

PROYECTO PROCESAMIENTO DE DATOS CON PYTHON



Integrantes del equipo



**Felix Alberto Nieto
García**

Integrante del equipo



**Ana Katherine Cuevas
Flores**

Integrante del equipo



**Ximena Ávila
Villagómez**

Integrante del equipo



**Alejandro De Fuentes
Martínez**

Integrante del equipo

1

Identificación del Problema



Identificación del problema

ACCIDENTES VIALES

Implicaciones

- ▶ Personas lesionadas o fallecidas
- ▶ Costo alto del mantenimiento de la ciudad
- ▶ Poca seguridad para ciclistas, motociclistas, peatones y personas involucradas
- ▶ Menos de la mitad de los ciudadanos con automóviles tienen seguro

Posibles causas

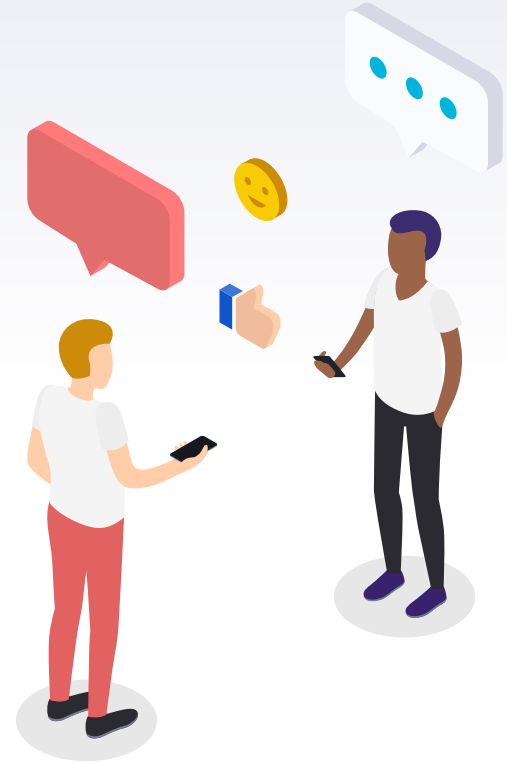
- ▶ Geografía de la ciudad
- ▶ Mala infraestructura de calles
- ▶ Falta de cultura vial



2

Planteamiento de preguntas

Planteamiento de al menos 5 preguntas pertinentes sobre el problema.



Planteamiento de preguntas

- ▶ ¿Cuáles son los principales tipos de accidentes que se han registrado como incidentes viales por el C5 de la CDMX?
- ▶ ¿Cómo se distribuyen los incidentes con código A acontecidos en las Alcaldías de la CDMX?
- ▶ ¿En qué año acontecieron la mayor cantidad de accidentes con código A (Afirmativos)?
- ▶ ¿Cómo se distribuyen los accidentes con código A (Afirmativos) por año en las Alcaldías de la CDMX?
- ▶ ¿Cuál es la proporción entre falsas alarmas y accidentes realmente acontecidos?
- ▶ ¿Cuántos accidentes involucran a peatones, ciclistas o motociclistas?

3

Colección de datos

Para responder a las preguntas planteadas anteriormente.

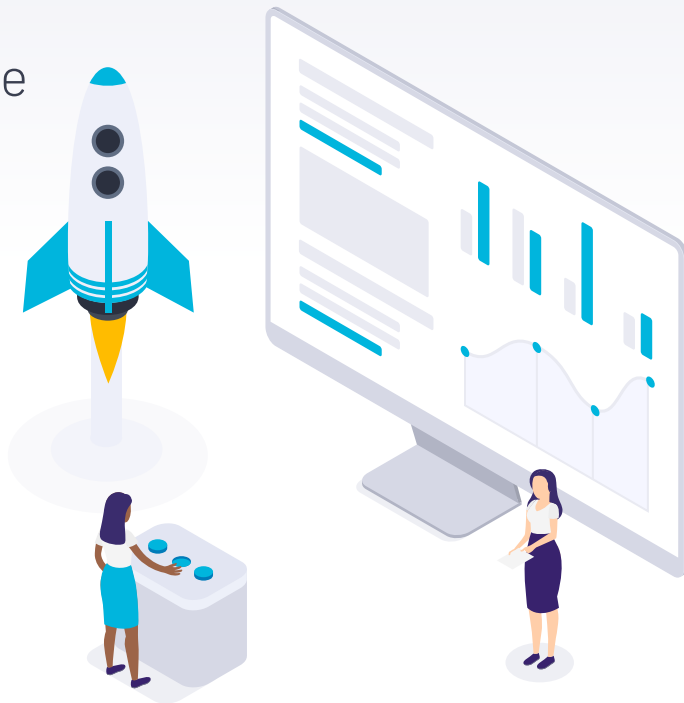


Colección de datos

- ▶ **Incidentes viales** reportados por el **C5** de la **CDMX**.
- ▶ Periodos **2014-2020 + Febrero 2021**.
- ▶ Portal de datos abiertos de la CDMX.
- ▶ **Diccionario de datos**.



<https://bit.ly/ProcesamientoPython>



Colección de datos

Tipo	Descripción	Link (URL)
<i>Dataset</i>	Conjunto de datos con los incidentes viales reportados por el Centro de Comando, Control, Cómputo, Comunicaciones y Contacto Ciudadano de la Ciudad de México (C5) desde 2014 y actualizado mensualmente.	<p>Datos 2014-2020 https://datos.cdmx.gob.mx/dataset/incidentes-viales-c5?activity_id=50471cf9-4e32-4e3c-a3f3-b280b9a97fd3</p> <p>Datos 2021 https://datos.cdmx.gob.mx/dataset/incidentes-viales-c5</p>

► Librerías utilizadas

```
#Manejo y adquisición de datos
import pandas as pd #Manejo de dataframes
import numpy as np #Manejo de arreglos numéricos y funciones de calculo
import json #Manejo de archivos json
import requests #Hacer que las solicitudes HTTP

#Graficación
import matplotlib.pyplot as plt #Graficación de datos
from IPython.display import display #Funciones de visualización en interfaz
from ipywidgets import interact, interactive # Funciones interactivas HTML widgets para Jupyter notebooks
import folium # Visualización de datos en mapas interactivos
import folium.plugins #Herramientas para mapas interactivos
```

Análisis exploratorio y limpieza de datos

4

```
[ ] print(f"Datos: \n 2021      {data_21.shape}\n 2014-2020 {data_14_20.shape} ")
```

Datos:

2021 (12947, 17)

2014-2020 (1412333, 19)

shape

head()

```
[ ] data_14_20.head(2)
```

		id	folio	fecha_creacion	hora_creacion	día_semana	codigo_cierre	fecha_cierre	año	mesdecierre	hora_cierre	delegacion_inicio	incidente_c4	latitud	long
0	0	GA/160123/05714		23/01/2016	22:35:04	Sábado	(A) La unidad de atención a emergencias fue de...	24/01/2016	2016	Enero	00:21:08	VENUSTIANO CARRANZA	accidente-choque sin lesionados	19.422113	-99.06
1	1	AO/160123/05826		23/01/2016	22:50:49	Sábado	(A) La unidad de atención a emergencias fue de...	24/01/2016	2016	Enero	04:40:37	CUAJIMALPA	accidente-choque con lesionados	19.358320	-99.25

```
data_21.head(2)
```

	Unnamed: 0	folio	fecha_creacion	hora_creacion	día_semana	fecha_cierre	hora_cierre	incidente_c4	delegacion_inicio	latitud	longitud	codigo_cierre	clas_con
0	1	C5/210201/04663	01/02/2021	19:14:39	Lunes	01/02/2021	20:36:48	accidente-choque con lesionados	GUSTAVO A. MADERO	19.46157	-99.11521	A	UR
1	2	C5/210201/00391	01/02/2021	01:02:02	Lunes	01/02/2021	01:20:09	accidente-choque sin lesionados	GUSTAVO A. MADERO	19.47144	-99.08177	N	EME

dtypes

data_14_20.dtypes

id	int64
folio	object
fecha_creacion	object
hora_creacion	object
dia_semana	object
codigo_cierre	object
fecha_cierre	object
ano	int64
mesdecierre	object
hora_cierre	object
delegacion_inicio	object
incidente_c4	object
latitud	float64
longitud	float64
clas_con_f_alarma	object
tipo_entrada	object
delegacion_cierre	object
geopoint	object
mes	int64
dtype:	object

data_21.dtypes

Unnamed: 0	int64
folio	object
fecha_creacion	object
hora_creacion	object
dia_semana	object
fecha_cierre	object
hora_cierre	object
incidente_c4	object
delegacion_inicio	object
latitud	float64
longitud	float64
codigo_cierre	object
clas_con_f_alarma	object
tipo_entrada	object
delegacion_cierre	object
ano	int64
mes	int64
dtype:	object



int64 – Enteros

float64- decimales

object – Puede ser número o carácter


```
columnas_en_comun = list(filter(lambda valor: valor in list(data_14_20.columns), list(data_21.columns)))
print(f'Existen {len(columnas_en_comun)} columnas en común y son: \n')
for i in columnas_en_comun:
    print(i)
```

Existen 16 columnas en común y son:

folio
fecha_creacion
hora_creacion
dia_semana
fecha_cierre
hora_cierre
incidente_c4
delegacion_inicio
latitud
longitud
codigo_cierre
clas_con_f_alarma
tipo_entrada
delegacion_cierre
ano
mes

Observaciones

1. Se observa que los dos dataframes tienen diferente cantidad de columnas. Entonces es necesario borrar las columnas sobrantes de los dos dataframes o seleccionar las columnas en común.
2. En el dataset del 2014 al 2020 la columna geopoint y mesdecierre es redundante. Por otra parte la columna id es innecesaria. Del mismo modo para el dataset del 2021 la columna Unamed: 0 es innecesaria.
3. La columna de codigo_cierre, tiene diferentes formatos para referirse a los eventos. Se debe de tener consistencia.
4. Se debe de cambiar a tipo fecha las columnas hora, día y fecha.
5. Se debe de concatenar los dos dataframes y para ello es necesario ordenar las columnas.



```
folio
fecha_creacion
hora_creacion
dia_semana
fecha_cierre
hora_cierre
incidente_c4
delegacion_inicio
latitud
longitud
codigo_cierre
clas_con_f_alarma
tipo_entrada
delegacion_cierre
ano
mes
```

```
#Se eligen las columnas en común
data_14_20 = data_14_20[columnas_en_comun]
data_21 = data_21[columnas_en_comun]
```

```
#Rectificar si el número de columnas es el mismo
data_21.shape[1] == data_14_20.shape[1] #True - tienen el mismo numero de columnas
```

```
True
```

```
#Se realiza una corrección en el código de cierre para el dataframe 2014-2021
data_14_20["codigo_cierre"] = data_14_20["codigo_cierre"].apply(lambda x: x[1])
```

codigo_cierre

(A) La unidad
de atención a
emergencias
fue de...

codigo_cierre

A

```
#Construimos un vector con el nombre de las columnas en el orden que queremos
```

```
orden = ['folio', 'ano', 'mes', 'dia_semana',  
         'incidente_c4', 'codigo_cierre', 'latitud', 'longitud',  
         'tipo_entrada', 'clas_con_f_alarma',  
         'delegacion_inicio', 'delegacion_cierre',  
         'fecha_creacion', 'hora_creacion',  
         'fecha_cierre', 'hora_cierre']
```

```
#Verifica si tiene el mismo orden
```

```
np.sum(data_14_20.columns == data_21.columns) == len(orden)
```

True

```
#se reordenan las columnas
```

```
data_21 = data_21[orden]
```

```
data_14_20 = data_14_20[orden]
```

```
#concatenación de los datos
data = pd.concat([data_14_20,data_21] ,axis=0)

#Características del nuevo dataframe
dimension_data = data.shape
dimension_data

(1425280, 16)
```

```
#Se puede observar que tenemos datos de los años 2014-2021, los 12 meses del año y los 7 días de la semana
years = sorted(data['ano'].unique())
meses = sorted(data['mes'].unique())
dia = sorted(data['dia_semana'].unique())
years,meses,dia

([2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021],
 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],
 ['Domingo', 'Jueves', 'Lunes', 'Martes', 'Miércoles', 'Sábado', 'Viernes'])
```



```
#Se puede ver que tenemos datos de
tipo_accidente = sorted(data['incidente_c4'].unique())
print(f"Tenemos {len(tipo_accidente)} tipos diferentes de accidentes:\n")
tipo_accidente
```

Tenemos 26 tipos diferentes de accidentes:

```
['Detención ciudadana-accidente automovilístico',
'accidente-choque con lesionados',
'accidente-choque con prensados',
'accidente-choque sin lesionados',
'accidente-ciclista',
'accidente-ferroviario',
'accidente-monopatín',
'accidente-motociclista',
'accidente-otros',
'accidente-persona atrapada / desbarrancada',
'accidente-vehículo atrapado',
'accidente-vehículo desbarrancado',
'accidente-vehículo atrapado-varado',
'accidente-volcadura',
'cadáver-accidente automovilístico',
'cadáver-atropellado',
'detención ciudadana-accidente automovilístico',
'detención ciudadana-atropellado',
'lesionado-accidente automovilístico',
'lesionado-atropellado',
'mi ciudad-calle-incidente de tránsito',
'mi ciudad-taxi-incidente de tránsito',
'sismo-choque con lesionados',
'sismo-choque con prensados',
'sismo-choque sin lesionados',
'sismo-persona atropellada']
```

```
#Códigos de cierre del accidente
codigo_accidente = sorted(data['codigo_cierre'].unique())
print(f"Tenemos {len(codigo_accidente)} tipos de codigo:\n")
codigo_accidente
```

Tenemos 5 tipos de codigo:

```
['A', 'D', 'F', 'I', 'N']
```

```
#Tipo de entrada
tipo_entrada = sorted(data['tipo_entrada'].unique())
print(f"Tenemos {len(tipo_entrada)} tipos de entradas\n")
tipo_entrada
```

Tenemos 9 tipos de entradas

```
['APLICATIVOS',
 'BOTÓN DE AUXILIO',
 'CÁMARA',
 'LLAMADA APP911',
 'LLAMADA DEL 066',
 'LLAMADA DEL 911',
 'RADIO',
 'REDES',
 'ZELLO']
```

```
#Tipo de entrada
clase_alarma = sorted(data['clas_con_f_alarma'].unique())
print(f"Tenemos {len(clase_alarma)} clases de alarma:\n")
clase_alarma
```

Tenemos 4 clases de alarma:

```
['DELITO', 'EMERGENCIA', 'FALSA ALARMA', 'URGENCIAS MEDICAS']
```

```
#Delegacion
delegaciones = data['delegacion_cierre'].unique()
#data['delegacion_inicio'].unique()
print(f"Tenemos {len(delegaciones)} delegaciones:\n")
delegaciones
```

Tenemos 17 delegaciones:

```
array(['VENUSTIANO CARRANZA', 'CUAJIMALPA', 'TLALPAN',
      'MAGDALENA CONTRERAS', 'MIGUEL HIDALGO', 'GUSTAVO A. MADERO',
      'TLAHUAC', 'ALVARO OBREGON', 'CUAUHTEMOC', 'COYOACAN',
      'IZTAPALAPA', 'BENITO JUAREZ', 'AZCAPOTZALCO', 'IZTACALCO',
      'MILPA ALTA', 'XOCHIMILCO', nan], dtype=object)
```

```
def conteo_catacteres(entrada):  
    return len(str(entrada))
```

```
#Se convierten cada fila de la columna seleccionada a string y se cuentan el numero de carecteres,  
#y si el número de carecteres es diferente entonces tenemos errores  
data['folio'].apply(conteo_catacteres).value_counts()  
#Se tiene que todos los datos tienen el consistencia en la longiutud
```

```
15    1425280  
Name: folio, dtype: int64
```

```
data['fecha_creacion'].apply(conteo_catacteres).value_counts()
```

```
10    1316233  
8      109047  
Name: fecha_creacion, dtype: int64
```

```
data['fecha_cierre'].apply(conteo_catacteres).value_counts()
```

```
10    1316233  
8      109047  
Name: fecha_cierre, dtype: int64
```

```
data['hora_creacion'].apply(conteo_catacteres).value_counts()
```

```
8      1396967  
7        26781  
11        1398  
10         116  
9           16  
6            1  
4            1  
Name: hora_creacion, dtype: int64
```

```
data['hora_cierre'].apply(conteo_catacteres).value_counts()
```

```
8      1396156  
7        27596  
11        1389  
10         119  
9           15  
6            3  
3            2  
Name: hora_cierre, dtype: int64
```

```
data.isna().sum()
```

```
folio          0
ano            0
mes            0
dia_semana     0
incidente_c4   0
codigo_cierre  0
latitud        443
longitud       435
tipo_entrada   0
clas_con_f_alarma 0
delegacion_inicio 161
delegacion_cierre 143
fecha_creacion 0
hora_creacion  0
fecha_cierre   0
hora_cierre    2
dtype: int64
```

no se distribuyen los NaN?

```
#Se obtiene los indices de las filas que tienen por lo menos un nan y las que no tienen ninguno
index_data_nan = (data.isna().sum(axis=1) != 0 )
index_data_clear = (data.isna().sum(axis=1) == 0)
```

```
data_clear = data[index_data_clear]
data_nan = data[index_data_nan]
```

```
data_clear.isna().sum()
```

```
folio      0
ano        0
mes        0
dia_semana 0
incidente_c4 0
codigo_cierre 0
latitud    0
longitud   0
tipo_entrada 0
clas_con_f_alarma 0
delegacion_inicio 0
delegacion_cierre 0
fecha_creacion 0
hora_creacion 0
fecha_cierre 0
hora_cierre 0
dtype: int64
```

```
data_nan.isna().sum()
```

```
folio      0
ano        0
mes        0
dia_semana 0
incidente_c4 0
codigo_cierre 0
latitud    443
longitud   435
tipo_entrada 0
clas_con_f_alarma 0
delegacion_inicio 161
delegacion_cierre 143
fecha_creacion 0
hora_creacion 0
fecha_cierre 0
hora_cierre 2
dtype: int64
```

```
#diferencia entre el número de filas original y el número de final sin contar las filas que contienen un NaN
```

```
print(f'''El número de filas borradas es {data_nan.shape[0]}, correspondiente a un {round(data_nan.shape[0]/dimension_data[0],4)*100} %  
de los datos originales.''')
```

El número de filas borradas es 609, correspondiente a un 0.04 %
de los datos originales.

```
data_clear = data_clear.drop(columns=['hora_creacion', 'hora_cierre'])
```

```
data_clear.columns
```

```
Index(['folio', 'ano', 'mes', 'dia_semana', 'incidente_c4', 'codigo_cierre',  
      'latitud', 'longitud', 'tipo_entrada', 'clas_con_f_alarma',  
      'delegacion_inicio', 'delegacion_cierre', 'fecha_creacion',  
      'fecha_cierre'],  
      dtype='object')
```

```
import datetime as dt  
def convertir_fecha(fecha):  
    try:  
        return dt.datetime.strptime(fecha, "%d/%m/%Y")  
    except ValueError:  
        return dt.datetime.strptime(fecha, "%d/%m/%y")
```

```
data_clear['fecha_creacion'] = data_clear['fecha_creacion'].apply(convertir_fecha)  
data_clear['fecha_cierre'] = data_clear['fecha_cierre'].apply(convertir_fecha)
```

```
data_clear.dtypes
```

folio	object
ano	int64
mes	int64
dia_semana	object
incidente_c4	object
codigo_cierre	object
latitud	float64
longitud	float64
tipo_entrada	object
clas_con_f_alarma	object
delegacion_inicio	object
delegacion_cierre	object
fecha_creacion	datetime64[ns]
fecha_cierre	datetime64[ns]
dtype:	object

```
data_clear.to_csv('datos_clear.csv')
```

► Procesamiento y resultados

- Dataframe limpio → **data_clear**
- Resultados → Respuestas a las preguntas de investigación.



Frecuencias generales de datos categóricos

```
# Información sobre las frecuencias de los datos limpios
```

```
columnas_categoricas = [ 'ano', 'mes', 'dia_semana', 'incidente_c4',  
                          'codigo_cierre', 'tipo_entrada',  
                          'clas_con_f_alarma', 'delegacion_inicio',  
                          'delegacion_cierre']
```

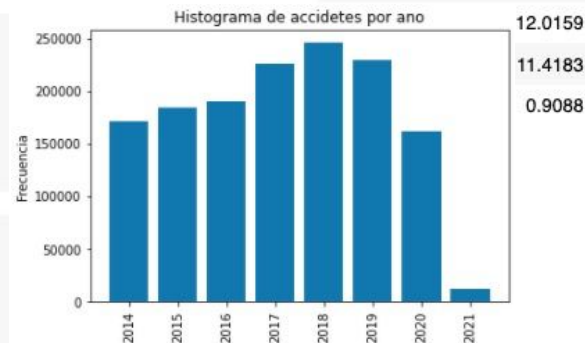
```
#Se calculas las frecuencias y se guardan en un diccionario
```

```
frecuencias_generales = {}  
for i in columnas_categoricas:  
    frecuencias_generales[i] = data_clear[i].value_counts()
```

```
#Función interactiva para visualizar la información de cada categoria
```

```
@interact
```

```
def mostrar_valores(column = columnas_categoricas):  
    x = frecuencias_generales[column].index  
    y = frecuencias_generales[column].values  
    data = pd.DataFrame(frecuencias_generales[column])  
    display(graf_bar(x,y,f"Histograma de accidentes por {column}"))  
    display(mostrar(data,column))
```



column	✓ ano
None	mes
	dia_semana
	incidente_c4
2018	246
	codigo_cierre
2019	229
	tipo_entrada
2017	226
	clas_con_f_alarma
	delegacion_inicio
	delegacion_cierre
2016	190984
	13.4055
2015	184810
	12.9721
	12.0159
	11.4183
	0.9088

Clasificación de accidentes afirmativos (Código A)

```
group_5 = data_clear.groupby(['codigo_cierre','incidente_c4']).size()
mostrar(pd.DataFrame(group_5['A'],columns=['frecuencia']), 'frecuencia')
```

lesionado-accidente automovilístico	3297	0.5365
accidente-persona atrapada / desbarrancada	3186	0.5184
accidente-otros	1630	0.2652
cadáver-atropellado	1561	0.2540
accidente-ciclista	1467	0.2387
cadáver-accidente automovilístico	1025	0.1668
accidente-vehículo atrapado	515	0.0838
accidente-choque con prensados	442	0.0719
accidente-vehículo atrapado-varado	409	0.0666
sismo-choque sin lesionados	203	0.0330
detención ciudadana-atropellado	194	0.0316
accidente-vehículo desbarrancado	183	0.0298
sismo-choque con lesionados	121	0.0197
detención ciudadana-accidente automovilístico	43	0.0070
accidente-ferroviario	17	0.0028
accidente-monopatín	16	0.0026
sismo-persona atropellada	8	0.0013
mi ciudad-calle-incidente de tránsito	5	0.0008
sismo-choque con prensados	1	0.0002

	frecuencia	porcentaje
incidente_c4		
accidente-choque sin lesionados	348310	56.6781
accidente-choque con lesionados	131631	21.4194
lesionado-atropellado	84471	13.7454
accidente-motociclista	23445	3.8150
accidente-volcadura	12361	2.0114

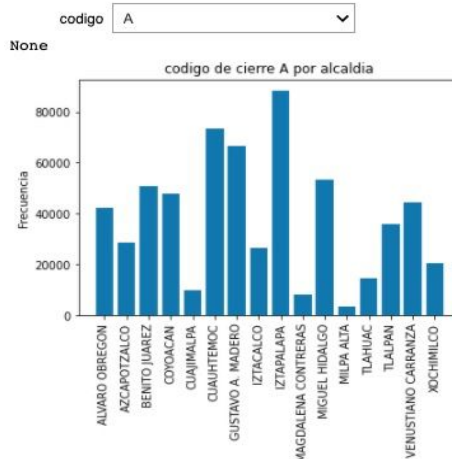
Accidentes afirmativos por delegación

delegacion_inicio	frecuencia	porcentaje
IZTAPALAPA	88375	14.3807
CUAUHTEMOC	73606	11.9774
GUSTAVO A. MADERO	66632	10.8426
MIGUEL HIDALGO	53390	8.6878
BENITO JUAREZ	50937	8.2886
COYOACAN	47791	7.7767
VENUSTIANO CARRANZA	44435	7.2306
ALVARO OBREGON	42239	6.8733
TLALPAN	35745	5.8165
AZCAPOTZALCO	28634	4.6594
IZTACALCO	26688	4.3428
XOCHIMILCO	20304	3.3039
TLAHUAC	14493	2.3583
CUAJIMALPA	9711	1.5802
MAGDALENA CONTRERAS	7989	1.3000
MILPA ALTA	3572	0.5812

```
group_1 = data_clear.groupby(['codigo_cierre','delegacion_inicio']).size()
```

```
#Accidentes clasificados por delegacion
mostrar(pd.DataFrame(group_1['A'],columns=['frecuencia'],'frecuencia'))
```

```
#Función interactiva para observar la distribución de incidentes
#en cada delegación con diferente tipo de codigo
@interact
def mostrar_valores(codigo = codigo_accidente ):
    x = group_1[codigo].index
    y = group_1[codigo].values
    display(graf_bar(x,y,f"codigo de cierre {codigo} por alcaldia"))
```



Accidentes afirmativos por año

```
group_6 = data_clear.groupby(['codigo_cierre','ano']).size()

mostrar(pd.DataFrame(group_6['A'],columns=['frecuencia']), 'frecuencia')
```

	frecuencia	porcentaje
ano		
2014	138456	22.5300
2015	109668	17.8455
2016	77489	12.6092
2017	75961	12.3606
2018	75927	12.3551
2019	75606	12.3028
2020	56915	9.2614
2021	4519	0.7353

lesionado-atropellado	84471	13.7454
accidente-motociclista	23445	3.8150
cadáver-atropellado	1561	0.2540
accidente-ciclista	1467	0.2387
detención ciudadana-atropellado	194	0.0316
sismo-persona atropellada	8	0.0013

Accidentes por delegaciones en un mes y año en particular

```
group_2 = data_clear.groupby(['codigo_cierre','ano','mes','delegacion_inicio']).size()
```

```
year = list(frecuencias_generales['ano'].index)
month = list(frecuencias_generales['mes'].index)
```

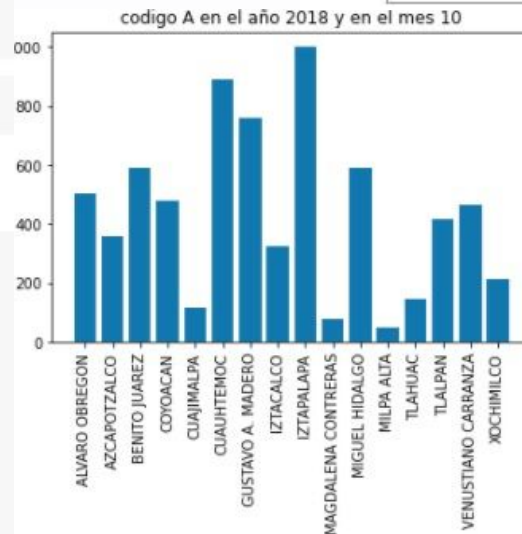
```
year,month
```

```
([2018, 2019, 2017, 2016, 2015, 2014, 2020, 2021],
 [10, 12, 11, 2, 8, 9, 3, 6, 7, 5, 1, 4])
```

```
#Grafica interactiva
@interact
def mostrar_valores(codigo = codigo_accidente,
                    year=year,
                    mes =month):

    try:
        x = group_2[codigo][year][mes].index
        y = group_2[codigo][year][mes].values
        display(graf_bar(x,y,f"codigo {codigo} en el año {year} y en el mes {mes}"))
    except KeyError:
        display(print("No se tienen datos de la selección"))
```

codigo	A	▼
year	2018	▼
mes	10	▼

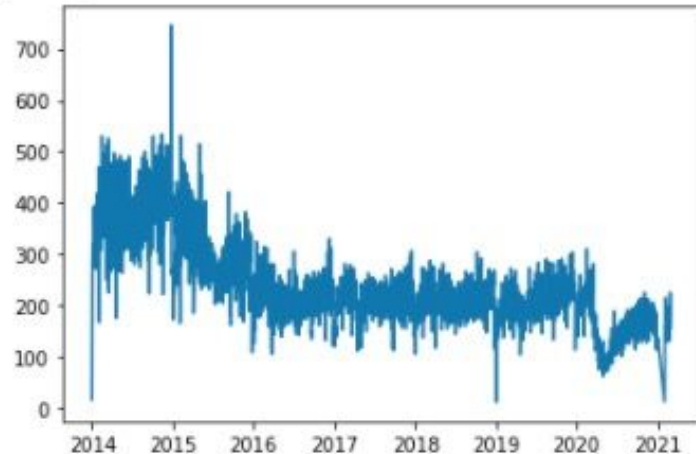


Serie de tiempo de accidentes de los accidentes afirmativos

```
group_3 = data_clear.groupby(['codigo_cierre', 'fecha_creacion']).size()
```

```
plt.plot(group_3['A'].index, group_3['A'].values)
```

[<matplotlib.lines.Line2D at 0x7ffae3010a10>]



► Procesamiento y resultados

- Experiencia y conocimiento del equipo.
- Funciones interactivas y dinámicas



<https://bit.ly/ProcesamientoPython>



Uso de APIs y planes a futuro

APIs de los Datos Abierto
de la CDMX

<https://datos.cdmx.gob.mx/>

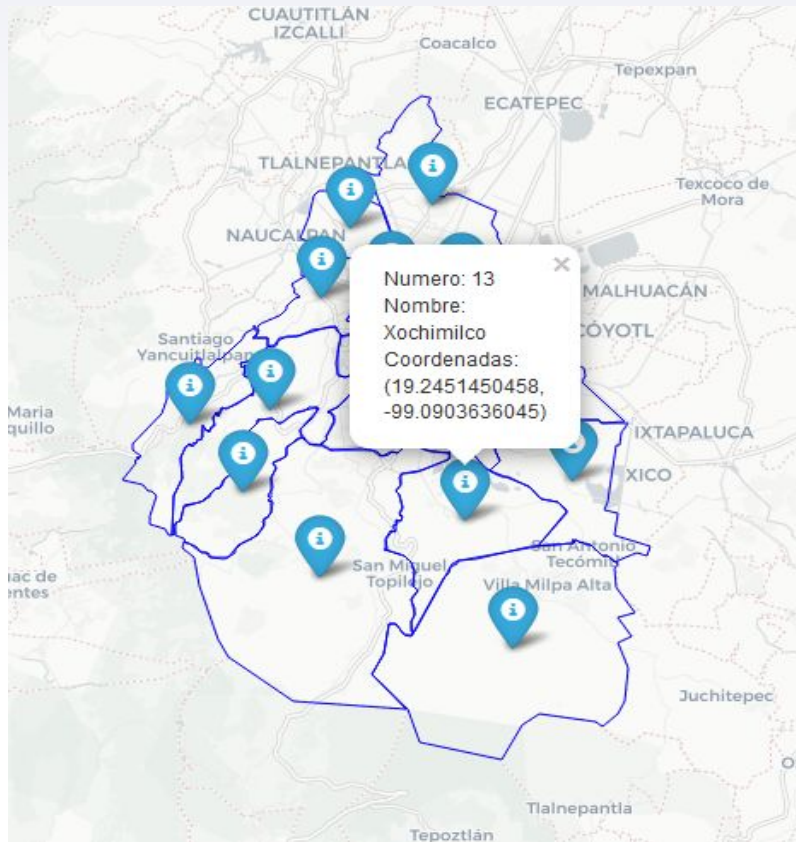
+



Folium

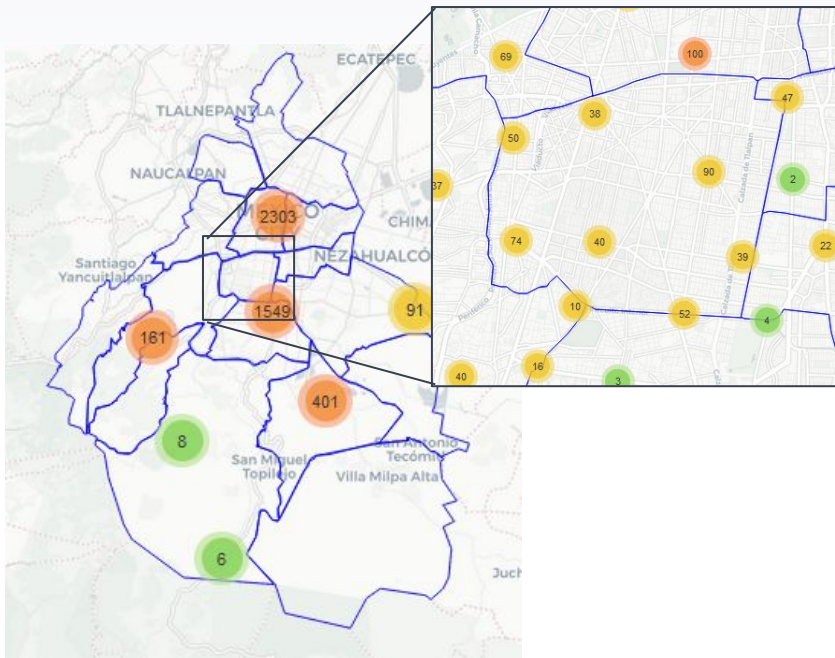


Mapas interactivos
Georeferencias puntos

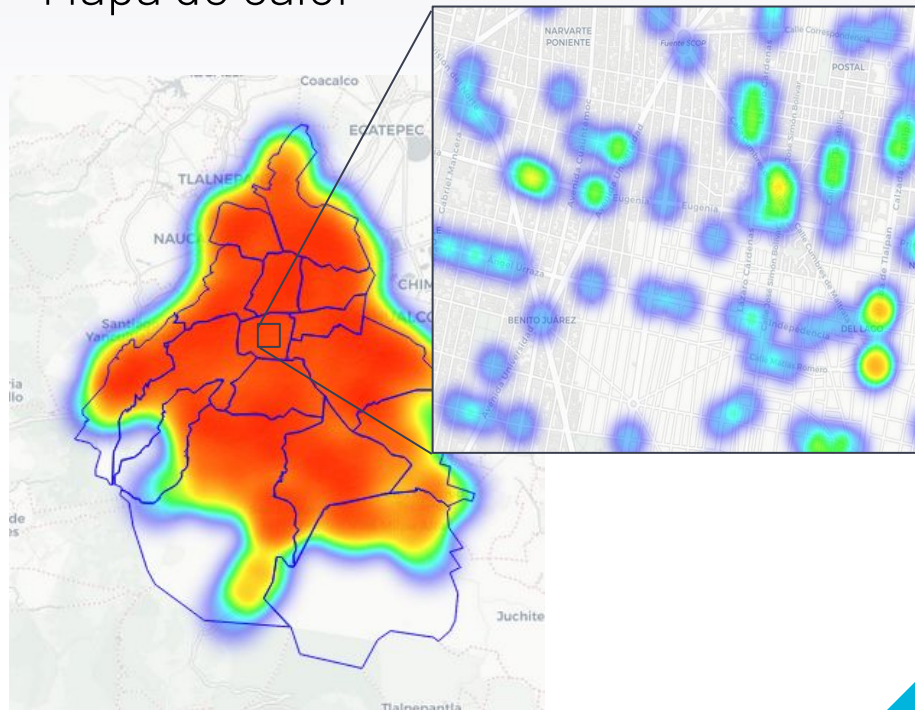


Uso de APIs y planeas a futuro

Cluster



Mapa de calor



Uso de APIs y planeas a futuro

Posibles Datasets

- Altas de riesgo de la CMDX.
- Localización de las estaciones y terminales del sistema de transporte unificado.
- Vías primarias de la ciudad de México.
- Ubicación de escuelas, hospitales, centros de salud y mercados.
- Índice de desarrollo social en la ciudad.
- Infracciones de tránsito.

Cluster de accidentes



Dar una posible explicación a los incidentes viales al relacionarlos con otros datasets y proponer soluciones con base a ellos.

Conclusiones

- ▶ Comprensión de la problemática.
- ▶ Amplitud en la perspectiva.
- ▶ Necesidad de mayor investigación e información complementaria de otras instituciones para sustentar soluciones integrales.



¡Gracias!

Pueden encontrarnos en:

- ▶ Ximena Ávila Villagómez
(a01423754@itesm.mx)
- ▶ Ana Katherine Cuevas Flores
(anakatherine.cuevas@upaep.edu.mx)
- ▶ Félix Alberto Nieto García
(ffelix_f@ciencias.unam.mx)
- ▶ Alejandro De Fuentes Martínez
(adefuentes29@alumnos.uaq.mx)





*Los datos son la nueva
ciencia. Big Data son las
respuestas"*

Pat Gelsinger
CEO de Intel