Análisis del Precio del Dólar con Machine Learning en Python

Dollar Price Analysis with Machine Learning in Python

Autor: Edisson Andres Montes, Alejandro Ríos
IS&C, Universidad Tecnológica de Pereira, Pereira, Colombia
Correo-e: edisson.montes@utp.edu.co, alejandro.rios@utp.edu.co

***Resumen*— Este documento presenta una implementación de Machine Learning en Python haciendo uso de las diferentes funcionalidades que nos provee este lenguaje de programación en el tratamiento de datos; esta implementación busca realizar predicciones en el transcurso del tiempo sobre el valor estimado del dólar en pesos colombianos, aquí se detalla cada línea de código utilizada de la forma más clara y sencilla posible que permita entender el funcionamiento de diferentes librerías de Python como numpy matplotlib, y scipy, así también dar a conocer de forma ilustrativa la forma en la que se comportan los diferentes datos en el tiempo y los diferentes modelos polinomiales que más se acercan a dicho comportamiento.**

***Palabras clave*— numpy, machine learning, matplotlib, modelos polinomiales, predicciones, programación, Python, scipy, tratamiento de datos.**

***Abstract*— This document presents an implementation of Machine Learning in Python making use of the different functionalities that this programming language provides us in data processing; This implementation seeks to make predictions over time about the estimated value of the dollar in Colombian pesos, here each line of code used is detailed in the clearest and easiest way possible to understand the operation of different Python libraries such as numpy matplotlib, and scipy, thus also showing in an illustrative way the way in which the different data behave in time and the different polynomial models that are closest to said behavior.**

***Key Word*— data processing, numpy, machine learning, matplotlib, predictions, programming, polynomial models, Python, scipy.**

I. INTRODUCCIÓN

Machine Learning puede definirse como un método analítico que permite que un sistema, por sí mismo sin intervención humana y en forma automatizada, aprenda a descubrir patrones, tendencias y relaciones en los datos, y gracias a dicho conocimiento, en cada interacción con información nueva se ofrecen mejores perspectivas. Esta competencia inherente para aprender de los datos, que sitúa a Machine Learning como una expresión de la Inteligencia Artificial, hoy todavía puede asombrar a algunos individuos; sin embargo, es una función analítica que ya determina múltiples aspectos de nuestra vida.
Python es un lenguaje de programación interpretado que busca desarrollar una sintaxis que priorice la legibilidad del código. Este lenguaje de programación es conocido como multiparadigma ya que soporta diferentes orientaciones. En Python podrás orientar el código a objetos, a programación imperativa y funcional. El machine learning en Python es la forma más popular de desarrollar estos modelos. Desde su lanzamiento en 1991 el lenguaje de programación Python ha tenido un crecimiento asombroso, convirtiéndose en uno de los preferidos por los programadores y con el auge de la inteligencia artificial tomo el liderato al ser el lenguaje de programación más utilizado para el desarrollo de este tipo de proyectos.

Para la implementación de estas funcionalidades se utilizará como base de datos, los valores diarios del dólar en pesos colombianos desde el año 2000 hasta el presente año 2020,
Se buscará a partir de esto realizar una predicción del valor del dólar en los próximos años.

II. IMPLEMENTACION EN CODIGO

Como primera medida se importa la librería del Sistema Operativo, seguido a esto importamos de la librería **utils** el directorio de datos **DATA_DIR** y el directorio de gráficos **CHART_DIR**, estos directorios nos permitirán almacenar los archivos de datos y los archivos generados por el programa para su posterior visualización
.
También se debe importar la librería **numpy** la cual es quien nos facilita la gestión de las listas en Python. Esta librería es muy útil a la hora de trabajar con machine learning debido a todas las funciones que trae consigo. Y para finalizar se importa la librería **scipy** y **matplotlib**. Estas librerías nos permitirán hacer uso de funciones de graficacion y tratamientos de datos para nuestras predicciones.

Eliminaremos las advertencias por el uso de funciones que en el futuro cambiarán con la función **seterr** de **numpy**

```
import os
from utils import DATA_DIR, CHART_DIR
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt

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

Ahora debemos de cargar en una variable el archivo en el cual están guardados los datos estadísticos con los que vamos a trabajar, para estos utilizaremos la función **genfromtxt** de **numpy** que nos permite obtener los datos de un archivo por medio de un carácter delimitador en este caso el carácter tabulación, ya que los datos que utilizaremos se encuentran guardados en dos columnas divididas por un espacio de tabulación.

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

Ahora Establecemos el tipo de dato que tomara 'data' para poder realizar las correctas operaciones con los datos, esto lo logramos asi:

```
data = np.array(data, dtype=np.float64)
```

Siguiente a esto definimos una lista de colores que utilizaremos en las gráficas con el fin de identificarlas más fácil entre ellas, asi también unos estilos de líneas con las cuales se dibujara cada gráfica polinomial

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

Ahora debemos dividir los datos almacenados en dos vectores X y Y los cuales representan el valor del dólar y el día en que se tomó ese dato respectivamente, esto debido a la arquitectura que se tiene de extrapolar estos datos en relación del tiempo y el valor establecido del dólar.

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

Así mismo realizaremos un escaneo y filtrado de datos nan (valores no acordes al sistema) en los vectores de datos para asegurar que no haya errores en el proceso de análisis.

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

Teniendo ya listo todos los datos, debidamente organizados y filtrados en sus correspondientes vectores procederemos a realizar su debido análisis, para esto crearemos una función que nos permita de la manera mas optima posible graficar los datos en diferentes formas y con sus diferentes ajustes polinomiales para realizar un buen análisis de los mismos.

Así entonces definiremos una función **plot_models** la cual realizara una gráfica de acuerdo a:

- Los datos en x
- Los datos en y
- Una lista con los diferentes modelos polinomiales a graficar
- Un límite de graficacion para el eje X y Y

Con esta función lograremos también establecer un diseño para la gráfica con títulos, labels y leyendas que permitan visualizar de manera mas clara la información, así también podremos guardar el grafico generado en la carpeta CHAR_DIR en formato .png

La función plot_models es muy importante debido a que es quién nos crea la gráfica, esta función toma el vector X y el vector Y, y ubica un único punto en plano cartesiano.

Para construir esta función debemos tener en cuenta:

Establecer las variables de entradas anteriormente dichas para la función

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

Luego creamos una nueva figura, con un tamaño predeterminado (anchura, altura) con la ayuda de **plt.figure**, definimos el tamaño de los puntos a graficar con **plt.scatter** y los diferentes títulos en la figura.

```
plt.figure(num=None, figsize=(25, 10))
plt.clf()
plt.scatter(x, y, s=10)
plt.title("VALOR DEL DOLAR DESDE 2000 HASTA EL
2020")
plt.xlabel("TIEMPO")
plt.ylabel("VALOR DOLAR/DIA EN COP")
```

Definimos el rango del eje x permitiendo agrupar los datos registrados en días, en cuadriculas de años y asignamos cada uno de los textos según el año en el eje x, esto con un rango de 30 años, sin embargo, si no existen datos hasta dicho año esta solo lo hará hasta el último dato existente.

```
plt.xticks( [w * 365 for w in range(30)],
        ['Año 20%02i' % w for w in range(30)])
```

Ahora realizaremos la graficacion de los modelos, de acuerdo a la lista de color y estilo, para graficar de manera clara cada uno de los modelos ingresados en la función

```
if models:
      if mx is None:
            mx = np.linspace(0, x[-1], 1000)

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

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

Crearemos una leyenda en la parte superior izquierda para visualizar la dimensión de cada uno de los modelos graficados con su respectivo estilo y color

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

plt.autoscale(tight=True)
```

Seguidamente establecemos un límite en los ejes según el valor ingresado esto con el fin de ampliar el grafico y visualizar el comportamiento de modelos graficados a lo largo del tiempo, en cado de no ser recibido ningún valor el límite será nulo y solo serán graficados los modelos hasta el ultimo dato en el tiempo.

```
plt.ylim(ymin=0)
if ymax:
   plt.ylim(ymax=ymax)
if xmin:
   plt.xlim(xmin=xmin)
```

Establecemos una pequeña malla opaca que permite ver la delimitación de los datos según los agrupamos anteriormente por años con la ayuda de **plt.grid.**

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

Finalmente, guardaremos la figura generada según el nombre de archivo suministrado en el llamado de la función.

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

Como primera mirada a los datos haremos uso de la función **plot_models** anteriormente construida, solo usaremos los datos como elementos a graficar y establecemos el nombre que tomara el archivo al guardarlo, esto nos permite en primera instancia ver todos los datos en un plano a lo largo del tiempo.

```
plot_models(x, y, None,
os.path.join(CHART_DIR, "1400_01_01.png"))
```
(Figura 1)

Ahora para entrar en materia con respecto al entrenamiento de los datos para su posterior prediccion primero debemos de crear el primer ajuste polinomial con ayuda de la función **np.polyfit** la cual nos crea todos los datos a partir de los vectores iniciales y los deja listos para poder graficarlos más fácilmente; entonces con la función **sp.poly1d** construiremos la clase polinomial la cual gráficamente será una línea de grado polinomial uno (1).

```
fp1 = np.polyfit(x, y, 1, full=True)
f1 = sp.poly1d(fp1)
```

Después se hace exactamente el mismo proceso pero cambiando el grado polinomial, lo único que se debe cambiar es el dato de entrada deg en la función **np.polyfit** dependiendo de qué grado queramos. En este caso se crearon polinomios de grado uno (1), dos (2a), tres (3), diez (10) y cincuenta (50).

```
f2 = sp.poly1d(np.polyfit(x, y, 2))
f3 = sp.poly1d(np.polyfit(x, y, 3))
f10 = sp.poly1d(np.polyfit(x, y, 10))
f50 = sp.poly1d(np.polyfit(x, y, 50))
```

Cabe resaltar que a pesar de que a la función **np.polyfit** se asignó un grado de polinomio cincuenta (50) el ajuste máximo posible para esta cantidad de datos es de grado treinta y nueve (39), entonces la función a pesar de que recibe un grado que no es posible procesar debido a su gran magnitud, automáticamente nos ajusta ese grado y nos grafica el máximo posible.

Después empezamos a mirar más ajustes polinomiales y a compararlos. Se grafican el ajuste polinomial de grado uno (1) y dos (2) juntos sin límites de proyección. Y finalmente los graficamos todos simultáneamente y sin límites de proyección.

```
plot_models(x, y, [f1],
os.path.join(CHART_DIR, "1400_01_02.png"))
```
(Figura 2)

```
plot_models(x, y, [f1, f2],
os.path.join(CHART_DIR, "1400_01_03.png"))
```
(Figura 3)

```
plot_models(x, y, [f1, f2, f3, f10, f50],
os.path.join(CHART_DIR, "1400_01_04.png"))
```
(Figura 4)

Ahora pasamos a una parte muy importante y es crear un punto de inflexión, esto se hace con el fin de tener una mejor predicción debido a que un porcentaje de los datos al inicio puede que no nos brinde mucha información para hacer la predicción y por el contrario nos la podría afectar. En este caso ese punto de inflexión se determinó que debía estar en el año quince (15) ya que los datos anteriores son muy similares y no nos proporcionan información relevante para hacer la predicción.

```
inflexion =  15 * 365

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

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

Al tener este punto de inflexión lo que tenemos que hacer es crear unos nuevos ajustes polinomiales, pero tomando como punto de partida y finalización para la toma de datos ese punto de inflexión, es decir desde un conjunto de datos el inicio hasta el punto de inflexión y otro desde el punto de inflexión hasta el final, y haremos su respectiva graficacion

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

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

(Figura 5)

Seguido a esto graficaremos todos los ajustes polinomiales con todos los datos desde el inicio hasta el final, pero ahora si asignaremos un límite para así poder apreciar la proyección futura que nos ofrece cada uno de estos ajustes, específicamente los limites los ajustaremos hasta el año 2024.

```
plot_models(
    x, y, [f1, f2, f3, f10, f100],
    os.path.join(CHART_DIR,
"1400_01_06.png"),
    mx=np.linspace(0 * 365, 24 * 365, 100),
    ymax=5000, xmin=0 * 365)
```

(Figura 6)

Después haremos exactamente el mismo proceso solo que ya solo se tomarán los conjuntos de datos después del punto de inflexión, así que procederemos a construir los modelos polinomiales de igual manera, pero solo con los datos que se encuentran después del punto de inflexión esto con el fin de acercar la predicción un poco más al comportamiento de los datos más actuales; de igual manera realizaremos su respectiva graficacion para su análisis con una proyección igual a la anterior hasta el año 2024.

```
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, 50))

plot_models(
    x, y, [fb1, fb2, fb3, fb10, fb100],
    os.path.join(CHART_DIR,
"1400_01_07.png"),
    mx=np.linspace(0 * 365, 24 * 365, 100),
    ymax=5000, xmin=0 * 365)
```

(Figura 7)

Posteriormente tomamos aleatoriamente el 70% de datos de los conjuntos utilizados anteriormente, estos los estableceremos al azar con la ayuda de la función **sp.random.permutation** y **sorted()** la cual nos permite dejar la lista de datos aleatorios ordenada.

```
frac = 0.3
split_idx = int(frac * len(xb))

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

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

Posteriormente establecemos nuevamente unos modelos polinomiales de diferentes grados, basados únicamente en los datoa anteriormente calculados, esto con el fin de ver el comportamiento de las graficas solo con estos datos obtenidos.

```
fbt1 = sp.poly1d(np.polyfit(xb[train],
yb[train], 1))
fbt2 = sp.poly1d(np.polyfit(xb[train],
yb[train], 2))
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], 50))
```

Igualmente creamos unas nuevas gráficas con este conjunto de datos; esto se hace porque un 30% de esos datos se utilizan como prueba y el restante si nos son útiles para hacer las predicciones.

```
plot_models(
    x, y, [fbt1, fbt2, fbt3, fbt10, fbt100],
    os.path.join(CHART_DIR,
"1400_01_08.png"),
    mx=np.linspace(0 * 365, 24 * 365, 100),
    ymax=5000, xmin=0 * 365)
```

(Figura 8)

Gracias al análisis que se le hizo a cada gráfica se determinó que el ajuste polinomial de grado dos (2) fue la mejor opción a elegir para hacer la predicción por la simple razón de que los ajustes polinomiales de grado más alto no brindaban predicciones lógicas y los de menor grado no se ajustaban correctamente a los datos.

Por esto para finalizar y para poner en práctica todo ese planteamiento matemático anterior, utilizamos el ajuste polinomial de grado dos (2) para predecir cual será el año en el que el dólar alcanzará el valor de $4500. Para esto nos ayudaremos de la función **fsolve** de **scipy.optimize** la cual nos permite conocer el valor en **Y** que tomara un determinado modelo polinomial en un valor **X**, esta función toma como valores de entrada el modelo polinomial que queremos analizar

y un estimador inicial que permitirá establecer un límite de la posible respuesta; procederemos a importar dicha función así:

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

Haremos la primera predicción con todos los datos, porque a pesar de que muchos de estos nos pueden alterar la precisión de la predicción no debemos descartarlos porque al fin y al cabo son valores individuales correctos, así que utilizaremos el modelo f2 el cual anteriormente fue creado a partir de todos los datos existentes.

```
print("\nPrediccion realizada a partir de
totos los datos")

alcanzado_max2 = fsolve(f2 - 4500, x0=7575) /
(365)

print("VALOR DOLAR $4500 ESPERADO EN EL AÑO
%f" % alcanzado_max2[0])
```

La segunda predicción la realizaremos con los datos tomados después del punto de inflexión, esto con el fin de conocer de manera más precisa el comportamiento actual de los datos, para esto utilizaremos el modelo fbt2, el cual construimos con el 70% de los datos aleatorios encontrados después del punto de inflexión

```
print("\nPrediccion realizada solo según los
datos siguientes al 2015")
alcanzado_max = fsolve(fbt2 - 4500, x0=7575)
/ (365)

print("VALOR DOLAR $4500 ESPERADO EN EL AÑO
%f" % alcanzado_max[0])
```

Finalmente realizaremos un promedio de las dos anteriores con el objetivo de contrarrestar diferencia y dar más precisión a la predicción final.

```
print("\nPromedio de Predicciones")
print((alcanzado_max[0]+alcanzado_max2[0])/2)
```

III. RESULTADOS

A partir de los diferentes llamados a la función **plot.models()** logramos obtener los siguientes resultados gráficos cada uno especificado anteriormente.

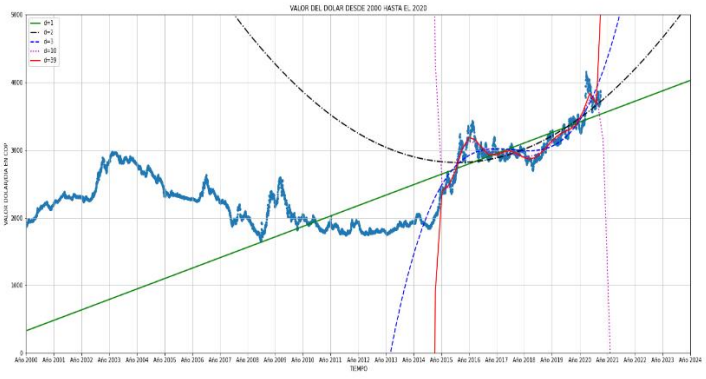

Figura 1

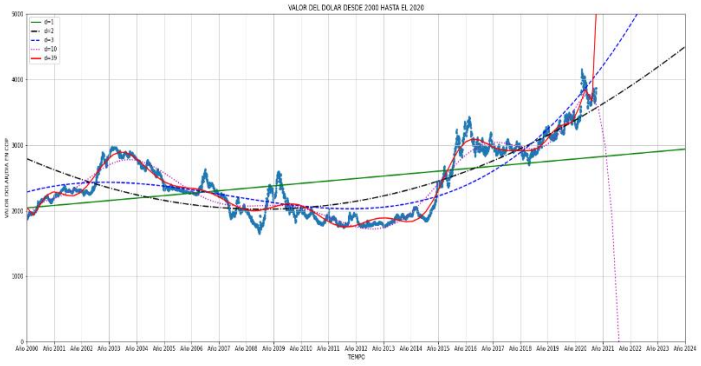

Figura 2

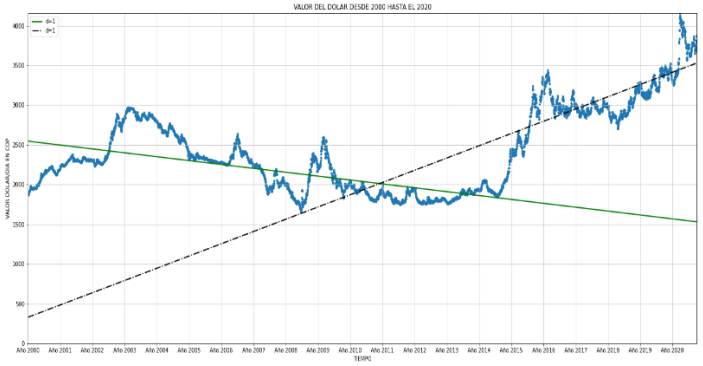

Figura 3

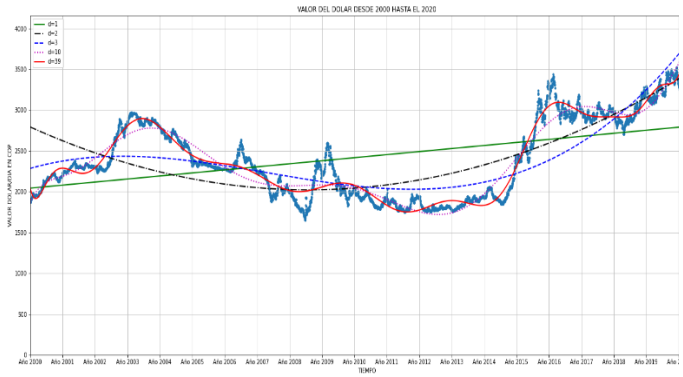

Figura 4

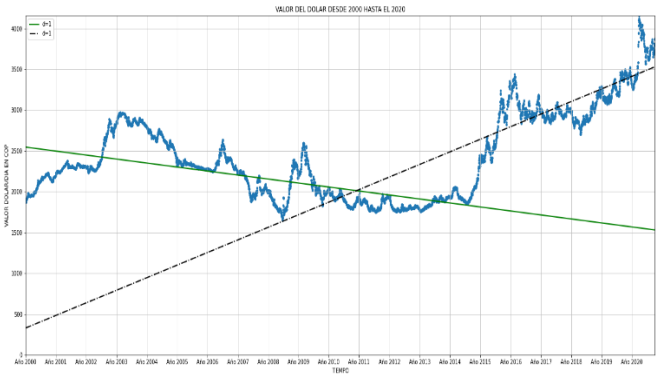

Figura 7

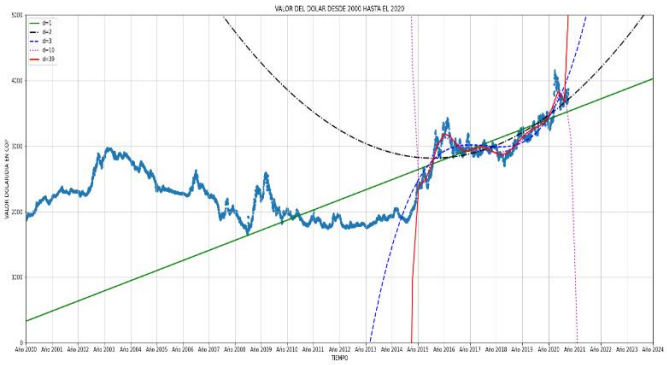

Figura 5

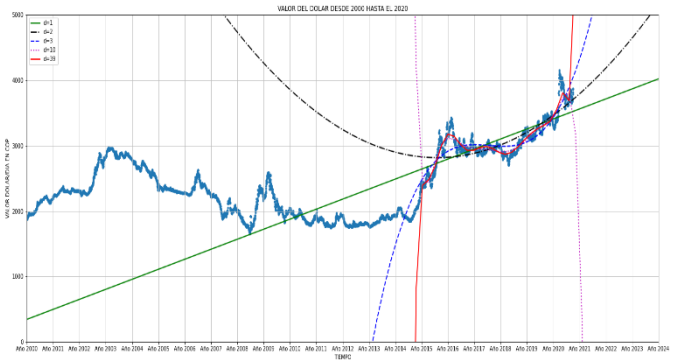

Figura 8

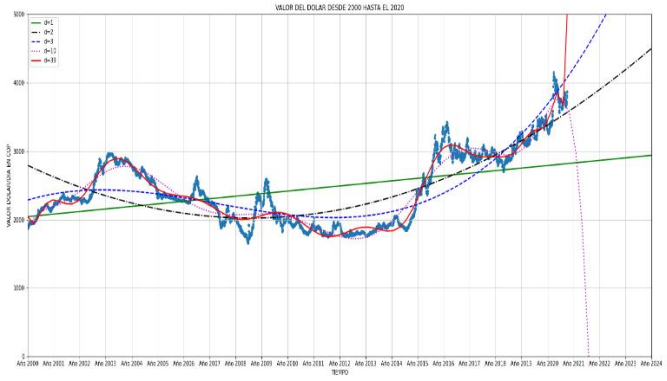

Figura 6

Como resultado final de la función **fsolve(),** la cual nos permitió realizar las diferentes proyección según el modelo dato obtuvimos lo siguiente:
La primera predicción nos proporciona como resultado que el dólar alcanzará un valor de $4500 en el año 23.99, es decir aproximadamente en los últimos meses del año 2023

La segunda predicción nos proporciona como resultado que el dólar alcanzará ese mismo valor en el año 22.65, es decir aproximadamente a mediados del año 2022.

Como resultado promedio obtuvimos un acercamiento del dólar a ese valor en el año 23.32, es decir aproximadamente a principios del año 2023

IV. CONCLUSIONES

- Gracias a los diferentes modelos podemos establecer unos rangos de predicción mas preciso, en nuestro caso, realizamos dos predicciones y un promedio de ellas lo cual nos deja ver una predicción mas acercada a las diferentes variantes de los datos.

- A partir de los cálculos realizados el dólar alcanzara un valor de $4500 COP entre mediados del año 2022 y finales del año 2023. Teniendo como predicción mas precisa los primeros meses del año 2023.

- El punto de inflexión es un punto muy crucial a la hora de tener una predicción lo más acertada posible debido a que este nos hace un filtro de datos que probablemente no nos estén aportando mucho y por el contrario lo que estén haciendo es afectar la predicción.

- La función plot_models es lo parte más importante del código, pero quién realmente hace la magia son las librerías que importamos porque nos facilitan todo el trabajo de graficado y sobre todo porque nos calculan los ajustes polinomiales automáticamente.

V. REFERENCIAS

Referencias en la Web:

[1]
https://www.sas.com/es_mx/whitepapers/local/machine-learning.html?gclid=CjwKCAjwlID8BRAFEiwAnUoK1WwWFV183vW4TsOw4rxQGkOrkoSjQu9CVyHHCJHT-Ug4kNYTPdYgQhoCNZsQAvD_BwE

[2]
https://www.grapheverywhere.com/machine-learning-en-python/