[redacted]2022-10-07 - OFAC-GBR_DBFS_Scala

[redacted]

Overview

This notebook outlines how we preprocess columns in ofac and gbr for entity resolution and how we perform a simple entity resolution.

The code creates blockers based on type(e.g. entity,individual-No vessels or aircraft from the gbr datase)-country(from address) and type-country(from address)-nationality-date-of-birth. The latter can apply well to individuals.

(1) EntityMatching: We first compare the dataset within itself (ofact alone, gbr alone)
(2) EntityLinking: We now compare ofac with gbr. For both (1) and (2) steps, we compare the data sets within each block to reduce the computation time and try to match the names as exact or fuzzy matches. For (1), we try to match identities within the same data source and for (2) we can try to match identities with the two different data sources.

With limited time, I went ahead to find the exact names in different two different data sources. If the blockers are the same, we consider them as strong candidates. If the blockers are different and only names match we will consider them as weak candidates. With more extensive datasets, we will go over each blocker and find matches and union the results (code not shown). For type ="individual", we can check passport numbers

We can improve these results for fuzzy names with accepted certain threshold values. I can also explore a simple machine learning model to allow the model to adjust the threshold values for features in the future. I did not work on the ML model in this notebook, since the exercise did not seem focus on building a ML model.

```
// import two data sets for entity resolution and check schema

val jsonSchema_ofac = spark
  .read
 // .option("multiLine", true) // <-- the trick
  .json("/FileStore/tables/ofac.jsonl")
  .schema

val df_ofac = spark.read.schema(jsonSchema_ofac)
       .json("/FileStore/tables/ofac.jsonl")

df_ofac.printSchema()
df_ofac.show()


val jsonSchema_gbr = spark
  .read
 // .option("multiLine", true) // <-- the trick
  .json("/FileStore/tables/gbr.jsonl")
  .schema


val df_gbr = spark.read.schema(jsonSchema_ofac)
       .json("/FileStore/tables/gbr.jsonl")

//df_gbr.printSchema()
display(df_gbr)
```

	addresses
1	▸ [{"country": "Democratic People's Republic of Korea", "postal_code": null, "value": '
2	▸ [{"country": "Lebanon", "postal_code": null, "value": ""}]
3	▸ [{"country": null, "postal_code": null, "value": ""}]
4	▸ [{"country": null, "postal_code": null, "value": ""}]
5	▸ [{"country": null, "postal_code": null, "value": ""}]
6	▸ [{"country": null, "postal_code": null, "value": ""}]

Truncated results, showing first 1000 rows.

```
val distinctDfOfacType = df_ofac.select("type").distinct()
display(distinctDfOfacType)
```

1	Individual
2	Entity
3	Aircraft
4	Vessel

Showing all 4 rows.

```
val distinctDfGbrType = df_gbr.select("type").distinct()
display(distinctDfGbrType)
```

	type
1	Individual
2	Entity

Showing all 2 rows.

```
val distinctDfOfacNationality = df_ofac.select("nationality").distinct()

display(distinctDfOfacNationality)
```

	nationality
1	["Lebanon", "Nigeria", "Sierra Leone"]
2	["Chad"]
3	["South Sudan", "Sudan"]
4	["Russia"]
5	["Egypt", "Kenya"]
6	["Yemen"]

Showing all 209 rows.

```
import org.apache.spark.sql.functions._

//udf functions to create more columns
//country: "Uganda (reportedly in prison as of September 2016)" -> Uganda
//nationality: "Indian" -> India
//birthdate: 03 dec 1930 -> 03-12-1930 or 01/04/1980 -> 01-04-1980

def fix_nationality(s: String) : String = (s != null) match  {
  case false => null
  case true => s.substring(0, s.length - 1) //remove "n" at the end:
Indian -> India. This is not a rigorous way to do it. We should improve
}

def fix_birthdate(s: String) : String = (s != null) match  {
  case false => null
  case true => s.replace(" ", "-").replace("/", "-")
  |.replace("Jan", "01").replace("Feb", "02").replace("Mar",
"03").replace("Apr", "04")
  |.replace("May", "05").replace("Jun", "06").replace("Jul",
"07").replace("Aug", "08")
  |.replace("Sep", "09").replace("Oct", "10").replace("Nov",
"11").replace("Dec", "12")
  |.replace("circa", "").replace("00-00", "")
}

def remove_paren(s: String) : String = {
  if (s != null) {(s contains "(") match {
                 case true => {val indexVal = s.indexOf("(");
s.substring(0, indexVal)}
                 case false => {s}}}
  else
     return null}

val remove_parenUDF = udf(remove_paren(_))
val fix_nationalityUDF = udf(fix_nationality(_))
val fix_birthdateUDF = udf(fix_birthdate(_))

import org.apache.spark.sql.functions._
fix_nationality: (s: String)String
fix_birthdate: (s: String)String
remove_paren: (s: String)String
remove_parenUDF: org.apache.spark.sql.expressions.UserDefinedFunction = S
parkUserDefinedFunction($Lambda$8927/1199971021@18342be0,StringType,List
(Some(class[value[0]: string])),Some(class[value[0]: string]),None,true,t
rue)
fix_nationalityUDF: org.apache.spark.sql.expressions.UserDefinedFunction
```

```
= SparkUserDefinedFunction($Lambda$8928/1621043257@1ecd494a,StringType,Li
st(Some(class[value[0]: string])),Some(class[value[0]: string]),None,tru
e,true)
fix_birthdateUDF: org.apache.spark.sql.expressions.UserDefinedFunction =
SparkUserDefinedFunction($Lambda$8929/1473248377@39fb8761,StringType,List
(Some(class[value[0]: string])),Some(class[value[0]: string]),None,true,t
rue)
```

```
// preprocessing to make blockers
```

```
//ofac

var sliceDfOfac = df_ofac
display(sliceDfOfac)

var parseDfOfac = (sliceDfOfac
                  .withColumn("addresses", explode($"addresses"))
                  .withColumn("country", $"addresses".getItem("country"))
                  .withColumn("nationality_item",
explode($"nationality"))
                  .withColumn("reported_dates_of_birth_item",
explode($"reported_dates_of_birth"))
                   )

display(parseDfOfac)

//val distinctBlockerOfac = parseDfOfac.select("blocker").distinct()
//display(distinctBlockerOfac)
```

	addresses
1	▸ {"country": "Afghanistan", "postal_code": null, "value": ", , , , "}
2	▸ {"country": "Afghanistan", "postal_code": null, "value": ", , , , "}
3	▸ {"country": "Lebanon", "postal_code": null, "value": ", , , , "}
4	▸ {"country": "Lebanon", "postal_code": null, "value": ", , , , "}

5	▸ {"country": "United Kingdom", "postal_code": null, "value": ", , , London, "}

Truncated results, showing first 1000 rows.

```
parseDfOfac.createOrReplaceTempView("tempOfac")

val parseDfOfac_tv = spark.sql("SELECT * FROM tempOfac")

var parseDfOfac_tv_1 = (parseDfOfac_tv
                        .withColumn("country_modified",
remove_parenUDF(col("country")))
                        .withColumn("birthdate_modified",
fix_birthdateUDF(col("reported_dates_of_birth_item")))
                        //.withColumn("nationality_modified",
fix_nationalityUDF(col("nationality_item"))) only for gbr
                        .withColumn("blocker_type_country",
concat(col("type"), lit("-"), col("country_modified")))
                        .withColumn("blocker_type_birth_nationality",
concat(col("type"), lit("-"), col("birthdate_modified"), lit("-"),
col("nationality_item")))
                         )

parseDfOfac = parseDfOfac_tv_1

//display(parseDfOfac)

val DfOfac_final = parseDfOfac.select("id", "name",
"blocker_type_country", "blocker_type_birth_nationality")
display(DfOfac_final)
```

	id	name	bl
1	6915	Abdullah Ahmed ABDULLAH	In
2	6916	Ahmed Mohammed Hamed ALI	In
3	6921	Ali ATWA	In
4	6926	Hasan IZZ-AL-DIN	In
5	6950	Omar Mahmoud UTHMAN	In
6	6950	Omar Mahmoud UTHMAN	In

Truncated results, showing first 1000 rows.

```
var sliceDfGbr = df_gbr
display(sliceDfGbr)

var parseDfGbr = (sliceDfGbr
                  .withColumn("id",monotonicallyIncreasingId) //id are
all null so I provide the ids
                  .withColumn("addresses", explode($"addresses"))
                  .withColumn("country", $"addresses".getItem("country"))
                  .withColumn("nationality_item",
explode($"nationality"))
                  .withColumn("reported_dates_of_birth_item",
explode($"reported_dates_of_birth"))
                  )

display(parseDfGbr)
```

	addresses
1	▸ {"country": "Democratic People's Republic of Korea", "postal_code": null, "value": "I
2	▸ {"country": "Lebanon", "postal_code": null, "value": ""}
3	▸ {"country": null, "postal_code": null, "value": ""}
4	▸ {"country": null, "postal_code": null, "value": ""}
5	▸ {"country": null, "postal_code": null, "value": ""}
6	▸ {"country": null, "postal_code": null, "value": ""}

Truncated results, showing first 1000 rows.

```
parseDfGbr.createOrReplaceTempView("tempFrame")

val parseDfGbr_tv = spark.sql("SELECT * FROM tempFrame")

var parseDfGbr_tv_1 = (parseDfGbr_tv
                       .withColumn("country_modified",
remove_parenUDF(col("country")))
                       .withColumn("birthdate_modified",
fix_birthdateUDF(col("reported_dates_of_birth_item")))
                       .withColumn("nationality_modified",
fix_nationalityUDF(col("nationality_item")))
                       .withColumn("blocker_type_country",
concat(col("type"), lit("-"), col("country_modified")))
                       .withColumn("blocker_type_birth_nationality",
concat(col("type"), lit("-"), col("birthdate_modified"), lit("-"),
col("nationality_modified")))
                        )

parseDfGbr = parseDfGbr_tv_1

//display(parseDfGbr)
```

```
parseDfGbr_tv: org.apache.spark.sql.DataFrame = [addresses: struct<countr
y: string, postal_code: string ... 1 more field>, aliases: array<struct<t
ype:string,value:string>> ... 11 more fields]
parseDfGbr_tv_1: org.apache.spark.sql.DataFrame = [addresses: struct<coun
try: string, postal_code: string ... 1 more field>, aliases: array<struct
<type:string,value:string>> ... 16 more fields]
parseDfGbr: org.apache.spark.sql.DataFrame = [addresses: struct<country:
string, postal_code: string ... 1 more field>, aliases: array<struct<typ
e:string,value:string>> ... 16 more fields]
```

```
val DfGbr_final =  (parseDfGbr
                .select("id", "name", "blocker_type_country",
"blocker_type_birth_nationality")
                .withColumnRenamed("id", "id_gbr")
                .withColumnRenamed("name", "name_gbr")
                .withColumnRenamed("blocker_type_country",
"blocker_type_country_gbr")
                .withColumnRenamed("blocker_type_birth_nationality",
"blocker_type_birth_nationality_gbr")
                     )
```

```
DfGbr_final: org.apache.spark.sql.DataFrame = [id_gbr: bigint, name_gbr:
string ... 2 more fields]
```

```
display(parseDfGbr.filter("name = 'Said BAHAJI'"))
```

	addresses
1	▸ {"country": "Germany", "postal_code": null, "value": "Bunatwiete 23, 21073, Hamburg"}

Showing all 1 rows.

```
//find the same names between two data sources
//if the blockers are the same, we consider them as strong candidates for match

val join_result_names_all =
DfOfac_final.as("a").join(DfGbr_final.as("b"),
                                        $"a.name" === $"b.name_gbr"
                                        )

display(join_result_names_all)
```

	id	name	blocker_type_count
1	6950	Omar Mahmoud UTHMAN	Individual-United Kin
2	6950	Omar Mahmoud UTHMAN	Individual-United Kin
3	6950	Omar Mahmoud UTHMAN	Individual-United Kin
4	6950	Omar Mahmoud UTHMAN	Individual-United Kin
5	6950	Omar Mahmoud UTHMAN	Individual-Jordan
6	6950	Omar Mahmoud UTHMAN	Individual-Jordan

Truncated results, showing first 1000 rows.

```
// good matches

display(join_result_names_all.filter("blocker_type_country = blocker_type_country_gbr")
       )
```

	id	name	blocker_type_count
1	7264	Said BAHAJI	Individual-Germany
2	7264	Said BAHAJI	Individual-Germany
3	7264	Said BAHAJI	Individual-Germany
4	7264	Said BAHAJI	Individual-Germany

5	7264	Said BAHAJI	Individual-Germany
6	7264	Said BAHAJI	Individual-Germany

Showing all 849 rows.

```
// good matches

display(join_result_names_all.filter("blocker_type_birth_nationality =
blocker_type_birth_nationality_gbr")
       )
```

	id	name	blocker_type_
1	7329	Mohamad Iqbal ABDURRAHMAN	Individual-Indo
2	7329	Mohamad Iqbal ABDURRAHMAN	Individual-Indo
3	7944	Djamel MOUSTFA	Individual-Ger
4	7944	Djamel MOUSTFA	Individual-Ger
5	7944	Djamel MOUSTFA	Individual-Ger
6	8098	Mohamed Amin MOSTAFA	Individual-Italy

Showing all 139 rows.

	id	name	blocker_type_count
1	7329	Mohamad Iqbal ABDURRAHMAN	Individual-Indonesia
2	7329	Mohamad Iqbal ABDURRAHMAN	Individual-Indonesia
3	8184	Jawhar Majid AL-DURI	Individual-Iraq
4	8185	Nidal AL-RABI'I	Individual-Iraq
5	8186	Ali Barzan Ibrahim Hasan AL-TIKRITI	Individual-Switzerlan
6	8189	Khawla Barzan Ibrahim Hasan AL-TIKRITI	Individual-Switzerlan

Showing all 54 rows.