Quality Control (QC)

Running the QC Script

Recommended:

You can run this script completely in the cloud by using Jupyter Notebook (hosted on GoogleColab):
https://colab.research.google.com/drive/14ibCmUjaxIcAkB1vh6tDLf8V8PUIxt3B?usp=sharing
Jupyter Notebooks provide much better functionality for debugging and data exploration as the code is separated into cells where each cell can be run separately without executing the rest of the code, making it easier to find errors, or call specific variables.
This version also handles fetching all documents automatically so no extra set up is required.

Locally:

You can run this script locally by downloading it here: https://pastebin.com/jp22ptrz
Or from the github repository of openTECR.
Required libraries: pandas, numpy, odfpy
Those can be installed using pip:

```
!pip install odfpy
!pip install numpy
!pip install pandas
```

Preparing the data

GoogleCollab:

Data is fetched automatically in .csv and .ods formats and saved into the current directory.

Linux (on Windows machines, it's recommended to use WSL):

Navigate to the desired directory and execute the following commands in the terminal.

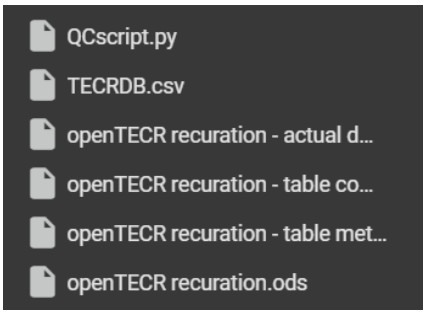
```
wget -O "openTECR recuration - actual data.csv"
"https://docs.google.com/spreadsheets/d/1jLIxEXVzE2SAzIB0UxBfcFoHrzjzf9euB6ART2VDE8c/e
xport?format=csv&gid=2123069643"
wget -O "openTECR recuration - table codes.csv"
"https://docs.google.com/spreadsheets/d/1jLIxEXVzE2SAzIB0UxBfcFoHrzjzf9euB6ART2VDE8c/e
xport?format=csv&gid=831893235"
```

```
wget -O "openTECR recuration - table metadata.csv"
"https://docs.google.com/spreadsheets/d/1jLIxEXVzE2SAzIB0UxBfcFoHrzjzf9euB6ART2VDE8c/e
xport?format=csv&gid=1475422539"
wget -O "openTECR recuration.ods"
"https://docs.google.com/spreadsheets/d/1jLIxEXVzE2SAzIB0UxBfcFoHrzjzf9euB6ART2VDE8c/e
xport?format=ods"
wget -O "TECRDB.csv"
"https://w3id.org/related-to/doi.org/10.5281/zenodo.3978439/files/TECRDB.csv"
```

Manually:

Navigate to:
https://docs.google.com/spreadsheets/d/1jLIxEXVzE2SAzIB0UxBfcFoHrzjzf9euB6ART2VDE8c/export?format=csv&gid=2123069643
https://docs.google.com/spreadsheets/d/1jLIxEXVzE2SAzIB0UxBfcFoHrzjzf9euB6ART2VDE8c/export?format=csv&gid=831893235
https://docs.google.com/spreadsheets/d/1jLIxEXVzE2SAzIB0UxBfcFoHrzjzf9euB6ART2VDE8c/export?format=csv&gid=1475422539
https://docs.google.com/spreadsheets/d/1jLIxEXVzE2SAzIB0UxBfcFoHrzjzf9euB6ART2VDE8c/export?format=ods
https://w3id.org/related-to/doi.org/10.5281/zenodo.3978439/files/TECRDB.csv

Visiting the links above will automatically download needed files. Place those files into the same directory as your QC script. At the end your folder should look like this:



Running and understanding the code:

```
READ_CSV = True
```

If set to “True”, the script will work with “openTECR recreation - actual data.csv”, otherwise it will extract this sheet from “openTECR recreation.ods” if set to “False”. The script works with both, but processing .ods takes longer time. It’s recommended to keep this value as True.

Checks

Check 1: Rows containing NaNs.

```
subset = ["part", "page", "col l/r", "table from top", "entry nr"]
counter = df[subset].isna().any(axis=1).sum()
print(f"A total of {counter} rows contained NaNs.")
```

This part counts **rows containing NaN values in columns "part", "page", "col l/r", "table from top", "entry nr" in the original script**.

Check 2: Duplicates & Errors.

```
## QC

## Checking for duplicates
## Replaced reference with reference_code as it is the real column name
## Not actually needed anymore due ot duplicable table column

if True:
    ## duplicates and errors
    test_df = df
    MANUALLY_EXCLUDED_DUPLICATES = [
        "54STA",
        "71TAN/JOH",
        "91HOR/UEH",
        "76SCH/KRI",
        "99TEW/SCH"
    ]
    ## shouldn't be necessary after introduction of duplicate_table
column; so we use:
    MANUALLY_EXCLUDED_DUPLICATES = []
    ##
    test_df =
test_df[~test_df.reference_code.isin(MANUALLY_EXCLUDED_DUPLICATES)]
    assert sum(test_df[((test_df["entry nr"]=="duplicate") |
(test_df["entry nr"]=="error"))].id.isna())==0, ("Duplicate or error found
for an empty-ID row", test_df[(((test_df["entry nr"]=="duplicate") |
(test_df["entry nr"]=="error"))) & test_df.id.isna()])
    ## Ensures that rows containing duplicate or error in entry nr column
also has an appropriate ID.
```

```
#print("I am removing the following duplicates and errors:")
df = df[~((df["entry nr"]=="duplicate") | (df["entry nr"]=="error"))]
## Removes such rows
```

This code removes **all duplicates and errors**, ensuring that only relevant data is left for analysis.

Check 3: Position & Reference.

```
#QC
if True:
    ##check completeness of position annotation
    na_counter = df[["part","page","col l/r","table from top", "entry
nr"]].isna().sum(axis="columns") ##Counts NaN values in position
annotation
    #print(na_counter)
    assert len(df[~na_counter.isin([0,5])])==0,
print(df[~na_counter.isin([0,5])][["id","reference","part","page","col
l/r","table from top", "entry nr"]].to_string())
    #Prints problematic rows if any

    ##Checks for spaces in /reference_code/ column.
    assert len(df[df.reference_code.str.contains(" ").fillna(False)])==0,
print(df[df.reference_code.str.contains(" ").fillna(False)].to_string())
    ## And output problematic rows.

## drop NaNs -- these entries just haven't been worked on and can't be
checked
df = df.dropna(subset=["part","page","col l/r","table from top", "entry
nr"])

## convert values in columns part, page, col l/r, table from top and entry
nr to integers
df[["part","page","col l/r","table from top", "entry nr"]] =
df[["part","page","col l/r","table from top", "entry nr"]].astype(int)
```

Converting position values into integers and ensuring their continuity (that all pointers corresponding to part, page, column and et cetera) are present. It also ensures that reference codes contain no white spaces (‘ ‘).

Check 4: Unique ID.

```
assert len(df.dropna(subset=["id"]).id.unique()) ==
len(df.dropna(subset=["id"])),
df.dropna(subset=["id"])[df.dropna(subset=["id"]).id.duplicated()].to_stri
ng()
```

Ensures that all IDs are unique.

Check 5: Sub-table integrity.

```
    for which, g in df.groupby(["part","page","col l/r","table from
top"]): # g is subset corresponding to specific part of the dataframe, as
grouped by previously
        #print((which,g))
        #if which == (3, 1091, 1, 1):
        #    continue

        assert len(g.reference_code.unique())==1, (which,
print(g.to_string())) ## Checks that all values in reference_code have the
same value--
        assert len(g.EC.unique()) == 1, (which, print(g.to_string())) ##
Check that all EC values are uniform
        assert len(g.reaction.unique())==1, (which, print(g.to_string()))
## - fixed "description" for correct column name
        assert sorted(g["entry nr"].values)==list(range(1,g["entry
nr"].max()+1)), (which, print(g.to_string())) ## Check that all values in
entry nr are ordered from lowest to highest
```

Sub-tables (g) are generated from grouping entries in “actual data” by values from part, page, col l/r and table from top values. It ensures that all values in “reference_code” are the same (so they come from the same source), while doing the same for reactions and EC values. It also makes sure that values in entry nr are arranged in consecutive way: from smallest to largest.

Check 6: Columns.

```
## column values either 1 or 2
    for which, g in df.groupby(["part", "page"]):
        assert all([i in [1,2] for i in g["col l/r"].values]), (which,
print(g.to_string())) # ensures that all values in col l/r are either 1 or
```

Ensures that all values in col l/r column are either one (left) or two (right).

Check 7: Pages.

```
## page numbers consistently continuous
    for which, g in df.groupby("part"):
        MANUALLY_EXCLUDED_PARTS = [
            #2,
            #3,
            #7
        ]
        if which in MANUALLY_EXCLUDED_PARTS: #ensures that page numbers
are continious by parts
            continue

        print(f"----- This is about part {which} -----")
        should_be = list(range(g["page"].min(), g["page"].max() + 1))
        for page in should_be:
            if page not in g["page"].unique():
                print(f"Missing page: {page}") #Finds missing pages and
prints them out
```

Ensures that all pages from the source are present and continuous. If not, outputs the missing pages from the entries.

Check 8: Noor Dataset - IDs

```
assert set(noor.id) - set(online.id) == set(), f"The following IDs were
deleted online: {set(noor.id)-set(online.id)}"
## Checks that all ID are present in both openTECR and Noor
```

Ensures that all IDs are present in both Noor (TECRDB.csv) dataset and openTECR.

Check 9: Noor Dataset - Prime_K value.

```
import numpy as np
kPn = []
for i in range(len(leftjoined.K_prime_x)):
  if leftjoined.K_prime_x[i] != leftjoined.K_prime_y[i]:
    if np.isnan(leftjoined.K_prime_x[i]) == False:
      #print(leftjoined.K_prime_x[i])
      #print(leftjoined.K_prime_y[i])
      ##Trying to match weird rounding up format of openTECR Kprime values
      if len(str(leftjoined.K_prime_x[i]).split('.')[1])>4:
```

```
        rounded =
round(leftjoined.K_prime_x[i],len(str(leftjoined.K_prime_x[i]).split('.')[
1])-1)
        #print(rounded)
      elif len(str(leftjoined.K_prime_x[i]).split('.')[1])==2:
        rounded = round(leftjoined.K_prime_x[i],1)
        #print(rounded)
      elif len(str(leftjoined.K_prime_x[i]).split('.')[1])==3:
        rounded = round(leftjoined.K_prime_x[i], 2)
        #print(rounded)
      elif len(str(leftjoined.K_prime_x[i]).split('.')[1])==1:
        rounded = round(leftjoined.K_prime_x[i], 0)
        #print(rounded)
      #print(round(leftjoined.K_prime_x[i], 2))
      if rounded+0.1 == leftjoined.K_prime_y[i]:
        kPn.append(rounded+0.1)
      else:
        kPn.append(rounded)
    else:
      kPn.append(leftjoined.K_prime_x[i])
  else:
    kPn.append(leftjoined.K_prime_x[i])
for i in range(len(leftjoined.K_prime_y)):
  if kPn[i] != leftjoined.K_prime_y[i]:
    if np.isnan(leftjoined.K_prime_y[i]) == False:
      print(f'Noor set K_prime value: {kPn[i]}')
      print(f'openTECR set K_prime value: {leftjoined.K_prime_y[i]}')
## Some cases couldn't be handled since they don't follow rounding logic
```

Checks that values in the “Prime_K” column of Noor and openTECR are the same while trying to adjust to values in openTECR being occasionally rounded. Prints out different values for further annotation and review.

Check 10: Noor Dataset - Parameters.

```
## Columns that should be the same
SHOULD_BE_THE_SAME = [
    "reference_code",
    "EC",
    "reaction",
```

```
    "K",
    "temperature",
    "ionic_strength",
    "p_h",
    "p_mg",
    #"K_prime", # - checked manually, weird rounding up in online data?
Handled by code above
]
for s in SHOULD_BE_THE_SAME:
    entries_where_both_are_nans = leftjoined[ leftjoined[f"{s}_x"].isna()
& leftjoined[f"{s}_y"].isna() ]
    if len(entries_where_both_are_nans) == 0:
        assert (leftjoined[f"{s}_x"] == leftjoined[f"{s}_y"]).all(), (s,
print(leftjoined[~(leftjoined[f"{s}_x"] ==
leftjoined[f"{s}_y"])][["id",f"{s}_x",f"{s}_y"]].to_string()))
    else:
        tmp = leftjoined[ ~ (leftjoined[f"{s}_x"].isna() &
leftjoined[f"{s}_y"].isna()) ]
        assert (tmp[f"{s}_x"] == tmp[f"{s}_y"]).all(), (s,
print(tmp[~(tmp[f"{s}_x"] ==
tmp[f"{s}_y"])][["id",f"{s}_x",f"{s}_y"]].to_string()))

## The code above checks that all values in those columns between both
Noor (_x) and openTECR (_y) are either NaN or equal. If not, raises
assertion error and prints out incorrect rows
```

Ensures that r**eference code, EC, reaction description, K, temperature, ionic strength, pH, and pMg** are the same across different entries across both Noor and openTECR datasets.

Check 11: Rows without ID.

```
potential_errors = potential_errors[
~potential_errors.id_y.isin(MANUALLY_EXCLUDED) ]
if len(potential_errors) > 0:
    print("The following entries might have been added as a new row
without id, where they should have corrected a row with id:")
    print(potential_errors.to_string())
```

Finds rows with no corresponding ID.

Table Codes.

In the rest of the code, the QC script annotates an online spreadsheet with table codes and handles several data validation, merging, and export tasks.

```
if True:
    ## tables intact in themselves
    for which, g in noor.groupby("table_code"):
        #print((which,g))
        assert len(g.reference_code.unique())==1, (which,
print(g.to_string()))
        assert len(g.method.unique())==1, (which, print(g.to_string()))
        assert len(g["eval"].unique()) == 1, (which, print(g.to_string()))
        assert len(g.EC.unique()) == 1, (which, print(g.to_string()))
        assert len(g.enzyme_name.unique())==1, (which,
print(g.to_string()))
        assert len(g.reaction.unique()) == 1, (which,
print(g.to_string()))
        assert len(g.reaction.unique()) == 1, (which,
print(g.to_string()))
```

Checks that all values for every table code are consistent within each other.

```
if True:
    assert sum(manual_table_codes.duplicated(["part", "page", "col l/r",
"table from top"])) == 0, print(
        manual_table_codes[manual_table_codes.duplicated(["part", "page",
"col l/r", "table from top"])])
```

Ensures that there are no duplicates or errors in the manual table codes.

```
# split into tables with table codes from Noor and those which needed to
be annotated manually
manual_table_codes = manual_table_codes.drop(["reference", "description"],
axis="columns")
tmp_with_table_codes = tmp[~tmp.table_code.isna()]
tmp_without_table_codes = tmp[tmp.table_code.isna()]
tmp_without_table_codes = tmp_without_table_codes.drop("table_code",
axis="columns")
```

```
tmp_without_table_codes_try_to_add_manual_ones =
pandas.merge(tmp_without_table_codes, manual_table_codes, how="left",
on=["part","page","col l/r","table from top"])
# concat the two
new = pandas.concat([tmp_with_table_codes,
tmp_without_table_codes_try_to_add_manual_ones], ignore_index=True)
## keep only one entry per table code, remove now-meaningless columns, but
keep id=NaN rows
new = new[~new.duplicated(["part","page","col l/r","table from top"])]
new = new.drop(["id","url"], axis="columns")
new["comment"] = ""
```

For rows missing table codes, the script attempts to annotate them using manual data, while adding one extra column for comments during next curation steps.

```
selector = []
for i,s in new.iterrows():
    if pandas.isna(s.table_code):
        if len(manual_table_codes[(manual_table_codes.part == s.part) &
                                  (manual_table_codes.page == s.page) &
                                  (manual_table_codes["col l/r"] == s["col
l/r"]) &
                                  (manual_table_codes["table from top"] ==
s["table from top"])
           ]) == 0:
            selector.append(i)
export = new.loc[selector]
(export[[
    "part","page","col l/r","table from top",
    "table_code",
    "reaction",
    "reference_code",
    "curator",
    ]]
 .sort_values(["part","page","col l/r","table from top"])
 .to_csv("2024-01-06-opentecr-recuration.missing_table_codes.csv",
index=False)
 )
```

Rows missing table code that are not found in manual table code lists are exported into a .csv file for additional review.

```
(tables_without_comments[[
    "part","page","col l/r","table from top",
    "reference_code",
    "manually spellchecked",
    "comment"
    ]]
 .sort_values(["part","page","col l/r","table from top"])
 .to_csv("2024-01-06-opentecr-recuration.missing-table-comments.csv",
index=False)
 )
```

Similarly rows missing table comments are extracted into a separate .csv file.