In [1]:

```
! pip install PyAthena

import io
import boto3
import pandas as pd
import os
from pyathena import connect
from pyathena.pandas.util import as_pandas
from pyathena.pandas.cursor import PandasCursor

## connect Athena with s3 & test
cursor = connect(s3_staging_dir='s3://query-results-bucket-athena-2021/',region_name='ap-northeast-2',cursor_class=PandasCursor).cursor()
```

```
Collecting PyAthena
  Downloading PyAthena-2.3.0-py3-none-any.whl (37 kB)
Requirement already satisfied: botocore>=1.5.52 in /home/ec2-user/anaconda3/envs/python3/lib/python3.6/site-packages (from PyAthena) (1.20.83)
Requirement already satisfied: boto3>=1.4.4 in /home/ec2-user/anaconda3/envs/python3/lib/python3.6/site-packages (from PyAthena) (1.17.83)
Collecting tenacity>=4.1.0
  Downloading tenacity-7.0.0-py2.py3-none-any.whl (23 kB)
Requirement already satisfied: s3transfer<0.5.0,>=0.4.0 in /home/ec2-user/anaconda3/envs/python3/lib/python3.6/site-packages (from boto3>=1.4.4
->PyAthena) (0.4.2)
Requirement already satisfied: jmespath<1.0.0,>=0.7.1 in /home/ec2-user/anaconda3/envs/python3/lib/python3.6/site-packages (from boto3>=1.4.4-
>PyAthena) (0.10.0)
Requirement already satisfied: urllib3<1.27,>=1.25.4 in /home/ec2-user/anaconda3/envs/python3/lib/python3.6/site-packages (from botocore>=1.5.5
2->PyAthena) (1.26.5)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /home/ec2-user/anaconda3/envs/python3/lib/python3.6/site-packages (from botocore>
=1.5.52->PyAthena) (2.8.1)
Requirement already satisfied: six>=1.5 in /home/ec2-user/anaconda3/envs/python3/lib/python3.6/site-packages (from python-dateutil<3.0.0,>=2.1->
botocore>=1.5.52->PyAthena) (1.15.0)
Installing collected packages: tenacity, PyAthena
Successfully installed PyAthena-2.3.0 tenacity-7.0.0
```

In [2]:

```
################################# prameters   :          ####################################
start_yr = '1997'
end_yr = '2010'

start_date = start_yr + '-01-01'
end_date = end_yr + '-12-31'

ipc_n = '500'
ranking = int(ipc_n)
# 2nd Filter (confusion) parameter
top_rank_m = 100

# 3rd Filter (assignment) paramenter
top_rank_n = 100
```

In [3]:

```
###   :
import time
start = time.time()
```

In [4]:

```
############################################### step 01. IPC          ##########################################



def yr_patent_ipc(start_yr, end_yr, ipc_n) :
    query1_1 = "SELECT dataset1.new_ipc_code AS IPC, dataset1.app_year AS application_year, COUNT(*) AS applications \
               FROM tm_database.df_us_datamart_assignee_parquet AS dataset1, ( \
               SELECT dataset2.new_ipc_code \
               FROM tm_database.df_us_datamart_assignee_parquet AS dataset2 \
               WHERE dataset2.app_year >= '"+start_yr+"' AND dataset2.app_year <= '"+end_yr+"' \
               GROUP BY dataset2.new_ipc_code  \
               ORDER BY count(*) DESC, dataset2.new_ipc_code DESC LIMIT "+ipc_n+") AS top_ipc \
               WHERE top_ipc.new_ipc_code = dataset1.new_ipc_code AND dataset1.app_year >= '"+start_yr+"' AND dataset1.app_year <= '"+end_yr+
"' \
               GROUP BY dataset1.new_ipc_code, dataset1.app_year \
               ORDER BY COUNT(*) DESC, dataset1.new_ipc_code DESC"

    return cursor.execute(query1_1).as_pandas()
```

```
## pivoting

saved_file_name1="data/" + start_yr + "_to_" + end_yr + "_ipc"+ ipc_n + "_yearly_patents_per_ipc.csv"

if os.path.isfile(saved_file_name1) == False:

    rs = yr_patent_ipc(start_yr, end_yr, ipc_n)
    yearly_patents_per_ipc= rs.pivot(index='IPC', columns='application_year', values='applications')
    yearly_patents_per_ipc = yearly_patents_per_ipc.fillna(0)

    yearly_patents_per_ipc.to_csv(saved_file_name1)

else:

    yearly_patents_per_ipc = pd.read_csv(saved_file_name1,index_col=0)
```

In [5]:

```
yearly_patents_per_ipc.head()
```

Out[5]:

application_year	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010
IPC														
A01H1/00	82	89	74	38	146	417	885	798	740	633	459	431	317	407
A01H1/02	34	112	61	3	67	14	26	47	35	77	149	440	500	716
A01H5/00	300	324	195	118	535	570	426	540	677	890	1265	1305	1277	1589
A01H5/10	124	165	98	25	75	43	146	187	184	217	199	297	379	729
A01K29/00	97	83	49	14	103	95	170	157	166	165	208	194	193	218

In [6]:

```
################################ step 02. IPC

def yr_applicants_ipc(start_yr, end_yr, ipc_n) :
    query2_1 = "SELECT dataset1.new_ipc_code AS IPC, dataset1.app_year AS application_year, dataset1.app_no AS application_num, \
                dataset1.applicant1 AS applicant1, dataset1.applicant2 AS applicant2, dataset1.neo_company_univ1 AS neo_company_univ1, dataset1.neo_company_univ2 AS neo_company_univ2 \
                FROM tm_database.df_us_datamart_assignee_parquet AS dataset1, ( \
                SELECT dataset2.new_ipc_code \
                FROM tm_database.df_us_datamart_assignee_parquet AS dataset2 \
                WHERE dataset2.app_year >= '"+start_yr+"' and dataset2.app_year <= '"+end_yr+"' \
                GROUP BY  dataset2.new_ipc_code \
                ORDER BY count(*) DESC, dataset2.new_ipc_code DESC LIMIT "+ipc_n+") AS top_ipc \
                WHERE top_ipc.new_ipc_code = dataset1.new_ipc_code AND dataset1.app_year >= '"+start_yr+"' AND dataset1.app_year <= '"+end_yr+"' "

    return cursor.execute(query2_1).as_pandas()
```

In [7]:

```
#          ("    "                         )

#                             (yearly applicants per ipc)

modi_ipc_yr_applicant_filename = "data/" + start_yr + "_to_" + end_yr + "_ipc"+ ipc_n + "_modi_ipc_yr_applicant_name.csv"


garbage_company_name=["incorporation", "incorporated", "inc.", "corporation", "corp.", "company", "co.", "limited", "ltd", "ltd.", "plc", "plc.", \
                     "llc", "llc.", "unlimited partnership", "ul", "ul.", "u.l.", "u. l.", "limited partnership", "lp.", "l.p.", "l. p.", "n. v.", \
                      "n.v.", "l.l.c.", "societe", "s.a.", "sa.", "s. a.", "s a.", "responsabilite", "collectif limite", \
                      "commandite", "corporacion", "sociedad", "sdrl.", "s.d.r.l.", "s.l.", "s.l", "s. l.", "sl.", "compania en comandita", \
                      "societa", "s.p.a.", "s. p. a.", "s.r.l.", "dionicko drustvo", "sdn bhd",\
                      "s. r. l.", "accomandita","osakeyhtio", "o.y.", "o. y.", "o y.", "o. y", "aktiebolag", "a.b.", "ab.", \
                      "kommanditbolag", "aktiengesellschaft", "ag.", "a.g.", "gmbh", "egmbh", "partnerschaft", "kommanditgesellschaft", \
                      "k.g.", "kg.", "kabushiki kaisha", "kabushikikaisha", "k.k.", "k. k.", "kk.", "k. k", "kabushiki", "kaisha", "l. p.", "a/s"]

acronym_company_name=["ab", "oy", "o y", "ag", "ohg", "kg", "kk", "snc", "spa", "n v", "sa", "inc", "corp", "lp", "sdrl", "co"]

garbage_univ_institute_name=["university", "college", "school", "foundation", "academy", "academies", "center", "laboratory", "laboratories", "lab." "association",\
                            "universite", "faculte", "institute", "centre", "universidad", "facultad", "instituto", "centro", "stazione", "istituto", "istituti",\
                            "yliopisto", "opisto",  "korkeakoulu", "instituutti", "station", "seminar", "anstalt", 'e.v.', 'e.v']
```

```
if os.path.isfile(modi_ipc_yr_applicant_filename) == False:

    rs2 = yr_applicants_ipc(start_yr, end_yr, ipc_n)

    ipc_yr_applicant = rs2.fillna(-1)


    ind_ipc = list(set(list(ipc_yr_applicant['IPC'])))

    com_univ1 = list(ipc_yr_applicant['neo_company_univ1'])
    com_univ2 = list(ipc_yr_applicant['neo_company_univ2'])

    new_com_univ1_list=[]
    new_com_univ2_list=[]

    new_splt_com1=[]
    new_splt_com2=[]

    com1_type=[]
    com2_type=[]

    i=0

    company_count = 0
    university_count = 0
    individual_count = 0

    while i < len(com_univ1):
        k = 0
        j = 0
        std_com =[]

        if com_univ1[i] != -1:
            new_com1 = com_univ1[i].lower()

            for garbage in garbage_company_name:
                garbage_word = garbage.lower()

                if garbage_word in new_com1:
                    new_com1 = new_com1.replace(garbage_word,'').strip()
                    new_com1 = new_com1.replace(',','').strip()
                    k = k + 1

            for garbage in acronym_company_name:
                garbage_word = " " + garbage.lower()

                if garbage_word in new_com1 and k == 0:
                    new_com1 = new_com1.replace(garbage_word,'').strip()
                    new_com1 = new_com1.replace(',','').strip()
                    k = k + 1

            if k == 0 :
                for garbage in garbage_univ_institute_name:
                    garbage_word = " " + garbage.lower()

                    if garbage_word in new_com1:
                    #  don't delete the words "university", and etc.
                        new_com1 = new_com1.strip()
                        new_com1 = new_com1.replace(',','').strip()
                        j = j + 1

            new_com1 = new_com1.replace('.','').strip()
            new_split_com1 = new_com1.strip().split()

            for split_com in new_split_com1:
                split_com = split_com.strip()
                std_com.append(split_com)

            normalized_company =" ".join(std_com)
            new_com_univ1_list.append(normalized_company.upper().strip())

        else:
            new_com_univ1_list.append("")

        if k > 0:
            com1_type.append('company')
            company_count = company_count + 1
        elif j > 0:
            com1_type.append('university')
            university_count = university_count + 1
        else:
            com1_type.append('individual')
            com1_type.append('individual')

        std_com =[]
        k = 0
        j = 0

        if com_univ2[i] != -1:
            new_com2 = com_univ2[i].lower()

            for garbage in garbage_company_name:
                garbage_word = garbage.lower()

                if garbage_word in new_com2:
                    new_com2 = new_com2.replace(garbage_word,'').strip()
                    new_com2 = new_com2.replace(',','').strip()
                    k = k + 1

            for garbage in acronym_company_name:
                garbage_word = " " + garbage.lower()

                # "co" is included in acronym, but it could be skipped when garbage data first deleted
                if garbage_word in new_com1 and k == 0:
                    new_com2 = new_com2.replace(garbage_word,'').strip()
                    new_com2 = new_com2.replace(',','').strip()
                    k = k + 1

            if k == 0:
                for garbage in garbage_univ_institute_name:
                    garbage_word = " " + garbage.lower()

                    if garbage_word in new_com2:
                    #  don't delete the words "university", and etc.
                        new_com2 = new_com2.strip()
                        new_com2 = new_com2.replace(',','').strip()
                        j = j + 1

            new_com2 = new_com2.replace('.','').strip()
            new_split_com2 = new_com2.strip().split()

            for split_com in new_split_com2:
                split_com = split_com.strip()
                std_com.append(split_com)

            normalized_company =" ".join(std_com)
            new_com_univ2_list.append(normalized_company.upper().strip())

        else:
            new_com_univ2_list.append("")

        if k > 0:
            com2_type.append('company')
            company_count = company_count + 1
        elif j > 0:
            com2_type.append('university')
            university_count = university_count + 1
        else:
            com2_type.append('individual')

        i = i + 1

    ipc_yr_applicant['new_company_univ1'] = new_com_univ1_list
    ipc_yr_applicant['type_of_new_company_univ1'] = com1_type
    ipc_yr_applicant['new_company_univ2'] = new_com_univ2_list
    ipc_yr_applicant['type_of_new_company_univ2'] = com2_type

    # calculating yearly applicants per ipc

    modi_ipc_yr_applicant = ipc_yr_applicant


    IPC = list(set(list(modi_ipc_yr_applicant['IPC'])))
    year = list(range(int(start_yr), int(end_yr)+1))

    pat_data = modi_ipc_yr_applicant.fillna(-1)

    pat_data.to_csv(modi_ipc_yr_applicant_filename)

else:

    pat_data = pd.read_csv(modi_ipc_yr_applicant_filename)


IPC = list(set(list(pat_data['IPC'])))
```

```
saved_file_name2="data/" + start_yr + "_to_" + end_yr + "_ipc"+ ipc_n + "_yearly_applicants_per_ipc.csv"

if os.path.isfile(saved_file_name2) == False:

    yearly_applicants_per_ipc=pd.DataFrame([],index=IPC)

    temp_data = pd.DataFrame([])
    target_data = pd.DataFrame([])
    pat_data_temp= pd.DataFrame([])

    pat_data_temp = pat_data_temp.append(pat_data.loc[pat_data['type_of_new_company_univ1'] == 'company']\
                                  .rename( columns={"new_company_univ1":"new_company_univ"}))
    pat_data_temp = pat_data_temp.append(pat_data.loc[pat_data['type_of_new_company_univ1'] == 'university']\
                                  .rename( columns={"new_company_univ1":"new_company_univ"}))

    pat_data_temp = pat_data_temp.append(pat_data.loc[pat_data['type_of_new_company_univ2'] == 'company']\
                                  .rename( columns={"new_company_univ2":"new_company_univ"}))

    pat_data_temp = pat_data_temp.append(pat_data.loc[pat_data['type_of_new_company_univ2'] == 'university']\
                                  .rename( columns={"new_company_univ2":"new_company_univ"}))

    temp_data = pat_data_temp[['IPC','application_year', 'new_company_univ']].reset_index(drop=True)
    temp_data = temp_data.drop_duplicates()

    yearly_applicants_list = temp_data.groupby(['IPC', 'application_year'])['new_company_univ'].apply(list) \
                        .reset_index(name='company_univ_list')

    for idx, ipca in yearly_applicants_list.iterrows():
        yearly_applicants_per_ipc.loc[ipca[0],ipca[1]]= len(ipca[2])


    yearly_applicants_per_ipc.to_csv(saved_file_name2)

else:

    yearly_applicants_per_ipc=pd.read_csv(saved_file_name2, index_col=0)
```

In [8]:

```
####
yearly_patents_per_ipc = yearly_patents_per_ipc.fillna(0)
yearly_applicants_per_ipc = yearly_applicants_per_ipc.fillna(0)

yearly_patents = yearly_patents_per_ipc
yearly_applicants = yearly_applicants_per_ipc

sorted_yearly_patents = yearly_patents.sum(axis=1).sort_values(ascending=False)
sorted_ipc = list (sorted_yearly_patents.index)[0:ranking]

year = list(yearly_patents.columns)
```

In [9]:

```
# yearly_applicants  yearly_patents  ipc          (Query   order by                              IPC          )
set(yearly_applicants.index).difference(set(yearly_patents.index))
```

Out[9]:

```
set()
```

In [10]:

```
# yearly_applicants  yearly_patents  ipc
set(yearly_patents.index).difference(set(yearly_applicants.index))
```

Out[10]:

```
set()
```

In [11]:

```
#### Calculating accumulated Pantets and Applicants for Growth index

period = int(end_yr) - int(start_yr) +1
averaged_patents = sorted_yearly_patents.iloc[0:ranking].sum() / len(sorted_ipc) / period
displayed_applicants=pd.DataFrame([], index=sorted_ipc, columns=yearly_applicants.columns)
displayed_patents=pd.DataFrame([], index=sorted_ipc, columns=yearly_patents.columns)
```

```
for ipc in sorted_ipc:
    displayed_applicants.loc[ipc,:]=yearly_applicants.loc[ipc]
    displayed_patents.loc[ipc,:]=yearly_patents.loc[ipc]

ipc = sorted_ipc


ipc_statistics = ['accumulated patents', 'accumulated applicants', 'averaged patents', \
                  'averaged applicants', 'averaged weighted CAGR of patents', \
                  'averaged weighted CAGR of applicants']

ipc_analysis_data=pd.DataFrame([],index=sorted_ipc, columns=ipc_statistics)

ipc_analysis_data['accumulated patents']=list(displayed_patents.sum(axis=1))
ipc_analysis_data['accumulated applicants']=list(displayed_applicants.sum(axis=1))
ipc_analysis_data['averaged patents']=list(displayed_patents.mean(axis=1))
ipc_analysis_data['averaged applicants']=list(displayed_applicants.mean(axis=1))

weight = list(range(1, len(year)))
```

In [12]:

```
ipc_analysis_data
```

Out[12]:

	accumulated patents	accumulated applicants	averaged patents	averaged applicants	averaged weighted CAGR of patents	averaged weighted CAGR of applicants
G06F15/16	49970.0	14153.0	3569.285714	1010.928571	NaN	NaN
G06F17/30	48526.0	13127.0	3466.142857	937.642857	NaN	NaN
C12Q1/68	37870.0	12566.0	2705.000000	897.571429	NaN	NaN
C07H21/04	32585.0	9736.0	2327.500000	695.428571	NaN	NaN
G06F17/00	29922.0	10801.0	2137.285714	771.500000	NaN	NaN
...	...	...	...	...	...	...
B32B33/00	1850.0	1075.0	132.142857	76.785714	NaN	NaN
A61K39/02	1848.0	953.0	132.000000	68.071429	NaN	NaN
A61B8/14	1848.0	722.0	132.000000	51.571429	NaN	NaN
B60R21/16	1842.0	689.0	131.571429	49.214286	NaN	NaN
H04N5/222	1840.0	677.0	131.428571	48.357143	NaN	NaN

500 rows × 6 columns

In [13]:

```
displayed_applicants
```

Out[13]:

	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010
G06F15/16	159	191	205	650	1388	1119	1115	1008	972	1282	1320	1474	1636	1634
G06F17/30	329	209	150	299	426	364	488	746	1203	1362	1842	2018	1911	1780
C12Q1/68	366	317	258	275	864	1248	1495	1494	1306	1164	1004	971	926	878
C07H21/04	384	352	310	360	773	1039	1140	1086	857	856	708	625	626	620
G06F17/00	224	186	159	332	601	650	923	1107	1112	1204	1188	1140	988	987
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
B32B33/00	17	26	6	19	34	54	77	92	141	137	135	126	108	103
A61K39/02	41	42	38	27	43	79	115	114	89	86	85	62	69	63
A61B8/14	7	18	9	7	45	44	57	65	79	80	61	66	101	83
B60R21/16	49	41	19	19	55	41	54	68	88	50	58	52	49	46
H04N5/222	17	12	14	26	36	44	62	76	60	61	60	65	80	64

500 rows × 14 columns

In [14]:

```
#### CAGR function

def CAGR (initial, final, period):
    cagr = pow((final/initial),(1/period)) - 1
    return round(cagr,3)
```

In [15]:

```
#### Calculating weighted CAGR of pantets and applicants for Growth index

i = 0
while i < len(ipc):
    ipc_patents = list(yearly_patents.loc[ipc[i]])
    ipc_applicants = list(yearly_applicants.loc[ipc[i]])

    j = 1

    weight_accum_cagr1 = 0
    weight_accum_cagr2 = 0

    while j < len (ipc_patents):
        if ipc_patents[0]==0: ipc_patents[0]=1        ###         0   ,
        if ipc_applicants[0]==0: ipc_applicants[0]=1  ###         0   ,

        cagr1 = CAGR(ipc_patents[0], ipc_patents[j], j)
        cagr2 = CAGR(ipc_applicants[0], ipc_applicants[j], j)

        weight_accum_cagr1 = weight[j-1]*cagr1 + weight_accum_cagr1
        weight_accum_cagr2 = weight[j-1]*cagr2 + weight_accum_cagr2

        j = j + 1

    weight_accum_cagr1 = weight_accum_cagr1/sum(weight)
    weight_accum_cagr2 = weight_accum_cagr2/sum(weight)
    ipc_analysis_data.loc[ipc[i],'averaged weighted CAGR of patents'] = round(weight_accum_cagr1,3)
    ipc_analysis_data.loc[ipc[i],'averaged weighted CAGR of applicants'] = round(weight_accum_cagr2,3)

    i = i + 1
```

In [16]:

```
saved_file_name_3 ="data/" + start_yr + "_to_" + end_yr + "_ipc"+ ipc_n + "_ranking"+ str(ranking) + "_ipc_analysis.csv"
ipc_analysis_data.to_csv(saved_file_name_3)
```

In [17]:

```
######################## 1st filtering :

sorted_yearly_patents = yearly_patents.sum(axis=1).sort_values(ascending=False)
sorted_ipc = list (sorted_yearly_patents.index)[0:ranking]
```

In [18]:

```
################################## Parameter adjustment #################################
################################ 1st output parameter ###################################
a = [10, 30, 50, 100, 150]
#b = [200, 300, 400, 500, 600]
c = [100, 200, 300, 400, 500, 600, 700]
d = [1500, 2000, 2500, 3000, 3500, 4000]
alpha = [0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.1]
beta = [0.02, 0.03,  0.04, 0.05, 0.06, 0.07, 0.08, 0.1]


#################################### 2nd output parameter: b (maximum applicants) ###################################

# b = [700, 800, 900]

################################## 3nd output parameter #######################################
b = [200, 300, 400, 500, 600, 700, 800, 900]

Picked_ipc_set_per_parameters = pd.DataFrame([])

parameter_list = []

Fst_picked_ipc_list = []

Fst_picked_ipc_no_list  = []
```

```
for aa in a:

    for bb in b:

        for cc in c:

            for dd in d:

                for alphi in alpha:

                    for beti in beta:

                        x_plus_range = bb
                        x_minus_range = aa
                        y_plus_range = dd
                        y_minus_range = cc

                        patent_cagr = alphi
                        applicant_cagr = beti


                        mask1 = (ipc_analysis_data['averaged applicants'] >= x_minus_range) & (ipc_analysis_data['averaged applicants'] <= x_plus_range)

                        mask2 = (ipc_analysis_data['averaged patents'] >= y_minus_range) & (ipc_analysis_data['averaged patents'] <= y_plus_range)

                        mask3 = ipc_analysis_data['averaged weighted CAGR of patents'] > patent_cagr

                        mask4 = ipc_analysis_data['averaged weighted CAGR of applicants'] > applicant_cagr

                        ipc_set1 = set(list(ipc_analysis_data.loc[mask1].index))
                        ipc_set2 = set(list(ipc_analysis_data.loc[mask2].index))
                        ipc_set3 = set(list(ipc_analysis_data.loc[mask3].index))
                        ipc_set4 = set(list(ipc_analysis_data.loc[mask4].index))

                        temp_picked_ipc1 = ipc_set1.intersection(ipc_set2)
                        temp_picked_ipc2 = ipc_set3.intersection(ipc_set4)

                        Fst_picked_ipc = temp_picked_ipc1.intersection(temp_picked_ipc2)

                        parameter_list.append([aa, bb, cc, dd, alphi, beti])
                        Fst_picked_ipc_list.append(Fst_picked_ipc)
                        Fst_picked_ipc_no_list.append(len(Fst_picked_ipc))


Picked_ipc_set_per_parameters ['parameters'] = parameter_list

Picked_ipc_set_per_parameters ['Fst_picked_ipc'] = Fst_picked_ipc_list

Picked_ipc_set_per_parameters ['Fst_picked_ipc no.'] = Fst_picked_ipc_no_list
```

In [19]:

```
######################################################### 2st filtering :      ##############################################

IPC = sorted_ipc

ipc_applicants={}

confusion_matrix_filename = "data/" + start_yr + "_to_" + end_yr + "_ipc_" +  ipc_n + "_confusion_matrix.csv"

if os.path.isfile(confusion_matrix_filename) == False:

    ipc_confusion_matrix = pd.DataFrame([],index=IPC, columns=IPC)

    temp_data1 = pd.DataFrame([])
    temp_data2 = pd.DataFrame([])

    ipca_application_num_list = []
    ipcb_application_num_list = []
    common_application_list = []

    pat_data2 = pat_data.groupby('IPC')['application_num'].apply(list).reset_index(name='application_num_list')

    i = 0

    for idx, ipca in pat_data2.iterrows():

        ipca_application_num_list = set(ipca[1])

        for idx, ipcb in pat_data2.iterrows():
            if ipca[0] != ipcb[0]:
                ipcb_application_num_list = set(ipcb[1])
                ipcb_application_num_list = set(ipcb[1])
                common_application_list = list(ipca_application_num_list.intersection(ipcb_application_num_list))

                ipc_confusion_matrix.loc[ipca[0],ipcb[0]]= len(common_application_list)

            else:
                ipc_confusion_matrix.loc[ipca[0],ipcb[0]]= 0

        i = i + 1

    ipc_confusion_matrix.to_csv(confusion_matrix_filename)

else:

    ipc_confusion_matrix = pd.read_csv(confusion_matrix_filename, index_col=0)
```

In [20]:

```
ipc_confusion_matrix.index[0:5]
```

Out[20]:

```
Index(['G06F15/16', 'G06F17/30', 'C12Q1/68', 'C07H21/04', 'G06F17/00'], dtype='object')
```

In [21]:

```
##################################### 2nd Filtering        top_rank_m     #####################################

ipc_confusion_data = ipc_confusion_matrix

IPC_x = list(ipc_confusion_data.columns)
IPC_y = list(ipc_confusion_data.index)
ipc_confusion_list = []

i = 0
for ipcy in IPC_y :
    j = 0
    ipc_confusion_index = 0

    for ipcx in IPC_x :
        if ipcx[0:4] != ipcy[0:4]:
            ipc_confusion_index = ipc_confusion_index + ipc_confusion_data.loc[ipcy,ipcx]

    ipc_confusion_list.append(ipc_confusion_index)


ipc_confusion_data['IPC confusion'] = ipc_confusion_list

ipc_confusion_data = ipc_confusion_data.sort_values('IPC confusion', ascending=False)

candidateIPC = set(list(ipc_confusion_data.index)[0:top_rank_m])
```

In [22]:

```
########################       2       IPC          "Picked_ipc_set_per_parametres" DataFrame     ################333

i =  0

Snd_picked_ipc_list = []

Snd_picked_ipc_no_list  = []


while i < len(Picked_ipc_set_per_parameters.index):

    Fst_picked_ipc = Picked_ipc_set_per_parameters.loc[i,'Fst_picked_ipc']

    Snd_picked_ipc = Fst_picked_ipc.intersection(candidateIPC)

    Snd_picked_ipc_list.append(Snd_picked_ipc)
    Snd_picked_ipc_no_list.append(len(Snd_picked_ipc))

    i = i + 1

#    print("The 2nd Picked IPC no. is %d" % len(Snd_picked_ipc))

Picked_ipc_set_per_parameters ['Snd_picked_ipc'] = Snd_picked_ipc_list

Picked_ipc_set_per_parameters ['Snd_picked_ipc no.'] = Snd_picked_ipc_no_list
```

In [23]:

```
####################################################      ###########################################

saved_file_name_4 ="data/" + start_yr + "_to_" + end_yr + "_us_assignment_data.csv"

def us_assignment_ipc(start_date, end_date) :
    query6_2 = "SELECT dataset1.app_no as application_no \
            ,dataset1.app_year as application_year \
            ,dataset1.name_of_invention as name_of_invention \
            ,dataset1.new_ipc_code as IPC \
            ,dataset1.neo_company_univ1 as applicant1 \
            ,dataset1.neo_company_univ2 as applicant2 \
            FROM tm_database.df_us_datamart_assignee_parquet AS dataset1, (\
            SELECT dataset2.appno_doc_num as app_no \
            FROM tm_database.df_assignment_mart_parquet AS dataset2 \
            WHERE dataset2.employer_assign = '0' \
            AND (dataset2.convey_ty ='assignment' OR dataset2.convey_ty ='govern') \
            AND dataset2.record_dt >= '"+start_date+"' \
            AND dataset2.record_dt <= '"+end_date+"' \
            AND dataset2.appno_date >= '"+start_date+"' \
            AND dataset2.appno_date <= '"+end_date+"' \
            GROUP BY dataset2.appno_doc_num ) as Assign_pat \
            WHERE dataset1.app_no = Assign_pat.app_no "

    return cursor.execute(query6_2).as_pandas()

#
if os.path.isfile(saved_file_name_4) == False:
    us_assignment_ipc_data = us_assignment_ipc(start_date, end_date)
    us_assignment_ipc_data.to_csv(saved_file_name_4, index=False)

else:
    us_assignment_ipc_data = pd.read_csv(saved_file_name_4)
```

In [24]:

```
################################################### 3rd filter #############################################

sorted_us_assignment_ipc = us_assignment_ipc_data['IPC'].value_counts()
top_rank = top_rank_n
us_assignment_ipc = list(sorted_us_assignment_ipc.index)[0:top_rank]
candidateIPC1 = set(us_assignment_ipc)

######################## Finally (3rd) Picked IPC : Trd_picked_ipc

i =  0

Trd_picked_ipc_list = []

Trd_picked_ipc_no_list  = []


while i < len(Picked_ipc_set_per_parameters.index):

    Snd_picked_ipc = Picked_ipc_set_per_parameters.loc[i,'Snd_picked_ipc']

    Trd_picked_ipc = Snd_picked_ipc.intersection(candidateIPC1)

    Trd_picked_ipc_list.append(Trd_picked_ipc)
    Trd_picked_ipc_no_list.append(len(Trd_picked_ipc))

    i = i + 1


Picked_ipc_set_per_parameters ['Trd_picked_ipc'] = Trd_picked_ipc_list

Picked_ipc_set_per_parameters ['Trd_picked_ipc no.'] = Trd_picked_ipc_no_list
```

In [25]:

```
########################################### IPC & Technology matching ############################################

ipc_tech_filename = 'data/IPC_Tech_matching4.csv'
IPC_Tech_correlation_data = pd.read_csv(ipc_tech_filename, encoding='UTF-8')

i = 0
tech_ipc_list = []
```

```
while i < 10:
    ipc = 'IPC' + str(i+1)
    ipc_list = list(IPC_Tech_correlation_data[ipc])

    for ipca in ipc_list:
        if ipca != '-1':
            tech_ipc_list.append(ipca)

    i = i + 1

# IPC_Tech_correlation_data1 = pd.DataFrame([])
# IPC_Tech_correlation_data1["IPC"]=tech_ipc_list
# IPC_Tech_correlation_data1 = IPC_Tech_correlation_data1["IPC"].value_counts()
IPC_Tech_correlation_set = set(tech_ipc_list)
```

In [26]:

```
IPC_Tech_correlation_data.head()
```

Out[26]:

	No	Tech_Product	Searched US Patents No.	IPC1	IPC2	IPC3	IPC4	IPC5	IPC6	IPC7	IPC8	IPC9	IPC10
0	1	3D Printing	745	B29C67/00	B41J2/01	B29C70/68	B41J3/407	B41J3/54	B41J11/00	G06F19/00	D04H1/16	B23K26/34	B22F3/00
1	2	Big Data Analysis	14,399	G06F17/30	G06F7/00	G06F17/00	G06F15/16	H04L29/06	G06F12/00	G06F19/00	G06F13/00	G06Q10/00	H04L29/08
2	3	Intelligent Cyber Security	1,924	H04L29/06	G06F21/00	G06F15/16	H04L9/00	G06F11/00	H04L9/32	G06F12/14	G06F15/173	G06F11/30	G06F17/00
3	4	Autonomous Things	1,643	G06F19/00	G05D1/02	G05D1/00	G06F17/00	G06K9/00	B25J5/00	G01C21/00	A47L9/00	G05B19/04	A47L5/30
4	5	AR/VR	6,068	G09G5/00	G06F3/00	G06F3/01	G06T15/00	G06F3/033	G06K9/00	G06F17/00	G06F3/048	G06F17/30	G06F19/00

In [27]:

```
########## Matched ones out of Finally(3rd) picked IPC(s) : Tech_matched_ipc ###########################

i = 0


matching_ipc_no_list = []

matching_ipc_list = []

ipc_matching_probability_list = []

matching_tech_name_list = []

matching_tech_no_list = []

tech_matching_probability_list = []


while i < len(Picked_ipc_set_per_parameters.index):

    Trd_picked_ipc = Picked_ipc_set_per_parameters.loc[i, 'Trd_picked_ipc']

    Tech_matched_ipc = Trd_picked_ipc.intersection(IPC_Tech_correlation_set)


    j = 0


    matching_ipc_no = []

    matching_ipc = []

    matching_tech_name = []

    matching_tech_no = []


    while j < len(IPC_Tech_correlation_data.index):

        Tech_name = IPC_Tech_correlation_data.loc[j,'Tech_Product']

        IPC_tech_set=list(IPC_Tech_correlation_data.iloc[j,3:13])
        matching_ipc_per_tech = Tech_matched_ipc.intersection(IPC_tech_set)

        if len(matching_ipc_per_tech) > 0:


            matching_tech_name.append(Tech_name)

        j = j + 1


    if len(Trd_picked_ipc) !=0:
        ipc_matching_probability = (len(Tech_matched_ipc) / len(Trd_picked_ipc)) * 100
    else:
        ipc_matching_probability = 0

    tech_matching_probability = (len (matching_tech_name) / len(IPC_Tech_correlation_data.index))*100

    matching_ipc_list.append(Tech_matched_ipc)
    matching_ipc_no_list.append(len(Tech_matched_ipc))
    ipc_matching_probability_list.append(round(ipc_matching_probability,1))
    matching_tech_name_list.append(matching_tech_name)
    matching_tech_no_list.append(len(matching_tech_name))
    tech_matching_probability_list.append(round(tech_matching_probability,1))


    i = i + 1


Picked_ipc_set_per_parameters ['Matching ipc'] = matching_ipc_list
Picked_ipc_set_per_parameters ['Matching ipc no.'] = matching_ipc_no_list
Picked_ipc_set_per_parameters ['IPC matching probabilitiy'] = ipc_matching_probability_list

Picked_ipc_set_per_parameters ['Matching Tech name'] = matching_tech_name_list

Picked_ipc_set_per_parameters ['Matching Tech no.'] = matching_tech_no_list
Picked_ipc_set_per_parameters ['Tech matching probabilitiy'] = tech_matching_probability_list

output_filename = "output/" + "High_potential_" + start_yr + "_to_" + end_yr + "_ipc"+ ipc_n + 'm' + str(top_rank_m) + "_n" + str(top_rank_n)+  "_outputs.csv"


#######        ipc (Trd_picked ipc)    10    15    , IPC matching     60%   , Tech Matching     50%       High Potential        ##########
target_IPC_matching_probability= ( Picked_ipc_set_per_parameters ['IPC matching probabilitiy'] >= 60 )
target_Tech_matching_probability = ( Picked_ipc_set_per_parameters ['Tech matching probabilitiy'] >= 50 )
Trd_picked_ipc_num = (Picked_ipc_set_per_parameters ['Trd_picked_ipc no.'] >= 10) & (Picked_ipc_set_per_parameters ['Trd_picked_ipc no.'] <= 20)

High_potential = Picked_ipc_set_per_parameters.loc[target_IPC_matching_probability & Trd_picked_ipc_num ,  \
                                                  ['parameters', 'Fst_picked_ipc', 'Fst_picked_ipc no.', 'Snd_picked_ipc',  'Snd_picked_ipc no.',  'Trd_picked_ipc' , \
                                                   'Trd_picked_ipc no.', 'Matching ipc' , 'Matching ipc no.', 'IPC matching probabilitiy','Matching Tech name', 'Matching Tech no.', 'Tech matching probabilitiy']]

High_potential = High_potential.sort_values(by='IPC matching probabilitiy', ascending = False)

High_potential.to_csv(output_filename, index=False)
```

In [28]:

```
######    :
r_time = round((time.time()-start)/60, 2)
print('Running time :', r_time, 'minutes')
```

```
Running time : 13.19 minutes
```

In [29]:

```
High_potential.head()
```

Out[29]:

	parameters	Fst_picked_ipc	Fst_picked_ipc no.	Snd_picked_ipc	Snd_picked_ipc no.	Trd_picked_ipc	Trd_picked_ipc no.	Matching ipc	Matching ipc no.	IP matchin probabiliti
85580	[100, 900, 600, 4000, 0.03, 0.06]	{C12Q1/68, C12N5/06, A63F9/24, H04L9/32,	34	{C12Q1/68, G06F17/00, C12N5/06, A61K48/00,	18	{C12Q1/68, G06F17/00, C12P21/02, A61K48/00,	18	{C12Q1/68, G06F17/00, A61K48/00, H04L9/00,	15	83

	0.03, 0.06]	H04L9/32, H04L1...		A61K48/00, H04...		A61K48/00, C1...		H04L9/00, H04...		
	parameters	**Fst_picked_ipc**	**Fst_picked_ipc no.**	**Snd_picked_ipc**	**Snd_picked_ipc no.**	**Trd_picked_ipc**	**Trd_picked_ipc no.**	**Matching ipc**	**Matching ipc no.**	**IP matchin probabiliti**
64454	[50, 900, 700, 4000, 0.02, 0.08]	{C12Q1/68, C12N5/06, H04L9/32, H04L12/56, G06F...	28	{C12Q1/68, G06F17/00, C12N5/06, A61K48/00, H04...	18	{C12Q1/68, G06F17/00, C12P21/02, A61K48/00, C1...	18	{C12Q1/68, G06F17/00, A61K48/00, H04L9/00, H04...	15	83
64422	[50, 900, 700, 3500, 0.06, 0.08]	{C12Q1/68, C12N5/06, H04L9/32, H04L12/56, G06F...	28	{C12Q1/68, G06F17/00, C12N5/06, A61K48/00, H04...	18	{C12Q1/68, G06F17/00, C12P21/02, A61K48/00, C1...	18	{C12Q1/68, G06F17/00, A61K48/00, H04L9/00, H04...	15	83
64428	[50, 900, 700, 3500, 0.07, 0.06]	{C12Q1/68, C12N5/06, H04L9/32, H04L12/56, G06F...	30	{C12Q1/68, G06F17/00, C12N5/06, A61K48/00, H04...	18	{C12Q1/68, G06F17/00, C12P21/02, A61K48/00, C1...	18	{C12Q1/68, G06F17/00, A61K48/00, H04L9/00, H04...	15	83
64429	[50, 900, 700, 3500, 0.07, 0.07]	{C12Q1/68, C12N5/06, H04L9/32, H04L12/56, G06F...	29	{C12Q1/68, G06F17/00, C12N5/06, A61K48/00, H04...	18	{C12Q1/68, G06F17/00, C12P21/02, A61K48/00, C1...	18	{C12Q1/68, G06F17/00, A61K48/00, H04L9/00, H04...	15	83

In [31]:

```
High_potential.to_csv(output_filename)
```

In [34]:

```
High_potential.loc[High_potential['IPC matching probabilitiy']==83.3]
```

Out[34]:

	parameters	Fst_picked_ipc	Fst_picked_ipc no.	Snd_picked_ipc	Snd_picked_ipc no.	Trd_picked_ipc	Trd_picked_ipc no.	Matching ipc	Matching ipc no.	IP matchin probabiliti
85580	[100, 900, 600, 4000, 0.03, 0.06]	{C12Q1/68, C12N5/06, A63F9/24, H04L9/32, H04L1...	34	{C12Q1/68, G06F17/00, C12N5/06, A61K48/00, H04...	18	{C12Q1/68, G06F17/00, C12P21/02, A61K48/00, C1...	18	{C12Q1/68, G06F17/00, A61K48/00, H04L9/00, H04...	15	83
64454	[50, 900, 700, 4000, 0.02, 0.08]	{C12Q1/68, C12N5/06, H04L9/32, H04L12/56, G06F...	28	{C12Q1/68, G06F17/00, C12N5/06, A61K48/00, H04...	18	{C12Q1/68, G06F17/00, C12P21/02, A61K48/00, C1...	18	{C12Q1/68, G06F17/00, A61K48/00, H04L9/00, H04...	15	83
64422	[50, 900, 700, 3500, 0.06, 0.08]	{C12Q1/68, C12N5/06, H04L9/32, H04L12/56, G06F...	28	{C12Q1/68, G06F17/00, C12N5/06, A61K48/00, H04...	18	{C12Q1/68, G06F17/00, C12P21/02, A61K48/00, C1...	18	{C12Q1/68, G06F17/00, A61K48/00, H04L9/00, H04...	15	83
64428	[50, 900, 700, 3500, 0.07, 0.06]	{C12Q1/68, C12N5/06, H04L9/32, H04L12/56, G06F...	30	{C12Q1/68, G06F17/00, C12N5/06, A61K48/00, H04...	18	{C12Q1/68, G06F17/00, C12P21/02, A61K48/00, C1...	18	{C12Q1/68, G06F17/00, A61K48/00, H04L9/00, H04...	15	83
64429	[50, 900, 700, 3500, 0.07, 0.07]	{C12Q1/68, C12N5/06, H04L9/32, H04L12/56, G06F...	29	{C12Q1/68, G06F17/00, C12N5/06, A61K48/00, H04...	18	{C12Q1/68, G06F17/00, C12P21/02, A61K48/00, C1...	18	{C12Q1/68, G06F17/00, A61K48/00, H04L9/00, H04...	15	83
...	...	...	...	...	...	...	...	...	...	
20972	[10, 900, 600, 3000, 0.07, 0.06]	{C12Q1/68, C12N5/06, A63F9/24, H04L9/32, H04L1...	34	{C12Q1/68, G06F17/00, C12N5/06, A61K48/00, H04...	18	{C12Q1/68, G06F17/00, C12P21/02, A61K48/00, C1...	18	{C12Q1/68, G06F17/00, A61K48/00, H04L9/00, H04...	15	83
64061	[50, 900, 600, 3500, 0.1, 0.07]	{C12Q1/68, C12N5/06, A63F9/24, H04L9/32, H04L1...	31	{C12Q1/68, G06F17/00, C12N5/06, A61K48/00, H04...	18	{C12Q1/68, G06F17/00, C12P21/02, A61K48/00, C1...	18	{C12Q1/68, G06F17/00, A61K48/00, H04L9/00, H04...	15	83

	parameters	Fst_picked_ipc	Fst_picked_ipc no.	Snd_picked_ipc	Snd_picked_ipc no.	Trd_picked_ipc	Trd_picked_ipc no.	Matching ipc	Matching ipc no.	IP matchin probabiliti
64045	[50, 900, 600, 3500, 0.07, 0.07]	{C12Q1/68, C12N5/06, A63F9/24, H04L9/32, H04L1...	33	{C12Q1/68, G06F17/00, C12N5/06, A61K48/00, H04...	18	{C12Q1/68, G06F17/00, C12P21/02, A61K48/00, C1...	18	{C12Q1/68, G06F17/00, A61K48/00, H04L9/00, H04...	15	83
20957	[10, 900, 600, 3000, 0.05, 0.07]	{C12Q1/68, C12N5/06, A63F9/24, H04L9/32, H04L1...	33	{C12Q1/68, G06F17/00, C12N5/06, A61K48/00, H04...	18	{C12Q1/68, G06F17/00, C12P21/02, A61K48/00, C1...	18	{C12Q1/68, G06F17/00, A61K48/00, H04L9/00, H04...	15	83
20964	[10, 900, 600, 3000, 0.06, 0.06]	{C12Q1/68, C12N5/06, A63F9/24, H04L9/32, H04L1...	34	{C12Q1/68, G06F17/00, C12N5/06, A61K48/00, H04...	18	{C12Q1/68, G06F17/00, C12P21/02, A61K48/00, C1...	18	{C12Q1/68, G06F17/00, A61K48/00, H04L9/00, H04...	15	83

720 rows × 13 columns

In [35]:

```
len(High_potential.loc[High_potential['IPC matching probabilitiy']==83.3])
```

Out[35]:

```
720
```

In [ ]:

```

```