صبا ابراهیمی ۹۸۲۴۳۰۰۲

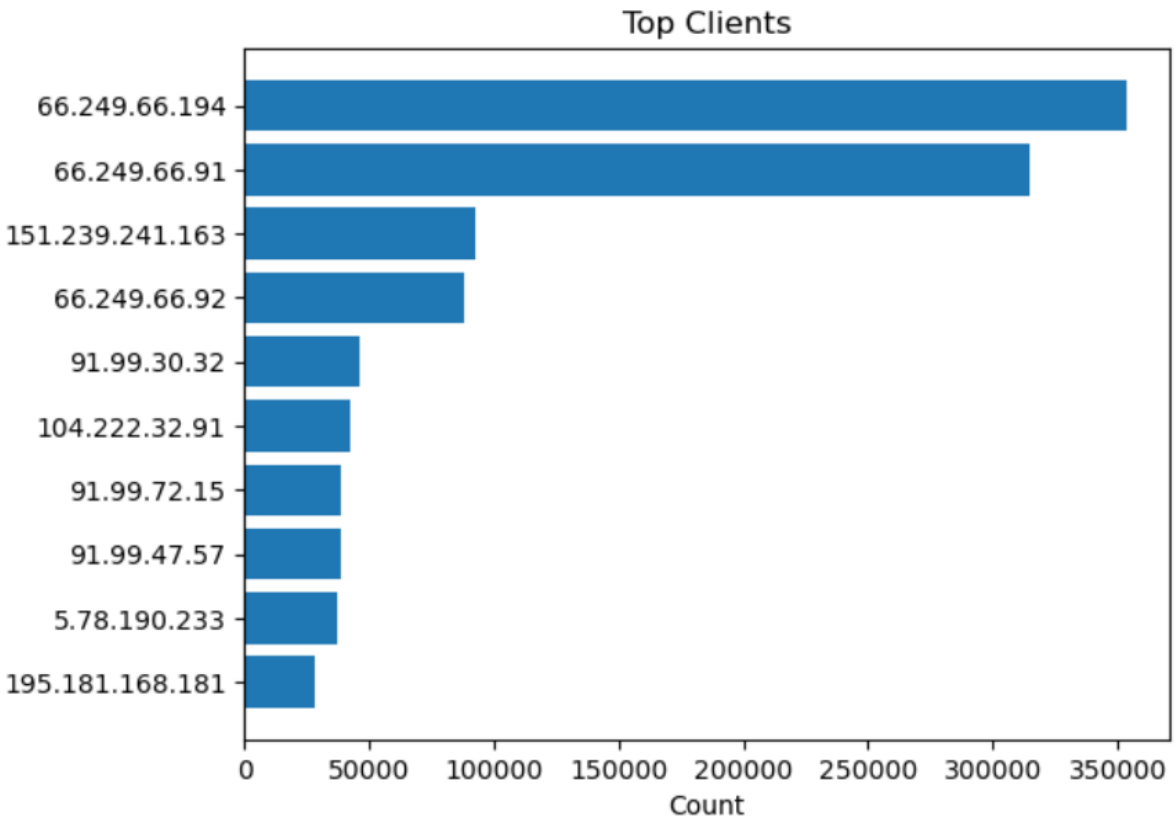
Q1

در این سوال هدف به دست آوردن بیشترین IP بازدید کننده بود.IP ها در ستون client قرار دارند. برای به دست آوردن تعداد unique هر کدام از مقادیر IP، از تابع values_count استفاده می‌کنیم.

```
############ Your code here ############
top_n_visited_clients = df['client'].value_counts()[:n].sort_values(ascending=False).to_frame()
```

دیتافریم top_n_visited_clients دارای index حاوی IP و یک ستون حاوی تعداد هر کدام است. یک rename انجام می‌دهیم تا مقادیر نام ستون درست شود. در نهایت نمودار را Plot می‌کنیم.

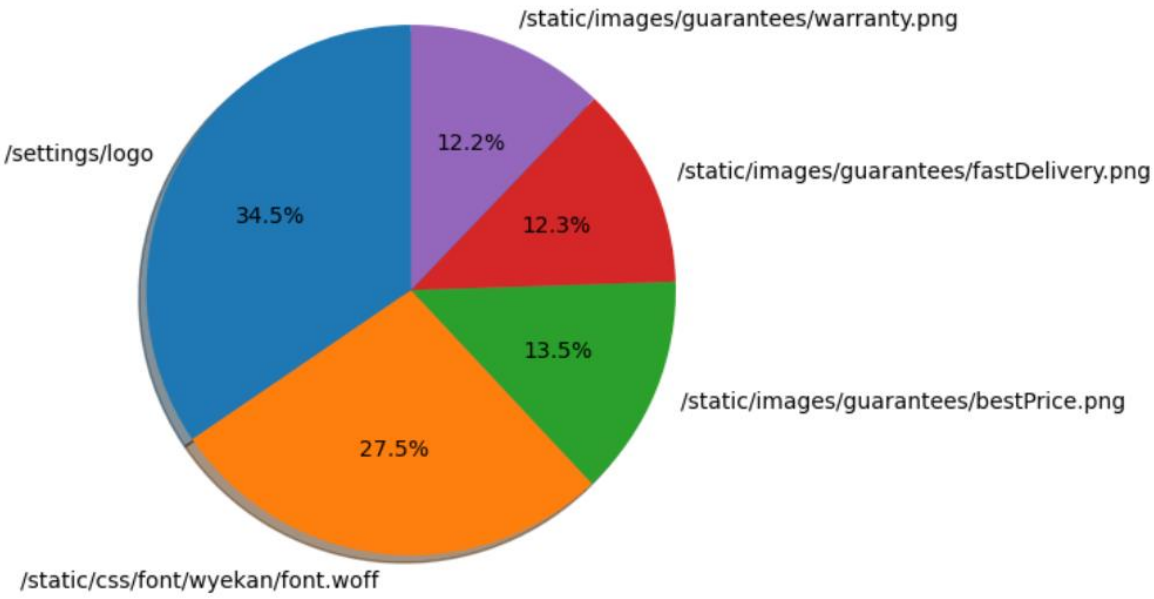
مقادیر دو ستون را تبدیل به numpy می‌کنیم تا بتوان با استفاده از matplotlib آن را نمایش داد.



client	count
66.249.66.194	353483
66.249.66.91	314522
151.239.241.163	92473
66.249.66.92	88332
91.99.30.32	45973
104.222.32.91	42058
91.99.72.15	38694
91.99.47.57	38609
5.78.190.233	37203
195.181.168.181	27979

Q2

در این سوال هدف به دست آوردن بیشترین تعداد endpointهای درخواست شده است. دقیقا مانند سوال اول عمل می‌کنیم. با این تفاوت که به جای خواندن از ستون client، از ستون request استفاده می‌کنیم.



به نظر می‌آید که بیشترین request، برای گرفتن logoی وبسایت بوده.

endpoint	count
/settings/logo	352047
/static/css/font/wyekan/font.woff	280176
/static/images/guarantees/bestPrice.png	138010
/static/images/guarantees/fastDelivery.png	125689
/static/images/guarantees/warranty.png	124127

Q3

در این سوال، regex برای تشخیص query param را اینگونه تعریف می‌کنیم:

```
query_regex = '\?.+$'
```

برای حذف query param از end pointها، از تابع apply استفاده می‌کنیم. با استفاده lambda function، تمام کوئری پارامترها را حذف می‌کنیم و در صورتی که هر کدام از endpointها .jpg, .png, .jpeg, .webp بودند، پسوندشان را نگه می‌داریم و در غیر این صورت همه‌ی query param را حذف می‌کنیم.
بعد از آن، رجکسی که برای تصاویر واقعی است را اینطور تعریف می‌کنیم:

```
real_image_regex = '(/image.+\.jpg)|(/image.+\.jpeg)|(/image.+\.png)|(/image.+\.webp)'
```

بقیه‌ی تصاویر اینregex را دارند:

```
all_images_regex = '/image.+'
```

در نهایت تعداد کل تصاویر با رجکس اولی و تعداد کل تصاویر با رجکس دومی را حساب می‌کنیم و نسبت می‌گیریم.

0.1422222323610473

Out[37]:

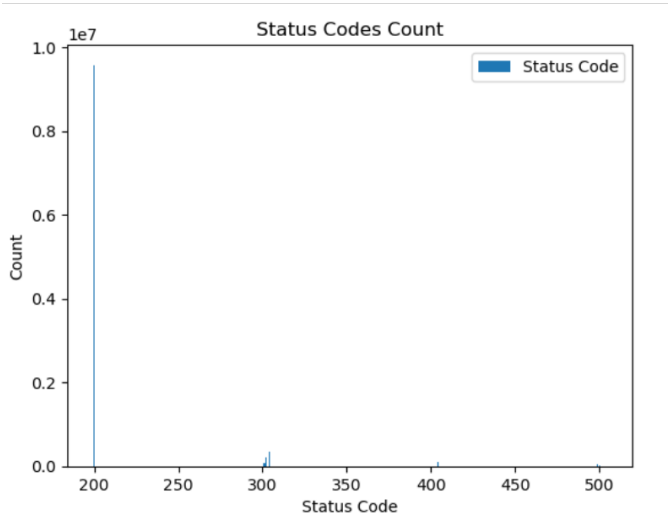
	client	datetime	method	request	status	size	referer	use
0	37.152.163.59	2019-01-22 12:38:27+03:30	GET	/image/29314.jpg	200	1105	https://www.zanbil.ir/product/29314/%DA%A9%D8%...	Mc (Windows WOW64; Trk
1	37.152.163.59	2019-01-22 12:38:27+03:30	GET	/static/images/zanbil-kharid.png	200	358	https://www.zanbil.ir/product/29314/%DA%A9%D8%...	Mc (Windows WOW64; Trk
2	85.9.73.119	2019-01-22 12:38:27+03:30	GET	/static/images/next.png	200	3045	https://znbl.ir/static/bundle-bundle_site_head...	Mc (Windows Win64; x64)
3	37.152.163.59	2019-01-22 12:38:27+03:30	GET	/image/29314.jpg	200	1457	https://www.zanbil.ir/product/29314/%DA%A9%D8%...	Mc (Windows WOW64; Trk
4	85.9.73.119	2019-01-22 12:38:27+03:30	GET	/static/images/checked.png	200	1083	https://znbl.ir/static/bundle-bundle_site_head...	Mc (Windows Win64; x64)
5	37.152.163.59	2019-01-22 12:38:27+03:30	GET	/static/images/loading.gif	200	7370	https://www.zanbil.ir/product/29314/%DA%A9%D8%...	Mc (Windows WOW64; Trk
6	77.245.233.52	2019-01-22 12:38:27+03:30	GET	/image/11082/productType/240x180	200	12458	https://www.zanbil.ir/browse/sports/%D8%AA%D8%...	Mc (Windows rv:64.0) Ge
7	37.27.128.139	2019-01-22 12:38:27+03:30	GET	/browse/Tablet-Arm-Chair/%D8%B5%D9%86%D8%AF%D9...	200	30604	https://www.zanbil.ir/browse/Classroom-Furnitu...	Mc (Windows AppleWebKit
8	77.245.233.52	2019-01-22 12:38:27+03:30	GET	/image/851/mainSlide	200	89859	https://www.zanbil.ir/browse/sports/%D8%AA%D8%...	Mc (Windows rv:64.0) Ge
9	77.245.233.52	2019-01-22 12:38:27+03:30	GET	/image/848/mainSlide	200	93168	https://www.zanbil.ir/browse/sports/%D8%AA%D8%...	Mc (Windows rv:64.0) Ge

Q4

در این سوال در مرحله اول باید تمام status code ها را پلات کنیم. برای این کار ابتدا یک دیتافریم به نام status_codes_count_df می‌سازیم و در آن تعداد status code ها را نگه‌داری می‌کنیم.

```
status_codes_list = status_codes_count_df.index.to_numpy()
```

این لیست همان index های دیتافریم status_codes_count_df است. سپس همین دیتافریم را پلات می‌کنیم:



از این نمودار می‌بینیم که درخواست‌هایی که با موفقیت رو به رو می‌شوند، با اختلاف زیاد بیشتر از بقیه است. بعد از آن، درخواست‌های 3xx، بعد 4xx و در نهایت 5xx ها در رتبه‌های بعدی قرار دارند.

برای بخش دوم این سوال، ابتدا یک ستون hour به logs_df اضافه می‌کنیم. این ستون را با این دستور اضافه می‌کنیم:

```
logs_df['hour'] = logs_df['datetime'].dt.hour
```

سپس یک دیتافریم به نام only_error_df می‌سازیم که در آن تنها سطرهایی از log_df وجود دارد که مقدار ارور دارند (یعنی کد ۴۰۰ و کد ۵۰۰). بعد از آن یک دیتافریم دیگر می‌سازیم که این only_error_df را تغییر می‌دهد. یعنی ستون‌های با مقدار 400 تا 499 را به 4xx و مقادیر 500 تا 599 را به 5xx تغییر می‌دهد. این اتفاق با استفاده از تابع apply انجام می‌شود. ستون hour را به این دیتافریم جدید اضافه می‌کنیم.

درنهایت با استفاده از pivot_table، این عملیات را انجام می‌دهیم:

```
pd.pivot_table(only_error_changed, values=['status_count'], index=['hour'], columns=['status'], aggfunc='count')
```

این دستور یعنی روی دیتافریم only_error_changed، بر اساس مقادیر hour، استاتوس‌ها را دسته بندی کن و مقادیر ستون‌هایی که از status به دست آمده را بر اساس تابع count به دست بیاور.

	status_count	
status	4xx	5xx
hour		
0	6021	17
1	4215	10
2	2394	8
3	1388	26
4	1871	3
5	1683	10
6	2036	3
7	2923	6
8	4985	76
9	7574	220
10	9117	124
11	10102	255
12	10395	429
13	10095	528
14	10584	597
15	10513	581
16	9823	621
17	9351	202
18	8979	4816
19	8302	6550
20	7758	36
21	7019	6
22	7342	22
23	7786	21

Q5

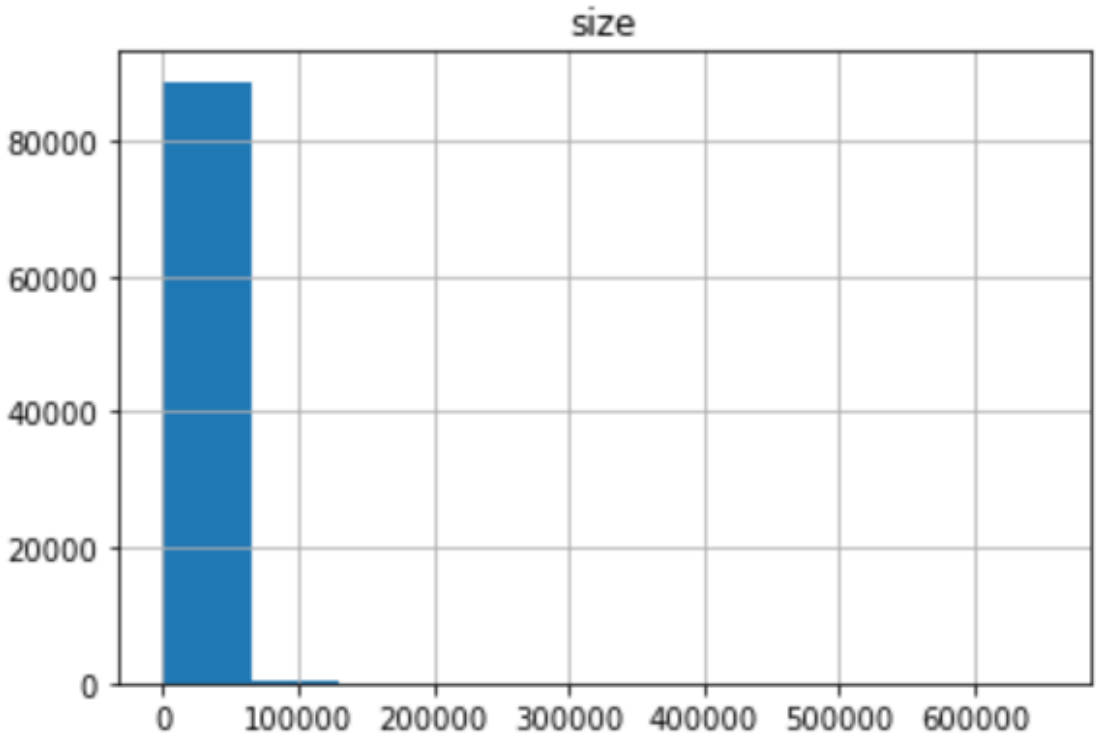
در این سوال ابتدا باید براساس client, user_agent گروه‌بندی کنیم و در یک دیتافریم اضافه بریزیم. برای استفاده از امکانات user_agent، استرینگ user_agent را parse می‌کنیم و از به سادگی ستون‌ها را پر می‌کنیم.

client	user_agent	browser_family	os_family	is_bot	is_pc
1.132.107.223	Mozilla/5.0 (Linux; Android 8.0.0; SAMSUNG SM-G965F Build/R16NW) AppleWebKit/537.36 (KHTML, like Gecko) SamsungBrowser/8.2 Chrome/63.0.3239.111 Mobile Safari/537.36	Samsung Internet	Android	False	False
1.132.108.133	Mozilla/5.0 (Linux; Android 9; Pixel 2 XL) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/71.0.3578.99 Mobile Safari/537.36	Chrome Mobile	Android	False	False
1.136.111.52	com.google.GoogleMobile/56.0.0 iPhone/12.0.1 hw/iPhone8_1	Mobile Safari UI/WKWebView	iOS	False	False
1.158.74.178	Mozilla/5.0 (iPhone; CPU iPhone OS 12_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) CriOS/71.0.3578.89 Mobile/15E148 Safari/605.1	Chrome Mobile iOS	iOS	False	False
1.159.185.202	Mozilla/5.0 (Linux; Android 4.4.2; SM-P600) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/71.0.3578.99 Safari/537.36	Chrome	Android	False	False
1.214.221.2	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/56.0.2924.87 Safari/537.36	Chrome	Mac OS X	False	True
	User-Agent:Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.2; .NET CLR 1.0.3705	IE	Windows	False	True
1.234.99.77	Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko	IE	Windows	False	True
	Mozilla/5.0 (Windows NT 10.0; Win64; x64; Trident/7.0; rv:11.0) like Gecko	IE	Windows	False	True
1.36.128.224	Mozilla/5.0 (iPhone; CPU iPhone OS 12_1_2 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/12.0 Mobile/15E148 Safari/604.1	Mobile Safari	iOS	False	False

Q5

در این سوال ابتدا دیتافریمی می‌سازیم که از sample_df تمام سطر هایی که size=0 هستند را بگیرد. تعداد هر کدام را حساب می‌کنیم و بر هم تقسیم می‌کنیم. در نهایت sample_df را پلات می‌کنیم.

```
0.28120253378188326
```



۰.۲۸ همان نسبت size=0 ها به تعداد کل است.

اندپوینت‌هایی که دارای filter هستند همگی سایز صفر دارند.

Q7

1- تعداد کلیک ها روی صفحه: کرالر می تواند تعداد بیشتری کلیک در صفحه بکند و در نتیجه ریکوئست های بیشتری در یک session مشخص بفرستد.
2- نسبت درخواست های html به تصاویر: معمولا ربات ها منتظر لود شدن تصویر نیستند. (یعنی گرافیک صفحه برای آن ها اهمیت ندارد.) در حالی که کلاینت معمولی منتظر لود شدن درخواست های تصویر می ماند.
3- تعداد درخواست های PDF/PS: این درخواست ها را اگر در یک session به دست بیاوریم، کرالر تعداد بیشتری درخواست برای PDF/PS نسبت به آدم عادی ارسال می کند.

4- درصد درخواست هایی که status code = 4xx دارند: کرالر ها احتمال بیشتری دارد که صفحات پاک شده یا outdated را برای دیدن درخواست بدهند. بنابراین اگر تعداد ارور ها نسبت به کل درخواست ها زیاد باشد، احتمالا کرالر است.

5- تعداد درخواست هایی که متد آن ها HEAD است. در صورتی که نسبت تعداد درخواست ها با متد HEAD به کل درخواست ها از حالت عادی بیشتر باشد، احتمالا کرالر است. چون کرالر ها برای اینکه حجم دیتایی که از سایت می گیرند را کاهش بدهند، بیشتر از HEAD استفاده می کنند. یوزرهای عادی از GET استفاده می کنند.

6- ریکوئست هایی که مرجع های درستی نداشته باشند: اگر نسبت کل ریکوئست هایی که مرجع دقیقی ندارند به کل ریکوئست ها بالا باشد، احتمال دارد که کرالر باشد. معمولا کرالر ها درخواست های HTTP خود را با مرجع خالی یا نامشخص ارسال می کنند.

7- در ریکوئست فایل، عبارت Robot.txt وجود داشته باشد: این یک مقدار بولینی است. اگر یک باشد یعنی فایل درخواستی عبارت robot.txt را دارد و در غیر این صورت ندارد. توسعه دهنده های یک وبسایت، برای هر آدرسی یک فایل به نام robot.txt قرار می دهند که در آن مشخص می شود که آیا ربات ها می توانند آن قسمت را visit کنند و ببینند یا خیر. معمولا کاربران عادی این درخواست را نمی زنند.

8- انحراف معیار عمق درخواست ها: عمق درخواست اینطور محاسبه میشود:
/cshome/courses/index.html این درخواست عمق 3 دارد. /cshome/calendar.html’ این درخواست عمق 2 دارد. اگر انحراف عمق ها کم باشد، ممکن است کرالر باشد.

9- تعداد درخواست های پشت سر هم تکراری زیاد باشد: تعداد درخواست های تکراری و پشت سر هم که به یک دایرکتوری ارسال می شود را نگهداری می کنیم. اگر این مقدار زیاد باشد، احتمال کرالر بودن هست. البته باید توجه داشت که مثلا اگر اول یک درخواست به /cshome/index.html داشتیم و بعد یک درخواست به cshome/courses/index.html داشتیم، پشت سر هم و تکراری حساب نمی شود.

10- انسان ها بیشتر از ربات ها از history.back, history.forward اسفاده می کنند. یعنی مثلا برای یک آدم احتمال کمتری وجود دارد که دایرکتوری های رندوم در وبسایت را باز بکند. معمولا پترن دایرکتوری هایی که باز می کند، پشت سر هم هستند.

Q8

در این سوال ابتدا بر اساس user_agent, client گروه‌بندی می‌کنیم و بعد سورت می‌کنیم.

:

client	user_agent	requests_count
66.249.66.194	Mozilla/5.0 (Linux; Android 6.0.1; Nexus 5X Build/MMB29P) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2272.96 Mobile Safari/537.36 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)	257701
66.249.66.91	Mozilla/5.0 (Linux; Android 6.0.1; Nexus 5X Build/MMB29P) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2272.96 Mobile Safari/537.36 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)	125945
	Googlebot-Image/1.0	108930
66.249.66.194	Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)	88879
66.249.66.91	Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)	79180
...	...	...
2.185.210.223	Mozilla/5.0 (Linux; Android 8.1.0; SM-J730F) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/71.0.3578.99 Mobile Safari/537.36	1
2.185.210.68	Mozilla/5.0 (Linux; Android 5.0; SM-G900H Build/LRX21T) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/43.0.2357.93 Mobile Safari/537.36	1
5.198.170.58	Dalvik/2.1.0 (Linux; U; Android 5.1.1; SM-J200H Build/LMY48B)	1
2.185.211.120	Dalvik/1.6.0 (Linux; U; Android 4.4.2; SM-G355H Build/KOT49H)	1
5.212.33.151	Dalvik/2.1.0 (Linux; U; Android 8.1.0; SM-G610F Build/M1AJQ)	1

همان‌طور که در سوال قبل هم خواندیم، اگر کلاینتی تعداد درخواست‌های زیادی را پشت سر هم در یک session ارسال بکند، احتمال این که کرالر باشد هست. مثلا در همین نمونه، کلاینت با IP ۶۶.۲۴۹.۶۶.۱۹۴، در یک session، ۲۵۷۷۰۱ درخواست فرستاده. این یعنی مشکلی در آن هست و احتمال ربات بودن آن هست.

Q9

در این سوال باید یکسری فیچر پیاده سازی کنیم که به طور کلی مشخص می‌کند که آیا آن کلاینت یک کرالر هست یا نه.

Request_count:

این در قسمت قبل پیاده سازی شده است. باید تابع count را برای ستون request دیتافریم logs_df، agg کنیم.

STD of endpoint lengths:

از تابع std در عملیات‌های aggregation که دقیقا به همین منظور است، استفاده می‌کنیم. ابتدا باید تعداد مقادیر جدا شده با / را در بیاوریم. این کار را با استفاده از تابع apply و ساخت یک ستون به نام requests_std در logs_df انجام می‌دهیم. در نهایت با تابع std روی آن agg می‌کنیم.

Percentage of 4xx status codes:

ابتدا یک ستون به نام is_error به logs_dfاضافه می‌کنیم. اگر مقدار status code آن بین ۴۰۰ تا ۵۰۰ بود، آن سطر مقدار صحیح می‌گیرد. در نهایت با استفاده از تابع mean میانگین تعدادهای مقادیر ارور در این ستون را می‌شماریم.

Percentage of HTTP HEAD requests:

دقیقا مشابه بالایی. با این تفاوت که باید یک ستون is_head بسازیم تا مقادیری که method آن‌ها head است را نگه داری کند.

Average of the size column:

به سادگی با تابع avg، روی ستون size دیتافریم logs_df، aggregate انجام می‌دهیم.

Robots.txt requests:

یک ستون به logs_df اضافه می‌کنیم تا اگر درخواستی robots.txt در آن بود، مقدار آن صحیح شود. در نهایت با count، مقادیری که robots.txt را دارند می‌شماریم.

Average of time between requests per session:

تابع time_average را روی ستون logs_df datetime اعمال می‌کنیم. این تابع اگر تعداد درخواست‌ها یک بود، مقدار صفر را برمی‌گرداند. در غیر این صورت، مقادیر آن را سورت کرده، بین آن ها اختلاف می‌گیرد و در نهایت میانگین می‌گیرد.

Percentage of requests with unassigned referrers:

ابتدا یک ستون در logs_df می‌سازیم تا مقادیری که unsassigned referrer دارند در آن مشخص شوند. شرط صحیح شدن آن این است که referrer آن یا مقدار خالی داشته باشد یا null باشد. در نهایت تابع mean را در aggregate برای آن اجرا می‌کنیم تا نسبت به کل درخواست‌ها، تعداد درخواست‌های خالی به دست بیاید.