2021-Individual-Project-PG v1.1

December 10, 2021

DATE: 2021-12

COURSE CODE AND TITLE: CMSE11433 Python Programming

ASSIGNMENT NAME: Individual Assignment

EXAM/B NUMBER: B199263

[1]:
```
pip install -U kaleido
```

```
Requirement already satisfied: kaleido in /opt/conda/lib/python3.9/site-packages
(0.2.1)
Note: you may need to restart the kernel to use updated packages.
```

[28]:
```
# list of library for this project
import requests
import pprint as pp
import datetime
from datetime import date
from datetime import datetime
import math
import numpy as np
import pandas as pd
import urllib
import json
import os
import time
import plotly.graph_objects as go
from functools import reduce
import plotly.express as px
from plotly.subplots import make_subplots
import plotly.io as pio
pio.renderers.default = 'notebook+pdf'
```

[3]:
```
class DataLoader:
    """This class is defined to load data from the company housing API
       and all the data would be saved with the assigned name.
    """

    def __init__(self, query_words, basic_info_file_name,
                  details_info_file_name, persons_info_file_name):
        """This is a initial function.

        Args:
            query_words:
                A list contains all the words to be queried.
            basic_info_file_name:
                The name of the basic information file to be saved.
            details_info_file_name:
                The name of the detail information file to be saved.
            persons_info_file_name:
                The name of the persons information file to be saved.
        """
        for word in query_words:

            # get the basic information of queried companies
            companies_basics = search_for_companies_with_query(word)
            add_data_to_file(companies_basics, basic_info_file_name)

            # get the detial information of queried companies
            companies_details = detailed_info_about_companies_with_name(word)
            add_data_to_file(companies_details, details_info_file_name)

            # get the significant persons of these companies
            companies_ids = [company['company_number']
                             for company in companies_basics]
            persons_info = [all_persons_in_company(company_id)
                            for company_id in companies_ids]
            add_data_to_file(persons_info, persons_info_file_name)
```

[4]:
```
class Company:
    """This class is defined to contain all the relevant atrributes
       of a company.

    Attributes:
        significant_persons:
            A list contains all the significant persons.
        name:
            The name of this company.
        number:
            The company number of this company.
        status:
            The status of this company, active or dissolved.
        date_of_creation:
            A string records when it is created.
        date_of_cessation:
            A string records when it is dissolved,
            if it is dissolved.
        age:
            The age of this company.
        sic_codes:
            The sic codes of this company.
    """

    significant_persons = []
    name = ""
    number = ""
    status = ""
    date_of_creation = ""
    date_of_cessation = ""
    age = 0
    date_false_occurred = False
    sic_codes = []

    def __init__(self, company_info, person_info):
        """This is a initial function.

        Args:
            company_info: The details dict of this company.
            person_info: The signicant persons dict.
        """
        self.name = company_info['company_name']
        self.number = company_info['company_number']
        if 'company_status' in company_info:
            self.status = company_info['company_status']
        if 'sic_codes' in company_info:
            self.sic_codes = company_info['sic_codes']
        if 'date_of_creation' in company_info:
            self.date_of_creation = company_info['date_of_creation']
        else:
            date_false_occurred = True
        if self.status == 'dissolved':
            self.date_of_cessation = company_info['date_of_cessation']
        # culculate the age of the company
        self.calculate_age_of_company()
        self.significant_persons = person_info


    def calculate_age_of_company(self):
        """This function is used to culculate the age of this company.
        """
        if not self.date_false_occurred:
            if self.status == "active":
                if self.date_of_creation != "":
                    self.age = 2021 - int(self.date_of_creation[0:4])
            elif self.status == "dissolved":
                    self.age = int(self.date_of_cessation[0:4]) - int(self.
↪date_of_creation[0:4])
        else:
            self.age = 0

    def get_number_of_doctoral_owner(self):
        """This function is used to culculate the number of doctor
           owners of this company.
        """
        count = 0
        if self.significant_persons is not None:
            for person in self.significant_persons:
                if 'name_elements' in person:
                    if 'title' in person['name_elements']:
                        if person['name_elements']['title'] == 'Dr':
                            count = count + 1
        return count

    def recognize_nationalities(self):
        """This function is used to culculate the number of
           nationalities with the owners of this company.

        Returns:
            the number of nationalities with the owners in this company.
        """
        if self.significant_persons  is not None :
            nationalities = []
            for person in self.significant_persons:
                if 'nationality' in person:
                    nationalities.append(person['nationality'])
            # Replace all these nationality with British
            repetition_list = ['United Kingdom', 'British',
                               'English', 'Welsh', 'Scottish','Irish']
            clean_nationalities = ['British' if nationality in repetition_list
                                   else nationality
                                   for nationality in nationalities]
            return len(clean_nationalities)
        else:
            return np.nan
```

[5]:
```
# file operating functions

def save_data_as_file(data, file_name):
    """This function is used to save the data as a file with this filename.
    """
    with open(file_name, 'w') as outfile:  # 'w' means open for writing.
        json.dump(data, outfile)


def load_file_named(file_name):
    """This function is used to load the file with this filename.
    """
    loaded_data = []  # or however else you want to represent empty file
    if os.path.isfile(file_name):
        with open(file_name) as infile:
            loaded_data = json.load(infile)
    return loaded_data


def add_data_to_file(data_list, file_name):
    """This function is used to add data to the file with this filename.
    """
    data = load_file_named(file_name)
    data.extend(data_list)  # if add in one item, instead of a list, use␣
 ↪append()
    save_data_as_file(data, file_name)
```

[6]:
```
# Functions used to requesting and querying with API

def call_api_with(url_extension):
    """This function is the basic function used to
       call the API with specific url extension.

    Returns:
        the json of the results.
    """
    your_company_house_api_key = "e1e62c1f-d5e1-41be-a050-633a93e91050"
    login_headers = {"Authorization": your_company_house_api_key}
    url = f"https://api.companieshouse.gov.uk/{url_extension}"
    res = requests.get(url, headers=login_headers)
    if res.status_code == 200:
        return res.json()
    elif res.status_code == 404:
        return None
    else:
        time.sleep(360)
        return call_api_with(url_extension)

# search company with specific query / keyword
def search_for_companies_with_query(query, number_of_companies=100):
    """This function is used to get the basic infomation with the query word.
    Returns:
        a list containning all the basic information of this query.
    """
    # for simplicity round up the number of returned companies
    #to the nearest hundred. eg. 130 becomes 200
    page_size = 100
    number_of_pages = math.ceil(number_of_companies / page_size)  # round up
    companies = []
    for page_index in range(0, number_of_pages):
        url = f"search/companies?
↪q={query}&items_per_page={page_size}&start_index={page_index * page_size}"
        json = call_api_with(url)
        if json is not None:
            companies += json.get('items', [])
    return companies

# request to get company data based on company number
def data_for_company(company_number):
    """This function is used to get the detail infomation
       with the company number.

    Returns:
        a list containning all the details of company with this number.
    """
    url = f"company/{company_number}"
    return call_api_with(url)

def all_persons_in_company(company_number):
    """This function is used to get all persons
       with significant control from company with this number.

    Returns:
        a list containning all persons.
    """
    url = f"company/{company_number}/persons-with-significant-control"
    json = call_api_with(url)
    if json is not None:
        return json.get('items', [])
    else:
        return None

def detailed_info_about_companies_with_ids(companies_numbers):
    """This function is used to get all details of companies with these numbers.

    Returns:
        a list containning all details of companies.
    """
```

```python
    results = []
    for company_number in companies_numbers:
        results.append(data_for_company(company_number))
    return results

def detailed_info_about_companies_with_name(name, how_many=100):
    """This function is used to get all details of companies a word or a name.

    Returns:
        a list containning all details of companies with this name.
    """
    # eg. unless otherwise stated, just grab 10 companies detailed info
    companies_basic_info = search_for_companies_with_query(name, how_many)
    companies_ids = [company['company_number']
                     for company in companies_basic_info]
    companies = detailed_info_about_companies_with_ids(companies_ids[:how_many])
    return companies
```

[7]:
```python
# function used to creat objects of Company with files

def creat_companies_with_files(companies_details_file,companies_persons_file):
    """This function is used to create objects with the type of Company.

    Args:
        companies_details_file: The details information of companies.
        companies_persons_file: The significant persons of companies.
    """
    companies_details = load_file_named(companies_details_file)
    companies_persons = load_file_named(companies_persons_file)
    companies = []
    for i in range(len(companies_details)):
        company_details = companies_details[i]
        significant_persons = companies_persons [i]
        companies.append(Company(company_details,significant_persons))
    return companies
```

[8]:
```python
# download the files with these query words for the future use

query_words_dict = {
        "tech": ['internet', 'technical', 'technology'],
        "manu": ['manufacturer', 'manufacturing', 'fabrication'],
        "farm": ["farm", "farmer", "farming"],
        "finance": ["financial", "consulting", "accounting"],
        "construction": ["construction", "design", "architecture"],
        "dessert":["dessert","cake","cookie","sweets",
                   "bagel","bakery","baking","bread"],
        "fish":["fish","fishing","fisher","seafood"],
        "energy":["oil","gas","energy","power","electricity"],
}

# dataloader_tech = DataLoader(query_words_dict['tech'],
#                              "tech_basic_info.txt",
#                              "tech_details.txt",
#                              "tech_persons.txt")

# dataloader_manu = DataLoader(query_words_dict['manu'],
#                              "manu_basic_info.txt",
#                              "manu_details.txt",
#                              "manu_persons.txt")

# dataloader_farm = DataLoader(query_words_dict['farm'],
#                              "farm_basic_info.txt",
#                              "farm_details.txt",
#                              "farm_persons.txt")

# dataloader_finance = DataLoader(query_words_dict['finance'],
#                                 "finance_basic_info.txt",
#                                 "finance_details.txt",
#                                 "finance_persons.txt")

# dataloader_construction = DataLoader(query_words_dict['construction'],
#                                      "construction_basic_info.txt",
#                                      "construction_details.txt",
#                                      "construction_persons.txt")

# dataloader_dessert = DataLoader(query_words_dict['dessert'],
#                                 "dessert_basic_info.txt",
#                                 "dessert_details.txt",
#                                 "dessert_persons.txt")

# dataloader_dessert = DataLoader(query_words_dict['fish'],
#                                 "fish_basic_info.txt",
#                                 "fish_details.txt",
#                                 "fish_persons.txt")

# dataloader_dessert = DataLoader(query_words_dict['energy'],
#                                 "energy_basic_info.txt",
#                                 "energy_details.txt",
#                                 "energy_persons.txt")
```

1 Business Question 1:

1.1 Which industires show the best performance in ethnic diversity considering the nationality of their significant persons?

1.1.1 Business Question 1: Code:

[9]:
```python
class Analyser_for_Ethnic:
    """This class is defined to analyze the ethnic diversity.

    Attributes:
        nationalities_for_companies_dict:
            Its keys is the name of a field like tech, manu, fish ect..
            Its values is  a dictionary which records the statistics
            of of the nationalities.
        normalized_value_dict:
            It records all the normalized values of
            each catogories of nationality diversity.
    """


    nationalities_for_companies_dict = {}
    normalized_value_dict = {}

    def analyse_with_file(self, details_file,persons_file, field):
        """This function is used to start the analysis with the file.

        Args:
            details_file:
                The list of the detailed information of these companies.
            persons_file:
                The list of the significant persons of these companies.

        """
        companies = creat_companies_with_files(details_file,persons_file)
        nationalities = [ company.recognize_nationalities()
                          for company in companies]
        # There are three categories of nationality diversity
        # one, two, three and more
        statistic_dict = {
            "one nationality":0,
            "two nationalities":0,
            "three and over three nationalities":0,
            "total":0,
                          }
        for i in nationalities:
            if i == 1:
                statistic_dict["one nationality"] += 1
            elif i == 2:
                statistic_dict["two nationalities"] += 1
            elif i >2:
                statistic_dict["three and over three nationalities"] += 1
        # Records the total number for the normalization
        total= reduce(lambda x,y:x+y,list(statistic_dict.values()))
        statistic_dict["total"] = total
        self.nationalities_for_companies_dict[field] = statistic_dict

    def normalization(self):
        """This function is used to conduct the normalization on each fields.
           Because they may have different number significant persons.
        """
        for key,value in self.nationalities_for_companies_dict.items():
            total = value["total"]
            self.normalized_value_dict[key] = list(map(lambda x:round(x/
↪total,3),list(value.values())))
```

[10]:
```
# declare an analyser first
analyser_ethnic = Analyser_for_Ethnic()

# add the farm and fish industry into analysis
analyser_ethnic.analyse_with_file("farm_details.txt",
                                  "farm_persons.txt",
                                  "farm")

analyser_ethnic.analyse_with_file("fish_details.txt",
                                  "fish_persons.txt",
                                  "fish")

# add the energy and construction industry into analysis
analyser_ethnic.analyse_with_file("energy_details.txt",
                                  "energy_persons.txt",
                                  "energy")

analyser_ethnic.analyse_with_file("construction_details.txt",
                                  "construction_persons.txt",
                                  "construction")

# add the technology and finance industry into analysis
analyser_ethnic.analyse_with_file("tech_details.txt",
                                  "tech_persons.txt",
                                  "tech")

analyser_ethnic.analyse_with_file("finance_details.txt",
                                  "finance_persons.txt",
                                  "finance")
```

```
# conduct the normalization
analyser_ethnic.normalization()
```

[11]:
```
pp.pprint(analyser_ethnic.nationalities_for_companies_dict)
```

```
{'construction': {'one nationality': 185,
                  'three and over three nationalities': 18,
                  'total': 274,
                  'two nationalities': 71},
 'energy': {'one nationality': 247,
            'three and over three nationalities': 29,
            'total': 380,
            'two nationalities': 104},
 'farm': {'one nationality': 137,
          'three and over three nationalities': 9,
          'total': 228,
          'two nationalities': 82},
 'finance': {'one nationality': 193,
             'three and over three nationalities': 15,
             'total': 276,
             'two nationalities': 68},
 'fish': {'one nationality': 202,
          'three and over three nationalities': 19,
          'total': 310,
          'two nationalities': 89},
 'tech': {'one nationality': 181,
          'three and over three nationalities': 18,
          'total': 253,
          'two nationalities': 54}}
```

[12]:
```
pp.pprint(analyser_ethnic.normalized_value_dict)
```

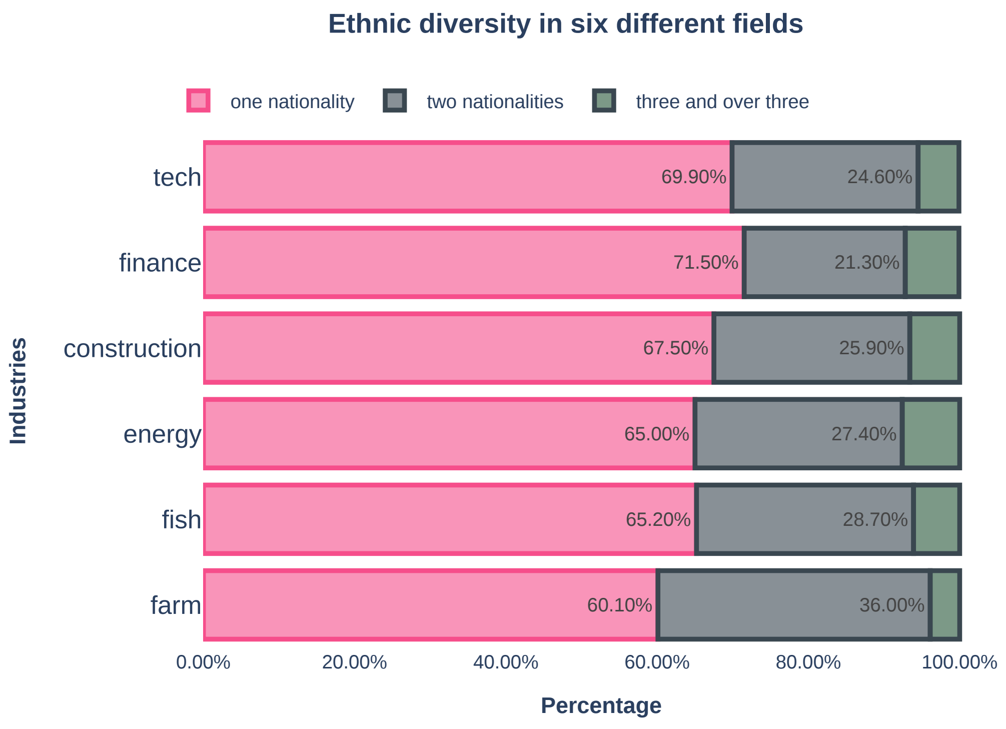
```
{'construction': [0.675, 0.259, 0.066, 1.0],
 'energy': [0.65, 0.274, 0.076, 1.0],
 'farm': [0.601, 0.36, 0.039, 1.0],
 'finance': [0.699, 0.246, 0.054, 1.0],
 'fish': [0.652, 0.287, 0.061, 1.0],
 'tech': [0.715, 0.213, 0.071, 1.0]}
```

[13]:
```
def draw_figure():

    y_data = ['farm','fish','energy','construction','finance','tech']
    fig = go.Figure()

    # add bar for one nationality
    fig.add_trace(go.Bar(
        y=y_data,
        x=[value[0] for value in analyser_ethnic.normalized_value_dict.
↪values()],
        name='one nationality',
        orientation='h',
        text = [value[0] for value in analyser_ethnic.normalized_value_dict.
↪values()],
        marker=dict(
            color='rgba(246, 78, 139, 0.6)',
            line=dict(color='rgba(246, 78, 139, 1.0)', width=3)
        ),
        texttemplate = '%{text:.2%}',
        textposition = "auto",
        legendrank=3
    ))

    # add bar for two nationalities
    fig.add_trace(go.Bar(
        y=y_data,
        x=[value[1] for value in analyser_ethnic.normalized_value_dict.
↪values()],
        name='two nationalities',
        orientation='h',
        text = [value[1] for value in analyser_ethnic.normalized_value_dict.
↪values()],
        marker=dict(
            color='rgba(58, 71, 80, 0.6)',
            line=dict(color='rgba(58, 71, 80, 1.0)', width=3)
        ),
        texttemplate = '%{text:.2%}',
        textposition = "auto",
        legendrank=2
    ))

    # add bar for three and more nationalities
    fig.add_trace(go.Bar(
        y=y_data,
        x=[value[2] for value in analyser_ethnic.normalized_value_dict.
↪values()],
        name='three and over three ',
        orientation='h',
        marker=dict(
            color='rgba(37, 85, 56, 0.6)',
            line=dict(color='rgba(58, 71, 80, 1.0)', width=3)
        ),
        legendrank=1
    ))
```

```
# set some format
fig.update_xaxes(tickformat=".2%")
fig.update_xaxes(title_text="<b>Percentage</b>")
fig.update_xaxes(showgrid=False)
fig.update_yaxes(showgrid=False)
fig.update_yaxes(title_text="<b>Industries</b>",tickfont = dict(size=16))

# set the layout format
fig.update_layout(
    barmode='stack',
    title_text="<b>Ethnic diversity in six different fields</b>",
    title_x=0.50,
    title_y=0.95,
    paper_bgcolor='rgba(0,0,0,0)',
    plot_bgcolor='rgba(0,0,0,0)',
    legend=dict(orientation="h",
                yanchor="bottom",
                y=1.02,
                xanchor="right",
                x=0.78)
)

fig.show()
```

2 Business Question 1: Mini-report and visualisation:

[14]:
```
draw_figure()
```



Loading [MathJax]/extensions/MathMenu.js

2.1 Backgroud

Along with the integration of the international market and the globalization of economic activities, more and more companies rely on international trade to expand their business and enhance their competitiveness. In this case, whether an enterprise has an inclusive and diversified culture becomes a key point that decides whether they could survice in the fierce market competition. Among a series of factors that could be used to quantify the diversity of a company, checking the ethnic of its owner or stakeholder may be an indirect but effective way.

2.2 Description

Which industries show the best performance in ethnic diversity considering the nationality of their significant persons? Six fields of industry are included, all the data is divided into three parts according to the number of nationalities with their significant persons.

2.3 Analysis

It's quite amazing that the farm industry shows the best performance in ethnic diversity. About 40% of farm companies have significant persons with over two nationalities. On the contrary, technology industry, as one of emerging industry, show the smallest percentage. Only about 30% of them contain owners of stakeholders with over two nationalities. On the other hand, when it comes to having persons with over three nationalities, the energy industry and finance industry show an impressive performance with a percentage of 7.6% and 7.1% respectively.

2.4 Conclusion

Primary industries like the farm industry and fishing industry show the best performance in ethnic diversity, followed by secondary industries like the construction industry and energy industry. The tertiary industries such as the technology industry and finance industries show the least diversity on average, though some of the head international companies may perform well in inclusion and diversity in the ownership.

3 Business Question 2:

3.1 Which tims is a good choice to start a tourism business? Will this month change over years?

3.1.1 Business Question 2: Code:

[15]:
```
# define some useful functions for further operations

def get_month(date):
    year_str,month_str,day_str = date.split('-')
    return month_str

def get_year(date):
    year_str,month_str,day_str = date.split('-')
    return year_str

def counter(list_to_be_counted):
    result_dict = {}
    for key in list_to_be_counted:
        result_dict[key] = result_dict.get(key,0)+1
    return result_dict

def export_as_oder(result_dict,oder):
    output_list = []
    for key in oder:
        output_list.append(result_dict[key])
    return output_list
```

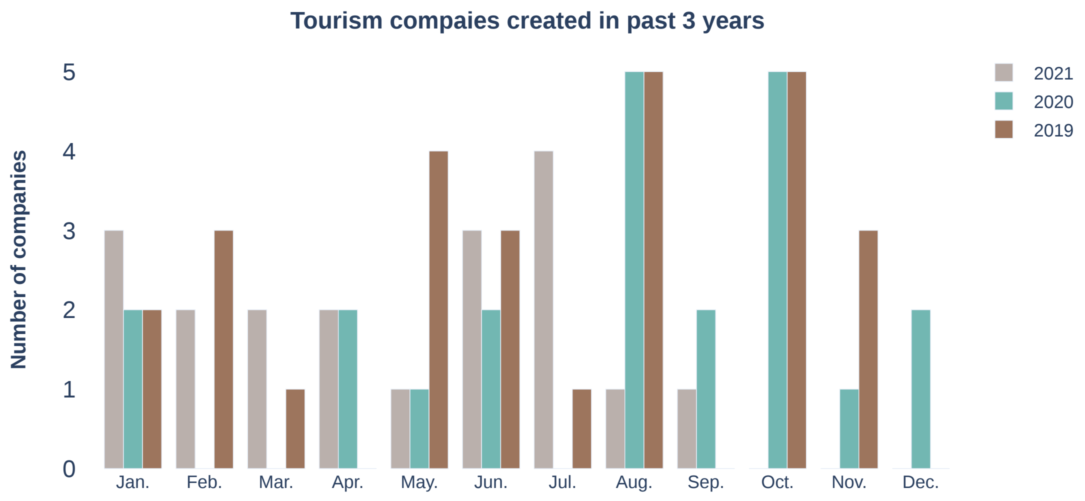
[16]:
```
class Analyser_date:
    """This class is defined to analyze the date distribution.

    Attributes:
        queried_companies:
            The list to save all the queried Company objects.
        target_sic_codes:
            A list of sic codes used to filter companies.
        target_companies:
            Filtered companies.
        dissolved_target_companies:
            Target companies that got dissolved.
        dissolved_months_dict:
            A list that contains all the months that target companies closed.
        create_months_dict:
            A list that contains all the months that target companies created.
    """
    queried_companies = []
    target_sic_codes = []
    target_companies = []
    dissolved_target_companies = []
    dissolved_months_dict = {}
    create_months_dict = {}

    def __init__(self,details_filename,persons_filename,target_sic_codes):
        """This initialization function.

        Args:
            details_file:
                The list of the detailed information of these companies.
            persons_file:
                The list of the significant persons of these companies.
            target_sic_codes:
                Target sic codes, used for filter.
        """
        self.queried_companies =␣
↪creat_companies_with_files(details_filename,persons_filename)
        self.target_sic_codes = target_sic_codes
        self.target_companies = list(filter(self.is_target,self.
↪queried_companies))
        self.dissolved_target_companies = [company
                                    for company in self.target_companies
                                    if company.status == 'dissolved' ]

    # function used for filter
    def is_target(self,company):
        sic_codes = company.sic_codes
        # if this two list have intersections, it is target, else not
        result = list(set(self.target_sic_codes)&set(sic_codes))
        if result:
            return True
        else:
            return False

    def summarize_dissoveld_month(self,year):
        """This function is used to summarize the months of cessation.

        Args:
            year: The key of this summary dictionary.
        """
        dissolved_months = [get_month(company.date_of_cessation)
                            for company in self.dissolved_target_companies
                            if get_year(company.date_of_cessation) == year]
        summary_dcit = {'01':0,'02':0,'03':0,'04':0,'05':0,'06':0,
                        '07':0,'08':0,'09':0,'10':0,'11':0,'12':0,}

        # use counter function to count the number of the occurence of months
        stat_dict = counter(dissolved_months)
        for key,value in stat_dict.items():
            summary_dcit[key] = value

        self.dissolved_months_dict[year] = summary_dcit

    def summarize_create_month(self,year):
        """This function is used to summarize the months of creation.

        Args:
            year: The key of this summary dictionary.
        """
        create_months = [get_month(company.date_of_creation)
                            for company in self.target_companies
                         if get_year(company.date_of_creation) == year]
        summary_dcit = {'01':0,'02':0,'03':0,'04':0,'05':0,'06':0,
                        '07':0,'08':0,'09':0,'10':0,'11':0,'12':0,}

        # use counter function to count the number of the occurence of months
        stat_dict = counter(create_months)
        for key,value in stat_dict.items():
            summary_dcit[key] = value

        self.create_months_dict[year] = summary_dcit
```

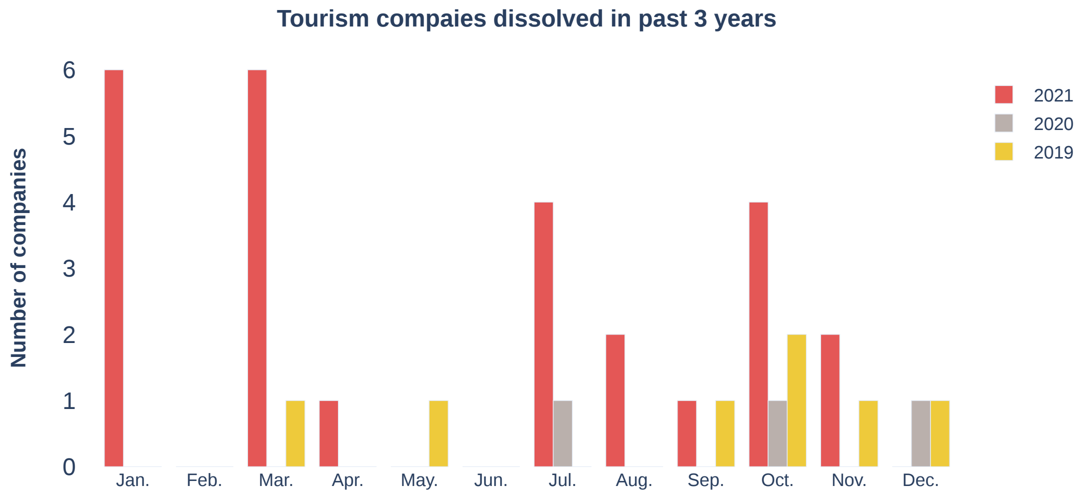
[17]:
```
# firstly, let's declare a analyser for tourism business
analyser_tour = Analyser_date("tourism_details.txt",
                              "tourism_persons.txt",
                              ['79110','79120','79901','79909'])

# let's get the cessation dates for past three years
analyser_tour.summarize_dissoveld_month('2021')
analyser_tour.summarize_dissoveld_month('2020')
analyser_tour.summarize_dissoveld_month('2019')

# let's get the creation dates for past three years
analyser_tour.summarize_create_month('2021')
analyser_tour.summarize_create_month('2020')
analyser_tour.summarize_create_month('2019')
```

```
[29]: def draw_graph():
    months=['Jan.', 'Feb.', 'Mar.','Apr.','May.','Jun.',
            'Jul.','Aug.','Sep.','Oct.','Nov.','Dec.']
    oder = ['01','02','03','04','05','06',
            '07','08','09','10','11','12']

    fig = make_subplots(rows=2, cols=1, shared_xaxes=False,
        subplot_titles=("<b>Tourism compaies created in past 3 years</b>",
                        "<b>Tourism compaies dissolved in past 3 years</b>",),
        vertical_spacing= 0.10,)

    fig.append_trace(
        go.Bar(name='2021',
               x=months,
               y=export_as_oder(analyser_tour.create_months_dict['2021'],oder),
               legendgroup = '2',
               marker_color=px.colors.qualitative.T10[9],),1,1
    )
    fig.append_trace(
        go.Bar(name='2020',
               x=months,
               y=export_as_oder(analyser_tour.create_months_dict['2020'],oder),
               legendgroup = '2',
               marker_color=px.colors.qualitative.T10[3]),1,1
    )
    fig.append_trace(
        go.Bar(name='2019',
               x=months,
               y=export_as_oder(analyser_tour.create_months_dict['2019'],oder),
               legendgroup = '2',
               marker_color=px.colors.qualitative.T10[8]),1,1
    )

    fig.append_trace(
        go.Bar(name='2021',
               x=months,
               y=export_as_oder(analyser_tour.
 ↪dissolved_months_dict['2021'],oder),
               legendgroup = '1',
               marker_color=px.colors.qualitative.T10[2],),2,1
    )
    fig.append_trace(
        go.Bar(name='2020',
               x=months,
              y=export_as_oder(analyser_tour.
↪dissolved_months_dict['2020'],oder),
              legendgroup = '1',
              marker_color=px.colors.qualitative.T10[9],),2,1
    )
    fig.append_trace(
        go.Bar(name='2019',
              x=months,
              y=export_as_oder(analyser_tour.
↪dissolved_months_dict['2019'],oder),
              legendgroup = '1',
              marker_color=px.colors.qualitative.T10[5],),2,1
    )

    fig.update_layout(
        height=800,
        width=750,
        legend_tracegroupgap = 300,
        paper_bgcolor='rgba(0,0,0,0)',
        plot_bgcolor='rgba(0,0,0,0)',
    )

    fig.update_yaxes(title_text="<b>Number of companies</b>",
                    tickfont = dict(size=16),ticks =␣
↪"outside",tickcolor='white', ticklen=10)

    fig.update_annotations(yshift=10)

    fig.show()
```

3.1.2 Business Question 2: Mini-report and visualisation:

[30]:
```
draw_graph()
```





3.2 Backgrouds

Suppose that you are an investor, recently an entrepreneur brought his proposal of establishing a new tour agency company to you and claimed that it's right the time to start a tourism business, should you trust him or not?

3.3 Description

As we all know that sometimes the success of a certain kind of business would highly rely on the timing you start it. Unfortunately, the tourism business must be one of them as people's passion to take a journey fluctuates during the whole year. So maybe by analyzing the creation and cessation of tourism companies during the past 3 years we may get some inspiration.

3.4 Analysis

It's quite obvious that most tourism companies were created in August and October in 2019 and 2020. It is deduced that they tried to catch the peak of tourists during the summer holiday and new year holiday. Besides, May also seemed to be good timing in 2019. However, these all changed in 2021, it seems that June and July became a better choice this year. Combining the second figure, it can be seen that the total number of created tourism companies is decreasing and more and more companies got dissolved. It is speculated that due to the constant influence of Covid-19 and the risk in policy, the whole market is holding a passive attitude.

3.5 Conclusion

Though in the past several years, August and October may be a good time to start a tourism business, due to the influence of covid-19, it may not be a good timing this year.

4 Business Question 3:

4.1 Whether the companies whose persons with significant control have a PhD degree would survive for longer time in the fierce market competition?

4.1.1 Business Question 3: Code:

[20]:
```
# Code starts here
```

[21]:
```
# some functions used to do the filter and culculation

def get_companies_with_PhD(target_companies):
    companies_with_phd = list(
        filter(lambda company: company.get_number_of_doctoral_owner() > 0,␣
 ↪target_companies)
    )
    return companies_with_phd

def get_companies_without_PhD(target_companies):
    companies_without_phd = list(
        filter(lambda company: company.get_number_of_doctoral_owner() == 0,␣
 ↪target_companies)
    )
    return companies_without_phd

def get_average_age(companies):
    return round(sum(company.age for company in companies )/len(companies),2)
```

[22]:
```
class Analyser_phd:
    """This class is defined to analyze the relationship
       between phd owners and lifetime of companies.
    Attributes:
        average_age_with_phd_owners:
            A dict contains the age of companies with phd in different fields
        average_age_without_phd_owners:
            A dict contains the age of companies without phd in different fields
        phd_rate:
            A dict contains the phd owner rate  in different fields
    """
    average_age_with_phd_owners = {}
    average_age_without_phd_owners = {}
    phd_rate = {}

    def analyse_with_file(self, details_file,persons_file, field):
        """This function is used to add file to the analyser.

        Args:
            details_file:
                The list of the detailed information of these companies.
            persons_file:
                The list of the significant persons of these companies.
            target_sic_codes:
                Target sic codes, used for filter.
        """
        companies = creat_companies_with_files(details_file,persons_file)
        companies_with_phd = get_companies_with_PhD(companies)
        companies_without_phd = get_companies_without_PhD(companies)

        self.average_age_with_phd_owners[field] =␣
↪get_average_age(companies_with_phd)
        self.average_age_without_phd_owners[field] =␣
↪get_average_age(companies_without_phd)
        self.phd_rate[field] = round(len(companies_with_phd) /␣
↪len(companies_without_phd), 3)
```

[23]:
```
# declare an instance for this Analyser_phd
analyser = Analyser_phd()

# let's add the fields we focus and the relevant files as inputs
analyser.analyse_with_file("tech_details.txt","tech_persons.txt","tech")
analyser.analyse_with_file("manu_details.txt","manu_persons.txt","manu")
analyser.analyse_with_file("farm_details.txt","farm_persons.txt","farm")
analyser.analyse_with_file("finance_details.txt","finance_persons.
↪txt","finance")
analyser.analyse_with_file("construction_details.txt","construction_persons.
↪txt","construction")
```

[24]:
```
print(analyser.average_age_with_phd_owners)
```

```
{'tech': 15.83, 'manu': 14.0, 'farm': 9.0, 'finance': 8.33, 'construction': 2.0}
```

[25]:
```
print(analyser.average_age_without_phd_owners)
```

```
{'tech': 9.91, 'manu': 11.03, 'farm': 9.01, 'finance': 8.69, 'construction':
7.09}
```

[42]:
```
def draw_picture():

    # Create figure with secondary y-axis
    fig = make_subplots(specs=[[{"secondary_y": True}]])
    x_data = list(analyser.average_age_with_phd_owners)

    #x_data = ["tech", "manu", "farm","finance","construction"]
    with_phd_age = list(analyser.average_age_with_phd_owners.values())
    without_phd_age = list(analyser.average_age_without_phd_owners.values())
    phd_percentage = list(analyser.phd_rate.values())

    # Add traces
    fig.add_trace(
        go.Bar(
            x=x_data,
            y=with_phd_age,
            name="with phd age",
            text = ["15.8","14.0","9.0","8.33","2.0"],
            textposition='inside',
            marker_color=px.colors.qualitative.D3[0],
            ),
        secondary_y=False,
    )

    fig.add_trace(
        go.Bar(
            x=x_data,
            y=without_phd_age,
            name="without phd age",
            text = ["9.91","11.03","9.01","8.69","7.09"],
            marker_color=px.colors.qualitative.Pastel2[7],
            textposition='inside',
        ),

        secondary_y=False,
    )

    fig.add_trace(
        go.Scatter(
            x=x_data,
            y=phd_percentage,
            name="phd_percentage",
            marker_color=px.colors.qualitative.Dark24[5],
        ),
        secondary_y=True,
    )

    # Add figure title
    fig.update_layout(
        title_text="Average ages of companies with or without phd owners and␣
↪its percentage",
        title_x=0.45,
        title_y=0.90,
        paper_bgcolor='rgba(0,0,0,0)',
        plot_bgcolor='rgba(0,0,0,0)'
    )

    # Set x-axis title
    fig.update_xaxes(title_text="<b>Industies</b>")

    # Set y-axes titles
    fig.update_yaxes(title_text="<b>Avergage Companies age</b>",␣
↪secondary_y=False)
    fig.update_yaxes(title_text="<b>PhD percentage</b>", secondary_y=True)

    # set some format
    fig.update_yaxes(tickformat=".2%",secondary_y=True)
    fig.update_xaxes(showgrid=False)
    fig.update_yaxes(showgrid=False)

    fig.show()
```

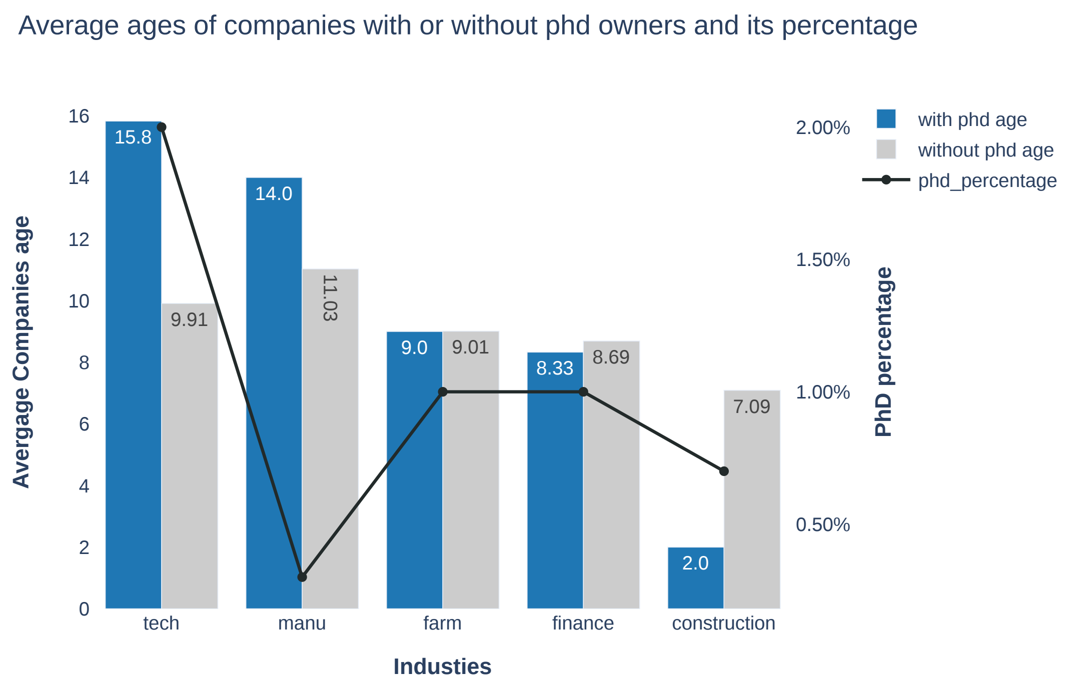
Firstly find the most fierce competition industry (quantified by age of companies)

Check that whether their significant persons have a PhD degree. If so, how about the percentage, whether the higher percentage lead to longer survival or have littile business with it

4.1.2 Business Question 3: Mini-report and visualisation:

5 Mini-report

[43]:
```
draw_picture()
```



5.1 Backgrouds

Suppose that you are an investor, someone tries to convince you with his/her doctoral degree that his/her start-ups would be the most promising ones that deserve an angel investment as he/she is quite professional in this area. Should you trust them or not?

5.2 Description

When it comes to doctor degree, there is an inherent impression that those people who earned a "Dr" title seems to be the most wise, professional, and outstanding ones in our humankind as all of them at least have had put, we human's exploration to the world, a small step forward. Thus, it's quite interesting that if these well-educated ones try to create or manage a company. Will they bring great benefits to the operation of the companies? Or maybe their talents are not suitable in this area?

5.3 Analysis

The figure shows that in the technology, and manufacturing industries, companies led by Ph.D. persons tend to live a longer time. The biggest gap occurs in tech companies which is about 6 years long, then about 3 years in the manufacturing area. This is reasonable, as PhD leaders tend to be good at using their knowledges to help their companies overcome difficulties in these technology-driven areas.

However, this effect seems doesn't work in the left three industries. The lifetime is even in farm and finance companies with or without Ph.D. leaders and a reverse gap shows in the construction industry. It is speculated that these three areas put high requirements on the practical experience thus the theoretical knowledge of PhDs may not show its power.

5.4 Conclusion

Whether having a Ph.D.leader is a favorable factor to a start-up or not depends on its area. In industries that emphasize technologies and researches, a Ph.D. owner would definitely be good at leading this company further. However, in those industries that stress practical experience, a doctoral degree may not be a benefit, or at least may not be a big advantage.