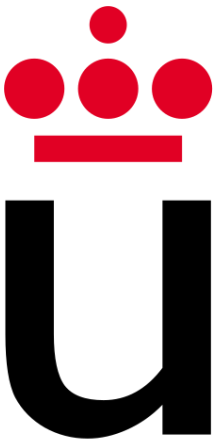


Universidad
Rey Juan Carlos

Práctica 2

Sistemas de Información

Ingeniería de la Ciberseguridad

Curso 2021-2022

Víctor Gallego
Óscar Mora
Iván Dominguez

Índice

GitHub

https://github.com/DguezzZ/SdI-Recopilacion-estructuracion-y-analisis-de-datos

Introducción

En esta práctica realizaremos un CMI para ver una representación visual de los datos obtenidos. Para ello utilizaremos el framework de Flask, el cuál nos ayudará a crear de manera sencilla una página web que nos permita ver los datos e interactuar con las distintas opciones de visualización de estos. Siendo el resultado final el siguiente:

Ejercicio 2

Para la implementación de estas funcionalidades, hemos desarrollado dos funciones que seleccionan los distintos elementos de la base de datos necesarios para la realización de cálculos que nos permitan ordenar por criticidad tanto a los usuarios como a las webs. Estos elementos han sido, en la función *usuarios_criticos,* el id del usuario, el número de emails de phishing recibidos y el número de emails clicados. Dividimos los emails clicados entre los emails de phishing recibidos y, dependiendo del valor obtenido, se ordena en el dataframe de mayor a menor. Para la representación en la web de esta funcionalidad hemos creado la ruta */vulnerables.* En esta hemos creado un gráfico de barras que representa la probabilidad de click y el id del usuario.

```python
@app.route('/webvulnerables')
def pagvulnerables():
    df=webs_vulnerables(int(request.args.get('value',default=20)))
    grafico=alt.Chart(df).mark_bar().encode(x="url",y="Seguridad")
    return render_template('webvulnerables.html',grafico=grafico.to_json(),value=int(request.args.get('value',default=20)))
```

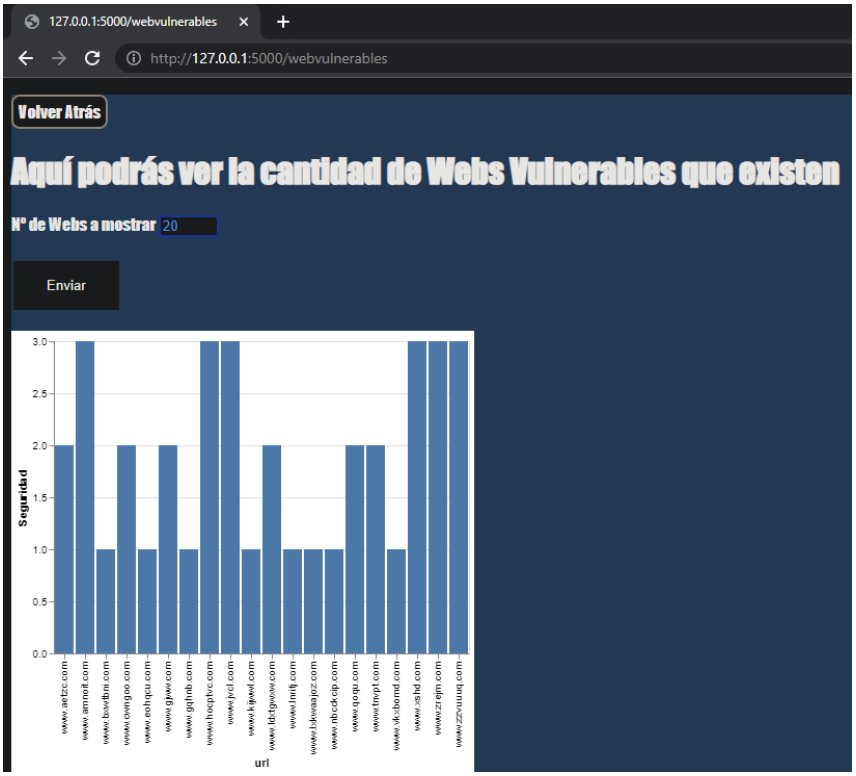
```python
@app.route('/vulnerables')
def vulnerables():
    df=usuarios_criticos(int(request.args.get('value',default=20)))
    grafico=alt.Chart(df).mark_bar().encode(x="id",y="probabilidad_click")
    dfspam=usuarios_spam(int(request.args.get('mayor',default=0)))
    return render_template('vulnerables.html',grafico=grafico.to_json(),value=int(request.args.get('value',default=20)),tablaUser=dfspam.to_html(),mayor=int(request.args.get('mayor',default=0)))
```

	ID	TELEFONO	PROVINCIA	EMAILS_TOTAL	EMAILS_PHISHING	EMAILS_CLICADOS
0	ines.diaz	62274087	Avila	48	23	4
1	alicia.castro	67187870	Burgos	221	165	17
2	directivo	73756283	Caceres	428	143	4
3	administrador	None	None	493	276	50
4	contrataciones	None	None	441	205	92

Para representar las webs más vulnerables ha sido un proceso similar al descrito anteriormente. Hemos seleccionado los elementos url, cookies, aviso y proteccion_de_datos. En el dataframe hemos organizado de menor a mayor las webs dependiendo del valor obtenido después de evaluar su seguridad. Si dispone de cookies, de algún tipo de aviso de seguridad o si dispone de protección de datos, se añade un 1 por cada uno de estos. Siendo las más vulnerables las que menos valor tienen.
Para mostrar los datos en la web hemos creado la ruta */webvulnerables* y añadido una representación en forma de gráfico en la que se muestran las webs más vulnerables. El número de webs a mostrar las puede seleccionar el usuario en todo momento.

```python
def webs_vulnerables(top: int):
    web = pd.read_sql_query("SELECT url, cookies, aviso, proteccion_de_datos FROM legal", con)
    web["Seguridad"] = web["cookies"] + web["aviso"] + web["proteccion_de_datos"]
    web = web.sort_values("Seguridad", ascending=True).head(top)
    return web
```

```python
@app.route('/webvulnerables')
def pagvulnerables():
    df=webs_vulnerables(int(request.args.get('value',default=20)))
    grafico=alt.Chart(df).mark_bar().encode(x="url",y="Seguridad")
    return render_template('webvulnerables.html',grafico=grafico.to_json(),value=int(request.args.get('value',default=20)))
```

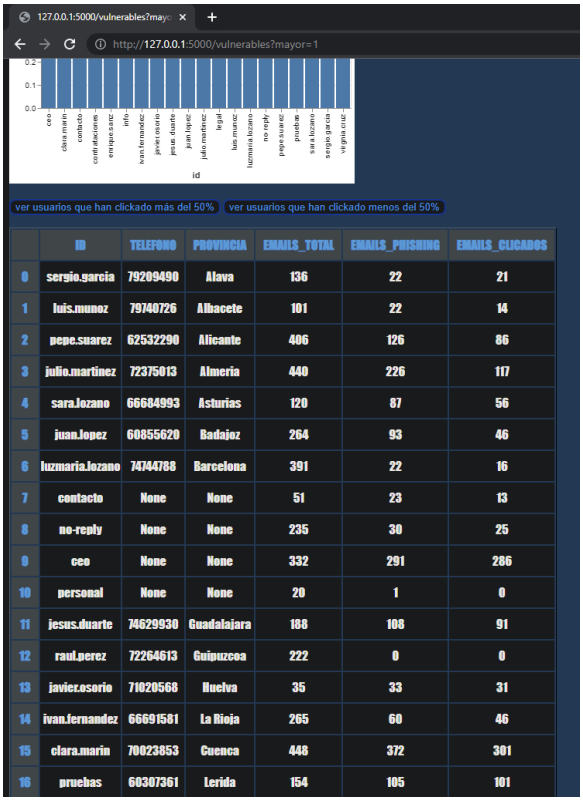


Ejercicio 3

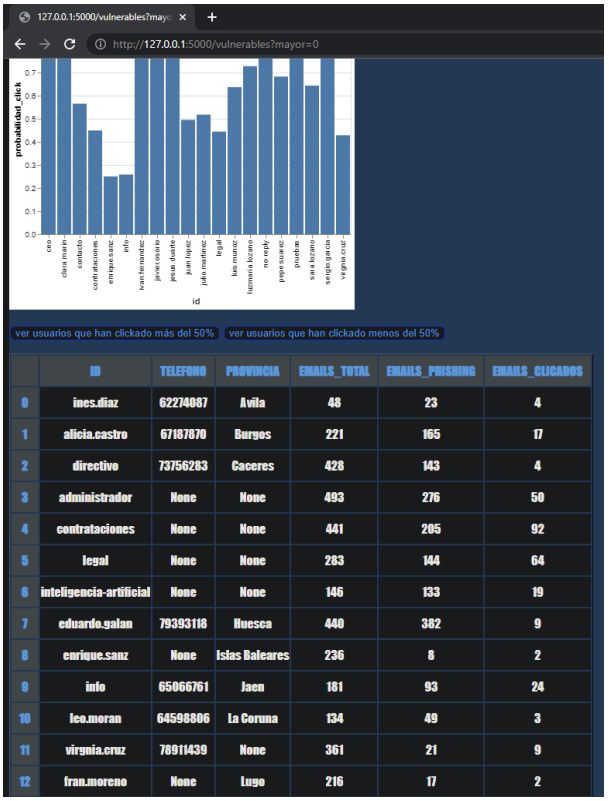
Esta funcionalidad se ha conseguido mediante el desarrollo de una función en la cual, dependiendo de si quieres que se muestren los usuarios que han pulsado más del 50% de veces los correos de spam o los que han pulsado menos del 50%, se devuelve toda la información del usuario. Con la comprobación de que el número de emails clicados sea mayor o menor (dependiendo de la selección) que la mitad del número de emails de phishing recibidos se seleccionan y devuelven a los usuarios afectados. La funcionalidad se ha añadido a la ruta */vulnerables*, pudiendo seleccionar, a parte de los usuarios más vulnerables, si se quiere ver los que “pican” con mayor probabilidad.

```python
def usuarios_spam(mayor: int):
    if bool(mayor):
        usr = pd.read_sql_query("SELECT id, telefono, provincia, emails_total, emails_phishing, emails_clicados FROM usuarios where emails_clicados>=usuarios.emails_phishing/2", con)
    else:
        usr = pd.read_sql_query("SELECT id, telefono, provincia, emails_total, emails_phishing, emails_clicados FROM usuarios where emails_clicados<usuarios.emails_phishing/2", con)
    return usr
```

```python
@app.route('/vulnerables')
def vulnerables():
    df=usuarios_criticos(int(request.args.get('value',default=20)))
    grafico=alt.Chart(df).mark_bar().encode(x="id",y="probabilidad_click")
    dfspam=usuarios_spam(int(request.args.get('mayor',default=0)))
    return render_template('vulnerables.html',grafico=grafico.to_json(),value=int(request.args.get('value',default=20)),tablaUser=dfspam.to_html(),mayor=int(request.args.get('mayor',default=0)))
```



	ID	TELEFONO	PROVINCIA	EMAILS_TOTAL	EMAILS_PHISHING	EMAILS_CLICADOS
0	sergio.garcia	79209490	Alava	136	22	21
1	luis.munoz	79740726	Albacete	101	22	14
2	pepe.suarez	62532290	Alicante	406	126	86
3	julio.martinez	72375013	Almeria	440	226	117
4	sara.lozano	66684993	Asturias	120	87	56
5	juan.lopez	60855620	Badajoz	264	93	46
6	luzmaria.lozano	74744788	Barcelona	391	22	16
7	contacto	None	None	51	23	13
8	no-reply	None	None	235	30	25
9	ceo	None	None	332	291	286
10	personal	None	None	20	1	0
11	jesus.duarte	74629930	Guadalajara	188	108	91
12	raul.perez	72264613	Guipuzcoa	222	0	0
13	javier.osorio	71020568	Huelva	35	33	31
14	ivan.fernandez	66691581	La Rioja	265	60	46
15	clara.marin	70023853	Cuenca	448	372	301
16	pruebas	60307361	Lerida	154	105	101



	ID	TELEFONO	PROVINCIA	EMAILS_TOTAL	EMAILS_PHISHING	EMAILS_CLICADOS
0	ines.diaz	62274087	Avila	48	23	4
1	alicia.castro	67187870	Burgos	221	165	17
2	directivo	73756283	Caceres	428	143	4
3	administrador	None	None	493	276	50
4	contrataciones	None	None	441	205	92
5	legal	None	None	283	144	64
6	inteligencia-artificial	None	None	146	133	19
7	eduardo.galan	79393118	Huesca	440	382	9
8	enrique.sanz	None	Islas Baleares	236	8	2
9	info	65066761	Jaen	181	93	24
10	leo.moran	64598806	La Coruna	134	49	3
11	virginia.cruz	78911439	None	361	21	9
12	fran.moreno	None	Lugo	216	17	2

Ejercicio 4

Para mostrar las últimas 10 vulnerabilidades hemos realizado una función que, mediante requets, capta la web *https://www.cve-search.org/api/* y la guardamos en formato de texto. Después creamos un dataframe en el que añadimos los ids y la información de cada vulnerabilidad y, por último, se devuelven las 10 últimas en formato html. Para representar esta información hemos creado la ruta */ultimasVulnerabilidades*. En esta se representan las vulnerabilidades en forma de tabla, con el id de la vulnerabilidad y una breve explicación de esta.

```python
def ultimas_vul():
    respuesta = requests.get("https://www.cve-search.org/api/")
    if respuesta.status_code != 200:
        raise Exception
    else:
        json = respuesta.text
        data = pd.DataFrame()
        data["summary"] = pd.read_json(json)["summary"]
        data["id"] = pd.read_json(json)["id"]
        return data.head(10).to_html()
```

```python
@app.route('/ultimasVulnerabilidades', methods = ['GET'])
def ultimas_vulnerabilidades():

    respuesta = requests.get("https://cve.circl.lu/api/last")
    if respuesta.status_code != 200:
       raise Exception
    else:
        archivo = respuesta.text
        df = pd.DataFrame()
        df["id"] = pd.read_json(archivo)["id"]
        df["summary"] = pd.read_json(archivo)["summary"]
    return render_template('ultimasVulnerabilidades.html',lista=df.head(10).to_html())
```

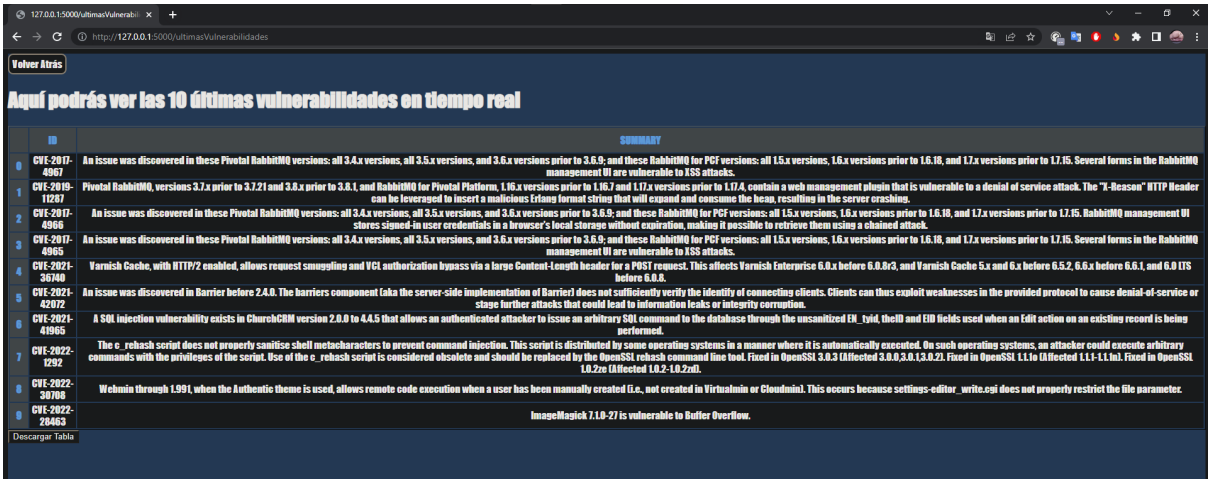
Ejercicio 5

Para añadir más información de valor a nuestro CMI hemos creado una función que busca productos disponibles de distintos servicios. Mediante requets, captamos la web *http://cve.circl.lu/api/browse/* que nos aporta información sobre los productos de los que disponen distintos servicios. Estos se muestran en forma de tabla en la ruta */obtenerservicios.* También hemos añadido la posibilidad de descargar en formato .pdf la información de las 10 últimas vulnerabilidades.

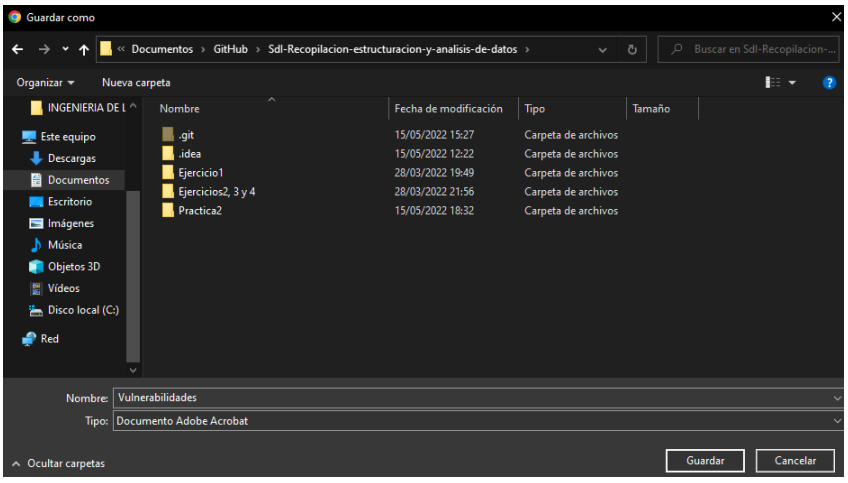
```python
def buscador_servicios2(servicio: str):
    respuesta = requests.get("http://cve.circl.lu/api/browse/"+urllib.parse.quote(servicio))
    if respuesta.status_code != 200:
        raise Exception
    else:
        json = respuesta.text
        data = pd.DataFrame()
        if "product" in pd.read_json(json):
            data["product"] = pd.read_json(json)["product"]
        return data["product"]
```

```python
@app.route('/obtenerservicios', methods = ['GET'])
def obtenerservicio():
    #return render_template('ultimasVulnerabilidades.html',lista=buscador_servicios(request.args.get('servicio')))
    return render_template('obtenerServiciosv2.html',lista=buscador_servicios2(request.args.get('servicio', default="Microsoft")),servicio=request.args.get('servicio', default="Microsoft"))
```

```
<link rel="stylesheet" href="/static/css/ultimasVulns.css">
<body>
<p><button2 class="backbutton" onclick="history.go(-1)"><span>Volver Atrás</span></button2></p>
<h1>Aquí podrás ver las 10 Últimas vulnerabilidades en tiempo real </h1>
 {% autoescape off %}
<div style="...">
<table style="...">{{ lista }}</table>
</div>
 {% endautoescape %}
<input type="button" id="btnExport" value="Descargar Tabla" onclick="Export()" />
    <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/pdfmake/0.1.22/pdfmake.min.js"></script>
    <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/html2canvas/0.4.1/html2canvas.min.js"></script>
    <script type="text/javascript">
        function Export() {
            html2canvas(document.getElementsByClassName('dataframe'), {
                onrendered: function (canvas) {
                    var data = canvas.toDataURL();
                    var docDefinition = {
                        pageOrientation: 'landscape',
                        content: [{
                            image: data,
                            width: 750,
                        }]
                    };
                    pdfMake.createPdf(docDefinition).download("Vulnerabilidades.pdf");
                }
            });
        }
    </script>
</body>
```



	ID	SUMMARY
0	CVE-2017-4967	An issue was discovered in these Pivotal RabbitMQ versions: all 3.4.x versions, all 3.5.x versions, and 3.6.x versions prior to 3.6.9; and these RabbitMQ for PCF versions: all 1.5.x versions, 1.6.x versions prior to 1.6.18, and 1.7.x versions prior to 1.7.15. Several forms in the RabbitMQ management UI are vulnerable to XSS attacks.
1	CVE-2019-11287	Pivotal RabbitMQ, versions 3.7.x prior to 3.7.21 and 3.8.x prior to 3.8.1, and RabbitMQ for Pivotal Platform, 1.16.x versions prior to 1.16.7 and 1.17.x versions prior to 1.17.4, contain a web management plugin that is vulnerable to a denial of service attack. The "X-Reason" HTTP Header can be leveraged to insert a malicious Erlang format string that will expand and consume the heap, resulting in the server crashing.
2	CVE-2017-4966	An issue was discovered in these Pivotal RabbitMQ versions: all 3.4.x versions, all 3.5.x versions, and 3.6.x versions prior to 3.6.9; and these RabbitMQ for PCF versions: all 1.5.x versions, 1.6.x versions prior to 1.6.18, and 1.7.x versions prior to 1.7.15. RabbitMQ management UI stores signed-in user credentials in a browser's local storage without expiration, making it possible to retrieve them using a chained attack.
3	CVE-2017-4965	An issue was discovered in these Pivotal RabbitMQ versions: all 3.4.x versions, all 3.5.x versions, and 3.6.x versions prior to 3.6.9; and these RabbitMQ for PCF versions: all 1.5.x versions, 1.6.x versions prior to 1.6.18, and 1.7.x versions prior to 1.7.15. Several forms in the RabbitMQ management UI are vulnerable to XSS attacks.
4	CVE-2021-36740	Varnish Cache, with HTTP/2 enabled, allows request smuggling and VCL authorization bypass via a large Content-Length header for a POST request. This affects Varnish Enterprise 6.0.x before 6.0.8r3, and Varnish Cache 5.x and 6.x before 6.5.2, 6.6.x before 6.6.1, and 6.0 LTS before 6.0.8.
5	CVE-2021-42072	An issue was discovered in Barrier before 2.4.0. The barriers component (aka the server-side implementation of Barrier) does not sufficiently verify the identity of connecting clients. Clients can thus exploit weaknesses in the provided protocol to cause denial-of-service or stage further attacks that could lead to information leaks or integrity corruption.
6	CVE-2021-41965	A SQL injection vulnerability exists in ChurchCRM version 2.0.0 to 4.4.5 that allows an authenticated attacker to issue an arbitrary SQL command to the database through the unsanitized EN_tyid, theID and EID fields used when an Edit action on an existing record is being performed.
7	CVE-2022-1292	The c_rehash script does not properly sanitise shell metacharacters to prevent command injection. This script is distributed by some operating systems in a manner where it is automatically executed. On such operating systems, an attacker could execute arbitrary commands with the privileges of the script. Use of the c_rehash script is considered obsolete and should be replaced by the OpenSSL rehash command line tool. Fixed in OpenSSL 3.0.3 (Affected 3.0.0,3.0.1,3.0.2). Fixed in OpenSSL 1.1.1o (Affected 1.1.1-1.1.1n). Fixed in OpenSSL 1.0.2ze (Affected 1.0.2-1.0.2zd).
8	CVE-2022-30708	Webmin through 1.991, when the Authentic theme is used, allows remote code execution when a user has been manually created (i.e., not created in Virtualmin or Cloudmin). This occurs because settings-editor_write.cgi does not properly restrict the file parameter.
9	CVE-2022-28463	ImageMagick 7.1.0-27 is vulnerable to Buffer Overflow.

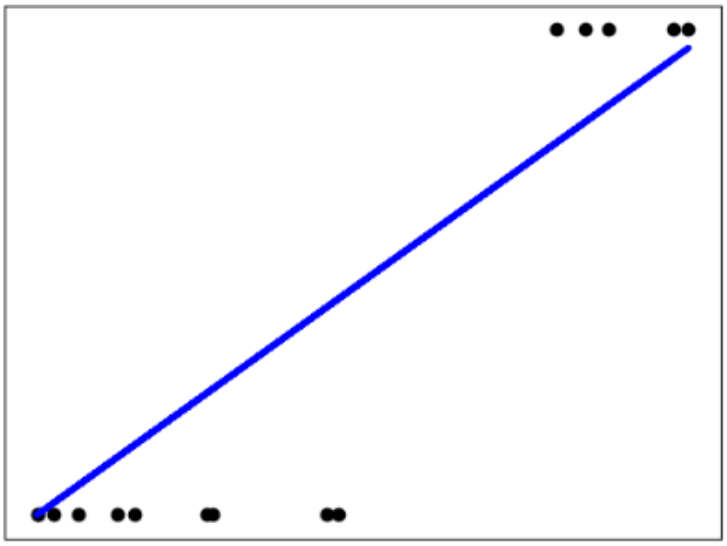


Ejercicio 6

En el ejercicio 6 nos separaremos de todo lo desarrollado hasta el momento con Flask para hacer un pequeño experimento de Machine Learning. Se nos facilitan dos archivos JSON, uno con la etiqueta sobre la que pretendemos entrenar y otro con nuevos datos sin dicha etiqueta. En primer lugar pasaremos dichos archivos JSON a arrays de almacenamiento y de esos arrays nos crearemos dos dataframes. El primero contendrá la probabilidad de clickar en un email phising y el segundo será el de la etiqueta con el valor de la variable “vulnerable”. Usando los códigos facilitados en el aula para los diferentes tipos de predicción haremos una adaptación a nuestros datos y generaremos un sistema entrenado.

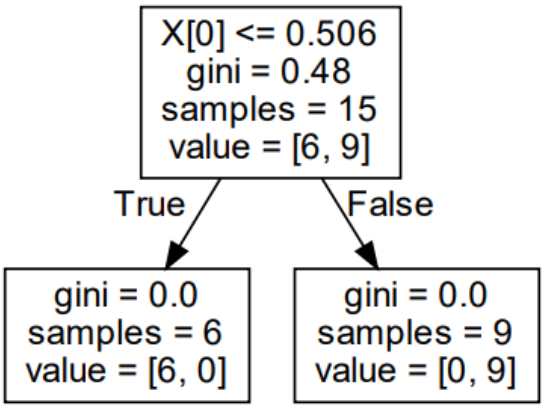
Una vez tenemos el sistema entrenado, queremos usar el segundo JSON sin etiquetas para que nos muestre los resultados tras pasarlo por dichos sistemas entrenados.

Los resultados obtenidos serán varios:
Una gráfica de la línea de regresión lineal+ coeficientes



```
[0.93571429]
[0.95833333]
[0.625     ]
[0.17419355]
[0.76315789]
[0.85      ]
[0.68125   ]
[0.35185185]
[0.53571429]
[0.94957983]
[0.60989011]
[0.05128205]
[0.34090909]
[0.67676768]
[0.66666667]
[0.24324324]
[0.625     ]]
[1 1 1 0 1 0 1 0 1 0 1 1 1 1 1 1 0 1 1 1 0 1 1 1 0 0 1 1 0 1]
[1 1 1 0 1 0 1 0 1 0 1 1 1 1 1 1 0 1 1 1 0 1 1 1 0 0 1 1 0 1]
```

Una imagen con el árbol de decisión y los parámetros del mismo



Varias imágenes con los arboles de decisión que componen nuestro random forest.