

```

1 2024/8/11作成
2 # 商標権を年ごとに切り分けるコード
3 # 2000年から2023年の年度ごとに存在する商標権が分かる。
4 # TRdataは、商標権マスターを年度ごとにするためのクラス
5 # 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
6 # 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
7 # 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
8 # 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
9 # 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、history_idのテーブルを作るクラス
10 # 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
11 # HolderToHistoryIDInterfaceは、特許庁の名前と住所情報に、history_idを付与するクラス
12 # 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
13 # 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
14 # makeHistoryIDToSecIDは、historyIDからSecIDを付与するためのクラス
15 # 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
16 # 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
17 # 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
18 # custom_winsorize(series, non_neg_pct=0.025, pos_pct=0.05): ウィンザー化の関数
19 # process_winsorization(dataframe, column_name):ウィンザー化の検証用
20 # 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。
21 # 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)

```

```

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

```

```

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 input:upd_mgt_info_t.tsv
6 output:年_trademark.csv
7 ""
8 ""
9 class TRdata(object):
10     def __init__(self, year=2000):
11         # データを読み込む
12         self.year = year
13         self.df = self.load_df()
14         self.df1 = self.preprocess_data(self.df, self.year)
15         self.save_df(self.df1, self.year)
16
17     def load_df(self, rfilename="upd_mgt_info_t.tsv"):
18         df = pd.read_csv(rfilename, sep="\t", low_memory=False)
19         return df
20
21     def preprocess_data(self, df, year):
22         # yearに基づくデータの预处理
23         df["registration_year"] = df["set_reg_year_month_day"] // 10000
24
25         # yearまでに抹消されていないデータの取得
26         disyear = year * 10000
27
28         # NaNデータを削除
29         df = df.dropna(subset=["right_disppr_year_month_day"])
30
31         # 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
32         df1 = df.loc[
33             (df["registration_year"] <= year)
34             & ((df["right_disppr_year_month_day"] > disyear) | (df["right_disppr_year_month_day"] == 0)),
35             ["app_num",
36              "reg_num",
37              "split_num",
38              "app_year_month_day",
39              "registration_year",
40              "set_reg_year_month_day",
41              "conti_prd_expire_ymd",
42              "right_disppr_year_month_day",
43              "pri_clim_year_month_day",
44              "pri_cntry_name_cd"]
45         ]
46         return df1
47
48     def save_df(self, df, year):
49         self.year = year
50         self.df1 = df
51         trfilename = f"{self.year}_trademark.csv"
52         self.df1.to_csv(trfilename, index=False)
53         self.df1 = None

```

```
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 (default 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):

```

```

25     # 生存商標データ (df2) の reg_num をセットに変換
26     df2_reg_nums = set(df2['reg_num'])
27
28     # 商品・役務テーブル(df1) に exist_flag 列を作成
29     df1['exist_flag'] = df1['reg_num'].apply(lambda x: 1 if x in df2_reg_nums else 0)
30
31     # 生存商標 (exist_flag = 1) の行を抽出して df を作成
32     df = df1[df1['exist_flag'] == 1]
33
34     return(df)
35
36     def save_df(self,df,year=2000):
37         self.year=year
38         self.df=df
39
40         trfilename = f"{self.year}_Exist_CGS.csv"
41         self.df.to_csv(trfilename, index=False)
42
43         self.df=None
44         self.df1=None
45         self.df2=None

```

```

1  """
2  class MergeExsitHolderData
3
4  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
5  クラスExistData()を使用した後に使う。
6
7  input:upd_right_person_art_t.tsv
8  input:2000_Exist_TradeMark.csv
9  output:年_Exist_Holder.csv
10 """
11 class MergeExsitHolderData(object):
12     def __init__(self):
13         self.df1 = self.load_df()
14         self.df2 = self.load_ExistData()
15         self.df = self.preprocess_data(self.df1, self.df2)
16         self.save_df(self.df)
17
18     def load_df(self, rfilename="upd_right_person_art_t.tsv"):
19         df1 = pd.read_csv(rfilename, sep="\t", low_memory=False)
20         return df1
21
22     def load_ExistData(self, rfilename="2000_Exist_TradeMark.csv"):
23         df2 = pd.read_csv(rfilename, low_memory=False)
24         return df2
25
26     def preprocess_data(self, df1, df2):
27         # 生存商標データ (df2) の reg_num をセットに変換
28         df2_reg_nums = set(df2['reg_num'])
29
30         # 権利者テーブル(df1) に exist_flag 列を作成
31         df1['exist_flag'] = df1['reg_num'].apply(lambda x: 1 if x in df2_reg_nums else 0)
32
33         # 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
34         df = df1[df1['exist_flag'] == 1][['reg_num', 'app_num', 'rec_num', 'pe_num',
35                                         'right_psn_art_upd_ymd', 'right_person_appl_id',
36                                         'right_person_addr', 'right_person_name', 'exist_flag']]
37
38         return df
39
40     def save_df(self,df,year=2000):
41         self.year=year
42         self.df=df
43
44         trfilename = f"{self.year}_Exist_Holder.csv"
45         self.df.to_csv(trfilename, index=False)
46
47         self.df=None
48         self.df1=None
49         self.df2=None

```

```

1  """
2  data = MergeExsitHolderData()
3  """

```

```

1  """
2  総務省のデータから地域コードのテーブルを作る。
3  https://www.soumu.go.jp/denshijiti/code.html
4  input:NLGCS.csv

```

```

5 output:NLGCS_Interface.csv
6
7 NLGCS.csvは、以下の形状
8 ❖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)
9 1 010006 北海道 ホッカイドウ
10 2 011002 北海道 札幌市 ホッカイドウ サッポロ シ
11 3 012025 北海道 函館市 ホッカイドウ ハコダテ シ
12 4 012033 北海道 小樽市 ホッカイドウ オタル シ
13 5 012041 北海道 旭川市 ホッカイドウ アサヒカワ シ
14 6 012050 北海道 室蘭市 ホッカイドウ ムロワン シ
15
16 この形に変更しないと動かない。
17 """
18 class makeNLGCS(object):
19     def __init__(self):
20         # データのロード
21         df = self.load_df()
22         # データの前処理
23         processed_df = self.preprocess_data(df)
24         # 処理済みデータの保存
25         self.save_df(processed_df)
26
27     def load_df(self):
28         df = pd.read_csv("NLGCS.csv", low_memory=False)
29         return df
30
31     def preprocess_data(self, df):
32         # 新しい列を作成
33         df['code'] = df['団体コード'].astype(str).str[:5]
34         df['address'] = df['都道府県名'].fillna("") + df['市区町村名'].fillna("")
35         df['municipality'] = df['市区町村名']
36         df['prefecture'] = df['都道府県名']
37
38         # 必要な列だけを選択
39         df1 = df[['code', 'address', 'municipality', 'prefecture']]
40
41         return df1
42
43     def save_df(self, df):
44         # 新しいファイルに保存
45         df.to_csv("NLGCS_Interface.csv", index=False)

```

```

1 """
2 商標権者のデータに地域コードを付与するためのクラス 旧バージョン
3 クラス MergeExsitholderDataと、makeNLGCSで作ったデータを使用
4
5 input df:2000_Exist_Holder.csv
6 input df1:NLGCS_Interface.csv
7 output:2000_Holder_Interface.csv
8 """
9 import pandas as pd
10 import re
11
12 class makeHolderInterface(object):
13     def __init__(self):
14         # データのロード
15         df = self.load_df("2000_Exist_Holder.csv")
16         df1 = self.load_df("NLGCS_Interface.csv")
17         input_name = "right_person_name"
18         output_name = "check_comp_name"
19
20         # データの前処理
21         self.processed_df = self.preprocess_data(df, df1)
22
23         # 名前を直す
24         self.processed_df[output_name] = self.processed_df[input_name].apply(self.remove_special_characters)
25         self.processed_df[output_name] = self.processed_df[output_name].apply(self.convert_characters)
26         self.processed_df = self.change_name(self.processed_df)
27
28         # 処理済みデータの保存
29         self.save_df(self.processed_df)
30
31     def load_df(self, filename):
32         df = pd.read_csv(filename, low_memory=False)
33         return df
34
35     def preprocess_data(self, df, df1):
36         # 商標権者名と住所でグループ化
37         df=df.groupby(['right_person_name', 'right_person_addr']).size().reset_index(name='count')
38
39         # 多い順に並べ替え
40         df = df.sort_values(by='count', ascending=False).reset_index(drop=True)

```

```

41
42     # 地域コードを初期化
43     df['code'] = 0
44
45     # step1: address を比較して code を設定
46     for i, addr in df.iterrows():
47         match = df1[df1['address'] == addr['right_person_addr']]
48         if not match.empty:
49             df.at[i, 'code'] = match['code'].values[0]
50
51     # step2: address で一致しなかった場合に municipality を比較して code を設定
52     for i, addr in df.iterrows():
53         if df.at[i, 'code'] == 0:
54             match = df1[df1['municipality'] == addr['right_person_addr']]
55             if not match.empty:
56                 df.at[i, 'code'] = match['code'].values[0]
57
58     return df
59
60 def remove_special_characters(self, text):
61     text = str(text) # 数値を文字列に変換
62
63     # 株式会社を先に削除
64     text = re.sub(r'株式会社', '', text)
65
66     pattern1 = r'(\?C)|(\?r)|(\?F)|(\?l)|(\?@)|(\?H)|(\?t)|(\?r)|(\?T)|(\?b)|(\?c)|(\?z)|[ ? ▲▼■'""' // 「」]'
67     pattern2 = r'[-\-\-]'
68
69     cleaned_text = re.sub(pattern1, '', text)
70     cleaned_text = re.sub(pattern2, '-', cleaned_text)
71
72     return cleaned_text
73
74 def convert_characters(self, text):
75     mapping = {
76         'ヤ': 'ヤ',
77         'ユ': 'ユ',
78         'ヨ': 'ヨ',
79         'ッ': 'ツ',
80         'ア': 'ア',
81         'イ': 'イ',
82         'ヂ': 'ジ'
83     }
84     if isinstance(text, str): # 文字列データのみ処理
85         for key, value in mapping.items():
86             text = text.replace(key, value)
87     return text
88
89 def change_name(self, df1):
90     # 列の順序を並び替え
91     df2 = df1[['right_person_name', 'check_comp_name', 'code', 'right_person_addr', 'count']]
92
93     # 列名を変更
94     df2.columns = ['original_name', 'check_comp_name', 'address_code', 'address', 'count']
95
96     return df2
97
98 def save_df(self, df):
99     # 新しいファイルに保存
100     df.to_csv("2000_Holder_Interface.csv", index=False)
101
102 def namelist(self, strings, name='check_comp_name'):
103     # 文字列がリストでない場合、リストに変換
104     # 元の名前を調べたいときには、'original_name'
105     if isinstance(strings, str):
106         strings = [strings]
107
108     # フィルタリングを開始
109     filtered_df = self.processed_df
110     for string in strings:
111         filtered_df = filtered_df[filtered_df[name].str.contains(string)]
112
113     return filtered_df
114

```

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

```

1 ""
2 商標権者のデータに地域コードを付与するためのクラス
3 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
4 クラス MergeExsitHolderDataと、makeNLGCSで作ったデータを使用

```

```

5
6 input df:2000_Exist_Holder.csv
7 input df1:NLGCS_Interface.csv
8 output:2000_Holder_Interface.csv
9
10 ""
11 import pandas as pd
12 import re
13
14 class makeHoldersInterface(object):
15     def __init__(self):
16         # データのロード
17         df = self.load_df("2000_Exist_Holder.csv")
18         df1 = self.load_df("NLGCS_Interface.csv")
19         input_name = "right_person_name"
20         output_name = "check_comp_name"
21
22         # データの前処理
23         self.processed_df = self.preprocess_data(df, df1)
24
25         # 名前を直す
26         self.processed_df[output_name] = self.processed_df[input_name].apply(self.remove_special_characters)
27         self.processed_df[output_name] = self.processed_df[output_name].apply(self.convert_characters)
28         self.processed_df = self.change_name(self.processed_df)
29
30         # 処理済みデータの保存
31         self.save_df(self.processed_df)
32
33     def load_df(self, filename):
34         df = pd.read_csv(filename, low_memory=False)
35         return df
36
37     def preprocess_data(self, df, df1):
38         # "right_person_name" 列に NaN が含まれている行を除外
39         df = df.dropna(subset=["right_person_name"])
40
41         # "original_name" 列に "株式会社" が含まれている行を抽出
42         df = df[df["right_person_name"].str.contains("株式会社")]
43
44         # 商標権者名と住所でグループ化
45         df=df.groupby(['right_person_name', 'right_person_addr']).size().reset_index(name='count')
46
47         # 多い順に並べ替え
48         df = df.sort_values(by='count', ascending=False).reset_index(drop=True)
49
50         # 地域コードを初期化
51         df['code'] = 0
52
53         # step1: address を比較して code を設定
54         for i, addr in df.iterrows():
55             match = df1[df1['address'] == addr['right_person_addr']]
56             if not match.empty:
57                 df.at[i, 'code'] = match['code'].values[0]
58
59         # step2: address で一致しなかった場合に municipality を比較して code を設定
60         for i, addr in df.iterrows():
61             if df.at[i, 'code'] == 0:
62                 match = df1[df1['municipality'] == addr['right_person_addr']]
63                 if not match.empty:
64                     df.at[i, 'code'] = match['code'].values[0]
65
66         return df
67
68     def remove_special_characters(self, text):
69         text = str(text) # 数値を文字列に変換
70
71         # 株式会社を先に削除
72         text = re.sub(r'株式会社', "", text)
73
74         pattern1 = r'(\?C)|(\?r)|(\?F)|(\?l)|(\?@)|(\?H)|(\?t)|(\?r)|(\?T)|(\?b)|(\?c)|(\?z)[ \? ▲▼■""""' / 「」]'
75         pattern2 = r'[-\-\—]'
76
77         cleaned_text = re.sub(pattern1, "", text)
78         cleaned_text = re.sub(pattern2, '-', cleaned_text)
79
80         return cleaned_text
81
82     def convert_characters(self, text):
83         mapping = {
84             'ヤ': 'ヤ',
85             'ユ': 'ユ',
86             'ヨ': 'ヨ',

```

```

87         'ッ': 'ツ',
88         'ア': 'ア',
89         'イ': 'イ',
90         'ヂ': 'ジ'
91     }
92     if isinstance(text, str): # 文字列データのみ処理
93         for key, value in mapping.items():
94             text = text.replace(key, value)
95     return text
96
97     def change_name(self, df1):
98         # 列の順序を並び替え
99         df2 = df1[['right_person_name', 'check_comp_name', 'code', 'right_person_addr', 'count']]
100
101         # 列名を変更
102         df2.columns = ['original_name', 'check_comp_name', 'address_code', 'address', 'count']
103
104         return df2
105
106     def save_df(self, df):
107         # 新しいファイルに保存
108         df.to_csv("2000_Holders_Interface.csv", index=False)
109
110     def namelist(self, strings, name='check_comp_name'):
111         # 文字列がリストでない場合、リストに変換
112         # 元の名前を調べたいときには、'original_name'
113         if isinstance(strings, str):
114             strings = [strings]
115
116         # フィルタリングを開始
117         filtered_df = self.processed_df
118         for string in strings:
119             filtered_df = filtered_df[filtered_df[name].str.contains(string)]
120
121         return filtered_df
122

```

```

1  """
2  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
3
4  input df:1_comp_name_main_TBL.txt
5  input df:3_address_TBL.txt
6  output:Name_Address_To_History_ID_Interface.csv
7  """
8  import pandas as pd
9  import re
10
11  class makeName_Address_To_History_ID(object):
12      def __init__(self, input_name='comp_name', output_name='check_comp_name'):
13          # ファイルのロード
14          self.df = self.load_df("1_comp_name_main_TBL.txt")
15          self.df1 = self.load_df("3_address_TBL.txt")
16          self.input_name = input_name
17          self.output_name = output_name
18
19          # データの前処理
20          self.processed_df = self.preprocess_data(self.df, self.df1)
21
22          # 名前を修正
23          self.processed_df[output_name] = self.processed_df[input_name].apply(self.remove_special_characters)
24          self.processed_df[output_name] = self.processed_df[output_name].apply(self.convert_characters)
25          self.processed_df = self.change_name(self.processed_df)
26
27          # 処理済みデータの保存
28          self.save_df(self.processed_df)
29
30      def load_df(self, filename):
31          try:
32              df = pd.read_csv(filename, sep="\t", low_memory=False)
33          except FileNotFoundError:
34              print(f"File {filename} not found.")
35              return pd.DataFrame()
36          print(df.dtypes) # データ型を確認
37          return df
38
39      def preprocess_data(self, df, df1):
40          # '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
41          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
42          df2 = pd.merge(df, df1_grouped[['comp_id', 'city_code']], on='comp_id', how='left')
43
44          # 'city_code'に値がある行だけを選択
45          df2 = df2[df2['city_code'].notna()]

```

```

1  """
2  class HolderToHistoryIDInterface
3
4  makeHoldersInterfaceとmakeName_Address_To_History_IDとで作ったデータを使用
5
6  地域コード 3 桁で一致したもの
7      一致 重複チェック
8          重複なし =>df2
9          重複あり 地域コード 4 桁でチェック
10         一致
11             重複なし =>df2
12             重複あり 地域コード 5 桁でチェック
13             一致 =>df2
14     不一致 =>df3
15     df2 必用な列 => df4
16
17 input df:2000_Holders_Interface.csv
18 input df1:Name_Address_To_History_ID_Interface.csv
19 output df2:2000_Holder_To_History_ID_Interface.csv
20     一致したもの
21 output df3:2000_Holder_To_History_ID_Interface_diff.csv
22     重複があったもの
23 output df4:Holders_Address_HistoryID_Interface.csv
24     最終的なインターフェース
25
26 """
27 import pandas as pd
28
29 #2024/8/16 イオン株式会社問題を回避？
30 class HolderToHistoryIDInterface:
31     def __init__(self, df_path, df1_path):
32         # データのロード
33         self.df = pd.read_csv(df_path, low_memory=False)

```

```

34     self.df1 = pd.read_csv(df1_path, low_memory=False)
35
36     # df2, df3の作成
37     self.df2, self.df3 = self.split_dataframes()
38
39     # 列の順序を調整
40     self.df2 = self.rearrange_columns(self.df2)
41     self.df3 = self.rearrange_columns(self.df3)
42
43     # df2の保存
44     self.save_df(self.df2, "2000_Holder_To_History_ID_Interface.csv")
45
46     # df3の保存
47     self.save_df(self.df3, "2000_Holder_To_History_ID_Interface_diff.csv")
48
49     # df4の作成と保存
50     self.df4 = self.create_df4()
51     self.save_df(self.df4, "Holders_Address_HistoryID_Interface.csv")
52
53     def split_dataframes(self):
54         # データフレームをcheck_comp_nameで結合
55         df_merged = pd.merge(self.df, self.df1, on='check_comp_name', how='inner', suffixes=('_', 'df1'))
56
57         # address_codeとcity_codeの最初の3文字、4文字、完全一致の比較
58         df_merged['first_three_match'] = df_merged.apply(
59             lambda row: str(row['address_code']).zfill(5)[:3] == str(row['city_code']).zfill(5)[:3], axis=1)
60
61         # 最初の3文字が一致するデータを分離
62         df_three_match = df_merged[df_merged['first_three_match']].copy()
63         df_no_three_match = df_merged[~df_merged['first_three_match']].copy()
64
65         # first_three_matchで重複チェック
66         df_three_match_unique = df_three_match.drop_duplicates(subset=['original_name', 'address_code', 'address'])
67         df_three_match_duplicates = df_three_match[df_three_match.duplicated(subset=['original_name', 'address_code', 'address'], k
68
69         # 重複がないものはdf2に
70         df2 = df_three_match_unique.copy()
71
72         # 重複があるものを処理 (step2)
73         df_three_match_duplicates = df_three_match_duplicates.copy()
74         df_three_match_duplicates['first_four_match'] = df_three_match_duplicates.apply(
75             lambda row: str(row['address_code']).zfill(5)[:4] == str(row['city_code']).zfill(5)[:4], axis=1)
76
77         df_four_match = df_three_match_duplicates[df_three_match_duplicates['first_four_match']].copy()
78         df_no_four_match = df_three_match_duplicates[~df_three_match_duplicates['first_four_match']].copy()
79
80         # first_four_matchで重複チェック
81         df_four_match_unique = df_four_match.drop_duplicates(subset=['original_name', 'address_code', 'address'])
82         df_four_match_duplicates = df_four_match[df_four_match.duplicated(subset=['original_name', 'address_code', 'address'], keep
83
84         # 重複がないものはdf2に
85         df2 = pd.concat([df2, df_four_match_unique])
86
87         # 重複があるものを処理 (step3)
88         df_four_match_duplicates = df_four_match_duplicates.copy()
89         df_four_match_duplicates['full_match'] = df_four_match_duplicates.apply(
90             lambda row: str(row['address_code']).zfill(5) == str(row['city_code']).zfill(5), axis=1)
91
92         df_full_match = df_four_match_duplicates[df_four_match_duplicates['full_match']].copy()
93         df_no_full_match = df_four_match_duplicates[~df_four_match_duplicates['full_match']].copy()
94
95         # full_matchはdf2に、残りはdf3に
96         df2 = pd.concat([df2, df_full_match])
97         df3 = pd.concat([df_no_three_match, df_no_four_match, df_no_full_match])
98
99         return df2, df3
100
101     def rearrange_columns(self, df):
102         # 必要な列を再配置
103         columns = list(df.columns)
104         # address_codeの次にcity_codeを配置
105         if 'address_code' in columns and 'city_code' in columns:
106             columns.remove('city_code')
107             address_code_index = columns.index('address_code')
108             columns.insert(address_code_index + 1, 'city_code')
109
110         return df[columns]
111
112     def create_df4(self):
113         # df2から original_name, address, history_id 列を抽出
114         df4 = self.df2[['original_name', 'address', 'history_id']].copy()
115

```

```

116     # history_id でグループ化し、並べ替える
117     df4 = df4.groupby(['original_name', 'address', 'history_id'], as_index=False).size()
118     df4 = df4.sort_values(by='history_id').reset_index(drop=True)
119
120     return df4
121
122     def save_df(self, df, filename):
123         # データフレームをCSVとして保存
124         df.to_csv(filename, index=False)
125
126 # ファイルパスを指定してインスタンスを作成
127 # interface = HolderToHistoryIDInterface("2000_Holders_Interface.csv", "Name_Address_To_History_ID_Interface.csv")
128

```

```

1 """
2 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
3  TRdataで作るデータを使用する。
4  出願番号=>comp_id=>history_id
5  年ごとの比較用のデータを作る。
6
7  input df1:年_trademark.csv
8  input df2:ct_comp_name_dic_vs_trademark_ver2019_2.txt
9      NISTEPのinterfaceテーブル
10 input df_main_tbl:1_comp_name_main_TBL.txt
11 output df:年_app_num_comp_id.csv
12 """
13 import pandas as pd
14 class Connect_app_num_history_id:
15     def __init__(self, year):
16         self.year = year
17         self.df1 = self.load_df1(f"{self.year}_trademark.csv")
18         self.df2 = self.load_df2("ct_comp_name_dic_vs_trademark_ver2019_2.txt")
19         self.df_main_tbl = self.load_df2("1_comp_name_main_TBL.txt")
20         self.df3 = self.create_df3()
21         self.save_df(self.df3, f"{self.year}_app_num_comp_id.csv")
22
23     def load_df1(self, file_name):
24         # df1のロード
25         df1 = pd.read_csv(file_name, low_memory=False)
26         return df1
27
28     def load_df2(self, file_name):
29         # df2のロード df_main_tblのロード
30         df2 = pd.read_csv(file_name, delimiter='\t', low_memory=False)
31         return df2
32
33     def create_df3(self):
34         # df1とdf2をapp_num (application_number)でマージ
35         df3 = pd.merge(self.df1, self.df2, left_on='app_num', right_on='application_number', how='inner')
36
37         # 必要な列を選択して新しいデータフレームを作成
38         df3 = df3[['app_num', 'reg_num', 'comp_id']]
39
40         # df3にdf_main_tblのhistory_idをcomp_idで結合して付与
41         df3 = pd.merge(df3, self.df_main_tbl[['comp_id', 'history_id']], on='comp_id', how='left')
42
43         return df3
44
45     def save_df(self, df, file_name):
46         # データフレームをCSVとして保存
47         df.to_csv(file_name, index=False)
48
49 """
50 # 使用例
51 year = 2000
52 connector = Connect_app_num_history_id(year)
53 """

```

```

1 """
2 Connect_app_num_history_idと
3 MergeExistHolderDataと、
4 HolderToHistoryIDInterfaceを使って作ったデータを使用
5
6 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
7
8 input df1:年_app_num_comp_id.csv
9      NISTEPのIFでhistory_idを付与したデータ
10 input df2:2000_Exist_Holder.csv
11     権利者名と住所データを付与するため
12 input df3:Holders_Address_HistoryID_Interface.csv
13     権利者名と住所データからhistory_idを付与

```

```

14 output:年_app_num_comp_id_with_history.csv
15 """
16 import pandas as pd
17
18 #比較用のテーブルを作成するクラス
19 class makeCheckInterface:
20     def __init__(self, year):
21         self.year = year
22         self.df1 = self.load_df(f"{self.year}_app_num_comp_id.csv")
23         #self.df2 = self.load_df(f"{self.year}_Exist_Holder.csv")
24         self.df2 = self.load_df("2000_Exist_Holder.csv")
25         self.df3 = self.load_df("Holders_Address_HistoryID_Interface.csv")
26         self.df1 = self.merge_right_person_data()
27         self.df1 = self.add_if_history_id()
28         self.save_df(self.df1, f"{self.year}_app_num_comp_id_with_history.csv")
29
30     def load_df(self, file_name):
31         # データフレームのロード
32         df = pd.read_csv(file_name, low_memory=False)
33         return df
34
35     def merge_right_person_data(self):
36         # 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
37         df1 = pd.merge(self.df1, self.df2[['reg_num', 'right_person_name', 'right_person_addr']], on='reg_num', how='left')
38         return df1
39
40     def add_if_history_id(self):
41         # 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
42         df1 = self.df1.copy()
43
44         # df1のright_person_nameとdf3のoriginal_name、df1のright_person_addrとdf3のaddressでマージ
45         df1 = pd.merge(df1, self.df3[['original_name', 'address', 'history_id']],
46                        left_on=['right_person_name', 'right_person_addr'],
47                        right_on=['original_name', 'address'],
48                        how='left')
49
50         # history_id列が存在するかチェックしてからif_history_id列を作成
51         if 'history_id' in df1.columns:
52             df1['if_history_id'] = df1['history_id']
53         else:
54             df1['if_history_id'] = None # マージに失敗した場合にはNoneを設定
55
56         # 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)
57         if 'history_id' in df1.columns:
58             df1['history_id_match'] = df1['history_id'] == df1['if_history_id']
59
60         # 不要な列を削除
61         df1 = df1.drop(columns=['original_name', 'address', 'history_id'], errors='ignore')
62
63         return df1
64
65     def save_df(self, df, file_name):
66         # データフレームをCSVとして保存
67         df.to_csv(file_name, index=False)
68
69 """
70 # 使用例
71 year = 2000
72 checker = makeCheckInterface(year)
73 """

```

```

1 """
2 makeCheckInterfaceで作ったデータの比較
3 NISTEPと自作のIFの類似度などを比較する。
4 年ごとに処理ができるので、ループして使える。
5
6 input:年_app_num_comp_id_with_history.csv
7 output:総行数、一致する行数、一致度
8 """
9 import pandas as pd
10
11 class CheckInterface:
12     def __init__(self, year):
13         self.year = year
14         self.df1 = self.load_df(f"{self.year}_app_num_comp_id_with_history.csv")
15         self.compare_history_ids()
16
17     def load_df(self, file_name):
18         # データフレームのロード
19         df = pd.read_csv(file_name, low_memory=False)
20         return df
21

```

```

22 def compare_history_ids(self):
23     # history_id_xとhistory_id_yの個数を取得
24     total_rows = len(self.df1)
25     matched_rows = (self.df1['history_id_x'] == self.df1['history_id_y']).sum()
26
27     # history_id_xとhistory_id_yが一致している%を計算
28     match_percentage = (matched_rows / total_rows) * 100
29
30     # 結果を表示
31     print(f"Total Rows: {total_rows}")
32     print(f"Matched Rows: {matched_rows}")
33     print(f"Match Percentage: {match_percentage:.2f}%")
34
35 def save_df(self, df, file_name):
36     # データフレームをCSVとして保存
37     df.to_csv(file_name, index=False)
38
39 """
40 # 使用例
41 year = 2000
42 checker = CheckInterface(year)
43 """

```

```

1 """
2 class makeHistoryIDToSecID:
3
4     history_idからsec_codeを付けるためのInterfaceを作るためのクラス
5
6     input df_comp_name_main:1_comp_name_main_TBL.txt
7     input df_sec_code:8_sec_code_TBL.txt
8     output:History_id_to_Sec_id.csv
9
10    step1
11    1_comp_name_main_TBL.txtからcomp_idとhistory_idを
12    8_sec_code_TBL.txtからcomp_idとsec_codeをとり、
13
14    sec_code, history_id
15    のテーブルを作り
16    History_id_to_Sec_id.csvとして保存する。
17 """
18 import pandas as pd
19
20 class makeHistoryIDToSecID:
21     def __init__(self, comp_name_main_tbl_path, sec_code_tbl_path):
22         self.comp_name_main_tbl_path = comp_name_main_tbl_path
23         self.sec_code_tbl_path = sec_code_tbl_path
24
25         # データのロード
26         self.df_comp_name_main = self.load_df(self.comp_name_main_tbl_path)
27         self.df_sec_code = self.load_df(self.sec_code_tbl_path)
28
29         # テーブルの作成
30         self.df_result = self.create_table()
31
32         # 重複と不要なデータを削除
33         self.df_result = self.remove_duplicates_and_delisted(self.df_result)
34
35         # CSVとして保存
36         self.save_df(self.df_result, "History_id_to_Sec_id.csv")
37
38     def load_df(self, file_path):
39         # データフレームをタブ区切りでロード
40         df = pd.read_csv(file_path, delimiter='\t', low_memory=False)
41         return df
42
43     def create_table(self):
44         # 必要な列を選択してテーブルを作成
45         df_merged = pd.merge(self.df_sec_code[['comp_id', 'sec_code', 'listed_date', 'delisted_date', 'reg_date', 'up_date']],
46                             self.df_comp_name_main[['comp_id', 'history_id']],
47                             on='comp_id',
48                             how='inner')
49
50         # sec_code, comp_id, history_id の順に列を並べ替え
51         df_result = df_merged[['sec_code', 'history_id', 'listed_date', 'delisted_date', 'reg_date', 'up_date']]
52
53         return df_result
54
55     def remove_duplicates_and_delisted(self, df):
56         # history_idとsec_codeの重複を削除
57         df_unique = df.drop_duplicates(subset=['sec_code'])
58         # delisted_dateに値がある行を削除
59         df_unique = df_unique[df_unique['delisted_date'] == r"\N"]

```

```

60
61     return df_unique
62
63     def save_df(self, df, file_name):
64         # データフレームをCSVとして保存
65         df.to_csv(file_name, index=False)
66
67     """
68     # 使用例
69     processor = makeHistoryIDToSecID("1_comp_name_main_TBL.txt", "8_sec_code_TBL.txt")
70     """

```

```

1     """
2     makeCheckInterface
3     makeHistoryIDToSecIDとで作ったデータを使用
4
5     NISETPと自作のIFで付与したhistory_idに対してsec_codeを付与する。
6     比較して、総行数、一致している行数、一致度を表示する。
7
8
9     input df1:"year"_app_num_comp_id_with_history.csv
10    input df2:History_id_to_Sec_id.csv
11    output :"year"_compare_sec_code.csv
12        総行数、一致する行数、一致度
13
14    "year"は、2000や2013などの年度なので、変化する。
15    ループ可能
16
17    処理
18
19    Step1
20    df1にsec_code_xという列を作る
21    df1のhistory_id_xとdf2のhistory_idを比較して、
22    同じ値の場合、df2のsec_codeを入れる。
23    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。
24
25    Step2
26    df1にsec_code_yという列を作る
27    df1のhistory_id_yとdf2のhistory_idを比較して、
28    同じ値の場合、df2のsec_codeを入れる。
29    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。
30
31    Step3
32    df1のsec_code_xとsec_code_yを比較する。
33    Total Rows
34    Matched Rows
35    Matched Percentageを調べ表示する。
36
37    Step4
38    df1をyear_compare_sec_code.csvとして保存する。
39
40    """
41    import pandas as pd
42
43    class CompareSecCode:
44        def __init__(self, year):
45            self.year = year
46            self.df1 = self.load_df(f"{self.year}_app_num_comp_id_with_history.csv")
47            self.df2 = self.load_df("History_id_to_Sec_id.csv")
48            self.process_data()
49            self.compare_sec_codes()
50            self.save_df(self.df1, f"{self.year}_compare_sec_code.csv")
51
52        def load_df(self, file_name):
53            # データフレームのロード
54            df = pd.read_csv(file_name, low_memory=False)
55            return df
56
57        def process_data(self):
58            # Step1: sec_code_x列を作成
59            self.df1["sec_code_x"] = self.df1["history_id_x"].map(self.df2.set_index("history_id")["sec_code"]).fillna(0).astype(int)
60
61            # Step2: sec_code_y列を作成
62            self.df1["sec_code_y"] = self.df1["history_id_y"].map(self.df2.set_index("history_id")["sec_code"]).fillna(0).astype(int)
63
64        def compare_sec_codes(self):
65            # Step3: sec_code_xとsec_code_yの比較
66            total_rows = len(self.df1)
67            matched_rows = (self.df1["sec_code_x"] == self.df1["sec_code_y"]).sum()
68            match_percentage = (matched_rows / total_rows) * 100
69
70            # 結果の表示

```

```

71     print(f"Total Rows: {total_rows}")
72     print(f"Matched Rows: {matched_rows}")
73     print(f"Match Percentage: {match_percentage:.2f}%")
74
75     def save_df(self, df, file_name):
76         # データフレームをCSVとして保存
77         df.to_csv(file_name, index=False)
78     """
79 # 使用例
80 year = 2000
81 comparator = CompareSecCode(year)
82 """

```

```

1  """
2  Class  TMaddSecCode()
3
4  TRdata
5  MergeExsitHolderData
6  HolderToHistoryIDInterface
7  makeHistoryIDToSecIDで、作成したデータを使用
8
9  特許庁のデータに証券コードを付与するためのクラス。
10
11 input df1:"year"_trademark.csv
12 input df2: 2000_Exist_Holder.csv
13 input df3: Holders_Address_HistoryID_Interface
14 input df4: History_id_to_Sec_id.csv
15 output df:year_TMaddSecCode_Ori.csv
16     結合データ
17 output df:year_TMaddSecCode.csv
18     証券コードが付与されたデータ
19
20 "year"は、2000や2013などの年度なので、変化する。
21
22 処理
23
24 Step1
25 df1とdf2をreg_numをキーにマージしたdfを作る
26 df1にdf2のright_person_addressとright_person_nameを付与する。
27
28 なお、df1のreg_numに対応するdf2のreg_numは複数ある。
29
30 df1
31   reg_num
32   1
33   2
34
35 df2
36   reg_num right_person_address right_person_name
37   1,A,A
38   1,B,B
39   2,C,C
40
41 dfは、
42   reg_num, right_person_address, right_person_name
43   1,A,A
44   1,B,B
45   2,C,C
46
47   といった形を想定する。
48
49 Step2
50 dfにhistory_idの列を作る。
51 dfのright_person_address, right_person_nameと、
52 df3のoriginal_name addressを比較して、
53 一致した場合df3のhistory_idを付与し、dfを作る。
54
55 Step3
56 dfにsec_codeの列を作る。
57 dfのhistory_idと
58 df4のhistory_idとを比較して、
59 一致した場合、df4のsec_codeをdfに付与する。
60 一致しない場合、sec_codeを0にする。
61
62 Step4
63 dfをyear_TMaddSecCode_Ori.csvとして保存する。
64
65 Step5
66 dfでsec_codeが0以外のものを
67 year_TMaddSecCode.csvとして保存する。
68
69 Step6

```

```

70 df,df1,df2,df3,df4のメモリーを解放する。
71 df=None
72 """
73
74 import pandas as pd
75
76 class TMaddSecCode:
77     def __init__(self, year):
78         self.year = year
79         self.df1 = None
80         self.df2 = None
81         self.df3 = None
82         self.df4 = None
83         self.df = None
84
85         # データのロード
86         self.load_data()
87
88         # ステップ1: df1とdf2のマージ
89         self.merge_df1_df2()
90
91         # ステップ2: history_idを付与
92         self.add_history_id()
93
94         # ステップ3: sec_codeを付与
95         self.add_sec_code()
96
97         # ステップ4: year_TMaddSecCode_Ori.csvとして保存
98         self.save_df(f"{self.year}_TMaddSecCode_Ori.csv")
99
100        # ステップ5: sec_codeが0以外のものを保存
101        self.save_filtered_df(f"{self.year}_TMaddSecCode.csv")
102
103        # ステップ6: メモリー解放
104        self.release_memory()
105
106    def load_data(self):
107        self.df1 = pd.read_csv(f"{self.year}_trademark.csv", low_memory=False)
108        self.df2 = pd.read_csv("2000_Exist_Holder.csv", low_memory=False)
109        #self.df2 = pd.read_csv(f"{self.year}_Exist_Holder.csv", low_memory=False)
110        self.df3 = pd.read_csv("Holders_Address_HistoryID_Interface.csv", low_memory=False)
111        self.df4 = pd.read_csv("History_id_to_Sec_id.csv", low_memory=False)
112
113    def merge_df1_df2(self):
114        # df1とdf2をreg_numでマージし、right_person_addressとright_person_nameを付与
115        self.df = pd.merge(self.df1, self.df2[['reg_num', 'right_person_addr', 'right_person_name']], on='reg_num', how='left')
116
117    def add_history_id(self):
118        # right_person_addressとright_person_nameでdf3のoriginal_name, addressを比較し、history_idを付与
119        self.df = pd.merge(self.df, self.df3[['original_name', 'address', 'history_id']],
120                           left_on=['right_person_name', 'right_person_addr'],
121                           right_on=['original_name', 'address'],
122                           how='left')
123        self.df.drop(columns=['original_name', 'address'], inplace=True)
124
125    def add_sec_code(self):
126        # history_idでdf4のhistory_idを比較し、sec_codeを付与
127        self.df['sec_code'] = self.df['history_id'].map(self.df4.set_index('history_id')['sec_code']).fillna(0).astype(int)
128
129    def save_df(self, file_name):
130        # データフレームをCSVとして保存
131        self.df.to_csv(file_name, index=False)
132
133    def save_filtered_df(self, file_name):
134        # sec_codeが0以外のものを保存
135        df_filtered = self.df[self.df['sec_code'] != 0]
136        df_filtered.to_csv(file_name, index=False)
137
138    def release_memory(self):
139        # メモリー解放
140        self.df = None
141        self.df1 = None
142        self.df2 = None
143        self.df3 = None
144        self.df4 = None
145
146    ""
147    # 使用例
148    year = 2001
149    processor = TMaddSecCode(year)
150    ""

```

```

1  """
2  countTMbySecCode(startyear,endyear)
3  TMaddSecCodeで作られたデータを使用する。
4
5  証券コード別、年度ごとの商標権データの表を作成する。
6
7  input df1:year_TMaddSecCode.csv
8  output df:f"{self.startyear}_{self.endyear}_TMcount.csv"
9
10 "year"は、startyear(例えば、2000年)からendyear (例えば、2023年) で、変化する。
11
12 Step1
13   dfにファイルを読み込む
14   sec_codeごとにカウントする。
15
16   sec_codeで昇順
17
18   イメージ
19   year, sec_code, count
20   2000, 4911, 634
21   2000, 5331, 10
22
23   df_TMcountに保存する。
24
25   print("yearの処理が終わりました。")
26
27 step2
28   この処理をendyearまで繰り返す。
29
30   例えば、2001年のデータは、2000年の下につく
31
32   イメージ
33   year, sec_code, count
34   2000, 4911, 634
35   2000, 5331, 10
36   2001, 4911, 650
37
38 step3
39   df_TMcountを
40   startyear_endyear_TMcount.csvとして保存する。
41   例えば、2000_2023_TMcount.csv
42
43 """
44 import pandas as pd
45
46 class countTMbySecCode:
47     def __init__(self, startyear, endyear):
48         self.startyear = startyear
49         self.endyear = endyear
50         self.df_TMcount = pd.DataFrame(columns=['year', 'sec_code', 'count'])
51
52         # 各年の処理を実行
53         self.process_years()
54
55         # 処理結果を保存
56         self.save_df_TMcount()
57
58     def process_years(self):
59         for year in range(self.startyear, self.endyear + 1):
60             # 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
61             df = pd.read_csv(f"{year}_TMaddSecCode.csv")
62             df_count = df.groupby('sec_code').size().reset_index(name='count')
63             df_count['year'] = year
64
65             # sec_codeで昇順に並べ替え
66             df_count = df_count[['year', 'sec_code', 'count']].sort_values(by='sec_code')
67
68             # df_TMcountにデータを追加
69             self.df_TMcount = pd.concat([self.df_TMcount, df_count], ignore_index=True)
70
71             # 処理が終わったらメッセージを表示
72             print(f"{year}の処理が終わりました。")
73
74     def save_df_TMcount(self):
75         # startyear_endyear_TMcount.csv として保存
76         file_name = f"{self.startyear}_{self.endyear}_TMcount.csv"
77         self.df_TMcount.to_csv(file_name, index=False)
78         print(f"結果を{file_name}に保存しました。")
79
80 ""
81 # 使用例

```

```
82 processor = countTMbySecCode(2000, 2023)
83 ""
```

```
1 ""
2 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
3 countTMbySecCodeの後に使う
4 8/17に新バージョンがある。
5
6
7 import pandas as pd
8
9 class makeDataforLinear:
10     def __init__(self, tmcount_path, finance_path, output_file='DataforLinear.csv'):
11         self.tmcount_path = tmcount_path
12         self.finance_path = finance_path
13         self.output_file = output_file
14
15         # データのロード
16         self.df1 = self.load_tmcount()
17         self.df2 = self.load_finance_data()
18
19         # データの処理
20         self.df3 = self.create_df3()
21
22         # データの保存
23         self.save_df(self.df3)
24
25     def load_tmcount(self):
26         # TMcountのデータをロードしてsec_codeにTを付ける
27         df1 = pd.read_csv(self.tmcount_path)
28         df1['sec_code'] = 'T' + df1['sec_code'].astype(str)
29         return df1
30
31     def load_finance_data(self):
32         # FinanceデータをロードしてClosedDateからyearを作成
33         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
34         for encoding in encodings:
35             try:
36                 df2 = pd.read_csv(self.finance_path, encoding=encoding)
37
38                 # ClosedDateの形式が "202301" や "2023/01" の場合に対応
39                 df2['ClosedDate'] = df2['ClosedDate'].astype(str) # 万が一、数値型の場合に文字列に変換
40                 df2['ClosedDate'] = pd.to_datetime(df2['ClosedDate'],
41                                                     format='%Y%m', errors='coerce').fillna(pd.to_datetime(df2['ClosedDate'],
42                                                         format='%Y/%m', errors='coerce'))
43
44                 df2['year'] = df2['ClosedDate'].dt.year
45
46                 return df2
47             except UnicodeDecodeError:
48                 print(f"エンコーディング {encoding} での読み込みに失敗しました。")
49                 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。")
50
51     def create_df3(self):
52         # df2の内容をコピーしてdf3を作成
53         df3 = self.df2.copy()
54
55         # nt列を作成して、初期値を0に設定
56         df3['nt'] = 0
57
58         # 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
59         for i, row in self.df1.iterrows():
60             matching_rows = (df3['SecCode'] == row['sec_code']) & (df3['year'] == row['year'])
61             df3.loc[matching_rows, 'nt'] = row['count']
62
63         return df3
64
65     def save_df(self, df):
66         # データフレームを指定されたファイル名で保存
67         df.to_csv(self.output_file, index=False)
68         print(f"データが{self.output_file}として保存されました。")
69 ""
70 ""
71 # 使用例
72 file1 = '2000_2023_TMcount.csv'
73 file2 = 'df_EditFinTMForLinear06111ROE1.csv'
74 processor = makeDataforLinear(file1, file2)
75 ""
```

```
1 """
2 class makeDataforLinear(df1,df2):
```

```

3
4 countTMbySecCodeによって作られたデータを使用する。
5 8/17バージョン
6
7 input df1:start_end_TMcount.csv
8 input df2:financedata.csv
9 output df3:DataforLinear.csv
10
11 df2の形式
12
13 Name SecCode ClosedDate TobinsQ PBR DR LnMV Aeeseets Debts Equity 年月 MarketValue ROE SalesProfitMargin Tur
14 1
15 SecCode ClosedDate ntが処理のために、必須
16
17 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
18 year列がすでにある場合には、作らない。
19 たまに、Closedyearが変な場合があるので、それを回避する。
20 ""
21 import pandas as pd
22
23 class makeDataforLinear:
24     def __init__(self, tmcount_path, finance_path, output_file='DataforLinear.csv'):
25         self.tmcount_path = tmcount_path
26         self.finance_path = finance_path
27         self.output_file = output_file
28
29         # データのロード
30         self.df1 = self.load_tmcount()
31         self.df2 = self.load_finance_data()
32
33         # データの処理
34         self.df3 = self.create_df3()
35
36         # データの保存
37         self.save_df(self.df3)
38
39     def load_tmcount(self):
40         # TMcountのデータをロードしてsec_codeにTを付ける
41         df1 = pd.read_csv(self.tmcount_path)
42         df1['sec_code'] = 'T' + df1['sec_code'].astype(str)
43         return df1
44
45     def load_finance_data(self):
46         # FinanceデータをロードしてClosedDateからyearを作成
47         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
48         for encoding in encodings:
49             try:
50                 df2 = pd.read_csv(self.finance_path, encoding=encoding)
51
52                 # 既にyear列が存在するかをチェック
53                 if 'year' not in df2.columns:
54                     # ClosedDateの形式が "202301" や "2023/01" の場合に対応
55                     df2['ClosedDate'] = df2['ClosedDate'].astype(str) # 万が一、数値型の場合に文字列に変換
56                     df2['ClosedDate'] = pd.to_datetime(df2['ClosedDate'],
57                                                         format='%Y%m', errors='coerce').fillna(pd.to_datetime(df2['ClosedDate'],
58                                                         format='%Y/%m', errors='coerce'))
59
60                     df2['year'] = df2['ClosedDate'].dt.year
61
62                 return df2
63             except UnicodeDecodeError:
64                 print(f"エンコーディング {encoding} での読み込みに失敗しました。")
65                 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。")
66
67
68
69     def create_df3(self):
70         # df2の内容をコピーしてdf3を作成
71         df3 = self.df2.copy()
72
73         # nt列を作成して、初期値を0に設定
74         df3['nt'] = 0.0
75
76         # 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
77         for i, row in self.df1.iterrows():
78             matching_rows = (df3['SecCode'] == row['sec_code']) & (df3['year'] == row['year'])
79             df3.loc[matching_rows, 'nt'] = row['count']
80
81         return df3
82
83     def save_df(self, df):
84         # データフレームを指定されたファイル名で保存

```

```
85     df.to_csv(self.output_file, index=False)
86     print(f"データが{self.output_file}として保存されました。")
87
88 """
89 # 使用例
90 file1 = '2000_2023_TMcount.csv'
91 file2 = 'df_EditFinTMForLinear06111ROE1.csv'
92 processor = makeDataforLinear(file1, file2)
93 """
```

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

```
1 """
2 商標権が共有の場合のテーブルを作成するクラス
3
4 重複している商標、重複数、その逆数（権利のシェア）
5
6 input 年_Exist_Holder.csv
7 output 年_Share_Holder.csv
8 """
9 import pandas as pd
10
11 class makeShare_Holder:
12     def __init__(self, year):
13         self.year = year
14         self.input_file = f"{self.year}_Exist_Holder.csv"
15         self.output_file = f"{self.year}_Share_Holder.csv"
16
17     def load_df(self):
18         """データフレームを読み込みます。"""
19         self.df = pd.read_csv(self.input_file)
20
21     def process_data(self):
22         """reg_numの重複を解消したデータフレームを作成します。"""
23         # reg_numの重複数をカウント
24         df_grouped = self.df['reg_num'].value_counts().reset_index()
25         df_grouped.columns = ['reg_num', 'count']
26
27         # share列を追加 (1 / count)
28         df_grouped['share'] = 1 / df_grouped['count']
29
30         # 結果のデータフレームを保存
31         self.df_processed = df_grouped
32
33     def save_df(self):
34         """処理されたデータフレームを保存します。"""
35         self.df_processed.to_csv(self.output_file, index=False)
36
37     def run(self):
38         """すべての処理を実行します。"""
39         self.load_df()
40         self.process_data()
41         self.save_df()
42 """
43 # 使用例
44 year = 2000
45 share_holder_processor = makeShare_Holder(year)
46 share_holder_processor.run()
47
48 2000_Exist_Trademark.csvにあって2000_Share_Holder.csvにないものがあるが、
49 元ネタ（upd_right_person_art_t.tsv）にもなかった。
50
51 2000_Share_Holder.csvに存在しないreg_num:
52 {4917992, 5144761, 6659036, 1010100}
53
54
55
56 """
```

1 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

