Hotel TULIP Web Server Data Analysis

Assignment 2 - SIT742 Modern Data Science

Alex Cummaudo <`ca@deakin.edu.au`>

Jake Renzella <`jake.renzella@deakin.edu.au`>

Deakin Software and Technology Innovation Laboratory

School of Information Technology

Deakin University, Australia

May 21, 2017

Executive Summary

This document reports session usage analysis on the Hotel TULIP website by investigating the frequency patterns of how people visited the website in a typical session. Following findings of Assignment 1, we juxtaposed session browsing between the following categories: (1) internal visitors (i.e., guests) of the website; (2) external visitors; (3) visitors from Hong Kong, the United States and Australia, being the top three countries identified who visit the website most; (4) differentiating usage patterns between PCs, Smartphone devices, Tablet Devices; and (5) how bots crawled the website. We developed a visual representation of these identified patterns and highlight areas where people are most likely to navigate between on the Hotel TULIP website to find more information about the hotel in differing fashions. Analysis was performed using the FPGrowth algorithm implementing using Apache Spark on the Databricks platform. Further details on the extraction implementation are attached and an interactive version of this file can be found on Databricks.

Contents

List of Figures

List of Tables

1 Key Findings

A list of key findings in the analysis are as thus:

- Internal visitors in the website tended to most frequently browse the ”About the Hotel” page, then click into rooms and the read about the hotel’s offers, though there is strong correlation between these visitors going from dining page into the offers page,
- external visitors, however, most frequently viewed the ”Above and Beyond” page first, then visited the ”Facilities” page before reading about the hotel’s rooms, though some also went from ”Above and Beyond” directly to the offers and dining in the hotel,
- visitors from Hong Kong usually followed similar patterns to those outlined above but in the inverse order of frequency (visited offers and dining more frequently),
- American visitors usually started by viewing the hotel’s rooms, then the offers made by the hotel, and lastly viewing ”About the Hotel” itself.
- Australian visitors have a further broader visiting pattern, but most frequently viewed the ”Above and Beyond” page first, then read about the hotel’s facilities, and lastly reading about the hotel’s Rooms,
- PC visitors most followed a frequency pattern similar to that of Australian visitors,
- Smartphone visitors had narrow visiting patterns, but started browsing by viewing about the hotel’s offers, then about the facilities of the hotel and lastly about the dining experience at the hotel,
- Tablet visitors most frequently read first about the facilities of the hotel, then about the hotel’s offers and lastly about the dining experiences,
- Bots crawled the website in a completely different fashion, by most frequently crawling between the ”Our City” page, then about the ”Location and Contacts” and lastly the ”About The Hotel” page.

2 Introduction

Browsing patterns on the Hotel TULIP Weblogs were assessed in order to gain insight on how customers navigate through the website within a typical *session*. A user session is defined as a typical visit to the website, and a collation of all the different *informational resources* that were accessed. Information resources refer to web pages that contain primary content about the hotel and its facilities, rather than multimedia and technical-related resources.

In order to assess how *different* customers do so, we extract different information based on the web log data format as prescribed in Appendix B. We contrast those users who make requests:

- internally, such as guests using the internet within the hotel's network,
- externally, such as prospective guests browsing the website for a potential stay in the hotel,
- from users within the top three countries that visit the website (refer to Assignment 1), and
- between users on PCs, Smartphones, Tablets and Bots.

Each of these criteria were analysed against matching sessions that satisfy such criteria. Data mined using the Frequency Pattern was done so using the FPGrowth Algorithm in Apache Spark.

3 Dataset

Hotel TULIP's web server runs Microsoft Internet Information Services (IIS) Server 7.5, and the attributes of this dataset as well as the relevant data dictionary can be found in Appendix B.3.

4 Method

4.1 Assumptions

4.1.1 User Sessions

A *web session* is defined as a single session of a particular user at a given moment of time. It consists of what pages were visited within that period of time, as determined by the requests made to the server in this period.

To extrapolate meaningful data, the definition of *what* a user session was needed to be determined. This is because we need to find patterns made in one particular user session; when a user visits the website in one sitting, what pages do they visit and in what order?

To do this, we assume that a session is made up of the following factors within a series of web requests:

1. The client's IP address must be the same,
2. The client's user agent string must be the same,
3. The timestamp of the session is grouped in the same day, and
4. The timestamp of the session is within the same hour.

We group every request using these four key factors, using a hash as a delimiter, into the field `session_identifier`. Using this identifier, we can group up a series of requests into one particular session. For example, a sample request is made to the server:

- The `c_ip` field is `1.2.3.4`,
- The `user_agent` field is `Safari`,
- The `timestamp` field is `Tue Feb 29 03:18:00 GMT 2017`

Therefore the `session_identifier` would be:

```
1.2.3.4#Safari#29-02-2017#03
```

Hence, multiple requests made within the 3rd hour of the 29th of February 2017 by the IP address `1.2.3.4` with the user agent `Safari` will be gathered together. Obviously this example is trivial and is more detailed in practice, but as an explanation it helps to be simple.

Advantages of this approach is that there can be multiple requests made from a single IP address that *are not* from the same user agent. For example, internal requests made by users within the hotel would use different computers, and therefore different `user_agent` strings would be made, making the session identifier unique.

Disadvantages of this approach are that there could be two users on an internal network (i.e., same `client_ip`) using the *exact* same user agent within the same exact hour who are technically not the same person. Additionally, by grouping sessions by the hour, any session that was near the hour may be split into two sessions. These are some limitations of our approach.

4.1.2 Selection of Resources

Not all requests were included in the analysis, as some requests were simply resource requests. We want to focus purely on *informational resources*, and therefore need to filter the number of requests to those that are relevant.

To do this, we consider the following:

1. The request resource is non-multimedia or functional, but informational. That is, requests whose resources are not JavaScript, Cascading Style Sheets, Images and the like, but rather ASPX and ASHX files (ASP.NET and Generic Web Handler pages that contain information about the hotel).
2. The request resource is not under the directories `media`, `layouts`, or `sitecore`, as these directories do not contain informational resources.
3. The request returns a client status that is only 200 (Success), or is not a placeholder page for a status page, e.g., a `404.aspx` page would not be appropriate to consider.

Additionally, we make all request resources consistent for comparison by making them all lower case, which is made visible in the diagrams shown in Section 5.

4.2 Extraction Process

4.2.1 Data Mapping

Following the process made in Assignment 1, all publicly known client IP addresses were extracted from the MaxMind GeoIP2[1] dataset to determine request locations. This therefore allowed us to retrieve sessions from the top three countries, that being: (1) Hong Kong; (2) the USA; and (3) Australia. Determination that these were the top three countries can be found in Assignment 1.

Additionally, user agent strings were parsed to analyse device and browser statistics using the Python `user-agents` library[2] These user agent strings allowed determination of whether or not

[1]See `http://dev.maxmind.com/geoip/geoip2/`.

[2]See `https://pypi.python.org/pypi/user-agents`.

users were using: (1) PCs; (2) Smartphones; (3) Tablets; or (4) if they were actually Bots crawling the website.

4.2.2 IP Address Source Regular Expression

To differentiate between private site visitors and external visitors, a regular expression was used to filter the private and public IP address ranges. The regular expression is shown below:

```
WHERE l.c_ip REGEXP
          '(^127\.)|(^10\.) |
          (^172\.1[6-9]\.) |
          (^172\.2[0-9]\.) |
          (^172\.3[0-1]\.) |
          (^192\.168\.)'
```

Negating this WHERE clause of the regular expression will select only public IP addresses.

4.3 Data Mining

Data analysis was gathered using Frequent Pattern mining of user sessions. The Apache Spark implementation (Wendell, 2017) uses FPGrowth, an algorithm described by Han et al. (2000) that calculates item frequencies to identify frequent items in a given set of data. The provided data in our case was the extracted data from our SQL queries which were developed inside the attached Python Notebook.

The Apache Spark implementation requires the data parameter minSupport, which is the the threshold for an itemset to be identified as frequent. The default support level chosen was 0.01 (i.e., if a frequent item appears 10 times out of 1000 transactions, then its support is 0.01).

However, this default support value was massaged and relaxed in some instances where few patterns could be identified. These cases are: Hong Kong, Smartphone and Tablet requests, where the support level was relaxed by 50%; and Australian requests, where the support level was relaxed by 75%.

Additionally, we wish to retrieve sessions that contain *three or more* requests in the session. This is to ensure there are multiple hits in the session, and to retrieve more meaningful patterns.

5 Results

In our results, we have visualised the frequency patterns of users via the use of directional network graphs. In these graphs, we are able to visualise the frequency patterns of how people made requests to the website given the assumptions and extraction methods made in Section 4.

Within each graph, a *sequence* is identified as a series of multiple clicks (edges) between pages (nodes). Each sequence is coloured using the same edge colour. This sequence is also identified using a number, which is drawn on the edge label. The frequency of this pattern for the particular sequence identified is given after the forward slash on the label.

For example, a sample directional network graph, a subset of the PC requests, is given in Figure 1. This data is also presented in tabular format as Table 1.

Here we can interpret that the graph has four key sequences, as differentiated by the sequence numbers. Sequence numbers are ordered by decreasing frequency; the higher the sequence number the increased likelihood of the pattern occurring. In this graph, we see that users of PCs are most likely to move between pages in the following order:

1. from the 'Above and Beyond' page to the 'Dining' page (frequency of 293),
2. from the 'Facilities' page to the 'Dining' page (Frequency of 288),
3. from the 'Above and Beyond' page to the 'Offers' page (frequency of 286), and, with equal frequency,
4. from the 'Facilities' page to the 'Offers' page.

Table 1: Sample Frequency Graph

Sequence	**From**		**To**	**Frequency**
2	above and beyond	$\rightarrow$	rooms	293
2	rooms	$\rightarrow$	dining	293
3	facilities	$\rightarrow$	rooms	288
3	rooms	$\rightarrow$	dining	288
4	above and beyond	$\rightarrow$	rooms	286
4	rooms	$\rightarrow$	offers	286
5	facilities	$\rightarrow$	rooms	286
5	rooms	$\rightarrow$	offers	286

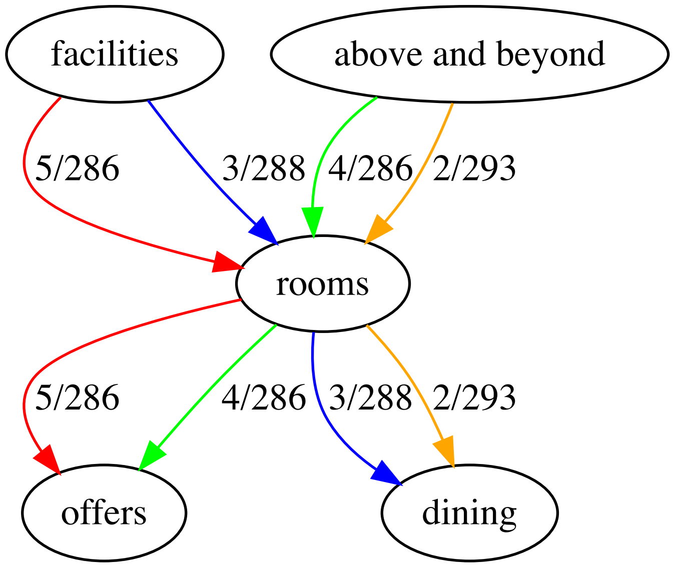


Figure 1: Sample frequency patterns identified in a subset of PC requests. Refer to Table 1 for frequency pattern interactions.

5.1 IP Request Sources

5.1.1 Internal Site Visitors

The internal IP request sources are visitors to the TULIP hotel who are accessing the site from the internal TULIP Hotel network (they are using computers or wifi of the hotel).

Refer to Figure 2.

The most common visitor pattern of internal visitors is ”about the hotel ” /rightarrow ”rooms”.

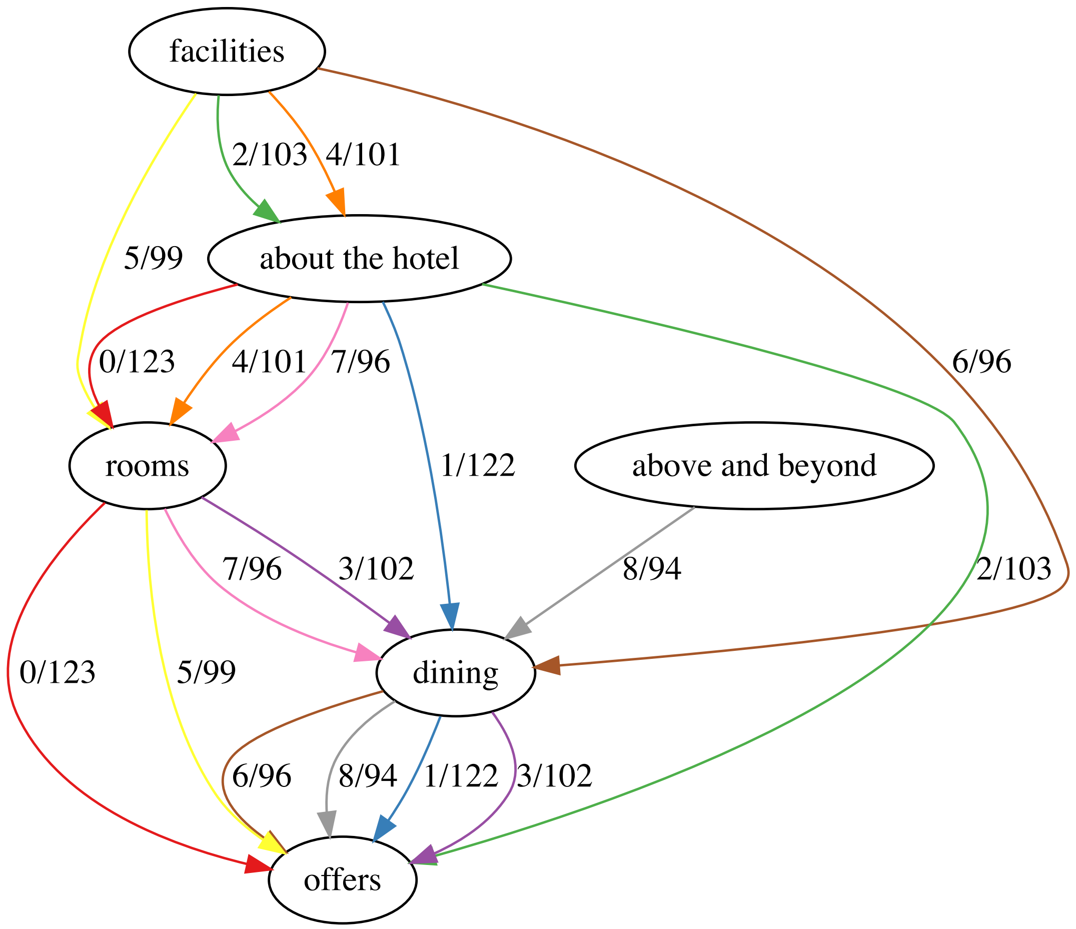


Figure 2: Directional network graph visualising frequency patterns of internal visitors made on an internal IP range. Refer to Table 2 for frequency pattern interactions.

5.1.2 External Site Visitors

The external IP request sources are visitors to the TULIP hotel who are accessing the site outside of the TULIP Hotel network (most visitors).

Sequences in order of sequence frequency:

1. Above and beyond /*rightarrow* facilities /*rightarrow* rooms
2. Above and beyond /*rightarrow* offers /*rightarrow* dining
3. facilties /*rightarrow* rooms /*rightarrow* offers
4. facilties /*rightarrow* offers /*rightarrow* dining
5. facilties /*rightarrow* rooms /*rightarrow* dining
6. above and beyond /*rightarrow* facilties /*rightarrow* dining

The most common visitor s of external visitors is: ”above and beyond” /rightarrow ”facilities” /rightarrow ”rooms” - indicating a common interest of these users. This information could indicate to TULIP Hotel that room information is highly sought after, and could be made more prominant on the website, possibly under the facilities page which will help users find the information they are looking for.

Patterns also highly suggest that many users navigate to the offers page, most commonly from ”above and beyond” followed by ”rooms” and finally ”facilities”.

Refer to Figure 3.

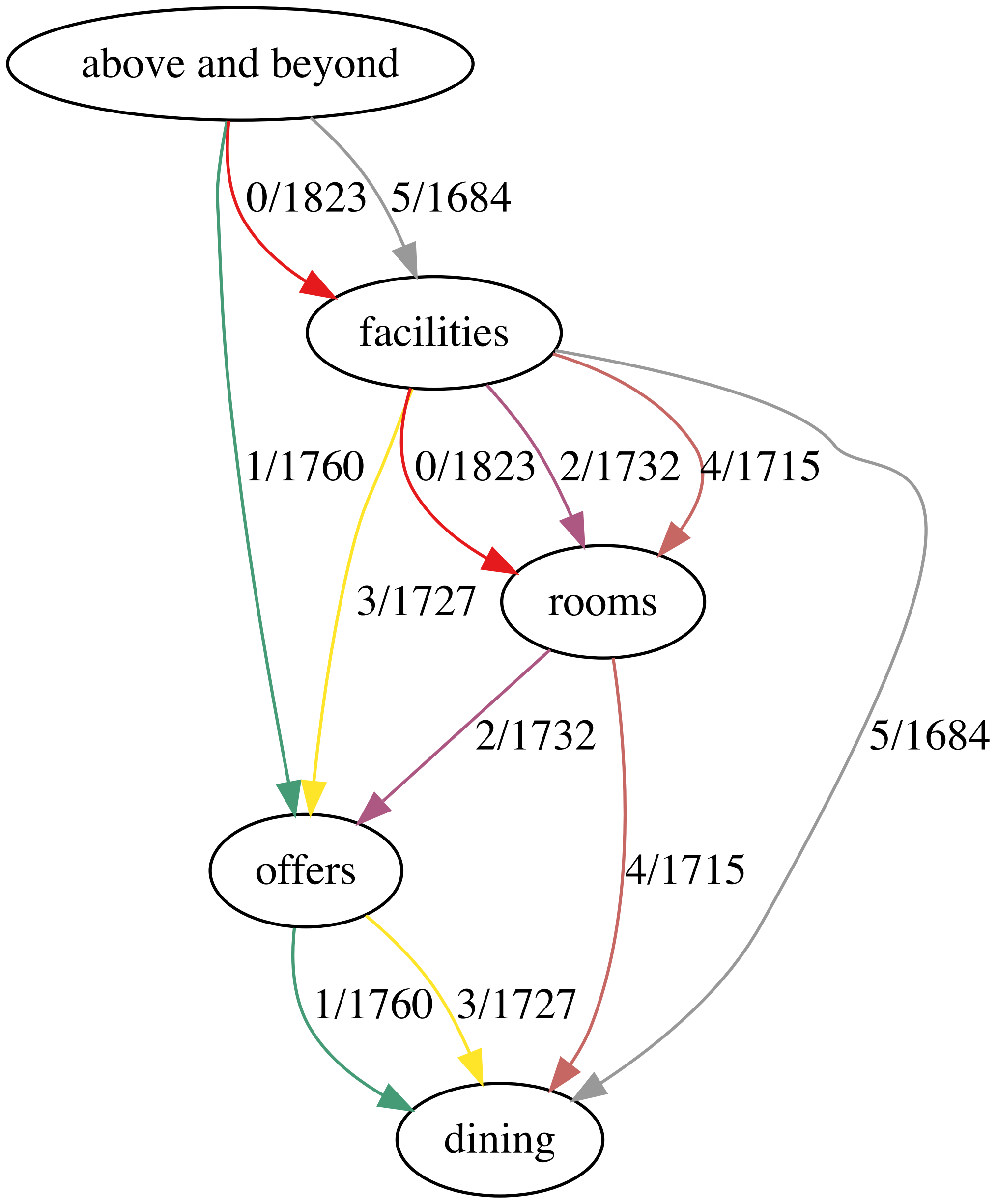


Figure 3: Directional network graph visualising frequency patterns of external visitors made on a non-internal IP range. Refer to Table 3 for frequency pattern interactions.

5.2 Top Three Countries

5.2.1 Hong Kong Visitors

Refer to Figure 4.

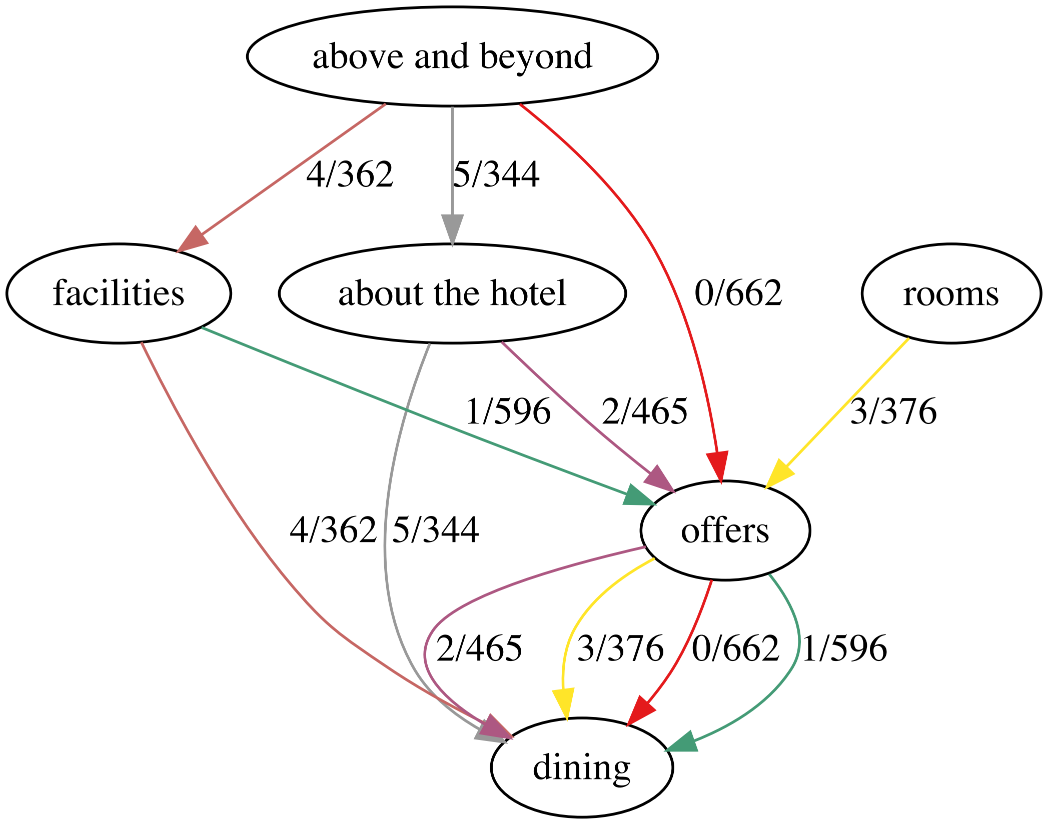


Figure 4: Directional network graph visualising frequency patterns of visitors from Hong Kong. Refer to Table 4 for frequency pattern interactions.

5.2.2 USA Visitors

Refer to Figure 5.

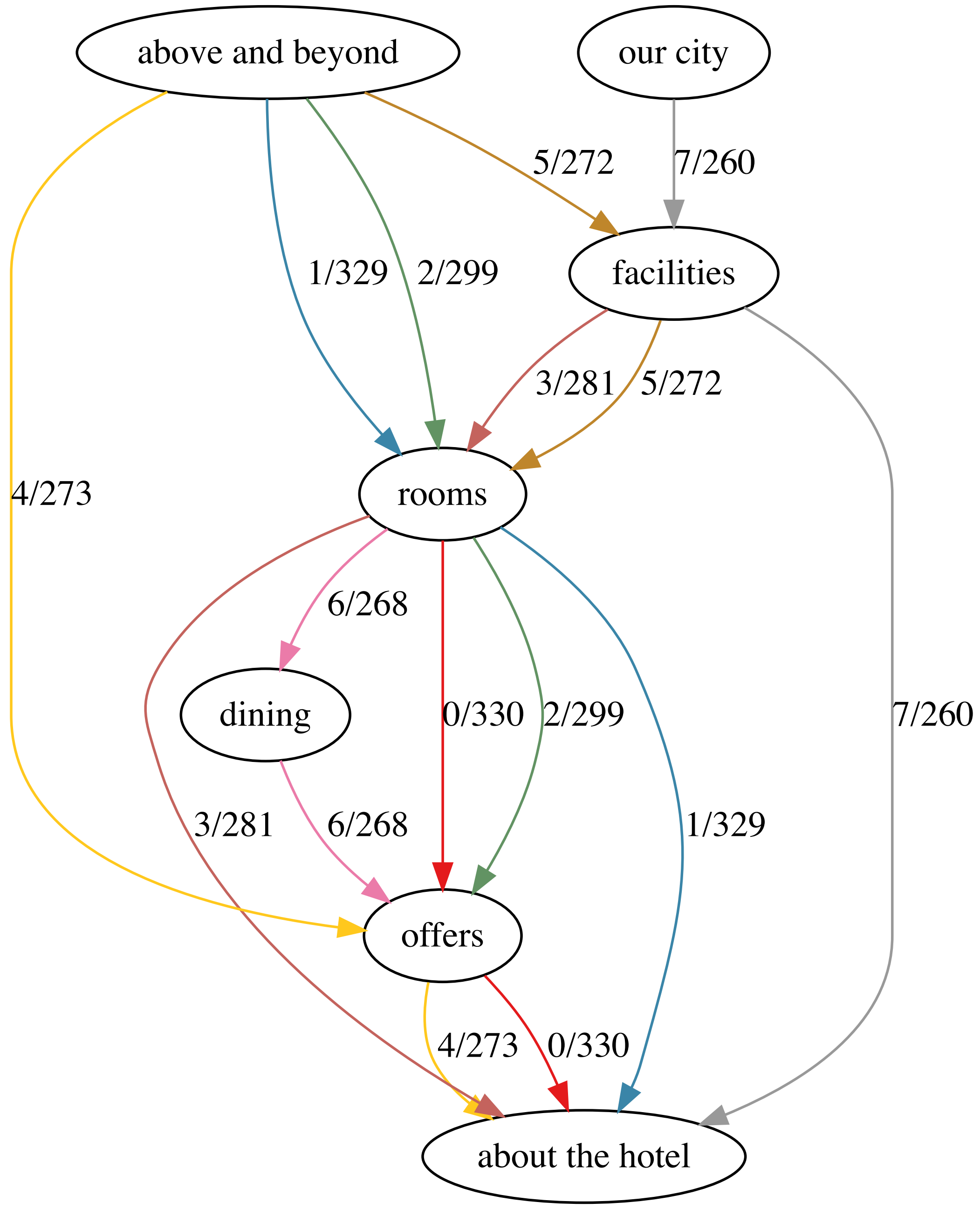


Figure 5: Directional network graph visualising frequency patterns of visitors from the United States. Refer to Table 5 for frequency pattern interactions.

5.2.3 Australian Visitors

Refer to Figure 6.

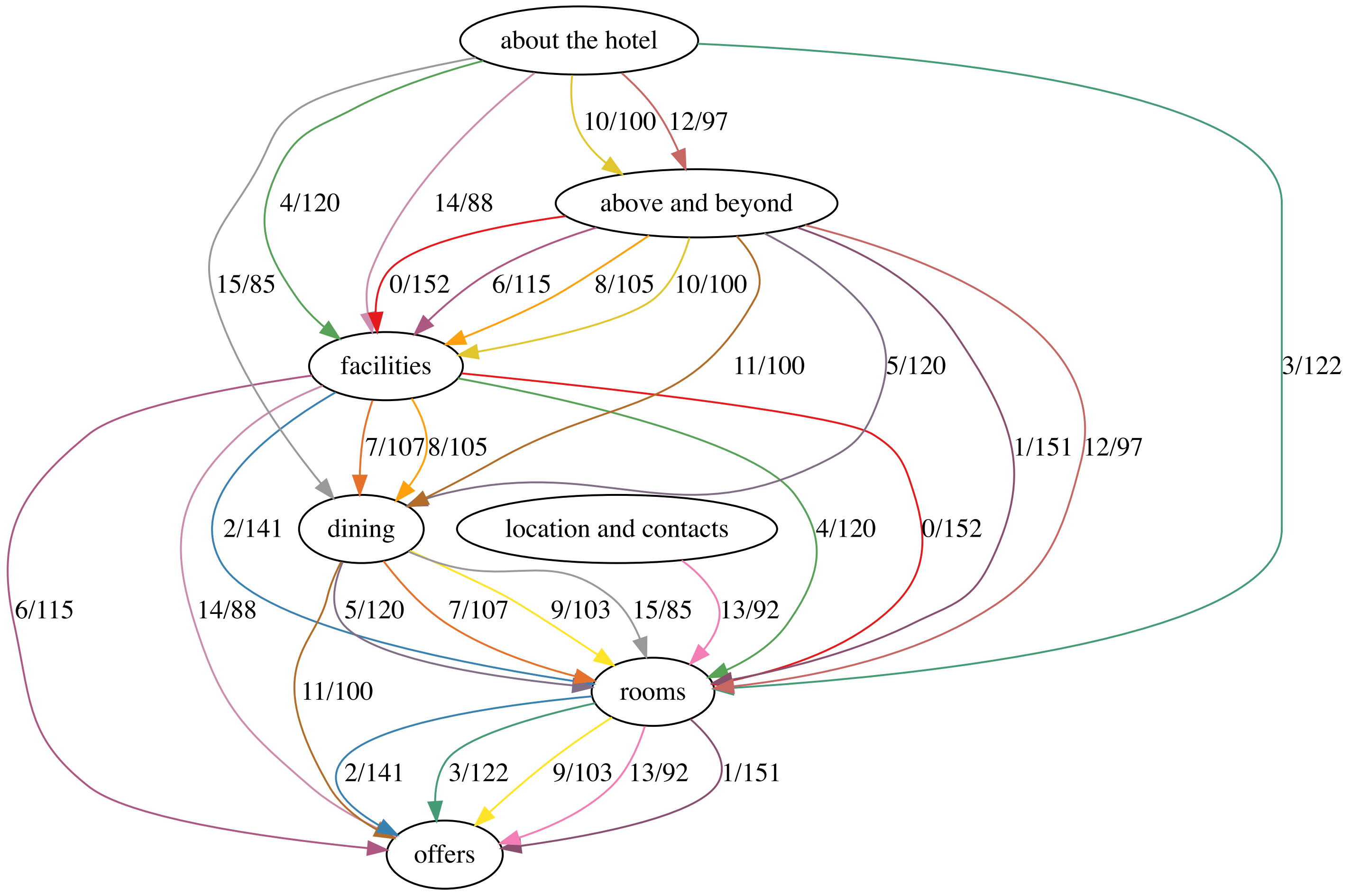


Figure 6: Directional network graph visualising frequency patterns of visitors from Australia. Refer to Table 6 for frequency pattern interactions.

5.3 Platform Categories

5.3.1 PC Visitors

Refer to Figure 7.

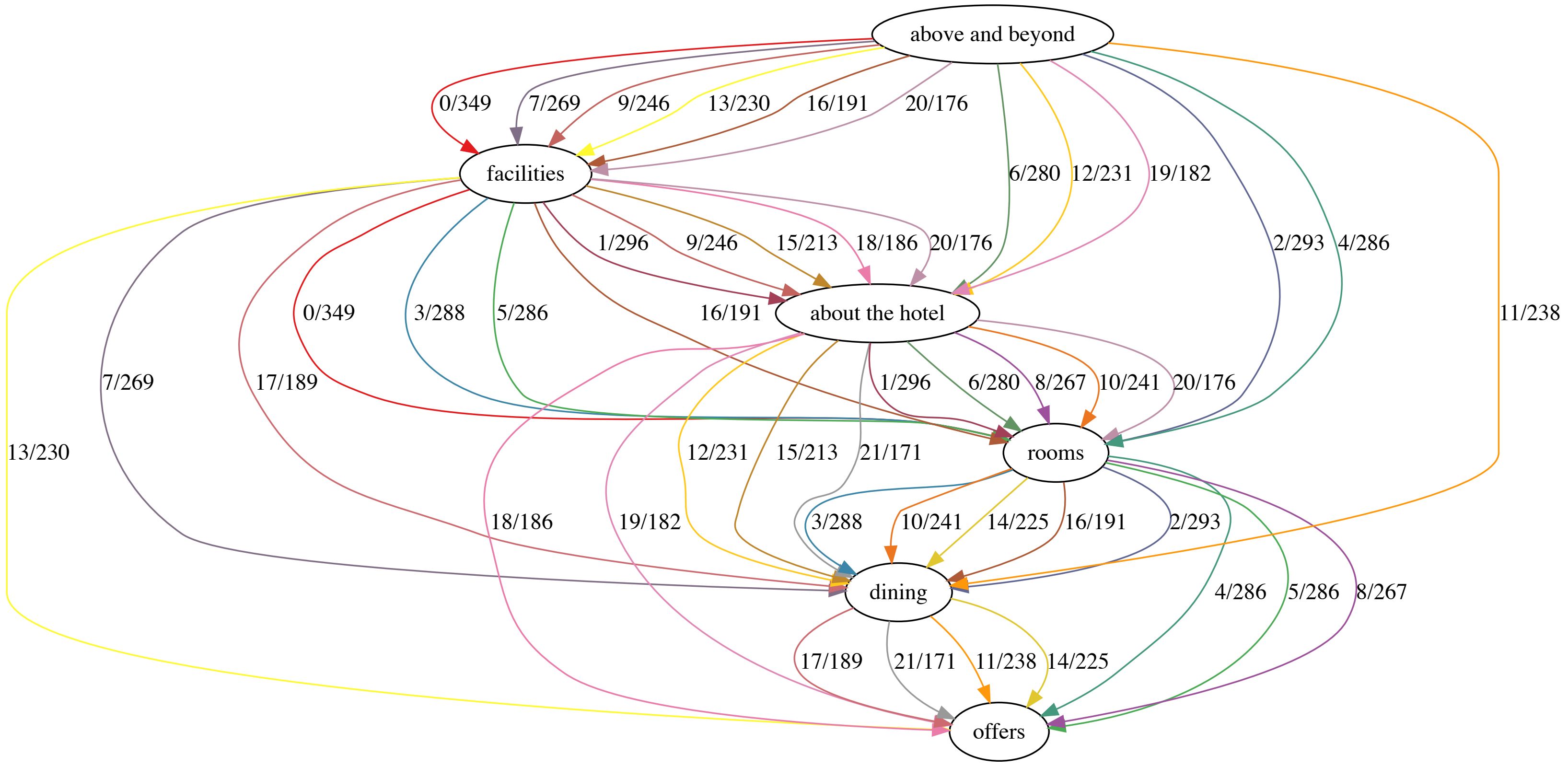


Figure 7: Directional network graph visualising frequency patterns of requests made by PCs. Refer to Table 7 for frequency pattern interactions.

5.3.2 Smartphone Visitors

Refer to Figure 8.

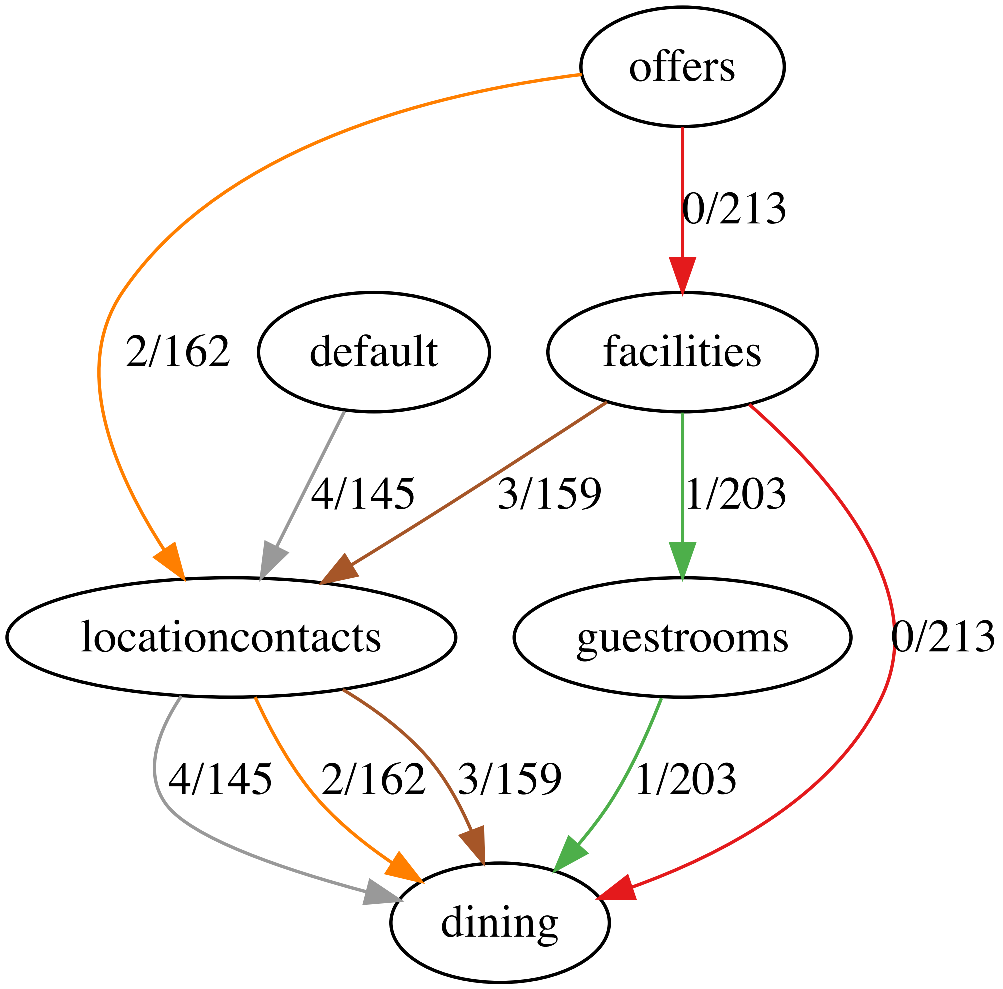


Figure 8: Directional network graph visualising frequency patterns of requests made on smartphones. Refer to Table 9 for frequency pattern interactions.

5.3.3 Tablet Visitors

Refer to Figure 9.

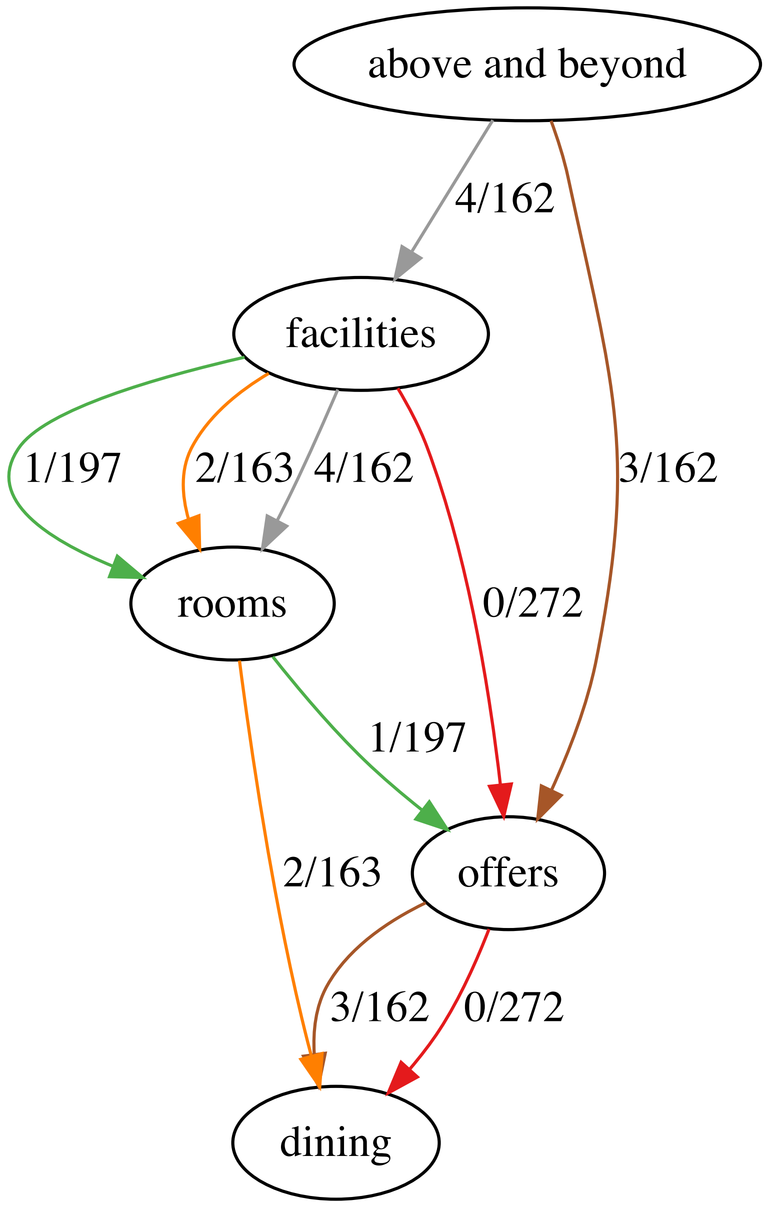


Figure 9: Directional network graph visualising frequency patterns of requests made on tablets. Refer to Table 8 for frequency pattern interactions.

5.3.4 Bots Visitors

Refer to Figure 10.

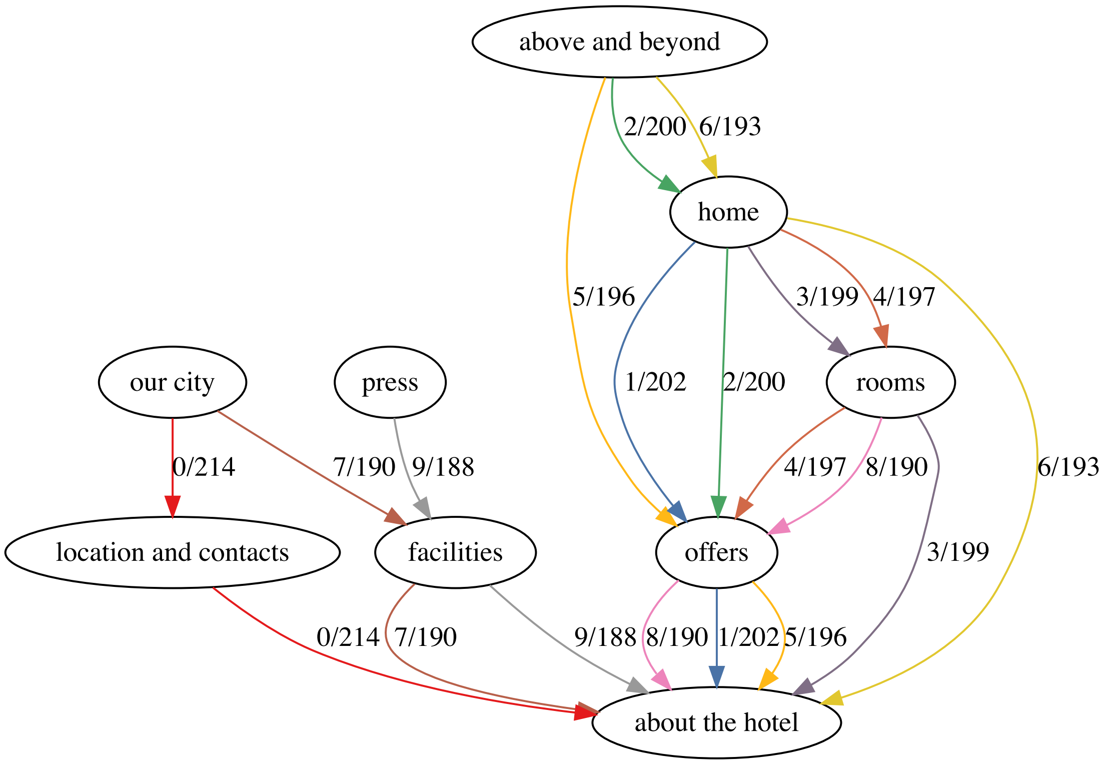


Figure 10: Directional network graph visualising frequency patterns of requests made by bots. Refer to Table 10 for frequency pattern interactions.

References

Fielding, R., J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee (1999). RFC 2616, Hypertext Transfer Protocol – HTTP/1.1. Retrieved 11 May 2017, <`http://www.rfc.net/rfc2616.html`>.

Hallam-Baker, P. and B. Behlendorf (1998). Extended Log File Format. Retrieved 11 May 2017, <`https://www.w3.org/TR/WD-logfile-960221.html`>.

Han, J., J. Pei, and Y. Yin (2000, May). Mining frequent patterns without candidate generation. *SIGMOD Rec. 29*(2), 1–12.

Microsoft Corporation (2003). W3C Extended Log File Format (IIS 6.0). Retrieved 11 May 2017, <`https://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/676400bc-8969-4aa7-851a-9319490a9bbb.mspx?mfr=true`>.

Microsoft Corporation (2017). Description of Microsoft Internet Information Services (IIS) 5.0 and 6.0 status codes. Retrieved 11 May 2017, <`https://support.microsoft.com/en-au/help/318380/description-of-microsoft-internet-information-services-iis-5.0-and-6.0-status-codes`>.

Wendell, P. (2017). Frequent Pattern Mining - RDD-based API. Retrieved 11 May 2017, <`https://people.apache.org/~pwendell/spark-nightly/spark-master-docs/latest/mllib-frequent-pattern-mining.html`>.

Wolf, M. and C. Wicksteed. Date and Time Formats. https://www.w3.org/TR/NOTE-datetime.

A Additional Tables

Below are tables of frequency pattern results for each section identified in Section 5.

Table 2: Internal Request Frequency Patterns

Sequence	From		To	Frequency
0	about the hotel	→	rooms	123
0	rooms	→	offers	123
1	about the hotel	→	dining	122
1	dining	→	offers	122
2	facilities	→	about the hotel	103
2	about the hotel	→	offers	103
3	rooms	→	dining	102
3	dining	→	offers	102
4	facilities	→	about the hotel	101
4	about the hotel	→	rooms	101
5	facilities	→	rooms	99
5	rooms	→	offers	99
6	facilities	→	dining	96
6	dining	→	offers	96
7	about the hotel	→	rooms	96
7	rooms	→	dining	96
8	above and beyond	→	dining	94
8	dining	→	offers	94

Table 3: External Request Frequency Patterns

Sequence	From		To	Frequency
0	above and beyond	→	facilities	1823

Continued on next page...

Table 3 (*continued from Page 32*): External Request Frequency Patterns

Sequence	**From**		**To**	**Frequency**
0	facilities	→	rooms	1823
1	above and beyond	→	offers	1760
1	offers	→	dining	1760
2	facilities	→	rooms	1732
2	rooms	→	offers	1732
3	facilities	→	offers	1727
3	offers	→	dining	1727
4	facilities	→	rooms	1715
4	rooms	→	dining	1715
5	above and beyond	→	facilities	1684
5	facilities	→	dining	1684

Table 4: Hong Kong Frequency Patterns

Sequence	**From**		**To**	**Frequency**
0	above and beyond	→	offers	662
0	offers	→	dining	662
1	facilities	→	offers	596
1	offers	→	dining	596
2	about the hotel	→	offers	465
2	offers	→	dining	465
3	rooms	→	offers	376
3	offers	→	dining	376
4	above and beyond	→	facilities	362
4	facilities	→	dining	362

Continued on next page...

Table 4 (*continued from Page 33*): Hong Kong Frequency Patterns

Sequence	**From**		**To**	**Frequency**
5	above and beyond	$\rightarrow$	about the hotel	344
5	about the hotel	$\rightarrow$	dining	344

Table 5: USA Request Frequency Patterns

Sequence	**From**		**To**	**Frequency**
0	rooms	$\rightarrow$	offers	330
0	offers	$\rightarrow$	about the hotel	330
1	above and beyond	$\rightarrow$	rooms	329
1	rooms	$\rightarrow$	about the hotel	329
2	above and beyond	$\rightarrow$	rooms	299
2	rooms	$\rightarrow$	offers	299
3	facilities	$\rightarrow$	rooms	281
3	rooms	$\rightarrow$	about the hotel	281
4	above and beyond	$\rightarrow$	offers	273
4	offers	$\rightarrow$	about the hotel	273
5	above and beyond	$\rightarrow$	facilities	272
5	facilities	$\rightarrow$	rooms	272
6	rooms	$\rightarrow$	dining	268
6	dining	$\rightarrow$	offers	268
7	our city	$\rightarrow$	facilities	260
7	facilities	$\rightarrow$	about the hotel	260

Table 6: Australian Request Frequency Patterns

Sequence	From		To	Frequency
0	above and beyond	→	facilities	152
0	facilities	→	rooms	152
1	above and beyond	→	rooms	151
1	rooms	→	offers	151
2	facilities	→	rooms	141
2	rooms	→	offers	141
3	about the hotel	→	rooms	122
3	rooms	→	offers	122
4	about the hotel	→	facilities	120
4	facilities	→	rooms	120
5	above and beyond	→	dining	120
5	dining	→	rooms	120
6	above and beyond	→	facilities	115
6	facilities	→	offers	115
7	facilities	→	dining	107
7	dining	→	rooms	107
8	above and beyond	→	facilities	105
8	facilities	→	dining	105
9	dining	→	rooms	103
9	rooms	→	offers	103
10	about the hotel	→	above and beyond	100
10	above and beyond	→	facilities	100
11	above and beyond	→	dining	100
11	dining	→	offers	100
12	about the hotel	→	above and beyond	97
12	above and beyond	→	rooms	97

Continued on next page...

Table 6 (*continued from Page 35*): Australian Request Frequency Patterns

Sequence	From		To	Frequency
13	location and contacts	$\rightarrow$	rooms	92
13	rooms	$\rightarrow$	offers	92
14	about the hotel	$\rightarrow$	facilities	88
14	facilities	$\rightarrow$	offers	88
15	about the hotel	$\rightarrow$	dining	85
15	dining	$\rightarrow$	rooms	85

Table 7: PC Request Frequency Patterns

Sequence	From		To	Frequency
0	above and beyond	$\rightarrow$	facilities	349
0	facilities	$\rightarrow$	rooms	349
1	facilities	$\rightarrow$	about the hotel	296
1	about the hotel	$\rightarrow$	rooms	296
2	above and beyond	$\rightarrow$	rooms	293
2	rooms	$\rightarrow$	dining	293
3	facilities	$\rightarrow$	rooms	288
3	rooms	$\rightarrow$	dining	288
4	above and beyond	$\rightarrow$	rooms	286
4	rooms	$\rightarrow$	offers	286
5	facilities	$\rightarrow$	rooms	286
5	rooms	$\rightarrow$	offers	286
6	above and beyond	$\rightarrow$	about the hotel	280
6	about the hotel	$\rightarrow$	rooms	280
7	above and beyond	$\rightarrow$	facilities	269

Continued on next page...

Table 7 (*continued from Page 36*): PC Request Frequency Patterns

Sequence	**From**		**To**	**Frequency**
7	facilities	→	dining	269
8	about the hotel	→	rooms	267
8	rooms	→	offers	267
9	above and beyond	→	facilities	246
9	facilities	→	about the hotel	246
10	about the hotel	→	rooms	241
10	rooms	→	dining	241
11	above and beyond	→	dining	238
11	dining	→	offers	238
12	above and beyond	→	about the hotel	231
12	about the hotel	→	dining	231
13	above and beyond	→	facilities	230
13	facilities	→	offers	230
14	rooms	→	dining	225
14	dining	→	offers	225
15	facilities	→	about the hotel	213
15	about the hotel	→	dining	213
16	above and beyond	→	facilities	191
16	facilities	→	rooms	191
16	rooms	→	dining	191
17	facilities	→	dining	189
17	dining	→	offers	189
18	facilities	→	about the hotel	186
18	about the hotel	→	offers	186
19	above and beyond	→	about the hotel	182

Continued on next page...

Table 7 (*continued from Page 36*): PC Request Frequency Patterns

Sequence	**From**		**To**	**Frequency**
19	about the hotel	→	offers	182
20	above and beyond	→	facilities	176
20	facilities	→	about the hotel	176
20	about the hotel	→	rooms	176
21	about the hotel	→	dining	171
21	dining	→	offers	171

Table 8: Tablet Request Frequency Patterns

Sequence	**From**		**To**	**Frequency**
0	facilities	→	offers	272
0	offers	→	dining	272
1	facilities	→	rooms	197
1	rooms	→	offers	197
2	facilities	→	rooms	163
2	rooms	→	dining	163
3	above and beyond	→	offers	162
3	offers	→	dining	162
4	above and beyond	→	facilities	162
4	facilities	→	rooms	162

Table 9: Smartphone Request Frequency Patterns

Sequence	From		To	Frequency
0	offers	→	facilities	213
0	facilities	→	dining	213
1	facilities	→	guestrooms	203
1	guestrooms	→	dining	203
2	offers	→	locationcontacts	162
2	locationcontacts	→	dining	162
3	facilities	→	locationcontacts	159
3	locationcontacts	→	dining	159
4	default	→	locationcontacts	145
4	locationcontacts	→	dining	145

Table 10: Bots Request Frequency Patterns

Sequence	From		To	Frequency
0	our city	→	location and contacts	214
0	location and contacts	→	about the hotel	214
1	home	→	offers	202
1	offers	→	about the hotel	202
2	above and beyond	→	home	200
2	home	→	offers	200
3	home	→	rooms	199
3	rooms	→	about the hotel	199
4	home	→	rooms	197
4	rooms	→	offers	197
5	above and beyond	→	offers	196
5	offers	→	about the hotel	196

Continued on next page...

Table 10 (*continued from Page 39*): Bots Request Frequency Patterns

Sequence	**From**		**To**	**Frequency**
6	above and beyond	$\rightarrow$	home	193
6	home	$\rightarrow$	about the hotel	193
7	our city	$\rightarrow$	facilities	190
7	facilities	$\rightarrow$	about the hotel	190
8	rooms	$\rightarrow$	offers	190
8	offers	$\rightarrow$	about the hotel	190
9	press	$\rightarrow$	facilities	188
9	facilities	$\rightarrow$	about the hotel	188

B Data Dictionary

B.1 Dataset Description

Table 11: Dataset Description

Key	Entry
Name	Hotel TULIP Web Log Dataset
Size	17.06 GB (954.7 MB compressed)
Release Date	30/4/17
Attributes	14
No. Records	73,368,256
Provider	Dr Beer Guts, CIO, Hotel TULIP (Information Technology Division)[3]
Privacy	Confidential[4]

B.2 Contact Information

Table 12: Dataset Contact Information

Key	Entry
Prepared by	Team-SIT742
Point of Contact	Alex Cummaudo <`ca@deakin.edu.au`>
	Jake Renzella <`jake.renzella@deakin.edu.au`>
Team Members	Alex Cummaudo <`ca@deakin.edu.au`>
	Jake Renzella <`jake.renzella@deakin.edu.au`>

[3]Download URL: `https://d2l.deakin.edu.au/d2l/le/content/520519/topics/files/download/3482057/DirectFileTopicDownload`, `https://d2l.deakin.edu.au/d2l/le/content/520519/viewContent/3482057/View?ou=520519`.

[4]Exclusively available for educational purposes only for the Deakin University unit SIT742. Redistribution is prohibited.

B.3 Data Dictionary

Table 13: Data Dictionary

Attribute Name	Data Type	Data Subtype	Description	Examples	Notes
date	MC	DATE - Date	Date: Date when request occurred	2014-08-01	UTC time zone; ISO 8601 date format (YYYY-MM-DD)
time	MC	DATE - Time	Time: Time when request occurred	09:51:23	UTC time zone; ISO 8601 24-hr time format (hh:mm:ss)
s-ip	CN	ADDR - Address - IP Address	Server IP Address: IP of the server generating the log responding to the request	10.130.0.12	N/A
cs-uri-stem	CN	URL - Uniform Resource Identifier (URI)	URI Stem: Stem portion of the full URI made in the client to server request	/sitecore	N/A
cs-uri-query	CN	URL - Uniform Resource Identifier (URI)	URI Query: Query portion of the full URI made in the client to server request	cmd=GetTreeview	May be empty; Query string is specifically matched as URI. See Hallam-Baker & Behlendorf (1998).

Continued on next page...

Table 13 (*continued from Page 42*): Data Dictionary

Attribute Name	**Data Type**	**Data Subtype**	**Description**	**Examples**	**Notes**
s-port	CN	ADDR - Address - Port Number	Server Port: Port for the server which request is made	80	N/A
cs-username	CN	STR - Free String	User Name: Name of authenticated user accessing the server	-	Anonymous users are represented with a hyphen.
c-ip	CN	ADDR - Address - IP Address	Client IP Address: IP of the client making the request	10.120.7.23	N/A
sc-substatus	CN	ID - Identification - Microsoft IIS Named HTTP Response Sub status Code	Protocol Sub status: The sub status error code	0	Refer to Microsoft (2017) for specific codes. A zero indicates no sub status error code.
sc-win32-status	CN	ID - Identification - Microsoft IIS Named Windows Code	Win32 Status: The Windows status code	0	Generally not applicable to non-Windows devices; defaults to zero.

Continued on next page...

Table 13 (*continued from Page 42*): Data Dictionary

Attribute Name	**Data Type**	**Data Subtype**	**Description**	**Examples**	**Notes**
time-taken	MC	DATE - Time - From zero	Duration: The time taken for the request to complete.	39	Measured in milliseconds.

C Extrapolation Results

Attached on the following pages are the results from Databricks. You may also interact with this online on Databricks.

SIT742 Assignment 2

Alex Cummaudo <ca@deakin.edu.au (mailto:ca@deakin.edu.au)>, Jake Renzella <jake.renzella@deakin.edu.au (mailto:jake.renzella@deakin.edu.au)>
Student IDs 217092024, 217108883 (CloudDeakin Group 35)
Deakin Software and Technology Innovation Laboratory (DSTIL)
School of Information Technology
Deakin University, Australia

1. Prerequisites

Follow the installation instructions here (https://gist.github.com/ololobus/4c221a0891775eaa86b0) to install the following:

- Apache Spark 2.1.0
- Python 2.7
- Java 1.8

In addition, the following third-party python packages are installed:

- GeoIP2 (http://geoip2.readthedocs.io/en/latest/) to extrapolate IP information
- UserAgents (https://pypi.python.org/pypi/user-agents) to extrapolate User Agent information
- Networkx (https://networkx.readthedocs.io) to visualise the findings

These can be installed using `pip` :

```
$ pip install geoip2 pyyaml ua-parser user-agents networkx
```

Alterinatively, attach using the Databricks library importer.

2. Getting Started

2.1. Package Imports

Begin by importing all necessary packages.

```
import networkx as nx
import matplotlib.colors as colors
import matplotlib.pyplot as plt
import numpy as np
from datetime import datetime
from user_agents import parse as ua_parse
from pyspark.mllib.fpm import FPGrowth
```

2.2 Loading data from S3 bucket

To begin with, we need to load the data from an S3 bucket, `sit742-htweblog-gz` . I unzipped the data and compressed it using strong gzip compression from the given zip file, as per:

```
$ unzip /path/to/HTWebLog.zip
$ gzip -r /path/to/HTWebLog -9
```

I referred to this guide (https://docs.databricks.com/user-guide/dbfs-databricks-file-system.html#mounting-an-s3-bucket) for assistance with mounting Databricks into the S3 bucket. I refer to the constant `PATH_TO_S3_MOUNT` to refer to the S3 mount in Databricks.

```python
# Define constant to path of log data
PATH_TO_S3_MOUNT = '/mnt/htweblog'

# Check if we have already mounted our S3 Bucket
htweblog_s3_mounted = len(filter(lambda mount: mount.mountPoint == PATH_TO_S3_MOUNT, dbutils.fs.mounts())) == 1

if not htweblog_s3_mounted:
  # Setup AWS configuration
  ACCESS_KEY = "AKIAJDP5QWKSBKP74XUA"
  SECRET_KEY = "9c1/vI3MNniajoK7dH7ko+24Ipr47Q4S4Q5ruO9z".replace("/", "%2F")
  AWS_BUCKET_NAME = "sit742-htweblog-gz"

  # Mount S3 bucket
  dbutils.fs.mount("s3n://%s:%s@%s/" % (ACCESS_KEY, SECRET_KEY, AWS_BUCKET_NAME), PATH_TO_S3_MOUNT)

# Show mounted files
display(dbutils.fs.ls(PATH_TO_S3_MOUNT))
```

path	name
dbfs:/mnt/htweblog/geolite-db/	geolite-db/
dbfs:/mnt/htweblog/internal_requests.dot/	internal_requests.dot/
dbfs:/mnt/htweblog/sample-set/	sample-set/
dbfs:/mnt/htweblog/u_ex140801.log.gz	u_ex140801.log.gz
dbfs:/mnt/htweblog/u_ex140802.log.gz	u_ex140802.log.gz
dbfs:/mnt/htweblog/u_ex140803.log.gz	u_ex140803.log.gz
dbfs:/mnt/htweblog/u_ex140804.log.gz	u_ex140804.log.gz
dbfs:/mnt/htweblog/u_ex140805.log.gz	u_ex140805.log.gz
dbfs:/mnt/htweblog/u_ex140806.log.gz	u_ex140806.log.gz

2.3. Setting constants

To begin we need to find where our data is located. I created the constant `SERVER_LOGS_GZIP_FILES` to point to where each log file is.

We can use a sample of the data using the `SAMPLE_LOGS_GZIP_FILES`, controlling whether to use a sample set (i.e., first 10 log files) using the `USE_SAMPLE_SET` constant, which is `False` if we should use all data (for final submission) or `True` for development and debugging purposes.

In addition, we will use *GeoIP2 (http://dev.maxmind.com/geoip/)* to extrapolate information about the client's IP. This requires downloading the GeoLite2 Cities Database (http://geolite.maxmind.com/download/geoip/database/GeoLite2-City.mmdb.gz). This has been downloaded in the S3 Bucket as `GeoLite2-City.mmdb`.

```
GEOLITE_CITIES_DB_FILE = '/dbfs/' + PATH_TO_S3_MOUNT + '/geolite-db/GeoLite2-City.mmdb'
SAMPLE_LOGS_GZIP_FILES = PATH_TO_S3_MOUNT + '/sample-set/*.log.gz'
SERVER_LOGS_GZIP_FILES = PATH_TO_S3_MOUNT + '/*.log.gz'
# Change this to False if we want to run on the entire dataset, otherwise keep to True for testing/debugging
USE_SAMPLE_SET = False
```

3. Data Acquisition

3.1. Extract Data

Extract the files from the `SERVER_LOGS_GZIP_FILES` as an *Apache Spark RDD (http://spark.apache.org/docs/latest/programming-guide.html#resilient-distributed-datasets-rdds)*, then caching (http://spark.apache.org/docs/latest/quick-start.html#caching) it for better performance. Note that we can utilise reading directly from within the gzip, as per the external datasets (http://spark.apache.org/docs/latest/programming-guide.html#external-datasets) guide.

```
# If USE_SAMPLE_SET, then read from sample set directory, otherwise use all data
data_files = SAMPLE_LOGS_GZIP_FILES if USE_SAMPLE_SET else SERVER_LOGS_GZIP_FILES
print("Using %s set files at: %s" % ('sample' if USE_SAMPLE_SET else 'full', data_files))
# Load in Apache Spark RDD (Resillient Distributed Dataset)
logs_rdd = sc.textFile(data_files, use_unicode=False)
# Caching the data to a cluster-wide in-memory cache
logs_rdd.cache()
```

```
Using full set files at: /mnt/htweblog/*.log.gz
Out[12]: /mnt/htweblog/*.log.gz MapPartitionsRDD[35] at textFile at NativeMethodAccessorImpl.java:0
```

Extracting the fields from the dataset file (using the `Fields` comment):

```
# Strip fields from dataset, underscoring each instead of dasherizing it
fields  = (logs_rdd
           .filter(lambda line: line.startswith('#Fields:'))
           .map(lambda line: line.replace('-', '_'))
           .first()
           .split(' ')
          )[1:]
fields
```

```
Out[13]:
['date',
 'time',
 's_ip',
 'cs_method',
 'cs_uri_stem',
 'cs_uri_query',
 's_port',
 'cs_username',
 'c_ip',
 'cs(User_Agent)',
 'sc_status',
 'sc_substatus',
 'sc_win32_status',
 'time_taken']
```

3.2. Transform Data

Tranform the data by zipping the contents with each value, thereby producing a structured format of the key/value pair of each log entry. Also perform additional transformation on the dataset, such as:

- converting relevant integer strings into actual `int` s,
- converting the `date` and `time` fields into one `timestamp` field, as a `DateTime` object, and
- making the `cs(User_Agent)` field a little nicer to work with by changing it to just `user_agent`

```python
def map_integers(record):
    """Maps integer types in the record from unicode strings"""
    record['s_port'] = int(record['s_port'])
    record['sc_status'] = int(record['sc_status'])
    record['sc_substatus'] = int(record['sc_substatus'])
    record['sc_win32_status'] = int(record['sc_win32_status'])
    record['time_taken'] = int(record['time_taken'])
    return record

def map_timestamp(record):
    """Maps a record's date and time into one timestamp"""
    record['timestamp'] = datetime.strptime(record.pop('date') + record.pop('time'), '%Y-%m-%d%H:%M:%S')
    return record

def map_user_agent(record):
    """Maps the user agent field to be better used"""
    record['user_agent'] = record.pop('cs(User_Agent)')
    return record

# Map dataset contents (log lines) as dictionary from fields, then map using additional map functions
data = (logs_rdd
        .filter(lambda line: not line.startswith('#'))
        .map(lambda line: dict(zip(fields, line.split(' '))))
        .map(map_integers)
        .map(map_timestamp)
        .map(map_user_agent)
       )
```

3.3 Load Data

Load the data as a structured data frame (http://spark.apache.org/docs/latest/sql-programming-guide.html#datasets-and-dataframes) from its RDD, registering the data under the `Log` in-memory table. To load it, we will need to map each record as a Row (https://spark.apache.org/docs/1.1.1/api/python/pyspark.sql.Row-class.html) type.

```
logs_df = data.toDF()
logs_df.registerTempTable("Log")
# Cache the table for improved performance
sqlContext.cacheTable("Log")
# Show that the table has been registered
sqlContext.sql("DESCRIBE TABLE Log").show()
```

```
+--------------+---------+-------+
|      col_name|data_type|comment|
+--------------+---------+-------+
|          c_ip|   string|   null|
|     cs_method|   string|   null|
|  cs_uri_query|   string|   null|
|   cs_uri_stem|   string|   null|
|   cs_username|   string|   null|
|          s_ip|   string|   null|
|        s_port|   bigint|   null|
|     sc_status|   bigint|   null|
|  sc_substatus|   bigint|   null|
|sc_win32_status|   bigint|   null|
|    time_taken|   bigint|   null|
|     timestamp|timestamp|   null|
|    user_agent|   string|   null|
+--------------+---------+-------+
```

3.4. Additional Data Extraction

We can extract the distinct user agents from the log and load this into its own table, `UserAgent` .

```
def map_user_agent_partition(partition):
    """Maps a parition of IP addresses"""
    def map_user_agent(user_agent_str):
        """Maps the user agent in a record to extrapolate more specific information about the client platform"""
        agent_lookup = ua_parse(user_agent_str)
        return {
          'user_agent': user_agent_str,
          'user_agent_browser_name': agent_lookup.browser.family,
          'user_agent_browser_version': agent_lookup.browser.version_string,
          'user_agent_os_name': agent_lookup.os.family,
          'user_agent_os_version': agent_lookup.os.version_string,
          'user_agent_device_brand': agent_lookup.device.brand,
          'user_agent_device_model': agent_lookup.device.model,
          'user_agent_device_family': agent_lookup.device.family,
          'user_agent_is_pc': agent_lookup.is_pc,
          'user_agent_is_smartphone': agent_lookup.is_mobile,
          'user_agent_is_tablet': agent_lookup.is_tablet,
          'user_agent_is_bot': agent_lookup.is_bot
        }
    return [map_user_agent(record['user_agent']) for record in partition]

# Select all unique user agents from the logs
user_agents_rdd = sqlContext.sql('SELECT DISTINCT user_agent FROM Log WHERE user_agent IS NOT NULL').rdd
# Map user agents them using the mapping function above
user_agents_df = user_agents_rdd.mapPartitions(map_user_agent_partition).toDF()
# Register the dataframe as a table, UserAgent
user_agents_df.registerTempTable("UserAgent")
# Cache the table for improved performance
sqlContext.cacheTable("UserAgent")
# Show that the table has been registered
sqlContext.sql("DESCRIBE TABLE UserAgent").show()
```

```
+--------------------+---------+-------+
|            col_name|data_type|comment|
+--------------------+--------+-------+
|          user_agent|  string|   null|
|user_agent_browse...|  string|   null|
|user_agent_browse...|  string|   null|
|user_agent_device...|  string|   null|
|user_agent_device...|  string|   null|
|user_agent_device...|  string|   null|
|   user_agent_is_bot| boolean|   null|
|    user_agent_is_pc| boolean|   null|
|user_agent_is_sma...| boolean|   null|
|user_agent_is_tablet| boolean|   null|
|  user_agent_os_name|  string|   null|
|user_agent_os_ver...|  string|   null|
+--------------------+--------+-------+
```

Similarly, we can extract all the IP addresses into the `IPAddr` table.

```python
def map_ip_address_partition(partition):
    """Maps a parition of IP addresses"""
    # Must re-import geoip as mapping within new context
    # Refer to: http://stackoverflow.com/a/33755564/519967
    from geoip2 import database as geoipdb
    geoip_reader = geoipdb.Reader(GEOLITE_CITIES_DB_FILE)
    def map_ip_address(ip_address):
      """Maps a single IP address to extrapolate more specific information about the IP"""
      try:
        ip_lookup = geoip_reader.city(ip_address)
        return {
          'ip_address': ip_address,
          'country_code': ip_lookup.country.iso_code,
          'country_name': ip_lookup.country.name,
          'state_code': ip_lookup.subdivisions.most_specific.iso_code,
          'state_name': ip_lookup.subdivisions.most_specific.name,
            'city_name': ip_lookup.city.name,
            'lat': ip_lookup.location.latitude,
            'lng': ip_lookup.location.longitude
          }
        except:
            return None
    result = [map_ip_address(record['ip']) for record in partition]
    # Must close reader!
    geoip_reader.close()
    return result

# Select all unique user agents from the logs, then map them using the mapping function above
ip_addrs_rdd = sqlContext.sql("SELECT DISTINCT c_ip AS ip FROM Log WHERE c_ip IS NOT NULL").rdd
# Map using mapPartitions functions, removing those countries we can't find (i.e., private IP address)
ip_address_df = ip_addrs_rdd.mapPartitions(map_ip_address_partition).filter(lambda record: record != None).toDF()
# Register the dataframe as a table, UserAgent
ip_address_df.registerTempTable("IPAddr")
# Cache the table for improved performance
sqlContext.cacheTable("IPAddr")
# Show that the table has been registered
sqlContext.sql("DESCRIBE TABLE IPAddr").show()
```

```
+------------+---------+-------+
|    col_name|data_type|comment|
+------------+---------+-------+
|   city_name|   string|   null|
|country_code|   string|   null|
|country_name|   string|   null|
|  ip_address|   string|   null|
|         lat|   double|   null|
|         lng|   double|   null|
|  state_code|   string|   null|
|  state_name|   string|   null|
+------------+---------+-------+
```

We can see that data has now be loaded by counting the records of our three tables.

```
 sqlContext.sql("SELECT COUNT(*) AS count_of_logs         FROM Log").show()
 sqlContext.sql("SELECT COUNT(*) AS count_of_user_agents  FROM UserAgent").show()
 sqlContext.sql("SELECT COUNT(*) AS count_of_ip_addresses FROM IPAddr").show()
```

```
+-------------+
|count_of_logs|
+-------------+
|     73368256|
+-------------+

+--------------------+
|count_of_user_agents|
+--------------------+
|               57870|
+--------------------+

+---------------------+
|count_of_ip_addresses|
+---------------------+
|               522416|
+---------------------+
```

3.5. Persist Data

As I am using the free Databricks tier, where the cluster will restart after a few hours inactivity, I persisted the data so that I can work on the assignment over multiple days without re-loading the data. To do this, I persisted the data frames using the following:

```
logs_df.write.mode("ignore").saveAsTable("Log")
ip_address_df.write.mode("ignore").saveAsTable("IPAddr")
user_agents_df.write.mode("ignore").saveAsTable("UserAgent")
```

Then when I want to work with the data, cache it (https://docs.databricks.com/spark/latest/sparkr/functions/cacheTable.html) for improved query performance:

```
sqlContext.cacheTable("IPAddr")
sqlContext.cacheTable("UserAgent")
sqlContext.cacheTable("Log")
```

4. Informational Resource Transaction Extraction

4.1. Defining a user session

To extract session information, we generate a new field, the `session_idenfitier` , which is a hash-delimited concatenated string of the following information:

1. The client's ip, `c_ip` ,
2. The client's specific user agent string, `user_agent` ,
3. The client's session date (the `DATE` of the `timestamp`), and
4. The client's session hour (extracted using `DATE_FORMAT(timestamp, 'H')`)

We assume that one session is grouped by every hour on a specific date. We can therefore group the order of our requested URIs by the unique `session_identifier` we have created above.

One client IP may have multiple users, e.g., an internet café or the hotel lobby. Therefore we must split the client IP into sessions based on user agent. Our limitation here is that there may be two *separate* users requesting the page with the same user agent at the same IP within the same hour.

4.2. Defining which resources to mine

To ensure that we extract *informational resources* only, we add the following conditions to our request:

1. The request must return an `ashx` or `aspx` resource, not a media resource (e.g., JavaScript, Cascading Stylesheet, Image file etc.),
2. The request must not be from the `media` , `layouts` or `sitecore` admin directories as this is non-informational data,
3. The request must return a `200` response, and must not be a placeholder error page (i.e., `404.aspx` should be removed as this is non-informational)

Lowercase all the URIs to prevent case sensitiity (i.e., a user types in `/Home.aspx` vs `/home.aspx` ; semantically the same).

4.3. Functionalising the query

This is all constructed for us in the `construct_sql_query` function to keep query selection consistent and reduce duplication.

Using this function we can:

- compare the requests internally versus externally
- compare how the top three countries differ in their requests (referencing from Assignment 1 we saw these countries are `Hong Kong` , `USA` , `Australia`)
- compare how mobile versus tablet versus PC vs bot requests differ

```
def construct_sql_query(where = None, join = None):
  """ Constructs a consistent SQL query for extracting data from the database

  Args:
      where_clause (string): An optional string to add an extra WHERE clause to the query
```

```
    join_clause (string): An optional string to add an extra JOIN clause to the query
                          that must be in the format `JOIN <Table> ON <Join>`
Returns:
    string: A string to run on the database to extract data
"""
standard_query = """
  SELECT
  -- Session identifier defined as thus:
  CONCAT(
    -- [1] The client's IP address
    l.c_ip, '#',
    -- [2] The client's user agent string
    l.user_agent, '#',
    -- [3] The date of the request
    DATE(l.timestamp), '#',
    -- [4] The hour of the request
    DATE_FORMAT(l.timestamp, 'H')
  ) AS session_identifier,
  -- URI stem requested, all lowercase to prevent case sensitvity
  LCASE(l.cs_uri_stem)
  FROM Log l
  -- Add extra JOIN clause
  {join_clause}
  WHERE
    -- [1] ASHX or ASPX requests only to filter out other resources
    (l.cs_uri_stem LIKE "%.ashx" OR l.cs_uri_stem LIKE "%.aspx") AND
    -- [2] Remove media, layout templates or admin sitecore requests (non-informational resources)
    NOT (l.cs_uri_stem LIKE "%/~/media%" OR l.cs_uri_stem LIKE "%/layouts%" OR l.cs_uri_stem LIKE "%/sitecore/%") AND
    -- [3] Response codes of 200 and not the x0x pages (e.g. 404.aspx)
    (l.sc_status = 200 AND l.cs_uri_stem NOT LIKE "/%0%.aspx")
    -- Add extra WHERE clause
    {where_clause}
  GROUP BY 1, l.timestamp, l.cs_uri_stem
  ORDER BY l.timestamp
"""
```

```
  # Add an "AND" to the where clause if it exists
  where_clause = ("AND %s" % where) if where is not None else ""
  join_clause = join if join is not None else ""
  formatted_query = standard_query.format(join_clause=join_clause, where_clause=where_clause)
  # Return the formatted query
  return formatted_query


# Define IP range string for all INTERNAL requests
internal_ip_range_string = '(^127\.)|(^10\.)|(^172\.1[6-9]\.)|(^172\.2[0-9]\.)|(^172\.3[0-1]\.)|(^192\.168\.)'

sql_queries = {
  # Internal vs external
  "internal_requests": construct_sql_query(where = "l.c_ip REGEXP '%s'" % internal_ip_range_string),
  "external_requests": construct_sql_query(where = "l.c_ip NOT REGEXP '%s'" % internal_ip_range_string),
  # Top three countries
  "hk_requests": construct_sql_query(join = "JOIN IPAddr i ON l.c_ip = i.ip_address", where="i.country_code = 'HK'"),
  "us_requests": construct_sql_query(join = "JOIN IPAddr i ON l.c_ip = i.ip_address", where="i.country_code = 'US'"),
  "au_requests": construct_sql_query(join = "JOIN IPAddr i ON l.c_ip = i.ip_address", where="i.country_code = 'AU'"),
  # PC vs bots vs tablets vs smartphones
  "pc_requests":         construct_sql_query(join = "JOIN UserAgent ua ON ua.user_agent = l.user_agent",
where="ua.user_agent_is_pc"),
  "tablet_requests":     construct_sql_query(join = "JOIN UserAgent ua ON ua.user_agent = l.user_agent",
where="ua.user_agent_is_tablet"),
  "bots_requests":       construct_sql_query(join = "JOIN UserAgent ua ON ua.user_agent = l.user_agent",
where="ua.user_agent_is_bot"),
  "smartphone_requests": construct_sql_query(join = "JOIN UserAgent ua ON ua.user_agent = l.user_agent",
where="ua.user_agent_is_smartphone")
}
```

Run this as an SQL query under the SQL query context to extract the transactional data we are interested in. Map it into a `tuple` type, representing the `extracted` data as a `(Key, Value)` tuple.

Define this as a function to allow for multiple queries to be made.

```
def extract_data(sql_query):
    """ Extracts data from the database given the SQL query
    Args:
        sql_query (str): a string containing the SQL query used to extract the data.
    Returns:
        list<tuple>: a list of all records as a tuple of `(session_identifier, cs_uri_stem)`.
    """
    return sqlContext.sql(sql_query).rdd.map(lambda record: (record[0], record[1]))
```

Now extract the data for every `sql_query` in our `sql_queries`:

```
# Loop through every query and extract data
extracted_data = {key: extract_data(sql_query) for key, sql_query in sql_queries.items()}
```

5. Training the Model

We now mine for frequent patterns in our transactions Spark FPGrowth implementation.

To do this, we group all of the extracted data by the unique `session_identifier` key. The `sessionPair` is a the Key/Value pair whose key is the `session_identifier` and whose value is a unique set of the `cs_uri_stem`s accessed. This becomes our list of transactions.

To ensure we access multiple hits in a given session, we will show only those patterns with at least 3 hits in the session.

From this, we produce a list of `FreqItemset`s representing the frequency pattern of resources from the above transactions extracted, sorted in descending frequency order.

```python
def train_model(extracted, min_support_level = 0.01):
    """ Train a model using the Frequency Pattern Growth imported from Spark.

    Extracts the transactions used to train the model and supply it with a provided minimum
    support level.

    Args:
        extracted (list<tuple>): a list of all extracted records from the database.
        min_support_level (float): the threshold for a `FreqItemset` to be identified as
                                   frequent, defaults to `0.01`.

    Returns:
        list<FreqItemset>: A list of the `FreqItemset` identified sorted by descending
                           frequency values.
    """
    transactions = (extracted
                    # Group by each session id
                    .groupByKey()
                    # Extract out a set of each URI hit
                    .map(lambda sessionPair: set(sessionPair[1]))
                   )
    model = FPGrowth.train(transactions, minSupport=min_support_level, numPartitions=6)
    sorted_itemsets = (model.freqItemsets()
                       # Only show item sets with 3 or more hits in the set
                       .filter(lambda itemset: len(itemset.items) >= 3)
                       # Sort in reverse order by frequencies
                       .sortBy(lambda itemset: itemset.freq, False)
                       .collect()
                      )
    return sorted_itemsets
```

Now train the models and print off each of our `FreqItemset` s. For some requests, we relax the pattern minimum support level to either 75% or 50%, as at a minimum support level is `0.01` retrieves few patterns.

```
# Set the default and relaxed min support levels
default_min_support_level = 0.01
relaxed_min_support_level = {
  "hk_requests": default_min_support_level * 0.5,
  "au_requests": default_min_support_level * 0.75,
  "smartphone_requests": default_min_support_level * 0.5,
  "pc_requests": default_min_support_level * 0.5,
  "tablet_requests": default_min_support_level * 0.5
}
# Loop through every extracted data and train using that model
sorted_itemsets = {
  key: train_model(data, min_support_level=relaxed_min_support_level.get(key, default_min_support_level))
  for key, data in extracted_data.items()
}
for key, itemset in sorted_itemsets.iteritems():
  print "%s itemset" % key
  print
  for item in itemset:
    print item
```

```
au_requests itemset

FreqItemset(items=[u'/above-and-beyond.aspx', u'/facilities.aspx', u'/rooms.aspx'], freq=152)
FreqItemset(items=[u'/above-and-beyond.aspx', u'/rooms.aspx', u'/offers.aspx'], freq=151)
FreqItemset(items=[u'/facilities.aspx', u'/rooms.aspx', u'/offers.aspx'], freq=141)
FreqItemset(items=[u'/about-the-hotel.aspx', u'/rooms.aspx', u'/offers.aspx'], freq=122)
FreqItemset(items=[u'/about-the-hotel.aspx', u'/facilities.aspx', u'/rooms.aspx'], freq=120)
FreqItemset(items=[u'/above-and-beyond.aspx', u'/dining.aspx', u'/rooms.aspx'], freq=120)
FreqItemset(items=[u'/above-and-beyond.aspx', u'/facilities.aspx', u'/offers.aspx'], freq=115)
FreqItemset(items=[u'/facilities.aspx', u'/dining.aspx', u'/rooms.aspx'], freq=107)
FreqItemset(items=[u'/above-and-beyond.aspx', u'/facilities.aspx', u'/dining.aspx'], freq=105)
FreqItemset(items=[u'/dining.aspx', u'/rooms.aspx', u'/offers.aspx'], freq=103)
FreqItemset(items=[u'/about-the-hotel.aspx', u'/above-and-beyond.aspx', u'/facilities.aspx'], freq=100)
FreqItemset(items=[u'/above-and-beyond.aspx', u'/dining.aspx', u'/offers.aspx'], freq=100)
FreqItemset(items=[u'/about-the-hotel.aspx', u'/above-and-beyond.aspx', u'/rooms.aspx'], freq=97)
FreqItemset(items=[u'/location-and-contacts.aspx', u'/rooms.aspx', u'/offers.aspx'], freq=92)
FreqItemset(items=[u'/about-the-hotel.aspx', u'/facilities.aspx', u'/offers.aspx'], freq=88)
FreqItemset(items=[u'/about-the-hotel.aspx', u'/dining.aspx', u'/rooms.aspx'], freq=85)
FreqItemset(items=[u'/facilities.aspx', u'/dining.aspx', u'/offers.aspx'], freq=79)
FreqItemset(items=[u'/our-city.aspx', u'/facilities.aspx', u'/rooms.aspx'], freq=76)
FreqItemset(items=[u'/about-the-hotel.aspx', u'/location-and-contacts.aspx', u'/rooms.aspx'], freq=74)
```

6. Visualisation of Model

6.1. Visualisation using NetworkX

Below we visualise how people navigate through the site using a Multi-Directional Network Graph (https://networkx.github.io/documentation/networkx-1.10/reference/classes.multidigraph.html).

To prevent excessive amounts of data being plotted, we can use the `frequency_threshold_percentile` variable to change how many FP Itemsets are shown. By default, only the top 25% (those with frequencies above the third percentile) will be plotted to keep the visualisations readable. Not doing so lead to unreadable graphs (https://i.imgur.com/LddbGNO.png).

The thick ends of the lines indicate the "to" direction (i.e., the line from `home` to `offers` has a thick stub toward `offers`, meaning that users would go from home to offers). The values in between each line indicate the frequency of the pattern.

```
def create_directed_network_graph(sorted_itemsets, frequency_threshold_percentile=75):
  """ Creates a directed network graph of the frequency interaction patterns.

  Args:
      sorted_itemsets (list<FreqItemset>): A list of the `FreqItemset` identified
                                           sorted by descending frequency values.

      frequency_threshold_percentile (int): The value of of the minimum percentile to accept
                                            when plotting. Defaults to the 75th percentile.

  Returns:
      tuple: A tuple containing the `NetworkX.DiGraph` and CSV representation (`string`)
             of interaction patterns: `(graph, csv)`
  """
  # Declare our new graph
  graph = nx.MultiDiGraph()

  # Declare an empty dictionary for the edge labels
  edge_labels = {}

  # CSV to be tabulated in LaTeX
  csv = "Sequence,From,To,Frequency"

  # Work out which frequencies we will plot within our threshold
  assert frequency_threshold_percentile <= 100 and frequency_threshold_percentile >= 0, "Threshold must be a percentage
between 0 and 1"
```

```
highest_frequency = sorted_itemsets[0].freq
all_frequencies = np.array([ itemset.freq for itemset in sorted_itemsets ])
# Accept the "top nth" percentile
accepted_minimum_frequency = np.percentile(all_frequencies, frequency_threshold_percentile)
# Filter out sorted_frequencies
accepted_itemsets = [itemset for itemset in sorted_itemsets if itemset.freq >= accepted_minimum_frequency]

# Define a colormap for each sequence
cmap = plt.cm.get_cmap('Set1', len(accepted_itemsets))
sequence_colors = [colors.rgb2hex(cmap(i)[:3]) for i in range(cmap.N)]

# Add in each node
for sequence, freq_itemset in enumerate(accepted_itemsets):
  # 'Clean up' the label by removing the '.aspx' and leading forward slash
  items = [label[1:-5].replace('-', ' ') for label in freq_itemset.items]
  num_items = len(items)
  # Define freq
  freq = freq_itemset.freq
  # Find the previous and following node in the set
  for i, item in enumerate(items):
    node_from, node_to = items[0 + i:2 + i]
    edge_labels[(node_from, node_to)] = freq
    # Add to our CSV
    csv = "%s\n%i,%s,%s,%i" % (csv, sequence, node_from, node_to, freq)
    label = "%i/%i" % (sequence, freq)
    graph.add_edge(node_from, node_to, weight=freq, label=label, color=sequence_colors[sequence])
    # Break the loop so we don't go out of range!
    if num_items - i == 2:
      break

# Set up the layout of the graph
pos = nx.shell_layout(graph, scale=8)

# Draw the nodes
nx.draw_networkx_nodes(graph, pos, node_size=1000)
```

```
    # Draw the edges
    nx.draw_networkx_edges(graph, pos)

    # Draw the labels
    nx.draw_networkx_labels(graph, pos, font_size=10, font_family='serif')
    nx.draw_networkx_edge_labels(graph, pos, font_family='serif', font_size=7, alpha=0.5, edge_labels=edge_labels)

    return (graph, csv)

def plot_visualisation(sorted_itemsets, frequency_threshold_percentile =75):
    """ Plots the visualisation of a specific set of sorted frequencies

    Args:
        sorted_itemsets (tuple): The sorted frequencies to visualise

        frequency_threshold_percentile (int): The value of of the minimum percentile to accept
                                              when plotting. Defaults to the 75th percentile.



    Returns:
        tuple: A tuple containing the `NetworkX.DiGraph` and CSV representation (`string`)
               of interaction patterns: `(graph, csv)`
    """
    # Clear last plotted functions
    plt.clf()

    graph, csv = create_directed_network_graph(sorted_itemsets, frequency_threshold_percentile)

    # Disable the axis and plot
    plt.axis('off')
    display(plt.show())

    return (graph, csv)
```

We can now call our function to visualise our respective graphs.

6.1.1. Internal Requests

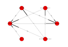
```
graph_data = {}
graph_data["internal_requests"] = plot_visualisation(sorted_itemsets["internal_requests"])
```

6.1.2. External Requests

```
graph_data["external_requests"] = plot_visualisation(sorted_itemsets["external_requests"])
```

6.1.3. Hong Kong Requests

```
graph_data["hk_requests"] = plot_visualisation(sorted_itemsets["hk_requests"])
```



6.1.4. USA Requests

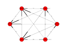
```
graph_data["us_requests"] = plot_visualisation(sorted_itemsets["us_requests"])
```

6.1.5. Australian Requests

```
graph_data["au_requests"] = plot_visualisation(sorted_itemsets["au_requests"])
```

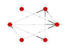
6.1.6. PC Requests

```
graph_data["pc_requests"] = plot_visualisation(sorted_itemsets["pc_requests"])
```



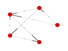
6.1.7. Smartphone Requests

```
graph_data["smartphone_requests"] = plot_visualisation(sorted_itemsets["smartphone_requests"])
```



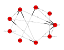
6.1.8. Tablet Requests

```
graph_data["tablet_requests"] = plot_visualisation(sorted_itemsets["tablet_requests"])
```



6.1.9. Bot Requests

```
graph_data["bots_requests"] = plot_visualisation(sorted_itemsets["bots_requests"])
```



6.2. Improved plotting using GraphViz

However, the above is hard to read, especially the frequency values. We can convert the graph into a Graphviz (http://www.graphviz.org) diagram string. Install the dependency as needed:

```
$ brew install graphviz
```

Running the command below, we can copy the output and run through the `dot` command provided by Graphviz:

```
 $ pbpaste > a2.dot
$ dot internal_requests.dot -T pdf > internal_requests.pdf
```

```
def graph_to_pydot_string(graph, layout="dot"):
  """ Converts the graph to a representable Graphviz diagram using pydot

  Args:
      graph (`NetworkX.DiGraph`): The graph to convert

  Returns:
      string: A string representing the Graphviz diagram string with the
              layout specified, defaults to `dot`.
  """
  string = nx.drawing.nx_pydot.to_pydot(graph).to_string()
  # Split all lines to add the specified layout
  lines = string.split("\n")
  lines.insert(1, 'layout="%s";' % layout)
  return "\n".join(lines)
```

6.2.1. Internal Requests

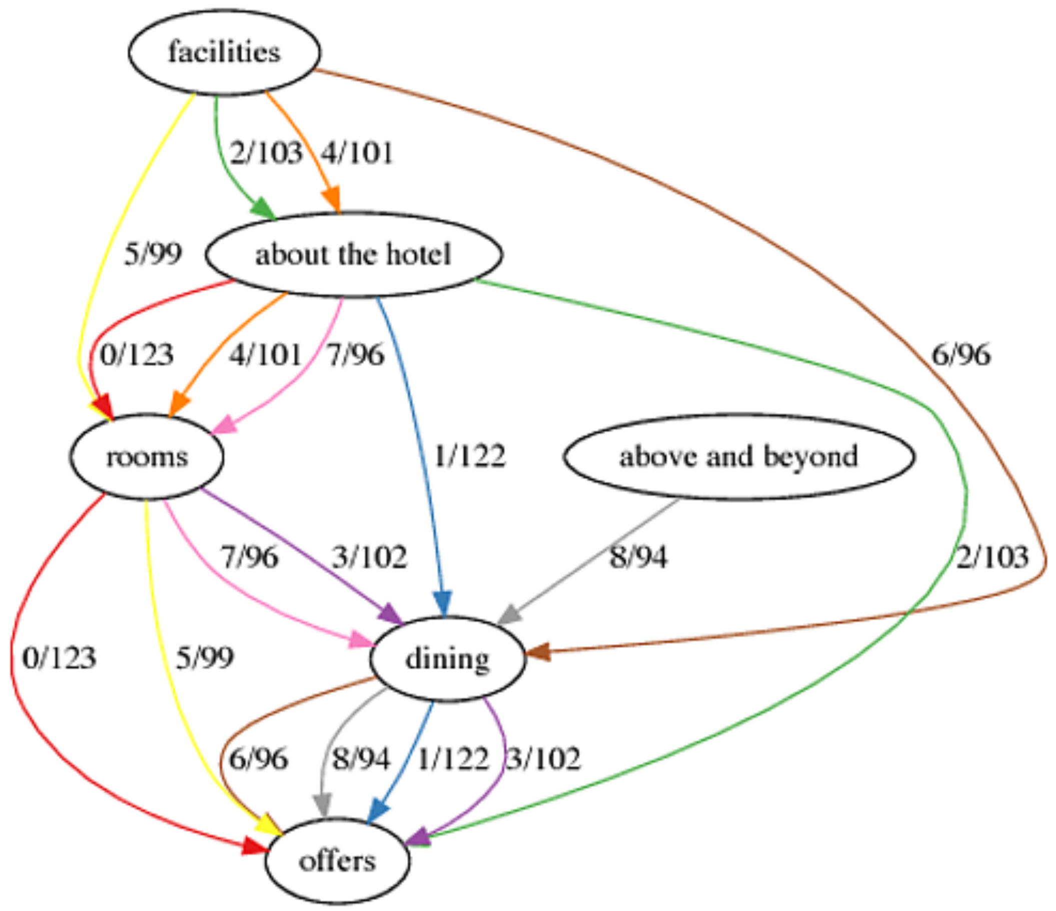
Now run our conversion function on our graph:

```
print graph_to_pydot_string(graph_data["internal_requests"][0])
```

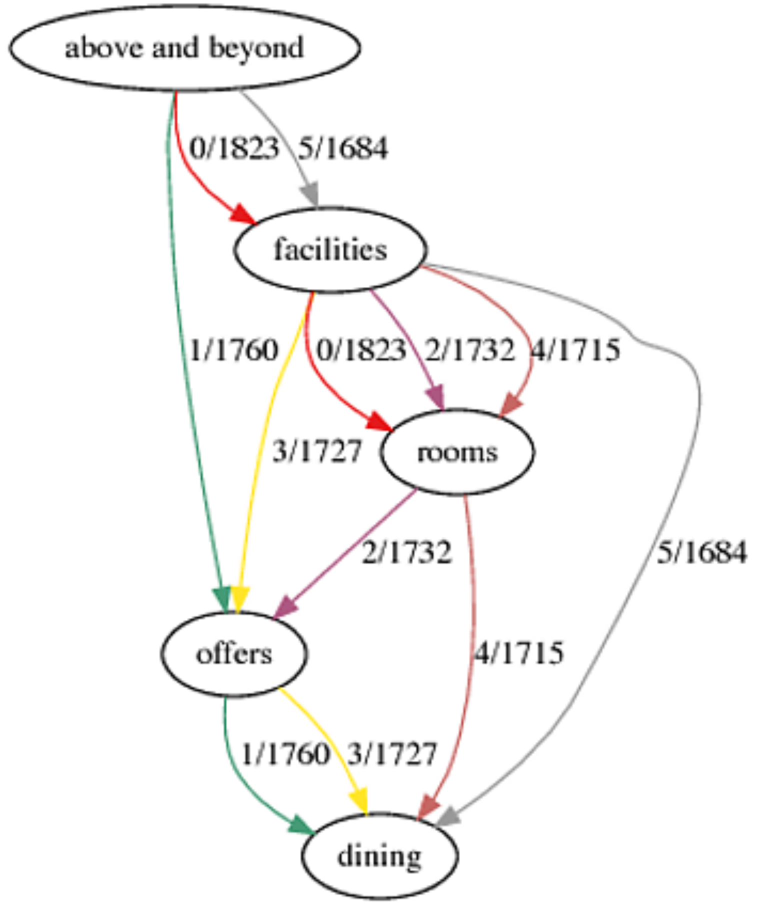
```
digraph "" {
layout="dot";
facilities;
"about the hotel";
dining;
offers;
rooms;
"above and beyond";
facilities -> dining  [color="#a65628", key=0, label="6/96", weight=96];
facilities -> rooms  [color="#ffff33", key=0, label="5/99", weight=99];
facilities -> "about the hotel"  [color="#4daf4a", key=0, label="2/103", weight=103];
facilities -> "about the hotel"  [color="#ff7f00", key=1, label="4/101", weight=101];
"about the hotel" -> dining  [color="#377eb8", key=0, label="1/122", weight=122];
"about the hotel" -> offers  [color="#4daf4a", key=0, label="2/103", weight=103];
"about the hotel" -> rooms  [color="#e41a1c", key=0, label="0/123", weight=123];
"about the hotel" -> rooms  [color="#ff7f00", key=1, label="4/101", weight=101];
"about the hotel" -> rooms  [color="#f781bf", key=2, label="7/96", weight=96];
dining -> offers  [color="#377eb8", key=0, label="1/122", weight=122];
dining -> offers  [color="#984ea3", key=1, label="3/102", weight=102];
dining -> offers  [color="#a65628", key=2, label="6/96", weight=96];
dining -> offers  [color="#999999", key=3, label="8/94", weight=94];
```

Save the output above to file `internal_requests.dot` and convert using the commands described above.

This produces a much cleaner looking output, where edges are colorised for assisting with reading frequency patterns between pages.



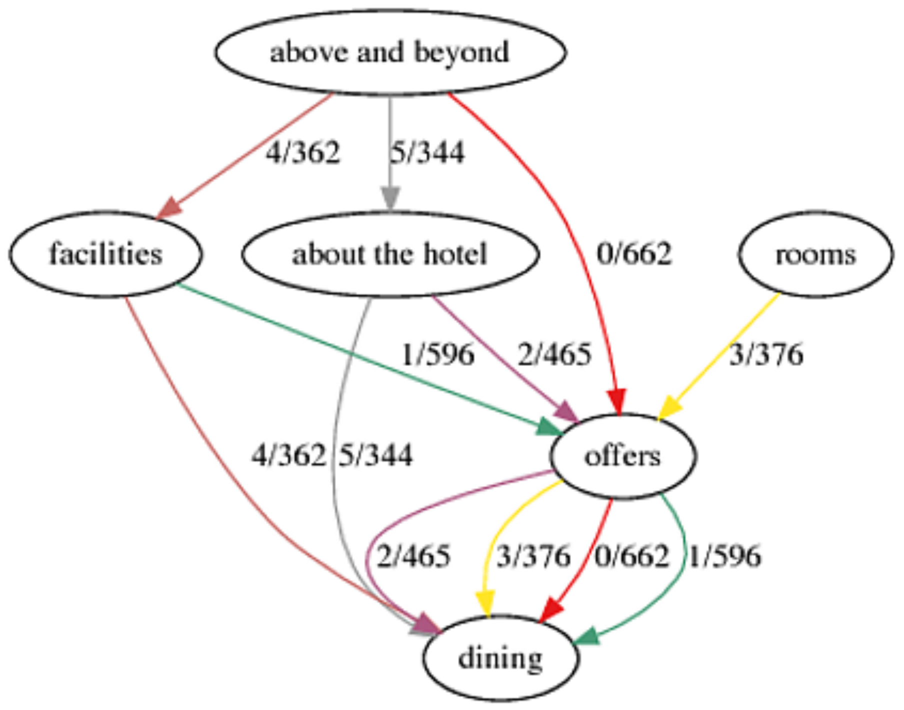
6.2.2. External Requests



```
print graph_to_pydot_string(graph_data["external_requests"][0])
```

```
digraph "" {
layout="dot";
dining;
"above and beyond";
offers;
rooms;
facilities;
"above and beyond" -> facilities  [color="#e41a1c", key=0, label="0/1823", weight=1823];
"above and beyond" -> facilities  [color="#999999", key=1, label="5/1684", weight=1684];
"above and beyond" -> offers  [color="#449b76", key=0, label="1/1760", weight=1760];
offers -> dining  [color="#449b76", key=0, label="1/1760", weight=1760];
offers -> dining  [color="#ffe529", key=1, label="3/1727", weight=1727];
rooms -> dining  [color="#c66764", key=0, label="4/1715", weight=1715];
rooms -> offers  [color="#ad5882", key=0, label="2/1732", weight=1732];
facilities -> dining  [color="#999999", key=0, label="5/1684", weight=1684];
facilities -> offers  [color="#ffe529", key=0, label="3/1727", weight=1727];
facilities -> rooms  [color="#e41a1c", key=0, label="0/1823", weight=1823];
facilities -> rooms  [color="#ad5882", key=1, label="2/1732", weight=1732];
facilities -> rooms  [color="#c66764", key=2, label="4/1715", weight=1715];
}
```

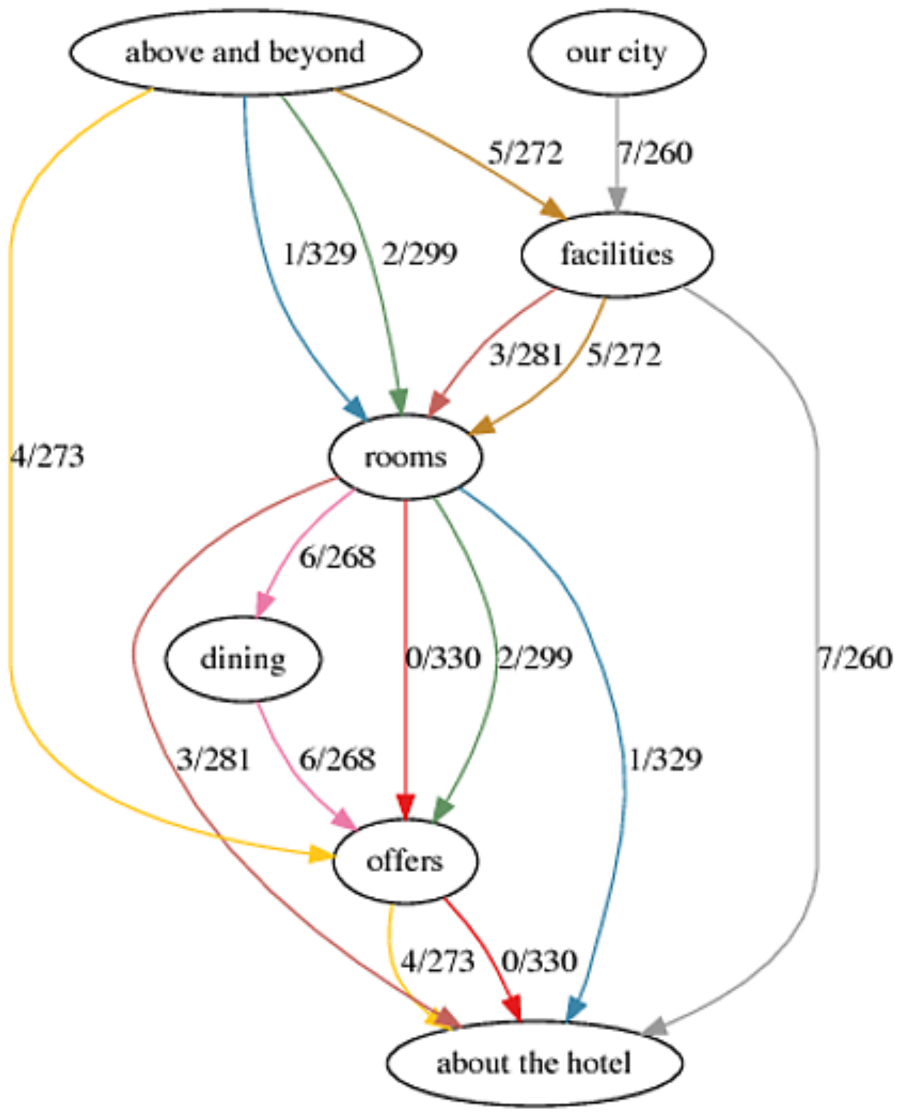
6.2.3. Hong Kong Requests



```
print graph_to_pydot_string(graph_data["hk_requests"][0])
```

```
digraph "" {
layout="dot";
dining;
"about the hotel";
facilities;
offers;
rooms;
"above and beyond";
"about the hotel" -> dining  [color="#999999", key=0, label="5/344", weight=344];
"about the hotel" -> offers  [color="#ad5882", key=0, label="2/465", weight=465];
facilities -> dining  [color="#c66764", key=0, label="4/362", weight=362];
facilities -> offers  [color="#449b76", key=0, label="1/596", weight=596];
offers -> dining  [color="#e41a1c", key=0, label="0/662", weight=662];
offers -> dining  [color="#449b76", key=1, label="1/596", weight=596];
offers -> dining  [color="#ad5882", key=2, label="2/465", weight=465];
offers -> dining  [color="#ffe529", key=3, label="3/376", weight=376];
rooms -> offers  [color="#ffe529", key=0, label="3/376", weight=376];
"above and beyond" -> facilities  [color="#c66764", key=0, label="4/362", weight=362];
"above and beyond" -> offers  [color="#e41a1c", key=0, label="0/662", weight=662];
"above and beyond" -> "about the hotel"  [color="#999999", key=0, label="5/344", weight=344];
}
```

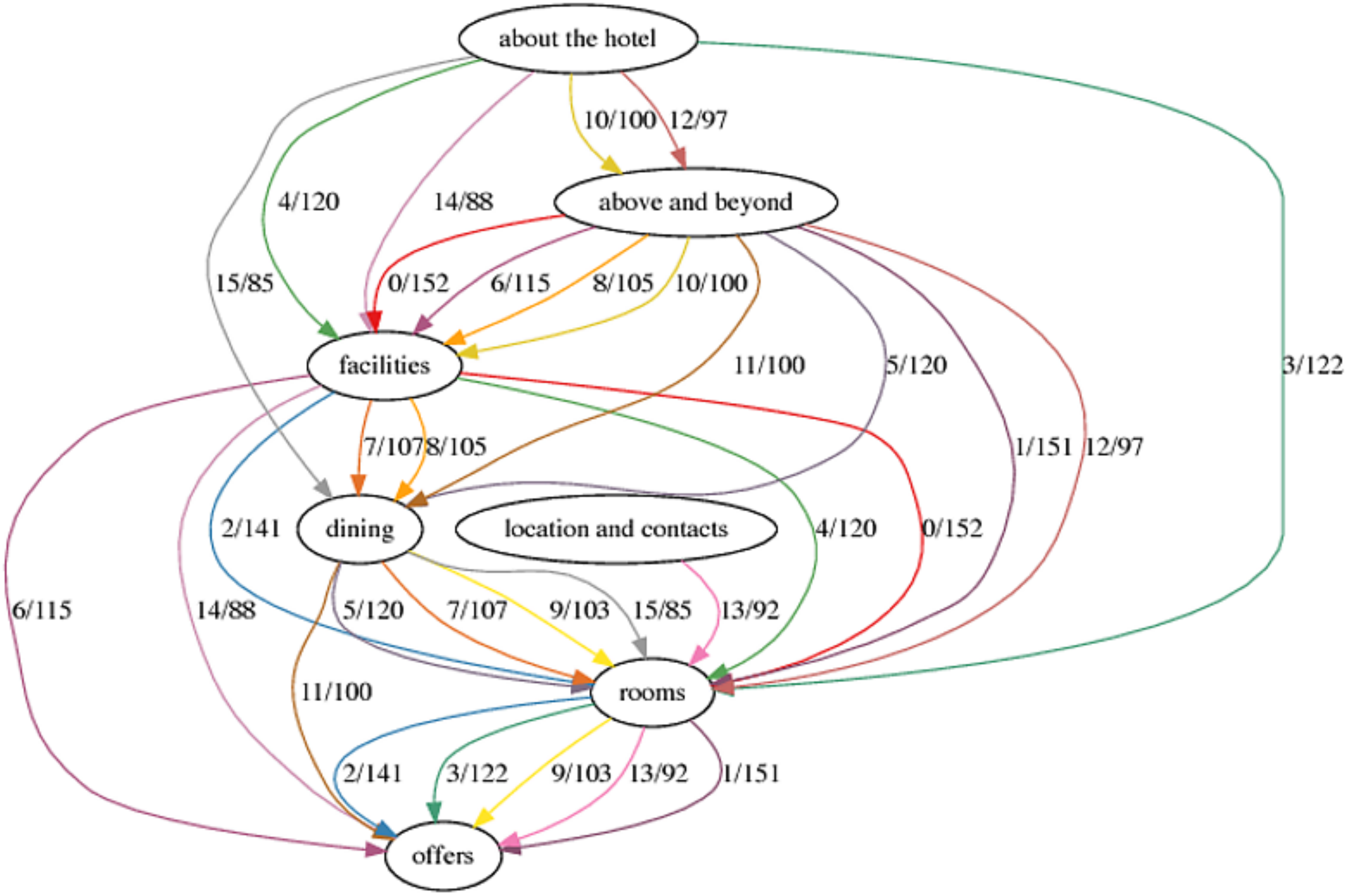
6.2.4. USA Requests



```
print graph_to_pydot_string(graph_data["us_requests"][0])
```

```
digraph "" {
layout="dot";
facilities;
"about the hotel";
dining;
offers;
rooms;
"our city";
"above and beyond";
facilities -> rooms  [color="#c4635d", key=0, label="3/281", weight=281];
facilities -> rooms  [color="#bf862b", key=1, label="5/272", weight=272];
facilities -> "about the hotel"  [color="#999999", key=0, label="7/260", weight=260];
dining -> offers  [color="#eb7ba9", key=0, label="6/268", weight=268];
offers -> "about the hotel"  [color="#e41a1c", key=0, label="0/330", weight=330];
offers -> "about the hotel"  [color="#ffc81d", key=1, label="4/273", weight=273];
rooms -> dining  [color="#eb7ba9", key=0, label="6/268", weight=268];
rooms -> offers  [color="#e41a1c", key=0, label="0/330", weight=330];
rooms -> offers  [color="#629363", key=1, label="2/299", weight=299];
rooms -> "about the hotel"  [color="#3a85a8", key=0, label="1/329", weight=329];
rooms -> "about the hotel"  [color="#c4635d", key=1, label="3/281", weight=281];
"our city" -> facilities  [color="#999999", key=0, label="7/260", weight=260];
"above and beyond" -> facilities  [color="#bf862b", key=0, label="5/272", weight=272];
"above and beyond" -> offers  [color="#ffc81d", key=0, label="4/273", weight=273];
"above and beyond" -> rooms  [color="#3a85a8", key=0, label="1/329", weight=329];
"above and beyond" -> rooms  [color="#629363", key=1, label="2/299", weight=299];
}
```

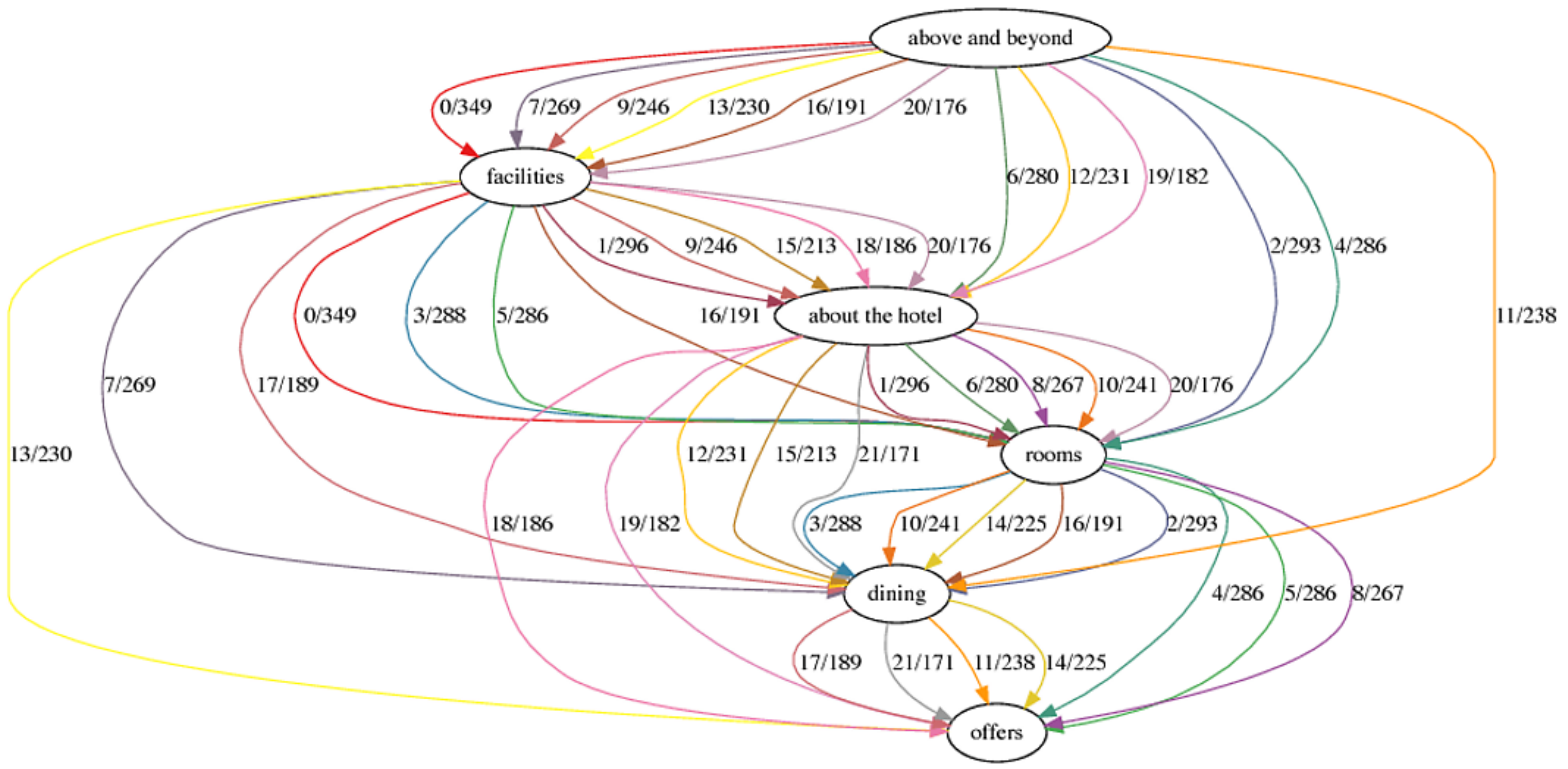
6.2.5. Australian Requests



```
print graph_to_pydot_string(graph_data["au_requests"][0])
```

```
digraph "" {
layout="dot";
facilities;
"about the hotel";
dining;
offers;
rooms;
"above and beyond";
"location and contacts";
facilities -> dining  [color="#e4722b", key=0, label="7/107", weight=107];
facilities -> dining  [color="#ffa10e", key=1, label="8/105", weight=105];
facilities -> offers  [color="#ad5882", key=0, label="6/115", weight=115];
facilities -> offers  [color="#cb8cad", key=1, label="14/88", weight=88];
facilities -> rooms  [color="#e41a1c", key=0, label="0/152", weight=152];
facilities -> rooms  [color="#3881b1", key=1, label="2/141", weight=141];
facilities -> rooms  [color="#57a256", key=2, label="4/120", weight=120];
"about the hotel" -> dining  [color="#999999", key=0, label="15/85", weight=85];
"about the hotel" -> facilities  [color="#57a256", key=0, label="4/120", weight=120];
"about the hotel" -> facilities  [color="#cb8cad", key=1, label="14/88", weight=88];
"about the hotel" -> rooms  [color="#449b76", key=0, label="3/122", weight=122];
"about the hotel" -> "above and beyond"  [color="#e1c72f", key=0, label="10/100", weight=100];
"about the hotel" -> "above and beyond"  [color="#c66764", key=1, label="12/97", weight=97];
dining -> offers  [color="#b26d29", key=0, label="11/100", weight=100];
dining -> rooms  [color="#7f6e85", key=0, label="5/120", weight=120];
dining -> rooms  [color="#e4722b", key=1, label="7/107", weight=107];
dining -> rooms  [color="#ffe529", key=2, label="9/103", weight=103];
dining -> rooms  [color="#999999", key=3, label="15/85", weight=85];
rooms -> offers  [color="#884f6f", key=0, label="1/151", weight=151];
rooms -> offers  [color="#3881b1", key=1, label="2/141", weight=141];
rooms -> offers  [color="#449b76", key=2, label="3/122", weight=122];
rooms -> offers  [color="#ffe529", key=3, label="9/103", weight=103];
rooms -> offers  [color="#f27eb5", key=4, label="13/92", weight=92];
"above and beyond" -> facilities  [color="#e41a1c", key=0, label="0/152", weight=152];
"above and beyond" -> facilities  [color="#ad5882", key=1, label="6/115", weight=115];
"above and beyond" -> facilities  [color="#ffa10e", key=2, label="8/105", weight=105];
"above and beyond" -> facilities  [color="#e1c72f", key=3, label="10/100", weight=100];
"above and beyond" -> rooms  [color="#884f6f", key=0, label="1/151", weight=151];
"above and beyond" -> rooms  [color="#c66764", key=1, label="12/97", weight=97];
"above and beyond" -> dining  [color="#7f6e85", key=0, label="5/120", weight=120];
"above and beyond" -> dining  [color="#b26d29", key=1, label="11/100", weight=100];
"location and contacts" -> rooms  [color="#f27eb5", key=0, label="13/92", weight=92];
}
```

6.2.6. PC Requests

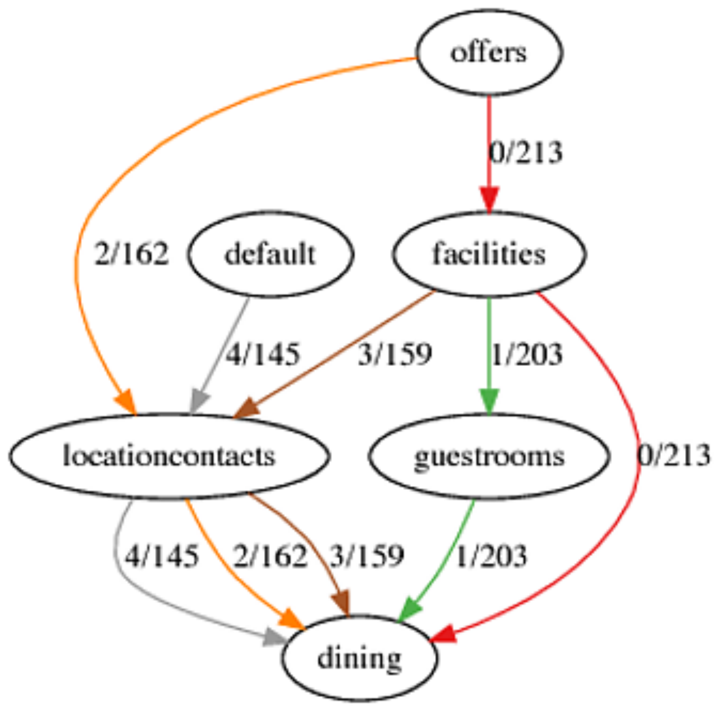


```
print graph_to_pydot_string(graph_data["pc_requests"][0])
```

```
digraph "" {
layout="dot";
facilities;
"about the hotel";
dining;
```

```
offers;
rooms;
"above and beyond";
facilities -> dining  [color="#7f6e85", key=0, label="7/269", weight=269];
facilities -> dining  [color="#cd6a70", key=1, label="17/189", weight=189];
facilities -> offers  [color="#fff931", key=0, label="13/230", weight=230];
facilities -> rooms  [color="#e41a1c", key=0, label="0/349", weight=349];
facilities -> rooms  [color="#3a85a8", key=1, label="3/288", weight=288];
facilities -> rooms  [color="#4baa54", key=2, label="5/286", weight=286];
facilities -> rooms  [color="#ae5a36", key=3, label="16/191", weight=191];
facilities -> "about the hotel"  [color="#a24057", key=0, label="1/296", weight=296];
facilities -> "about the hotel"  [color="#c4635d", key=1, label="9/246", weight=246];
facilities -> "about the hotel"  [color="#bf862b", key=2, label="15/213", weight=213];
facilities -> "about the hotel"  [color="#eb7ba9", key=3, label="18/186", weight=186];
facilities -> "about the hotel"  [color="#bd90a7", key=4, label="20/176", weight=176];
"about the hotel" -> dining  [color="#ffc81d", key=0, label="12/231", weight=231];
"about the hotel" -> dining  [color="#bf862b", key=1, label="15/213", weight=213];
"about the hotel" -> dining  [color="#999999", key=2, label="21/171", weight=171];
"about the hotel" -> offers  [color="#eb7ba9", key=0, label="18/186", weight=186];
"about the hotel" -> offers  [color="#e187b6", key=1, label="19/182", weight=182];
"about the hotel" -> rooms  [color="#a24057", key=0, label="1/296", weight=296];
"about the hotel" -> rooms  [color="#629363", key=1, label="6/280", weight=280];
"about the hotel" -> rooms  [color="#9d509b", key=2, label="8/267", weight=267];
"about the hotel" -> rooms  [color="#eb761f", key=3, label="10/241", weight=241];
"about the hotel" -> rooms  [color="#bd90a7", key=4, label="20/176", weight=176];
dining -> offers  [color="#ff970a", key=0, label="11/238", weight=238];
dining -> offers  [color="#e1c72f", key=1, label="14/225", weight=225];
dining -> offers  [color="#cd6a70", key=2, label="17/189", weight=189];
dining -> offers  [color="#999999", key=3, label="21/171", weight=171];
rooms -> dining  [color="#606693", key=0, label="2/293", weight=293];
rooms -> dining  [color="#3a85a8", key=1, label="3/288", weight=288];
rooms -> dining  [color="#eb761f", key=2, label="10/241", weight=241];
rooms -> dining  [color="#e1c72f", key=3, label="14/225", weight=225];
rooms -> dining  [color="#ae5a36", key=4, label="16/191", weight=191];
rooms -> offers  [color="#43987e", key=0, label="4/286", weight=286];
rooms -> offers  [color="#4baa54", key=1, label="5/286", weight=286];
rooms -> offers  [color="#9d509b", key=2, label="8/267", weight=267];
"above and beyond" -> dining  [color="#ff970a", key=0, label="11/238", weight=238];
"above and beyond" -> facilities  [color="#e41a1c", key=0, label="0/349", weight=349];
"above and beyond" -> facilities  [color="#7f6e85", key=1, label="7/269", weight=269];
"above and beyond" -> facilities  [color="#c4635d", key=2, label="9/246", weight=246];
"above and beyond" -> facilities  [color="#fff931", key=3, label="13/230", weight=230];
"above and beyond" -> facilities  [color="#ae5a36", key=4, label="16/191", weight=191];
"above and beyond" -> facilities  [color="#bd90a7", key=5, label="20/176", weight=176];
"above and beyond" -> rooms  [color="#606693", key=0, label="2/293", weight=293];
"above and beyond" -> rooms  [color="#43987e", key=1, label="4/286", weight=286];
"above and beyond" -> "about the hotel"  [color="#629363", key=0, label="6/280", weight=280];
"above and beyond" -> "about the hotel"  [color="#ffc81d", key=1, label="12/231", weight=231];
"above and beyond" -> "about the hotel"  [color="#e187b6", key=2, label="19/182", weight=182];
}
```

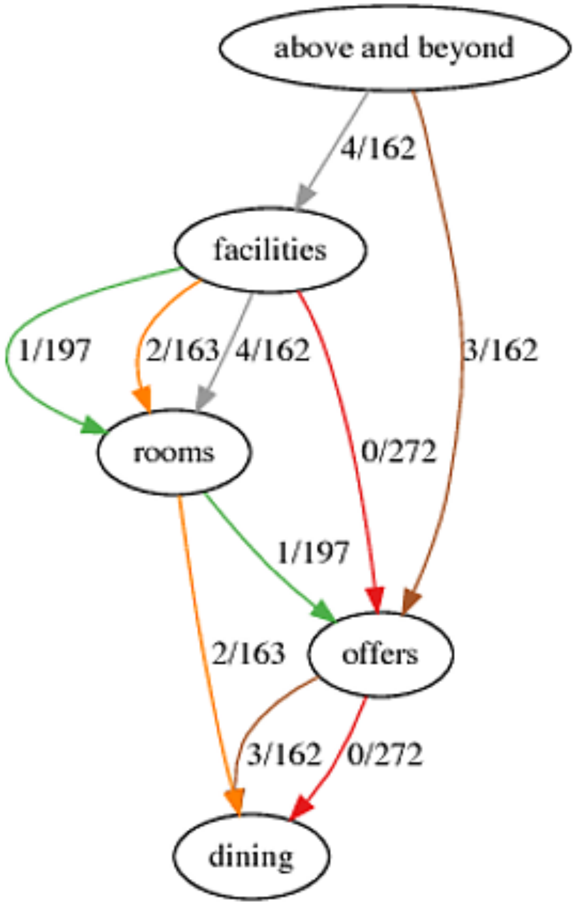
6.2.7. Smartphone Requests



```
print graph_to_pydot_string(graph_data["smartphone_requests"][0])
```

```
digraph "" {
layout="dot";
locationcontacts;
dining;
default;
facilities;
offers;
guestrooms;
locationcontacts -> dining  [color="#ff7f00", key=0, label="2/162", weight=162];
locationcontacts -> dining  [color="#a65628", key=1, label="3/159", weight=159];
locationcontacts -> dining  [color="#999999", key=2, label="4/145", weight=145];
default -> locationcontacts  [color="#999999", key=0, label="4/145", weight=145];
facilities -> dining  [color="#e41a1c", key=0, label="0/213", weight=213];
facilities -> guestrooms  [color="#4daf4a", key=0, label="1/203", weight=203];
facilities -> locationcontacts  [color="#a65628", key=0, label="3/159", weight=159];
offers -> facilities  [color="#e41a1c", key=0, label="0/213", weight=213];
offers -> locationcontacts  [color="#ff7f00", key=0, label="2/162", weight=162];
guestrooms -> dining  [color="#4daf4a", key=0, label="1/203", weight=203];
}
```

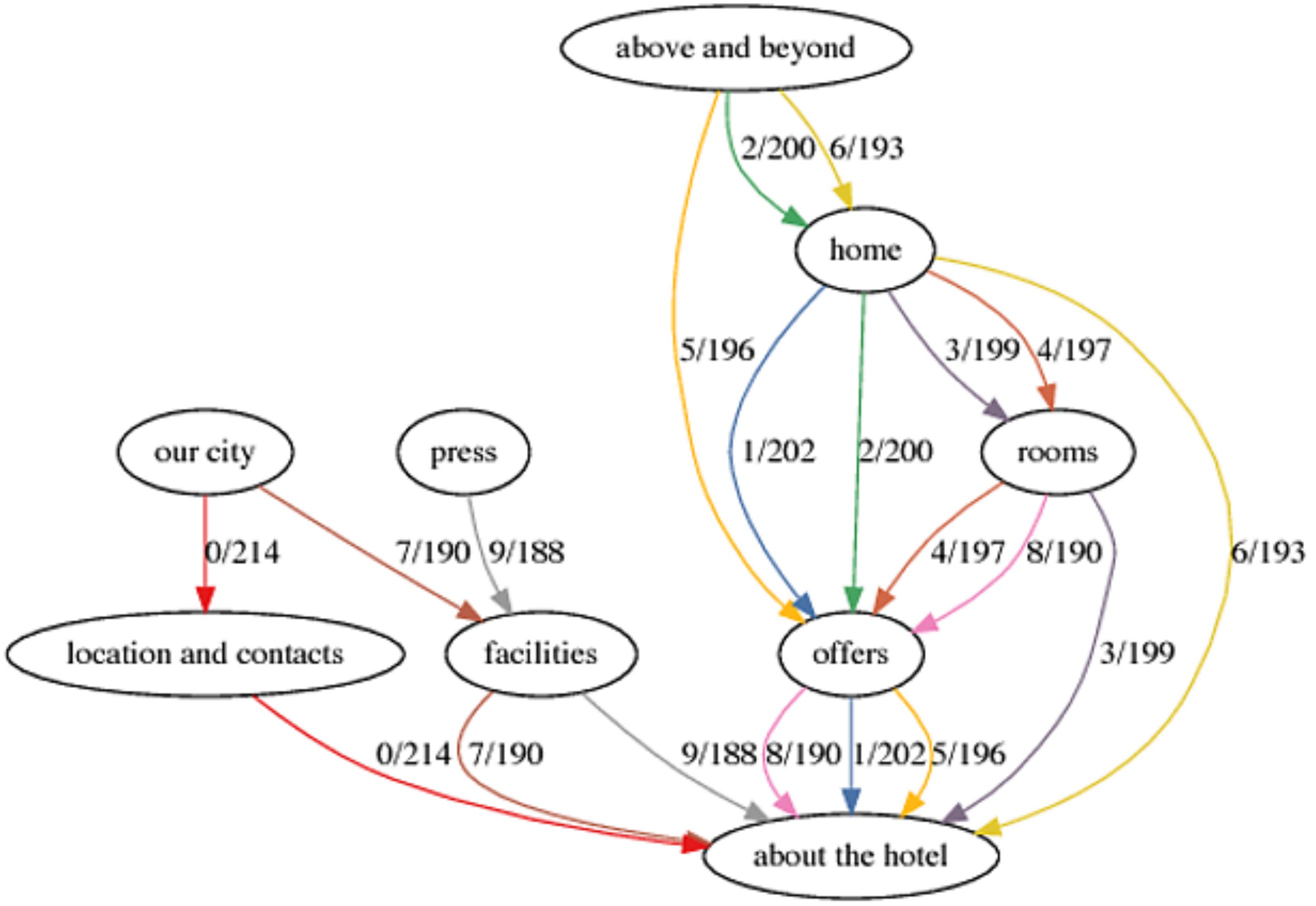
6.2.8. Tablet Requests



```
print graph_to_pydot_string(graph_data["tablet_requests"][0])
```

```
digraph "" {
layout="dot";
"above and beyond";
facilities;
offers;
rooms;
dining;
"above and beyond" -> facilities  [color="#999999", key=0, label="4/162", weight=162];
"above and beyond" -> offers  [color="#a65628", key=0, label="3/162", weight=162];
facilities -> offers  [color="#e41a1c", key=0, label="0/272", weight=272];
facilities -> rooms  [color="#4daf4a", key=0, label="1/197", weight=197];
facilities -> rooms  [color="#ff7f00", key=1, label="2/163", weight=163];
facilities -> rooms  [color="#999999", key=2, label="4/162", weight=162];
offers -> dining  [color="#e41a1c", key=0, label="0/272", weight=272];
offers -> dining  [color="#a65628", key=1, label="3/162", weight=162];
rooms -> dining  [color="#ff7f00", key=0, label="2/163", weight=163];
rooms -> offers  [color="#4daf4a", key=0, label="1/197", weight=197];
}
```

6.2.9. Bot Requests



```
print graph_to_pydot_string(graph_data["bots_requests"][0])
```

```
digraph "" {
layout="dot";
"about the hotel";
"our city";
facilities;
offers;
rooms;
"above and beyond";
press;
home;
"location and contacts";
"our city" -> facilities  [color="#b8604a", key=0, label="7/190", weight=190];
"our city" -> "location and contacts"  [color="#e41a1c", key=0, label="0/214", weight=214];
facilities -> "about the hotel"  [color="#b8604a", key=0, label="7/190", weight=190];
facilities -> "about the hotel"  [color="#999999", key=1, label="9/188", weight=188];
offers -> "about the hotel"  [color="#4a73a7", key=0, label="1/202", weight=202];
offers -> "about the hotel"  [color="#ffb817", key=1, label="5/196", weight=196];
offers -> "about the hotel"  [color="#ed84bb", key=2, label="8/190", weight=190];
rooms -> offers  [color="#d16948", key=0, label="4/197", weight=197];
rooms -> offers  [color="#ed84bb", key=1, label="8/190", weight=190];
rooms -> "about the hotel"  [color="#7f6e85", key=0, label="3/199", weight=199];
"above and beyond" -> home  [color="#48a462", key=0, label="2/200", weight=200];
"above and beyond" -> home  [color="#e1c72f", key=1, label="6/193", weight=193];
"above and beyond" -> offers  [color="#ffb817", key=0, label="5/196", weight=196];
press -> facilities  [color="#999999", key=0, label="9/188", weight=188];
home -> offers  [color="#4a73a7", key=0, label="1/202", weight=202];
home -> offers  [color="#48a462", key=1, label="2/200", weight=200];
home -> rooms  [color="#7f6e85", key=0, label="3/199", weight=199];
home -> rooms  [color="#d16948", key=1, label="4/197", weight=197];
home -> "about the hotel"  [color="#e1c72f", key=0, label="6/193", weight=193];
"location and contacts" -> "about the hotel"  [color="#e41a1c", key=0, label="0/214", weight=214];
}
```

7. CSV Output

We can provide a reference table to support the diagram as a table and import this into our LaTeX report:

```
print "internal_requests.csv"
print graph_data["internal_requests"][1]
print
print "external_requests.csv"
print graph_data["external_requests"][1]
print
print "hk_requests.csv"
print graph_data["hk_requests"][1]
print
print "us_requests.csv"
print graph_data["us_requests"][1]
print
print "au_requests.csv"
print graph_data["au_requests"][1]
print
print "pc_requests.csv"
print graph_data["pc_requests"][1]
print
print "smartphone_requests.csv"
print graph_data["smartphone_requests"][1]
print
print "tablet_requests.csv"
print graph_data["tablet_requests"][1]
print
print "bots_requests.csv"
print graph_data["bots_requests"][1]
```

```
internal_requests.csv
Sequence,From,To,Frequency
0,about the hotel,rooms,123
0,rooms,offers,123
1,about the hotel,dining,122
1,dining,offers,122
2,facilities,about the hotel,103
```

```
2,about the hotel,offers,103
3,rooms,dining,102
3,dining,offers,102
4,facilities,about the hotel,101
4,about the hotel,rooms,101
5,facilities,rooms,99
5,rooms,offers,99
6,facilities,dining,96
6,dining,offers,96
7,about the hotel,rooms,96
7,rooms,dining,96
8,above and beyond,dining,94
8,dining,offers,94

external_requests.csv
Sequence,From,To,Frequency
0,above and beyond,facilities,1823
0,facilities,rooms,1823
1,above and beyond,offers,1760
1,offers,dining,1760
2,facilities,rooms,1732
2,rooms,offers,1732
3,facilities,offers,1727
3,offers,dining,1727
4,facilities,rooms,1715
4,rooms,dining,1715
5,above and beyond,facilities,1684
5,facilities,dining,1684

hk_requests.csv
Sequence,From,To,Frequency
0,above and beyond,offers,662
0,offers,dining,662
1,facilities,offers,596
1,offers,dining,596
```

```
2,about the hotel,offers,465
2,offers,dining,465
3,rooms,offers,376
3,offers,dining,376
4,above and beyond,facilities,362
4,facilities,dining,362
5,above and beyond,about the hotel,344
5,about the hotel,dining,344

us_requests.csv
Sequence,From,To,Frequency
0,rooms,offers,330
0,offers,about the hotel,330
1,above and beyond,rooms,329
1,rooms,about the hotel,329
2,above and beyond,rooms,299
2,rooms,offers,299
3,facilities,rooms,281
3,rooms,about the hotel,281
4,above and beyond,offers,273
4,offers,about the hotel,273
5,above and beyond,facilities,272
5,facilities,rooms,272
6,rooms,dining,268
6,dining,offers,268
7,our city,facilities,260
7,facilities,about the hotel,260

au_requests.csv
Sequence,From,To,Frequency
0,above and beyond,facilities,152
0,facilities,rooms,152
1,above and beyond,rooms,151
1,rooms,offers,151
2,facilities,rooms,141
```

```
2,rooms,offers,141
3,about the hotel,rooms,122
3,rooms,offers,122
4,about the hotel,facilities,120
4,facilities,rooms,120
5,above and beyond,dining,120
5,dining,rooms,120
6,above and beyond,facilities,115
6,facilities,offers,115
7,facilities,dining,107
7,dining,rooms,107
8,above and beyond,facilities,105
8,facilities,dining,105
9,dining,rooms,103
9,rooms,offers,103
10,about the hotel,above and beyond,100
10,above and beyond,facilities,100
11,above and beyond,dining,100
11,dining,offers,100
12,about the hotel,above and beyond,97
12,above and beyond,rooms,97
13,location and contacts,rooms,92
13,rooms,offers,92
14,about the hotel,facilities,88
14,facilities,offers,88
15,about the hotel,dining,85
15,dining,rooms,85

pc_requests.csv
Sequence,From,To,Frequency
0,above and beyond,facilities,349
0,facilities,rooms,349
1,facilities,about the hotel,296
1,about the hotel,rooms,296
2,above and beyond,rooms,293
```

```
2,rooms,dining,293
3,facilities,rooms,288
3,rooms,dining,288
4,above and beyond,rooms,286
4,rooms,offers,286
5,facilities,rooms,286
5,rooms,offers,286
6,above and beyond,about the hotel,280
6,about the hotel,rooms,280
7,above and beyond,facilities,269
7,facilities,dining,269
8,about the hotel,rooms,267
8,rooms,offers,267
9,above and beyond,facilities,246
9,facilities,about the hotel,246
10,about the hotel,rooms,241
10,rooms,dining,241
11,above and beyond,dining,238
11,dining,offers,238
12,above and beyond,about the hotel,231
12,about the hotel,dining,231
13,above and beyond,facilities,230
13,facilities,offers,230
14,rooms,dining,225
14,dining,offers,225
15,facilities,about the hotel,213
15,about the hotel,dining,213
16,above and beyond,facilities,191
16,facilities,rooms,191
16,rooms,dining,191
17,facilities,dining,189
17,dining,offers,189
18,facilities,about the hotel,186
18,about the hotel,offers,186
19,above and beyond,about the hotel,182
```

```
19,about the hotel,offers,182
20,above and beyond,facilities,176
20,facilities,about the hotel,176
20,about the hotel,rooms,176
21,about the hotel,dining,171
21,dining,offers,171

smartphone_requests.csv
Sequence,From,To,Frequency
0,offers,facilities,213
0,facilities,dining,213
1,facilities,guestrooms,203
1,guestrooms,dining,203
2,offers,locationcontacts,162
2,locationcontacts,dining,162
3,facilities,locationcontacts,159
3,locationcontacts,dining,159
4,default,locationcontacts,145
4,locationcontacts,dining,145

tablet_requests.csv
Sequence,From,To,Frequency
0,facilities,offers,272
0,offers,dining,272
1,facilities,rooms,197
1,rooms,offers,197
2,facilities,rooms,163
2,rooms,dining,163
3,above and beyond,offers,162
3,offers,dining,162
4,above and beyond,facilities,162
4,facilities,rooms,162

bots_requests.csv
Sequence,From,To,Frequency
```

```
0,our city,location and contacts,214
0,location and contacts,about the hotel,214
1,home,offers,202
1,offers,about the hotel,202
2,above and beyond,home,200
2,home,offers,200
3,home,rooms,199
3,rooms,about the hotel,199
4,home,rooms,197
4,rooms,offers,197
5,above and beyond,offers,196
5,offers,about the hotel,196
6,above and beyond,home,193
6,home,about the hotel,193
7,our city,facilities,190
7,facilities,about the hotel,190
8,rooms,offers,190
8,offers,about the hotel,190
9,press,facilities,188
9,facilities,about the hotel,188
```

```
internal_requests.dot
digraph "" {
layout="dot";
facilities;
"about the hotel";
dining;
offers;
rooms;
"above and beyond";
facilities -> dining  [color="#a65628", key=0, label="6/96", weight=96];
facilities -> rooms  [color="#ffff33", key=0, label="5/99", weight=99];
facilities -> "about the hotel"  [color="#4daf4a", key=0, label="2/103", weight=103];
facilities -> "about the hotel"  [color="#ff7f00", key=1, label="4/101", weight=101];
"about the hotel" -> dining  [color="#377eb8", key=0, label="1/122", weight=122];
"about the hotel" -> offers  [color="#4daf4a", key=0, label="2/103", weight=103];
"about the hotel" -> rooms  [color="#e41a1c", key=0, label="0/123", weight=123];
"about the hotel" -> rooms  [color="#ff7f00", key=1, label="4/101", weight=101];
"about the hotel" -> rooms  [color="#f781bf", key=2, label="7/96", weight=96];
dining -> offers  [color="#377eb8", key=0, label="1/122", weight=122];
dining -> offers  [color="#984ea3", key=1, label="3/102", weight=102];
dining -> offers  [color="#a65628", key=2, label="6/96", weight=96];
dining -> offers  [color="#999999", key=3, label="8/94", weight=94];
rooms -> dining  [color="#984ea3", key=0, label="3/102", weight=102];
rooms -> dining  [color="#f781bf", key=1, label="7/96", weight=96];
rooms -> offers  [color="#e41a1c", key=0, label="0/123", weight=123];
rooms -> offers  [color="#ffff33", key=1, label="5/99", weight=99];
"above and beyond" -> dining  [color="#999999", key=0, label="8/94", weight=94];
}
```

```
external_requests.dot
digraph "" {
layout="dot";
dining;
"above and beyond";
offers;
rooms;
facilities;
"above and beyond" -> facilities  [color="#e41a1c", key=0, label="0/1823", weight=1823];
"above and beyond" -> facilities  [color="#999999", key=1, label="5/1684", weight=1684];
"above and beyond" -> offers  [color="#449b76", key=0, label="1/1760", weight=1760];
offers -> dining  [color="#449b76", key=0, label="1/1760", weight=1760];
offers -> dining  [color="#ffe529", key=1, label="3/1727", weight=1727];
rooms -> dining  [color="#c66764", key=0, label="4/1715", weight=1715];
rooms -> offers  [color="#ad5882", key=0, label="2/1732", weight=1732];
facilities -> dining  [color="#999999", key=0, label="5/1684", weight=1684];
facilities -> offers  [color="#ffe529", key=0, label="3/1727", weight=1727];
facilities -> rooms  [color="#e41a1c", key=0, label="0/1823", weight=1823];
facilities -> rooms  [color="#ad5882", key=1, label="2/1732", weight=1732];
facilities -> rooms  [color="#c66764", key=2, label="4/1715", weight=1715];
}
```

```
hk_requests.dot
digraph "" {
layout="dot";
dining;
"about the hotel";
facilities;
offers;
rooms;
"above and beyond";
"about the hotel" -> dining  [color="#999999", key=0, label="5/344", weight=344];
"about the hotel" -> offers  [color="#ad5882", key=0, label="2/465", weight=465];
facilities -> dining  [color="#c66764", key=0, label="4/362", weight=362];
facilities -> offers  [color="#449b76", key=0, label="1/596", weight=596];
offers -> dining  [color="#e41a1c", key=0, label="0/662", weight=662];
offers -> dining  [color="#449b76", key=1, label="1/596", weight=596];
offers -> dining  [color="#ad5882", key=2, label="2/465", weight=465];
offers -> dining  [color="#ffe529", key=3, label="3/376", weight=376];
rooms -> offers  [color="#ffe529", key=0, label="3/376", weight=376];
"above and beyond" -> facilities  [color="#c66764", key=0, label="4/362", weight=362];
"above and beyond" -> offers  [color="#e41a1c", key=0, label="0/662", weight=662];
"above and beyond" -> "about the hotel"  [color="#999999", key=0, label="5/344", weight=344];
}
```

```
us_requests.dot
digraph "" {
layout="dot";
facilities;
"about the hotel";
dining;
offers;
rooms;
"our city";
"above and beyond";
facilities -> rooms  [color="#c4635d", key=0, label="3/281", weight=281];
facilities -> rooms  [color="#bf862b", key=1, label="5/272", weight=272];
facilities -> "about the hotel"  [color="#999999", key=0, label="7/260", weight=260];
dining -> offers  [color="#eb7ba9", key=0, label="6/268", weight=268];
offers -> "about the hotel"  [color="#e41a1c", key=0, label="0/330", weight=330];
offers -> "about the hotel"  [color="#ffc81d", key=1, label="4/273", weight=273];
rooms -> dining  [color="#eb7ba9", key=0, label="6/268", weight=268];
rooms -> offers  [color="#e41a1c", key=0, label="0/330", weight=330];
rooms -> offers  [color="#629363", key=1, label="2/299", weight=299];
rooms -> "about the hotel"  [color="#3a85a8", key=0, label="1/329", weight=329];
rooms -> "about the hotel"  [color="#c4635d", key=1, label="3/281", weight=281];
"our city" -> facilities  [color="#999999", key=0, label="7/260", weight=260];
"above and beyond" -> facilities  [color="#bf862b", key=0, label="5/272", weight=272];
"above and beyond" -> offers  [color="#ffc81d", key=0, label="4/273", weight=273];
"above and beyond" -> rooms  [color="#3a85a8", key=0, label="1/329", weight=329];
"above and beyond" -> rooms  [color="#629363", key=1, label="2/299", weight=299];
}
```

```
au_requests.dot
digraph "" {
layout="dot";
facilities;
"about the hotel";
dining;
offers;
rooms;
"above and beyond";
"location and contacts";
facilities -> dining  [color="#e4722b", key=0, label="7/107", weight=107];
facilities -> dining  [color="#ffa10e", key=1, label="8/105", weight=105];
facilities -> offers  [color="#ad5882", key=0, label="6/115", weight=115];
facilities -> offers  [color="#cb8cad", key=1, label="14/88", weight=88];
facilities -> rooms  [color="#e41a1c", key=0, label="0/152", weight=152];
facilities -> rooms  [color="#3881b1", key=1, label="2/141", weight=141];
facilities -> rooms  [color="#57a256", key=2, label="4/120", weight=120];
"about the hotel" -> dining  [color="#999999", key=0, label="15/85", weight=85];
"about the hotel" -> facilities  [color="#57a256", key=0, label="4/120", weight=120];
"about the hotel" -> facilities  [color="#cb8cad", key=1, label="14/88", weight=88];
"about the hotel" -> rooms  [color="#449b76", key=0, label="3/122", weight=122];
"about the hotel" -> "above and beyond"  [color="#e1c72f", key=0, label="10/100", weight=100];
"about the hotel" -> "above and beyond"  [color="#c66764", key=1, label="12/97", weight=97];
dining -> offers  [color="#b26d29", key=0, label="11/100", weight=100];
dining -> rooms  [color="#7f6e85", key=0, label="5/120", weight=120];
dining -> rooms  [color="#e4722b", key=1, label="7/107", weight=107];
dining -> rooms  [color="#ffe529", key=2, label="9/103", weight=103];
dining -> rooms  [color="#999999", key=3, label="15/85", weight=85];
rooms -> offers  [color="#884f6f", key=0, label="1/151", weight=151];
rooms -> offers  [color="#3881b1", key=1, label="2/141", weight=141];
rooms -> offers  [color="#449b76", key=2, label="3/122", weight=122];
rooms -> offers  [color="#ffe529", key=3, label="9/103", weight=103];
rooms -> offers  [color="#f27eb5", key=4, label="13/92", weight=92];
"above and beyond" -> facilities  [color="#e41a1c", key=0, label="0/152", weight=152];
"above and beyond" -> facilities  [color="#ad5882", key=1, label="6/115", weight=115];
"above and beyond" -> facilities  [color="#ffa10e", key=2, label="8/105", weight=105];
"above and beyond" -> facilities  [color="#e1c72f", key=3, label="10/100", weight=100];
"above and beyond" -> rooms  [color="#884f6f", key=0, label="1/151", weight=151];
"above and beyond" -> rooms  [color="#c66764", key=1, label="12/97", weight=97];
"above and beyond" -> dining  [color="#7f6e85", key=0, label="5/120", weight=120];
"above and beyond" -> dining  [color="#b26d29", key=1, label="11/100", weight=100];
"location and contacts" -> rooms  [color="#f27eb5", key=0, label="13/92", weight=92];
}
```

```
pc_requests.dot
digraph "" {
layout="dot";
```

```
facilities;
"about the hotel";
dining;
offers;
rooms;
"above and beyond";
facilities -> dining  [color="#7f6e85", key=0, label="7/269", weight=269];
facilities -> dining  [color="#cd6a70", key=1, label="17/189", weight=189];
facilities -> offers  [color="#fff931", key=0, label="13/230", weight=230];
facilities -> rooms  [color="#e41a1c", key=0, label="0/349", weight=349];
facilities -> rooms  [color="#3a85a8", key=1, label="3/288", weight=288];
facilities -> rooms  [color="#4baa54", key=2, label="5/286", weight=286];
facilities -> rooms  [color="#ae5a36", key=3, label="16/191", weight=191];
facilities -> "about the hotel"  [color="#a24057", key=0, label="1/296", weight=296];
facilities -> "about the hotel"  [color="#c4635d", key=1, label="9/246", weight=246];
facilities -> "about the hotel"  [color="#bf862b", key=2, label="15/213", weight=213];
facilities -> "about the hotel"  [color="#eb7ba9", key=3, label="18/186", weight=186];
facilities -> "about the hotel"  [color="#bd90a7", key=4, label="20/176", weight=176];
"about the hotel" -> dining  [color="#ffc81d", key=0, label="12/231", weight=231];
"about the hotel" -> dining  [color="#bf862b", key=1, label="15/213", weight=213];
"about the hotel" -> dining  [color="#999999", key=2, label="21/171", weight=171];
"about the hotel" -> offers  [color="#eb7ba9", key=0, label="18/186", weight=186];
"about the hotel" -> offers  [color="#e187b6", key=1, label="19/182", weight=182];
"about the hotel" -> rooms  [color="#a24057", key=0, label="1/296", weight=296];
"about the hotel" -> rooms  [color="#629363", key=1, label="6/280", weight=280];
"about the hotel" -> rooms  [color="#9d509b", key=2, label="8/267", weight=267];
"about the hotel" -> rooms  [color="#eb761f", key=3, label="10/241", weight=241];
"about the hotel" -> rooms  [color="#bd90a7", key=4, label="20/176", weight=176];
dining -> offers  [color="#ff970a", key=0, label="11/238", weight=238];
dining -> offers  [color="#e1c72f", key=1, label="14/225", weight=225];
dining -> offers  [color="#cd6a70", key=2, label="17/189", weight=189];
dining -> offers  [color="#999999", key=3, label="21/171", weight=171];
rooms -> dining  [color="#606693", key=0, label="2/293", weight=293];
rooms -> dining  [color="#3a85a8", key=1, label="3/288", weight=288];
rooms -> dining  [color="#eb761f", key=2, label="10/241", weight=241];
rooms -> dining  [color="#e1c72f", key=3, label="14/225", weight=225];
rooms -> dining  [color="#ae5a36", key=4, label="16/191", weight=191];
rooms -> offers  [color="#43987e", key=0, label="4/286", weight=286];
rooms -> offers  [color="#4baa54", key=1, label="5/286", weight=286];
rooms -> offers  [color="#9d509b", key=2, label="8/267", weight=267];
"above and beyond" -> dining  [color="#ff970a", key=0, label="11/238", weight=238];
"above and beyond" -> facilities  [color="#e41a1c", key=0, label="0/349", weight=349];
"above and beyond" -> facilities  [color="#7f6e85", key=1, label="7/269", weight=269];
"above and beyond" -> facilities  [color="#c4635d", key=2, label="9/246", weight=246];
"above and beyond" -> facilities  [color="#fff931", key=3, label="13/230", weight=230];
"above and beyond" -> facilities  [color="#ae5a36", key=4, label="16/191", weight=191];
"above and beyond" -> facilities  [color="#bd90a7", key=5, label="20/176", weight=176];
"above and beyond" -> rooms  [color="#606693", key=0, label="2/293", weight=293];
"above and beyond" -> rooms  [color="#43987e", key=1, label="4/286", weight=286];
"above and beyond" -> "about the hotel"  [color="#629363", key=0, label="6/280", weight=280];
"above and beyond" -> "about the hotel"  [color="#ffc81d", key=1, label="12/231", weight=231];
"above and beyond" -> "about the hotel"  [color="#e187b6", key=2, label="19/182", weight=182];
}
```

```
smartphone_requests.dot
digraph "" {
layout="dot";
locationcontacts;
dining;
default;
facilities;
offers;
guestrooms;
locationcontacts -> dining  [color="#ff7f00", key=0, label="2/162", weight=162];
locationcontacts -> dining  [color="#a65628", key=1, label="3/159", weight=159];
locationcontacts -> dining  [color="#999999", key=2, label="4/145", weight=145];
default -> locationcontacts  [color="#999999", key=0, label="4/145", weight=145];
facilities -> dining  [color="#e41a1c", key=0, label="0/213", weight=213];
facilities -> guestrooms  [color="#4daf4a", key=0, label="1/203", weight=203];
facilities -> locationcontacts  [color="#a65628", key=0, label="3/159", weight=159];
offers -> facilities  [color="#e41a1c", key=0, label="0/213", weight=213];
offers -> locationcontacts  [color="#ff7f00", key=0, label="2/162", weight=162];
guestrooms -> dining  [color="#4daf4a", key=0, label="1/203", weight=203];
}
```

```
tablet_requests.dot
digraph "" {
layout="dot";
"above and beyond";
facilities;
offers;
rooms;
dining;
"above and beyond" -> facilities  [color="#999999", key=0, label="4/162", weight=162];
"above and beyond" -> offers  [color="#a65628", key=0, label="3/162", weight=162];
facilities -> offers  [color="#e41a1c", key=0, label="0/272", weight=272];
facilities -> rooms  [color="#4daf4a", key=0, label="1/197", weight=197];
facilities -> rooms  [color="#ff7f00", key=1, label="2/163", weight=163];
facilities -> rooms  [color="#999999", key=2, label="4/162", weight=162];
offers -> dining  [color="#e41a1c", key=0, label="0/272", weight=272];
offers -> dining  [color="#a65628", key=1, label="3/162", weight=162];
rooms -> dining  [color="#ff7f00", key=0, label="2/163", weight=163];
rooms -> offers  [color="#4daf4a", key=0, label="1/197", weight=197];
}
```

```
bots_requests.dot
digraph "" {
layout="dot";
"about the hotel";
"our city";
facilities;
offers;
rooms;
"above and beyond";
press;
home;
"location and contacts";
"our city" -> facilities  [color="#b8604a", key=0, label="7/190", weight=190];
"our city" -> "location and contacts"  [color="#e41a1c", key=0, label="0/214", weight=214];
facilities -> "about the hotel"  [color="#b8604a", key=0, label="7/190", weight=190];
facilities -> "about the hotel"  [color="#999999", key=1, label="9/188", weight=188];
offers -> "about the hotel"  [color="#4a73a7", key=0, label="1/202", weight=202];
offers -> "about the hotel"  [color="#ffb817", key=1, label="5/196", weight=196];
offers -> "about the hotel"  [color="#ed84bb", key=2, label="8/190", weight=190];
rooms -> offers  [color="#d16948", key=0, label="4/197", weight=197];
rooms -> offers  [color="#ed84bb", key=1, label="8/190", weight=190];
rooms -> "about the hotel"  [color="#7f6e85", key=0, label="3/199", weight=199];
"above and beyond" -> home  [color="#48a462", key=0, label="2/200", weight=200];
"above and beyond" -> home  [color="#e1c72f", key=1, label="6/193", weight=193];
"above and beyond" -> offers  [color="#ffb817", key=0, label="5/196", weight=196];
press -> facilities  [color="#999999", key=0, label="9/188", weight=188];
home -> offers  [color="#4a73a7", key=0, label="1/202", weight=202];
home -> offers  [color="#48a462", key=1, label="2/200", weight=200];
home -> rooms  [color="#7f6e85", key=0, label="3/199", weight=199];
home -> rooms  [color="#d16948", key=1, label="4/197", weight=197];
home -> "about the hotel"  [color="#e1c72f", key=0, label="6/193", weight=193];
"location and contacts" -> "about the hotel"  [color="#e41a1c", key=0, label="0/214", weight=214];
}
```