

Introducción

Un proyecto de ETL se define como *Extract, Transform y Load* (o extracción, transformación y carga en español) y a continuación expondremos paso a paso este proceso dividido en 3 partes, con 5 transformaciones en total.

Parte 1: Extracción

1.1 Creación de Máquina Virtual con Ubuntu

1.1.1 En primer lugar, descargamos e instalamos VirtualBox desde el sitio web oficial en <https://www.virtualbox.org/> para poder crear una Máquina Virtual (VM). En este caso se muestra la descarga para sistema operativo Windows.

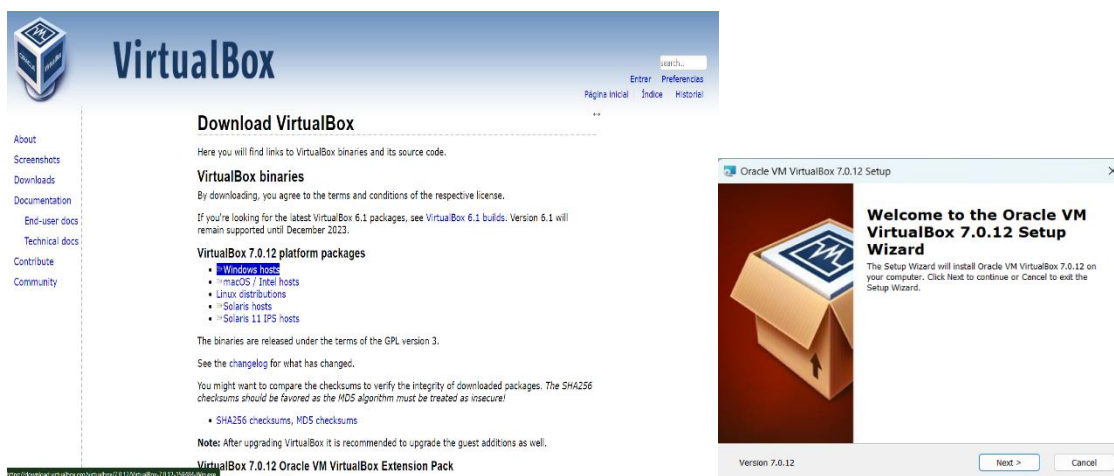


Ilustración 1. Descarga de VirtualBox para Windows. Captura de pantalla desde <https://www.virtualbox.org/>

1.1.2 Una vez descargada e instalada la máquina virtual, procedemos a descargar Ubuntu Desktop en su versión 22.04.3 LTS.

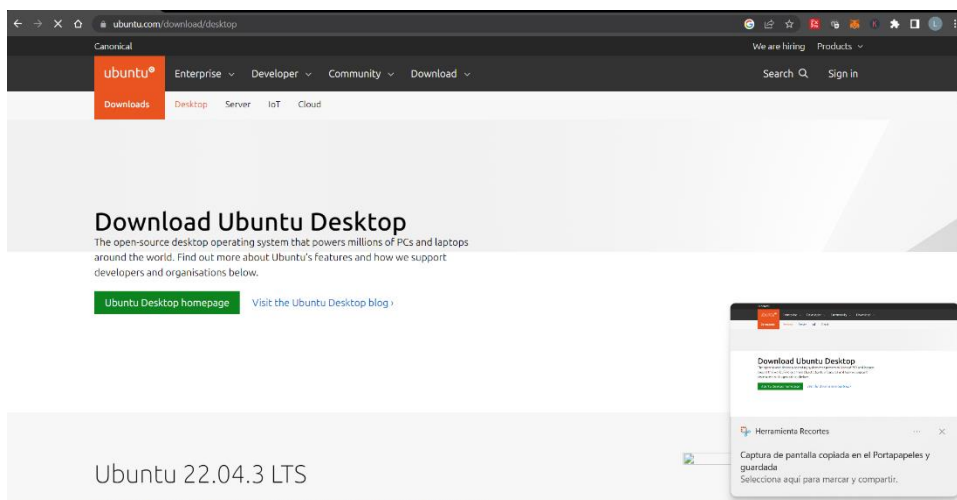


Ilustración 2. Descarga de Ubuntu 22.04.3LTS. Captura de pantalla desde <https://www.ubuntu.com>

1.1.3 Luego de descargar Ubuntu, en el Oracle Virtual Machine, hacemos:

- Click a “New”, ponemos "Ubuntu" como el nombre;
- Confirmamos que estamos de acuerdo con el folder donde va a ser creada;
- Seleccionamos la ISO image buscando la versión de Ubuntu que descargamos
- Hacemos click en “Next”

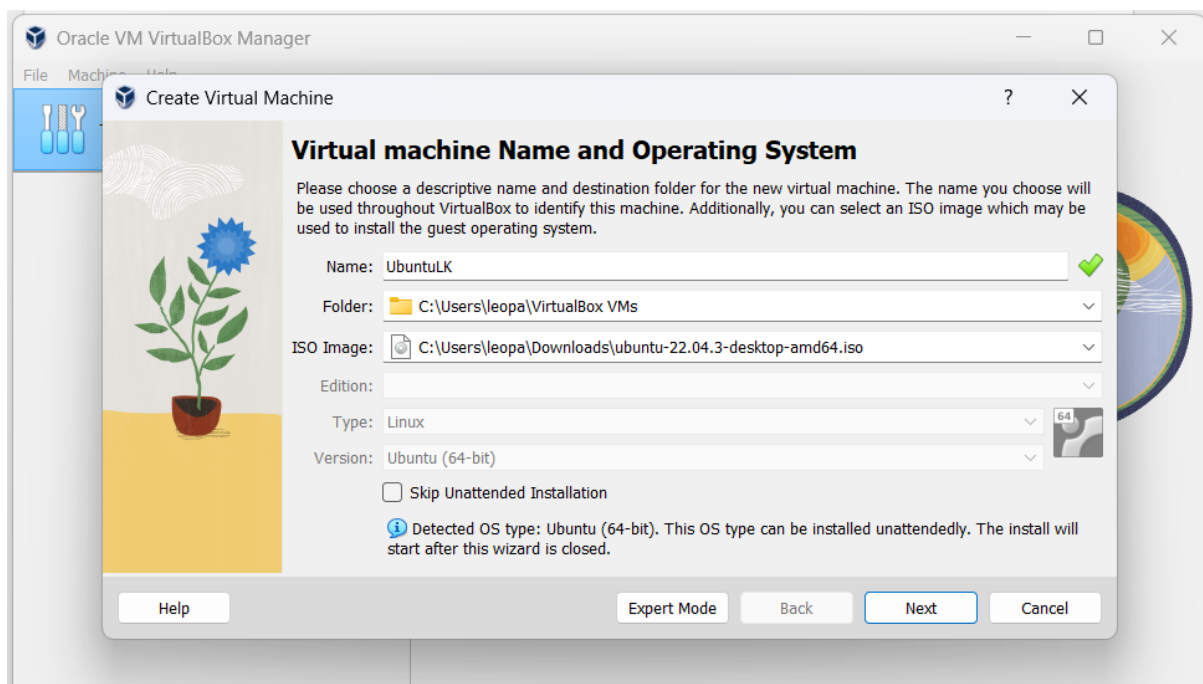


Ilustración 3. Creación de una nueva máquina virtual. Captura de pantalla desde Oracle VM VirtualBox®

1.1.3. Para este caso en particular, seleccionamos 2 GB de RAM y en procesadores le asignamos 2 CPUs. Para disco duro, asignamos 25 GB de espacio, dejando habilitada la opción de preasignar todo el tamaño del disco duro.

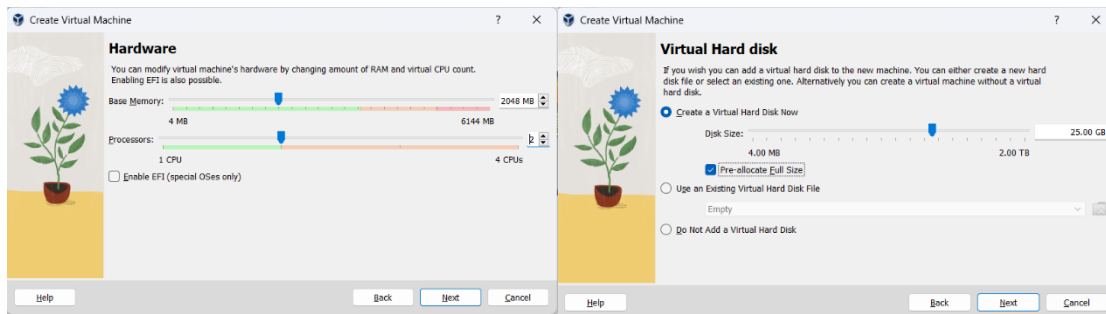
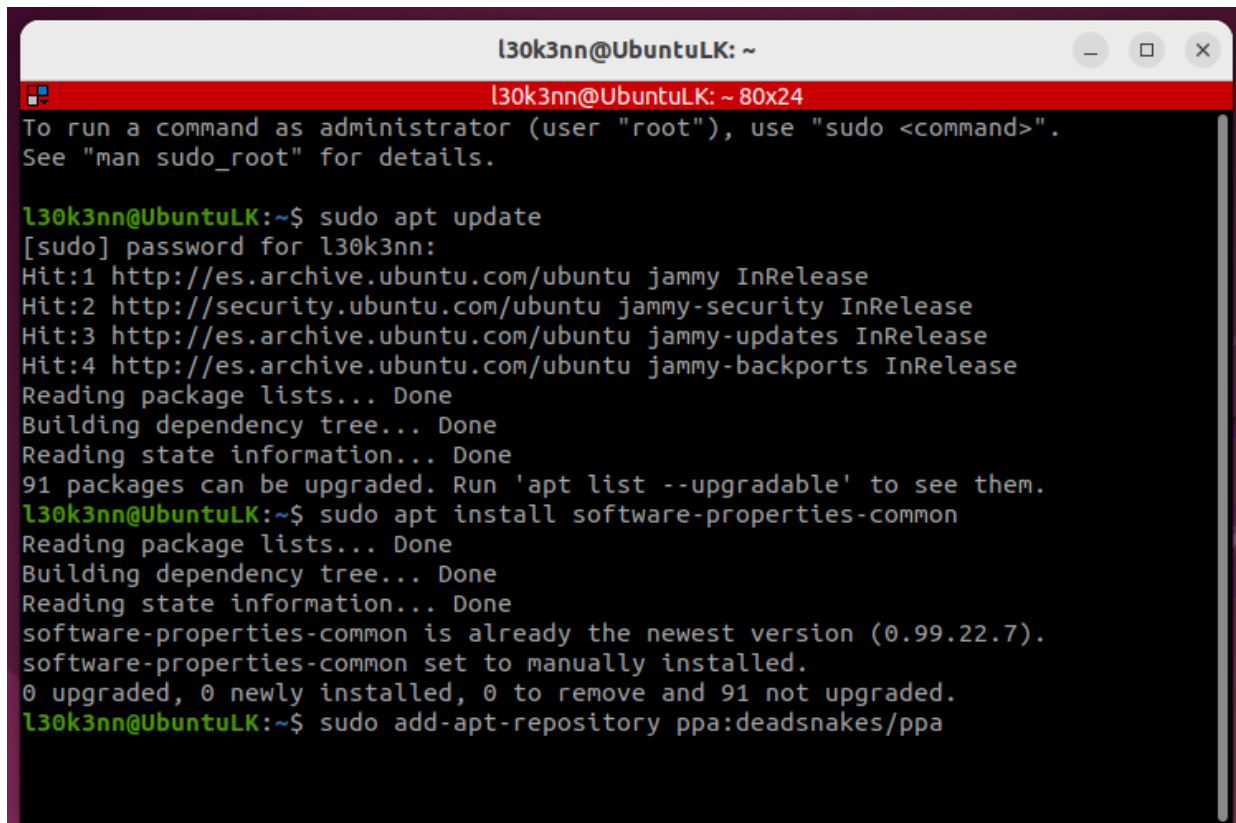


Ilustración 4. Creación de atributos de la nueva máquina virtual creada. Captura de pantalla desde Oracle VM VirtualBox®

1.2 Instalación de Python en Ubuntu

1.2.1 Una vez instalado Ubuntu en la Virtual Machine, procedemos a abrir la terminal de comandos y actualizamos la lista de paquetes disponibles en los repositorios configurados del sistema. Luego invocamos el software que se requiere para instalar Python en su versión actual. Actualizamos también a los lanzamientos de repositorios más recientes, y volvemos a actualizar.

```
$ sudo apt update  
$ sudo apt-get install software-properties-common  
$ sudo add-apt-repository ppa:deadsnakes/ppa  
$ sudo apt update
```

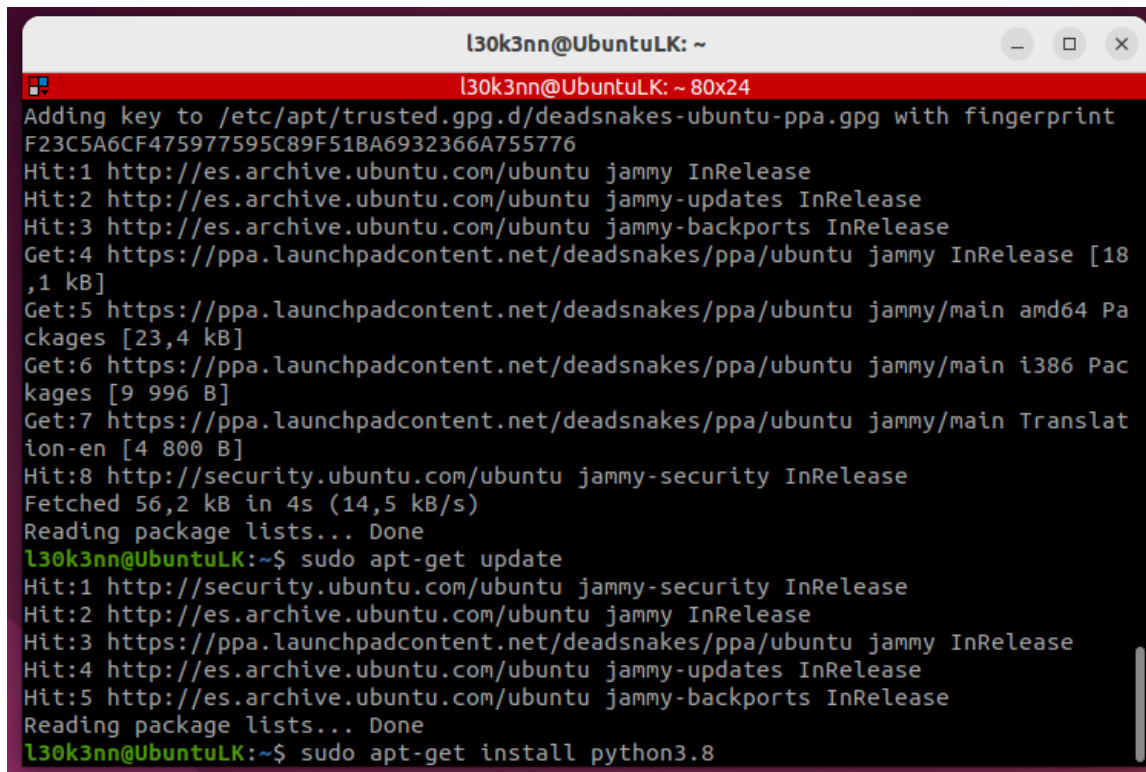
A screenshot of a terminal window titled 'l30k3nn@UbuntuLK: ~'. The window has a red title bar and standard Ubuntu window controls. The terminal text shows a message about using 'sudo' for administrative commands. The user then runs 'sudo apt update', which prompts for a password and shows the process of updating package lists and building a dependency tree. It reports that 91 packages can be upgraded. Next, the user runs 'sudo apt install software-properties-common', which shows that the package is already installed. Finally, the user runs 'sudo add-apt-repository ppa:deadsnakes/ppa'.

```
l30k3nn@UbuntuLK: ~  
To run a command as administrator (user "root"), use "sudo <command>".  
See "man sudo_root" for details.  
  
l30k3nn@UbuntuLK:~$ sudo apt update  
[sudo] password for l30k3nn:  
Hit:1 http://es.archive.ubuntu.com/ubuntu jammy InRelease  
Hit:2 http://security.ubuntu.com/ubuntu jammy-security InRelease  
Hit:3 http://es.archive.ubuntu.com/ubuntu jammy-updates InRelease  
Hit:4 http://es.archive.ubuntu.com/ubuntu jammy-backports InRelease  
Reading package lists... Done  
Building dependency tree... Done  
Reading state information... Done  
91 packages can be upgraded. Run 'apt list --upgradable' to see them.  
l30k3nn@UbuntuLK:~$ sudo apt install software-properties-common  
Reading package lists... Done  
Building dependency tree... Done  
Reading state information... Done  
software-properties-common is already the newest version (0.99.22.7).  
software-properties-common set to manually installed.  
0 upgraded, 0 newly installed, 0 to remove and 91 not upgraded.  
l30k3nn@UbuntuLK:~$ sudo add-apt-repository ppa:deadsnakes/ppa
```

Ilustración 4. Parte I: Instalación de Python en Ubuntu por medio de la Terminal de comandos. Screenshot de Ubuntu.

1.2.2. Una vez actualizado, procedemos a instalar Python y posteriormente a verificar la versión instalada, en este caso instalamos la versión 3.10.12.

```
$ sudo apt-get install python3.8  
$ python3 --version
```



```
l30k3nn@UbuntuLK: ~  
l30k3nn@UbuntuLK: ~ 80x24  
Adding key to /etc/apt/trusted.gpg.d/deadsnakes-ubuntu-ppa.gpg with fingerprint  
F23C5A6CF475977595C89F51BA6932366A755776  
Hit:1 http://es.archive.ubuntu.com/ubuntu jammy InRelease  
Hit:2 http://es.archive.ubuntu.com/ubuntu jammy-updates InRelease  
Hit:3 http://es.archive.ubuntu.com/ubuntu jammy-backports InRelease  
Get:4 https://ppa.launchpadcontent.net/deadsnakes/ppa/ubuntu jammy InRelease [18  
,1 kB]  
Get:5 https://ppa.launchpadcontent.net/deadsnakes/ppa/ubuntu jammy/main amd64 Pa  
ckages [23,4 kB]  
Get:6 https://ppa.launchpadcontent.net/deadsnakes/ppa/ubuntu jammy/main i386 Pac  
kages [9 996 B]  
Get:7 https://ppa.launchpadcontent.net/deadsnakes/ppa/ubuntu jammy/main Translat  
ion-en [4 800 B]  
Hit:8 http://security.ubuntu.com/ubuntu jammy-security InRelease  
Fetched 56,2 kB in 4s (14,5 kB/s)  
Reading package lists... Done  
l30k3nn@UbuntuLK:~$ sudo apt-get update  
Hit:1 http://security.ubuntu.com/ubuntu jammy-security InRelease  
Hit:2 http://es.archive.ubuntu.com/ubuntu jammy InRelease  
Hit:3 https://ppa.launchpadcontent.net/deadsnakes/ppa/ubuntu jammy InRelease  
Hit:4 http://es.archive.ubuntu.com/ubuntu jammy-updates InRelease  
Hit:5 http://es.archive.ubuntu.com/ubuntu jammy-backports InRelease  
Reading package lists... Done  
l30k3nn@UbuntuLK:~$ sudo apt-get install python3.8
```

Ilustración 5. Parte II: Instalación de Python en Ubuntu por medio de la Terminal de comandos. Screenshot de Ubuntu.

1.3. Instalación de paquetería para Python

1.3.1 Ahora instalamos Python y el gestor de paquetería:

```
$python3  
Instalación  
$apt update  
$sudo apt update  
$sudo apt -y upgrade  
Para verificar Instalación de Python  
$python3 -V  
Instalación de gestor de paquetes de dependencias  
$sudo apt install -y python3-pip  
Para verificar Instalación del gestor  
$pip3 -V  
Instalacion de dependencias en entorno profesional  
$apt install -y build-essential libssl-dev libffi-dev python3-dev
```

```

l30k3nn@UbuntuLK:~$ sudo apt install ipython3
[sudo] password for l30k3nn:
Sorry, try again.
[sudo] password for l30k3nn:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
ipython3 is already the newest version (7.31.1-1).
0 upgraded, 0 newly installed, 0 to remove and 30 not upgraded.
l30k3nn@UbuntuLK:~$ sudo apt-get install python3-setuptools
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
python3-setuptools is already the newest version (59.6.0-1.2ubuntu0.22.04.1).
0 upgraded, 0 newly installed, 0 to remove and 30 not upgraded.
l30k3nn@UbuntuLK:~$ sudo add-apt-repository universe
Adding component(s) 'universe' to all repositories.
Press [ENTER] to continue or Ctrl-c to cancel.
Hit:1 http://es.archive.ubuntu.com/ubuntu jammy InRelease
Get:2 http://es.archive.ubuntu.com/ubuntu jammy-updates InRelease [119 kB]
Get:3 http://security.ubuntu.com/ubuntu jammy-security InRelease [110 kB]
Hit:4 https://ppa.launchpadcontent.net/deadsnakes/ppa/ubuntu jammy InRelease
Get:5 http://es.archive.ubuntu.com/ubuntu jammy-backports InRelease [109 kB]
Get:6 http://es.archive.ubuntu.com/ubuntu jammy-updates/main amd64 DEP-11 Metadata [101 kB]
Get:7 http://es.archive.ubuntu.com/ubuntu jammy-updates/universe amd64 DEP-11 Metadata [305 kB]
Get:8 http://es.archive.ubuntu.com/ubuntu jammy-updates/multiverse amd64 DEP-11 Metadata [940 B]
Get:9 http://security.ubuntu.com/ubuntu jammy-security/main amd64 DEP-11 Metadata [43,0 kB]
Get:10 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 DEP-11 Metadata [55,2 kB]
Get:11 http://es.archive.ubuntu.com/ubuntu jammy-backports/main amd64 DEP-11 Metadata [4 932 B]
Get:12 http://es.archive.ubuntu.com/ubuntu jammy-backports/universe amd64 DEP-11 Metadata [18,8 kB]
Fetched 866 kB in 3s (272 kB/s)
Reading package lists... Done
l30k3nn@UbuntuLK:~$ sudo apt-get install python3-pip
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
python3-pip is already the newest version (22.0.2+dfsg-1ubuntu0.3).
0 upgraded, 0 newly installed, 0 to remove and 30 not upgraded.

```

Ilustración 6. Parte II: Instalación de paquetería para Python en Ubuntu por medio de la Terminal de comandos. Screenshot de Ubuntu.

Parte 2: Transformación

2.1. Creamos un nuevo archivo Python, lo llamamos “codIfraestructuras.py”, y en nuestra Virtual Machine abrimos un IDE (en nuestro caso VSCode).

2.2.1 Abrimos el archivo “codIfraestructuras.py” e importamos las siguientes librerías:

- “requests” (se utiliza para realizar solicitudes HTTP y obtener datos desde una URL externa)
- “statistics” (se utiliza para realizar operaciones estadísticas, como calcular la media o la moda)
- “mysql.connector” (se utiliza para interactuar con bases de datos MySQL, estableciendo conexiones y creando cursores)

2.2.2 Usamos la función “requests.get” para obtener datos de una URL que genera usuarios aleatorios. El parámetro “results=100” limita el número de datos que recibimos a 100.

```

1 #Se importa la libreria que se van a utilizar en el proyecto
2 import requests
3 import mysql.connector
4 import statistics
5
6 #Por medio de este comando se hace el llamado de los datos de la URL
7 #El igual despues de la URL sirve para limitar el número de datos
8 url = requests.get('https://randomuser.me/api?results=100')
9
10 #Convierte la información de la URL en un formato tipo JSON
11 informacion = url.json()
12
13 #Hace el llamado a las claves del diccionario
14 informacion.keys()
15
16 #Creamos una variable con los datos que estan en la llave 'results'
17 resultados = informacion['results']

```

2.3. Transformación I

La primera transformación consiste en encontrar la edad más frecuente en la que los usuarios se registraron. Para ello, extraemos sus edades actuales y el tiempo que llevan registrados. Luego calculamos la diferencia entre estas para obtener nuestra respuesta, y proporcionamos *insights* sobre estas (como la edad media, moda, máxima y mínima).

A continuación presentamos una forma de extraer el porcentaje de usuarios en cada rango de edad, clasificados en las categorías de edades [0-20], [21-40], [41-60], y [61+]. El formato del resultado debe seguir el patrón de “El X% de usuarios se registraron a una edad menor de X años.”

```

20 #####
21 #Muestra de información 1
22 #Saca el porcentaje de personas que se registraron en diferentes rangos de edad
23
24 registro =[] #Lista que contendrá los elementos del registro
25 nacimiento =[] #Lista que contendrá las edades actuales
26 for i in range(len(informacion['results'])): #se itera entre el rango dado.
27     edadReg = informacion['results'][i]['registered']['age'] #Los años desde que se dió de alta en la página
28     edadAct = informacion['results'][i]['dob']['age'] #Edad actual
29     registro.append(edadReg) #Agrega un elemento a la lista
30     nacimiento.append(edadAct) #Agrega un elemento a la lista
31
32 edad_registro = [] #Lista para las edades en las cual se dieron de alta en la página
33 for i in range(len(informacion['results'])): #Itera en el rango dado
34     edad_registro.append(nacimiento[i] - registro[i]) #realiza la operación, restando la edad actual con la edad de registro
35 edad_registro # se hace el llamado de la lista
36 #import statistics #Se importa la libreria de estadística
37 statistics.mean(edad_registro) #Método que calcula la media
38
39
40 statistics.mode(edad_registro) #Calcula la moda
41 max(edad_registro) #Calcula la edad máxima actual
42 min(edad_registro) #Calcula la edad mínima actual

```



```

44 #####
45 def rango_etario(listValores=[]):
46     longitud = len(listValores)
47     edades = {'0-20':0,'21-40':0,'41-60':0, '61+':0} #Guarda los rangos de edades
48
49     for edad in listValores: #Para cada edad
50         if edad < 21: #Si es menor que 21
51             edades['0-20'] = edades['0-20'] + 1 #Agrego el elemento a esta llave
52         if (edad >= 21 and edad < 41)==True:
53             edades['21-40'] = edades['21-40'] + 1 #Agrego el elemento a esta llave
54         if (edad > 40 and edad < 61)==True:
55             edades['41-60'] = edades['41-60']+ 1 #Agrego el elemento a esta llave
56         if edad > 60:
57             edades['61+'] = edades['61+'] + 1 #Agrego el elemento a esta llave
58     resultEdad1 = (edades['0-20']/longitud*100) #Valor de la Key '0-20'
59     resultEdad2 = (edades['21-40']/longitud*100) #Valor de la Key '21-40'
60     resultEdad3 = (edades['41-60']/longitud*100) #Valor de la Key '41-60'
61     resultEdad4 = (edades['61+']/longitud*100) #Valor de la Key '61+'
62
63     return(resultEdad1,resultEdad2,resultEdad3,resultEdad4) #Retorna los valores a una lista
64
65     rango_etario(edad_registro)

```

2.4. Transformación II

En la segunda transformación nos centramos en buscar el país con el mayor número de usuarios definiendo la función “numUsuarios”. Para ello, realizamos un análisis para encontrar la moda, o el país más frecuente en las listas de “location” y “country”. Nuestra función devuelve el nombre del país y la cantidad de usuarios que tienen en ese país. Esto podría ser de interés si se quiere averiguar cuál es el país con mayor número de usuarios en la plataforma. El formato del resultado debe seguir el patrón de “El país en donde se encuentra el mayor número de usuarios es: X con X personas.”

```

70 #Muestra de información 2
71 #País con el mayor número de usuarios
72 pais = informacion['results'][2]['location']['country'] #Realiza una búsqueda del país
73 def numUsuarios(): #Función para hallar el país con mayor número de usuarios
74     vModaUsuPais = 0 #Se le asigna a la variable 0
75     paises = [] #Se crea una lista de paises
76     for i in range(len(informacion['results'])): #Itera en el rango de la lista
77         paises.append(informacion['results'][i]['location']['country']) #Se añaden los paises a la lista
78     modaPais = statistics.mode(paises) #Se halla la moda en la lista paises
79     for i in range(len(paises)): #Itera en en la longitud de la lista dada (paises)
80         if paises[i] == modaPais: #La condición saca el nombre del país que sea igual a la moda
81             vModaUsuPais = vModaUsuPais + 1 #Reasigna la variable de la moda de los usuarios por país
82
83     return(modaNPaís,vModaUsuPais) #Retorna los valores a una lista
84
85
86
87 numUsuarios() #Se hace el llamado de la función

```

2.5. Transformación III

El tercer análisis consiste en evaluar el grado de seguridad de la contraseña de cada usuario, utilizando funciones para examinar la complejidad de las contraseñas y clasificarlas en tres grados según los siguientes criterios:

- una seguridad de grado 1 incluye 2 caracteres repetidos
- una de grado 2 tiene solo letras o solo números
- una de grado 3 tiene una combinación de tanto letras como números

Las funciones “separar_letras” y “separar_ltrs_num” se encargan de contar las repeticiones de letras y números, respectivamente. La función “grado_seguridad”

determina el grado de seguridad según nuestros criterios, y la función “seguridad” calcula y nos muestra el porcentaje de contraseñas en cada grado.

El formato del resultado debe seguir el patrón de “La contraseña de grado 1 representan un X%, las de grado 2 representan un X% y las de grado 3 un X%.”

```
90 #Muestra de información 3
91 #grado de seguridad de la contraseña
92 #Grado 1: 2 caracteres repetidos
93 #Grado 2: Solo letras o solo números
94 #Grado 3: Combinación de letras y números
95
96
97 def separar_letras(palabra = str): #Crea la función para separar las letras
98     letras_dic = dict() #Guarda repetición de letras
99     contador = 0 #Caracteres que se repiten
100
101     for letra in palabra: #Por cada letra
102         if letra in letras_dic: #Si ya estaba en el dic() significa que se repite
103             if letras_dic[letra] == 1:
104                 contador += 1 #Se agrega al contador
105                 letras_dic[letra] += 1 #Continúa el conteo
106             else:
107                 letras_dic[letra] = 1 #Si la letra no está en el diccionario, la agrega
108     return letras_dic
109
110
111 def separar_ltrs_num(palabra):
112     abc = 'qwertyuiopasdfghjklzxcvbnm' #Se establece un str de las letras
113     numeros = '0123456789' #Se establece un str de los números
114     letras_rep = {'abcd':0,'nume':0,'especial':0} #Guarda repetición de letras
115
116     for letra in palabra: #Por cada letra
117         if letra in numeros: #Si el elemento es un número
118             letras_rep['nume'] += 1 #Continúa el conteo
119         if letra in abc:
120             letras_rep['abcd'] += 1 #Si es una letra
121         else: #Si resulta un carácter diferente
122             letras_rep['especial'] += 1
123     return letras_rep
124
125
126 def grado_seguridad(contra =str): #Crea la función del grado de seguridad
127     sepaLtrs = separar_letras(contra) #Guarda la información en el diccionario
128     sepaLtrsNum = separar_ltrs_num(contra) #Guarda la información en el diccionario
129     for i in sepaLtrs.keys(): #Itera en las letras separadas
130         if sepaLtrs[i] > 1: #Cuando sea mayor a 1
131             return 0 #Vuelve a realizar el proceso
132         break
133     if (sepaLtrsNum['abcd'] >= 1 and sepaLtrsNum['nume'] >= 1) == True: #Si las letras y números repetidos es mayor que 1
134
135         return 2
136     else:
137
138         return 1
139
140 #####
141
142 def seguridad(): #Crea la variable seguridad
143     grados = {'grado1':0, 'grado2':0, 'grado3':0} #Se inician las variables en 0
144     for i in range(len(informacion['results'])): #Itera en los resultados
145         selecGrado = grado_seguridad(informacion['results'][i]['login']['password']) #Se le asigna el diccionario a la variable
146         if selecGrado == 0: #Revisa si se encuentra en grado 1
147             grados['grado1'] = grados['grado1'] + 1
148         if selecGrado == 1: #Revisa si se encuentra en grado 2
149             grados['grado2'] = grados['grado2'] + 1
150         if selecGrado == 2: #Revisa si se encuentra en grado 3
151             grados['grado3'] = grados['grado3'] + 1
152
153     SegGrado1 = (grados['grado1']/len(informacion['results'])*100) #Valor de la Key '0-20'
154     SegGrado2 = (grados['grado2']/len(informacion['results'])*100) #Valor de la Key '21-40'
155     SegGrado3 = (grados['grado3']/len(informacion['results'])*100) #Valor de la Key '41-60'
156     #print('Las contraseñas de grado 1 representan un:',SegGrado1,'% , las de grado 2 un:',SegGrado2,'% y las de grado 3 un:',SegGrado3,'%')
157     return(SegGrado1,SegGrado2,SegGrado3)#Retorna los valores a una lista
158     seguridad()
```

2.6. Transformación IV

La cuarta transformación analiza la distribución de la zona horaria de los usuarios. Utilizando la función “zona_horaria” recopilamos información sobre las diferencias de usos horarios de cada persona e identificamos la zona horaria más común (junto con el número de personas en esa zona).

```

162 #Muestra de información 4
163 #Distribucion de la zona horaria
164
165 informacion['results'][2]['location']['timezone']['offset'] #
166 def Zona_horaria(): #Crea la función
167     numUsuarios = 0 #Inicializa la variable en 0
168     horario = [] #Crea una lista
169     for i in range(len(informacion['results'])): #Itera en el resultado
170         horario.append(informacion['results'][i]['location']['timezone']['offset']) #Agrega el elemento a la lista horario
171     modaZonHr = statistics.mode(horario) #Se saca la moda a la lista horario
172     for i in range(len(horario)): #Itera en la lista
173         if horario[i] == modaZonHr: #Si el horario es igual a la moda buscada en la lista
174             numUsuarios = numUsuarios + 1 #Reasigna la variable número de usuarios
175     #print ('La mayor zona horaria se encuentra entre las:',modaZonHr,f'con {numUsuarios} usuarios')
176     return(modaZonHr,numUsuarios)
177
178
179 Zona_horaria() #Llama la función

```

El formato de los resultados debe seguir el patrón de “la mayor zona horaria se encuentra entre las X con X usuarios” y mostrar la lista.

2.7. Transformación V

Aquí identificamos la persona más joven y la persona más vieja de la lista de usuarios.

```

183 #Muestra de información 5
184 # Inicializamos variables para la persona más joven y la persona más vieja.
185 #Estas se utilizarán para mantener un seguimiento de la persona más joven
186 # y la persona más vieja, respectivamente.
187
188 #Al principio, se establecen en None porque aún no hemos encontrado ninguna persona.
189 personamasjovencita = None
190 personamasviejita = None
191
192
193 for person in resultados: #Itera entre los resultados
194     age = person['dob']['age'] #Busca la edad
195     city = person['location']['city'] #Busca la ciudad
196
197     if personamasjovencita is None or age < personamasjovencita['age']: #Busca la persona mas joven
198         personamasjovencita = {'age': age, 'city': city}
199     if personamasviejita is None or age > personamasviejita['age']: #Busca la ciudad de la persona mas joven
200         personamasviejita = {'age': age, 'city': city}

```

Utilizamos las variables “personamasjovencita” y “personamasviejita” con valor “none”, y luego iteramos a través de los resultados, obteniendo la edad y ciudad de cada. Finalmente, comparamos la edad del usuario actual con las edades de los usuarios más jóvenes y más mayores registrados en nuestra base de datos.

2.8. Presentación de Resultados

¿Qué edad que tenían los usuarios cuando se dieron de alta en la plataforma?

```
#Muestra 1
vEdad1 = rango_etario(edad_registro)[0]
vEdad2 = rango_etario(edad_registro)[1]
vEdad3 = rango_etario(edad_registro)[2]
vEdad4 = rango_etario(edad_registro)[3]

print("Muestra de información 1: ¿Qué edad que tenían los usuarios cuando se dieron de alta en la plataforma?")
print("El", + vEdad1,"% de usuarios se registraron a una edad menor de 20 años.")
print("El", + vEdad2,"% de usuarios se registraron a una edad entre los 21 - 40 años.")
print("El", + vEdad3,"% de usuarios se registraron a una edad entre los 41 - 60 años.")
print("El", + vEdad4,"% de usuarios se registraron a una edad mayor de 60 años.")
```

Muestra de información 1: ¿Qué edad que tenían los usuarios cuando se dieron de alta en la plataforma?
 El 15.0 % de usuarios se registraron a una edad menor de 20 años.
 El 36.0 % de usuarios se registraron a una edad entre los 21 - 40 años.
 El 34.0 % de usuarios se registraron a una edad entre los 41 - 60 años.
 El 15.0 % de usuarios se registraron a una edad mayor de 60 años.

¿Cuál es el país con mayor número de usuarios en la plataforma?

```
#Muestra 2
vPaís = numUsuarios()[0]
vUsuPaís = numUsuarios()[1]

print("Muestra de información 2:¿Cuál es el país con mayor número de usuarios en la plataforma?")
print("El país en donde se encuentra el mayor número de usuarios es:", vPaís, "con", vUsuPaís, "personas" )
```

Muestra de información 2:¿Cuál es el país con mayor número de usuarios en la plataforma?
 El país en donde se encuentra el mayor número de usuarios es: France con 8 personas

¿Qué tan seguras son las contraseñas que usan los usuarios de la plataforma?

```
#Muestra 3
vSeguridad1 = seguridad()[0]
vSeguridad2 = seguridad()[1]
vSeguridad3 = seguridad()[2]

print("Muestra de información 3:¿Qué tan seguras son las contraseñas que usan los usuarios de la plataforma?")
print("Las contraseñas de grado 1 representan un:",vSeguridad1,"% , las de grado 2 un:",vSeguridad2,'% y las de grado 3 un:',
      vSeguridad3,'%')
```

Las contraseñas de grado 1 representan un: 61.0 %, las de grado 2 un: 35.0 % y las de grado 3 un: 4.0 %
 Las contraseñas de grado 1 representan un: 61.0 %, las de grado 2 un: 35.0 % y las de grado 3 un: 4.0 %
 Las contraseñas de grado 1 representan un: 61.0 %, las de grado 2 un: 35.0 % y las de grado 3 un: 4.0 %
 Muestra de información 3:¿Qué tan seguras son las contraseñas que usan los usuarios de la plataforma?
 Las contraseñas de grado 1 representan un: 61.0 %, las de grado 2 un: 35.0 % y las de grado 3 un: 4.0 %

¿En qué zona horaria se conectan más los usuarios de la plataforma?

```
#Muestra 4
vZonHr = Zona_horaria()[0]
vUsuZonHr = Zona_horaria()[1]

print("Muestra de información 4:¿En qué zona horaria se conectan más los usuarios de la plataforma?")
print("La mayor zona horaria se encuentra entre las:",vZonHr, "con",vUsuZonHr,"usuarios")
```

Muestra de información 4:¿En qué zona horaria se conectan más los usuarios de la plataforma?
 La mayor zona horaria se encuentra entre las: +7:00 con 8 usuarios

¿Qué edad tiene el usuario más joven y el de mayor edad y de qué ciudad son?

```
#Muestra 5
edPerJov = personasmasjovencita['age']
ciuPerJov = personasmasjovencita['city']

edPerMay = personasmasviejita['age']
ciuPerMay = personasmasviejita['city']

print("Muestra de información 5:¿Qué edad tiene el usuario más joven y el de mayor edad y de qué ciudad son?")
print("La edad de la persona mas joven es: ",edPerJov, "con una edad de: ",ciuPerJov)
print("La edad de la persona mas vieja es: ",edPerMay, "con una edad de: ",ciuPerMay)

Muestra de información 5:¿Qué edad tiene el usuario más joven y el de mayor edad y de qué ciudad son?
La edad de la persona mas joven es: 23 con una edad de: Bor
La edad de la persona mas vieja es: 79 con una edad de: Enterprise
```

Parte 3: Carga

3.1. La parte final consiste en que carguemos la información a una base de datos, y a continuación mostramos una de las formas de almacenar los resultados de nuestro ETL para poder analizar y realizar un seguimiento posterior.

Primero, obtenemos todas las variables necesarias a partir de varias funciones. Después, establecemos una conexión a la base de datos y creamos un cursor para facilitar la interacción. A continuación, definimos la consulta SQL (“InsertInfo”) que nos permitirá insertar los datos en la tabla “infraestructuras” y ejecutamos la transacción utilizando la función “execute” para llevar a cabo la inserción de datos. Finalmente, confirmamos la transacción mediante “commit”, asegurándonos de esta forma que todos los cambios se guarden correctamente en la base de datos.

```
253 #Se establece la conexión con la base de datos
254 mydb = mysql.connector.connect(
255     host="localhost",
256     user="root",
257     password="Root",
258     database="bdd_turnos"
259 )
260 #Crea la interacción con la base de datos
261 mycursor = mydb.cursor()
262
263 #Establece a que tabla va a ir la información y a que columnas
264 InsertInfo = """INSERT INTO infraestructuras
265 (UsuRegMenor20, UsuRegEnt20Y40, UsuRegEnt40Y60, PaisMaxUsu, NumUsuPais,
266 PassSegGrado1, PassSegGrado2, PassSegGrado3, MaxZonaHoraria, NumUsuZonHor,
267 EdadPerJoven,CiudPerJoven,EdadPerMayor,CiudPerMayor)
268 VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)"""
269
270 #Se añaden las variables donde esta almacenada la información
271 variables = (vEdad1, vEdad2, vEdad3, vEdad4, vPais, vUsuPais,
272             vSeguridad1, vSeguridad2, vSeguridad3, vZonHr,
273             vUsuZonHr, edPerJov, ciuPerJov, edPerMay, ciuPerMay)
274
275 #Ejecuta la transacción de información
276 mycursor.execute(InsertInfo, variables)
277
278 # Confirmar la transacción
279 mydb.commit()
280
281 #Imprime si fue con exito la transacción de información
282 print(mycursor.rowcount, "Se ingreso la información con éxito")
```

Esta tabla nos muestra la carga de la información en formato de lista, con ejemplos de pruebas que se hicieron para verificar la conexión y la subida a la base de datos.

```
1 ● SELECT * FROM bdd_infraestructuras.infraestructuras;
```

[illegible]