# Análisis de la cantidad de Monóxido de Carbono en la zona Noroeste de la Ciudad de México

Norma Angélica Márquez Sulca

427278
Universidad de Guanajuato - División de Ciencias e Ingenierías UG-CL

17 de junio de 2020

**Resumen**

Con datos obtenidos respecto a la calidad del aire en las distintas zonas de Ciudad de México, se determinó un área y gas específico a estudiar. Se predijó la cantidad del mismo en la zona seleccionada para años posteriores.

## I. INTRODUCCIÓN

Dentro de la página del Gobierno de la ciudad de México [1] se descargaron datos de "Índice de calidad del aire", fueron utilizados los de 2010 hasta 2018, se emplearon únicamente los datos del noroeste de la ciudad de México y del gas Monóxido de carbono y se desea predecir la contaminación con base en estos parámetros para 2019, 2020, 2021.

## II. DESARROLLO

El análisis se realizó en Python, por lo que, lo primero que se hizo fue importar las librerías necesarias para la manipulación de los datos.

```
import numpy as np
import matplotlib.pyplot as plt
import os
from scipy.optimize import minimize
import glob
import pandas as pd
import re
import unicodecsv as cs
import csv
import seaborn as sns
from seaborn import kdeplot
```

**Figura 1:** Importando las librerías.

Después de esto, se cargaron los archivos necesarios para realizar la predicción, en total son 9 archivos, contenidos en una carpeta nombrada "DatosHIGI2", uno por cada año a estudiar, y se creó una variable que los almacenara a todos.

```
file='/Users/Angie/Downloads/DatosHIGI2'
files=glob.glob("./DatosHIGI2/*.csv")
files
```

**Figura 2:** Ruta de los archivos.

Para estudiar el comportamiento de los datos, se filtraron únicamente las columnas "Fecha" y "Noroeste monóxido de carbono"de los archivos , dado que estas son las únicas que serán de nuestro interés, inmediatamente después del filtro, se uso la función "pd.to_datetime()", para que reconociera las fechas escritas en la columna de fechas y nos permitiese agruparlo en el intervalo necesario, ya que se nos proporcionaba la información por horas, por lo que cada día del año se repetía 24 veces.

```
#Seleccionamos las columnas necesarias
#y transformamos todas las fechas encontradas en los csv a datetime, para poder agruparlas

data1 = pd.read_csv (files[0], usecols= ['Fecha','Noroeste mon�xido de carbono'])
data1["Fecha"] = pd.to_datetime(data1["Fecha"], errors="coerce")
print (data1)

data2 = pd.read_csv (files[1], usecols= ['Fecha','Noroeste mon�xido de carbono'])
data2["Fecha"] = pd.to_datetime(data2["Fecha"], errors="coerce")

data3 = pd.read_csv (files[2], usecols= ['Fecha','Noroeste mon�xido de carbono'])
data3["Fecha"] = pd.to_datetime(data3["Fecha"], errors="coerce")

data4 = pd.read_csv (files[3], usecols= ['Fecha','Noroeste mon�xido de carbono'])
data4["Fecha"] = pd.to_datetime(data4["Fecha"], errors="coerce")

data5 = pd.read_csv (files[4], usecols= ['Fecha','Noroeste mon�xido de carbono'])
data5["Fecha"] = pd.to_datetime(data5["Fecha"], errors="coerce")

data6 = pd.read_csv (files[5], usecols= ['Fecha','Noroeste mon�xido de carbono'])
data6["Fecha"] = pd.to_datetime(data6["Fecha"], errors="coerce")

data7 = pd.read_csv (files[6], usecols= ['Fecha','Noroeste mon�xido de carbono'])
data7["Fecha"] = pd.to_datetime(data7["Fecha"], errors="coerce")

data8 = pd.read_csv (files[7], usecols= ['Fecha','Noroeste mon�xido de carbono'])
data8["Fecha"] = pd.to_datetime(data8["Fecha"], errors="coerce")

data9 = pd.read_csv (files[8], usecols= ['Fecha','Noroeste mon�xido de carbono'])
data9["Fecha"] = pd.to_datetime(data9["Fecha"], errors="coerce")
```

**Figura 3:** Lectura y filtro de los archivos.

Durante la elaboración de tablas, se presentó un sorting, causando que la agrupación de datos fuese incorrecta, ya que comenzaba brindando los días primero de cada mes, después los segundos y progresivamente, por tanto, si se deseaba obtener la suma de un mes en específico, esta resultaba incorrecta, para resolver este error se utlizó la función g̈roupbyc̈on "sort=Flase".

```
#Así evitariamos el sorting de las tablas pivote

table1 = data1.groupby(('Fecha'), sort=False).sum()
print (table1)
```

**Figura 4:** Código para evitar sorting en las tablas.

| Fecha | Noroeste mon�xido de carbono |
|---|---|
| 2010-01-01 | 246 |
| 2010-01-02 | 278 |
| 2010-01-03 | 253 |
| 2010-01-04 | 253 |
| 2010-01-05 | 283 |
| ... | ... |
| 2010-12-27 | 458 |
| 2010-12-28 | 491 |
| 2010-12-29 | 462 |
| 2010-12-30 | 456 |
| 2010-12-31 | 513 |

[365 rows x 1 columns]

**Figura 5:** Tablas con sorting.

```
            Noroeste mon🔲xido de carbono
Fecha
2010-01-01                          246
2010-02-01                          263
2010-03-01                          190
2010-04-01                          205
2010-05-01                          265
...                                 ...
2010-12-27                          458
2010-12-28                          491
2010-12-29                          462
2010-12-30                          456
2010-12-31                          513

[365 rows x 1 columns]
```

**Figura 6:** Tablas sin sorting.

Se decidió agrupar por año los valores, de esta manera, el sorting dejó de ser relevante. Esto se realizó a través de tablas pivote con la función "pd.pivot_tableÿ el comando: freq="y".

```
#Así agrupamos la suma de todo un año del Mónoxido de carbono data por data

table11 = pd.pivot_table(data1, index=pd.Grouper(key="Fecha", freq="Y", sort=False), columns= None, values='Noroes
print(table11)

table21 = pd.pivot_table(data2, index=pd.Grouper(key="Fecha", freq="Y", sort=False), columns= None, values='Noroes
print(table21)

table31 = pd.pivot_table(data3, index=pd.Grouper(key="Fecha", freq="Y", sort=False), columns= None, values='Noroes
print(table31)

table41 = pd.pivot_table(data4, index=pd.Grouper(key="Fecha", freq="Y", sort=False), columns= None, values='Noroes
print(table41)

table51 = pd.pivot_table(data5, index=pd.Grouper(key="Fecha", freq="Y", sort=False), columns= None, values='Noroes
print(table51)

table61 = pd.pivot_table(data6, index=pd.Grouper(key="Fecha", freq="Y", sort=False), columns= None, values='Noroes
print(table61)

table71 = pd.pivot_table(data7, index=pd.Grouper(key="Fecha", freq="Y", sort=False), columns= None, values='Noroes
print(table71)

table81 = pd.pivot_table(data8, index=pd.Grouper(key="Fecha", freq="Y", sort=False), columns= None, values='Noroes
print(table81)

table91 = pd.pivot_table(data9, index=pd.Grouper(key="Fecha", freq="Y", sort=False), columns= None, values='Noroes
print(table91)
```

**Figura 7:** Tablas pivote que regresan la suma anual de Mónoxido.

Se elaboró un ciclo que te le asignará a cada año, su archivo correspondiente. Primero, se agruparon los valores de cada fecha, en este caso de cada año, y se acomodaron cronológicamente, posteriormente, se le destino a cada una el archivo que coincidera con la fecha. Además se creó un arreglo que contiene cada uno de lo años utilizados.

```
#Creamos un arreglo con los años que utilizaremos en la tabla

date=[]
date_=[]
files_=[]
for i,file in enumerate(files):
    date.append(re.findall(r'\d+',file)[1])
Fechas=np.asarray(date)

temp=sorted(range(len(date)), key=date.__getitem__)

for i in temp:
    date_.append(date[i])
    print(date[i],files[i])
    files_.append(files[i])
files_

Fechas
```

**Figura 8:** Ciclo para elaborar el arreglo de Fechas.

Se estructuró otro ciclo, que imprimera cada uno de los años y su suma correspondiente de Mónoxido de carbono, a su vez se guardó en un arregló llamado .ªrreglo_infoçada uno de los pares de datos que generó el ciclo recién mencionado.

```
#De aquí obtenemos la suma anual de monóxido de carbono en la zona noroeste de cdmx
arreglo_info=[]
for i,file in enumerate(files):
    data=pd.read_csv(file)
    data["Fecha"] = pd.to_datetime(data["Fecha"], errors="coerce")
    monoxido=pd.pivot_table(data, index=pd.Grouper(key="Fecha", freq="Y"), columns= None, values='Noroeste mon\ufffdxi
    arreglo_info.append(monoxido)

arreglo_info
#  monoxido=pd.pivot_table(data, index=pd.Grouper(key="Fecha", freq="Y"), columns= None, values='Noroeste mon\ufffdxid
```

**Figura 9:** Ciclo que imprime las tablas pivote de la figura[7]

Para facilitar el manejo de la información, se creó un arreglo con la suma del Monóxido de carbono de cada año.

```
#Creamos un arreglo con los datos del Monóxido para evitar que se visualice en gráficas y tablas el "\ufffd" generado.
Monóxido=np.array([114911 ,98999 ,76557 ,87032, 80897, 78152, 70309, 57781, 48179])
```

**Figura 10:** Arreglo de las sumas de Monóxido.

Se creó un DataFrame llamado "tabla_t"que contuviese únicamente dos columnas, utilizando los arreglos "Fechasÿ "Monóxido".

```
#Creamos un dataframe con los valores obtenidos
tabla_t = {'Año' : Fechas, 'Noroeste Monóxido de Carbono' : Monóxido}

tabla_t = pd.DataFrame(tabla_t, columns = ['Año', 'Noroeste Monóxido de Carbono'])
tabla_t
```

**Figura 11:** DataFame agrupado por Fechas y Monóxido.

Utilizando Matplotlib, graficamos ambos arreglos, colocando los años en el eje x, y la cantidad de Monóxido de carbono en el eje y,

```
#Graficamos para ver el comportamiento del Monóxido a través de los años
fig, ax = plt.subplots(figsize=(10, 5),dpi=100)

plt.plot(Fechas, Monóxido, marker ='o', color ='#9768d8')
ax.set(xlabel = "Año", ylabel = "Monóxido de carbono [ppm]")

plt.suptitle("Mónoxido de carbono")

plt.grid(True)
plt.show()
```

**Figura 12:** Código para la gráfica de los datos del DataFrame.

Al observar la gráfica y su comportamiento, se decidió hacer un ajuste lineal para elaborar nuestra predicción.Con la función "polyfit"se realizó la regresión lineal y se almacenaron los valores de la pendiente y ordenada al origen.

```
#Hacemos el ajuste lineal
pendiente, ordenada =np.polyfit(Fecha,Monóxido,1)
```

**Figura 13:** Ajuste lineal.

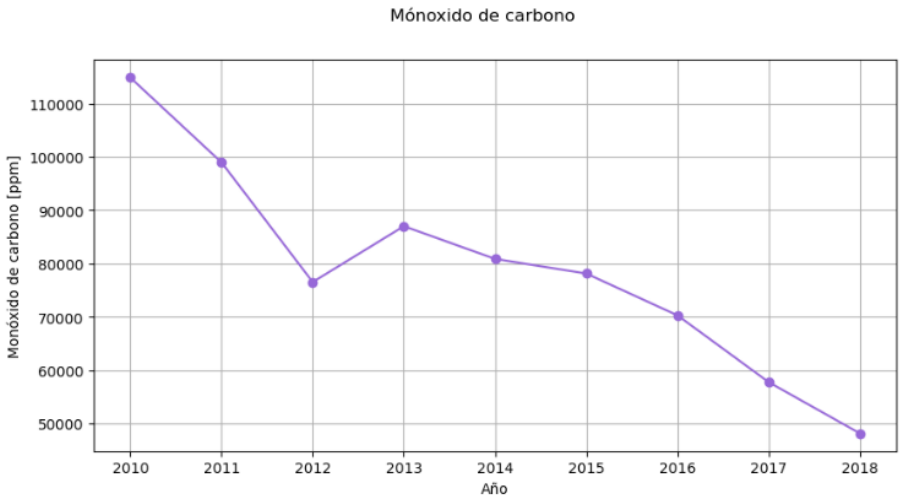


**Figura 14:** Grafica de los datos antes del ajuste lineal.

Conociendo la ecuación de una recta donde y=mx+b, se sustituyeron los valores de "mz "b.ºbtenidos del código anterior, guardados en "pendientez .ºrdenadaz asignado a "x"los valores: 10, 11 y 12, correspondientes a los años 2019, 20201 y 2021, respectivamente; para de esta manera ser capaces de cálcular nuestra predicción.

```
#Hacemos las predicciones, considerando y=mx+b
pred_2019=(pendiente*10)+ordenada
pred_2020=(pendiente*11)+ordenada
pred_2021=(pendiente*12)+ordenada
```

**Figura 15:** Predicción.

# III. CONCLUSIÓN

Para 2019 se predijo un total de 44 872.05555555553 ppm de Mónoxido de carbono en la zona Noroeste de la Ciudad de Mexico; para 2020, 38006.08888888886 ppm y para 2021, 31140.1222222222 ppm, redondeando los datos, obtendriamos la siguiente tabla como resultado, podemos observar que verdaderamete se ha reducido su cantidad en el aire con el transcurrir de los años y se espera que siga disminuyendo.

| Año | Monóxido de carbono [ppm] |
|---|---|
| 2019 | 44 872 |
| 2020 | 38 006 |
| 2021 | 31 140 |

**Cuadro 1:** Tabla de resultados.

# Análisis de número de casillas de votación electoral a instalar en el estado de Guanjauato

**Resumen**

Se analizaron los datos de la Estadística de Padrón Electoral y Lista Nominal de Electores de Guanajuato con los datos proporcionados por el INE, para, filtrando y manipulando adecuadamente los datos, ser capaces de predecir la cantidad de actas y de casillas que serán necesarias para febrero 2021.

## IV. INTRODUCCIÓN

Dentro de la página del Instituto Nacional Electoral [2] se descargaron datos de "Estadística de Padrón Electoral y Lista Nominal de Electores", fueron utilizados los de la lista nominal desde septiembre de 2019 hasta diciembre de 2020, se utilizaron únicamente los datos del estado de Guanajuato (estado=11) y se desea predecir los de febrero de a través de un ajuste lineal.

## V. DESARROLLO

Este análisis se realizó en Phyton, por lo que lo primero que se realizó fue importar las librerías que se usarían para el mismo, considerando las necesidades de filtrar tablas, realizar cálculos entre columnas, crear arreglos, entre otros.

```
import numpy as np
import matplotlib.pyplot as plt
import os
import glob
import pandas as pd
import re
from scipy.optimize import minimize
```

**Figura 16:** Importando las librerías.

Después de esto, se cargaron los archivos necesarios para realizar la predicción y se creó una variable que los almacenara a todos. Fueron 16 archivos en total nombrados al principio por el mes, textualmente, y después, por su año y mes, númericamente, por ejemplo: Septiembre_201909.

```
file='/Users/Angie/Downloads/Datos HIGI'
files=glob.glob("./Datos HIGI/*.txt")
```

**Figura 17:** Carga de los archivos ".txt".

A continuación, se eligió uno de los archivos, del que se seleccionaron únicamente 5 columnas: .ENTIDAD", "DISTRITO", "MUNICIPIO", "SECCION Y "LISTA_NAL", en "data1"; este también se filtró por entidad, en "data_gto", eligiendo únicamente la 11, correspondiente a Guanajauto. Esto fue realizado para analizar el orden y comportamiento de los datos de maneras más simple y clara.

```
data1 = pd.read_csv(files[0],usecols= ['ENTIDAD', 'DISTRITO', 'MUNICIPIO', 'SECCION', 'LISTA_NAL'])
data1_gto = data1[data1['ENTIDAD']==11]
data1_gto
```

| | ENTIDAD | DISTRITO | MUNICIPIO | SECCION | LISTA_NAL |
|---|---|---|---|---|---|
| **17838** | 11.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **17839** | 11.0 | 1.0 | 6.0 | 331.0 | 695.0 |
| **17840** | 11.0 | 1.0 | 6.0 | 332.0 | 769.0 |
| **17841** | 11.0 | 1.0 | 6.0 | 333.0 | 1061.0 |
| **17842** | 11.0 | 1.0 | 6.0 | 334.0 | 666.0 |
| **...** | ... | ... | ... | ... | ... |
| **20975** | 11.0 | 15.0 | 17.0 | 1173.0 | 1407.0 |
| **20976** | 11.0 | 15.0 | 17.0 | 1174.0 | 1929.0 |
| **20977** | 11.0 | 15.0 | 17.0 | 1175.0 | 1210.0 |
| **20978** | 11.0 | 15.0 | 17.0 | 1176.0 | 4517.0 |
| **20979** | 11.0 | 15.0 | 17.0 | 1177.0 | 2269.0 |

3142 rows × 5 columns

**Figura 18:** Creación de la primer tabla.

Se elaboró un ciclo que te le asignará a cada año, su archivo correspondiente. Primero, se agruparon los valores de cada fecha, en este caso de cada año, y se acomodaron cronológicamente, posteriormente, se le destino a cada una el archivo que coincidera con la fecha.

```
date=[]
date_=[]
files_=[]
```

```
for i,file in enumerate(files):
    date.append(re.findall(r'\d+',file)[0])

temp=sorted(range(len(date)), key=date.__getitem__)

for i in temp:
    date_.append(date[i])
    print(date[i],files[i])
    files_.append(files[i])
```

**Figura 19:** Ciclo de las fechas existentes en los archivos.

El siguiente paso fue graficar los datos para obeservar su comportamiento y comprobar que siguiese una tendencia lineal para poder aplicar una regresión lineal y ser capaces de realizar nuestra predicción. Como podemos observar en la fig[20].

```
plt.figure(figsize=(20,10))

for i in range(5):
    plt.plot(df_municipio.iloc[i])
    plt .yscale('log')
```

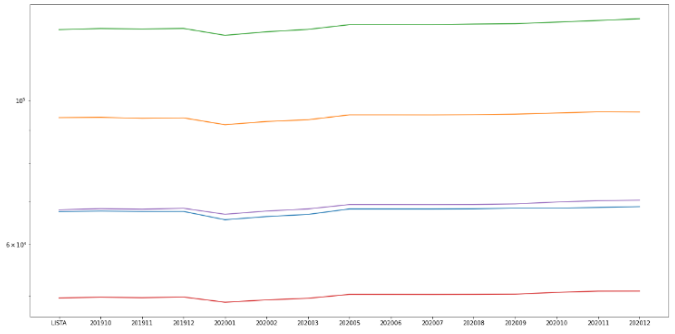

**Figura 20:** Grafica de 5 municipios.

Convertimos nuestro DataFrame en un arreglo, de está manera podremos utilizarlo fácilmente en la función de la regresión. Volvemos a graficar para asegurarnos de que no hemos modificado los valores.

```
#Convertimos la tabla en un arreglo para hacer la regresión lineal
municipios=np.asarray(df_municipio)
for i in range(5):
    plt.plot(municipios[i])
```

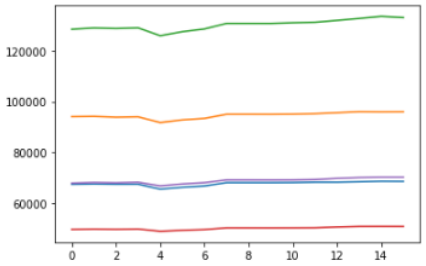


**Figura 21:** Conviertiendo nuestra tabla en un arreglo.

Posteriormente, elaboramos un arreglo que genere la regresión lineal por cada municipio y guardamos los valores que nos arroja de la pendiente y ordenada al origen en "na" y "ba", reespectivamente; los aplicamos, utilizando la fórmúla y=mx+b y guardamos los datos en la variable "predicción_lineal".

```
#Ajuste lineal
fits=[]
prediccion_lineal=[]

for i in range(len(municipios)):
    xx=np.arange(len(municipios[i]))
    na, ba= np.polyfit(xx, municipios[i], 1, w=municipios[i])
    fits.append([na,ba])
    pred=na*(xx[-12]+12)+ba
   # if pred < municipios[i][-1]:
     #   pred=municipios[i][-1]

    prediccion_lineal.append(pred)
prediccion_lineal
```

**Figura 22:** Regresión lineal por municipio.

Añadimos la columna de predicción a la tabla que ya habíamos filtrado anteriormente por municipio.

```
df_municipio['PREDICCION_LINEAL']= prediccion_lineal
```

**Figura 23:** Añadimos la predicción al DataFrame

Realizamos una suma con todos los valores dados por la predicción, para obtener la cantidad de actas nececarias.

```
sum_pre=sum(prediccion_lineal)
sum_pre
```

**Figura 24:** Suma de las predicciones.

Finalmente, divimos entre 750, ya que se consideran 750 actas por cada casillas

```
total_casillas=sum_pre/750
total_casillas
```

**Figura 25:** División para obtener el número de actas.

# VI. CONCLUSIÓN

Se obtuvo un total de 6042.567323217931 actas, ya de que debemos reportar un número entero, resultaría en 6043 actas necesarias para febrero 2021. También se intentó estimar esta misma cantidad aplicando el ajuste por secciones, lo que nos daría un resultado de 7688 casillas, valor obtenido en el mismo proyecto cuando fue elaborado en Excel, sin embargo, no fui capaz de crear un código que nos brindará tal resultado.

# Referencias


[1] Gobierno de la Ciudad de México. *Calidad del aire*. URL: `http://www.aire.cdmx.gob.mx/default.php?opc=%5C%27aKBhnmI=%5C%27&opcion=aw==`.

[2] Instituto Nacional Electoral. *Padrón Electoral*. URL: `https://www.ine.mx/transparencia/datos-abiertos/#/archivo/estadistica-padron-electoral-lista-nominal-electores`.