Import Modules

In [ ]:
```
# Import modules
import easygui
import nbimporter
import numpy as np
import pandas as pd
import re
import threading
import time
import xlwings as xw
import unidecode

from thefuzz import fuzz
from openpyxl.utils import column_index_from_string
from openpyxl.utils import get_column_letter
from pathlib import Path
from files import FindFiles
from transfer import ImportData, ExportData
```

In [ ]:
```
# Search for files
find_files = FindFiles()
```

Data Deduplication

In [ ]:
```
# Files and absoloute paths for database search
data_files_1 = [
    Path(r"C:\Users\OneDrive\Academic\MSc Research Project\Literature
Search\Data Collection (28-Feb-2023)\01 - Cinhal.xlsx"), \
    Path(r"C:\Users\OneDrive\Academic\MSc Research Project\Literature
Search\Data Collection (28-Feb-2023)\02 - Embase.xlsx"), \
    Path(r"C:\Users\OneDrive\Academic\MSc Research Project\Literature
Search\Data Collection (28-Feb-2023)\03 - Medline.xlsx"), \
    Path(r"C:\Users\OneDrive\Academic\MSc Research Project\Literature
Search\Data Collection (28-Feb-2023)\04 - Scopus.xlsx"), \
    Path(r"C:\Users\OneDrive\Academic\MSc Research Project\Literature
Search\Data Collection (28-Feb-2023)\05 - Web of Science.xlsx"), \
            ]

# Instantiate ImportData class
```

```
import_data_1 = ImportData(files_for_import = data_files_1, \
                           worksheet_index = 1, \
                           headings_data_row = 1, \
                           index_column = None, \
                           right_data_column = "L",
                           has_spacing_columns = True,
                           column_to_copy = False)

# Create a dataframe
import_data_1.make_dataframes()
dataframe_1 = import_data_1.dataframe
```

In [ ]:
```
# Files and absoloute paths for references screen
data_files_2 = [
    Path(r"C:\Users\OneDrive\Academic\MSc Research Project\Literature
Search\Data Collection (28-Feb-2023)\06 - References Screen.xlsx"), \
             ]

# Instantiate ImportData class
import_data_2 = ImportData(files_for_import = data_files_2, \
                           worksheet_index = 1, \
                           headings_data_row = 1, \
                           index_column = None, \
                           right_data_column = "L",
                           has_spacing_columns = True,
                           column_to_copy = False)

# Create a dataframe
import_data_2.make_dataframes()
dataframe_2 = import_data_2.dataframe
```

In [ ]:
```
""" DataFrame Feltling """;
```

In [ ]:
```
class Fettling:
    def __init__(self, df, name):
        self.df = df
        self.name = name


    def add_key_column(self):
        if 'Key' in self.df.columns:
            return None
        self.df = self.df.reset_index().rename(columns={'index': 'Key'})
```

```
    def double_to_int(self):
        self.df = self.df.astype({'Year': pd.Int64Dtype(), 'Key':
pd.Int64Dtype()})


    def key_as_first_column(self):
        self.df = self.df[['Key',
                           'Author',
                           'Title',
                           'Year',
                           'Journal',
                           'Name of Database',
                           'Abstract']].copy()


    def run(self):
        self.add_key_column()
        self.double_to_int()
        self.key_as_first_column()
        export_data = ExportData(self.df, f"{self.name} {self.df.shape}")
        export_data.export_to_excel()
        return self.df


if __name__ == '__main__':
    fettling_1 = Fettling(dataframe_1, "database_results")
    dataframe_1 = fettling_1.run()
    fettling_2 = Fettling(dataframe_2, "references screen")
    dataframe_2 = fettling_2.run()
```

In [ ]:
```
""" Dataframe Merge """
dataframe_2.index = dataframe_2.index + 9000
dataframe_2["Key"] = dataframe_2.index
dataframe = pd.concat([dataframe_1, dataframe_2], axis="rows",
ignore_index=False)
```

In [ ]:
```
""" Add Missing Values """;
```

In [ ]:
```
# Files and absoloute paths
data_files_3 = [
    Path(r"C:\Users\OneDrive\Academic\MSc Research Project\Literature
```

```
Search\Data Collection (28-Feb-2023)\07 - Missing Updated.xlsx"), \
            ]

# Instantiate ImportData class
import_data_3 = ImportData(files_for_import = data_files_3, \
                           worksheet_index = 1, \
                           headings_data_row = 1, \
                           index_column = False, \
                           right_data_column = "N",
                           has_spacing_columns = True,
                           column_to_copy = "Key")

# Create a dataframe
import_data_3.make_dataframes()
df_missing_updated = import_data_3.dataframe
```

In [ ]:

```
class UpdateMissing:
    def __init__(self, primary_dataframe, df_missing_updated):
        self.primary_dataframe = primary_dataframe
        self.df_missing_updated = df_missing_updated


    def missing_values(self):
        df_missing_values =
self.primary_dataframe[self.primary_dataframe[["Author", "Title", "Year",
"Journal"]] \
                                                .isna().any(axis="columns")]
\
                                                [["Key", "Author", "Title",
"Year", "Journal", "Name of Database"]]

        export_data = ExportData(df_missing_values, f"missing_values
{df_missing_values.shape}")
        export_data.export_to_excel()


    def missing_added(self):
        export_data = ExportData(self.df_missing_updated, f"missing_added
{self.df_missing_updated.shape}")
        export_data.export_to_excel()


    def run(self):
```

```
        self.missing_values()
        self.missing_added()
        self.primary_dataframe.update(self.df_missing_updated)
        return self.primary_dataframe

if __name__ == '__main__':
    update_missing = UpdateMissing(dataframe, df_missing_updated)
    dataframe = update_missing.run()
```

In [ ]:
```
""" Deduplication """;
```

In [ ]:
```
class InitialDeduplication:
    def __init__(self, df):
        self.df = df

    def new_columns(self):
        # Replace the accented characters with non-accented characters
        self.df['Title'] = self.df['Title'].apply(lambda x:
unidecode.unidecode(x) if isinstance(x, str) else x)

        # Create a first author column
        if not 'First Author' in self.df.columns:
            self.df.insert(2, 'First Author', self.df['Author'] \
                                                .str.split().str[0] \
                                                .str.lower() \
                                                .str.strip(","))

        # Create a title lower column
        if not 'Title Lower' in self.df.columns:
            self.df.insert(4, 'Title Lower', self.df["Title"] \
                                                .str.replace(r'\d+', '',
regex=True) \
                                                .str.replace("+", " ",
regex=False) \
                                                .str.replace("]", " ",
regex=False) \
                                                .str.replace("[", " ",
regex=False) \
                                                .str.replace(":", " ",
regex=False) \
                                                .str.replace("-", " ",
regex=False) \
                                                .str.replace("()", " ",
```

```
regex=False) \
                                                .str.replace(r'\s+', ' ',
regex=True) \
                                                .str.replace("hematology",
"haematology") \
                                                .str.lower() \
                                                .str.strip())


    def originals_and_duplicates_dataframes(self):
        # Make orignals duplicates dataframe and updatae
        df_originals = self.df[~self.df.duplicated(subset=["Title Lower",
"First Author"], keep=False)] \
                            .sort_values(by=["Title Lower"], ascending=[True])
\
                            [["Key", "Title", "Title Lower", "Author", "First
Author","Year", "Journal", "Name of Database", "Abstract"]]

        df_duplicates = self.df [self.df.duplicated(subset=["Title Lower",
"First Author"], keep=False)] \
                             .sort_values(by=["Title Lower"], ascending=
[True]) \
                             [["Key", "Title", "Title Lower", "Author", "First
Author","Year", "Journal", "Name of Database", "Abstract"]]

        # Export dataframes
        export_data_1 = ExportData(df_originals, f"originals
{df_originals.shape}")
        export_data_2 = ExportData(df_duplicates, f"duplicates
{df_duplicates.shape}")
        export_data_1.export_to_excel()
        export_data_2.export_to_excel()


    def export_dataframe(self):
        self.df = self.df.drop_duplicates(subset=["Title Lower", "First
Author"], keep="first") \
                        .sort_values(by=["Key"], ascending=[True]) \
                        [["Key", "Title", "Title Lower", "Author", "First
Author", "Year", "Journal", "Name of Database", "Abstract"]] \
                        .copy()
        export_data_3 = ExportData(self.df, f"initial_deduplication
{self.df.shape}")
        export_data_3.export_to_excel()
```

```
    def run(self):
        self.new_columns()
        self.originals_and_duplicates_dataframes()
        self.export_dataframe()
        return self.df

if __name__ == '__main__':
    initial_deduplication = InitialDeduplication(dataframe)
    dataframe = initial_deduplication.run()
```

In [ ]:

```
""" Title Similarity Check """;
```

In [ ]:

```
class QueryDuplicates:
    def __init__(self, df):
        self.df = df
        self.flag_idx = 1
        self.df["Query Duplicate"] = False
        self.df["Flag Number"] = np.nan


    def similarity_check(self, target, other):
        # Find document titles with > 90% similarity
        if isinstance(target[1], float) or isinstance(other[1], float):
return None
        if target[1] == other[1]: return None
        if self.df.loc[target[0], "Query Duplicate"] == True: return None
        match_score = fuzz.ratio(target[1], other[1])

        # Add flag if match score > 90
        if match_score >= 90:
            self.df.loc[[target[0], other[0]], "Query Duplicate"] = True
            self.df.loc[[target[0], other[0]], "Flag Number"] = self.flag_idx
            self.flag_idx += 1


    def title_lower_query(self):
        for target_index, target_value in self.df["Title Lower"].iteritems():
            for other_index, other_value in self.df["Title
Lower"].iteritems():
                self.similarity_check((target_index,target_value),
(other_index, other_value))
```

```
    def run(self):
        self.title_lower_query()
        filt = self.df["Query Duplicate"] == True
        df_query_duplicates = self.df[filt].sort_values(by="Flag Number")
        export_data = ExportData(df_query_duplicates, f"query_duplicates
{df_query_duplicates.shape}")
        export_data.export_to_excel()
        return self.df


if __name__ == '__main__':
    query_duplicates = QueryDuplicates(dataframe)
    dataframe =  query_duplicates.run()
```

In [ ]:
```
""" Deduplcation Groups""";
```

In [ ]:
```
# Make query duplicate groups
duplicates_dataframe_list = [pd.DataFrame(group) for _, group in
dataframe.groupby("Flag Number")]
duplicates_dataframe_list = [df.sort_values(by="Title Lower",
ascending=True).reset_index(drop=True) for df in duplicates_dataframe_list]

# Make each duplicate group into a dataframe
for idx in range(0, len(duplicates_dataframe_list), 1):
    sheet_name = (f'{int(duplicates_dataframe_list[idx].loc[0,
"Key"])}_{duplicates_dataframe_list[idx].loc[0, "First Author"]}')

    # Export dataframes
    export_data = ExportData(duplicates_dataframe_list[idx], f"{sheet_name}")
    export_data.export_to_excel()
```

In [ ]:
```
""" Final Deduplication """;
```

In [ ]:
```
class FinalDeduplication:
    def __init__(self, df):
        self.df = df


    def edit_flag_numbers(self):
        self.df.loc[217, "Flag Number"] = 25
        self.df.loc[1125, "Flag Number"] = np.nan
```

```
        self.df.loc[1613, "Flag Number"] = np.nan


    def duplicate_on_flag_numbers(self):
        non_flagged = self.df[self.df["Flag Number"].isna()]
        flagged = self.df[~self.df["Flag Number"].isna()]

        flagged = flagged.drop_duplicates(subset=["Flag Number"],
keep="first") \
                         .sort_values(by=["First Author"], ascending=[True])
\
                         [["Key", "Title", "Title Lower", "Author", "First
Author", "Year", "Journal", "Name of Database", "Abstract"]]

        self.df = pd.concat([non_flagged, flagged],
ignore_index=False).sort_values(by="Year",ascending=True) \
                            [["Key", "Title", "Title Lower", "Author", "First
Author", "Year", "Journal", "Name of Database", "Abstract"]] \
                            .astype({'Year': pd.Int64Dtype(), 'Key':
pd.Int64Dtype()})


    def export_dataframe(self):
        export_data = ExportData(self.df, f"final_deduplication
{self.df.shape}")
        export_data.export_to_excel()


    def run(self):
        self.edit_flag_numbers()
        self.duplicate_on_flag_numbers()
        self.export_dataframe()
        return self.df


if __name__ == '__main__':
    final_deduplication = FinalDeduplication(dataframe)
    dataframe =  final_deduplication.run()
```