



UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN
FACULTAD DE INGENIERÍA MECÁNICA Y
ELÉCTRICA
POSGRADO EN INGENIERÍA DE SISTEMAS

Portafolio

GABRIELA SANCHEZ YEPEZ
saphira3000@hotmail.es
1935064

PROFESOR
Dra. Satu Elisa Schaeffer



Práctica 1: Preparación de datos con bash

Introducción

Se trabajará con los datos reportados de violencia intrafamiliar, pandillerismo y suicidio en ocho municipios del estado de Nuevo León en el periodo 2006-2009. Los datos incluyen el cuadrante y plano de la colonia, dentro del municipio en el cual se reportó el incidente. La base de datos de los suicidios registrados en el año 2009, contienen la fecha, lugar y método usado, así como si la persona dejó una nota o no.

Con los datos obtenidos se plantean las siguientes preguntas de investigación:

- ¿La zona influye en el tipo de incidente y número de casos?
- ¿Qué factores influyen en los casos de suicidios?

This website does not host notebooks, it only renders notebooks available on other websites.

Delivered by [Fastly](#), Rendered by [Rackspace](#)

[nbviewer GitHub repository](#).

nbviewer version: aa567da

nbconvert version: 5.3.1

Rendered 2 minutes ago

Branch: **master** [▼](#) [Ciencia_de_Datos / p1.ipynb](#) [Find file](#) [Copy path](#)

 **Saphira3000** cambios 6da5b5c 1 hour ago

1 contributor

70 lines (69 sloc) 2.71 KB

[Raw](#) [Blame](#) [History](#) [Edit](#) [Delete](#)

Práctica 1: Preparación de datos con bash

Gabriela Sánchez Y.

Introducción

Se trabajará con los datos de incidentes de violencia intrafamiliar y comunitaria reportados en municipios del estado de Nuevo León en el periodo enero-diciembre 2018.

Los registros contienen 47 columnas de las cuales únicamente 17 proporcionan información relevante, sin datos personales, que serán usados en el análisis. Estos registros proveen la siguiente información

- Mes
- Municipio
- Fecha
- Hora,

estos sobre el lugar, fecha y hora del incidente. De las víctimas y agresores se recopilan los siguientes datos

- Número de víctimas y agresores en el incidente
- Sexo
- Edad
- Parentesco con el agresor
- Estado civil
- Ocupación
- Escolaridad. En total se tienen 16410 registros.

El principal objetivo del análisis de los datos es identificar los grupos vulnerables de víctimas y las posibles causas al problema que permitan planear estrategias de prevención.

Con esto en mente se plantean algunas preguntas de investigación que pueden cambiar a lo largo del curso del estudio:

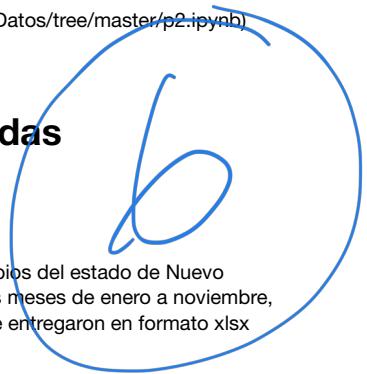
- ¿Cuál es la influencia del factor fecha y hora en la cantidad de incidentes?
- ¿Cuál es la influencia del factor lugar en la cantidad de incidentes?
- ¿La escolaridad influye en el número de casos?
- ¿Cuál es el grupo de edades más vulnerable?

Preprocesamiento de los datos

Los datos fueron proporcionados por la doctora Patricia L. Cerdá, en formato xlsx. Se presentan en dos archivos, el primero contiene los incidentes de violencia intrafamiliar y comunitaria presentados durante los meses de enero a noviembre en el año 2018 y el segundo, los incidentes del resto del año, teniendo un total de 16 410 reportes.

El único preprocesamiento que se realizó con bash fue convertir los archivos de los datos a un formato manejable, esto es, csv, con la siguiente instrucción:

```
ssconvert datos.xlsx vf.csv
```



Práctica 2: Lectura y manipulación de datos con pandas

Gabriela Sánchez Y.

Se tienen datos de incidentes de violencia familiar registrados en el año 2018 en ocho diferentes municipios del estado de Nuevo León. Los datos se encuentran en dos grupos, el primer grupo contiene los incidentes registrados en los meses de enero a noviembre, mientras que el segundo grupo contiene los incidentes registrados en el mes de diciembre. Los datos se entregaron en formatoxlsx por lo que primero los cambiamos a formato csv.

El objetivo de la práctica es la manipulación de datos usando la librería pandas, se probarán algunas de las siguientes funcionalidades:

- leer data frames de archivos
- escribir data frames en CSV
- agregar o eliminar columnas en data frames
- combinar dos o más data frames
- filtrar renglones.

El primer paso es cargar nuestros datos.

```
In [13]: import pandas as pd  
vf1 = pd.read_csv("/home/saphira/Desktop/datos/vf_Ene-nov.csv")  
vf2 = pd.read_csv("/home/saphira/Desktop/datos/vf_dic.csv")
```

Ahora intentaremos combinar ambos data frames para obtener uno solo que contenga los datos registrados durante todo el año de estudio.

```
In [14]: vf = pd.concat([vf1,vf2])
```

Al hacer la concatenación se conservan los índices originales. Para evitar futuros problemas lo solucionamos con la siguiente instrucción:

```
In [15]: vf.index = range(vf.shape[0])
```

Por curiosidad, revisamos si todo va bien con la lectura de los datos. De antemano sabemos que tenemos 47 columnas y 16411 renglones contando el encabezado.

```
In [50]: vf.size
```

```
Out[50]: 771270
```

```
In [26]: print(16410*47)
```

```
771270
```

Siguiendo con los objetivos de la práctica, agregaremos una columna dummy al final del data frame y después la eliminaremos junto con las últimas tres columnas del data frame combinado y revisamos que lo hayamos hecho bien.

```
In [16]: vf.loc[:, "dummy"] = 0  
vf.columns
```

```
Out[16]: Index(['MES', 'MUNICIPIO HECHOS', 'COLONIA', 'DELITO', 'MODALIDAD',  
    'FECHA HECHOS', 'VIOLENCIA', 'TIPO VIOLENCIA', 'CALLE', 'ENTRE CALLE',  
    'NUMERO EXTERIOR', 'NUMERO INTERIOR', 'HORA HECHOS', 'CARPETA',  
    'FECHA APERTURA', 'UNIDAD DE INVESTIGACION', 'ESPECIALIDAD', 'JEFATURA',  
    'REGION', 'TIPO LUGAR', 'TIPO ARMA', 'VICTIMA', 'SEXO',  
    'EADADES (VICTIMA)', 'PARENTESCO', 'ESTADO CIVIL', 'OCCUPACION',  
    'ESCOLARIDAD', 'DIRECCION (VICTIMA)', 'RELIGION',  
    'ES LA PRIMERA AGRESION', 'DESDE CUANDO RECIBE AGRESIONES',  
    'ALGUIEN MAS RECIBIO AGRESIONES', 'RECIBIO ATENCION PSICOLOGICA',  
    'RECIBIO ATENCION MEDICA', 'AGRESOR', 'SEXO.1', 'EADADES (AGRESOR)',  
    'ESTADO CIVIL.1', 'OCCUPACION.1', 'ESCOLARIDAD.1', 'DIRECCION (AGRESOR)',  
    'RELIGION.1', 'RECIBIO ATENCION PSICOLOGICA.1',  
    'TIPO Y GRAVEDAD DE LA AGRESION', 'INGRESO MENSUAL (FAMILIAR)',  
    'MOTIVO DE LA AGRESION', 'dummy'],  
   dtype='object')
```

```
In [17]: del vf["MOTIVO DE LA AGRESION"]  
vf = vf.drop(["TIPO Y GRAVEDAD DE LA AGRESION", "INGRESO MENSUAL (FAMILIAR)", "dummy"], axis = 1)  
vf.columns
```

```
Out[17]: Index(['MES', 'MUNICIPIO HECHOS', 'COLONIA', 'DELITO', 'MODALIDAD',  
    'FECHA HECHOS', 'VIOLENCIA', 'TIPO VIOLENCIA', 'CALLE', 'ENTRE CALLE',  
    'NUMERO EXTERIOR', 'NUMERO INTERIOR', 'HORA HECHOS', 'CARPETA',  
    'FECHA APERTURA', 'UNIDAD DE INVESTIGACION', 'ESPECIALIDAD', 'JEFATURA',  
    'REGION', 'TIPO LUGAR', 'TIPO ARMA', 'VICTIMA', 'SEXO',  
    'EADADES (VICTIMA)', 'PARENTESCO', 'ESTADO CIVIL', 'OCCUPACION',  
    'ESCOLARIDAD', 'DIRECCION (VICTIMA)', 'RELIGION',  
    'ES LA PRIMERA AGRESION', 'DESDE CUANDO RECIBE AGRESIONES',  
    'ALGUIEN MAS RECIBIO AGRESIONES', 'RECIBIO ATENCION PSICOLOGICA',  
    'RECIBIO ATENCION MEDICA', 'AGRESOR', 'SEXO.1', 'EADADES (AGRESOR)',  
    'ESTADO CIVIL.1', 'OCCUPACION.1', 'ESCOLARIDAD.1', 'DIRECCION (AGRESOR)',  
    'RELIGION.1', 'RECIBIO ATENCION PSICOLOGICA.1'],  
   dtype='object')
```

Efectivamente hemos eliminado las columnas que deseábamos. Despues queremos filtrar la información con el objetivo de ver cuántos casos se tienen tal que el tipo de violencia es en perjuicio de menor de 12 años. Finalmente guardamos este filtrado en un archivo CSV.

```
In [47]: vfcaso = vf[vf['MODALIDAD'] == 'VIOLENCIA FAMILIAR EN PERJUICIO DE MENOR DE 12 AÑOS']  
print(len(vfcaso.index))
```

118

```
In [49]: vfcaso.to_csv('vf_caso.csv', header=True, index=False)
```

?

Practica 3: Estadística descriptiva básica.

Gabriela Sanchez Yepez

El objetivo de esta práctica es obtener información estadística básica de los datos. Recordemos qué datos tenemos y veamos qué podemos hacer.

```
In [35]: import pandas as pd  
datos = pd.read_csv("/home/saphira/Desktop/datos/vf_p3.csv")  
datos.dtypes
```

```
Out[35]: mes           int64  
mpio          object  
col            object  
delito         object  
mod             object  
fecha           object  
violencia      object  
tipo            object  
hora            object  
carpeta          object  
fecha_ap        object  
u_inv           object  
especialidad    object  
jefatura        object  
region           object  
tipo_lugar       object  
tipo_arma        object  
victima          int64  
sexo_v           object  
edad_v           object  
parentesco       object  
edo_civil        object  
ocupacion        object  
escolaridad      object  
atencion_psi     object  
atencion_med     object  
agresor          float64  
sexo_a            object  
edad_a            object  
edo_civil_a       object  
ocupacion_a       object  
escolaridad_a     object  
atencion_psi.1    object  
dtype: object
```

Analicemos primero cuál es el mes más violento.

```
In [36]: datos.mes.value_counts(sort=True, normalize=True, dropna=True)
```

```
Out[36]: 7      0.108958
6      0.097197
5      0.095917
9      0.095247
8      0.094942
3      0.083608
10     0.082572
4      0.082084
11     0.068860
2      0.066971
12     0.062888
1      0.060756
Name: mes, dtype: float64
```

El mes más violento fue el mes de Julio, mientras que Enero tuvo el menor número de incidentes reportados. Veamos ahora qué pasa con los municipios.

```
In [37]: datos.mpio.value_counts(sort=True, normalize=True, dropna=True)
```

```
Out[37]: Monterrey          0.173918
Guadalupe           0.107983
García              0.104388
Apodaca             0.102986
Escobedo            0.094394
Juárez              0.090311
San Nicolás De Los Garza 0.056002
Santa Catarina      0.055515
Cadereyta Jiménez   0.030225
Pesquería           0.029677
El Carmen            0.017489
Zuazua              0.017428
Ciénega De Flores   0.015539
Linares              0.015174
Salinas Victoria     0.014808
Montemorelos         0.013102
San Pedro Garza García 0.010969
Allende              0.010664
Sabinas Hidalgo     0.005667
Santiago             0.005606
Doctor Arroyo        0.004144
Galeana              0.003230
Anáhuac              0.002681
Mina                 0.002133
Cerralvo             0.001950
Hidalgo              0.001767
Marín                0.001645
Terán                0.001097
China                0.001036
Lampazos             0.000853
Aramberri            0.000853
Hualahuises          0.000792
Ramones              0.000670
Bustamante           0.000670
Villaldama           0.000548
Abasolo               0.000488
Higueras             0.000427
Iturbide              0.000427
Zaragoza              0.000366
Bravo                0.000366
Anahuac              0.000305
Mier Y Noriega        0.000244
Doctor Coss           0.000244
Doctor González       0.000244
Parás                0.000183
Treviño              0.000183
Melchor Ocampo        0.000183
Herreras              0.000122
Agualeguas            0.000122
Aldamas              0.000122
Vallecillo            0.000061
Name: mpio, dtype: float64
```

Población
(Por Col.)

```
In [38]: mpios = datos.mpio.value_counts()
mpios.size
```

```
Out[38]: 51
```

El municipio más violento del estado en el año 2018 fue Monterrey. Cabe destacar que el estado de Nuevo León se divide en 51 municipios y hay datos reportados de incidentes de violencia familiar en todos ellos en el año 2018.

Analicemos un poco más el municipio de Monterrey. Revisemos cuál es el *top five* de colonias con violencia familiar.

```
In [39]: mont = datos[datos["mpio"] == "Monterrey"]
cols = mont.col.value_counts(sort=True, normalize=True, dropna=True)
cols[0:5]
```

```
Out[39]: CENTRO DE MONTERREY      0.058164
INDEPENDENCIA          0.033287
MODERNA                0.017870
CROC                   0.017519
VALLE DE SANTA LUCIA   0.017169
Name: col, dtype: float64
```

En cuanto a la cantidad de víctimas...

```
In [40]: datos.victima.unique()
```

```
Out[40]: array([1, 3, 2, 4, 5, 6, 7])
```

```
In [41]: datos.victima.value_counts(sort=True, normalize=True, dropna=True)
```

```
Out[41]: 1      0.931932
2      0.054845
3      0.010055
4      0.002620
5      0.000366
6      0.000122
7      0.000061
Name: victima, dtype: float64
```

```
In [42]: datos.tipo_lugar.value_counts(sort=True, normalize=True, dropna=True)
```

```
Out[42]: DOMICILIO PARTICULAR           0.886289
VÍA PÚBLICA                         0.097075
NEGOCIO                            0.010786
OFICINA                            0.001463
ESTACIONAMIENTO DE CENTRO COMERCIAL 0.000975
HOSPITAL - CLINICA                  0.000914
INSTITUCION EDUCATIVA               0.000731
TERRENO BALDÍO                      0.000609
CARRETERA                           0.000305
TRANSPORTE PUBLICO INDIVIDUAL       0.000305
CENTRO DE ESPECTACULOS              0.000183
DESPoblADO                          0.000183
ESTADIO                             0.000061
TRANSPORTE PUBLICO COLECTIVO        0.000061
TRANSPORTE INDIVIDUAL                 0.000061
Name: tipo_lugar, dtype: float64
```

La mayoría de los casos se reportan para incidentes en los que hay una sola víctima y sucedieron en un domicilio particular.

Analicemos ahora las edades de las víctimas.

```
In [43]: datos.edad_v.unique()
```

```
Out[43]: array(['46', '26', '42', '38', '28', '61', '54', '39', '19', '23', '36',
   '33', '40', '47', '31', '16', '44', '20', '64', '48', '7 10 34',
   '62', '29', '50', '41', '60', '34', '43', '80', '32', '27', '37',
   '35', '30', '59', '25', '51', '24', '22', '18', '11 45', '13 35',
   '48 49 66', '52', '75', '57', '53', '67', '69', '21', '11 51',
   '14 52', '4 25', 'NE', '17 30', '13 33', '5 6 33', '56', '72',
   '55', '8 37', '23 48', '58', '2 32', '12 13', '20 43', '49', '13',
   '63', '78', '8 12', '8 15', '24 45', '13 40', '6 38',
   '14 23 52 70', '6', '2 26', '14 46', '44 63', '66', '10', '6 21',
   '13 37', '29 38', '8 13 45', '45', '15', '16 40', '8 11', '8', '2',
   '9', '5 52', '18 35', '10 28', '2 7 25', '86', '14 42', '13 30',
   '23 36', '6 6 14 16 37', '71', '6 29', '46 11 a 157', '13 19',
   '40 43', '4 5 33', '8 14 30', '17', '6 35', '9 10', '4 32',
   '4 7 26', '11', '13 27', '6 26', '9 NE', '26 30', '5 6', '17 34',
   '11 29', '7 74', '15 38', '14 39', '14', '77', '18 53', '18 42',
   '11 37', '15 41', '13 15', '14 37', '28 44', '14 27', '9 11',
```

(0 a 100 (cada)) - split()

11 + 15
—
2

'10 44', '8 13 38', '14 17 44', '3', '8 42', '54 80', '18 36',
'17 46', '5', '8 31', '18 21', '42 52', '7 12', '65', '76', '6 40',
'9 34', '18 69', '1 8 17 39', '94', '2 24', '13 32', '12 18 41',
'14 54', '19 41', '26 56', '7', '83', '18 38', '17 18 36', '4',
'68', '82', '8 24', '23 37', '6 6 39', '2 28', '9 44', '11 35',
'4 28', '4 26', '8 9 26', '1 a 5', '84', '16 42', '9 26', '15 55',
'16 44', '12 16 37', '24 46', '2 4 32', '11 67', '31 49 52',
'6 33', '9 46', '15 30', '49 53', '6 8 28', '9 11 47', '12 29',
'18 44', '15 42', '74', '15 34', '10 49', '10 11', '79', '9 40',
'6 10 43', '9 15 18 39', '21 44', '15 33', '7 30', '35 54', '3 21',
'3 25', '2 3 4', '10 47', '42 82', '9 28', '12', '5 25', '10 35',
'73', '30 33', '2 27', '14 32', '14 36', '1', '61 84', '1 20',
'11 31', '45 50', '13 34', '5 30', '21 28 28', '16 39', '12 36',
'14 44', '21 24', '8 32', '10 27', '8 8 12', '7 80', '7 31',
'18 49', '5 11', '28 58', '15 45', '21 33', '10 20', '9 30', '70',
'16 60', '8 30', '2 6 8 52', '7 32', '11 25', '25 70', '7 28',
'7 39', '1 29', '7 8 29', '15 47', '15 36', '5 6 10 29', '5 40',
'8 27', '3 20', '4 20', '81', '16 54', '16 33', '9 25', '15 35',
'5 8 9 10 36', '9 14', '5 43', '8 45', '21 46', '17 49', '5 12 14',
'37 52', '8 61', '70 71', '2 22', '20 41', '3 27', '87', '7 36',
'17 39', '8 10 36', '4 19', '36 a 40', '56 81', '18 43', '1 16',
'16 36', '11 30', '2 21', '7 26', '16 34', '21 76', '8 12 39',
'7 7 9 29', '9 54', '9 36', '28 31', '17 53', '19 40', '18 40',
'17 43', '46 48', '3 24', '47 98', '16 41', '14 48', '41 49',
'5 7 9 44', '11 21 51', '28 47', '44 65', '35 36 68',
'10 12 17 40', '23 47', '7 42', '30 30', '10 31', '10 13 41',
'3 6 43', '56 72', '11 15 48', '59 65', '9 38', '5 10 32', '22 49',
'7 38', '14 16 60', '15 43', '6 25', '5 9 14', '96', '7 22',
'7 33', '13 46', '14 34', '16 32', '25 51', '21 48', '34 NE',
'18 47', '88', '7 46', '13 23', '4 6', '9 37', '10 30', '28 53',
'16 47', '8 25', '16 17 44', '6 44', '49 80 87', '4 11 12',
'29 82', '6 30', '18 22 25', '9 11 15 33', '11 a 15', '28 51',
'14 16', '14 69', '11 a 15 46 a 50', '8 35', '49 49', '6 13 19 36',
'4 42', '2 4 25', '11 36', '17 28', '6 22', '22 50', '17 17',
'45 45', '2 3 4 23', '14 38', '27 56', '12 44', '16 46', '5 38',
'4 17 38', '5 23', '13 38', '20 36', '24 11 a 15', '5 6 26',
'12 13 16 35', '8 36', '14 33 63', '29 1 a 5', '3 22', '12 16 33',
'6 7 27', '11 48', '11 49', '11 39', '17 62', '5 16 36', '6 27',
'2 8', '15 50', '17 NE', '13 75', '18 68 80', '10 29', '29 39',
'13 43', '4 15', '45 48', '38 39', '8 28', '14 15 49', '28 50',
'30 39', '38 50', '11 34', '17 38', '18 41', '19 37', '11 12 34',
'8 14 42', '17 42', '17 37', '9 10 33', '9 11 53', '11 17 37',
'12 31', '7 10', '26 a 30', '12 41', '14 26', '11 33', '12 28',
'2 37', '14 33', '32 37', '14 45', '8 9 10', '10 15', '4 12 13 30',
'14 40', '6 13 19', '12 33', '16 26', '7 12 32', '89', '13 45',
'12 43', '2 13 52', '18 54', '6 12', '5 20', '14 43', '8 9 35',
'4 7 47', '8 12 42', '10 15 37', '13 16 16 43', '5 27', '7 34',
'6 34', '31 41', '18 70', '2 5 23', '13 29', '24 48', '15 37',
'16 37', '24 39', '12 32', '7 14', '17 35', '5 11 33', '1 19',
'7 8 30', '18 37', '16 20 37', '31 61', '3 23', '44 82',
'13 20 47', '5 6 8 50', '17 36', '15 32', '28 54', '6 8 30', '85',
'25 34', '17 19 48', '7 11 11 13 38', '10 43', '21 23 52', '26 38',
'1 23', '67 68', '56 59', '10 26', '15 46', '10 36', '15 22 32',
'11 15 27', '3 7', '3 47', '4 37', '13 48', '16 24', '8 14',
'18 45', '34 58', '3 19', '12 37', '8 29', '20 52', '1 27',
'4 10 28', '16 17 54', '13 16 45', '17 40', '15 24', '67 73',
'17 41', '10 42', '68 93', '1 18', '10 13 34', '13 28', '11 41',
'12 13 32', '6 9', '9 12 35', '19 35', '33 38', '3 14 17 53',
'63 66', '46 62 73', '11 40', '16 43', '14 14', '14 29', '12 17',
'3 35', '2 29', '8 8', '16 18 39', '12 38', '13 51', '17 19',
'58 66', '19 51', '5 31', '8 33', '2 4', '15 54', '10 57', '38 70',
'8 13 28', '4 10 12 53', '16 52', '21 40', '1 3 19', '10 23',
'3 50', '5 24', '3 15 33', '8 10 12', '5 7 11 33', '14 35', '1 30',
'6 10 29', '10 40', '90', '5 37', '18 a 20', '33 73', '17 48',
'16 17 23 40', '3 26', '12 18 46', '1 5', '36 58', '6 8',
'10 15 19 39', '20 40', '2 7 9', '28 56', '12 16', '7 27',
'7 9 11 13 34', '48 52', '21 47', '23 24', '3 8', '16 18 42',
'4 21', '3 32', '6 36', '46 81', '10 12 39', '16 a 17', '15 44',
'4 23', '10 46', '19 22 34', '8 13 32', '2 31', '41 63', '13 42',
'1 31', '2 23', '1 2 4', '3 5 8 31', '6 a 10', '15 31', '27 34',
'35 56', '6 10', '6 12 35', '15 25 45', '27 42', '25 42', '50 74',

```
'19 46', '33 60', '15 18 48', '3 6 31', '2 6 8 31', '15 39',
'23 45', '28 49', '13 14 32', '8 8 25', '26 26', '36 39',
'6 25 NE', '17 36 44', '3 12 13', '10 13 19 44', '6 9 15 33',
'9 12', '18 48', '6 8 41', '8 40', '5 6 7 8 12 58', '25 78',
'10 32', '6 28', '15 19', '22 42', '71 a 80', '8 34', '7 12 42',
'6 34 6 a 10', '6 11 12 36', '11 13', '80 85', '16 18 40', '93',
'8 16', '8 63', '10 70', '8 10 14 16 36', '31 a 35', '26 49',
'8 9', '11 42', '50 86', '13 15 33', '20 38', '9 16 44', '2 35',
'12 42', '12 15', '19 19', '18 59', '16 64', '1 10 11 15', '11 28',
'3 37', '35 1 a 5', '7 9 29', '45 NE', '36 63', '24 57 61',
'6 8 9', '6 6 10 12 14 16 37', '10 45', '7 13', '2 18', '27 46',
'16 21', '25 52', '14 47', '3 34', '15 29', '26 44', '18 50',
'4 22', '5 6 34', '12 19 62', '2 25', '31 59', '5 32', '17 24',
'63 68', '10 41', '16 NE', '22 22', '31 52', '3 6 24',
'6 20 23 44', '12 46', '6 9 11 38', '4 8', '3 31', '8 73', '17 44',
'13 32 NE', '9 12 38', '16 38', '43 43', '26 37', '43 67',
'8 11 14', '13 31', '4 12 33', '17 69', '6 31', '21 49', '27 27',
'2 19', '13 44', '60 90', '14 75', '4 16 37', '19 21', '1 3',
'11 13 32', '18 18 44', '33 39', '24 37', '11 12 33', '4 11 29',
'2 4 49', '16 35', '9 13', '4 35', '3 8 8 18', '15 40', '4 10',
'6 59', '12 60', '4 29', '9 48', '8 44', '12 18 43', '14 50',
'19 49', '7 11 37', '11 46', '4 4 26', '2 36', '5 35', '15 48',
'19 28', '3 6 7', '14 17 38', '3 18', '12 30', '8 12 15',
'1 3 5 29', '23 56', '8 38', '10 14', '1 22', '12 40', '26 46',
'11 32', '40 NE', '4 24', '32 55', '36 88', '2 17 41', '19 27',
'10 34', '17 47', '69 86', '4 9 12', '4 7 34', '39 45', '34 36 64',
'15 70', '14 33 34', '25 28', '6 8 12 34', '27 47', '26 32',
'27 39 54', '23 43', '4 12 14', '2 5', '60 66', '13 16 44',
'13 56', '1 45', '9 11 17', '4 13 31', '8 15 42', '9 24', '16 55',
'13 57', '12 15 44', '27 31', '9 16 40', '4 5 23', '57 74', '4 43',
'25 56', '50 80', '6 24', '6 32', '52 86', '56 51 a 55', '3 5 29',
'10 11 43', '12 34', '2 5 9 55', '1 a 5 1 a 5', '6 6 23',
'3 6 11 16 18 32', '19 38', '11 18', '36 6 a 10 6 a 10', '7 11 39',
'10 12 14 42', '18 33', '9 29', '21 23 39', '75 81', '8 46',
'9 11 29', '19 47', '16 17', '58 62', '12 51', '13 47', '13 49',
'21 NE', '5 6 15 36', '25 43', '62 82', '4 27', '58 68', '4 9 35',
'3 3 23', '9 16 18 42', '9 9 28', '95', '9 31', '2 6 7 53',
'61 a 70', '4 41', '13 50', '1 21', '8 10', '9 12 12 42',
'6 9 10 29 57', '4 31', '5 47', '15 16 41', '33, 2', '0', '27, 6',
'44, 18', '27, 7', '10, 8', '33, 13, 13, 10', '31, 8, 2', '33, 15',
'33, 11', '32, 12, 10', '34, 10', '38, 31', '30, 28', '16, 1',
'38, 14', '26, 7', '28, 8, 7', '29, 11', '65, 12', '24, 4',
'11, 10', '29, 8', '9, 6', '27, 9, 4', '35, 8, 3', '42, 22, 11, 7',
'67, 16, 10', '35, 16', '38, 11', '16, 13', '80, 55', '36, 13',
'39, 17', '36, 22, 16, 4', '10, 8, 7, 3', '36, 14, 9, 5', '30, 4',
'39, 16', '91', '34, 13', '44, 10', '41, 21', '25, 5', '50, 12',
'41, 11', '11, 5', '49, 14', '32, 3', '78, 59', '43, 13, 7',
'47, 17, 12, 8', '51, 21', '33, 27', '26, 4', '25, 6', '43, 4',
'68, 25, 6', '10, 3', '47, 35', '57, 13', '25, 19, 40, 1, 20, 29,
34, 33, 23, 39, 16, 38, 21, 48, 18, 46, 32, 45], dtype=object)
```

Hay incidentes en los que hubo más de una víctima, por lo que hay más de un valor en algunas entradas. La forma en que fueron capturados los datos nos dará problemas para analizarlos ya que en algunos casos las distintas edades se separan con espacios, en otros casos con comas, se escribieron en una rango o se usa NE cuando no se proporcionó la información. Este problema también se tiene para las edades de los agresores. Analizaremos ésta información más tarde.

Veamos a qué hora se presentan más incidentes.

```
In [49]: datos.hora.unique()

Out[49]: array(['23:40', '08:00', '11:00', '19:00', '19:30', '13:30', '02:30',
   '10:20', '06:00', '21:00', '08:40', '03:00', '00:30', '21:40',
   '21:30', '05:00', '04:00', '10:45', '05:30', '22:30', '19:40',
   '12:30', '14:00', '22:00', '09:30', '18:30', '00:00', '01:00',
   '00:15', '05:50', '20:00', '16:20', '20:30', '08:30', '02:00',
   '21:42', '17:12', '01:10', '17:00', '16:00', '10:00', '17:40',
   '06:05', '09:00', '15:00', '14:51', '13:00', '23:00', '23:30',
   '23:16', '14:40', '06:15', '04:55', '14:30', '17:30', '15:20',
   '01:05', '12:00', '01:30', '18:00', '07:30', '11:30', '12:07',
   '07:00', '21:15', '22:52', '16:30', '22:20', '20:40', '16:47',
   '10:30', '18:45', '16:15', '15:40', '13:20', '18:40', '15:30'])
```

'10:58', '10:55', '10:52', '10:49', '10:28', '10:15', '10:02',
'03:40', '02:35', '00:20', '05:35', '09:15', '15:50', '16:23',
'09:05', '11:08', '17:15', '20:45', '08:20', '14:01', '02:05',
'01:20', '21:03', '23:45', '08:15', '04:10', '08:45', '22:05',
'06:30', '21:20', '22:18', '06:20', '22:40', '11:14', '03:30',
'05:40', '00:40', '06:50', '19:10', '22:15', '02:55', '23:55',
'17:10', '13:40', '04:30', '08:23', '13:50', '22:41', '20:20',
'15:27', '11:50', '02:39', '12:50', '00:45', '21:50', '21:37',
'02:50', '16:40', '11:29', '18:48', '13:45', '14:15', '08:55',
'08:10', '09:58', '00:24', '14:25', '00:10', '19:05', '10:08',
'04:45', '02:44', '09:46', '02:25', '19:13', '15:43', '07:40',
'23:13', '00:21', '10:10', '10:15', '14:17', '15:08', '15:45',
'21:45', '07:10', '14:20', '12:45', '16:05', '13:02', '18:03',
'09:40', '16:33', '10:50', '22:10', '13:31', '09:12', '23:51',
'12:55', '11:21', '16:50', '18:10', '02:20', '23:10', '06:25',
'08:47', '00:35', '18:50', '06:40', '23:18', '12:25', '23:50',
'22:33', '11:25', '00:28', '14:50', '19:55', '15:21', '22:32',
'09:50', '02:04', '02:40', '14:45', '07:45', '11:10', '04:15',
'16:06', '21:10', '13:35', '07:20', '20:15', '19:25', '12:20',
'00:05', '22:19', '20:50', '02:37', '07:15', '02:02', '00:50',
'16:19', '01:36', '21:52', '09:45', '22:50', '08:14', '14:53',
'23:15', '19:50', '23:58', '11:20', '08:03', '01:45', '19:45',
'04:50', '22:45', '15:15', '22:38', '16:08', '10:54', '11:40',
'12:05', '17:20', '07:50', '13:37', '21:19', '22:11', '04:39',
'10:35', '12:10', '16:10', '16:17', '09:20', '23:32', '10:13',
'17:50', '18:35', '18:25', '18:36', '23:35', '10:40', '12:40',
'13:15', '20:22', '15:24', '23:23', '03:59', '19:39', '18:15',
'14:29', '23:27', '19:20', '15:22', '23:20', '06:31', '17:25',
'19:54', '18:27', '15:37', '09:55', '21:35', '13:05', '15:10',
'11:45', '14:24', '20:25', '00:43', '20:35', '11:15', '12:15',
'02:31', '07:29', '21:24', '20:05', '16:18', '16:25', '17:05',
'10:25', '05:20', '21:38', '13:25', '23:25', '21:55', '09:36',
'00:54', '14:10', '13:10', '09:47', '14:35', '22:59', '18:51',
'02:45', '12:33', '08:04', '05:45', '22:56', '23:03', '15:23',
'16:42', '23:37', '11:24', '04:33', '20:44', '04:20', '11:42',
'01:58', '23:04', '07:25', '09:21', '23:14', '19:15', '21:56',
'16:04', '16:13', '00:18', '23:36', '00:39', '03:35', '09:57',
'22:27', '06:10', '22:35', '11:54', '09:13', '03:15', '15:13',
'07:52', '04:53', '22:25', '10:36', '01:34', '11:55', '17:53',
'21:14', '11:59', '15:35', '20:19', '17:36', '00:02', '05:24',
'17:42', '12:35', '20:27', '01:15', '13:28', '16:39', '07:04',
'22:37', '07:57', '18:20', '01:28', '15:48', '12:29', '02:10',
'00:03', '16:48', '22:12', '17:39', '18:13', '21:43', '23:33',
'20:31', '10:03', '13:11', '17:44', '07:42', '21:09', '19:23',
'04:25', '20:55', '02:58', '18:49', '01:27', '00:48', '15:58',
'15:03', '17:24', '01:50', '19:51', '00:25', '15:12', '23:54',
'01:40', '23:22', '22:31', '14:12', '16:45', '11:47', '15:52',
'00:01', '13:03', '12:18', '03:55', '04:23', '01:55', '13:59',
'21:05', '07:58', '17:02', '22:36', '13:34', '06:27', '19:18',
'16:07', '23:49', '03:20', '09:22', '23:47', '00:13', '09:10',
'12:52', '15:34', '09:35', '12:09', '21:18', '09:25', '08:32',
'22:44', '08:44', '00:46', '21:25', '17:45', '08:31', '13:39',
'10:12', '17:13', '08:59', '05:10', '15:25', '20:01', '06:35',
'03:18', '15:01', '08:35', '22:47', '01:07', '18:01', '22:07',
'23:56', '16:41', '14:48', '15:54', '19:28', '16:28', '01:18',
'15:44', '02:28', '13:56', '13:14', '23:43', '13:26', '01:25',
'02:15', '12:53', '20:46', '21:53', '20:10', '23:19', '13:42',
'22:54', '00:07', '17:28', '05:25', '16:29', '10:19', '03:50',
'12:51', '23:44', '02:12', '13:43', '14:19', '21:12', '20:06',
'04:47', '15:55', '08:05', '12:04', '07:09', '11:35', '12:48',
'11:04', '06:11', '09:52', '12:41', '20:16', '01:46', '03:13',
'10:05', '16:14', '12:27', '20:32', '23:38', '11:52', '16:56',
'01:52', '17:27', '21:41', '07:43', '10:28', '21:58', '03:45',
'06:21', '16:37', '09:42', '17:35', '04:49', '04:08', '23:05',
'21:32', '00:14', '11:05', '11:31', '11:22', '10:55', '19:47',
'17:29', '01:57', '17:23', '10:18', '20:38', '22:08', '15:14',
'01:29', '13:55', '17:38', '20:07', '16:32', '07:44', '20:26',
'12:43', '02:21', '02:43', '08:42', '00:55', '03:36', '02:34',
'11:46', '02:53', '02:17', '22:14', '05:19', '17:03', '13:49',
'00:59', '17:01', '22:23', '18:28', '00:31', '00:23', '11:36',
'00:12', '19:26', '02:49', '06:45', '23:59', '08:25', '00:29',
'19:35', '23:42', '14:46', '19:14', '19:57', '06:17', '12:17'.

```
'03:32', '05:15', '00:26', '00:44', '02:08', '09:06', '20:02',
'03:26', '04:19', '04:37', '22:48', '10:46', '00:19', '11:34',
'19:17', '01:48', '06:04', '18:55', '08:50', '00:09', '01:12',
'23:39', '21:22', '22:57', '09:11', '17:55', '21:27', '15:51',
'00:58', '01:32', '09:26', '10:22', '12:06', '14:34', '16:57',
'18:59', '17:52', '22:22', '05:34', '22:55', '09:32', '13:57',
'14:26', '00:38', '07:24', '23:26', '09:37', '21:57', '18:21',
'02:47', '14:38', '20:36', '01:02', '22:29', '23:12', '20:28',
'14:22', '18:41', '21:54', '20:39', '11:11', '12:08', '01:26',
'04:40', '16:58', '18:05', '05:51', '10:01', '01:44', '17:48',
'01:35', '14:16', '12:58', '23:48', '21:06', '07:55', '05:05',
'13:48', '13:51', '00:33', '19:46', '08:49', '15:09', '03:33',
'00:51', '10:17', '16:27', '20:29', '05:37', '08:52', '18:38',
'19:27', '18:18', '06:48', '06:55', '06:09', '10:26', '13:27',
'22:03', '19:34', '21:39', '02:29', '22:46', '10:24', '20:54',
'07:32', '21:13', '01:54', '10:37', '02:14', '14:37', '20:48',
'13:07', '04:36', '23:08', '18:17', '21:47', '14:52', '14:11',
'19:03', '20:13', '05:38', '13:18', '18:54', '21:31', '22:04',
'10:29', '13:16', '12:47', '20:58', '14:03', '08:01', '10:38',
'14:54', '08:51', '06:19', '12:14', '01:11', '17:07', '03:10',
'07:35', '13:23', '09:31', '21:04', '08:07', '01:13', '19:52',
'13:13', '06:12', '04:05', '23:11', '20:18', '03:53', '08:48',
'14:47', '14:32'], dtype=object)
```

Por ahora consideremos sólo las horas sin contar minutos.

```
In [50]: datos["hora"] = [t.hour for t in pd.DatetimeIndex(datos.hora)]
datos.hora.value_counts(sort=True, normalize=True, dropna=True)
```

```
Out[50]: 22    0.073126
21    0.071725
23    0.069104
20    0.065265
0     0.059476
19    0.052834
12    0.050091
18    0.048751
16    0.043144
1    0.041987
10    0.041194
17    0.041133
15    0.037660
14    0.036076
13    0.035832
11    0.034491
2     0.033029
9     0.031383
8     0.028763
3     0.026508
7     0.025411
4     0.019744
6     0.016819
5     0.016453
Name: hora, dtype: float64
```

El rango 20:00-23:00 es el horario que muestra mayor incidencia. Veamos si existe una correlación entre la hora y la cantidad de víctimas.

```
In [51]: datos['hora'].corr(datos['victima'])
```

```
Out[51]: 0.026163253631910043
```

Parece que no hay una relación significativa entre ambas variables.

Práctica 4: Visualización de información con plotly

Gabriela Sánchez Y.

El objetivo de la práctica es la visualización de la información a través de gráficos.

Se cargan los datos para comenzar a trabajar.

```
In [103]: import pandas as pd  
datos = pd.read_csv("/home/saphira/Desktop/datos/vf_p3.csv")
```

Edades de víctimas y agresores

El primer paso que se realiza en la práctica es visualizar la información correspondiente a las edades de las víctimas y agresores. Esta información no fue analizada previamente ya que se requería de cierto preprocesamiento.

Se muestra el preprocesamiento correspondiente para cada uno de los casos:

```
In [104]: edadv = datos.edad_v # edades de las victimas  
edad_v = []  
for dato in edadv:  
    s = str(dato).replace(", ", " ")  
    pedazos = s.split()  
    while "a" in pedazos:  
        pos = pedazos.index("a")  
        desde = int(pedazos[pos - 1])  
        hasta = int(pedazos[pos + 1])  
        prom = (desde + hasta) // 2  
        edad_v.append(prom)  
        pedazos = pedazos[: (pos - 1)] + pedazos[(pos + 2):]  
    edad_v += pedazos  
edad_v = list(filter(lambda dato: dato != "NE", edad_v))  
  
edad_a = datos.edad_a  
edad_a = []  
for dato in edad_a:  
    s = str(dato).replace(", ", " ")  
    pedazos = s.split()  
    while "a" in pedazos:  
        pos = pedazos.index("a")  
        desde = int(pedazos[pos - 1])  
        hasta = int(pedazos[pos + 1])  
        prom = (desde + hasta) // 2  
        edad_a.append(prom)  
        pedazos = pedazos[: (pos - 1)] + pedazos[(pos + 2):]  
    edad_a += pedazos  
  
edad_a = list(filter(lambda dato: dato != "NE", edad_a))  
edad_a = list(filter(lambda dato: dato != "nan", edad_a))  
edad_a = [float(v) for v in edad_a]  
edad_a = [int(v) for v in edad_a]
```

```
In [105]: di_edadA = {}
di_edadV = {}

from collections import Counter

conteos = Counter([int(d) for d in edad_a])
for edad in conteos:
    di_edadA[edad] = conteos[edad]

conteos2 = Counter([int(d) for d in edad_v])
for edad in conteos2:
    di_edadV[edad] = conteos2[edad]
```

Se grafica en un diagrama de barras las edades de víctimas y agresores en los casos reportados. El rango de edades para las víctimas es desde 0-93, encontrándose la mayoría de los casos en el rango 16-55 años, es importante recalcar que se presentan incidentes incluso en menores de cinco años y mayores de 80 años.

El rango de edades de los agresores es de los 18 a los 87 años. Las edades con mayor incidencia son 23, 27, 33 y 38 años.

La media para las víctimas es de 48 años, mientras que para los agresores es 52.

```
In [106]: import plotly
import plotly.plotly as py
import plotly.graph_objs as go
import ssl

plotly.tools.set_credentials_file(username='SaphiraSan', api_key='1MHT02zHoYLTi0YpIiBz')

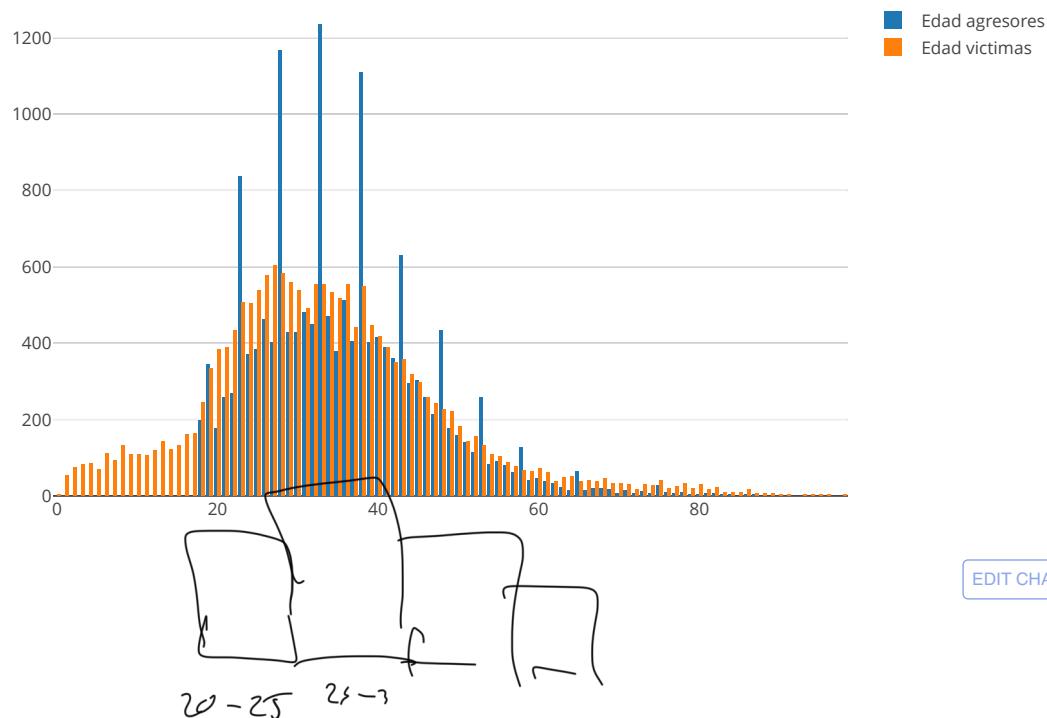
if getattr(ssl, '_create_unverified_context', None):
    ssl._create_default_https_context = ssl._create_unverified_context

trace1 = go.Bar(
    x = list(di_edadA.keys()),
    y = list(di_edadA.values()),
    name = 'Edad agresores'
)
trace2 = go.Bar(
    x = list(di_edadV.keys()),
    y = list(di_edadV.values()),
    name = 'Edad victimas'
)

data = [trace1, trace2]
layout = go.Layout(
    barmode = 'group'
)

fig = go.Figure(data=data, layout=layout)
py.iplot(fig, filename='edades')
```

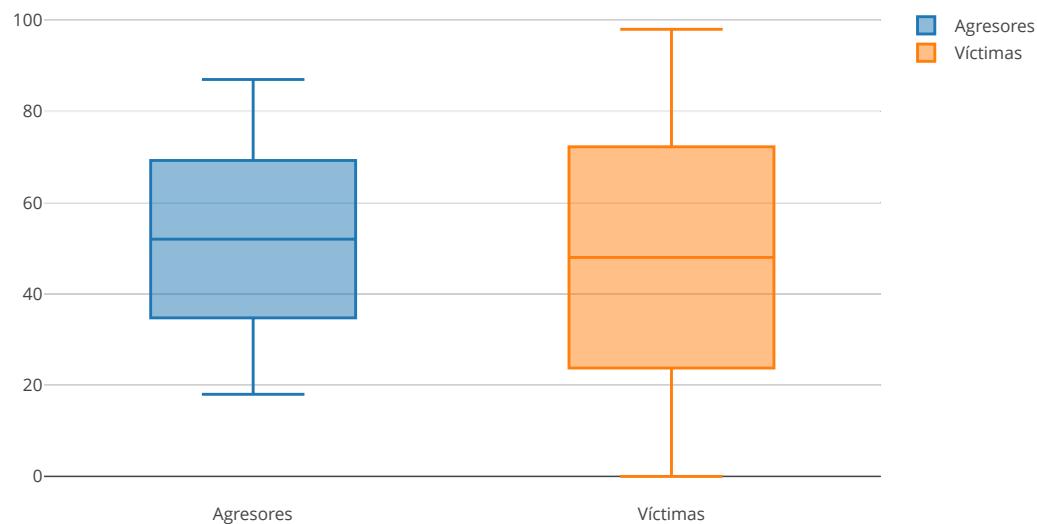
Out[106]:



[EDIT CHART](#)

```
In [107]: trace0 = go.Box(
    y=list(di_edadA.keys()),
    name = 'Agresores',
)
trace1 = go.Box(
    y=list(di_edadV.keys()),
    name = 'Víctimas',
)
data = [trace0, trace1]
py.iplot(data)
```

Out[107]:



[EDIT CHART](#)

Se analiza si existe una correlación entre las edades de los agresores y la cantidad de ocurrencias.

```
In [108]: a = pd.DataFrame.from_dict(di_edadA, orient='index')
a.loc[:, "edad"] = list(di_edadA.keys())
a.index = range(a.shape[0])
cols = ['incidentes', 'edad']
a.columns = cols
a['edad'].corr(a['incidentes'])
```

Out[108]: -0.6997018900857516

Existe una relación fuerte inversa entre los datos. Para edades mayores, la cantidad de indicentes disminuye, esto puede observarse del gráfico de barras anterior.

Incidentes por mes

Se visualiza la información correspondiente a la cantidad de incidentes reportados por mes durante el año 2018. Como se había estudiado previamente, Junio fue el mes más agresivo con un total de 1788 incidentes correspondiente al 10.9% del total de casos reportados, mientras que Enero fue el mes con menor incidencia.

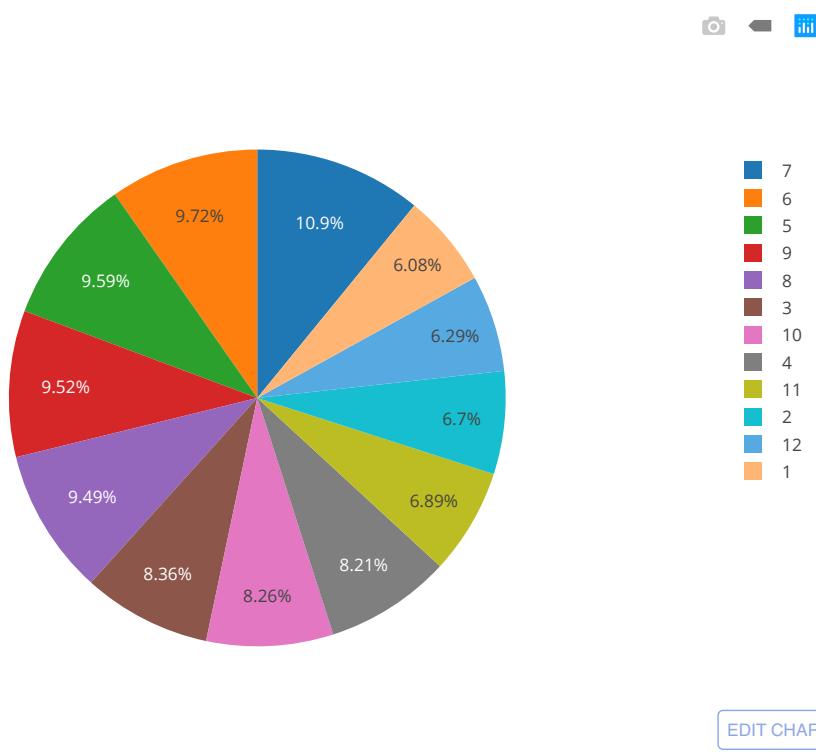
La mayoría de los casos se presentan en el periodo Mayo-Septiembre.

```
In [109]: labels = datos.mes.unique()
values = datos.mes.value_counts(sort=False)

trace = go.Pie(labels=labels, values=values)

py.iplot([trace], filename='incidentes_mes')
```

Out[109]:



EDIT CHART

Incidentes por horas

En el gráfico es posible observar que el rango 20:00-23:00 es el horario que muestra mayor incidencia.

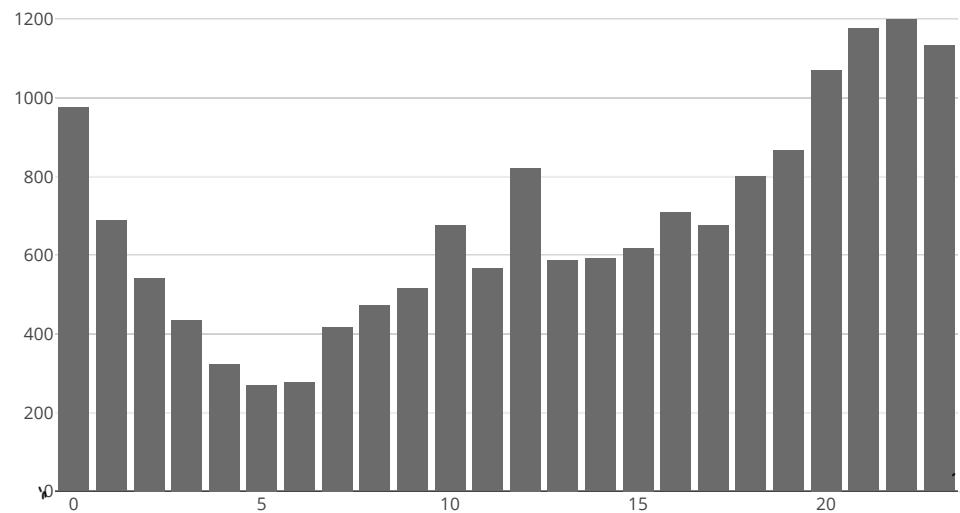
```
In [110]: datos["hora"] = [t.hour for t in pd.DatetimeIndex(datos.hora)]
datos.to_csv('vf_p4.csv', header=True, index=False)
```

```
In [111]: x = datos['hora'].value_counts()
values = x.values
a = x.to_frame().T

data = [go.Bar(
    x = a.columns.values,
    y = values,
    marker=dict(
        color='rgb(107, 107, 107)'))
]

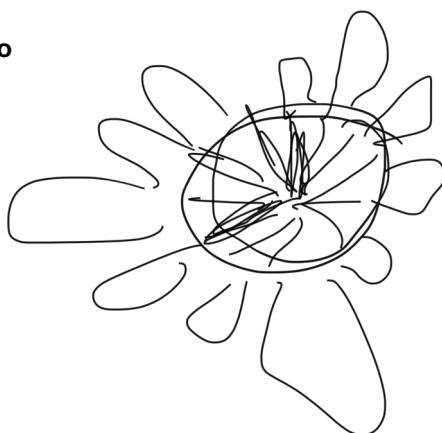
py.iplot(data, filename='horas')
```

Out[111]:



[EDIT CHART](#)

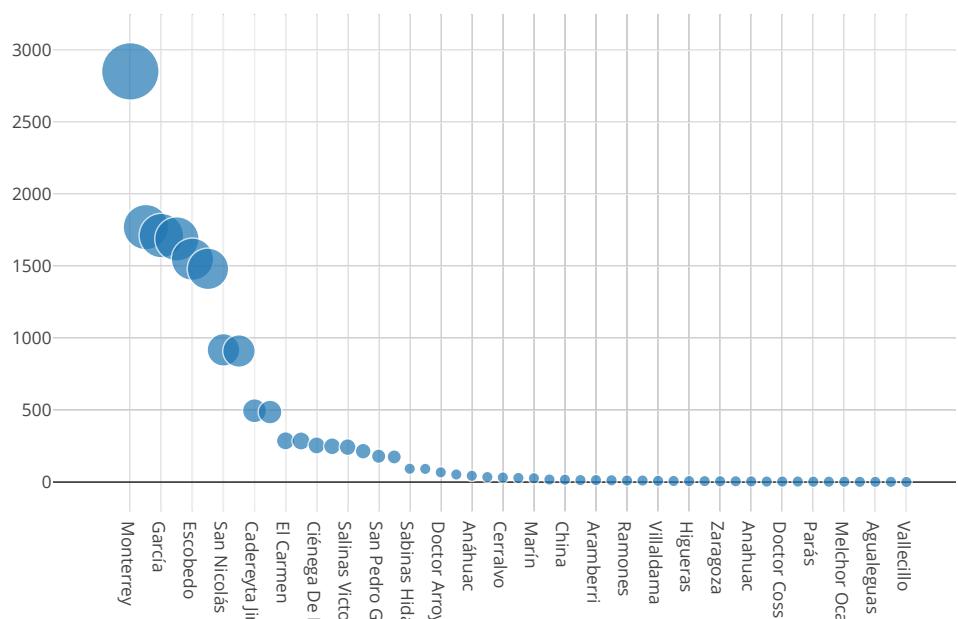
Incidentes por municipio



```
In [112]: x = datos['mpio'].value_counts()
values = x.values
a = x.to_frame().T

data = [go.Scatter(
    x = a.columns.values,
    y = values,
    mode='markers',
    marker=dict(
        size=values,
        sizemode='area',
        sizeref=2.*max(values)/(40.***2),
        sizemin=4
    )
)]
py.iplot(data, filename='mpio')
```

Out[112]:



[EDIT CHART](#)

El municipio de Monterrey fue el que reportó la mayor cantidad de incidentes durante el año, le siguen Guadalupe, García, Apodaca y Escobedo.

Práctica 5: NP

Ciencia_de_Datos (/github/Saphira3000/Ciencia_de_Datos/tree/master)
/ p5.ipynb (/github/Saphira3000/Ciencia_de_Datos/tree/master/p5.ipynb)

Práctica 5: Pruebas estadísticas

Gabriela Sánchez Y.

En su quinto reporte, apliquen por lo menos tres diferentes tipos de pruebas estadísticas de hipótesis, argumentando por qué las pruebas con aplicables en cada caso. Pueden ser las mismas de los ejemplos o algunas otras (hay docenas si no cientos disponibles).

El objetivo de la práctica es aplicar diferentes tipos de pruebas estadísticas de hipótesis a las variables de estudio que se piensa que causan diferencia en los resultados. Primero se revisa cómo es la distribución de los datos ya que varias pruebas estadísticas dependen de la distribución de los mismos.

Se cargan los datos.

In [2]: `import pandas as pd
datos = pd.read_csv("vf_datos.csv")
datos.head()`

Out[2]:

	mes	mpio	fecha	hora	victima	sexo_v	edad_v	parentesco	edo_civil_v	ocupacion
0	1	9	2018/01/01	23	1	F	46	UNION LIBRE	UNIÓN LIBRE	EMPLE/
1	1	9	2017/12/27	8	1	F	26	CONCUBINO (A)	UNIÓN LIBRE	HO
2	1	9	2018/01/01	11	1	F	42	NINGUNO	UNIÓN LIBRE	EMPLE/
3	1	12	2017/12/22	11	1	F	26	CONCUBINO (A)	UNIÓN LIBRE	HO
4	1	12	2017/12/25	19	1	F	38	ESPOSO (A)	CASADO	HO

Hora

Se analiza si la hora influye en el número de incidentes. Hay que determinar si los datos siguen una distribución normal. Primero se grafica la información en un histograma para tener una idea remota de si la información sigue o no una distribución normal y después se aplica una prueba de normalidad de Shapiro-Wilk.

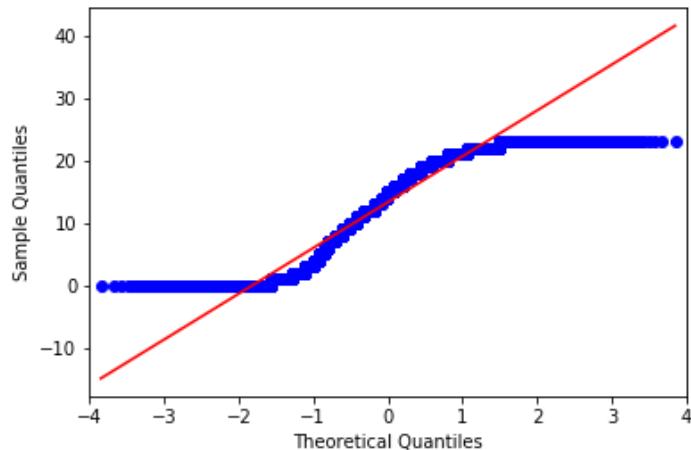
```
In [3]: import matplotlib.pyplot as plt
from scipy.stats import shapiro

hora = datos.hora
plt.title('Incidentes por hora')
plt.hist(hora, edgecolor = 'black', linewidth=1, bins = 24)
plt.savefig('horas.eps')
```

Si se inicia la curva a partir de las 5:00 horas la curva parecería tener una distribución normal aunque con dos picos en las 11:00 y 13:00 horas.

Se realiza un gráfico de cuantil-cuantil y se realiza la prueba de normalidad.

```
In [4]: from statsmodels.graphics.gofplots import qqplot
qqplot_data = qqplot(hora, line='s').gca().lines
```



```
In [5]: from scipy.stats import shapiro
import plotly.figure_factory as ff
import plotly.plotly as py
stat, p = shapiro(hora)

# interpret
alpha = 0.05
if p > alpha:
    msg = 'Sample looks Gaussian (fail to reject H0)'
else:
    msg = 'Sample does not look Gaussian (reject H0)'

result_mat = [
    ['Length of the sample data', 'Test Statistic', 'p-value', 'C',
     [len(hora), stat, p, msg]]
]

swt_table = ff.create_table(result_mat)
swt_table['data'][0].colorscales=[[0, '#2a3f5f'],[1, '#ffffff']]
swt_table['layout']['height']=200
swt_table['layout']['margin']['t']=50
swt_table['layout']['margin']['b']=50

py.iplot(swt_table, filename='shapiro-wilk-table')

/usr/local/lib/python3.5/dist-packages/scipy/stats/morestats.py:104:
p-value may not be accurate for N > 5000.

/usr/local/lib/python3.5/dist-packages/IPython/core/display.py:681:
Consider using IPython.display.IFrame instead
```

Out[5]:

Length of the sample data	Test Statistic	p-value
16410	0.9186199903488159	0.0

[EDIT CHART](#)

El gráfico cuantil-cuantil permite observar que los datos no parecen seguir una distribución normal, resultado que se verifica con lo obtenido con la prueba de normalidad.

Meses

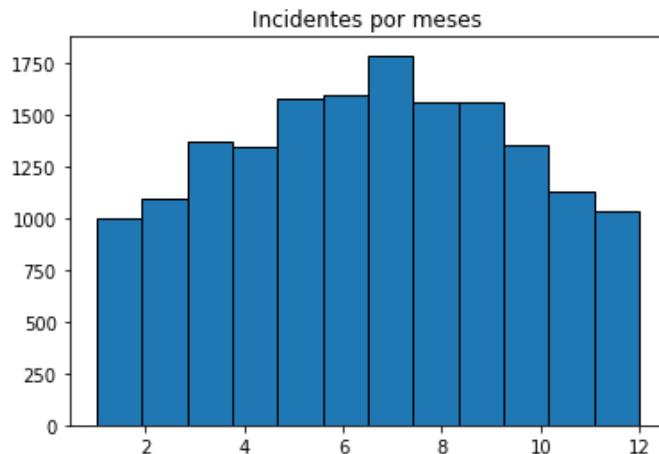
Ahora se revisa la normalidad de los datos correspondientes a los meses. Primero se grafica un histograma para tener una idea de lo que sucede con los datos.

```
In [13]: mes = datos.mes.value_counts()
plt.title('Incidentes por meses')
plt.hist(datos.mes, edgecolor = 'black', linewidth=1, bins = 12)
plt.savefig('mes.eps')

meses = datos.mes
stat, p = shapiro(meses)
print('Statistics=%.3f, p=%.3f' % (stat, p))
# interpret
alpha = 0.05
if p > alpha:
    print('Normal')
else:
    print('NO normal')

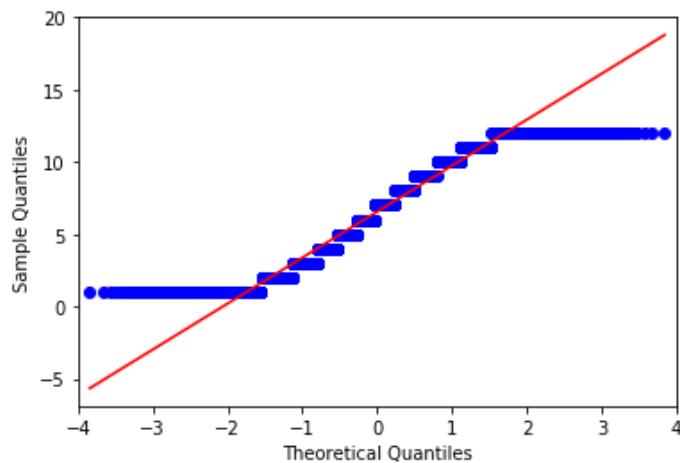
Statistics=0.959, p=0.000
NO normal

/usr/local/lib/python3.5/dist-packages/scipy/stats/morestats.py:105:
UserWarning: p-value may not be accurate for N > 5000.
```



La prueba indica que los datos no siguen una distribución normal, esto puede observarse en el gráfico cuantil-cuantil.

```
In [14]: qqplot_data = qqplot(meses, line='s').gca().lines
```



Finalmente, se revisa la normalidad de los datos correspondientes a las edades de las víctimas y agresores.

Para esto primero se grafica el histograma correspondiente, después el gráfico cuantil-cuantil y por último se realiza la prueba de normalidad.

Los datos no se encuentran en el formato adecuado por lo que primero se procede a realizar la limpieza correspondiente.

```
In [8]: edadv = datos.edad_v # edades de las victimas
edad_v = []
for dato in edadv:
    s = str(dato).replace(", ", " ")
    pedazos = s.split()
    while "a" in pedazos:
        pos = pedazos.index("a")
        desde = int(pedazos[pos - 1])
        hasta = int(pedazos[pos + 1])
        prom = (desde + hasta) // 2
        edad_v.append(prom)
        pedazos = pedazos[: (pos - 1)] + pedazos[(pos + 2):]
    edad_v += pedazos
edad_v = list(filter(lambda dato: dato != "NE", edad_v))
edad_v = list(filter(lambda dato: dato != "nan", edad_v))
edad_v = [float(v) for v in edad_v]
edad_v = [int(v) for v in edad_v]

edad_a = datos.edad_a
edad_a = []
for dato in edad_a:
    s = str(dato).replace(", ", " ")
    pedazos = s.split()
    while "a" in pedazos:
        pos = pedazos.index("a")
        desde = int(pedazos[pos - 1])
        hasta = int(pedazos[pos + 1])
        prom = (desde + hasta) // 2
        edad_a.append(prom)
        pedazos = pedazos[: (pos - 1)] + pedazos[(pos + 2):]
edad_a += pedazos

edad_a = list(filter(lambda dato: dato != "NE", edad_a))
edad_a = list(filter(lambda dato: dato != "nan", edad_a))
edad_a = [float(v) for v in edad_a]
edad_a = [int(v) for v in edad_a]
```

```
In [9]: edad_a = [0 if (v >= 0 and v<=10) else 10 if (v>10 and v<=20) else  
               30 if (v>30 and v<=40) else 40 if (v>40 and v<=50)  
               else 60 if (v>60 and v<=70) else 70 if (v>70 and v<=80)  
               else 90 if (v>90 and v<=100) else 100 for v in edad_a]  
  
edad_v = [0 if (v >= 0 and v<=10) else 10 if (v>10 and v<=20) else  
               30 if (v>30 and v<=40) else 40 if (v>40 and v<=50)  
               else 60 if (v>60 and v<=70) else 70 if (v>70 and v<=80)  
               else 90 if (v>90 and v<=100) else 100 for v in edad_v]  
di_edadA = {}  
di_edadV = {}  
  
from collections import Counter  
  
conteos = Counter([int(d) for d in edad_a])  
for edad in conteos:  
    di_edadA[edad] = conteos[edad]  
print(di_edadA)  
  
conteos2 = Counter([int(d) for d in edad_v])  
for edad in conteos2:  
    di_edadV[edad] = conteos2[edad]  
print(di_edadV)  
  
{80: 15, 50: 1034, 20: 5004, 70: 93, 40: 3217, 10: 720, 60: 248,  
{0: 922, 80: 103, 50: 1010, 20: 5229, 70: 273, 40: 2840, 10: 1910}
```

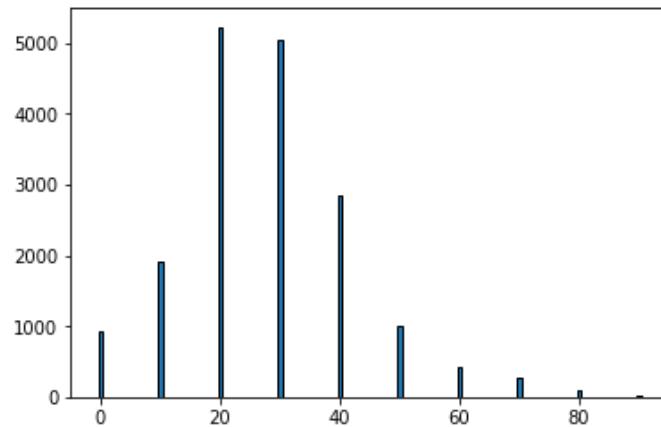
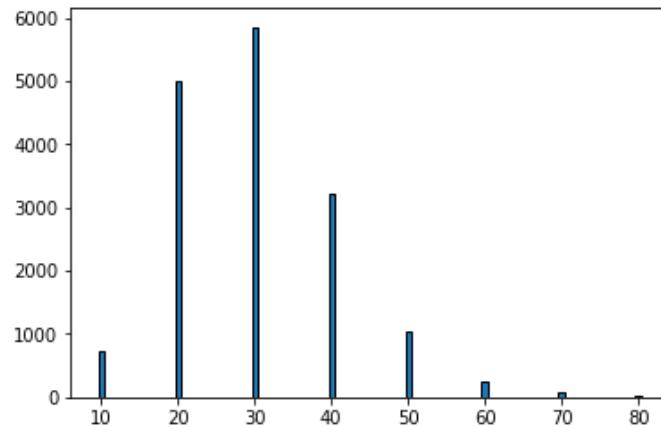
```
In [10]: xa = list(di_edadA.keys())
ya = list(di_edadA.values())

xv = list(di_edadV.keys())
yv = list(di_edadV.values())

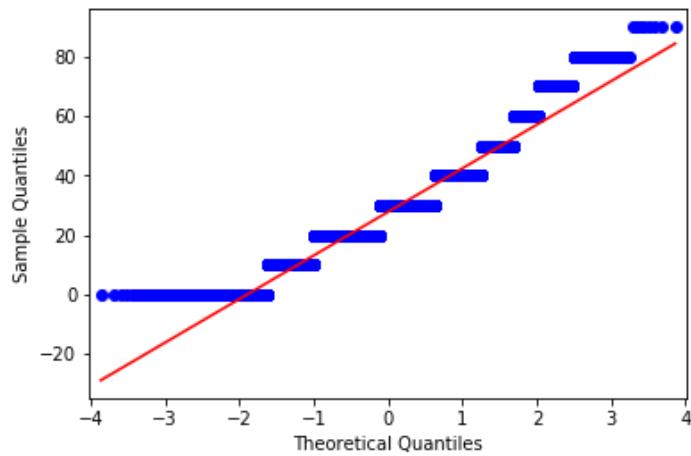
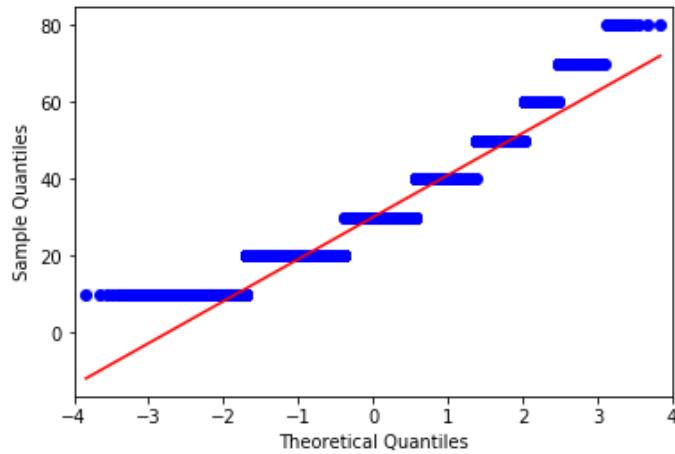
from matplotlib import pyplot

pyplot.figure()
pyplot.bar(xa, ya, edgecolor = 'black', linewidth=1)
pyplot.show()

pyplot.figure()
pyplot.bar(xv, yv, edgecolor = 'black')
pyplot.show()
```



```
In [11]: import numpy as np  
myarray = np.asarray(edad_a)  
qqplot_data = qqplot(myarray, line='s').gca().lines  
  
myarray2 = np.asarray(edad_v)  
qqplot_data = qqplot(myarray2, line='s').gca().lines
```



```
In [12]: print("Edad víctimas")
stat, p = shapiro(yv)
print('Statistics=% .3f, p=% .3f' % (stat, p))
# interpret
alpha = 0.05
if p > alpha:
    print('Normal')
else:
    print('NO normal')

print('-----')
print("Edad agresores")
stat, p = shapiro(ya)
print('Statistics=% .3f, p=% .3f' % (stat, p))
# interpret
alpha = 0.05
if p > alpha:
    print('Normal')
else:
    print('NO normal')
```

```
Edad víctimas
Statistics=0.817, p=0.024
NO normal
-----
Edad agresores
Statistics=0.820, p=0.047
NO normal
```

Se rechaza la hipótesis de normalidad ya que el *p*-valor en ambos casos es menor que el estadístico de prueba.

Práctica 6: Modelos lineales con scipy.stats

Gabriela Sánchez Y.

El objetivo de la práctica es emplear modelos lineales para representar una variable de interés como una función de uno o más factores conocidos.

Anteriormente se realizaron pruebas de normalidad a algunos datos. Se encontró que los datos en meses parecen seguir una distribución normal. Ahora se emplea un modelo lineal para determinar cómo afecta.

Para comenzar a trabajar, se cargan los datos.

```
In [79]: import pandas as pd
datos = pd.read_csv("/home/saphira/Desktop/datos/vf_p5.csv")
```

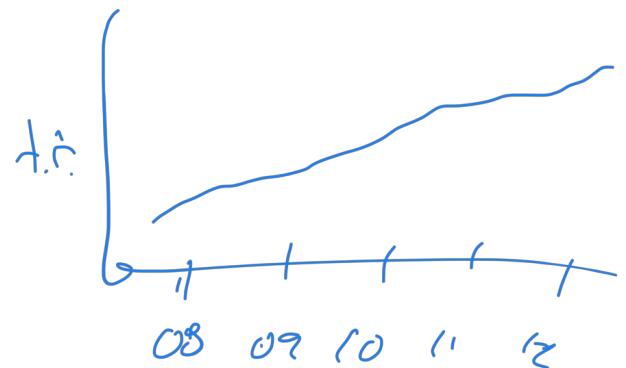
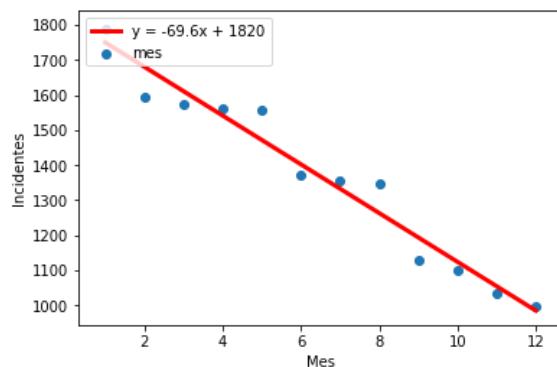
```
In [80]: import matplotlib.pyplot as plt
from scipy import stats

# intento modelar los incidentes en función de los meses
y = datos.mes.value_counts() # lo que se modela
x = datos.mes.unique() # en función de que

a, b, r, p, e = stats.linregress(x, y)
print("y = f(x) = {:.4f} x + {:.4f}".format(a, b))
print("error", e)
print("valor p", p)
print("pendiente {:s}significativo".format("no " if p >= 0.05 else ""))
print("R^2", r**2)

plt.plot(x, (a * x + b), label = 'y = {:.1f}x + {:.0f}'.format(a, b), color = 'red', linewidth = 3)
plt.scatter(x, y)
plt.legend(loc='upper left')
plt.xlabel("Mes")
plt.ylabel("Incidentes")
plt.show()
```

```
y = f(x) = -69.6084 x + 1819.9545
error 4.713696004307858
valor p 4.062837034855093e-08
pendiente significativo
R^2 0.9561542114108509
```



El p-valor es menor que 0.05 lo que sugiere que los datos están relacionados. El coeficiente 1820 indica que para cada mes se espera que la cantidad de incidentes aumente en una media de 1820.

```
In [81]: # se intenta modelar los incidentes en funcion de la hora

y = datos.hora.value_counts() # lo que se modela
#x = datos.hora.unique() # en funcion de que
x = [0,1,2]
#x = x.replace("M", 0) # ocupan ser números / manana
#x = x.replace("T", 1) # tarde
#x = x.replace("N", 2) # noche

a, b, r, p, e = stats.linregress(x, y)
print("y = f(x) = {:.4f} x + {:.4f}".format(a, b))
print("error", e)
print("valor p", p)
print("pendiente {:s}significativo".format("no " if p >= 0.05 else ""))
print("R^2", r**2)

y = f(x) = -1231.5000 x + 6701.5000
error 388.84540629921287
valor p 0.19470534841790751
pendiente no significativo
R^2 0.9093408174755501
```

El p-valor es mayor que el nivel de significancia 0.05 lo que indica que estadísticamente no hay relación significativa entre el horario y el número de incidentes.

Ahora, se analiza la relación del estado civil de las víctimas con los incidentes. Primero se revisa normalidad.

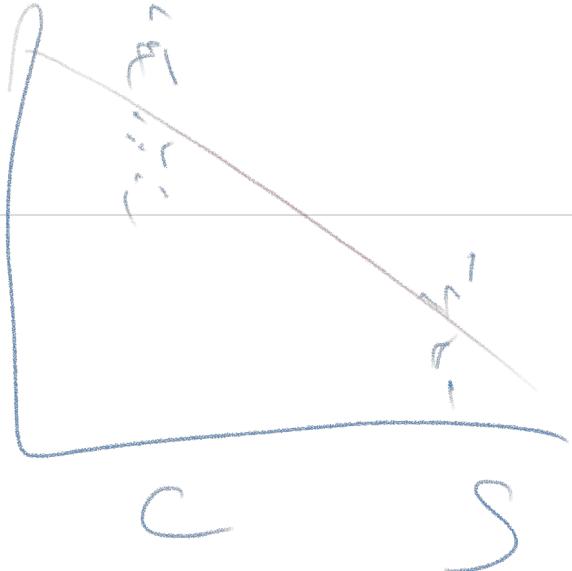
```
In [82]:edo_civil = datos.edo_civil
civil = []
for dato in edo_civil:
    s = str(dato).replace(", ", " ")
    s = str(s).replace("UNIÓN LIBRE", "UL")
    pedazos = s.split()
    civil += pedazos

civil = list(filter(lambda dato: dato != "NE", civil))
civil = list(filter(lambda dato: dato != "nan", civil))

from collections import Counter
conteos = Counter([d for d in civil])

from scipy.stats import shapiro
stat, p = shapiro(list(conteos.values()))
print('Statistics={:.3f}, p={:.3f}'.format(stat, p))
# interpret
alpha = 0.05
if p > alpha:
    print('Normal')
else:
    print('NO normal')

Statistics=0.858, p=0.184
Normal
```



```
In [83]: y = list(conteos.values()) # lo que se modela
x = [0,1,2,3,4,5] # en función de que
# separado/soltero/divorciado/viudo/ul/casado

a, b, r, p, e = stats.linregress(x, y)
print("y = f(x) = {:.4f} x + {:.4f}".format(a, b))
print("error", e)
print("valor p", p)
print("pendiente {:s}significativo".format("no " if p >= 0.05 else ""))
print("R^2", r**2)

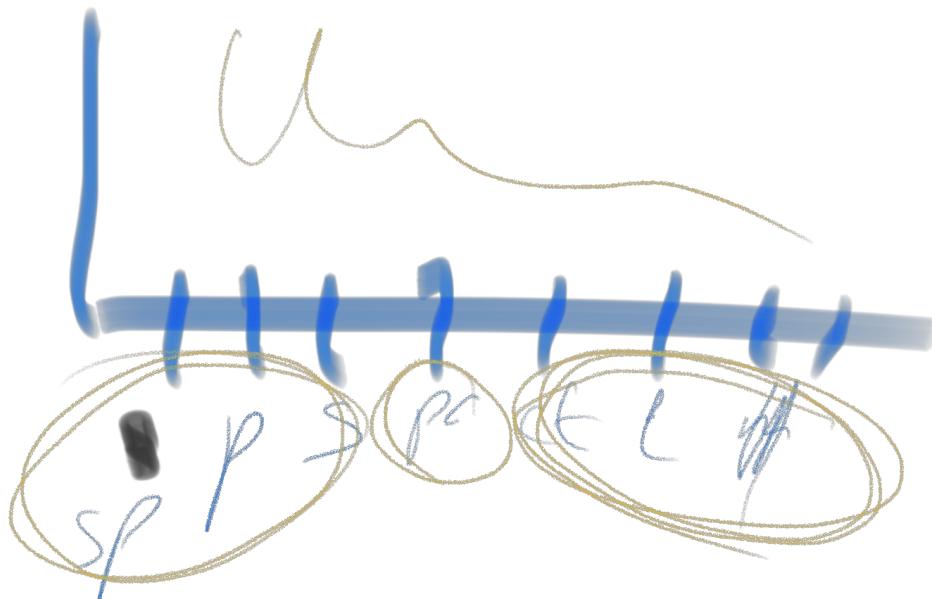
plt.plot(x, (a * x + b), label = 'y = {:.1f}x + {:.0f}'.format(a, b), color = 'red', linewidth = 3)
plt.scatter(x, y)
plt.legend(loc='upper left')
plt.xlabel("Edo Civil")
plt.ylabel("Incidentes")
plt.show()

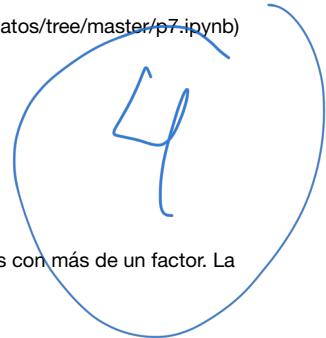
y = f(x) = 1227.7143 x + -107.2857
error 550.7029867286329
valor p 0.0896747944201482
pendiente no significativo
R^2 0.55407103435543
```

```
-----
TypeError                                 Traceback (most recent call last)
<ipython-input-83-70812f853a0a> in <module>
    10 print("R^2", r**2)
    11
--> 12 plt.plot(x, (a * x + b), label = 'y = {:.1f}x + {:.0f}'.format(a, b), color = 'red', linewidth = 3)
    13 plt.scatter(x, y)
    14 plt.legend(loc='upper left')

TypeError: 'numpy.float64' object cannot be interpreted as an integer
```

Los datos parecen seguir una distribución normal. Estadísticamente la pendiente no es significativa con un p-valor de 0.0896, lo que indica que no hay relación en el estado civil de las víctimas y los incidentes reportados de violencia.





Práctica 7: Regresión múltiple con statsmodels

Gabriela Sánchez Y.

El objetivo de la práctica es extender el análisis realizado previamente, ahora se analizan modelos lineales con más de un factor. La variable de interés es la cantidad de incidentes reportados de violencia.

Se revisa primero el efecto combinado que tiene el mes y el horario de los incidentes.

```
In [227]: import matplotlib.pyplot as plt
import statsmodels.api as sm
from numpy import isnan
import pandas as pd
import ssl

if getattr(ssl, '_create_unverified_context', None):
    ssl._create_default_https_context = ssl._create_unverified_context

datos = pd.read_csv("/home/saphira/Desktop/datos/vf_p5.csv")
datos.hora.replace(to_replace = dict(M = 0, T = 1, N = 2), inplace = True)
datos['cuantos'] = 1
```

```
In [228]: a = datos.groupby(['mes', 'hora'], as_index=False).agg({"cuantos": "sum"})

y = a['cuantos'] # variable dependiente

x = a[['mes', 'hora']] # variables independientes
x = sm.add_constant(x)

m = sm.OLS(y, x).fit()
print(m.summary())
```

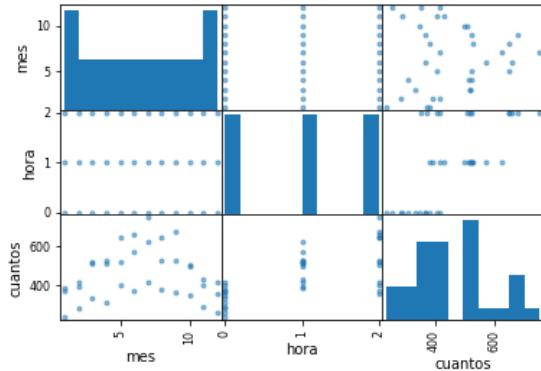
```
OLS Regression Results
=====
Dep. Variable: cuantos R-squared: 0.450
Model: OLS Adj. R-squared: 0.417
Method: Least Squares F-statistic: 13.50
Date: Mon, 25 Mar 2019 Prob (F-statistic): 5.19e-05
Time: 02:21:01 Log-Likelihood: -214.24
No. Observations: 36 AIC: 434.5
Df Residuals: 33 BIC: 439.2
Df Model: 2
Covariance Type: nonrobust
=====

            coef  std err      t      P>|t|      [0.025      0.975]
-----
const    339.7992   39.790     8.540     0.000    258.847    420.752
mes       2.0629    4.688     0.440     0.663     -7.475     11.601
hora     102.6250   19.820     5.178     0.000     62.301    142.949
=====
Omnibus:          0.461 Durbin-Watson:        0.759
Prob(Omnibus):    0.794 Jarque-Bera (JB):    0.560
Skew:             -0.235 Prob(JB):         0.756
Kurtosis:          2.608 Cond. No.           19.1
=====

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Los resultados del modelo indican que las variables independientes no describen muy bien a la variable dependiente, ya que el valor de *R cuadrado* es 0.450. Además los resultados del *t-test* indican que la hora se relaciona de forma significativa con el número de casos registrados.

```
In [229]: pd.scatter_matrix(a)
plt.show()
```



Se revisa además el efecto de la edad, el estado civil y el sexo de las víctimas en el número de incidentes reportados.

```
In [230]: edo_civil = datos.edo_civil
civil = []
for dato in edo_civil:
    s = str(dato).replace(", ", " ")
    s = str(s).replace("UNIÓN LIBRE", "UL")
    pedazos = s.split()
    pedazos = ["SOLTERO" if (v == "DIVORCIADO" or v == "SEPARADO" or v == "VIUDO" or v == "SOLTERO") else "CASADO" if (v == "UL" or v == "CASADO") else "NE" for v in pedazos]
    civil += pedazos

edadv = datos.edad_v # edades de las victimas
edad_v = []
for dato in edadv:
    s = str(dato).replace(", ", " ")
    pedazos = s.split()
    while "a" in pedazos:
        pos = pedazos.index("a")
        desde = int(pedazos[pos - 1])
        hasta = int(pedazos[pos + 1])
        prom = (desde + hasta) // 2
        edad_v.append(prom)
        pedazos = pedazos[: (pos - 1)] + pedazos[(pos + 2):]
    edad_v += pedazos

sexo_v = datos.sexo_v
sexo = []
for dato in sexo_v:
    s = str(dato).replace(", ", " ")
    pedazos = s.split()
    sexo += pedazos

b = pd.DataFrame({'edad': edad_v, 'edo_civil': civil, 'sexo': sexo})
b['cuantos'] = 1
b.edo_civil.replace(to_replace = dict(SOLTERO = 0, CASADO = 1), inplace = True)
b.sexo.replace(to_replace = dict(M = 0, F = 1), inplace = True)

c = b.groupby(['edad', 'edo_civil', 'sexo'], as_index=False).agg({"cuantos": "sum"})
c = c[c.edad != 'NE']
```

```

c['edad'] = [int(t) for t in c.edad]
c = c[c.edo_civil != 'NE']
#c = c[c.edo_civil != NaN]
c['edo_civil'] = [int(t) for t in c.edo_civil]

##### modelo
y = c['cuantos'] # variable dependiente

x = c[['edad','edo_civil','sexo']] # variables independientes
x = sm.add_constant(x)

m = sm.OLS(y, x).fit()
print(m.summary())

```

OLS Regression Results

Dep. Variable:	cuantos	R-squared:	0.284			
Model:	OLS	Adj. R-squared:	0.278			
Method:	Least Squares	F-statistic:	46.87			
Date:	Mon, 25 Mar 2019	Prob (F-statistic):	1.57e-25			
Time:	02:21:02	Log-Likelihood:	-2070.2			
No. Observations:	358	AIC:	4148.			
Df Residuals:	354	BIC:	4164.			
Df Model:	3					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
const	38.5281	10.274	3.750	0.000	18.322	58.734
edad	-1.0005	0.162	-6.180	0.000	-1.319	-0.682
edo_civil	56.7665	8.375	6.778	0.000	40.295	73.238
sexo	63.3021	8.351	7.580	0.000	46.879	79.725
Omnibus:	161.152	Durbin-Watson:			2.006	
Prob(Omnibus):	0.000	Jarque-Bera (JB):			685.804	
Skew:	1.964	Prob(JB):			1.20e-149	
Kurtosis:	8.527	Cond. No.			150.	

Warnings:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

El resultado para el *R cuadrado* indica que el ajuste del modelo no es bueno. Sin embargo, los resultados en la significancia del *t-test* sugieren que todas las variables independientes se relacionan de forma significativa con el número de incidentes.

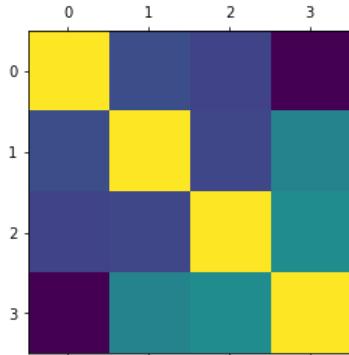
El análisis del coeficiente beta expresa que la relación entre los casos reportados y la edad de la víctima es negativa, esto es, el número de incidentes de violencia familiar disminuyen para víctimas de mayor edad.

Hay que revisar la correlación entre las variables independientes.

```
In [231]: print(c.corr())
plt.matshow(c.corr())
```

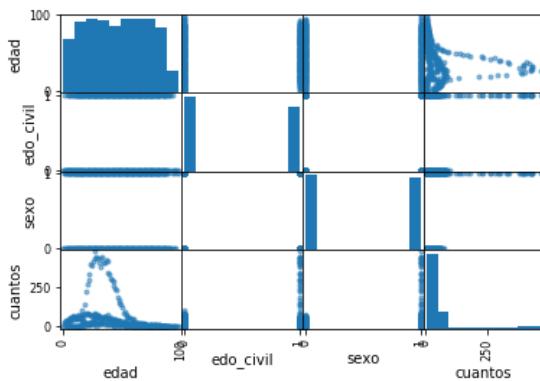
	edad	edo_civil	sexo	cuantos
edad	1.000000	0.033120	-0.018708	-0.274371
edo_civil	0.033120	1.000000	-0.000814	0.295435
sexo	-0.018708	-0.000814	1.000000	0.345863
cuantos	-0.274371	0.295435	0.345863	1.000000

```
Out[231]: <matplotlib.image.AxesImage at 0x7ff01c634588>
```



Parece no haber una relación fuerte entre las variables independientes.

```
In [232]: pd.scatter_matrix(c)
plt.show()
```



Práctica 8: Análisis de varianza y de componentes principales

Gabriela Sánchez Y.

El objetivo de la práctica es llevar a cabo un análisis de varianza (ANOVA) y un análisis de componentes principales (PCA) a los datos con los que se ha estado trabajando, visualizar los resultados y discutirlos.

La variable de interés es la cantidad de incidentes de violencia familiar, ya que se desea encontrar al grupo más vulnerable.

Se ha realizado un preprocesamiento de los datos para incluir solamente algunas características de las víctimas tales como la edad, el estado civil, escolaridad, hora, mes y lugar donde ocurrió el indicente y el parentesco de la víctima con el agresor. Se han eliminado los incidentes para los cuales no se especifica alguno de los datos antes mencionados.

```
In [179]: import pandas as pd
pd.set_option('max_rows', 10)

datos = pd.read_csv("victimas.csv")
datos = datos.drop('Unnamed: 0',1)
datos
```

Out[179]:

	edad	edo_civil	escolaridad	hora	mes	mpio	parentesco	cuantos
0	46	CASADO	SECUNDARIA	N	1	CADEREYTA JIMÉNEZ	UL	1
1	26	CASADO	SECUNDARIA	M	1	CADEREYTA JIMÉNEZ	CONCUBINO	1
2	42	CASADO	PREPARATORIA	M	1	CADEREYTA JIMÉNEZ	NINGUNO	1
3	26	CASADO	PREPARATORIA	M	1	CIÉNEGA DE FLORES	CONCUBINO	1
4	38	CASADO	SECUNDARIA	T	1	CIÉNEGA DE FLORES	ESPOSO	1
...
17036	48	CASADO	PRIMARIA	M	12	SALINAS VICTORIA	NINGUNO	1
17037	32	CASADO	PREPARATORIA	N	12	SALINAS VICTORIA	NINGUNO	1
17038	21	CASADO	PREPARATORIA	M	12	SALINAS VICTORIA	CONCUBINO	1
17039	32	CASADO	SECUNDARIA	T	12	SALINAS VICTORIA	ESPOSO	1
17040	12	SOLTERO	SECUNDARIA	M	12	SALINAS VICTORIA	NINGUNO	1

17041 rows × 8 columns

```
In [160]: datos.escolaridad.unique()
```

```
Out[160]: array(['SECUNDARIA', 'PREPARATORIA', 'PRIMARIA', 'NINGUNO',
       'LICENCIATURA', 'TÉCNICO', 'POSGRADO'], dtype=object)
```

Para no trabajar con tantos niveles, se agrupan las edades en bloques de 10.

W

```
In [180]: datos.edad = [0 if (v >= 0 and v <= 9) else 1 if (v >= 10 and v <= 19) else 2 if (v >= 20 and v <= 29)
                    else 3 if (v >= 30 and v <= 39) else 4 if (v >= 40 and v <= 49) else 5 if (v >= 50 and
v <= 59)
                    else 6 if (v >= 60 and v <= 69) else 7 if (v >= 70 and v <= 79) else 8 if (v >= 80 and
v <= 89)
                    else 9 for v in datos.edad]

datos.escolaridad = [0 if v == "NINGUNO" else 1 if v == "PRIMARIA" else 2 if v == "SECUNDARIA"
                     else 3 if v == "PREPARATORIA" else 4 if v == "TÉCNICO" else 5 if v == "LICENCIA
TURA"
                     else 6 for v in [str(v) for v in datos.escolaridad]]
```

```
In [182]: datos.hora.replace(to_replace = dict(M = 0, T = 1, N = 2), inplace = True)
datos.edo_civil.replace(to_replace = dict(SOLTERO = 0, CASADO = 1), inplace = True)

datos['hora'] = [int(t) for t in datos.hora]
datos['edo_civil'] = [int(t) for t in datos.edo_civil]
datos['escolaridad'] = [int(t) for t in datos.escolaridad]

c = datos.groupby(['edad', 'edo_civil', 'escolaridad', 'hora', 'mes'], as_index=False).agg({"cuantos
": "sum"})
```

Los datos no son balanceados. Se prueba un primer modelo con un ANOVA tipo 2, en el cual no se consideran interacciones entre los distintos factores.

```
In [183]: import statsmodels.api as sm
from statsmodels.formula.api import ols
import ssl

if getattr(ssl, '_create_unverified_context', None):
    ssl._create_default_https_context = ssl._create_unverified_context

m = ols('cuantos ~ edad + edo_civil + escolaridad + hora + mes', data = c).fit()
a = sm.stats.anova_lm(m, typ = 2)

print(a)
n = len(a)
alpha = 0.05
for i in range(n):
    print("{}:{} es significativo".format(a.index[i], "" if a['PR(>F)'][i] < alpha else "NO"))
```

	sum_sq	df	F	PR(>F)
edad	28103.315009	1.0	213.730001	2.858234e-46
edo_civil	30232.123042	1.0	229.919911	1.767271e-49
escolaridad	5895.187520	1.0	44.833801	2.712031e-11
hora	3115.409120	1.0	23.693162	1.209809e-06
mes	140.921029	1.0	1.071726	3.006693e-01
Residual	290197.987725	2207.0	Nan	Nan

edad es significativo
edo_civil es significativo
escolaridad es significativo
hora es significativo
mes NO es significativo
Residual NO es significativo

Parece ser que casi todos los factores son significativos para la variable de interés. Sin embargo, el modelo tiene algunos residuos son muy altos. Habrá que intentar otra cosa.

La siguiente prueba incluye interacciones entre los factores.

```
In [185]: m = ols('cuantos ~ edad * edo_civil + edad * hora + edad * escolaridad', data = c).fit()
a = sm.stats.anova_lm(m, typ = 2)
print(a)
n = len(a)
alpha = 0.05
for i in range(n):
    print("{}:{} es significativo".format(a.index[i], "" if a['PR(>F)'][i] < alpha else "NO"))

      sum_sq      df          F      PR(>F)
edad        28047.938691   1.0  218.876307  2.726351e-47
edo_civil   30247.078317   1.0  236.037623  1.103524e-50
edad:edo_civil  6916.724660   1.0   53.975701  2.843324e-13
hora        3020.502432   1.0   23.570945  1.288347e-06
edad:hora    645.207120   1.0    5.034971  2.493937e-02
escolaridad  6346.876698   1.0   49.528807  2.599533e-12
edad:escolaridad  500.301989   1.0    3.904182  4.829056e-02
Residual     282560.071260  2205.0           NaN           NaN
edad es significativo
edo_civil es significativo
edad:edo_civil es significativo
hora es significativo
edad:hora es significativo
escolaridad es significativo
edad:escolaridad es significativo
Residual NO es significativo
```

Los residuales siguen altos. Se revisan otras interacciones.

```
In [186]: m = ols('cuantos ~ edo_civil * hora + edo_civil * escolaridad + hora * escolaridad', data = c).fit()

a = sm.stats.anova_lm(m, typ = 2)
print(a)
n = len(a)
alpha = 0.05
for i in range(n):
    print("{}:{} es significativo".format(a.index[i], "" if a['PR(>F)'][i] < alpha else "NO"))

      sum_sq      df          F      PR(>F)
edo_civil    25510.639510   1.0  177.684314  4.819435e-39
hora         4042.701514   1.0   28.157845  1.229997e-07
edo_civil:hora  1177.105137   1.0    8.198662  4.231717e-03
escolaridad   5619.332279   1.0   39.139246  4.725898e-10
edo_civil:escolaridad  398.412854   1.0    2.774988  9.588936e-02
hora:escolaridad  105.131678   1.0    0.732253  3.922467e-01
Residual     316721.659781  2206.0           NaN           NaN
edo_civil es significativo
hora es significativo
edo_civil:hora es significativo
escolaridad es significativo
edo_civil:escolaridad NO es significativo
hora:escolaridad NO es significativo
Residual NO es significativo
```

De acuerdo a los modelos anteriores los factores de edad, estado civil, escolaridad y hora de los incidentes influyen en la variable de interés. Así como también influye la interacción entre algunos de ellos. Aunque aún hay mucha variación por explicar.

El siguiente paso es realizar un análisis de componentes principales. Para este análisis se consideran los factores de edad, mes, hora, escolaridad y edo_civil. Previamente estos datos se han modificado para tener datos numéricos.

```
In [202]: from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

features = ['cuantos', 'edad', 'escolaridad', 'edo_civil', 'mes']
x = c.loc[:, features].values
y = c.loc[:,['hora']].values
x = StandardScaler().fit_transform(x)

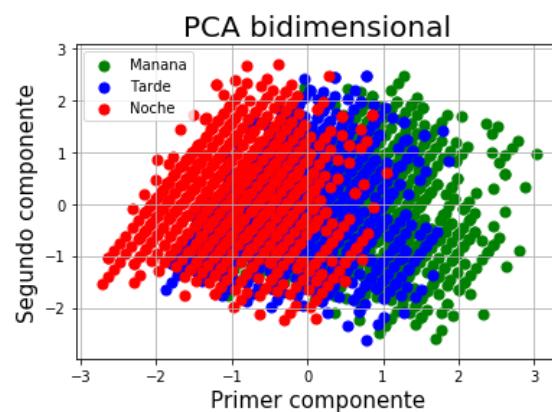
pca = PCA(n_components = 2)
cd = pd.DataFrame(data = pca.fit_transform(x), columns = [ 'comp_{:d}'.format(i) for i in range(k)])
cd['hora'] = y
display(cd.head(5))

pri = cd.hora == 0
seg = cd.hora == 1
ter = cd.hora == 2

plt.title('PCA bidimensional', fontsize = 20)
plt.xlabel('Primer componente', fontsize = 15)
plt.ylabel('Segundo componente', fontsize = 15)
plt.scatter(d.loc[pri].comp_0, d.loc[pri].comp_1, c = 'g', s = 50)
plt.scatter(d.loc[seg].comp_0, d.loc[seg].comp_1, c = 'b', s = 50)
plt.scatter(d.loc[ter].comp_0, d.loc[ter].comp_1, c = 'r', s = 50)
plt.legend(['Manana', 'Tarde', 'Noche'])
plt.grid()

/usr/local/lib/python3.5/dist-packages/sklearn/utils/validation.py:595: DataConversionWarning: Data with input dtype int64 was converted to float64 by StandardScaler.
    warnings.warn(msg, DataConversionWarning)
/usr/local/lib/python3.5/dist-packages/sklearn/utils/validation.py:595: DataConversionWarning: Data with input dtype int64 was converted to float64 by StandardScaler.
    warnings.warn(msg, DataConversionWarning)
```

	comp_0	comp_1	hora
0	0.257331	2.357384	0
1	0.498128	2.371462	0
2	0.320782	2.384512	0
3	0.681049	2.398883	0
4	0.862112	2.412814	0



De los resultados obtenidos se observa que, considerando dos componentes principales, los incidentes no pueden separarse de acuerdo a la hora de los hechos.

```
In [212]: features = ['cuantos', 'edad', 'edo_civil', 'hora', 'mes']
x = c.loc[:, features].values
y = c.loc[:,['escolaridad']].values
x = StandardScaler().fit_transform(x)
```

```

pca = PCA(n_components = 2)
cd = pd.DataFrame(data = pca.fit_transform(x), columns = [ 'comp_{:d}'.format(i) for i in range(k)])
cd[ 'escolaridad' ] = y
display(cd.head(5))

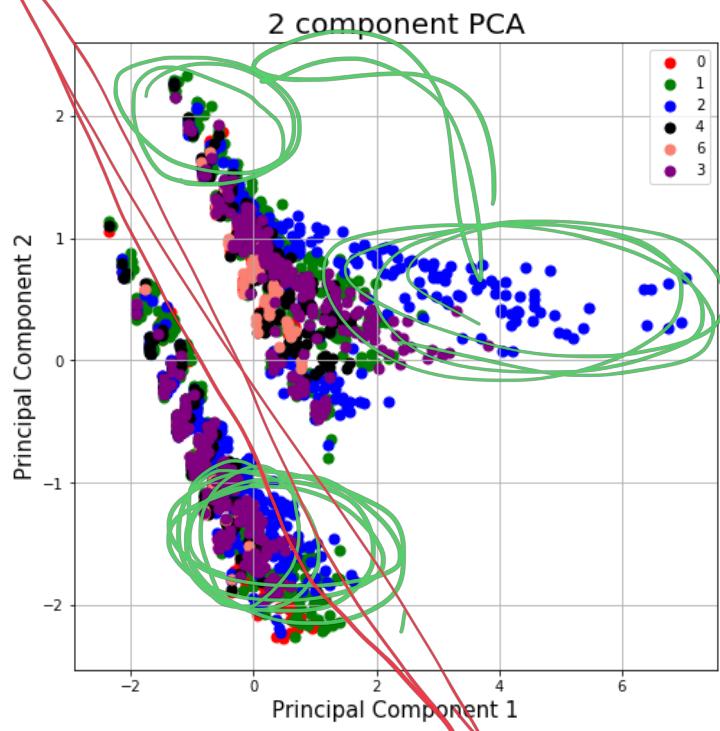
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA', fontsize = 20)

targets = cd.escolaridad.unique()
colors = ['r', 'g', 'b', 'black', 'salmon', 'purple']
for target, color in zip(targets,colors):
    indicesToKeep = cd[ 'escolaridad' ] == target
    ax.scatter(cd.loc[indicesToKeep, 'comp_0']
               , cd.loc[indicesToKeep, 'comp_1']
               , c = color
               , s = 50)
ax.legend(targets)
ax.grid()

/usr/local/lib/python3.5/dist-packages/sklearn/utils/validation.py:595: DataConversionWarning: Dat
a with input dtype int64 was converted to float64 by StandardScaler.
    warnings.warn(msg, DataConversionWarning)
/usr/local/lib/python3.5/dist-packages/sklearn/utils/validation.py:595: DataConversionWarning: Dat
a with input dtype int64 was converted to float64 by StandardScaler.
    warnings.warn(msg, DataConversionWarning)

```

	comp_0	comp_1	escolaridad
0	-0.310096	-1.719780	0
1	-0.078859	-1.720835	0
2	-0.249945	-1.747321	0
3	0.096243	-1.741110	0
4	0.270005	-1.745798	0



La división de los datos podría explicarse por el estado civil, solteros y casados. De cualquier forma no se pueden separar los datos de acuerdo a la escolaridad de la víctima.



Práctica 9: Pronósticos con statsmodels

Gabriela Sánchez Y.

El objetivo en esta práctica es aplicar la librería *statsmodels* para pronósticos que se puedan realizar con los datos que se han estado estudiando.

En este caso, se ha estado trabajando con datos de incidentes reportados de violencia familiar durante el año 2018 en el estado de Nuevo León. Se intenta predecir el número de incidentes reportados a partir de valores previos obtenidos en distintas escalas de tiempo, algunos pronósticos pueden ser:

- Determinar la cantidad de incidentes de violencia para el mes de diciembre, a partir de los incidentes reportados en los meses de enero a noviembre,
- Determinar el número de incidentes en la cuarta semana del mes de enero a partir de los incidentes reportados durante las primeras tres semanas,

act dec

entre otros.

El primer paso es cargar los datos.

```
In [75]: import pandas as pd  
datos = pd.read_csv("/home/saphira/Desktop/datos/vf_p4.csv")
```

Meses

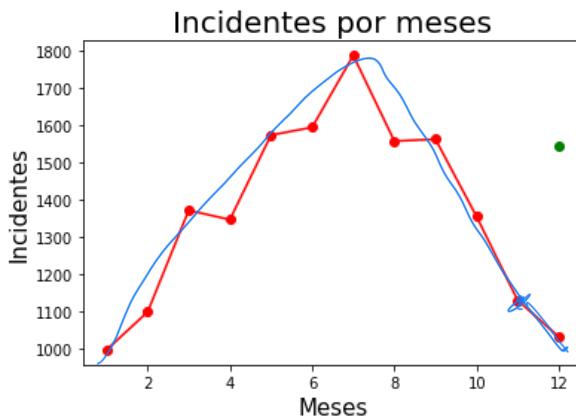
Se usa método de Holt para predecir el número de incidentes en el mes de diciembre a partir de los datos de los meses de enero a noviembre.

```
In [76]: from collections import Counter  
  
meses = datos.mes  
prev = list(filter(lambda dato: dato != 12, meses))  
x = Counter({d for d in prev})  
mes = pd.DataFrame.from_dict(x, orient='index', columns=['cuantos'])
```

```
In [77]: from numpy import asarray
from statsmodels.tsa.holtwinters import Holt
import matplotlib.pyplot as plt

x = Counter([d for d in meses])
f = Holt(asarray(mes.cuantos)).fit(smoothing_level = 0.1)
plt.title('Incidentes por meses', fontsize = 20)
plt.xlabel('Meses', fontsize = 15)
plt.ylabel('Incidentes', fontsize = 15)
plt.plot([12], f.forecast(1), 'go') # predicción
plt.plot([1,2,3,4,5,6,7,8,9,10,11,12], x.values(), 'r-o') # enero-dic
plt.show()
print('Real', x[12])
print('Pronóstico', int(f.forecast(1)))
```

1
10
11
2
3
4
5



Real 1032
Pronóstico 1545

La predicción de Holt (en verde) se aleja del resultado real. Intentar predecir en esta escala de tiempo tal vez no ayuda mucho, se revisa por semanas. No se usan métodos más sofisticados por el tamaño de la serie.

Semanas

```
In [78]: d = datos.fecha
d.head()
```

```
Out[78]: 0    2018/01/01
1    2017/12/27
2    2018/01/01
3    2017/12/22
4    2017/12/25
Name: fecha, dtype: object
```

Lo primero que se nota y que no se había revisado previamente, es que los datos contienen incidentes reportados en diciembre del año 2017. Para los próximos trabajos, éstos no se consideran.

Se usan los valores de las primeras 41 semanas y se predicen las últimas 10 del año, esto es, los incidentes de la semana 42 a la 52.

```
In [79]: datos.loc[:, 't'] = 1
datos.fecha = pd.to_datetime(pd.Series(datos.fecha))
```

```
In [80]: dates = pd.DataFrame()
dates['fecha'] = datos.fecha
dates.index = dates['fecha']
dates['valor'] = list(datos.t)
dates = dates.loc[dates.fecha.dt.year == 2018]
```

```
In [81]: a = dates.groupby(lambda x: x.weekofyear, as_index=False).agg({"valor": "sum"})
plt.title('Incidentes por semanas', fontsize = 20)
plt.xlabel('Semana', fontsize = 15)
plt.ylabel('Incidentes', fontsize = 15)
plt.plot(a.valor, 'b-o')
```

```
Out[81]: [<matplotlib.lines.Line2D at 0x7fad46ccf8d0>]
```



$\{a, b\}$

In [82]: $w = []$

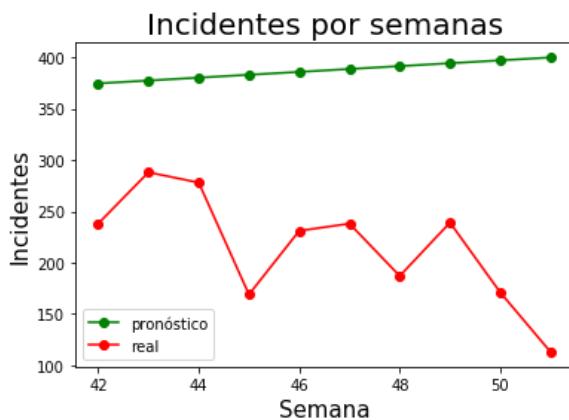
$w = \{i \text{ for } i \text{ in range(52)}$

```
w = []
for i in range(52):
    w.append(i)

f = Holt(asarray(a.valor[0:41])).fit(smoothing_level = 0.1)
plt.title('Incidentes por semanas', fontsize = 20)
plt.xlabel('Semana', fontsize = 15)
plt.ylabel('Incidentes', fontsize = 15)
plt.plot(w[42:52], f.forecast(10), 'g-o', label = 'pronóstico') # predicción
plt.plot(w[42:52], a.valor[42:52], 'r-o', label = 'real') # enero-dic
plt.legend(loc='best')

print('Reales', list(a.valor[42:52]))
print('Pronóstico', [int(d) for d in f.forecast(10)])
```

```
Reales [238, 288, 278, 169, 231, 238, 187, 239, 171, 113]
Pronóstico [374, 377, 380, 383, 385, 388, 391, 394, 397, 399]
```



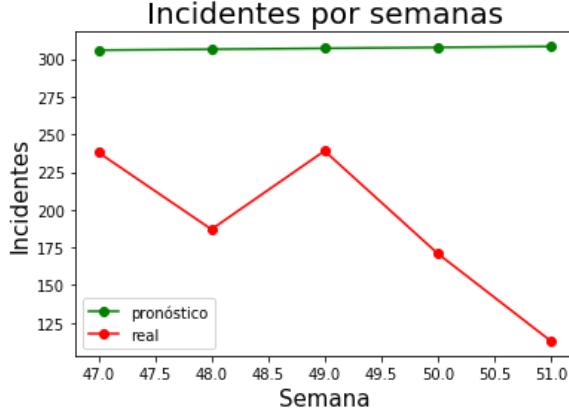
El pronóstico con el método de Holt no fue bueno, siempre es mayor que el valor real. Nótese que se eligió predecir el número de incidentes a partir de la semana 42, es justo en esta semana donde se presenta un pico en los datos.

Se intenta entonces, tomar los datos de las primeras 47 semanas para predecir las últimas cinco semanas del año.

```
In [83]: f = Holt(asarray(a.valor[0:46])).fit(smoothing_level = 0.1)
plt.title('Incidentes por semanas', fontsize = 20)
plt.xlabel('Semana', fontsize = 15)
plt.ylabel('Incidentes', fontsize = 15)
plt.plot(w[47:52], f.forecast(5), 'g-o', label = 'pronóstico') # predicción
plt.plot(w[47:52], a.valor[47:52], 'r-o', label = 'real') # enero-dic
plt.legend(loc='best')

print('Reales', list(a.valor[47:52]))
print('Pronóstico', [int(d) for d in f.forecast(5)])
```

Reales [238, 187, 239, 171, 113]
Pronóstico [305, 306, 307, 307, 308]



Dados los resultados, se prueba la predicción con un método más sofisticado, el modelo ARIMA. El primer paso es revisar que las series de tiempo sean estacionarias.

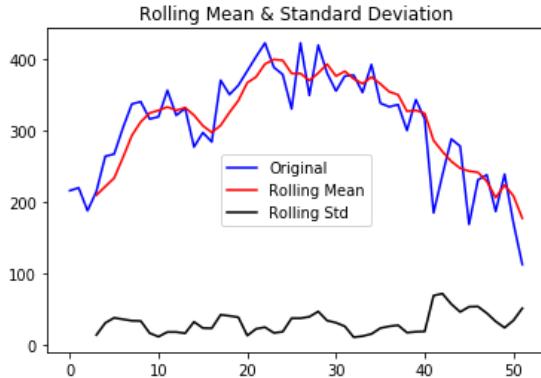
```
In [84]: from statsmodels.tsa.stattools import adfuller

# rutina de https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/
def test_stationarity(ts):
    #Determining rolling statistics
    rolmean = ts.rolling(4).mean()
    rolstd = ts.rolling(4).std()

    #Plot rolling statistics:
    orig = plt.plot(ts, color='blue',label='Original')
    mean = plt.plot(rolmean, color='red', label='Rolling Mean')
    std = plt.plot(rolstd, color='black', label = 'Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean & Standard Deviation')
    plt.show(block=False)

    #Perform Dickey-Fuller test:
    print ('Results of Dickey-Fuller Test:')
    dfoutput = pd.Series(adfuller(ts, autolag='AIC')[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])
    for key,value in dfoutput[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    print (dfoutput)

test_stationarity(a.valor)
```



```
Results of Dickey-Fuller Test:
Test Statistic           -0.713550
p-value                  0.843208
#Lags Used               2.000000
Number of Observations Used 49.000000
Critical Value (10%)      -2.599336
Critical Value (5%)       -2.922629
Critical Value (1%)        -3.571472
dtype: float64
```

El estadístico es mayor que los valores críticos, por lo que se puede concluir que no se cumple la estacionalidad, además esto se puede notar de la curva roja.

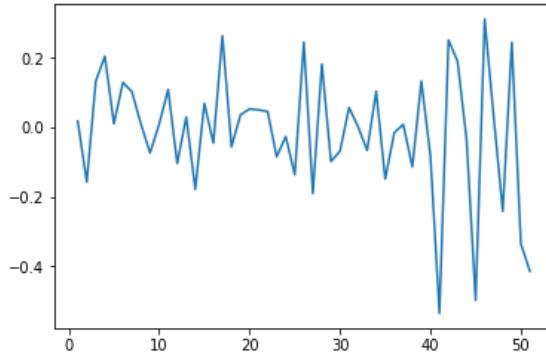
Se siguen las técnicas del [tutorial](https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/) (<https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/>) para lograr la estacionalidad.

De acuerdo con el tutorial, hay dos razones principales por las cuales una serie de tiempo no es estacionaria, una de ellas es la tendencia, es decir la variación de la media en el tiempo.

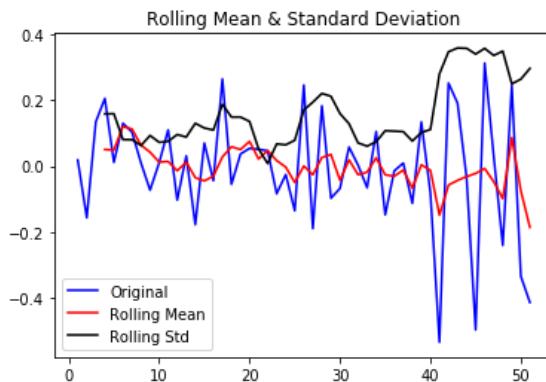
Así, lo primero que se prueba es estimarla y tratar de eliminarla. Uno de los trucos más simples para reducir la tendencia es una diferenciación. Esta técnica toma la diferencia de la observación en un instante particular con la del instante anterior.

```
In [85]: import numpy as np  
ts_log = np.log(a.valor)  
ts_log_diff = ts_log - ts_log.shift()  
plt.plot(ts_log_diff)
```

```
Out[85]: <matplotlib.lines.Line2D at 0x7fad46c120f0>
```



```
In [86]: ts_log_diff.dropna(inplace=True)  
test_stationarity(ts_log_diff)
```



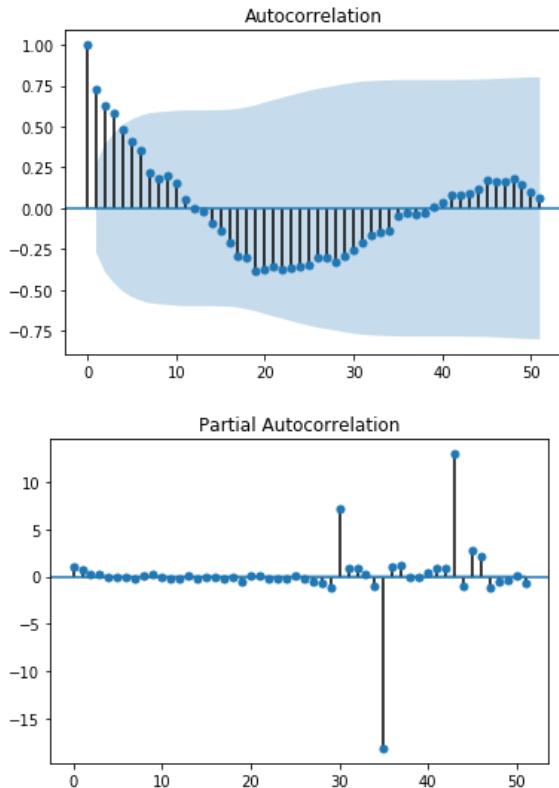
```
Results of Dickey-Fuller Test:  
Test Statistic          -6.822837e+00  
p-value                1.981762e-09  
#Lags Used             1.000000e+00  
Number of Observations Used 4.900000e+01  
Critical Value (10%)    -2.599336e+00  
Critical Value (5%)     -2.922629e+00  
Critical Value (1%)      -3.571472e+00  
dtype: float64
```

Con esto, aún hay variación en la media y la desviación estándar pero el *estadístico de prueba* es más pequeño que el 1% de los valores críticos por lo que se puede decir, con un 99% de confianza que se tiene una serie estacionaria.

```
In [87]: from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf
from matplotlib import pyplot

pyplot.figure()
plot_acf(a.valor, ax=pyplot.gca())
pyplot.show()

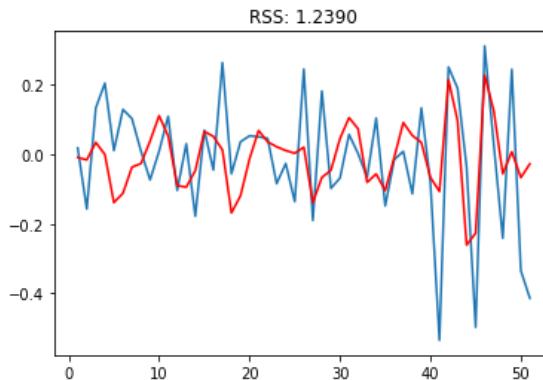
pyplot.figure()
plot_pacf(a.valor, ax=pyplot.gca())
pyplot.show()
```



De acuerdo a las gráficas, se obtiene que $q=6$ y $p=0$.

```
In [88]: from statsmodels.tsa.arima_model import ARIMA
model = ARIMA(ts_log, order=(0, 1, 6))
results_ARIMA = model.fit(disp=-1)
plt.plot(ts_log_diff)
plt.plot(results_ARIMA.fittedvalues, color='red')
plt.title('RSS: %.4f' % sum((results_ARIMA.fittedvalues-ts_log_diff)**2))
```

Out[88]: Text(0.5, 1.0, 'RSS: 1.2390')

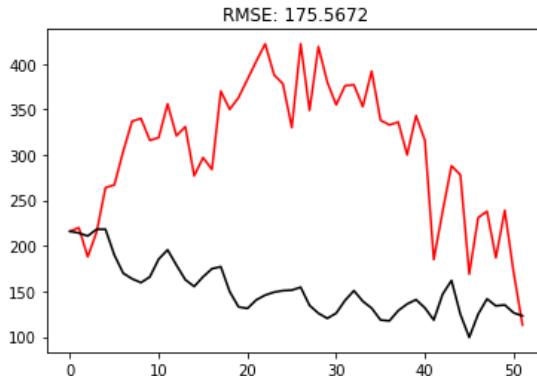


```
In [89]: predictions_ARIMA_diff = pd.Series(results_ARIMA.fittedvalues, copy=True)
predictions_ARIMA_diff_cumsum = predictions_ARIMA_diff.cumsum()

predictions_ARIMA_log = pd.Series(ts_log.loc[0], index=ts_log.index)
predictions_ARIMA_log = predictions_ARIMA_log.add(predictions_ARIMA_diff_cumsum, fill_value=0)

predictions_ARIMA = np.exp(predictions_ARIMA_log)
plt.plot(a.valor, 'r')
plt.plot(predictions_ARIMA, 'k')
plt.title('RMSE: %.4f' % np.sqrt(sum((predictions_ARIMA-a.valor)**2)/len(a.valor)))
```

Out[89]: Text(0.5, 1.0, 'RMSE: 175.5672')



El desempeño de la predicción resultó ser malo.

Práctica 10: NP

June 3, 2019

1 Práctica 10: Clasificación de datos con sklearn

1.1 Gabriela Sánchez Y.

El objetivo en esta práctica es aplicar distintos métodos de clasificación a alguna división de interés de los datos estudiados, evaluar dicha clasificación e intentar mejorarla con un ajuste de parámetros si es que se tienen.

Una clasificación asigna de manera automática datos en grupos previamente conocidos según sus atributos. Para el caso de estudio, se determinan dos clasificaciones binarias - sexo del agresor, - si la víctima es mayor o menor de edad.

Se usará el 60% de los datos para entrenar y el 40% para validar. Para la clasificación se consideran los siguientes atributos: - sexo del agresor - sexo y edad de la víctima - mes y hora del incidente, tomando únicamente los incidentes para los cuales solo hay una víctima y un agresor.

```
[1]: import pandas as pd
datos = pd.read_csv("/home/saphira/Desktop/datos/vf_p5.csv")
datos.head()
```

```
/usr/local/lib/python3.5/dist-packages/IPython/core/interactiveshell.py:3020:
 DtypeWarning: Columns (19,28) have mixed types. Specify dtype option on import
 or set low_memory=False.
     interactivity=interactivity, compiler=compiler, result=result)
```

```
[1]:      mes          mpio          col          delito  \
0      1 CADEREYTA JIMÉNEZ VALLE DE LOS ENCINOS VIOLENCIA FAMILIAR
1      1 CADEREYTA JIMÉNEZ VALLE DEL ROBLE VIOLENCIA FAMILIAR
2      1 CADEREYTA JIMÉNEZ VALLE DEL ROBLE VIOLENCIA FAMILIAR
3      1 CIÉNEGA DE FLORES LOMAS DE CIENEGA VIOLENCIA FAMILIAR
4      1 CIÉNEGA DE FLORES REAL DEL SOL VIOLENCIA FAMILIAR

      mod      fecha violencia          tipo hora  \
0 VIOLENCIA FAMILIAR 2018/01/01      CON  FÍSICA, PSICOLOGICA N
1 VIOLENCIA FAMILIAR 2017/12/27      CON  FÍSICA, PSICOLOGICA M
2 VIOLENCIA FAMILIAR 2018/01/01      CON  PSICOLOGICA, PATRIMONIAL M
3 VIOLENCIA FAMILIAR 2017/12/22      CON  FÍSICA, PSICOLOGICA M
4 VIOLENCIA FAMILIAR 2017/12/25      CON        FÍSICA T

carpeta ... escolaridad atencion_psi atencion_med agresor sexo_a \
```

0	3/2018	...	SECUNDARIA	SI	NO	1.0	M
1	4/2018	...	SECUNDARIA	SI	NO	1.0	M
2	2/2018	...	PREPARATORIA	SI	NO	1.0	M
3	5/2018	...	PREPARATORIA	SI	NO	1.0	M
4	4/2018	...	SECUNDARIA	SI	NO	1.0	M

	edad_a	edo_civil_a	ocupacion_a	escolaridad_a	atencion_psi.1
0	36 a 40	UNIÓN LIBRE	PRESTADOR DE SERVICIOS	SECUNDARIA	NO
1	21 a 25	UNIÓN LIBRE	SIN OFICIO	SECUNDARIA	NO
2	41	CASADO	CHOFER	SECUNDARIA	NO
3	34	UNIÓN LIBRE	EMPLEADO	PRIMARIA	NO
4	45	CASADO	EMPLEADO	PREPARATORIA	NO

[5 rows x 33 columns]

```
[2]: from numpy import isnan
datos = datos.loc[datos['victima'] == 1]
datos = datos.loc[datos['agresor'] == 1]
datos = datos.loc[~isnan(datos.agresor)]
datos = datos.loc[~isnan(datos.victima)]

prueba = datos.edad_v
limpios = []
for dato in prueba:
    s = str(dato).replace(","," ")
    pedazos = s.split()
    while "a" in pedazos:
        pos = pedazos.index("a")
        desde = int(pedazos[pos - 1])
        hasta = int(pedazos[pos + 1])
        prom = (desde + hasta) // 2
        limpios.append(prom)
        pedazos = pedazos[: (pos - 1)] + pedazos[(pos + 2):]
    limpios += pedazos
datos.edad_v = list(limpios)
datos = datos.loc[datos['edad_v'] != 'NE']

datos.hora.replace(to_replace = dict(M = 0, T = 1, N = 2), inplace = True)
datos.sexo_a.replace(to_replace = dict(M = 0, F = 1), inplace = True)
datos.sexo_v.replace(to_replace = dict(M = 0, F = 1), inplace = True)
datos.edad_v = [int(v) for v in datos.edad_v]

d = {'sa': datos.sexo_a, 'sv': datos.sexo_v, 'ev': datos.edad_v, 'mes': datos.
      ~mes, 'hr': datos.hora}
d = pd.DataFrame(data = d)
d.head()
d.to_csv('vf_p10.csv', header=True, index=False)
```

1.1.1 Sexo

```
[9]: import ssl
from math import ceil, sqrt
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from matplotlib.colors import ListedColormap
from numpy import isnan, nan, arange, meshgrid, c_
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB

if getattr(ssl, '_create_unverified_context', None):
    ssl._create_default_https_context = ssl._create_unverified_context

pri = d.sa == 0
seg = d.sa == 1

d['etiquetas'] = [1 if pri[i] else 2 if seg[i] else "NA" for i in pri.keys()] # ↵etiquetas
print(d.etiquetas.value_counts())
y = d.etiquetas
xVars = ['hr', 'mes', 'ev', 'sv']
x = d.loc[:, xVars].values
x = StandardScaler().fit_transform(x)
pca = PCA(n_components = 2) # pedimos uno bidimensional
X = pca.fit_transform(x)

# código de https://scikit-learn.org/stable/auto_examples/classification/
# →plot_classifier_comparison.html
names = ["Nearest Neighbors", "Linear SVM", "RBF SVM", \
         "Decision Tree", "Random Forest", "AdaBoost", "Naive Bayes"]
classifiers = [KNeighborsClassifier(3),SVC(kernel="linear", C=0.025), \
               →SVC(gamma=2, C=1), \
               DecisionTreeClassifier(max_depth=5), \
               →RandomForestClassifier(max_depth=5, n_estimators=10, max_features=1), \
               AdaBoostClassifier(), GaussianNB()]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.4, \
          →random_state=42) # división
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
k = int(ceil(sqrt(len(classifiers) + 1)))
```

```

xx, yy = meshgrid(arange(x_min, x_max, 0.01), arange(y_min, y_max, 0.01))
cm = plt.cm.RdBu
cm_bright = ListedColormap(['#FF0000', '#0000FF'])
plt.rcParams["figure.figsize"] = [16, 16]
figure = plt.figure()
ax = plt.subplot(k, k, 1)
ax.set_title("Datos de entrada")
ax.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cm_bright, alpha=0.2, □
→edgecolors='k') # entrenamiento
ax.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=cm_bright, alpha=0.2, □
→edgecolors='k') # validación
ax.set_xlim(xx.min(), xx.max())
ax.set_ylim(yy.min(), yy.max())
ax.set_xticks(())
ax.set_yticks(())
i = 2
for name, clf in zip(names, classifiers):
    ax = plt.subplot(k, k, i)
    clf.fit(X_train, y_train)
    score = clf.score(X_test, y_test)
    if hasattr(clf, "decision_function"):
        Z = clf.decision_function(c_[xx.ravel(), yy.ravel()])
    else:
        Z = clf.predict_proba(c_[xx.ravel(), yy.ravel()])[:,1]
    Z = Z.reshape(xx.shape)
    ax.contourf(xx, yy, Z, cmap=cm, alpha=.8)
    ax.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cm_bright, □
→edgecolors='k')
    ax.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=cm_bright, □
→edgecolors='k', alpha=0.6)
    ax.set_xlim(xx.min(), xx.max())
    ax.set_ylim(yy.min(), yy.max())
    ax.set_xticks(())
    ax.set_yticks(())
    ax.set_title(name)
    ax.text(xx.max() - .3, yy.min() + .3, ('%.3f' % score).lstrip('0'), size=40, □
→horizontalalignment='right')
    i += 1
plt.tight_layout()
plt.show()

```

```

1      12934
2      1819
Name: etiquetas, dtype: int64

```

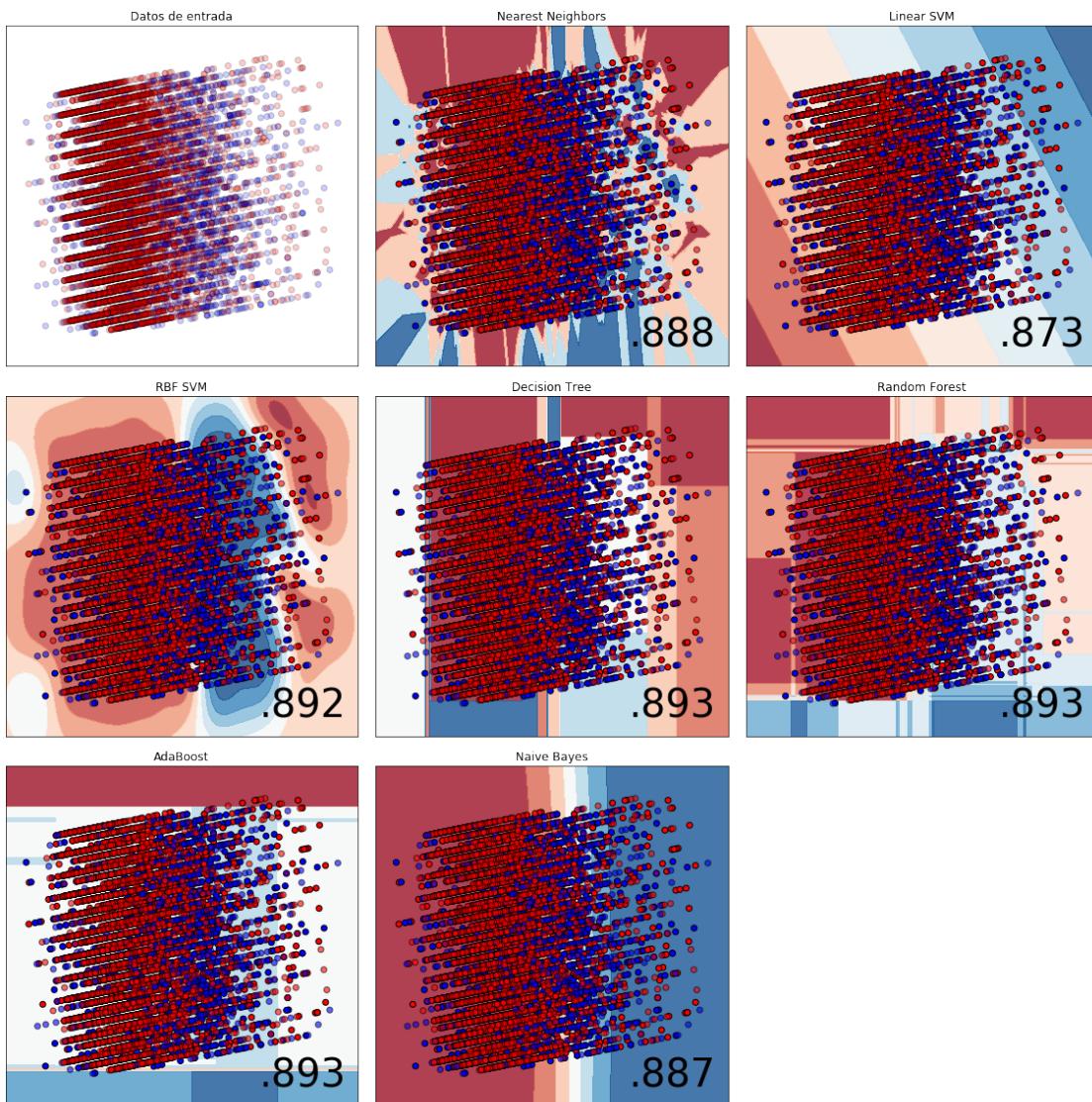
```

/usr/local/lib/python3.5/dist-packages/sklearn/utils/validation.py:595:
DataConversionWarning: Data with input dtype int64 was converted to float64 by

```

```
StandardScaler.
```

```
    warnings.warn(msg, DataConversionWarning)
/usr/local/lib/python3.5/dist-packages/sklearn/utils/validation.py:595:
DataConversionWarning: Data with input dtype int64 was converted to float64 by
StandardScaler.
    warnings.warn(msg, DataConversionWarning)
```



Los clasificadores que tienen el mayor puntaje de precisión son *Decision Tree* y *AdaBoost*. Para un mejor análisis, hay que revisar las matrices de confusión.

```
[4]: from sklearn import metrics
for name, clf in zip(names, classifiers):
    clf.fit(X_train, y_train)
    print(name, clf.score(X_test, y_test))
```

```

expected, predicted = y_test, clf.predict(X_test)
print(metrics.classification_report(expected, predicted))
print(metrics.confusion_matrix(expected, predicted))
print('-' * 60)

```

Nearest Neighbors 0.8880040664181633

	precision	recall	f1-score	support
1	0.92	0.95	0.94	5152
2	0.57	0.47	0.52	750
micro avg	0.89	0.89	0.89	5902
macro avg	0.75	0.71	0.73	5902
weighted avg	0.88	0.89	0.88	5902

```

[[4888 264]
 [ 397 353]]
-----
```

Linear SVM 0.8729244323957981

	precision	recall	f1-score	support
1	0.87	1.00	0.93	5152
2	0.00	0.00	0.00	750
micro avg	0.87	0.87	0.87	5902
macro avg	0.44	0.50	0.47	5902
weighted avg	0.76	0.87	0.81	5902

```

[[5152  0]
 [ 750  0]]
-----
```

```

/usr/local/lib/python3.5/dist-packages/sklearn/metrics/classification.py:1143:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
```

RBF SVM 0.8922399186716368

	precision	recall	f1-score	support
1	0.92	0.96	0.94	5152
2	0.60	0.46	0.52	750
micro avg	0.89	0.89	0.89	5902
macro avg	0.76	0.71	0.73	5902
weighted avg	0.88	0.89	0.89	5902

```

[[4922 230]]
```

[406 344]]

Decision Tree 0.8932565232124704

	precision	recall	f1-score	support
1	0.93	0.95	0.94	5152
2	0.60	0.47	0.53	750
micro avg	0.89	0.89	0.89	5902
macro avg	0.76	0.71	0.73	5902
weighted avg	0.88	0.89	0.89	5902

[[4918 234]

[396 354]]

Random Forest 0.8866485936970518

	precision	recall	f1-score	support
1	0.92	0.96	0.94	5152
2	0.57	0.42	0.48	750
micro avg	0.89	0.89	0.89	5902
macro avg	0.75	0.69	0.71	5902
weighted avg	0.87	0.89	0.88	5902

[[4921 231]

[438 312]]

AdaBoost 0.8934259573026093

	precision	recall	f1-score	support
1	0.93	0.95	0.94	5152
2	0.60	0.48	0.53	750
micro avg	0.89	0.89	0.89	5902
macro avg	0.76	0.72	0.74	5902
weighted avg	0.89	0.89	0.89	5902

[[4914 238]

[391 359]]

Naive Bayes 0.8866485936970518

	precision	recall	f1-score	support
1	0.91	0.96	0.94	5152
2	0.59	0.37	0.45	750
micro avg	0.89	0.89	0.89	5902

macro avg	0.75	0.67	0.70	5902
weighted avg	0.87	0.89	0.88	5902

```
[[4955 197]
 [ 472 278]]
```

Un buen desempeño del modelo se representaría en una matriz de confusión con valores grandes en el diagonal y valores bajos en otros lados. Ninguno de los modelos dió resultados muy buenos, como se mencionó anteriormente, los modelos con mayor puntaje de precisión son *Decision Tree* y *AdaBoost* con 0.893.

El clasificador *Linear SVM* tiene buen desempeño clasificando todos los incidentes en los que el sexo es masculino, sin embargo, falla en todos los casos contrarios. En general, parece que para los casos en los que el sexo del agresor es femenino son más difíciles de clasificar.

Ahora, se revisará qué pasa con los resultados si no se especifica semilla a la división entre conjuntos de entrenamiento y prueba.

```
[5]: pri = d.sa == 0
seg = d.sa == 1

d['etiquetas'] = [1 if pri[i] else 2 if seg[i] else "NA" for i in pri.keys()] # →etiquetas
print(d.etiquetas.value_counts())
y = d.etiquetas
xVars = ['hr', 'mes', 'ev', 'sv']
x = d.loc[:, xVars].values
x = StandardScaler().fit_transform(x)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.4) # cada ejecución tiene una división al azar
for name, clf in zip(names, classifiers):
    clf.fit(X_train, y_train)
    y_m = clf.predict(X_test)
    print(name, clf.score(X_test, y_test))
    expected, predicted = y_test, clf.predict(X_test)
    print(metrics.confusion_matrix(y_test, y_m))
    print('-' * 60)
```

```
1      12934
2      1819
Name: etiquetas, dtype: int64
```

```
/usr/local/lib/python3.5/dist-packages/sklearn/utils/validation.py:595:
DataConversionWarning: Data with input dtype int64 was converted to float64 by
StandardScaler.
    warnings.warn(msg, DataConversionWarning)
/usr/local/lib/python3.5/dist-packages/sklearn/utils/validation.py:595:
DataConversionWarning: Data with input dtype int64 was converted to float64 by
```

```

StandardScaler.
    warnings.warn(msg, DataConversionWarning)

Nearest Neighbors 0.8896984073195527
[[4944  262]
 [ 389  307]]
-----
Linear SVM 0.8820738732633006
[[5206    0]
 [ 696    0]]
-----
RBF SVM 0.8985089800067774
[[4972  234]
 [ 365  331]]
-----
Decision Tree 0.9008810572687225
[[4951  255]
 [ 330  366]]
-----
Random Forest 0.896645205015249
[[4980  226]
 [ 384  312]]
-----
AdaBoost 0.894103693663165
[[4903  303]
 [ 322  374]]
-----
Naive Bayes 0.8939342595730261
[[5036  170]
 [ 456  240]]

```

No se obtuvo una mejora. El mayor puntaje de precisión alcanzado en este caso es menor que en el caso anterior.

1.1.2 Edad

Ahora se repite el procedimiento anterior, solo que ahora se busca clasificar sí o no en el incidente la víctima es menor o mayor de edad.

```
[6]: pri = d.ev >= 18
d['etiquetas'] = [1 if pri[i] else 0 for i in pri.keys()] # etiquetas
print(d.etiquetas.value_counts())
y = d.etiquetas
xVars = ['hr', 'mes', 'sa', 'sv']
x = d.loc[:, xVars].values
x = StandardScaler().fit_transform(x)
pca = PCA(n_components = 2) # pedimos uno bidimensional
X = pca.fit_transform(x)
```

```

# código de https://scikit-learn.org/stable/auto_examples/classification/
→plot_classifier_comparison.html
names = ["Nearest Neighbors", "Linear SVM", "RBF SVM", \
         "Decision Tree", "Random Forest", "AdaBoost", "Naive Bayes"]
classifiers = [KNeighborsClassifier(3),SVC(kernel="linear", C=0.025), \
→SVC(gamma=2, C=1), \
               DecisionTreeClassifier(max_depth=5), \
→RandomForestClassifier(max_depth=5, n_estimators=10, max_features=1), \
               AdaBoostClassifier(), GaussianNB()]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.4, \
→random_state=42) # división
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
k = int(ceil(sqrt(len(classifiers) + 1)))
xx, yy = meshgrid(arange(x_min, x_max, 0.01), arange(y_min, y_max, 0.01))
cm = plt.cm.RdBu
cm_bright = ListedColormap(['#FF0000', '#0000FF'])
plt.rcParams["figure.figsize"] = [16, 16]
figure = plt.figure()
ax = plt.subplot(k, k, 1)
ax.set_title("Datos de entrada")
ax.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cm_bright, alpha=0.2, \
→edgecolors='k') # entrenamiento
ax.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=cm_bright, alpha=0.2, \
→edgecolors='k') # validación
ax.set_xlim(xx.min(), xx.max())
ax.set_ylim(yy.min(), yy.max())
ax.set_xticks(())
ax.set_yticks(())
i = 2
for name, clf in zip(names, classifiers):
    ax = plt.subplot(k, k, i)
    clf.fit(X_train, y_train)
    score = clf.score(X_test, y_test)
    if hasattr(clf, "decision_function"):
        Z = clf.decision_function(c_[xx.ravel(), yy.ravel()])
    else:
        Z = clf.predict_proba(c_[xx.ravel(), yy.ravel()])[:,1]
    Z = Z.reshape(xx.shape)
    ax.contourf(xx, yy, Z, cmap=cm, alpha=.8)
    ax.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cm_bright, \
→edgecolors='k')
    ax.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=cm_bright, \
→edgecolors='k', alpha=0.6)
    ax.set_xlim(xx.min(), xx.max())

```

```

    ax.set_ylim(yy.min(), yy.max())
    ax.set_xticks(())
    ax.set_yticks(())
    ax.set_title(name)
    ax.text(xx.max() - .3, yy.min() + .3, ('%.3f' % score).lstrip('0'), size=40, u
    ↪horizontalalignment='right')
    i += 1
plt.tight_layout()
plt.show()

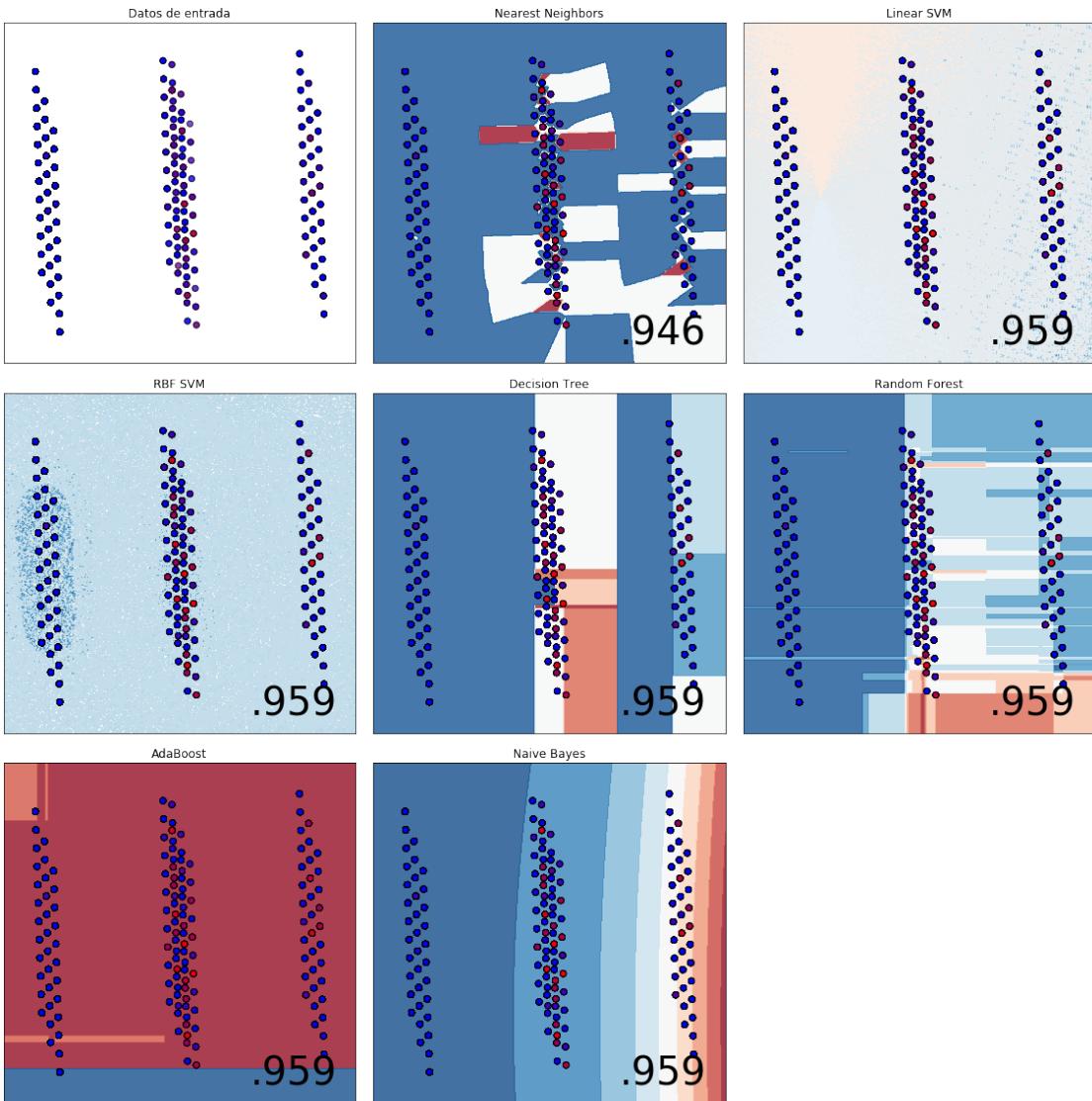
```

```

1      14204
0      549
Name: etiquetas, dtype: int64

/usr/local/lib/python3.5/dist-packages/sklearn/utils/validation.py:595:
DataConversionWarning: Data with input dtype int64 was converted to float64 by
StandardScaler.
    warnings.warn(msg, DataConversionWarning)
/usr/local/lib/python3.5/dist-packages/sklearn/utils/validation.py:595:
DataConversionWarning: Data with input dtype int64 was converted to float64 by
StandardScaler.
    warnings.warn(msg, DataConversionWarning)

```



```
[7]: for name, clf in zip(names, classifiers):
    clf.fit(X_train, y_train)
    y_m = clf.predict(X_test)
    print(name, clf.score(X_test, y_test))
    expected, predicted = y_test, clf.predict(X_test)
    print(metrics.confusion_matrix(y_test, y_m))
    print('-' * 60)
```

Nearest Neighbors 0.9457810911555405

```
[[ 19  224]
 [ 96 5563]]
```

Linear SVM 0.9588275160962385

```

[[ 0 243]
 [ 0 5659]]
```

```
RBF SVM 0.9588275160962385
[[ 0 243]
 [ 0 5659]]
```

```
Decision Tree 0.9588275160962385
[[ 0 243]
 [ 0 5659]]
```

```
Random Forest 0.9588275160962385
[[ 0 243]
 [ 0 5659]]
```

```
AdaBoost 0.9588275160962385
[[ 0 243]
 [ 0 5659]]
```

```
Naive Bayes 0.9588275160962385
[[ 0 243]
 [ 0 5659]]
```

El modelo que tiene el menor desempeño de acuerdo al puntaje de precisión es el clasificador *Nearest Neighbors*, aunque al revisar las matrices de confusión se aprecia que todos los otros modelos no acertaron en clasificar los incidentes donde la víctima es menor de edad. *Nearest Neighbors* es el único que logra clasificar 19 de éstos incidentes.

```
[8]: pri = d.ev >= 18
d['etiquetas'] = [1 if pri[i] else 0 for i in pri.keys()] # etiquetas
print(d.etiquetas.value_counts())
y = d.etiquetas
xVars = ['hr', 'mes', 'sa', 'sv']
x = d.loc[:, xVars].values
x = StandardScaler().fit_transform(x)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.4) # cada
→ ejecución tiene una división al azar
for name, clf in zip(names, classifiers):
    clf.fit(X_train, y_train)
    y_m = clf.predict(X_test)
    print(name, clf.score(X_test, y_test))
    expected, predicted = y_test, clf.predict(X_test)
    print(metrics.confusion_matrix(y_test, y_m))
    print('-' * 60)
```

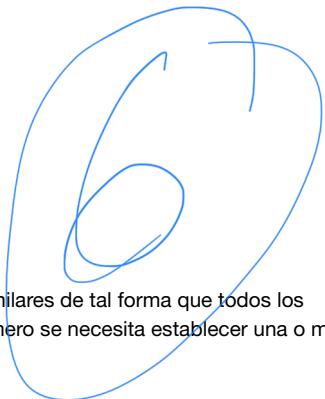
```

0      549
Name: etiquetas, dtype: int64

/usr/local/lib/python3.5/dist-packages/sklearn/utils/validation.py:595:
DataConversionWarning: Data with input dtype int64 was converted to float64 by
StandardScaler.
    warnings.warn(msg, DataConversionWarning)
/usr/local/lib/python3.5/dist-packages/sklearn/utils/validation.py:595:
DataConversionWarning: Data with input dtype int64 was converted to float64 by
StandardScaler.
    warnings.warn(msg, DataConversionWarning)

Nearest Neighbors 0.9576414774652661
[[ 13 225]
 [ 25 5639]]
-----
Linear SVM 0.9596746865469332
[[ 0 238]
 [ 0 5664]]
-----
RBF SVM 0.9596746865469332
[[ 0 238]
 [ 0 5664]]
-----
Decision Tree 0.9596746865469332
[[ 0 238]
 [ 0 5664]]
-----
Random Forest 0.9596746865469332
[[ 0 238]
 [ 0 5664]]
-----
AdaBoost 0.9596746865469332
[[ 0 238]
 [ 0 5664]]
-----
Naive Bayes 0.9596746865469332
[[ 0 238]
 [ 0 5664]]
-----
```

Ahora que no se especifica semilla a la división entre conjuntos de entrenamiento y prueba, se obtienen mejores resultados en el puntaje de precisión.



Práctica 11: Agrupamiento de datos

Gabriela Sánchez Y.

El objetivo del agrupamiento es descubrir si o no los datos forman grupos con elementos similares de tal forma que todos los grupos difieren entre sí en algunos aspectos. Para poder llevar a cabo un agrupamiento, primero se necesita establecer una o más medidas de similitud entre los datos o alternativamente una o más medidas de distancia.

El objetivo de esta práctica es seleccionar por lo menos dos algoritmos distintos para llevar a cabo agrupamiento con por lo menos dos atributos del caso de estudio, reportando los hallazgos con gráficas y con medidas de calidad.

Se realizó un preprocesamiento a los datos, se trabajará solo con el mes y la hora del incidente, sexo y edad de agresor y víctima, considerando solo los incidentes en los cuales hay una víctima y un agresor.

```
In [2]: import pandas as pd  
d = pd.read_csv("vf_p11.csv")  
d.head()
```

Out[2]:

	edad_a	edad_v	escolaridad	hora	mes	mpio	parentesco	sexo_a	sexo_v
0	38	46	SECUNDARIA	2	1	CADEREYTA JIMÉNEZ	UL	0	1
1	23	26	SECUNDARIA	0	1	CADEREYTA JIMÉNEZ	CONCUBINO	0	1
2	41	42	PREPARATORIA	0	1	CADEREYTA JIMÉNEZ	NINGUNO	0	1
3	34	26	PREPARATORIA	0	1	CIÉNEGA DE FLORES	CONCUBINO	0	1
4	45	38	SECUNDARIA	1	1	CIÉNEGA DE FLORES	ESPOSO	0	1

K Medias

El primer algoritmo usado es el algoritmo de k medias que requiere como parámetro el número de grupos en los que se desea agrupar los datos. El procedimiento que sigue este método es asignar k centroides y asociar los datos al centroide más cercano, iterativamente reacomoda los centroides para minimizar las distancias entre los datos y sus centroides.

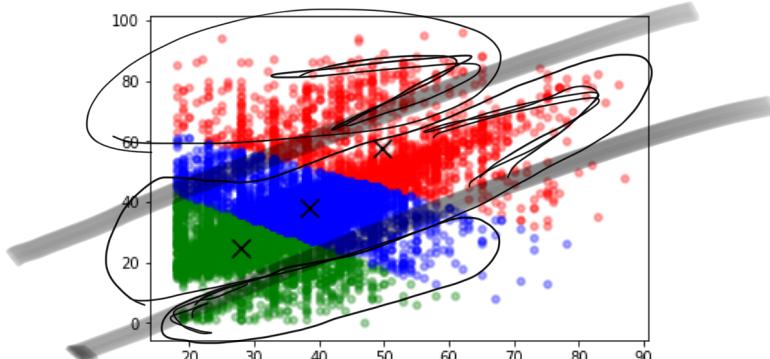
El agrupamiento se llevará a cabo para los atributos de **edad** tanto de la víctima como del agresor. Se inicia con **k=3**.

```
In [4]: import ssl
from sklearn import metrics
from numpy.random import seed
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from numpy import isnan, nan, take, where

seed(42)
if getattr(ssl, '_create_unverified_context', None):
    ssl._create_default_https_context = ssl._create_unverified_context

keep = ['edad_a', 'edad_v'] # dos atributos numéricos para comenzar (edades de agresor y víctima)
d = d.loc[:, keep]
x = d.values
k = 3 # pedimos tres grupos
m = KMeans(init = 'random', n_clusters = k, n_init = 10)
m.fit(x)
centroides = m.cluster_centers_
grupos = m.predict(x)
plt.figure(1)
plt.scatter(centroides[:, 0], centroides[:, 1], marker='x', s=150, linewidths=3, color='black', zorder=10)
colores = ['r', 'g', 'b']
for g in range(k):
    incl = where(grupos == g)[0]
    print(len(incl), "integrantes en grupo", g)
    grupo = take(x, incl, 0)
    plt.scatter(grupo[:, 0], grupo[:, 1], marker='o', s=20, linewidths=2, color=colores[g], alpha = 0.3, zorder=5)
plt.show()
print(metrics.silhouette_score(x, grupos, metric='euclidean'))
```

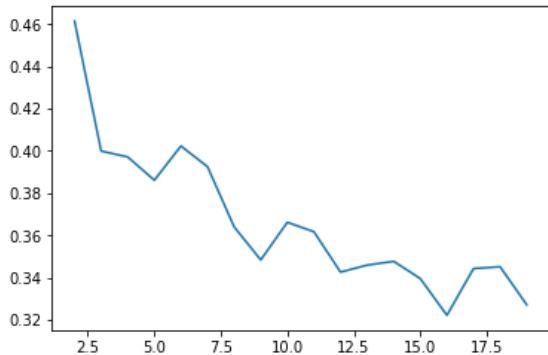
1820 integrantes en grupo 0
5948 integrantes en grupo 1
5863 integrantes en grupo 2



0.3999465029185761

Ahora, se prueba con diferentes valores de k para ver cuál da el mejor valor del coeficiente de silueta.

```
In [5]: x = d.values
ks = [k for k in range(2, 20)]
sil = []
for k in ks:
    m = KMeans(init = 'random', n_clusters = k, n_init = 10)
    m.fit(x)
    sil.append(metrics.silhouette_score(x, m.predict(x), metric='euclidean'))
plt.figure(1)
plt.plot(ks, sil)
plt.show()
```



El mejor coeficiente de silueta se encuentra para k=2.

Nuevamente, se usa el algoritmo pero ahora se utilizan más atributos: sexo del agresor y la víctima, mes y hora del incidente. Para poder tener una visualización bidimensional se aplica primero un análisis de componentes principales.

```
In [6]: from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

datos = pd.read_csv("vf_p11.csv")
d = {'sa': datos.sexo_a, 'sv': datos.sexo_v, 'edad_v': datos.edad_v, 'mes': datos.mes, 'hr': datos.hora, 'edad_a': datos.edad_a}
d = pd.DataFrame(data = d)

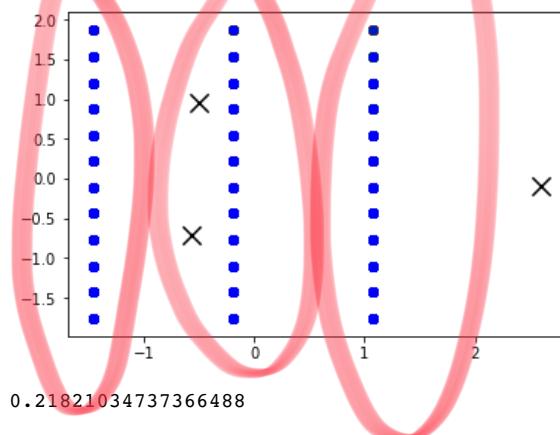
pri = d.edad_v >= 18
d['etiquetas'] = [1 if pri[i] else 0 for i in pri.keys()] # etiquetas
print(d.etiquetas.value_counts())
y = d.etiquetas
xVars = ['hr', 'mes', 'sa', 'sv', 'edad_a']
x = d.loc[:, xVars].values
x = StandardScaler().fit_transform(x)
pca = PCA(n_components = 2) # pedimos uno bidimensional
X = pca.fit_transform(x)

k = 3 # pedimos tres grupos
m = KMeans(init = 'random', n_clusters = k, n_init = 10)
m.fit(X)
centroides = m.cluster_centers_
grupos = m.predict(X)
plt.figure(1)
plt.scatter(centroides[:, 0], centroides[:, 1], marker='x', s=150, linewidths=3, color='black', zorder=10)
colores = ['r', 'g', 'b']
for g in range(k):
    incl = where(grupos == g)[0]
    print(len(incl), "integrantes en grupo", g)
    grupo = take(x, incl, 0)
    plt.scatter(grupo[:, 0], grupo[:, 1], marker='o', s=20, linewidths=2, color=colores[g], alpha=0.3, zorder=5)
plt.show()
print(metrics.silhouette_score(x, grupos, metric='euclidean'))
```

```
1      13135
0       496
Name: etiquetas, dtype: int64
6360 integrantes en grupo 0
```

```
/usr/local/lib/python3.5/dist-packages/sklearn/utils/validation.py:595: DataConversionWarning
: Data with input dtype int64 was converted to float64 by StandardScaler.
warnings.warn(msg, DataConversionWarning)
/usr/local/lib/python3.5/dist-packages/sklearn/utils/validation.py:595: DataConversionWarning
: Data with input dtype int64 was converted to float64 by StandardScaler.
warnings.warn(msg, DataConversionWarning)
```

```
2344 integrantes en grupo 1
4927 integrantes en grupo 2
```



MeanShift

El agrupamiento de MeanShift (<https://scikit-learn.org/stable/modules/clustering.html#mean-shift>) es un algoritmo basado en centroides, que funciona mediante la actualización de candidatos a centroides de forma que éstos sean la media de los puntos dentro de una región determinada. Estos candidatos luego se filtran en una etapa de post-procesamiento para eliminar los duplicados cercanos y formar el conjunto final de centroides.

```
In [7]: from sklearn.cluster import MeanShift, estimate_bandwidth
import matplotlib.pyplot as plt
from itertools import cycle
import numpy as np

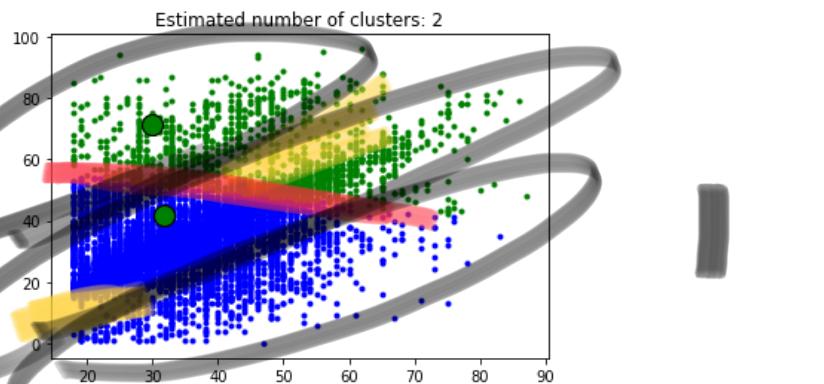
datos = pd.read_csv("vf_p11.csv")
keep = ['edad_a', 'edad_v'] # dos atributos numéricos para comenzar (edades de agresor y víctima)
datos = datos.loc[:, keep]
x = datos.values
bandwidth = estimate_bandwidth(x, quantile=0.2)
m = MeanShift(bandwidth = bandwidth)
m.fit(x)
labels = m.labels_
centroide = m.cluster_centers_
print(centroide)

labels_unique = np.unique(labels)
nc = len(labels_unique)

plt.figure(1)
plt.clf()

colors = cycle('bgrcmykbgrcmykbgrcmykbgrcmyk')
for k, col in zip(range(nc), colors):
    my_members = labels == k
    cluster_center = centroide[k]
    plt.plot(x[my_members, 0], x[my_members, 1], col + '.')
    plt.plot(centroide[0], centroide[1], 'o', markerfacecolor=col,
             markeredgecolor='k', markersize=14)
plt.title('Estimated number of clusters: %d' % nc)
plt.show()
```

[[31.7765685 30.03729193]
[41.65238095 71.23809524]]



Práctica 12: NP

Práctica 13: NP