

# A Reproducibility Analysis: Learning Points and Routes to Recommend Trajectories

188.992 Experiment Design for Data Science - Exercise 2

Group 02

Yanqi Jia  
TU Wien  
Vienna, Austria  
e11770915@student.tuwien.ac.at

Stefan Pazourek  
TU Wien  
Vienna, Austria  
e1621455@student.tuwien.ac.at

Marina Sommer  
TU Wien  
Vienna, Austria  
e11778902@student.tuwien.ac.at

## Abstract

In this project, we want to reproduce the results of the paper "*Learning Points and Routes to Recommend Trajectories*" written by Dawei Chen, Cheng Soon Ong and Lexing Xie.<sup>1</sup> In addition, we are asked to deal with the experimental setup used in the paper and to answer the following questions:

- Is the workflow well documented?
- Is a program code available? If yes, is it running out of the box? If not, what is needed to be changed?
- Do we get the same results as in the paper?
- Does the usage of different seeds affect the results?

**Keywords:** Reproducibility; reproducing results; trajectory recommendation; learning to rank; planning

## ACM Reference Format:

Yanqi Jia, Stefan Pazourek, and Marina Sommer. 2022. A Reproducibility Analysis: Learning Points and Routes to Recommend Trajectories: 188.992 Experiment Design for Data Science - Exercise 2 Group 02. In *Proceedings of ACM Conference (Conference'17)*. ACM, New York, NY, USA, 6 pages. <https://doi.org/10.1145/nnnnnnn.nnnnnn>

## 1 Introduction

The paper "*Learning Points and Routes to Recommend Trajectories*" describes and compares several methods to optimize travel routes in different cities, namely, Edinburgh, Glasgow, Melbourne, Osaka and Toronto. The data sets used for the spots to visit, called points of interest (POIs), are from Flickr and Lim. They used nine different methods of route planning and the external implementation rankSVM for ranking the POIs. In addition, an Integer Linear Program (ILP) is part of the experiment, where presumably the cbc solver was used.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

© 2022 Association for Computing Machinery.

The setup chosen in the paper used one single random seed 1234567890 and the authors did not give any insights how often they have repeated their experiment. In the next sections, we present at first our strategies, followed by what is needed to be done to re-run the experiment. Unfortunately, the code did not run out of the box. Secondly, we present our results with the same seed as in the paper and with an additional one. For one city, Osaka, we provide results of seven different executions. Furthermore, a statistical test is provided to compare the our results and the results of the original paper.

We were using 5.15.12-1-MANJARO, 20.04.1-Ubuntu, MS Windows 10, coin-cbc 2.10.3 and 2.10.5, and Python 3.8.10, 3.9.7 and 3.10.1. All Python packages were at the newest state on 2022-01-30.

## 2 Strategies

After a first look into the paper, we had two main strategies: First of all, we wanted to run the provided IPython notebooks in order to reproduce the two tables displayed in the paper, see figure 1. As a second step, we could reproduce the results with different seeds and afterwards, test for statistical significance with an appropriate test statistic, see section 6. Besides that, we wanted to focus on the experimental setup of the original implementation.

## 3 Steps to Run Code

### 3.1 Clone Repository

Kindly, a Bitbucket repository was published by the authors of the original paper (see <https://bitbucket.org/d-chen/tour-cikm16/src/master/>). It contains the code, the training data sets to re-run the experiment and a README file, where the workflow of the paper is shortly documented. The main file is called `rank_markov.ipynb`. Here, it is necessary to manually specify in the code (with choosing a specific index) for the city the scores should be calculated. The experiment consists of five different cities (Osaka, Glasgow, Edinburgh, Toronto and Melbourne), therefore we need to execute the Jupyter Notebook five times. The intermediate results are

stored as pickle files in a data folder. From there, the file `parse_results.ipynb` merges all intermediate results together and creates two  $\LaTeX$  tables. One shows the results using the  $F_1$  score as performance metric and the other one displays the results of the newly introduced pairs- $F_1$  score.

### 3.2 Install rankSVM

Before we can start executing the provided Jupyter Notebook, we have to install an implementation called Large-scale rankSVM. The authors of the paper pointed to the exact implementation which they have used for their work (see [https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/#large\\_scale\\_ranksvm](https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/#large_scale_ranksvm)). There are two different packages available, one for linear and one for kernel cases, namely LIBLINEAR and LIBSVM. According to the paper, both versions are suitable for running the experiment. We decide to choose the first option (LIBLINEAR) since we can adopt the code as it stands, regarding the argument `useLinear=True` used in diverse functions. After downloading the right zip file, the included C files have to be compiled in a terminal.

### 3.3 Install ILP

In addition to the rankSVM implementation, we have to install an Integer Linear Program (ILP) solver. In the paper, only two things are mentioned concerning a solver. On the one hand, that the solver should be available in the path of the working directory. On the other hand, there is a single code line, `USE_GUROBI = False` # whether to use GUROBI as ILP solver, from which we can only gain the information that the GUROBI solver was not used. This is the first important information which is missing in the paper, since in our opinion, it could make a difference regarding the results when using another solver. In the end, we decided to use the cbc (Coin-or branch and cut) solver, which is an open-source MILP solver written in C++. It is available in the following GitHub repository: <https://github.com/coin-or/Cbc>.

### 3.4 Edit Python Code

Unfortunately, the code was not running out of the box. It was necessary to change several parts. The main issues were outdated functions, caused by old python versions/packages. It costed a lot of time to find out that we had to compile the C files included in the rankSVM folder in order to link this implementation with our IPython notebook. As already mentioned, there was not clearly stated that we have to install a ILP solver. After a lot of debugging and googeling, we could solve these issues. We also had to reduce some warnings, caused by implicit typecasts and outdated numpy datatypes. All issues are described in detail below.

**3.4.1 Cython.** With the package cython it is possible to execute C code via Python. In the file `rank_markov.ipynb`, the float division is used wrongly with a single `/`. We changed it to `//`.

**3.4.2 scipy.special.** In `rank_markov.ipynb` an old package version is used. To use the function `comb`, we changed `scipy.misc` to `scipy.special`

**3.4.3 Typecast.** Because of an old numpy coding style, several implicate typecast warnings appeared. Therefore, we had to change some variable types, e.g. `np.bool` to `bool` as a datatype argument.

**3.4.4 set\_value/get\_value to at.** Because of an old python package, we needed to change the methods `set_value` and `get_value` to `at` from the pandas package.

**3.4.5 Link external modules.** We had to set our link to the installed rankSVM module.

## 4 Encountered Difficulties

### 4.1 Operating Systems - MS Windows vs. Linux

In the paper, neither the platform, where the experiments were executed, e.g. OS, nor the applied Python version were mentioned. We have already learned in the lecture about reproducibility that different operating systems do make a difference when it comes to the task of reproducibility. We have experienced exactly the same in our project. A main problem was MS Windows. Here, we were not able to run the code at all due to problems with linking the compiled C files from the rankSVM implementation. Cryptic error messages led us to the conclusion that the notebook could not find these files.

Since one of us uses a notebook with Linux, we decided to use this operating system. But here we experienced RAM limitations with the higher-dimensional data sets. More details on this issue follow in the next subsection.

Finally, we managed to run the code for four out of five cities in a virtual machine using Ubuntu. More precisely, the virtual machine is called Oracle VM VirtualBox. It was very interesting to see that the same virtual machine on another notebook showed the same issues as the notebook using Linux directly as operating system (execution failed after a certain time). Nevertheless, we were not able to reproduce the results for Melbourne after trying it several times since the procedure crashed after approximately 52 hours.

### 4.2 RAM

Using 5.15.12-1-MANJARO with cbc-solver 2.10.5 with 16GB led by some cities after some hours of computation into Out of RAM errors. At VirtualBox 5.13.0-27-generic 20.04.1-Ubuntu with cbc-solver 2.10.3, which is also restricted to 16GB RAM, we had no problems. Afterwards, we would guess that the MILP solver cbc caused these failures, because the execution broke every time at this point in the procedure.

### 4.3 Runtime

The runtimes of the execution of the data sets differ significantly from each other, which can be seen in the table below. That is caused by the diverse dimensions of the data sets and the convergence of the `find_ILP` function. Only one city, namely Osaka, needed less than one hour. That is why we produced several samples only for this city to carry out significance testing. In the end, we reproduced the results for every city except Melbourne two times, once with the provided seed by the authors of the paper and once with another seed. Due to the very long runtime of Melbourne of over 50 hours, we gave up on trying to get it through.

**Table 1.** Approximate Runtimes of Cities

Osaka	~ 30 minutes
Glasgow	~ 60-90 minutes
Edinburgh	~ 27 hours
Toronto	~ 9 hours
Melbourne	failed after ~ 52 hours

## 5 Results

The regarding tables are displayed at the end of the report due to appropriate placement reasons.

### 5.1 Paper

The original results can be seen in Figure 1.

### 5.2 Seed 1234567890

We were able to reproduce similar scores as illustrated in the paper. See Table 2 and Table 3.

### 5.3 Seed 1289365212

The results of seed 1289365212 are very similar to the other ones. See Table 4 and Table 5.

## 6 Test of Significance

To test if our reproduced results statistically significantly differ from the results provided in the original paper, we run the code for Osaka, one of the five cities, with seven different seeds and for Edinburgh, Glasgow and Toronto with two different seeds and obtain the corresponding intervals for  $F_1$  and pairs- $F_1$  score.

To identify the statistically significant differences between the results, it is not always enough to only consider the overlap of the confidence intervals. It is true that not overlapping intervals result in significant difference. However, the results could still statistically significantly differ from each other when the intervals overlap. Therefore, we perform the Cohen's D test which does not only test the difference, but also

measures the size of the difference.

The test statistic is defined as

$$D = \frac{M_1 - M_2}{\sqrt{\frac{S_1^2 + S_2^2}{2}}},$$

where  $M_1$  and  $M_2$  are the sample means and  $S_1$  and  $S_2$  stand for the standard deviations. It uses the standard deviation as a unit to measure the difference in the means. This formula is a simplified version without the sample size, as both our reproduced results and those from the original paper are obtained by leave-one-out cross validation and therefore, have the same sample size. The size of the difference is measured in this way that  $|D| = 0.20$  represents small difference,  $|D| = 0.50$  medium difference and  $|D| = 0.80$  large difference.<sup>2</sup>

The test results are summarized in Table 6 and Table 7. Among all we only have 11 values with absolute value above 0.10 and 3 above 0.15, all others below 0.10. Therefore we conclude that there is no to very small difference between the our reproduced results and those provided in the paper.

## 7 Conclusion

Finally, we were able to reproduce the results very well. As described above, we were running low of resources to reproduce the results for the fifth data set, the city Melbourne. On the whole, we would rate the degree of difficulty of reproducibility with the mark Good. The data sets and IPython notebooks are open to the public and easy to find. We got very similar results in comparison to the authors' outcomes. We would have been happier if some descriptions were added in more detail, e.g. which external tools are necessary and how to install them, in particular for the ILP solver. However, with some knowledge and research skills, it was possible to re-run the experiments. Obviously, it costs a lot of time to keep coding up to date. To find changes in package versions, e.g. outsourcing of functions, is very tedious, therefore, a list of all package versions should be compulsory. Unfortunately, something like that was missing in the original paper, but with reverse engineering of the problems through error messages, we could fix them in the end. Annoying is the fact that we were not able to run the experiment on MS Windows based machines. That is why we had to make use of a virtual machine.

## References

- <sup>1</sup> Dawei Chen, Cheng Soon Ong, and Lexing Xie. Learning points and routes to recommend trajectories. *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, Oct 2016.
- <sup>2</sup> Ruben Geert van den Berg. Cohen's d – effect size for t-test. <https://www.spss-tutorials.com/cohens-d/>.

## 8 Appendix

Figure 1. Results of the original paper.

Table 3: Performance comparison on five datasets in terms of  $F_1$  score. The best method for each dataset (i.e., a column) is shown in bold, the second best is shown in *italic*.

	Edinburgh	Glasgow	Melbourne	Osaka	Toronto
RANDOM	$0.570 \pm 0.139$	$0.632 \pm 0.123$	$0.558 \pm 0.149$	$0.621 \pm 0.115$	$0.621 \pm 0.129$
PERS TOUR[14]	$0.656 \pm 0.223$	<b><math>0.801 \pm 0.213</math></b>	$0.483 \pm 0.208$	$0.686 \pm 0.231$	$0.720 \pm 0.215$
PERS TOUR-L	$0.651 \pm 0.143$	$0.660 \pm 0.102$	$0.576 \pm 0.141$	$0.686 \pm 0.137$	$0.643 \pm 0.113$
POI POPULARITY	<b><math>0.701 \pm 0.160</math></b>	$0.745 \pm 0.166$	$0.620 \pm 0.136$	$0.663 \pm 0.125$	$0.678 \pm 0.121$
POI RANK	<i><math>0.700 \pm 0.155</math></i>	<i><math>0.768 \pm 0.171</math></i>	<i><math>0.637 \pm 0.142</math></i>	<b><math>0.745 \pm 0.173</math></b>	<b><math>0.754 \pm 0.170</math></b>
MARKOV	$0.645 \pm 0.169$	$0.725 \pm 0.167$	$0.577 \pm 0.168$	$0.697 \pm 0.150$	$0.669 \pm 0.151$
MARKOV PATH	$0.678 \pm 0.149$	$0.732 \pm 0.168$	$0.595 \pm 0.148$	$0.706 \pm 0.150$	$0.688 \pm 0.138$
RANK+MARKOV	$0.659 \pm 0.174$	$0.754 \pm 0.173$	$0.613 \pm 0.166$	$0.715 \pm 0.164$	$0.723 \pm 0.185$
RANK+MARKOV PATH	$0.697 \pm 0.152$	$0.762 \pm 0.167$	<b><math>0.639 \pm 0.146</math></b>	<i><math>0.732 \pm 0.162</math></i>	<i><math>0.751 \pm 0.170</math></i>

Table 4: Performance comparison on five datasets in terms of pairs- $F_1$  score. The best method for each dataset (i.e., a column) is shown in bold, the second best is shown in *italic*.

	Edinburgh	Glasgow	Melbourne	Osaka	Toronto
RANDOM	$0.261 \pm 0.155$	$0.320 \pm 0.168$	$0.248 \pm 0.147$	$0.304 \pm 0.142$	$0.310 \pm 0.167$
PERS TOUR[14]	$0.417 \pm 0.343$	<b><math>0.643 \pm 0.366</math></b>	$0.216 \pm 0.265$	$0.468 \pm 0.376$	$0.504 \pm 0.354$
PERS TOUR-L	$0.359 \pm 0.207$	$0.352 \pm 0.162$	$0.266 \pm 0.140$	$0.406 \pm 0.238$	$0.333 \pm 0.163$
POI POPULARITY	<i><math>0.436 \pm 0.259</math></i>	$0.507 \pm 0.298$	$0.316 \pm 0.178$	$0.365 \pm 0.190$	$0.384 \pm 0.201$
POI RANK	$0.432 \pm 0.251$	<i><math>0.548 \pm 0.311</math></i>	$0.339 \pm 0.203$	<b><math>0.511 \pm 0.309</math></b>	<b><math>0.518 \pm 0.296</math></b>
MARKOV	$0.417 \pm 0.248$	$0.495 \pm 0.296$	$0.288 \pm 0.195$	$0.445 \pm 0.266$	$0.407 \pm 0.241$
MARKOV PATH	$0.400 \pm 0.235$	$0.485 \pm 0.293$	$0.294 \pm 0.187$	$0.442 \pm 0.260$	$0.405 \pm 0.231$
RANK+MARKOV	<b><math>0.444 \pm 0.263</math></b>	$0.545 \pm 0.306$	<b><math>0.351 \pm 0.220</math></b>	$0.486 \pm 0.288$	$0.512 \pm 0.303$
RANK+MARKOV PATH	$0.428 \pm 0.245$	$0.533 \pm 0.303$	<i><math>0.344 \pm 0.206</math></i>	<i><math>0.489 \pm 0.287</math></i>	<i><math>0.514 \pm 0.297</math></i>

Table 2. Performance comparison on five datasets in terms of  $F_1$  scores. The best method for each dataset is shown in bold, the second best is shown in *italic*.

	Edinburgh	Glasgow	Osaka	Toronto
RANDOM	$0.580 \pm 0.127$	$0.632 \pm 0.108$	$0.618 \pm 0.129$	$0.605 \pm 0.118$
PERS TOUR	$0.656 \pm 0.223$	<b><math>0.801 \pm 0.213</math></b>	$0.686 \pm 0.231$	$0.720 \pm 0.215$
PERS TOUR-L	$0.651 \pm 0.143$	$0.660 \pm 0.102$	$0.686 \pm 0.137$	$0.643 \pm 0.113$
POI POPULARITY	<b><math>0.701 \pm 0.160</math></b>	$0.745 \pm 0.166$	$0.663 \pm 0.125$	$0.678 \pm 0.121$
POI RANK	<i><math>0.699 \pm 0.153</math></i>	<i><math>0.770 \pm 0.174</math></i>	<b><math>0.745 \pm 0.173</math></b>	<b><math>0.753 \pm 0.169</math></b>
MARKOV	$0.645 \pm 0.169$	$0.725 \pm 0.167$	$0.697 \pm 0.150$	$0.669 \pm 0.151$
MARKOV PATH	$0.678 \pm 0.150$	$0.735 \pm 0.170$	$0.706 \pm 0.150$	$0.689 \pm 0.138$
RANK+MARKOV	$0.662 \pm 0.174$	$0.754 \pm 0.173$	$0.715 \pm 0.164$	$0.722 \pm 0.184$
RANK+MARKOV PATH	$0.698 \pm 0.152$	$0.762 \pm 0.167$	<i><math>0.737 \pm 0.166</math></i>	<i><math>0.750 \pm 0.169</math></i>

**Table 3.** Performance comparison on five datasets in terms of pairs-F<sub>1</sub> scores. The best method for each dataset is shown in **bold**, the second best is shown in *italic*.

	Edinburgh	Glasgow	Osaka	Toronto
RANDOM	0.266 ± 0.136	0.316 ± 0.144	0.305 ± 0.172	0.286 ± 0.132
PERS TOUR	0.417 ± 0.343	<b>0.643 ± 0.366</b>	0.468 ± 0.376	0.504 ± 0.354
PERS TOUR-L	0.359 ± 0.207	0.352 ± 0.162	0.406 ± 0.238	0.333 ± 0.163
POI POPULARITY	<i>0.436 ± 0.259</i>	0.507 ± 0.298	0.365 ± 0.190	0.384 ± 0.201
POI RANK	0.431 ± 0.249	<i>0.552 ± 0.313</i>	<b>0.511 ± 0.309</b>	<b>0.517 ± 0.295</b>
MARKOV	0.417 ± 0.248	0.495 ± 0.296	0.445 ± 0.266	0.407 ± 0.241
MARKOV PATH	0.401 ± 0.235	0.491 ± 0.297	0.442 ± 0.260	0.405 ± 0.231
RANK+MARKOV	<b>0.448 ± 0.264</b>	0.546 ± 0.305	0.486 ± 0.288	0.511 ± 0.302
RANK+MARKOV PATH	0.429 ± 0.247	0.534 ± 0.303	<i>0.496 ± 0.293</i>	<i>0.513 ± 0.297</i>

**Table 4.** Performance comparison on five datasets in terms of F<sub>1</sub> scores. The best method for each dataset is shown in **bold**, the second best is shown in *italic*.

	Edinburgh	Glasgow	Osaka	Toronto
RANDOM	0.581 ± 0.136	0.629 ± 0.115	0.618 ± 0.129	0.610 ± 0.118
PERS TOUR	0.656 ± 0.223	<b>0.801 ± 0.213</b>	0.686 ± 0.231	0.720 ± 0.215
PERS TOUR-L	0.651 ± 0.143	0.660 ± 0.102	0.686 ± 0.137	0.643 ± 0.113
POI POPULARITY	<b>0.701 ± 0.160</b>	0.745 ± 0.166	0.663 ± 0.125	0.678 ± 0.121
POI RANK	<i>0.699 ± 0.153</i>	<i>0.770 ± 0.174</i>	<b>0.745 ± 0.173</b>	<b>0.753 ± 0.169</b>
MARKOV	0.645 ± 0.169	0.725 ± 0.167	0.697 ± 0.150	0.669 ± 0.151
MARKOV PATH	0.678 ± 0.150	0.735 ± 0.170	0.706 ± 0.150	0.689 ± 0.138
RANK+MARKOV	0.662 ± 0.174	0.742 ± 0.171	0.709 ± 0.161	0.700 ± 0.173
RANK+MARKOV PATH	0.696 ± 0.154	0.750 ± 0.166	<i>0.717 ± 0.160</i>	<i>0.726 ± 0.160</i>

**Table 5.** Performance comparison on five datasets in terms of pairs-F<sub>1</sub> scores. The best method for each dataset is shown in **bold**, the second best is shown in *italic*.

	Edinburgh	Glasgow	Osaka	Toronto
RANDOM	0.270 ± 0.159	0.315 ± 0.158	0.305 ± 0.172	0.292 ± 0.140
PERS TOUR	0.417 ± 0.343	<b>0.643 ± 0.366</b>	<i>0.468 ± 0.376</i>	<i>0.504 ± 0.354</i>
PERS TOUR-L	0.359 ± 0.207	0.352 ± 0.162	0.406 ± 0.238	0.333 ± 0.163
POI POPULARITY	<i>0.436 ± 0.259</i>	0.507 ± 0.298	0.365 ± 0.190	0.384 ± 0.201
POI RANK	0.431 ± 0.249	<i>0.552 ± 0.313</i>	<b>0.511 ± 0.309</b>	<b>0.517 ± 0.295</b>
MARKOV	0.417 ± 0.248	0.495 ± 0.296	0.445 ± 0.266	0.407 ± 0.241
MARKOV PATH	0.401 ± 0.235	0.491 ± 0.297	0.442 ± 0.260	0.405 ± 0.231
RANK+MARKOV	<b>0.450 ± 0.264</b>	0.525 ± 0.303	0.467 ± 0.287	0.467 ± 0.281
RANK+MARKOV PATH	0.428 ± 0.250	0.515 ± 0.299	0.465 ± 0.282	0.472 ± 0.275

**Table 6.** Test results on five datasets with different seeds in terms of  $F_1$  scores

City	Seeds	Random	Rank+Markov	Rank+Markovpath	PersTour	PersTour-L	PoiPopularity	PoiRank	Markov	MarkovPath
Osaka	234	-0.043	-0.063	-0.063	0.000	0.000	0.000	0.000	0.000	0.000
Osaka	1503	0.029	-0.094	-0.120	0.000	0.000	0.000	0.000	0.000	0.000
Osaka	2409	0.055	0.000	0.030	0.000	0.000	0.000	0.000	0.000	0.000
Osaka	75738	0.123	0.095	0.072	0.000	0.000	0.000	0.000	0.000	0.000
Osaka	1621455	-0.018	0.048	0.018	0.000	0.000	0.000	0.000	0.000	0.000
Osaka	1234567890	-0.025	0.000	0.030	0.000	0.000	0.000	0.000	0.000	0.000
Osaka	1289365212	-0.025	-0.037	-0.093	0.000	0.000	0.000	0.000	0.000	0.000
Edinburgh	1234567890	0.075	0.017	0.007	0.000	0.000	0.000	-0.006	0.000	0.000
Edinburgh	1289365212	0.080	0.017	-0.007	0.000	0.000	0.000	-0.006	0.000	0.000
Glasgow	1234567890	0.000	0.000	0.000	0.000	0.000	0.000	0.012	0.000	0.018
Glasgow	1289365212	-0.025	-0.070	-0.072	0.000	0.000	0.000	0.012	0.000	0.018
Toronto	1234567890	-0.129	-0.005	-0.006	0.000	0.000	0.000	-0.006	0.000	0.007
Toronto	1289365212	-0.089	-0.128	-0.151	0.000	0.000	0.000	-0.006	0.000	0.007

**Table 7.** Test results on five datasets with different seeds in terms of pairs- $F_1$  scores

City	Seeds	Random	Rank+Markov	Rank+Markovpath	PersTour	PersTour-L	PoiPopularity	PoiRank	Markov	MarkovPath
Osaka	234	-0.043	-0.078	-0.068	0.000	0.000	0.000	0.000	0.000	0.000
Osaka	1503	-0.042	-0.103	-0.118	0.000	0.000	0.000	0.000	0.000	0.000
Osaka	2409	0.099	0.000	0.024	0.000	0.000	0.000	0.000	0.000	0.000
Osaka	75738	0.148	0.084	0.078	0.000	0.000	0.000	0.000	0.000	0.000
Osaka	1621455	-0.065	0.034	0.034	0.000	0.000	0.000	0.000	0.000	0.000
Osaka	1234567890	0.006	0.000	0.024	0.000	0.000	0.000	0.000	0.000	0.000
Osaka	1289365212	0.006	-0.066	-0.084	0.000	0.000	0.000	0.000	0.000	0.000
Edinburgh	1234567890	0.034	0.015	0.004	0.000	0.000	0.000	-0.004	0.000	0.004
Edinburgh	1289365212	0.057	0.023	0.000	0.000	0.000	0.000	-0.004	0.000	0.004
Glasgow	1234567890	-0.026	0.003	0.003	0.000	0.000	0.000	0.013	0.000	0.020
Glasgow	1289365212	-0.031	-0.066	-0.060	0.000	0.000	0.000	0.013	0.000	0.020
Toronto	1234567890	-0.159	-0.003	-0.003	0.000	0.000	0.000	-0.003	0.000	0.000
Toronto	1289365212	-0.117	-0.154	-0.147	0.000	0.000	0.000	-0.003	0.000	0.000