ACEDEMICA

- **Anggie Paola Villada Ortiz.** Nacida en 18/07/1998Guática Risaralda identificada con cedula de ciudadanía 1089721336. Me considero una persona responsable, dinámica y creativa, con facilidad de adaptación y capacidad de trabajar en equipo, en condiciones de alta presión para resolver problemas eficientemente y lograr las metas y objetivos trazados al nivel educativo. Estudios realizados Técnico en contabilización de operaciones comerciales y financieras. Colegio María Reina. Guática. (2015),Secretariado sistematizado. Instituto Panamericano. Guática. (2016),Curso de Inglés. Colombo Americano. Guática. (2014-2015),Curso de Salud Ocupacional. Sena. Guática. (2016),Curso de Excel. Sena. Guática. (2013).

- **Adriana Quitumbo**. Nací el 26 de Julio de 1993 en Marsella Risaralda, después de cinco años mi familia y yo nos mudamos a Pereira. Inicie mis estudios a los cinco años en el colegio manos unidas, en ese colegio curse hasta noveno grado. Luego entre al colegio la julita donde me gradué como técnico bachiller en ventas de productos y servicios en el año 2012. En el año 2014 me gradué en el Sena como técnico en asistencia administrativa, en el 2015 realice prácticas laborales en una empresa de telecomunicaciones por dos años, finalmente me inscribe en un programa de la alcaldía para una beca de estudio la cual finalmente me gane esto fue en el año 2017, para estudiar ingeniería en sistemas y telecomunicaciones en la universidad tecnológica de Pereira aún estoy realizando mis estudios actualmente estoy en el octavo semestre.