```
2024/8/11作成
# 商標権を年ごとに切り分けるコード
# 2000年から2023年の年度ごとに存在する商標権が分かる。
# TRdataは、商標権マスターを年度ごとにするためのクラス
# ExistDataは、2000年まで存続している商標権の登録番号と、出願番号を知るためのクラス
# MergeExsitGSDataは、2000年まで存続している商標権の区分テーブルを作成するクラス
# MergeExsitHolderDataは、2000年まで存続している商標権者のテーブルを作成するクラス
# makeNLGCSは、総務省からのデータから地域コードを付与するためのテーブルを作成するクラス
# makeName_Address_To_History_IDは、企業データの権利者名の整形と、地域コードと、hisory_idのテーブルを作るクラス
# makeHolderInterfaceは、特許庁のholderデータの権利者名の整形と、地域コードを付与するためのクラス
# HolderToHistoryIDInterfaceは、特許庁の名前と住所情報に、history_idを付与するクラス
# Connect_app_num_history_idは、NISTEPのインターフェースを用いて、history_idを付与するクラス
# NISTEPと作成したインターフェースを比較するためのクラス
# makeHistoryIDToSecIDは、historyIDからSecIDを付与するためのクラス
# CompareSecCodeは、NISTEPと作成したインターフェースを証券コードで比較するためのクラス
# TMaddSecCodeは、特許庁の商標データに証券コードを付与するためのクラス
# countTMbySecCodは、証券コードごとの商標権数をカウントするためのクラス
# custom_winsorize(series, non_neg_pct=0.025, pos_pct=0.05): ウィンザー化の関数
# process_winsorization(dataframe, column_name):ウィンザー化の検証用
# class makedNTBVDataforLinear(df1,df2): dNTBVなどがある場合の回帰分析用のパネルデータを作る。
# NISTEP企業名辞書を使用しています（NISTEP企業名辞書ver.2024_1, http://doi.org/10.15108/data_compdic001_2024_1）
```

```
#分析と表示用パッケージセット
import pandas as pd
import numpy as np
```

```
'''
class TRdata(object)

指定した年度に存在する商標権を抽出するクラス
input:upd_mgt_info_t.tsv
output:年_trademark.csv
'''
'''
class TRdata(object):
    def __init__(self, year=2000):
        # データを読み込む
        self.year = year
        self.df = self.load_df()
        self.df1 = self.preprocess_data(self.df, self.year)
        self.save_df(self.df1,self.year)

    def load_df(self, rfilename="upd_mgt_info_t.tsv"):
        df = pd.read_csv(rfilename, sep="\t", low_memory=False)
        return df

    def preprocess_data(self, df, year):
        # yearに基づくデータの前処理
        df["registration_year"] = df["set_reg_year_month_day"] // 10000

        # yearまでに抹消されていないデータの取得
        disyear = year * 10000

        # NaNデータを削除
        df = df.dropna(subset=["right_disppr_year_month_day"])

        # yearまでに登録され、yearまでに抹消されていないデータを抽出
        df1 = df.loc[
            (df["registration_year"] <= year)
            & ((df["right_disppr_year_month_day"] > disyear) | (df["right_disppr_year_month_day"] == 0)),
            ["app_num",
            "reg_num",
            "split_num",
            "app_year_month_day",
            "registration_year",
            "set_reg_year_month_day",
            "conti_prd_expire_ymd",
            "right_disppr_year_month_day",
            "pri_clim_year_month_day",
            "pri_cntry_name_cd"]
        ]
        return df1

    def save_df(self,df,year):
        self.year=year
        self.df1=df
        trfilename = f"{self.year}_trademark.csv"
        self.df1.to_csv(trfilename, index=False)
        self.df1=None
```

```
54
55 '''
```

```
'\nclass TRdata(object):\n    def __init__(self, year=2000):  \n        # データを読み込む\n        self.year = year\n        self.df = self.load_df()\n        self.df1 = self.preprocess_data(self.df, self.year)\n        self.save_df(self.df1,self.year)\n        \n    def load_df(self, rfilename="upd_mgt_info_t.tsv"):\n        df = pd.read_csv(rfilename, sep="\t", low_memory=False)\n        return df\n\n    def preprocess_data(self, df, year):\n        # yearに基づくデータの前処理\n        df["registration_year"] = df["set_reg_year_month_day"] // 10000\n\n        # yearまでに抹消されていないデータの取得\n        disyear = year * 10000\n\n        # NaNデータを削除\n        df = df.dropna(subset=["right_disppr_year_month_day"])\n\n        # yearまでに登録され、yearまでに抹消されていないデータを抽出\n        df1 = df.loc[\n            (df["registration_year"] <= year) \n            & ((df["right_disppr_year_month_day"] > disyear) | (df["right_disppr_year_month_day"] == 0)), \n            ["app_num",\n            "reg_num",\n            "split_num",\n            "app_year_month_day",\n            "registration_year",\n            "set_reg_year_month_day",\n            "conti_prd_expire_ymd",\n            "right_disppr_year_month_day",\n            "pri_clim_year_month_day",\n            "pri_cntry_name_cd"]\n        ]\n        return df1\n\n    def save_df(self,df,year):\n        self.year=year\n        self.df1=df\n        trfilename = f"{self.year}_trademark.csv"\n        self.df1.to_csv(trfilename, index=False)\n        self.df1=None\n        \n'
```

```
1 '''
2 class TRdata(object)
3
4 指定した年度に存在する商標権を抽出するクラス
5
6 2024/10/7
7 その年に抹消された商標権が入らないようにする。
8 以前は、2000年だったら、2000年に登録され、1999年中に抹消された商標権が入らないようにしていた。
9 今回の改変は、
10
11 input:upd_mgt_info_t.tsv
12 output:年_trademark.csv
13 '''
14
15 class TRdata(object):
16     def __init__(self, year=2000):
17         # データを読み込む
18         self.year = year
19         self.df = self.load_df()
20         self.df1 = self.preprocess_data(self.df, self.year)
21         self.save_df(self.df1,self.year)
22
23     def load_df(self, rfilename="upd_mgt_info_t.tsv"):
24         df = pd.read_csv(rfilename, sep="\t", low_memory=False)
25         return df
26
27     def preprocess_data(self, df, year):
28         # yearに基づくデータの前処理
29         df["registration_year"] = df["set_reg_year_month_day"] // 10000
30
31         # yearまでに抹消されていないデータの取得
32         disyear = (year +1)* 10000
33
34         # NaNデータを削除
35         df = df.dropna(subset=["right_disppr_year_month_day"])
36
37         # yearまでに登録され、yearまでに抹消されていないデータを抽出
38         df1 = df.loc[
39             (df["registration_year"] <= year)
40             & ((df["right_disppr_year_month_day"] > disyear) | (df["right_disppr_year_month_day"] == 0)),
41             ["app_num",
42             "reg_num",
43             "split_num",
44             "app_year_month_day",
45             "registration_year",
46             "set_reg_year_month_day",
47             "conti_prd_expire_ymd",
48             "right_disppr_year_month_day",
49             "pri_clim_year_month_day",
50             "pri_cntry_name_cd"]
51         ]
52         return df1
53
54     def save_df(self,df,year):
55         self.year=year
56         self.df1=df
57         trfilename = f"{self.year}_trademark.csv"
58         self.df1.to_csv(trfilename, index=False)
59         self.df1=None
60
61
```

```
1 '''
2 data = TRdata(2000)
3
4 df1=data.preprocess_data(data.df, 2001)
```

```
5
6 data.save_df(df1,2001)
7 #2002から2023のデータを作成
8 for year in range(2002, 2024):
9     df1=data.preprocess_data(data.df, year)
10    data.save_df(df1,year)
11 data=None
12 '''
```

```
1 '''
2 年（defalt 2000年）以降に存在する商標権を選ぶクラス。
3 TRdataの後に使う
4 区分数や権利者の処理を軽くするために、使用
5 input:upd_trademark.csv
6 output:年_Exist_Trademark.csv
7 '''
8 class ExistData(TRdata):
9     def __init__(self, year=2000):
10        # データを読み込む
11        self.year = year
12        self.df = self.load_df()
13        self.df1 = self.preprocess_data(self.df, self.year)
14        self.save_df()
15
16    def load_df(self, rfilename="upd_mgt_info_t.tsv"):
17        df = pd.read_csv(rfilename, sep="\t", low_memory=False)
18        return df
19
20    def preprocess_data(self, df, year):
21        # yearに基づくデータの前処理
22        df["registration_year"] = df["set_reg_year_month_day"] // 10000
23
24        # yearまでに抹消されていないデータの取得
25        disyear = year * 10000
26
27        # NaNデータを削除
28        df = df.dropna(subset=["right_disppr_year_month_day"])
29
30        # yearまでに抹消されていないデータを抽出
31        df1 = df.loc[
32           (df["right_disppr_year_month_day"] > disyear) | (df["right_disppr_year_month_day"] == 0),
33            ["app_num", "reg_num"]
34        ]
35        return df1
36
37    def save_df(self):
38        trfilename = f"{self.year}_Exist_Trademark.csv"
39        self.df1.to_csv(trfilename, index=False)
40        self.df1 = None  # データを保存した後に df1 を None に設定してメモリを解放
41
```

```
1 '''
2 #加工のための、2000年までの商標権データを取得
3 data = ExistData(2000)
4
5 '''
```

```
1 '''
2 2000年以降に生存している商品・役務のリストを作るためのクラス
3 クラスExistData()を使用した後に使う。
4
5 input df1:upd_goods_class_art.tsv
6 input df2:2000_Exist_Trademark.csv
7 output:年(2000)_Exist_CGS.csv
8 '''
9 class MergeExsitGSData(object):
10    def __init__(self):
11        self.df1 = self.load_df()
12        self.df2 = self.load_ExistData()
13        self.df = self.preprocess_data(self.df1, self.df2)
14        self.save_df(self.df)
15
16    def load_df(self, rfilename="upd_goods_class_art.tsv"):
17        df1 = pd.read_csv(rfilename, sep="\t", low_memory=False)
18        return df1
19
20    def load_ExistData(self, rfilename="2000_Exist_Trademark.csv"):
21        df2 = pd.read_csv(rfilename, low_memory=False)
22        return df2
23
24    def preprocess_data(self, df1, df2):
```

```
        # 生存商標データ（df2）の reg_num をセットに変換
        df2_reg_nums = set(df2['reg_num'])

        # 商品・役務テーブル(df1) に exist_flag 列を作成
        df1['exist_flag'] = df1['reg_num'].apply(lambda x: 1 if x in df2_reg_nums else 0)

        # 生存商標（exist_flag = 1）の行を抽出して df を作成
        df = df1[df1['exist_flag'] == 1]

        return(df)

    def save_df(self,df,year=2000):
        self.year=year
        self.df=df

        trfilename = f"{self.year}_Exist_CGS.csv"
        self.df.to_csv(trfilename, index=False)

        self.df=None
        self.df1=None
        self.df2=None
```

```
'''
class MergeExsitHolderData

2000年以降に生存している権利者のリストを作るためのクラス
クラスExistData()を使用した後に使う。

input:upd_right_person_art_t.tsv
input:2000_Exist_Trademark.csv
output:年_Exist_Holder.csv
'''
class MergeExsitHolderData(object):
    def __init__(self):
        self.df1 = self.load_df()
        self.df2 = self.load_ExistData()
        self.df = self.preprocess_data(self.df1, self.df2)
        self.save_df(self.df)

    def load_df(self, rfilename="upd_right_person_art_t.tsv"):
        df1 = pd.read_csv(rfilename, sep="\t", low_memory=False)
        return df1

    def load_ExistData(self, rfilename="2000_Exist_Trademark.csv"):
        df2 = pd.read_csv(rfilename, low_memory=False)
        return df2

    def preprocess_data(self, df1, df2):
        # 生存商標データ（df2）の reg_num をセットに変換
        df2_reg_nums = set(df2['reg_num'])

        # 権利者テーブル(df1) に exist_flag 列を作成
        df1['exist_flag'] = df1['reg_num'].apply(lambda x: 1 if x in df2_reg_nums else 0)

        # 生存商標（exist_flag = 1）の行を抽出し、必要な列だけを選択して df を作成
        df = df1[df1['exist_flag'] == 1][['reg_num', 'app_num', 'rec_num', 'pe_num',
                                          'right_psn_art_upd_ymd', 'right_person_appl_id',
                                          'right_person_addr', 'right_person_name', 'exist_flag']]

        return df

    def save_df(self,df,year=2000):
        self.year=year
        self.df=df

        trfilename = f"{self.year}_Exist_Holder.csv"
        self.df.to_csv(trfilename, index=False)

        self.df=None
        self.df1=None
        self.df2=None
```

```
'''
data = MergeExsitHolderData()
'''
```

```
'''
総務省のデータから地域コードのテーブルを作る。
https://www.soumu.go.jp/denshijiti/code.html
input:NLGCS.csv
```

```
output:NLGCS_Interface.csv

NLGCS.csvは、以下の形状
   団体コード 都道府県名  市区町村名  都道府県名（カナ）  市区町村名（カナ）
1   010006  北海道       ﾎｯｶｲﾄﾞｳ
2   011002  北海道  札幌市  ﾎｯｶｲﾄﾞｳ ｻｯﾎﾟﾛｼ
3   012025  北海道  函館市  ﾎｯｶｲﾄﾞｳ ﾊｺﾀﾞﾃｼ
4   012033  北海道  小樽市  ﾎｯｶｲﾄﾞｳ ｵﾀﾙｼ
5   012041  北海道  旭川市  ﾎｯｶｲﾄﾞｳ ｱｻﾋｶﾜｼ
6   012050  北海道  室蘭市  ﾎｯｶｲﾄﾞｳ ﾑﾛﾗﾝｼ

この形に変更しないと動かない。
'''
class makeNLGCS(object):
    def __init__(self):
        # データのロード
        df = self.load_df()
        # データの前処理
        processed_df = self.preprocess_data(df)
        # 処理済みデータの保存
        self.save_df(processed_df)

    def load_df(self):
        df = pd.read_csv("NLGCS.csv", low_memory=False)
        return df

    def preprocess_data(self, df):
        # 新しい列を作成
        df['code'] = df['団体コード'].astype(str).str[:5]
        df['address'] = df['都道府県名'].fillna('') + df['市区町村名'].fillna('')
        df['municipality'] = df['市区町村名']
        df['prefecture'] = df['都道府県名']

        # 必要な列だけを選択
        df1 = df[['code', 'address', 'municipality', 'prefecture']]

        return df1

    def save_df(self, df):
        # 新しいファイルに保存
        df.to_csv("NLGCS_Interface.csv", index=False)
```

```
'''
商標権者のデーターに地域コードを付与するためのクラス　旧バージョン
クラス MergeExsitHolderDataと、makeNLGCSで作ったデータを使用

input df:2000_Exist_Holder.csv
input df1:NLGCS_Interface.csv
output:2000_Holder_Interface.csv
'''
import pandas as pd
import re

class makeHolderInterface(object):
    def __init__(self):
        # データのロード
        df = self.load_df("2000_Exist_Holder.csv")
        df1 = self.load_df("NLGCS_Interface.csv")
        input_name = "right_person_name"
        output_name = "check_comp_name"

        # データの前処理
        self.processed_df = self.preprocess_data(df, df1)

        # 名前を直す
        self.processed_df[output_name] = self.processed_df[input_name].apply(self.remove_special_characters)
        self.processed_df[output_name] = self.processed_df[output_name].apply(self.convert_characters)
        self.processed_df = self.change_name(self.processed_df)

        # 処理済みデータの保存
        self.save_df(self.processed_df)

    def load_df(self, filename):
        df = pd.read_csv(filename, low_memory=False)
        return df

    def preprocess_data(self, df, df1):
        # 商標権者名と住所でグループ化
        df=df.groupby(['right_person_name', 'right_person_addr']).size().reset_index(name='count')

        # 多い順に並べ替え
        df = df.sort_values(by='count', ascending=False).reset_index(drop=True)
```

```
        # 地域コードを初期化
        df['code'] = 0

        # step1: address を比較して code を設定
        for i, addr in df.iterrows():
            match = df1[df1['address'] == addr['right_person_addr']]
            if not match.empty:
                df.at[i, 'code'] = match['code'].values[0]

        # step2: address で一致しなかった場合に municipality を比較して code を設定
        for i, addr in df.iterrows():
            if df.at[i, 'code'] == 0:
                match = df1[df1['municipality'] == addr['right_person_addr']]
                if not match.empty:
                    df.at[i, 'code'] = match['code'].values[0]

        return df

    def remove_special_characters(self, text):
        text = str(text)  # 数値を文字列に変換

        # 株式会社を先に削除
        text = re.sub(r'株式会社', '', text)

        pattern1 = r'(\?C)|(\?ｳ)|(\?F)|(\?ﾚ)|(\?@)|(\?H)|(\?ｵ)|(\?ｳ)|(\?T)|(\?b)|(\?c)|(\?z)|[　？▲▼■‘“”〃「」]'
        pattern2 = r'[ー-\-－—]'

        cleaned_text = re.sub(pattern1, '', text)
        cleaned_text = re.sub(pattern2, '—', cleaned_text)

        return cleaned_text

    def convert_characters(self, text):
        mapping = {
            'ャ': 'ヤ',
            'ュ': 'ユ',
            'ョ': 'ヨ',
            'ッ': 'ツ',
            'ァ': 'ア',
            'ィ': 'イ',
            'ヂ': 'ジ'
        }
        if isinstance(text, str):  # 文字列データのみ処理
            for key, value in mapping.items():
                text = text.replace(key, value)
        return text

    def change_name(self, df1):
        # 列の順序を並び替え
        df2 = df1[['right_person_name', 'check_comp_name', 'code', 'right_person_addr', 'count']]

        # 列名を変更
        df2.columns = ['original_name', 'check_comp_name', 'address_code', 'address', 'count']

        return df2

    def save_df(self, df):
        # 新しいファイルに保存
        df.to_csv("2000_Holder_Interface.csv", index=False)

    def namelist(self, strings, name='check_comp_name'):
        # 文字列がリストでない場合、リストに変換
        # 元の名前を調べたいときには、'original_name'
        if isinstance(strings, str):
            strings = [strings]

        # フィルタリングを開始
        filtered_df = self.processed_df
        for string in strings:
            filtered_df = filtered_df[filtered_df[name].str.contains(string)]

        return filtered_df
```

```
# data=makeHolderInterface()
```

```
'''
商標権者のデーターに地域コードを付与するためのクラス
処理を軽くするため、株式会社のみに絞ることにした。2024/8/15
クラス MergeExsitHolderDataと、makeNLGCSで作ったデータを使用
```

```

input df:2000_Exist_Holder.csv
input df1:NLGCS_Interface.csv
output:2000_Holder_Interface.csv

'''
import pandas as pd
import re

class makeHoldersInterface(object):
    def __init__(self):
        # データのロード
        df = self.load_df("2000_Exist_Holder.csv")
        df1 = self.load_df("NLGCS_Interface.csv")
        input_name = "right_person_name"
        output_name = "check_comp_name"

        # データの前処理
        self.processed_df = self.preprocess_data(df, df1)

        # 名前を直す
        self.processed_df[output_name] = self.processed_df[input_name].apply(self.remove_special_characters)
        self.processed_df[output_name] = self.processed_df[output_name].apply(self.convert_characters)
        self.processed_df = self.change_name(self.processed_df)

        # 処理済みデータの保存
        self.save_df(self.processed_df)

    def load_df(self, filename):
        df = pd.read_csv(filename, low_memory=False)
        return df

    def preprocess_data(self, df, df1):
        # "right_person_name" 列に NaN が含まれている行を除外
        df = df.dropna(subset=["right_person_name"])

        # "original_name" 列に "株式会社" が含まれている行を抽出
        df = df[df["right_person_name"].str.contains("株式会社")]

        # 商標権者名と住所でグループ化
        df=df.groupby(['right_person_name', 'right_person_addr']).size().reset_index(name='count')

        # 多い順に並べ替え
        df = df.sort_values(by='count', ascending=False).reset_index(drop=True)

        # 地域コードを初期化
        df['code'] = 0

        # step1: address を比較して code を設定
        for i, addr in df.iterrows():
            match = df1[df1['address'] == addr['right_person_addr']]
            if not match.empty:
                df.at[i, 'code'] = match['code'].values[0]

        # step2: address で一致しなかった場合に municipality を比較して code を設定
        for i, addr in df.iterrows():
            if df.at[i, 'code'] == 0:
                match = df1[df1['municipality'] == addr['right_person_addr']]
                if not match.empty:
                    df.at[i, 'code'] = match['code'].values[0]

        return df

    def remove_special_characters(self, text):
        text = str(text)  # 数値を文字列に変換

        # 株式会社を先に削除
        text = re.sub(r'株式会社', '', text)

        pattern1 = r'(\?C)|(\?ヤ)|(\?F)|(\?レ)|(\?@)|(\?H)|(\?ォ)|(\?ヤ)|(\?T)|(\?b)|(\?c)|(\?z)|[　？▲▼■ぅ“”〃〃「」]'
        pattern2 = r'[一-\-ー—]'

        cleaned_text = re.sub(pattern1, '', text)
        cleaned_text = re.sub(pattern2, '—', cleaned_text)

        return cleaned_text

    def convert_characters(self, text):
        mapping = {
            'ャ': 'ヤ',
            'ュ': 'ユ',
            'ョ': 'ヨ',
```

```
            'ッ': 'ツ',
            'ァ': 'ア',
            'ィ': 'イ',
            'ヂ': 'ジ'
        }
        if isinstance(text, str):  # 文字列データのみ処理
            for key, value in mapping.items():
                text = text.replace(key, value)
        return text

    def change_name(self, df1):
        # 列の順序を並び替え
        df2 = df1[['right_person_name', 'check_comp_name', 'code', 'right_person_addr', 'count']]

        # 列名を変更
        df2.columns = ['original_name', 'check_comp_name', 'address_code', 'address', 'count']

        return df2

    def save_df(self, df):
        # 新しいファイルに保存
        df.to_csv("2000_Holders_Interface.csv", index=False)

    def namelist(self, strings, name='check_comp_name'):
        # 文字列がリストでない場合、リストに変換
        # 元の名前を調べたいときには、'original_name'
        if isinstance(strings, str):
            strings = [strings]

        # フィルタリングを開始
        filtered_df = self.processed_df
        for string in strings:
            filtered_df = filtered_df[filtered_df[name].str.contains(string)]

        return filtered_df

```

```
'''
NISTEPの企業TBLから企業名と地域コードとcomp_idとのhistory_idのInterfaceを作るクラス

input df:1_comp_name_main_TBL.txt
input df1:3_address_TBL.txt
output:Name_Address_To_History_ID_Interface.csv
'''
import pandas as pd
import re

class makeName_Address_To_History_ID(object):
    def __init__(self, input_name='comp_name', output_name='check_comp_name'):
        # ファイルのロード
        self.df = self.load_df("1_comp_name_main_TBL.txt")
        self.df1 = self.load_df("3_address_TBL.txt")
        self.input_name = input_name
        self.output_name = output_name

        # データの前処理
        self.processed_df = self.preprocess_data(self.df, self.df1)

        # 名前を修正
        self.processed_df[output_name] = self.processed_df[input_name].apply(self.remove_special_characters)
        self.processed_df[output_name] = self.processed_df[output_name].apply(self.convert_characters)
        self.processed_df = self.change_name(self.processed_df)

        # 処理済みデータの保存
        self.save_df(self.processed_df)

    def load_df(self, filename):
        try:
            df = pd.read_csv(filename, sep="\t", low_memory=False)
        except FileNotFoundError:
            print(f"File {filename} not found.")
            return pd.DataFrame()
        print(df.dtypes)  # データ型を確認
        return df

    def preprocess_data(self, df, df1):
        # 'comp_id'でdfとdf1を結合し、'city_code'が存在する場合のみその値を使用
        df1_grouped = df1.groupby('comp_id').first().reset_index()  # 'comp_id' でグループ化し、最初の 'city_code' を取得
        df2 = pd.merge(df, df1_grouped[['comp_id', 'city_code']], on='comp_id', how='left')

        # 'city_code'に値がある行だけを選択
        df2 = df2[df2['city_code'].notna()]
```

```
        # 必要なカラムのみを選択
        df2 = df2[['comp_id', 'history_id', self.input_name, 'city_code', 'reg_date', 'up_date']]

        return df2

    def remove_special_characters(self, text):
        text = str(text)  # 数値を文字列に変換

        # 「株式会社」を先に削除
        text = re.sub(r'株式会社', '', text)

        # 特殊文字を削除
        pattern1 = r'(\?C)|(\?ヶ)|(\?F)|(\?レ)|(\?@)|(\?H)|(\?ォ)|(\?ヶ)|(\?T)|(\?b)|(\?c)|(\?z)|[　？▲▼■‘’“”〃「」]'
        pattern2 = r'[ー-\-－—]'

        cleaned_text = re.sub(pattern1, '', text)
        cleaned_text = re.sub(pattern2, 'ー', cleaned_text)

        return cleaned_text

    def convert_characters(self, text):
        mapping = {
            'ヤ': 'ヤ',
            'ユ': 'ユ',
            'ヨ': 'ヨ',
            'ッ': 'ツ',
            'ァ': 'ア',
            'ィ': 'イ',
            'ヂ': 'ジ'
        }
        if isinstance(text, str):  # 文字列データのみ処理
            for key, value in mapping.items():
                text = text.replace(key, value)
        return text

    def change_name(self, df):
        # 列の順序を並び替え
        df = df[[self.input_name,'comp_id', 'history_id', self.output_name, 'city_code', 'reg_date', 'up_date']]
        return df

    def save_df(self, df):
        # 処理結果を新しいファイルに保存
        df.to_csv("Name_Address_To_History_ID_Interface.csv", index=False)

# クラスを使って処理を実行
# processor = makeName_Address_To_History_ID(input_name='comp_name', output_name='check_comp_name')

```

```
'''
class HolderToHistoryIDInterface

makeHoldersInterfaceとmakeName_Address_To_History_IDとで作ったデータを使用

地域コード３桁で一致したもの
  一致　重複チェック
    重複なし　＝＞df2
    重複あり　地域コード４桁でチェック
     一致
       重複なし　=>df2
       重複あり　地域コード５桁でチェック
       一致　=>df2
 不一致　=>df3
 df2 必用な列　=>　df4

input df:2000_Holders_Interface.csv
input df1:Name_Address_To_History_ID_Interface.csv
output df2:2000_Holder_To_History_ID_Interface.csv
          一致したもの
output df3:2000_Holder_To_History_ID_Interface_diff.csv
            重複があったもの
output df4:Holders_Address_HistoryID_Interface.csv
            最終的なインターフェース

'''
import pandas as pd

#2024/8/16　イオン株式会社問題を回避？
class HolderToHistoryIDInterface:
    def __init__(self, df_path, df1_path):
        # データのロード
        self.df = pd.read_csv(df_path, low_memory=False)
```

```
        self.df1 = pd.read_csv(df1_path, low_memory=False)

        # df2, df3の作成
        self.df2, self.df3 = self.split_dataframes()

        # 列の順序を調整
        self.df2 = self.rearrange_columns(self.df2)
        self.df3 = self.rearrange_columns(self.df3)

        # df2の保存
        self.save_df(self.df2, "2000_Holder_To_History_ID_Interface.csv")

        # df3の保存
        self.save_df(self.df3, "2000_Holder_To_History_ID_Interface_diff.csv")

        # df4の作成と保存
        self.df4 = self.create_df4()
        self.save_df(self.df4, "Holders_Address_HistoryID_Interface.csv")

    def split_dataframes(self):
        # データフレームをcheck_comp_nameで結合
        df_merged = pd.merge(self.df, self.df1, on='check_comp_name', how='inner', suffixes=('', '_df1'))

        # address_codeとcity_codeの最初の3文字、4文字、完全一致の比較
        df_merged['first_three_match'] = df_merged.apply(
            lambda row: str(row['address_code']).zfill(5)[:3] == str(row['city_code']).zfill(5)[:3], axis=1)

        # 最初の3文字が一致するデータを分離
        df_three_match = df_merged[df_merged['first_three_match']].copy()
        df_no_three_match = df_merged[~df_merged['first_three_match']].copy()

        # first_three_matchで重複チェック
        df_three_match_unique = df_three_match.drop_duplicates(subset=['original_name', 'address_code', 'address'])
        df_three_match_duplicates = df_three_match[df_three_match.duplicated(subset=['original_name', 'address_code', 'address'], k

        # 重複がないものはdf2に
        df2 = df_three_match_unique.copy()

        # 重複があるものを処理 (step2)
        df_three_match_duplicates = df_three_match_duplicates.copy()
        df_three_match_duplicates['first_four_match'] = df_three_match_duplicates.apply(
            lambda row: str(row['address_code']).zfill(5)[:4] == str(row['city_code']).zfill(5)[:4], axis=1)

        df_four_match = df_three_match_duplicates[df_three_match_duplicates['first_four_match']].copy()
        df_no_four_match = df_three_match_duplicates[~df_three_match_duplicates['first_four_match']].copy()

        # first_four_matchで重複チェック
        df_four_match_unique = df_four_match.drop_duplicates(subset=['original_name', 'address_code', 'address'])
        df_four_match_duplicates = df_four_match[df_four_match.duplicated(subset=['original_name', 'address_code', 'address'], keep

        # 重複がないものはdf2に
        df2 = pd.concat([df2, df_four_match_unique])

        # 重複があるものを処理 (step3)
        df_four_match_duplicates = df_four_match_duplicates.copy()
        df_four_match_duplicates['full_match'] = df_four_match_duplicates.apply(
            lambda row: str(row['address_code']).zfill(5) == str(row['city_code']).zfill(5), axis=1)

        df_full_match = df_four_match_duplicates[df_four_match_duplicates['full_match']].copy()
        df_no_full_match = df_four_match_duplicates[~df_four_match_duplicates['full_match']].copy()

        # full_matchはdf2に、残りはdf3に
        df2 = pd.concat([df2, df_full_match])
        df3 = pd.concat([df_no_three_match, df_no_four_match, df_no_full_match])

        return df2, df3

    def rearrange_columns(self, df):
        # 必要な列を再配置
        columns = list(df.columns)
        # address_codeの次にcity_codeを配置
        if 'address_code' in columns and 'city_code' in columns:
            columns.remove('city_code')
            address_code_index = columns.index('address_code')
            columns.insert(address_code_index + 1, 'city_code')

        return df[columns]

    def create_df4(self):
        # df2から original_name, address, history_id 列を抽出
        df4 = self.df2[['original_name', 'address', 'history_id']].copy()

```

```
        # history_id でグループ化し、並べ替える
        df4 = df4.groupby(['original_name', 'address', 'history_id'], as_index=False).size()
        df4 = df4.sort_values(by='history_id').reset_index(drop=True)

        return df4

    def save_df(self, df, filename):
        # データフレームをCSVとして保存
        df.to_csv(filename, index=False)

# ファイルパスを指定してインスタンスを作成
# interface = HolderToHistoryIDInterface("2000_Holders_Interface.csv", "Name_Address_To_History_ID_Interface.csv")

```

```
'''
比較用にNISTEPのインターフェースを使ったhistory_idを商標権データに付与
TRdataで作るデータを使用する。
出願番号=>comp_id=>history_id
年ごとの比較用のデータを作る。

input df1:年_trademark.csv
input df2:ct_comp_name_dic_vs_tradmark_ver2019_2.txt
          NISTEPのinterfaceテーブル
input df_main_tbl:1_comp_name_main_TBL.txt
output df:年_app_num_comp_id.csv
'''
import pandas as pd
class Connect_app_num_history_id:
    def __init__(self, year):
        self.year = year
        self.df1 = self.load_df1(f"{self.year}_trademark.csv")
        self.df2 = self.load_df2("ct_comp_name_dic_vs_tradmark_ver2019_2.txt")
        self.df_main_tbl = self.load_df2("1_comp_name_main_TBL.txt")
        self.df3 = self.create_df3()
        self.save_df(self.df3, f"{self.year}_app_num_comp_id.csv")

    def load_df1(self, file_name):
        # df1のロード
        df1 = pd.read_csv(file_name, low_memory=False)
        return df1

    def load_df2(self, file_name):
        # df2のロード df_main_tblのロード
        df2 = pd.read_csv(file_name, delimiter='\t', low_memory=False)
        return df2

    def create_df3(self):
        # df1とdf2をapp_num (application_number)でマージ
        df3 = pd.merge(self.df1, self.df2, left_on='app_num', right_on='application_number', how='inner')

        # 必要な列を選択して新しいデータフレームを作成
        df3 = df3[['app_num', 'reg_num', 'comp_id']]

        # df3にdf_main_tblのhistory_idをcomp_idで結合して付与
        df3 = pd.merge(df3, self.df_main_tbl[['comp_id', 'history_id']], on='comp_id', how='left')

        return df3

    def save_df(self, df, file_name):
        # データフレームをCSVとして保存
        df.to_csv(file_name, index=False)

'''
# 使用例
year = 2000
connector = Connect_app_num_history_id(year)
'''
```

```
'''
Connect_app_num_history_idと
MergeExsitHolderDataと、
HolderToHistoryIDInterfaceを使って作ったデータを使用

NISTEPのIFと自作のIFの精度を比較するためのファイルを作るためのクラス

input df1:年_app_num_comp_id.csv
        NISTEPのIFでhistory_idを付与したデータ
input df2:2000_Exist_Holder.csv
        権利者名と住所データを付与するため
input df3:Holders_Address_HistoryID_Interface.csv
        権利者名と住所データからhistory_idを付与
```

```
output:年_app_num_comp_id_with_history.csv
'''
import pandas as pd

#比較用のテーブルを作成するクラス
class makeCheckInterface:
    def __init__(self, year):
        self.year = year
        self.df1 = self.load_df(f"{self.year}_app_num_comp_id.csv")
        #self.df2 = self.load_df(f"{self.year}_Exist_Holder.csv")
        self.df2 = self.load_df("2000_Exist_Holder.csv")
        self.df3 = self.load_df("Holders_Address_HistoryID_Interface.csv")
        self.df1 = self.merge_right_person_data()
        self.df1 = self.add_if_history_id()
        self.save_df(self.df1, f"{self.year}_app_num_comp_id_with_history.csv")

    def load_df(self, file_name):
        # データフレームのロード
        df = pd.read_csv(file_name, low_memory=False)
        return df

    def merge_right_person_data(self):
        # df1のreg_numに一致するdf2のreg_numに対応するright_person_nameとright_person_addrをdf1に付与
        df1 = pd.merge(self.df1, self.df2[['reg_num', 'right_person_name', 'right_person_addr']], on='reg_num', how='left')
        return df1

    def add_if_history_id(self):
        # df1のright_person_nameとdf3のoriginal_name、df1のright_person_addrとdf3のaddressが一致した場合、history_idを付与
        df1 = self.df1.copy()

        # df1のright_person_nameとdf3のoriginal_name、df1のright_person_addrとdf3のaddressでマージ
        df1 = pd.merge(df1, self.df3[['original_name', 'address', 'history_id']],
                       left_on=['right_person_name', 'right_person_addr'],
                       right_on=['original_name', 'address'],
                       how='left')

        # history_id列が存在するかチェックしてからif_history_id列を作成
        if 'history_id' in df1.columns:
            df1['if_history_id'] = df1['history_id']
        else:
            df1['if_history_id'] = None  # マージに失敗した場合にはNoneを設定

        # 元のhistory_idとif_history_idが異なるかどうかを比較する列を追加（オプション）
        if 'history_id' in df1.columns:
            df1['history_id_match'] = df1['history_id'] == df1['if_history_id']

        # 不要な列を削除
        df1 = df1.drop(columns=['original_name', 'address', 'history_id'], errors='ignore')

        return df1

    def save_df(self, df, file_name):
        # データフレームをCSVとして保存
        df.to_csv(file_name, index=False)

'''
# 使用例
year = 2000
checker = makeCheckInterface(year)
'''
```

```
'''
makeCheckInterfaceで作ったデータの比較
NISTEPと自作のIFの類似度などを比較する。
年ごとに処理ができるので、ループして使える。

input:年_app_num_comp_id_with_history.csv
output:総行数、一致する行数、一致度
'''
import pandas as pd

class CheckInterface:
    def __init__(self, year):
        self.year = year
        self.df1 = self.load_df(f"{self.year}_app_num_comp_id_with_history.csv")
        self.compare_history_ids()

    def load_df(self, file_name):
        # データフレームのロード
        df = pd.read_csv(file_name, low_memory=False)
        return df

```

```
    def compare_history_ids(self):
        # history_id_xとhistory_id_yの個数を取得
        total_rows = len(self.df1)
        matched_rows = (self.df1['history_id_x'] == self.df1['history_id_y']).sum()

        # history_id_xとhistory_id_yが一致している%を計算
        match_percentage = (matched_rows / total_rows) * 100

        # 結果を表示
        print(f"Total Rows: {total_rows}")
        print(f"Matched Rows: {matched_rows}")
        print(f"Match Percentage: {match_percentage:.2f}%")

    def save_df(self, df, file_name):
        # データフレームをCSVとして保存
        df.to_csv(file_name, index=False)

'''
# 使用例
year = 2000
checker = CheckInterface(year)
'''
```

```
"""
class makeHistoryIDToSecID:

history_idからsec_codeを付けるためのInterfaceを作るためのクラス

input df_comp_name_main:1_comp_name_main_TBL.txt
input df_sec_code:8_sec_code_TBL.txt
output:History_id_to_Sec_id.csv

step1
 1_comp_name_main_TBL.txtからcomp_idとhistory_idを
8_sec_code_TBL.txtからcomp_idとsec_codeをとり、

 sec_code,  history_id
のテーブルを作り
History_id_to_Sec_id.csvとして保存する。
"""
import pandas as pd

class makeHistoryIDToSecID:
    def __init__(self, comp_name_main_tbl_path, sec_code_tbl_path):
        self.comp_name_main_tbl_path = comp_name_main_tbl_path
        self.sec_code_tbl_path = sec_code_tbl_path

        # データのロード
        self.df_comp_name_main = self.load_df(self.comp_name_main_tbl_path)
        self.df_sec_code = self.load_df(self.sec_code_tbl_path)

        # テーブルの作成
        self.df_result = self.create_table()

        # 重複と不要なデータを削除
        self.df_result = self.remove_duplicates_and_delisted(self.df_result)

        # CSVとして保存
        self.save_df(self.df_result, "History_id_to_Sec_id.csv")

    def load_df(self, file_path):
        # データフレームをタブ区切りでロード
        df = pd.read_csv(file_path, delimiter='\t', low_memory=False)
        return df

    def create_table(self):
        # 必要な列を選択してテーブルを作成
        df_merged = pd.merge(self.df_sec_code[['comp_id', 'sec_code','listed_date','delisted_date','reg_date','up_date']],
                             self.df_comp_name_main[['comp_id', 'history_id']],
                             on='comp_id',
                             how='inner')

        # sec_code, comp_id, history_id の順に列を並べ替え
        df_result = df_merged[['sec_code', 'history_id','listed_date','delisted_date','reg_date','up_date']]

        return df_result

    def remove_duplicates_and_delisted(self, df):
        # history_idとsec_codeの重複を削除
        df_unique = df.drop_duplicates(subset=['sec_code'])
        # delisted_dateに値がある行を削除
        df_unique = df_unique[df_unique['delisted_date']==r"\N"]
```

```
        return df_unique

    def save_df(self, df, file_name):
        # データフレームをCSVとして保存
        df.to_csv(file_name, index=False)

'''
# 使用例
processor = makeHistoryIDToSecID("1_comp_name_main_TBL.txt", "8_sec_code_TBL.txt")
'''
```

```
"""
makeCheckInterface
makeHistoryIDToSecIDとで作ったデータを使用

NISETPと自作のIFで付与したhistory_idに対してsec_codeを付与する。
比較して、総行数、一致している行数、一致度を表示する。


input df1:"year"_app_num_comp_id_with_history.csv
input df2:History_id_to_Sec_id.csv
output :"year"_compare_sec_code.csv
        総行数、一致する行数、一致度

"year"は、2000や2013などの年度なので、変化する。
ループ可能

処理

Step1
 df1にsec_code_xという列を作る
 df1のhistory_id_xとdf2のhistory_idを比較して、
 同じ値の場合、df2のsec_codeを入れる。
 history_id_xがNanやdf2に該当する値がない場合には0を入れる。

Step2
 df1にsec_code_yという列を作る
 df1のhistory_id_yとdf2のhistory_idを比較して、
 同じ値の場合、df2のsec_codeを入れる。
 history_id_yがNanやdf2に該当する値がない場合には0を入れる。

Step3
 df1のsec_code_xとsec_code_yを比較する。
 Total Rows
 Matched Rows
 Matched Persentageを調べ表示する。

 Step4
 df1をyear_compare_sec_code.csvとして保存する。

"""
import pandas as pd

class CompareSecCode:
    def __init__(self, year):
        self.year = year
        self.df1 = self.load_df(f"{self.year}_app_num_comp_id_with_history.csv")
        self.df2 = self.load_df("History_id_to_Sec_id.csv")
        self.process_data()
        self.compare_sec_codes()
        self.save_df(self.df1, f"{self.year}_compare_sec_code.csv")

    def load_df(self, file_name):
        # データフレームのロード
        df = pd.read_csv(file_name, low_memory=False)
        return df

    def process_data(self):
        # Step1: sec_code_x列を作成
        self.df1['sec_code_x'] = self.df1['history_id_x'].map(self.df2.set_index('history_id')['sec_code']).fillna(0).astype(int)

        # Step2: sec_code_y列を作成
        self.df1['sec_code_y'] = self.df1['history_id_y'].map(self.df2.set_index('history_id')['sec_code']).fillna(0).astype(int)

    def compare_sec_codes(self):
        # Step3: sec_code_xとsec_code_yの比較
        total_rows = len(self.df1)
        matched_rows = (self.df1['sec_code_x'] == self.df1['sec_code_y']).sum()
        match_percentage = (matched_rows / total_rows) * 100

        # 結果の表示
```

```
        print(f"Total Rows: {total_rows}")
        print(f"Matched Rows: {matched_rows}")
        print(f"Match Percentage: {match_percentage:.2f}%")

    def save_df(self, df, file_name):
        # データフレームをCSVとして保存
        df.to_csv(file_name, index=False)
'''
# 使用例
year = 2000
comparator = CompareSecCode(year)
'''
```

```
"""
Class   TMaddSecCode()

TRdata
MergeExsitHolderData
HolderToHistoryIDInterface
makeHistoryIDToSecIDで、作成したデータを使用

特許庁のデータに証券コードを付与するためのクラス。

input df1:"year"_trademark.csv
input df2: 2000_Exist_Holder.csv
input df3: Holders_Address_HistoryID_Interface
input df4: History_id_to_Sec_id.csv
output df:year_TMaddSecCode_Ori.csv
            結合データ
output df:year_TMaddSecCode.csv
                  証券コードが付与されたデータ

"year"は、2000や2013などの年度なので、変化する。

処理

Step1
 df1とdf2をreg_numをキーにマージしたdfを作る
 df1にdf2のright_person_addressとright_person_nameを付与する。

 なお、df1のreg_numに対応するdf2のreg_numは複数ある。

 df1
 reg_num
 1
 2

 df2
 reg_num right_person_address right_person_name
1,A,A
1,B,B
2,C,C

dfは、
 reg_num, right_person_address, right_person_name
1,A,A
1,B,B
2,C,C

といった形を想定する。

Step2
dfにhistory_idの列を作る。
 dfのright_person_address, right_person_nameと、
 df3のoriginal_name addressを比較して、
 一致した場合df3のhistory_idを付与し、dfを作る。

Step3
   dfにsec_codeの列を作る。
   dfのhistory_idと
   df４のhistory_idとを比較して、
   一致した場合、df4のsec_codeをdfに付与する。
   一致しない場合、sec_codeを0にする。

Step4
 dfをyear_TMaddSecCode_Ori.csvとして保存する。

Step5
 dfでsec_codeが0以外のものを
 year_TMaddSecCode.csvとして保存する。

Step6
```

```
df,df1,df2,df3,df4のメモリーを解放する。
df=None
"""

import pandas as pd

class TMaddSecCode:
    def __init__(self, year):
        self.year = year
        self.df1 = None
        self.df2 = None
        self.df3 = None
        self.df4 = None
        self.df = None

        # データのロード
        self.load_data()

        # ステップ1: df1とdf2のマージ
        self.merge_df1_df2()

        # ステップ2: history_idを付与
        self.add_history_id()

        # ステップ3: sec_codeを付与
        self.add_sec_code()

        # ステップ4: year_TMaddSecCode_Ori.csvとして保存
        self.save_df(f"{self.year}_TMaddSecCode_Ori.csv")

        # ステップ5: sec_codeが0以外のものを保存
        self.save_filtered_df(f"{self.year}_TMaddSecCode.csv")

        # ステップ6: メモリー解放
        self.release_memory()

    def load_data(self):
        self.df1 = pd.read_csv(f"{self.year}_trademark.csv", low_memory=False)
        self.df2 = pd.read_csv("2000_Exist_Holder.csv", low_memory=False)
        #self.df2 = pd.read_csv(f"{self.year}_Exist_Holder.csv", low_memory=False)
        self.df3 = pd.read_csv("Holders_Address_HistoryID_Interface.csv", low_memory=False)
        self.df4 = pd.read_csv("History_id_to_Sec_id.csv", low_memory=False)

    def merge_df1_df2(self):
        # df1とdf2をreg_numでマージし、right_person_addressとright_person_nameを付与
        self.df = pd.merge(self.df1, self.df2[['reg_num', 'right_person_addr', 'right_person_name']], on='reg_num', how='left')

    def add_history_id(self):
        # right_person_addressとright_person_nameでdf3のoriginal_name, addressを比較し、history_idを付与
        self.df = pd.merge(self.df, self.df3[['original_name', 'address', 'history_id']],
                           left_on=['right_person_name', 'right_person_addr'],
                           right_on=['original_name', 'address'],
                           how='left')
        self.df.drop(columns=['original_name', 'address'], inplace=True)

    def add_sec_code(self):
        # history_idでdf4のhistory_idを比較し、sec_codeを付与
        self.df['sec_code'] = self.df['history_id'].map(self.df4.set_index('history_id')['sec_code']).fillna(0).astype(int)

    def save_df(self, file_name):
        # データフレームをCSVとして保存
        self.df.to_csv(file_name, index=False)

    def save_filtered_df(self, file_name):
        # sec_codeが0以外のものを保存
        df_filtered = self.df[self.df['sec_code'] != 0]
        df_filtered.to_csv(file_name, index=False)

    def release_memory(self):
        # メモリー解放
        self.df = None
        self.df1 = None
        self.df2 = None
        self.df3 = None
        self.df4 = None

'''
# 使用例
year = 2001
processor = TMaddSecCode(year)
'''
```

```
"""
countTMbySecCode(startyear,endyear)
TMaddSecCodeで作られたデータを使用する。

証券コード別、年度ごとの商標権データの表を作成する。

input df1:year_TMaddSecCode.csv
output df:f"{self.startyear}_{self.endyear}_TMcount.csv"

"year"は、startyear(例えば、2000年)からendyear（例えば、2023年）で、変化する。

Step1
  dfにファイルを読み込む
  sec_codeごとにカウントする。

  sec_codeで昇順

  イメージ
  year, sec_code, count
  2000, 4911, 634
  2000, 5331, 10

  df_TMcountに保存する。

  print("yearの処理が終わりました。")

step2
  この処理をendyearまで繰り返す。

  例えば、2001年のデータは、2000年の下につく

  イメージ
  year, sec_code, count
  2000, 4911, 634
  2000, 5331, 10
   2001, 4911, 650

step3
  df_TMcountを
  startyear_endyear_TMcount.csvとして保存する。
  例えば、2000_2023_TMcount.csv

"""
import pandas as pd

class countTMbySecCode:
    def __init__(self, startyear, endyear):
        self.startyear = startyear
        self.endyear = endyear
        self.df_TMcount = pd.DataFrame(columns=['year', 'sec_code', 'count'])

        # 各年の処理を実行
        self.process_years()

        # 処理結果を保存
        self.save_df_TMcount()

    def process_years(self):
        for year in range(self.startyear, self.endyear + 1):
            # ステップ1: 年ごとにファイルを読み込み、sec_codeごとにカウント
            df = pd.read_csv(f"{year}_TMaddSecCode.csv")
            df_count = df.groupby('sec_code').size().reset_index(name='count')
            df_count['year'] = year

            # sec_codeで昇順に並べ替え
            df_count = df_count[['year', 'sec_code', 'count']].sort_values(by='sec_code')

            # df_TMcountにデータを追加
            self.df_TMcount = pd.concat([self.df_TMcount, df_count], ignore_index=True)

            # 処理が終わったらメッセージを表示
            print(f"{year}の処理が終わりました。")

    def save_df_TMcount(self):
        # startyear_endyear_TMcount.csv として保存
        file_name = f"{self.startyear}_{self.endyear}_TMcount.csv"
        self.df_TMcount.to_csv(file_name, index=False)
        print(f"結果を{file_name}に保存しました。")

'''
# 使用例
```

```
processor = countTMbySecCode(2000, 2023)
'''
```

```
'''
商標データとファイナンスデータをマージして回帰分析用に作成するためのクラス
countTMbySecCodeの後に使う
8/17に新バージョンがある。


import pandas as pd

class makeDataforLinear:
    def __init__(self, tmcount_path, finance_path, output_file='DataforLinear.csv'):
        self.tmcount_path = tmcount_path
        self.finance_path = finance_path
        self.output_file = output_file

        # データのロード
        self.df1 = self.load_tmcount()
        self.df2 = self.load_finance_data()

        # データの処理
        self.df3 = self.create_df3()

        # データの保存
        self.save_df(self.df3)

    def load_tmcount(self):
        # TMcountのデータをロードしてsec_codeにTを付ける
        df1 = pd.read_csv(self.tmcount_path)
        df1['sec_code'] = 'T' + df1['sec_code'].astype(str)
        return df1

    def load_finance_data(self):
        # FinanceデータをロードしてclosedDateからyearを作成
        encodings = ['utf-8', 'shift_jis', 'cp932']  # ここに使用できる可能性のあるエンコーディングを追加
        for encoding in encodings:
            try:
                df2 = pd.read_csv(self.finance_path, encoding=encoding)

                # ClosedDateの形式が "202301" や "2023/01" の場合に対応
                df2['ClosedDate'] = df2['ClosedDate'].astype(str)  # 万が一、数値型の場合に文字列に変換
                df2['ClosedDate'] = pd.to_datetime(df2['ClosedDate'],
                                        format='%Y%m', errors='coerce').fillna(pd.to_datetime(df2['ClosedDate'],
                                         format='%Y/%m', errors='coerce'))

                df2['year'] = df2['ClosedDate'].dt.year

                return df2
            except UnicodeDecodeError:
                print(f"エンコーディング {encoding} での読み込みに失敗しました。")
        raise ValueError("対応するエンコーディングが見つかりませんでした。")

    def create_df3(self):
        # df2の内容をコピーしてdf3を作成
        df3 = self.df2.copy()

        # nt列を作成して、初期値を0に設定
        df3['nt'] = 0

        # df1のsec_codeとdf2のSecCode、df1とdf2のyearが一致する場合にcountを入力
        for i, row in self.df1.iterrows():
            matching_rows = (df3['SecCode'] == row['sec_code']) & (df3['year'] == row['year'])
            df3.loc[matching_rows, 'nt'] = row['count']

        return df3

    def save_df(self, df):
        # データフレームを指定されたファイル名で保存
        df.to_csv(self.output_file, index=False)
        print(f"データが{self.output_file}として保存されました。")
'''
'''
# 使用例
file1 = '2000_2023_TMcount.csv'
file2 = 'df_EditFinTMForLinear06111ROE1.csv'
processor = makeDataforLinear(file1, file2)
'''
```

```
"""
class makeDataforLinear(df1,df2):
```

```
countTMbySecCodeによって作られたデータを使用する。
8/17バージョン

input df1:start_end_TMcount.csv
input df2:financedata.csv
output df3:DataforLinear.csv

df2の形式

    Name    SecCode ClosedDate  TobinsQ PBR DR  LnMV    Aeesets Debts   Equity  年月    MarketValue ROE SalesProfitMargin   Tur
1
SecCode ClosedDate ntが処理のために、必須

商標データとファイナンスデータをマージして回帰分析用に作成するためのクラス
year列がすでにある場合には、作らない。
たまに、Closedyearが変な場合があるので、それを回避する。
"""
import pandas as pd

class makeDataforLinear:
    def __init__(self, tmcount_path, finance_path, output_file='DataforLinear.csv'):
        self.tmcount_path = tmcount_path
        self.finance_path = finance_path
        self.output_file = output_file

        # データのロード
        self.df1 = self.load_tmcount()
        self.df2 = self.load_finance_data()

        # データの処理
        self.df3 = self.create_df3()

        # データの保存
        self.save_df(self.df3)

    def load_tmcount(self):
        # TMcountのデータをロードしてsec_codeにTを付ける
        df1 = pd.read_csv(self.tmcount_path)
        df1['sec_code'] = 'T' + df1['sec_code'].astype(str)
        return df1

    def load_finance_data(self):
        # FinanceデータをロードしてClosedDateからyearを作成
        encodings = ['utf-8', 'shift_jis', 'cp932']  # ここに使用できる可能性のあるエンコーディングを追加
        for encoding in encodings:
            try:
                df2 = pd.read_csv(self.finance_path, encoding=encoding)

                # 既にyear列が存在するかをチェック
                if 'year' not in df2.columns:
                    # ClosedDateの形式が "202301" や "2023/01" の場合に対応
                    df2['ClosedDate'] = df2['ClosedDate'].astype(str)  # 万が一、数値型の場合に文字列に変換
                    df2['ClosedDate'] = pd.to_datetime(df2['ClosedDate'],
                                       format='%Y%m', errors='coerce').fillna(pd.to_datetime(df2['ClosedDate'],
                                        format='%Y/%m', errors='coerce'))

                    df2['year'] = df2['ClosedDate'].dt.year

                return df2
            except UnicodeDecodeError:
                print(f"エンコーディング {encoding} での読み込みに失敗しました。")
        raise ValueError("対応するエンコーディングが見つかりませんでした。")



    def create_df3(self):
        # df2の内容をコピーしてdf3を作成
        df3 = self.df2.copy()

        # nt列を作成して、初期値を0に設定
        df3['nt'] = 0.0

        # df1のsec_codeとdf2のSecCode、df1とdf2のyearが一致する場合にcountを入力
        for i, row in self.df1.iterrows():
            matching_rows = (df3['SecCode'] == row['sec_code']) & (df3['year'] == row['year'])
            df3.loc[matching_rows, 'nt'] = row['count']

        return df3

    def save_df(self, df):
        # データフレームを指定されたファイル名で保存
```

```
        df.to_csv(self.output_file, index=False)
        print(f"データが{self.output_file}として保存されました。")

'''
# 使用例
file1 = '2000_2023_TMcount.csv'
file2 = 'df_EditFinTMForLinear06111ROE1.csv'
processor = makeDataforLinear(file1, file2)
'''
```

```
"\n# 使用例\nfile1 = '2000_2023_TMcount.csv'\nfile2 = 'df_EditFinTMForLinear06111ROE1.csv'\nprocessor = makeDataforLinear(file1, file2)\n"
```

```
'''
商標権が共有の場合のテーブルを作成するクラス

重複している商標、重複数、その逆数（権利のシェア）

input 年_Exist_Holder.csv
output 年_Share_Holder.csv
'''
import pandas as pd

class makeShare_Holder:
    def __init__(self, year):
        self.year = year
        self.input_file = f"{self.year}_Exist_Holder.csv"
        self.output_file = f"{self.year}_Share_Holder.csv"

    def load_df(self):
        """データフレームを読み込みます。"""
        self.df = pd.read_csv(self.input_file)

    def process_data(self):
        """reg_numの重複を解消したデータフレームを作成します。"""
        # reg_numの重複数をカウント
        df_grouped = self.df['reg_num'].value_counts().reset_index()
        df_grouped.columns = ['reg_num', 'count']

        # share列を追加 (1 / count)
        df_grouped['share'] = 1 / df_grouped['count']

        # 結果のデータフレームを保存
        self.df_processed = df_grouped

    def save_df(self):
        """処理されたデータフレームを保存します。"""
        self.df_processed.to_csv(self.output_file, index=False)

    def run(self):
        """すべての処理を実行します。"""
        self.load_df()
        self.process_data()
        self.save_df()
'''
# 使用例
year = 2000
share_holder_processor = makeShare_Holder(year)
share_holder_processor.run()

2000_Exist_Trademark.csvにあって2000_Share_Hplder.csvにないものがあるが、
元ネタ（upd_right_person_art_t.tsv）にもなかった。

2000_Share_Holder.csvに存在しないreg_num:
{4917992, 5144761, 6659036, 1010100}



'''
```

```
コーディングを開始するか、AI で生成します。
```

編集するにはダブルクリックするか Enter キーを押してください