
PEC 4
PRÁCTICA FINAL

UNIVERSITAT OBERTA DE CATALUNYA
CIENCIA DE DATOS APLICADA
PROGRAMACIÓN EN SCRIPTING

Autor:

FRANCISCO LAFARGA POYO

Índice

| | |
|---|----------|
| 1. Ejercicio 2 | 1 |
| 1.1. Acerca del dataset | 1 |
| 1.2. Comprobando posibles errores | 4 |
| 1.2.1. Campo desc | 4 |
| 1.2.2. Campo lat y lng | 5 |
| 1.3. Modificación de datos | 6 |
| 1.3.1. Eliminación de la última columna. | 6 |
| 1.3.2. Formateo de fecha en la columna timeStamp. | 7 |
| 1.4. Tratamiento valores nulos | 7 |
| 1.5. Creando nuevas variables | 8 |
| 1.5.1. Variables year, month, day | 8 |
| 1.5.2. Variable type | 9 |
| 1.6. Dataset tras las modificaciones | 10 |
| 1.7. Scripts generar tablas | 11 |
| 1.7.1. Generar tabla en formato html | 11 |
| 1.7.2. Generar tabla en formato L ^A T _E X | 12 |

1. Ejercicio 2

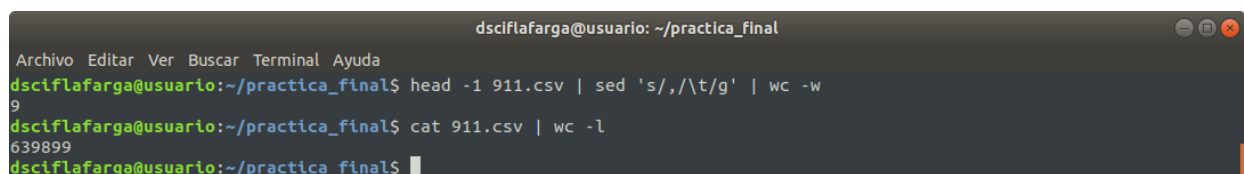
1.1. Acerca del dataset

El dataset que he seleccionado se llama **Emergency - 911 Calls**, se puede encontrar en el siguiente enlace: <https://www.kaggle.com/mchirico/montcoalert>.

Se trata de un solo fichero de 112 MB, el cual contiene un registro de llamadas al 911 (llamadas de emergencia) en el condado de Montgomery, Pensilvania.

Comprobamos las dimensiones del dataset de la siguiente manera:

```
head -1 911.csv | sed 's/,/\t/g' | wc -w # Columns
cat 911.csv | wc -l                      # Filas
```



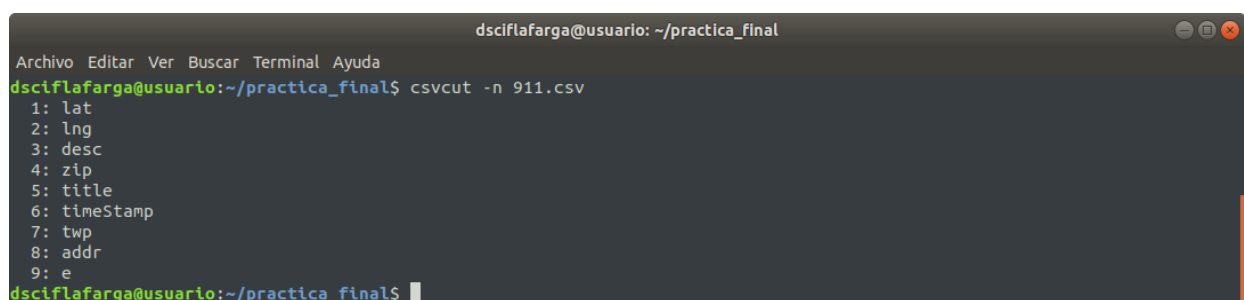
```
dsciflafarga@usuario: ~/practica_final
Archivo Editar Ver Buscar Terminal Ayuda
dsciflafarga@usuario:~/practica_final$ head -1 911.csv | sed 's/,/\t/g' | wc -w
9
dsciflafarga@usuario:~/practica_final$ cat 911.csv | wc -l
639899
dsciflafarga@usuario:~/practica_final$
```

Figura 1: Cálculo del número de filas y columnas.

El archivo esta compuesto por 9 variables o columnas y un total de 639898 registros. Este repositorio se actualiza cada pocas semanas, por lo que voy a trabajar con los datos descargados a día 31 de mayo de 2020, los cuales corresponden a la versión 25 según la página de Kaggle.

Para seguir examinando el dataset, vemos la cabecera o campos del mismo:

```
csvcut -n 911.csv
```



```
dsciflafarga@usuario: ~/practica_final
Archivo Editar Ver Buscar Terminal Ayuda
dsciflafarga@usuario:~/practica_final$ csvcut -n 911.csv
1: lat
2: lng
3: desc
4: zip
5: title
6: timeStamp
7: twp
8: addr
9: e
dsciflafarga@usuario:~/practica_final$
```

Figura 2: Número de columnas del archivo y sus nombres.

Para ver información acerca de cada campo, uso el siguiente comando:

```
csvstat 911.csv
```

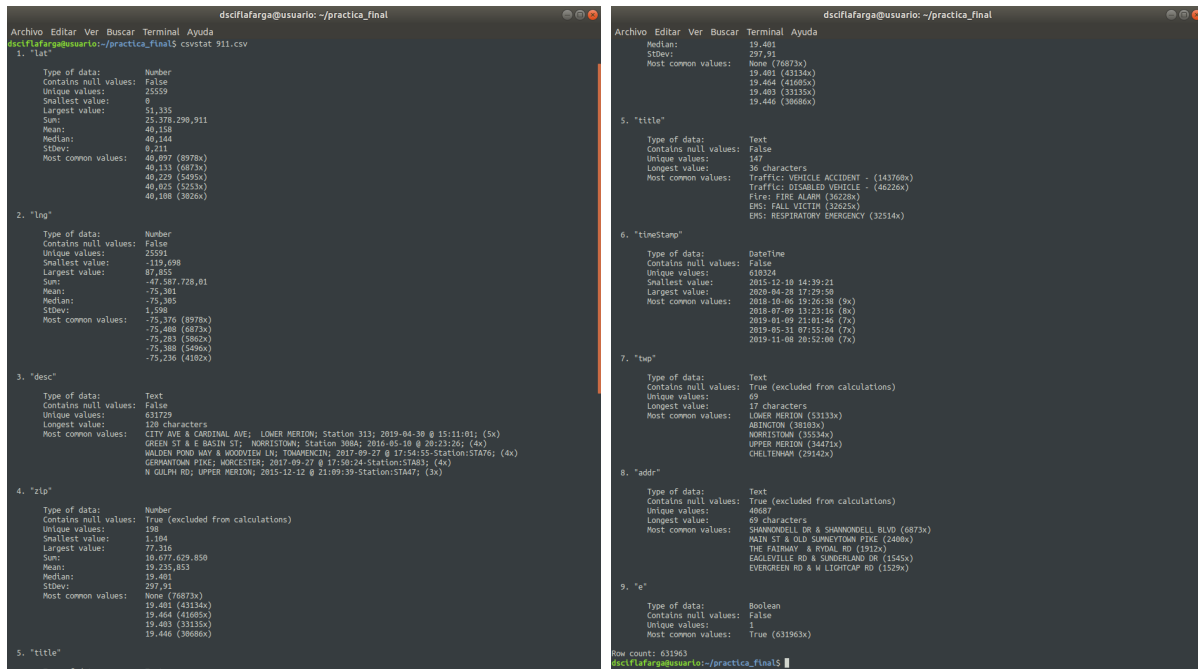


Figura 3: Salida del comando csvstat.

En la Tabla siguiente se puede ver un resumen de cada campo.

| Campo | Tipo | Ejemplo | Valor más repetido |
|-----------|----------|--------------------------------|--|
| lat | Número | 51335 | 40097 |
| lng | Número | 87855 | -75376 |
| desc | Texto | LOWER MERION | CITY AVE & CARDINAL AVE |
| zip | Número | 1104 | |
| title | Texto | EMS: FALL VICTIM | Traffic: VEHICLE ACCIDENT - (143760x) |
| timeStamp | DateTime | 2015-12-10 14:39:21 | 2018-10-06 19:26:38 (9X) |
| twp | Texto | CHELTENHAM (29142x) | LOWER MERION (53133x) |
| addr | Texto | THE FAIRWAY & RYDAL RD (1912x) | SHANNONDELL DR & SHANNONDEL BLVD (6873x) |
| e | Boolean | 1 | 1 |

Tabla 1: Variables del dataset.

A continuación paso a explicar los valores de algunos de los campos del dataset y cómo voy a trabajar con ellos.

Campo lat y lng

Estos campos corresponden a la latitud y la longitud donde se produce el incidente de la llamada. Como explico más adelante, el campo lat tiene que ser positivo y el campo lng negativo por lo que voy a comprobarlo.

Campo desc

Este campo es de tipo Texto, se compone a su vez por 3 o 4 subcampos en función del tipo de llamada. Inspeccionando el dataset, se puede ver que en el caso de las llamadas de tipo EMS tiene 4 subcampos y en el resto tiene 3. Como son muchas entradas en el fichero csv, lo voy a comprobar de la siguiente manera:

```
awk -F, '$5 ~ "EMS"{gsub(/[^\;]/, "", $3); print length($3)}' 911.csv | sort | uniq -c
```

```
dsciflafarga@usuario: ~/practica_final/datos_parte1
Archivo Editar Ver Buscar Terminal Ayuda
dsciflafarga@usuario:~/practica_final/datos_parte1$ awk -F, '$5 ~ "EMS"{gsub(/[^\;]/, "", $3); print length($3)}' 911.csv | sort | uniq -c
7 3
320326 4
dsciflafarga@usuario:~/practica_final/datos_parte1$ awk -F, '$5 ~ "Traffic"{gsub(/[^\;]/, "", $3); print length($3)}' 911.csv | sort | uniq -c
223395 3
dsciflafarga@usuario:~/practica_final/datos_parte1$ awk -F, '$5 ~ "Fire"{gsub(/[^\;]/, "", $3); print length($3)}' 911.csv | sort | uniq -c
96177 3
dsciflafarga@usuario:~/practica_final/datos_parte1$
```

Figura 4: Comprobación del número de subcampos con awk.

Vemos como la salida no arroja lo que esperábamos, ya que hay 7 casos en el tipo EMS que tienen 3 subcampos en lugar de 4.

Campo title

Este campo es de tipo Texto y corresponde al título de la llamada, el cual se compone del tipo de llamada y una explicación de la misma. Para trabajar con los tipos de llamada, voy a crear una columna nueva donde incluiré este campo.

Campo timeStamp

Este campo es de tipo DateTime, tiene el formato %y:%m:%d %h:%m:%s. En este análisis no nos interesa la hora, minutos y segundos, por lo que voy a formatearla para quedarme

Añadir
a que
por
eso se
usa
bash

solo con el año, mes y día. Para trabajar mejor con estos datos, también voy a crear una variable nueva para cada uno de ellos.

1.2. Comprobando posibles errores

1.2.1. Campo desc

Como he comentado anteriormente, el campo desc se compone de otros subcampos, los cuales se corresponden con los campos addr, twp, y timeStamp, pudiendo contener un tercer campo que corresponde a la estación.

Para comprobar posibles errores, he elaborado un script que comprueba si se corresponden los subcampos del campo desc, con los campos addr y twp.

```

1  #!/bin/bash
2  # comprobarDatos.sh
3  # Script que comprueba los campos que se almacenan en desc con sus correspondientes
   ↪  addr y twp
4  # -----
5  # |desc                                |.../addr      |twp      |
6  # |-----+-----+-----+-----|
7  # |desc_addr;twp_addr;...           |.../addr_value|twp_value|
8  # |...
9  #
10 # Compara desc_addr con addr_value y twp_addr con twp_value
11 # En caso de que no coincidan, imprime error por pantalla
12
13
14 # Compruebo si se le pasa archivo como parámetro o imprimo error.
15 INPUT=${1?Error: debes pasarle un archivo csv como parametro}
16 [ ! -f $INPUT ] && { printf "$INPUT Archivo no encontrado \n "; exit 99; }
17
18 # Función que comprueba si dos campos son iguales o no
19 # $1 y $2 campos a comparar
20 # $3 Campo que se esta evaluando
21 # $4 número línea en la que no coinciden
22 function comparar_valores() {
23     if [[ "$1" != "$2" ]]; then
24         echo -e "Error no coinciden los datos del campo $3 en la línea número
   ↪  $4 del fichero \n En la descripción es $1 y en el campo $3 es $2"
25     fi
26 }
27
28 {
29 head -1 > /dev/null; # Ignoro primera línea (header)
30 linea=2 # Contador para caso de no coincidir imprimir error(linea 1=header)
31
32 #while IFS=, read -r lat lng desc zip title timeStamp twp addr anio mes dia Type
33 while IFS=, read -r lat lng desc zip title timeStamp twp addr e

```

```

34 do
35     IFS=';' read -r -a VALORES <<< "$desc" #guardo en array VALORES la col
        ↳ description
36
37     addr_desc=$(echo "${VALORES[0]}" | sed 's/^ *//g; s/ *$//')
38     twp_desc=$(echo "${VALORES[1]}" | sed 's/^ *//g; s/ *$//')
39
40     comparar_valores "$addr_desc" "$addr" "addr" "$linea"
41     comparar_valores "$twp_desc" "$twp" "twp" "$linea"
42
43     let "linea++"
44 done
45 } < $INPUT

```

1.2.2. Campo lat y lng

El primer y segundo campo del dataset corresponden a la latitud y longitud en la que se localiza el incidente de la llamada. Al ser llamadas procedentes de EEUU, la altitud siempre debe ser un número positivo y la longitud un número negativo tal y como se muestra en la Figura 5. Lo comprobamos de la siguiente manera para ver posibles errores:

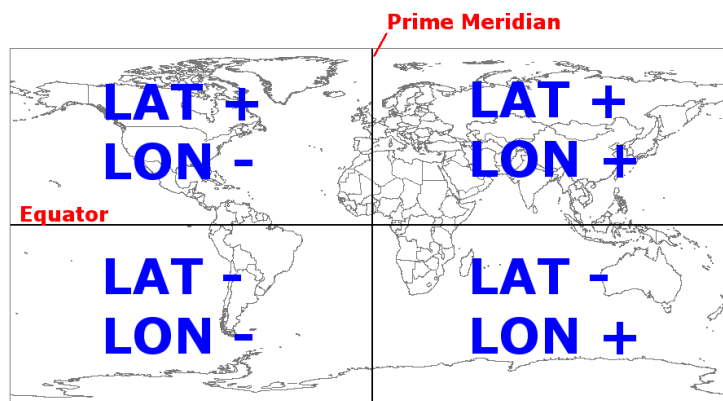


Figura 5: Signos de la latitud y longitud en diferentes regiones.

```

csvcut -c lat 911.csv | grep "^-" | sort | uniq -c      # Latitud
csvcut -c lng 911.csv | grep "[0-9]" | sort | uniq -c  # Longitud

```

```
dsciflafarga@usuario: ~/Documentos/practicas/PROGRAMACION_EN_SCRIPTING/PEC4/datos
Archivo Editar Ver Buscar Terminal Ayuda
dsciflafarga@usuario:~/Documentos/practicas/PROGRAMACION_EN_SCRIPTING/PEC4/datos$ csvcut -c lat 911.csv | grep "^-" | sort | uniq
dsciflafarga@usuario:~/Documentos/practicas/PROGRAMACION_EN_SCRIPTING/PEC4/datos$ csvcut -c lng 911.csv | grep "[0-9]" | sort | uniq -c
1 30.8024980
59 87.8549755
dsciflafarga@usuario:~/Documentos/practicas/PROGRAMACION_EN_SCRIPTING/PEC4/datos$
```

Figura 6: Comprobación del campo latitud y longitud.

Vemos como hay una serie de valores en la longitud que no corresponden a la región de donde supuestamente deberían ser, por lo que considero que se trata de un error y procedo a eliminar esos registros. Buscando en un geolocalizador vemos como efectivamente no corresponden con la región que nos interesa en este estudio.

```
#Primero compruebo cuantos casos hay
awk -F',' ' $1 ~ /^-/' 911.csv | wc -l # Hay 62
# Me quedo solo con los $1 positivos y $2 negativos
cat 911.csv | head -n 1 > file_temp.csv # Incluyo cabecera
awk -F',' ' $1 ~ /^[0-9]/ && $2 ~ /^-/' 911.csv >> file_temp.csv
mv file_temp.csv 911.csv
```

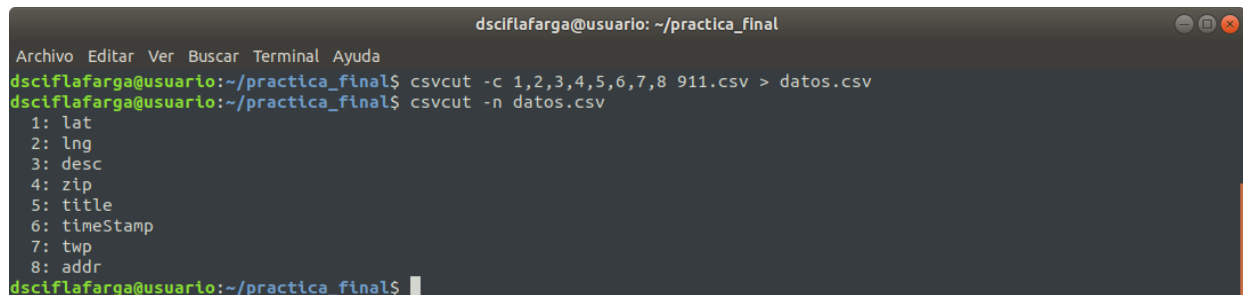
1.3. Modificación de datos

Para trabajar mejor con el dataset procedo a cambiar algunos aspectos del dataset:

1.3.1. Eliminación de la última columna.

La última variable (e) siempre está a 1, por lo que voy a eliminarla del dataset para poder tratar mejor el resto de variables. Para ello creo un fichero temporal y luego lo reemplazo por el fichero original.

```
awk -F, '{for(i=1;i<=NF;i++)if(i!=x)f=f?f FS $i:$i;print f;f=""}' x=9 911.csv >
↪ datos_temp.csv
# O usando csvcut ..
# csvcut -c 1,2,3,4,5,6,7,8 911.csv > datos_temp.csv
mv datos_temp.csv 911.csv
```

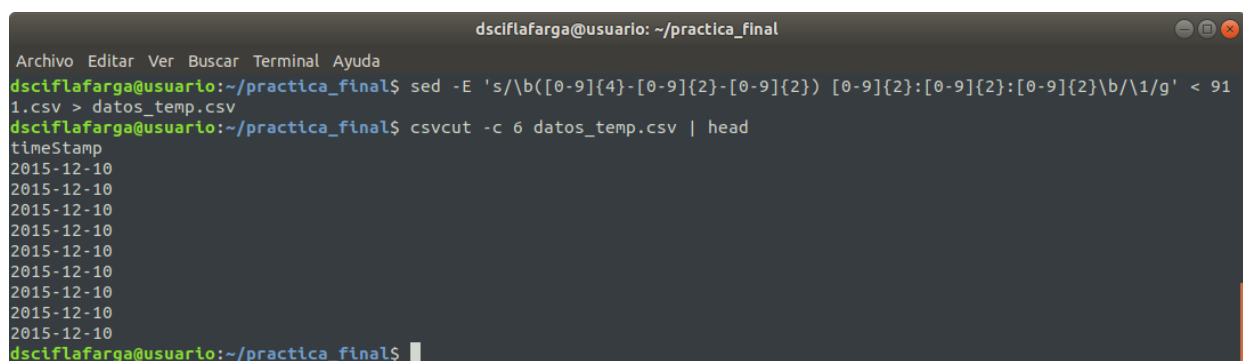
```
dsciflafarga@usuario: ~/practica_final
Archivo Editar Ver Buscar Terminal Ayuda
dsciflafarga@usuario:~/practica_final$ csvcut -c 1,2,3,4,5,6,7,8 911.csv > datos.csv
dsciflafarga@usuario:~/practica_final$ csvcut -n datos.csv
1: lat
2: lng
3: desc
4: zip
5: title
6: timeStamp
7: twp
8: addr
dsciflafarga@usuario:~/practica_final$
```

Figura 7: Creación de archivo nuevo sin la columna e.

1.3.2. Formateo de fecha en la columna timeStamp.

En la columna 6 (timeStamp) vemos como la fecha incluye la hora, minutos y segundos. Voy a formatear esta fecha para que solo muestre el año, el mes y el día.

```
sed -E 's/\b([0-9]{4})-([0-9]{2})-([0-9]{2}) ([0-9]{2}):([0-9]{2}):([0-9]{2})\b/\1/g' < 911.csv >
↵ datos_temp.csv
mv datos_temp.csv 911.csv # Junto en el fichero original los datos formateados
```



```
dsciflafarga@usuario: ~/practica_final
Archivo Editar Ver Buscar Terminal Ayuda
dsciflafarga@usuario:~/practica_final$ sed -E 's/\b([0-9]{4})-([0-9]{2})-([0-9]{2}) ([0-9]{2}):([0-9]{2}):([0-9]{2})\b/\1/g' < 911.csv > datos_temp.csv
dsciflafarga@usuario:~/practica_final$ csvcut -c 6 datos_temp.csv | head
timeStamp
2015-12-10
2015-12-10
2015-12-10
2015-12-10
2015-12-10
2015-12-10
2015-12-10
2015-12-10
2015-12-10
dsciflafarga@usuario:~/practica_final$
```

Figura 8: Formateo de la columna timeStamp.

1.4. Tratamiento valores nulos

Como podemos ver en el dataset hay valores nulos, voy a cambiar estos valores por el valor NULL, para poder trabajar mejor con ellos.

Según el campo en que estemos tratando los valores nulos de una forma u otra. En la mayoría de campos los valores nulos se muestran simplemente dejando el campo vacío, salvo en el caso de la latitud y altitud que lo indican con un 0.0000000 y el campo addr que lo indica con un ' '.

Con el siguiente comando transformamos todos estos valores a NULL:

```
awk -F, -v OFS=',' '{for(i=1;i<NF;i++) if(!$i ||$i=="0.000000" || $i==".")$i="NULL"}1'
↪ 911.csv > valores_null.csv
mv valores_null.csv 911.csv
```

Más tarde usaré el comando `csvjoin` el cual transforma estos valores NULL a un campo vacío.

1.5. Creando nuevas variables

Como se ha indicado anteriormente, voy a crear cuatro variables nuevas:

year → Año en el que se ha registrado la llamada.
 month → Mes en el que se ha registrado la llamada.
 day → Día en el que se ha registrado la llamada.
 type → Tipo de llamada.

1.5.1. Variables year, month, day

Para ello creo un archivo temporal con la columna timeStamp, la cual mediante el comando `cut` divido el año, el mes y el día para posteriormente juntarlo con `csvjoin`.

```
csvcut -c 6 911.csv > fecha.csv
#Creo nuevos ficheros con el campo que me interesa
cut -d '-' -f1 < fecha.csv > anio.csv
cut -d '-' -f2 < fecha.csv > mes.csv
cut -d '-' -f3 < fecha.csv > dia.csv
# Cambio nombre cabecera
sed -i '1s/timeStamp/year/' anio.csv
sed -i '1s/timeStamp/month/' mes.csv
sed -i '1s/timeStamp/day/' dia.csv

csvjoin 911.csv anio.csv > 911_join.csv
mv 911_join.csv 911.csv
```

En la [Figura 9](#) vemos como se ha creado la variable Anio, la misma operación se repite para la variable Mes y Dia. Cambiarlo

```

dsciflafarga@usuario: ~/practica_final
Archivo Editar Ver Buscar Terminal Ayuda
dsciflafarga@usuario:~/practica_final$ mv 911_join.csv 911.csv
dsciflafarga@usuario:~/practica_final$ csvcut -n 911.csv
1: lat
2: lng
3: desc
4: zip
5: title
6: timeStamp
7: twp
8: addr
9: Anio
dsciflafarga@usuario:~/practica_final$

```

Figura 9: Formateo de la columna timeStamp y creación de variable anio.

1.5.2. Variable type

Para crear esta variable he elaborado un script llamado crearColumnaTipo.sh:

```

1  #!/bin/bash
2  # crearColumnaTipo.sh
3  # Este script añade una columna llamada Type, el cual corresponde al tipo de la
   ↳ llamada.
4  # Se le debe pasar un archivo como parametro.
5  #
6  # Ejemplo de como queda el csv
7  # -----
8  # |desc                                     |...|type|
9  # |-----+---+---|
10 # |...;Fire: VEHICLE ACCIDENT      |...|Fire|
11 # |...;EMS: RESPIRATORY EMERGENCY|...|EMS |
12 # |...
13 #
14
15 # Compruebo si se le pasa archivo como parámetro o imprimo error.
16 INPUT=${1?Error: debes pasarle un archivo csv como parametro}
17 [ ! -f $INPUT ] && { printf "$INPUT Archivo no encontrado \n "; exit 99; }
18
19 # Leo las columnas del fichero, obtengo el tipo de la llamada y lo guardo en un array
20 i=0 #Contador posicion array
21 while IFS=, read -r lat lng desc zip title timeStamp twp addre anio mes dia tipo; do
22     tipo=$(printf "$title" | cut -d ':' -f1)
23     #TODO: Contemplar caso en que tipo sea null (crear campo con "NULL"). NO HAY CASOS
24     array[$i]="$tipo"
25     let "i++"
26 done < $INPUT
27
28 # Cambio cabecera en el array
29 array[0]="type"
30
31 #Recorro el array y creo un archivo temporal con la columna añadida
32 j=0 #Contador posicion array
33 while read -r; do
34     echo "$REPLY,${array[j]}"
35     let "j++"

```

```

36 done < $INPUT > temp.csv
37
38 # Reemplazo por el archivo original
39 mv temp.csv $INPUT

```

1.6. Dataset tras las modificaciones

Después de todos los cambios hechos en el fichero csv, nos queda un fichero de la siguiente forma:

| lat | lng | desc | zip | title | timeStamp | twp | addr | year | month | day | type |
|--------|---------|---------------------|--------|----------------------------|------------|-------------------|-------------------------------|-------|-------|-----|------|
| 40,298 | -75,581 | 2015-12-10 17:10:52 | 19,525 | EMS: BACK PAINS/INJURY | 2015-12-10 | NEW HANOVER | REINDEER CT & DEAD END | 2,015 | 12 | 10 | EMS |
| 40,258 | -75,265 | 2015-12-10 17:29:21 | 19,446 | EMS: DIABETIC EMERGENCY | 2015-12-10 | HATFIELD TOWNSHIP | BRIAR PATH & WHITEMARSH LN | 2,015 | 12 | 10 | EMS |
| 40,121 | -75,352 | 2015-12-10 14:39:21 | 19,401 | Flre: GAS-ODOR/LEAK | 2015-12-10 | NORRISTOWN | HAMS AVE | 2,015 | 12 | 10 | Fire |
| 40,116 | -75,344 | 2015-12-10 16:47:36 | 19,401 | EMS: CARDIAC EMERGENCY | 2015-12-10 | NORRISTOWN | AIRY ST & SWEDE ST | 2,015 | 12 | 10 | EMS |
| 40,251 | -75,603 | 2015-12-10 16:50:52 | | EMS: DIZZINESS | 2015-12-10 | LOWER POTTS GROVE | CHERRYWOOD CT & DEAD END | 2,015 | 12 | 10 | EMS |
| 40,253 | -75,283 | 2015-12-10 15:39:04 | 19,446 | EMS: HEAD INJURY | 2015-12-10 | LANSDALE | CANNON AVE & W 9TH ST | 2,015 | 12 | 10 | EMS |
| 40,182 | -75,128 | 2015-12-10 16:46:48 | 19,044 | EMS: NAUSEA/VOMITING | 2015-12-10 | HORSHAM | LAUREL AVE & OAKDALE AVE | 2,015 | 12 | 10 | EMS |
| 40,217 | -75,405 | 2015-12-10 16:17:05 | 19,426 | EMS: RESPIRATORY EMERGENCY | 2015-12-10 | SKIPPACK | COLLEGEVILLE RD & LYWISKI RD | 2,015 | 12 | 10 | EMS |
| 40,289 | -75,400 | 2015-12-10 16:51:42 | 19,438 | EMS: SYNCOPAL EPISODE | 2015-12-10 | LOWER SALFORD | MAIN ST & OLD SUMNEYTOWN PIKE | 2,015 | 12 | 10 | EMS |

Figura 10: Visualización del dataset tras modificarlo.

Con estos datos ya podemos plantearnos algunas cuestiones a resolver mediante el uso de técnicas que hemos visto en la asignatura. A continuación expongo algunos ejemplos:

- ¿Cuántos tipos diferentes de llamadas existen y cuál es el tipo que más se repite?

```
csvcut -c type 911.csv | sed 1d | sort -n | uniq -c
```

```

320326 EMS
 96177 Fire
223395 Traffic

```

Figura 11: Tipos diferentes de llamadas con su frecuencia.

Existen 3 tipos y el que más se repite es el tipo EMS.

- ¿Cuántas llamadas se han registrado del tipo EMS en el año 2020?

```
awk -v FS=, '$9 == "2020" && $12 == "EMS" {count++} END {print count}' 911.csv
```

```
dsciflafarga@usuario: ~/practica_final/datos_parte1
Archivo Editar Ver Buscar Terminal Ayuda
dsciflafarga@usuario:~/practica_final/datos_parte1$ awk -v FS=, '$9 == "2020" && $12 == "EMS" {count++} END {print count}' 911.csv
27655
dsciflafarga@usuario:~/practica_final/datos_parte1$
```

Figura 12: Número de llamadas en el año 2020 de tipo EMS.

- ¿Cuántos valores nulos hay en la columna zip?

```
awk -F',' ' $4==" " {count++} END {print count}' 911.csv
```

```
dsciflafarga@usuario: ~/practica_final/datos_parte1
Archivo Editar Ver Buscar Terminal Ayuda
dsciflafarga@usuario:~/practica_final/datos_parte1$ csvcut -c zip 911.csv | grep -c ""
639899
dsciflafarga@usuario:~/practica_final/datos_parte1$
```

Figura 13: Visualización del dataset tras modificarlo.

1.7. Scripts generar tablas

1.7.1. Generar tabla en formato html

Para analizar los datos he creado un script en awk llamado `crear_tabla_html.awk`, al cual se le pasa por parámetro el año(year) y el tipo de llamada(type), de la siguiente forma:

```
./script.awk -v type=EMS year=2015 911.csv > pagina.html
```

Este script filtra los datos que correspondan al tipo de la llamada y al año de la llamada pasados por parámetro y elabora una tabla en html con ellos. En caso de no coincidir ningún dato también lo indica.

El script es el siguiente:

```
1  #!/usr/bin/awk -f
2  # crear_tabla_html.awk
3  # Este script crea una tabla en html con las columnas title, tup y addr de los
4  # campos correspondientes al año y tipo de llamada que se le pase por parametro.
5  # Se le tiene que pasar como parámetro el año y el tipo de la forma:
6  # ./crear_tabla_html.awk -v year=valor_año type=valor_tipo 911.csv
7
8  BEGIN {
9      FS=","
10     print "<html><body></br><h1>Tabla informe archivo 911.csv</h1></br>"
11     print "<table border=1 cellspacing=1 cellpadding=1 >"
12 }
13
14 #Cabecera
```

```

15 NR==1 {
16     printf ("Esta tabla contiene los campos title, twp y addr en el año %s, del
    ↪ tipo de llamada %s. </br></br></br>", year, type)
17     print "<tr>"
18     for ( i = 1; i <= NF; i++ ) {
19         if($i == "title" || $i == "twp" || $i == "addr"){
20             print "<th><b>"$i"</b></th>"
21         }
22     } #Fin for
23     print "</tr>"
24 }
25
26 #Resto líneas
27 NR>1 {
28     if($12 == type && $9 == year) {
29         contador++ #Contador para imprimir cuantos resultados hay
30         print "<tr>"
31         print "<td>"$5"</td><td>"$7"</td><td>"$8"</td>"
32         print "</tr>"
33     } #Fin if
34 }
35
36 END {
37     print "</table></br></br></body></html>"
38     if(contador>0){
39         printf ("Hay un total de: %s resultados",contador)
40     }else{
41         printf("<h2>La tabla esta vacía ya que no hay resultados para el tipo
    ↪ %s y año %s .</h2>",type, year)
42     }
43 }

```

1.7.2. Generar tabla en formato L^AT_EX

He elaborado un script para que genere un tabla en L^AT_EX en la que incluyo el código zip y la frecuencia (número de llamadas registradas) en un año y mes en concreto, los cuales se le tienen que pasar como parámetro al script junto con el archivo csv. El listado esta ordenado en función del código zip. El script es el siguiente:

```

1  #!/bin/bash
2  # crear_tabla_latex.sh
3
4  # Script que genera un fichero .tex con una tabla en la que se registra
5  # la cantidad de llamadas por códigos zip en un año y mes en concreto
6  # $1 = año; $2 = mes; $3 = fichero_csv.csv
7  if [ $# -lt 3 ]; then
8      echo "Tienes que pasarle 3 parámetros y el fichero csv!Solo has pasado $# " && exit
9  fi
10 year=$1
11 month=$2

```

```

12 # arr <- Elimino cabecera, elimino los nulos, saco frecuencia de cada uno
13 arr=$(awk -F , -v anio=$year -v mes=$month '$9==anio && $10==mes {print $4}' $3 | sed
    ↪ 1d | sed '/~$/d' | sort -n | uniq -c))
14
15 len_arr=${#arr[@]}
16 if [ $len_arr -eq 0 ]; then
17     printf "%s\n" "No se han encontrado datos con los parametros pasados por
    ↪ argumentos, la tabla no se creará." && exit
18 fi
19
20 printf "%s\n" "\begin{center}" "\begin{tabular}{cc}" "\textbf{Frecuencia} &
    ↪ \textbf{Código zip} \\\ \hline \hline" > tabla_latex.tex
21 {
22 for i in "${!arr[@]}"
23 do
24     if (( i % 2 != 1 ));then
25         printf '%s & ' "${arr[i]}"
26     else # if (( i % 2 == 1 )); then
27         printf '%s \\\ \n' "${arr[i]}"
28     fi
29 done
30 printf "%s\n" "\\hline" >> tabla_latex.tex
31 printf "%s\n" "\end{tabular}" "\end{center}" >> tabla_latex.tex
32
33 } >> tabla_latex.tex

```

Como este documento lo estoy elaborando en \LaTeX , incluyo las primeras líneas de un ejemplo que he hecho de la tabla y su visualización en el pdf tras ser compilado.

```

\begin{center}
\begin{tabular}{cc}
\textbf{Frecuencia} & \textbf{Código zip} \\
↪ \\\ \hline \hline
39 & 18041 \\\
19 & 18054 \\\
1 & 18056 \\\
    & 3 & 18070 \\\
    & ... & ... \\\
\hline
\end{tabular}
\end{center}

```

| Frecuencia | Código zip |
|------------|------------|
| 39 | 18041 |
| 19 | 18054 |
| 1 | 18056 |
| 3 | 18070 |
| ... | ... |

Figura 14: Código \LaTeX generado por el script y la tabla tras ser compilado.

Basta con poner en el documento .tex

```
\IfFileExists{tabla_latex.tex}{  
\input{tabla_latex.tex}  
}{}  

```

En el caso de que no encuentre datos con el año y mes pasados por parámetros, el script imprime por pantalla un mensaje indicándonos que no creará ninguna tabla, por lo que el código anterior testea si existe o no el fichero antes de importarlo.