

[🏠](#) / [cookbook](#) / [entity-resolution-1](#)

entity resolution

#CrossSectionalReporting #DataWrangling #ConcentrationIndices #OnlineDatabases #SQL #MSSQL #RecordLinkage #FuzzyMatching #StringSimilarity #KNIME #Python
#ElasticSearch #SPARQL #RDF #Wikidata #ConfusionMatrix

Summary

We want to prepare a data-set to support a research like the ones detailed in these papers:

- <https://www.researchgate.net/publication/354353499> [🔗](#)
How_does_urbanization_affect_the_human_development_index - A cross-country analysis
- <https://www.nature.com/articles/s42949-021-00040-y> [🔗](#)
Urbanization can help reduce income inequality
- <https://www.researchgate.net/publication/264637431> [🔗](#)
The Global Pattern of Urbanization and Economic Growth Evidence from the Last Three Decades
- <https://www.adruk.org/our-work/browse-all-projects/data-first> [🔗](#)
ADR UK: Harnessing the potential of linked administrative data for the justice system
- other JEL O15 / O18 / R11 / R12 analyses

We will generate the following country-level table:

country name	population	Urban Concentration Ratio CR8	Gini index	HDI	GDP USD	year
Afghanistan	37,172,386	0.184	27.8	0.478	14,786,861,638	2021

country name	population	Urban Concentration Ratio CR8	Gini index	HDI	GDP USD	year
...
Zimbabwe	14,439,018	0.267	50.3	0.593	28,371,238,666	2021

We will have to retrieve the constituents of such table from multiple online databases:

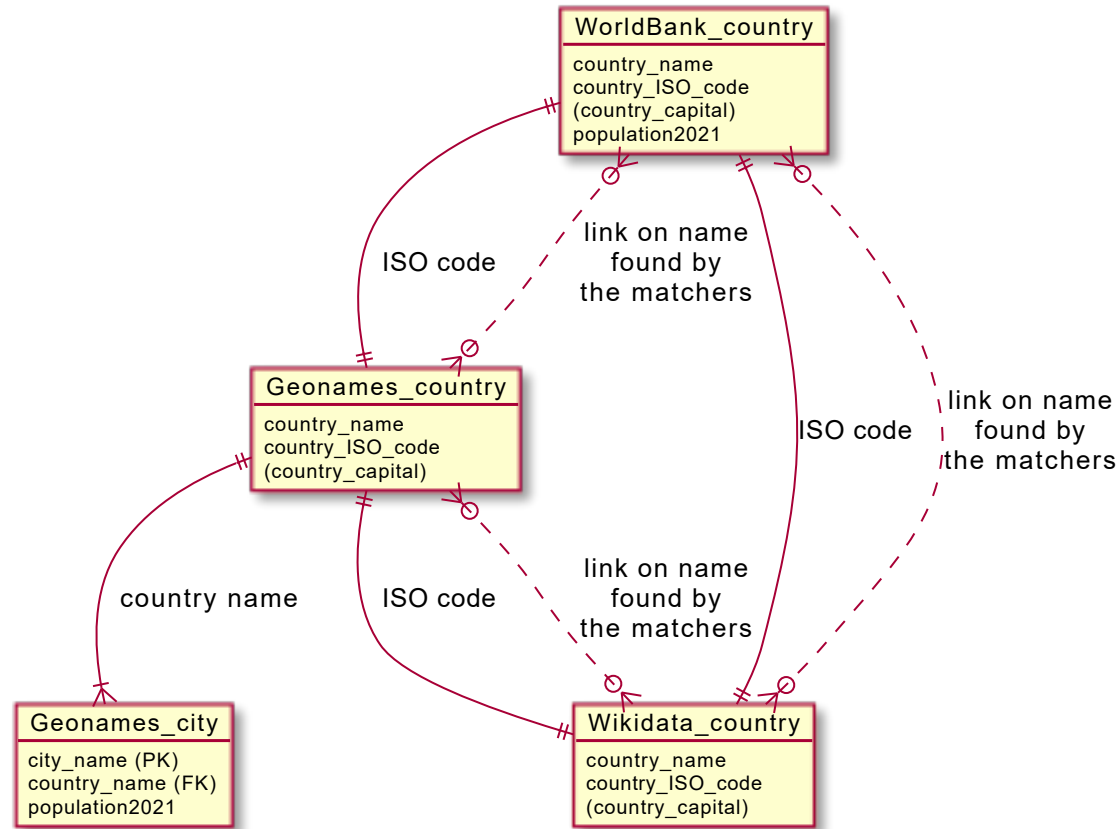
- ▶ www.geonames.org - for the population of the cities of the world. We will compute our own Urban Concentration Ratio by applying the [CR8](#) formula - a common market concentration measure - to the population of the cities
- ▶ data.worldbank.org - for the GDP
- ▶ www.wikidata.org - for the Gini index and the HDI, which we will get through a SPARQL query

The three online databases have 191 countries in common
but 8% of the country names do not match (US and China included):

www.geonames.org	data.worldbank.org	www.wikidata.org
Congo Republic	Congo, Rep.	Republic of the Congo
DR Congo	Congo, Dem. Rep.	Democratic Republic of the Congo
...

We will test several algorithms for a [record-linkage](#) based on country names.




After performing name-matching, we will use the country ISO codes available in each database for back-testing.

data model

We will:

- ▶ choose algorithm implementations that could be appropriate for different development preferences (analytics / low-code / no-code) and provide a score to support further human evaluation. We will not use Machine Learning algorithms.
- ▶ measure the statistical performance of the shortlisted algorithms

We will explore these technologies:

- the parametric [Similarity Search](#)  offered by KNIME for the 'no-code' option
- the python library [difflib](#)  for the 'analytical coding' option
- [Elasticsearch](#)  for the 'low-code, non-analytical coding' option

In terms of solution tuning,
we will always be careful on what meets the goal of software evaluation and what meets the goal of getting the final report done:

sub-solution	goal
pre-isolate the exact matches to keep the choices available to the matching algorithms as narrow as possible	final report
remove items already matched i.e., enforce constraints so that the matching will always be one-to-one and never one-to-many	final report
expand our horizon by leveraging on contextual data (e.g., the names of the capitals)	final report & evaluation
using country ISO codes only	final report
using country names only	evaluation
keep non relevant names included in some source datasets (e.g., aggregated names like South Asia or past country names)	evaluation

On the basis of the collected metrics, we will identify the technology with the best performance and we will generate the final report with its output.

Keywords

```
#CrossSectionalReporting #DataWrangling #ConcentrationIndices #OnlineDatabases #SQL #MSSQL  
#RecordLinkage #FuzzyMatching #StringSimilarity #KNIME #Python #ElasticSearch #SPARQL #RDF #Wikidata  
#ConfusionMatrix
```

Execution

Data capture - side notes

Below is the SPARQL query to extract the relevant data from Wikidata.

There may be multiple records for each country due to multiple capitals (as for South Africa), multiple HDI, multiple Gini. For this exercise, in case of such multiple attributes, we will take a random one by using the SAMPLE aggregating function.

```
1  # API: https://query.wikidata.org/?query=  
2  # problem: there may be duplicates, due to multiple capitals, multiple HDI, multiple Gini.  
3  # solution: SAMPLE  
4  # alt: SELECT ?countryId ?countryLabel ?IS02 ?IS03 (SAMPLE(?capitalLabel) AS ?capitalLabel_) (SAMPLE(?capitalId) AS  
5  
6  SELECT ?countryLabel ?IS02 ?IS03 (sample(?labelOfCapital) as ?Capital_) (SAMPLE(?HDI) AS ?HDI_) (SAMPLE(?Gini) AS ?G  
7  WHERE  
8  {  
9    ?country wdt:P31 wd:Q3624078 .  
10   #...sovereign state  
11   #not a former country...  
12   #FILTER NOT EXISTS {?country wdt:P31 wd:Q3024240}  
13   #and no an ancient civilisation (needed to exclude ancient Egypt)...  
14   #FILTER NOT EXISTS {?country wdt:P31 wd:Q28171280}  
15   #not required as long as you demand an ISO  
16   OPTIONAL { ?country wdt:P36 ?capital } .
```

```

17  OPTIONAL { ?country wdt:P1081 ?HDI } .
18  OPTIONAL { ?country wdt:P1125 ?Gini } .
19  { ?country wdt:P297 ?IS02 } .
20  { ?country wdt:P298 ?IS03 } .
21  #
22  OPTIONAL {?capital rdfs:label ?labelOfCapital}
23
24  SERVICE wikibase:label { bd:serviceParam wikibase:language "en" }
25  }
26  #GROUP BY ?countryId ?countryLabel ?IS02 ?IS03
27  GROUP BY ?countryLabel ?IS02 ?IS03
28  ORDER BY ?countryLabel
29  OFFSET 0
30  LIMIT 1000
31  # ... 196 results, Apr-2023

```

python 3: difflib

The behavior of this library is symmetrical and can be used in this way:

```

1  def string_similarity(str1, str2):
2      result = difflib.SequenceMatcher(a=str1, b=str2)
3      rt = result.ratio()
4      assert rt >= 0.0 and rt <= 1.0
5      return rt

```

We will use a matching process that computes the score for each pair and will start an iterative withdraw, picking the best match at each step, as below:

```

1  # dfL = left table # dfR = right table # left_src = left table tag # right_src = right table tag
2

```

```

3 # initializing the matrix
4 out_of_scale_init_val = -3.14 # no score will ever have this value, so: max(score,out_of_scale_init_val) = score, al
5 dfscores = pd.DataFrame(out_of_scale_init_val, index = dfL.index.values, columns = dfR.index.values)
6
7 for colR in dfscores.columns:
8     for rowL in dfscores.index.values:
9         dfscores.at[rowL, colR] = string_similarity(rowL,colR)
10        # we do not make assumptions on the symmetry
11
12 #finding and popping the best at each step:
13 for i in range(min(dfscores.shape)):
14     cols_bests = dfscores.max()
15     abs_best = max(cols_bests)
16     col_of_best = cols_bests.idxmax()
17     row_of_best = dfscores[col_of_best].idxmax()
18     output_new_match({'left_src',left_src, 'left':row_of_best, 'right_src',right_src, 'right':col_of_best, 'score':
19     # eg {'left_src':'wb', 'left':,"Turkiye", 'right_src':'gn','right':"Turkey", 'score':0.7692}
20     dfscores.drop(col_of_best,inplace=True,axis=1)
21     dfscores.drop(row_of_best,inplace=True,axis=0)

```

Confusion matrix and other performance metrics for this matching:


```

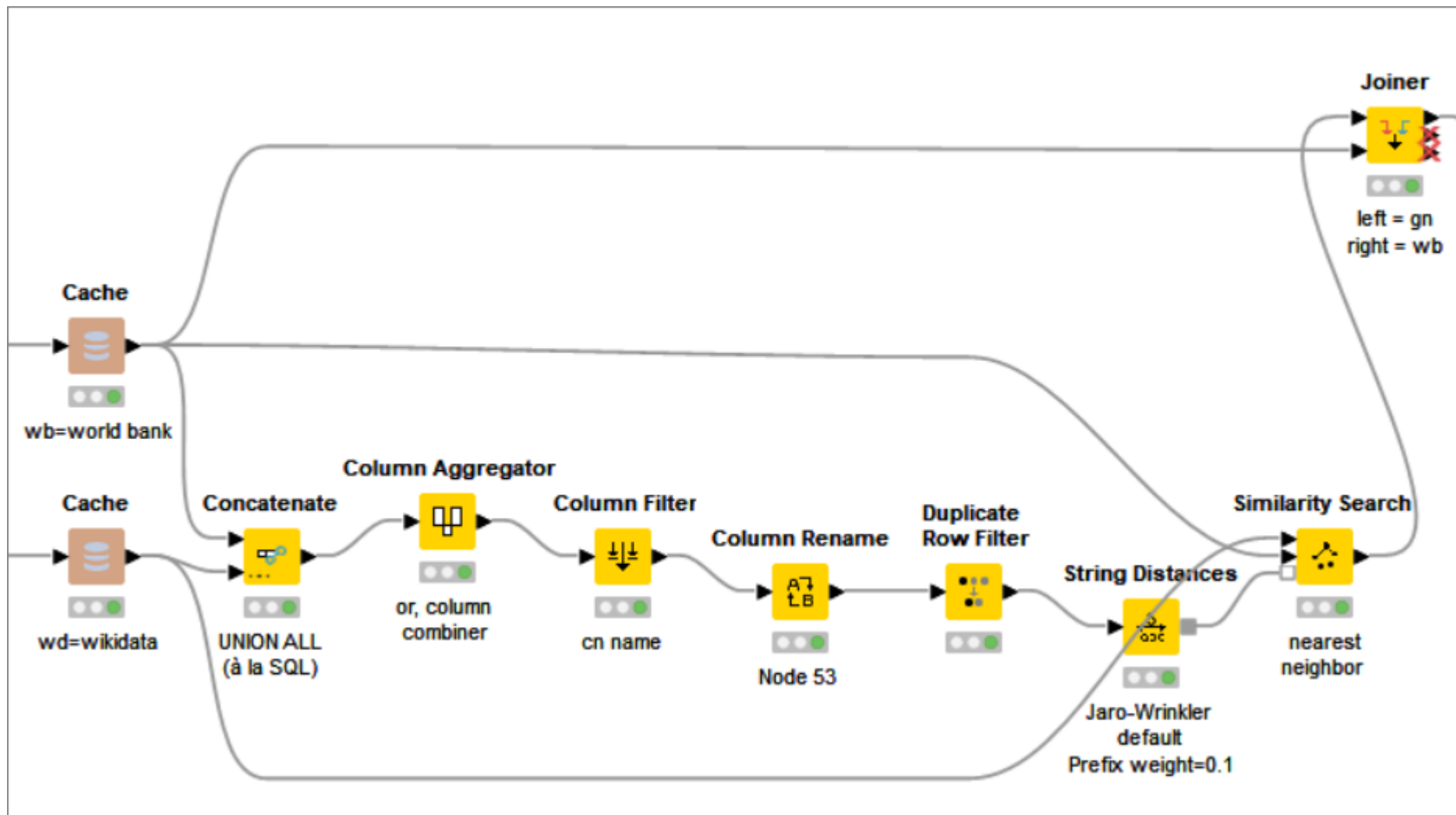
1 wb-vs-gn, cn only: TP 205 TN 37 FP 6 FN 4 TPR 0.981 FNR 0.019 TNR 0.86 FPR 0.14 accuracy 0.96 precision 0.972 recall
2 wd-vs-gn, cn only: TP 191 TN 54 FP 4 FN 3 TPR 0.985 FNR 0.015 TNR 0.931 FPR 0.069 accuracy 0.972 precision 0.979 rec
3 wb-vs-wd, cn only: TP 181 TN 13 FP 14 FN 3 TPR 0.984 FNR 0.016 TNR 0.481 FPR 0.519 accuracy 0.919 precision 0.928 re
4
5 wb-vs-gn, cn+cap: TP 211 TN 41 FP 0 FN 0 TPR 1.0 FNR 0.0 TNR 1.0 FPR 0.0 accuracy 1.0 precision 1.0 recall 1.0 F1-sc
6 wd-vs-gn, cn+cap: TP 195 TN 57 FP 0 FN 0 TPR 1.0 FNR 0.0 TNR 1.0 FPR 0.0 accuracy 1.0 precision 1.0 recall 1.0 F1-sc
7 wb-vs-wd, cn+cap: TP 191 TN 16 FP 4 FN 0 TPR 1.0 FNR 0.0 TNR 0.8 FPR 0.2 accuracy 0.981 precision 0.979 recall 1.0 F
8
9 NB
10 wb-vs-gn = matching world bank names (wb) with geonames names (gn) / wikidata (wd)


```

```
11 | cn only = matching based on country names only
12 | cn+cap  = matching based on country names + capital names
```

KNIME 4: Similarity Search

The String Distance task implemented in [KNIME 4.6](#)  creates a matrix, and the Similarity Search task runs the Jaro-Wrinkler's algorithm to compute the string distances with prefix weighting on the matrix. Other algorithms are also available.



The above workflow is integrated in a complete workflow as in this [picture](#) .

Confusion matrix and other performance metrics for this matching:

1	wb-vs-gn, cn only: TP 201 TN 0 FP 51 FN 10 TPR 0.953 FNR 0.047 TNR 0.0 FPR 1.0 accuracy 0.767 precision 0.798 recall
2	wd-vs-gn, cn only: TP 188 TN 0 FP 64 FN 6 TPR 0.969 FNR 0.031 TNR 0.0 FPR 1.0 accuracy 0.729 precision 0.746 recall
3	wb-vs-wd, cn only: TP 176 TN 18 FP 19 FN 12 TPR 0.936 FNR 0.064 TNR 0.486 FPR 0.514 accuracy 0.862 precision 0.903 r

```
4
5 wb-vs-gn, cn+cap: TP 201 TN 0 FP 51 FN 10 TPR 0.953 FNR 0.047 TNR 0.0 FPR 1.0 accuracy 0.767 precision 0.798 recall
6 wd-vs-gn, cn+cap: TP 188 TN 0 FP 64 FN 6 TPR 0.969 FNR 0.031 TNR 0.0 FPR 1.0 accuracy 0.729 precision 0.746 recall 0
7 wb-vs-wd, cn+cap: TP 176 TN 18 FP 19 FN 12 TPR 0.936 FNR 0.064 TNR 0.486 FPR 0.514 accuracy 0.862 precision 0.903 re
8
9 NB
10 wb-vs-gn = matching world bank names (wb) with geonames names (gn) / wikidata (wd)
11 cn only = matching based on country names only
12 cn+cap = matching based on country names + capital names
```

Elasticsearch:

The elasticsearch instance tested:

```
1 # docker
2 image: "elasticsearch:5.3.2"
```

The index creation command:

```
1 {
2   "settings": {
3     "analysis": {
4       "analyzer": {
5         "custom_analyzer1_nGram": {
6           "tokenizer": "custom_tokenizer1_nGram",
7           "filter": [ "lowercase", "asciifolding" ]
8         }
9       },
10      "tokenizer": {
11        "custom_tokenizer1_nGram": {
```

```

12         "type": "ngram", "min_gram": 1, "max_gram": 15,
13         "token_chars": [ "letter", "digit", "punctuation", "symbol" ]
14     }
15 }
16 }
17 },
18 "mappings": {
19     "cnmatchmapping": {
20         "properties": {
21             "cn": { "type": "text", "copy_to": "allfields" },
22             "capt": { "type": "text", "copy_to": "allfields" },
23             "src": { "type": "keyword" },
24             "allfields": { "type": "text", "analyzer": "custom_analyzer1_nGram" }
25         }
26     }
27 }
28 }

```

An example of a search command used:

```

1  {
2      "size": 1,
3      "query": {
4          "bool": {
5              "should": [
6                  { "match": { "cn": { "query": "country name X", "boost": 1, "fuzziness": "AUTO" } } },
7                  { "match": { "capt": { "query": "capital name Y or NIL", "boost": 2, "fuzziness": "AUTO" } } }
8              ],
9              "minimum_should_match": 2,
10             "filter": { "term": { "src": "dataset tag TT" } }
11         }
12     }

```

```

13     },
14     "sort": [ "_score" ]
15 }

```

The matching candidate is removed after it gets paired.

Confusion matrix and other performance metrics for this matching:

```

1  gn-vs-wb, cn only: TP 138 TN 112 FP 1 FN 1 TPR 0.993 FNR 0.007 TNR 0.991 FPR 0.009 accuracy 0.992 precision 0.993 re
2  gn-vs-wb, cn only: TP 195 TN 57 FP 0 FN 0 TPR 1.0 FNR 0.0 TNR 1.0 FPR 0.0 accuracy 1.0 precision 1.0 recall 1.0 F1-s
3  wb-vs-wd, cn only: TP 136 TN 71 FP 2 FN 2 TPR 0.986 FNR 0.014 TNR 0.973 FPR 0.027 accuracy 0.981 precision 0.986 rec
4
5  wb-vs-wd, cn+cap: TP 180 TN 29 FP 1 FN 1 TPR 0.994 FNR 0.006 TNR 0.967 FPR 0.033 accuracy 0.991 precision 0.994 reca
6  wd-vs-gn, cn+cap: TP 126 TN 69 FP 0 FN 0 TPR 1.0 FNR 0.0 TNR 1.0 FPR 0.0 accuracy 1.0 precision 1.0 recall 1.0 F1-sc
7  wd-vs-gn, cn+cap: TP 183 TN 12 FP 0 FN 0 TPR 1.0 FNR 0.0 TNR 1.0 FPR 0.0 accuracy 1.0 precision 1.0 recall 1.0 F1-sc
8
9  NB
10 wb-vs-gn = matching world bank names (wb) with geonames names (gn) / wikidata (wd)
11 cn only = matching based on country names only
12 cn+cap  = matching based on country names + capital names

```

Wrap-up: CR8 index calculation and mash-up

For our objective, we can accept FN (unmatched items) and we want to avoid having FP (mismatched items) because we are happy to manually match unmatched items but we don't want to verify the matches done.

In terms of country-names matching performance, with respect to how we have integrated and configured the shortlisted implementations, it is worth noting that the test done with Elasticsearch has had zero false negatives and zero false positives even without resorting to additional contextual data (i.e., the names of the capitals).

Summary of the approaches:

technology	strategy	data
KNIME	native many-to-many routine, closed-box	countrysnames, no capitals
python difflib	one-to-one routine, no-replacement	countrysnames, no capitals
elasticsearch fuzzy	one-to-one routine, no-replacement	countrysnames, no capitals

The Excel performance metrics table is available [here](#) 

After getting the paired country names, what is left to do to compose the final report is the just computation of the CR8 ratios and the joining of all our datasets. Below is the analytical code based on python pandas that, given the Geonames datasets with all the cities of the world with a population of 1000+ and the total population of each country, will compute the CR8 ratio as: sum of the populations of the 8 most populated cities of each country / total population

```

1  # cr8.py
2  import pandas as pd
3  in_csv_filename=r".\geonames-all-cities-with-a-population-1000-and-above.csv"
4
5  class ConcentrRatio:
6      def __init__(self, dataframe):
7          self.df = dataframe
8      def groupedBy(self, groupcol, valuecol):
9          self.dfsubtotAll = self.df.groupby(groupcol).agg(sum_all=(valuecol, 'sum'), count_all=(valuecol, 'count'))
10         self.groupcol, self.valuecol = [groupcol, valuecol]
11         return self
12     def excludedIfSizeBelow(self, min_group_size):
13         self.dfsubtotSignif = self.dfsubtotAll.query('count_all >= @min_group_size')
14         self.df = self.df.join(other=self.dfsubtotSignif, how='inner', on=self.groupcol)
15         self.min_group_size = min_group_size
16         return self
17     def withValuesRankedByGroup(self):

```

```

18         self.df['rank'] = self.df.groupby([self.groupcol])[self.valuecol].rank(ascending=False, method='dense')
19         self.df = self.df.query('rank <= @self.min_group_size')
20         return self
21     def getSubtotals(self):
22         self.dftop = self.df.groupby(self.groupcol).agg(sum_topn=(self.valuecol, 'sum'))
23         return self
24     def getDFwithRatio(self):
25         dfj = self.dfsubtotAll.join(other=self.dftop) # left outer
26         dfj['CR8'] = round(dfj['sum_topn'] / dfj['sum_all'], 3)
27         return dfj
28
29 df=pd.read_csv(in_csv_filename) #,skiprows=0,encoding='utf-8'
30 cr = ConcentrRatio(df).groupedBy('Country name EN', 'Population').excludedIfSizeBelow(8).withValuesRankedByGroup().ge
31 cr.getDFwithRatio().to_csv(in_csv_filename+'.out.csv')

```

equivalent in SQL for MS SQL Server:

```

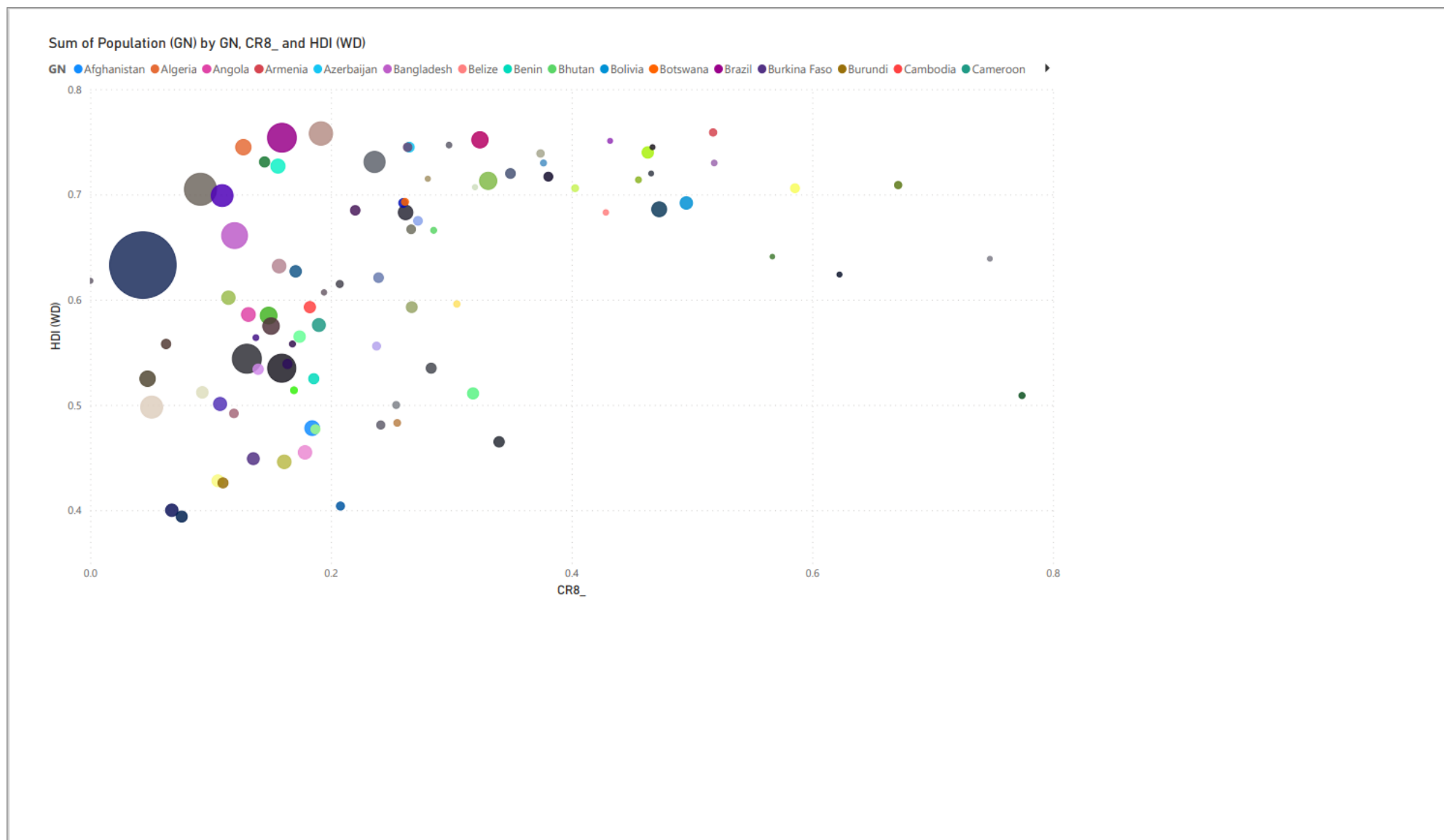
1  -- tested on SQL Server 2016 13.0
2  -- based on table cnct(cn,ct,pop) e.g. Italy,Rome,2873000
3  DECLARE @minctn INT, @maxctn INT;
4  SET @minctn = 8;
5  SET @maxctn = 8;
6
7  WITH
8  CTE_only_cn_with_many_ct AS (
9      SELECT cn FROM cnct GROUP BY cn HAVING count(*) >= @minctn
10 ),
11 CTE_only_cn_with_many_ct_ranked AS (
12     SELECT cn, ct, pop, ROW_NUMBER() /*not:DENSE_RANK()*/
13     OVER ( PARTITION BY cn ORDER BY pop DESC) rank_no
14     FROM cnct
15     WHERE cn in (SELECT cn FROM CTE_only_cn_with_many_ct)

```

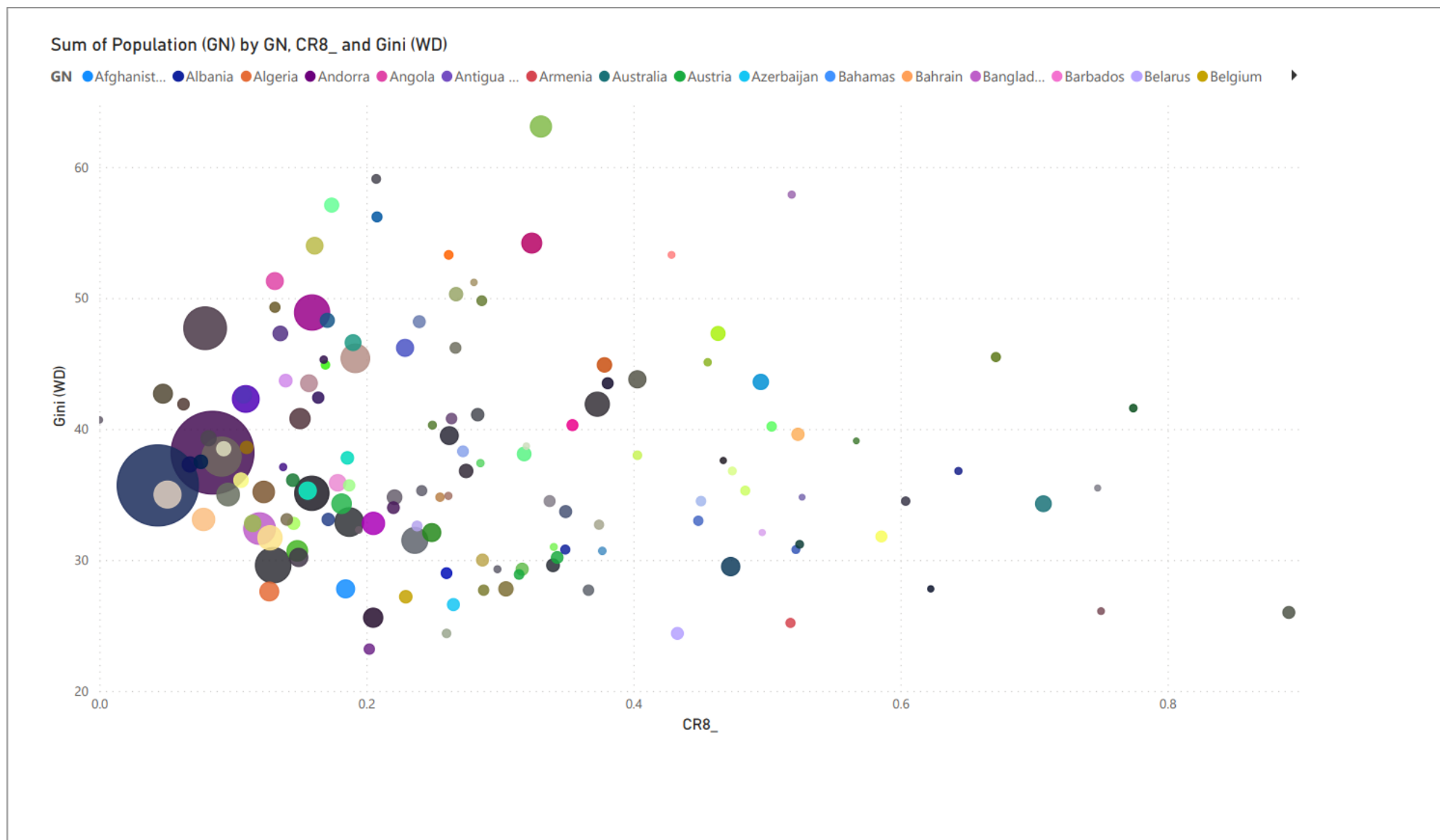
```
16 | )  
17 | SELECT cn,sum(pop) AS sumtoppop  
18 | FROM CTE_only_cn_with_many_ct_ranked  
19 | WHERE rank_no <= @maxctn  
20 | GROUP BY cn  
21 | ORDER BY cn
```

The final Excel data report is available [here](#) 

CR8 vs. HDI (PowerBI)



CR8 vs. Gini (PowerBI)








Appendix

recap on technology used

- Python difflib
- KNIME String Similarity
- Elasticsearch
- MS SQL
- Python pandas

see also

- [Splink: MoJ's open source library for probabilistic record linkage at scale](#) 
- [Data First: Harnessing the potential of linked administrative data for the justice system](#) 
 - <https://moj-analytical-services.github.io/splink/index.html> 
- <https://recordlinkage.readthedocs.io/en/latest/> 
- <https://www.kaggle.com/code/mikeyhogarth/record-linkage-tutorial> 

back to [Portfolio](#)

[backlog](#) 