Malverde Data Challenge

Contents

[Initial Data Analysis 3](#_Toc195272282)

[Initial Data Transforming and Load 5](#_Toc195272283)

[Import to Python 5](#_Toc195272284)

[Handling UTF-8 Characters 6](#_Toc195272285)

[Handling SQL Date Types 6](#_Toc195272286)

[Exporting Cleansed CSV 7](#_Toc195272287)

[Loading into MySQL Workbench 7](#_Toc195272288)

[Analysis & Data Cleansing Process 10](#_Toc195272289)

[Data Quality Check Assessment 13](#_Toc195272290)

[Duplicates 13](#_Toc195272291)

[Consistency of Data 14](#_Toc195272292)

[Missing Fields 14](#_Toc195272293)

[Exporting Data 15](#_Toc195272294)

# Initial Data Analysis

Upon downloading the CSV file, I performed an initial look at the data to understand the data structures. The CSV has a first line, which I deleted, that states the last updated date, however this is unnecessary to my data structure, so having deleted it, I am now left with a clean CSV file with first row column names.

The column names are:

Name 6

Name 1

Name 2

Name 3

Name 4

Name 5

Title

Name Non-Latin Script

Non-Latin Script Type

Non-Latin Script Language

DOB

Town of Birth

Country of Birth

Nationality

Passport Number

Passport Details

National Identification Number

National Identification Details

Position

Address 1

Address 2

Address 3

Address 4

Address 5

Address 6

Post/Zip Code

Country

Other Information

Group Type

Alias Type

Alias Quality

Regime

Listed On

UK Sanctions List Date Designated

Last Updated

Group ID

There seems to be quite a few empty values in some columns, such as names and addresses, I will check the data population of those once I import the data in MySQL Workbench.

# Initial Data Transforming and Load

When attempting to import data to MySQL, I appear to be facing multiple errors, which appears to be due to unsupported characters outside the UTF-8 standard, and similar formatting issues. As such, I will begin by exploring, then cleaning the data in Python, before exporting it as a CSV to be imported into SQL.

## Import to Python

I imported the data to Python using typical Pandas library code, to read in the file as a CSV and save to a dataframe:

A screenshot of a computer program

AI-generated content may be incorrect.

Having checked the logs of MySQL Workbench during importing, there were 2 main issues I came across, one before the other. Firstly, I was facing format errors due to some characters in the file being unrecognised when importing as UTF-8, and so I need to create a list of columns that need to be cleaned in that regard. The second issue I faced is that dates in the file are written as DD-MM-YYYY, however SQL dates are standardised in the format YYYY-MM-DD, and as such I would have to convert those too. To begin with, I created a list of columns that need to be cleaned in each case:

A screenshot of a computer program

AI-generated content may be incorrect.

With these, I could direct cleansing functions at specific columns so as to not disrupt others, for example, to not remove slashes used in the date column (eg 10/10/1990 becoming 10101990 after cleansing)

## Handling UTF-8 Characters

To handle UTF-8 characters, I initially tried to write a simple regex to remove all special characters except those used for normal text and CSV attributes (like LF and ,) but this unfortunately would not work, as it would break CSV imports later on in MySQL Workbench. Instead, I used a simple regex to retain all alpha-numeric characters in my data, and remove everything else, which while unfortunately removes non-Latin script-based languages, it would at least allow data to be imported into Workbench. My cleanse code is as follows:

A screen shot of a computer

AI-generated content may be incorrect.

## Handling SQL Date Types

When I looked through the data initially, I noticed that firstly dates were in the form DD-MM-YYYY, but also as I dug further down, I noticed that some dates were initialized as 00-00-YYYY, likely due to missing information due to lost records or data. As such, to preserve as much information as possible, I created a function that will return the date YYYY/01/01 for those invalid dates, and will flip all dates around to the correct format. That function looks like so:

A screenshot of a computer program

AI-generated content may be incorrect.

## Exporting Cleansed CSV

With these transformations done, I could then export the CSV as a cleansed file, to then be imported into MySQL Workbench

A screenshot of a computer

AI-generated content may be incorrect.

I check the shape at the start and the end of the script to ensure that no significant changes happened to the length of the file without realising.

## Loading into MySQL Workbench

With this done, I then loaded the data into SQL, firstly by defining a database and a long table to fit the current data structure, a snippet of which can be seen below:

A screen shot of a computer program

AI-generated content may be incorrect.

I used VARCHAR where possible to optimize performance on those fields, and for fields where there were much longer descriptions that surpassed the 255 limit, I used TEXT on those instead. After table creation, I then used the SQL Import Table Wizard to import the CSV into this table, the process can be seen below:

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

After this, I then did a quick query to check the table, which appeared to have loaded in just fine:

A screenshot of a computer

AI-generated content may be incorrect.

From here, I can then next check the data, clean it, and remove columns that might not be required anymore.

# Analysis & Data Cleansing Process

Firstly, conscious of removing the non-Latin scripts, it is likely that these columns will be either empty or unusable due to my alphanumeric filtering and can be removed. I did a quick query to check the contents of the column:

A screenshot of a computer

AI-generated content may be incorrect.

As can be seen, the data in this is mostly unusable, so all columns related to non-Latin scripts can be dropped, which results in this:

A screenshot of a computer

AI-generated content may be incorrect.

The next large data quality problem I see is the sheer number of name columns, I can check how many values each one has first. By running a distinct count across all name columns, I can determine how many values each has:

A screenshot of a computer

AI-generated content may be incorrect.

The issue of names is also compounded by the fact that the data seems to group multiple types of entities together:

A screenshot of a computer

AI-generated content may be incorrect.

And when checking the names split per group, the results show that only Individual really needs the name columns, while entity and ship only need the one column:

A screenshot of a computer

AI-generated content may be incorrect.

As such, I will split this data into 3 different tables of data, but I will first perform a quality check on the remaining columns to see what I need to define per table. Of course, Ships and Entities will not have personal IDs or dates/locations associated with them, so we can query those to be sure, and exclude those from our new tables:

A screenshot of a computer

AI-generated content may be incorrect.

As seen, they all contain either 1 or 0, meaning just the empty null value for the row. Therefore, these can be removed from the entity and ship tables, which I will define now for Individuals:

A screenshot of a computer

AI-generated content may be incorrect.

I then repeated this process with Entity and Ships:

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

This results in 3 new tables with my database, with a much cleaner data structure overall, optimising space used and improving human readability.

From here, I will go on to perform a data quality check, focusing on the sanctioned\_individuals table.

# Data Quality Check Assessment

## Duplicates

The first thing I want to check is for duplicate values within the individuals table, as I haven’t checked for this yet. To do so, I run a simple group query to check for multiple occurrences in names:

A screenshot of a computer

AI-generated content may be incorrect.

This query shows that there are duplicate names, even when every name column is selected. Looking into this by selecting a single row of names, I can see what the duplicates look like, to see if they are actually duplicates, or if there are differences in the other fields, in this case I am looking at a specific example with a fairly specific name which had 10 values associated with it:

A screenshot of a computer

AI-generated content may be incorrect.

Interestingly, the only differences between these records are the date of birth, and the country listed for the record. All other details in the record, including identification numbers like Passport and National ID number, are the exact same, meaning that this is likely the same person across multiple records. Considering the dates are set to the default of 01/01/YYYY, it’s likely that these are duplicate records where country and date of birth were not narrowed down due to some reasons.

## Consistency of Data

From my observations, the data throughout the process has been mostly consistent; Some values have been missing in some fields, like DOBs, and of course before splitting, there were larger voids for mismatched entity types and the columns associated with them, like National ID with entity/ships, but aside from that, the data contained within seems to usually be in the same format, with consistent typefaces and structure.

Of course there was the issue of UTF-8 characters, and some characters not being eligible for importing into MySQL, which may be a compatibility issue, or a encoding issue, which was a problem as I had to lose the non-Latin typeface data, which banks in other countries may have held onto for identifying customers, eg, banks in Saudi Arabia could have their customers in Arabic character names rather than English.

## Missing Fields

This has already been handled on my end to help with my analysis, but of course inconsistent and missing fields due to mismatched entities was a problem at the start, causing bloated, empty columns, which is especially a problem when around 27% of the data is not eligible to have that missing data like date of births and such.

Aside from this, there is still quite a large number of empty fields across the dataset, with address fields in a lot of areas being blank (likely due to having 6 different address columns), as well as positions and ID numbers for Individuals being empty in quite a few cases.

# Exporting Data

Finally, I want to export my Individual table to CSV. To do this, I first need to find the output folder that MySQL has permissions to write to securely, which can be found via this command:

A screenshot of a computer

AI-generated content may be incorrect.

With this, I can define my OUTFILE function which exports the table data into a CSV in that directory, which looks like so:

A screen shot of a computer code

AI-generated content may be incorrect.

The first SELECT is for defining column headings as first row of the file, followed up by a UNION with the rest of the data from the table. The result is a CSV output like so that can be used anywhere with CSV support:

A screen shot of a computer screen

AI-generated content may be incorrect.