**Tutorial for Data Preprocessing**

The questions are as follows:

* How can we get data at once from a public website?
* How can we extract structured data from pdf files?

# **Download PDF Files**

An inquiry example is as follows:

|  |
| --- |
| https://dnn.beckshybrids.com/YieldDataHandler16.ashx?zipcode=46031&mileRadius=1500&family=1&product=&year=2019&challenge=0&method=getmapmarkers |

When users input the year of interest, the website will return the inquired results. However, the website developer will require users to click marker one by one. Fortunately, we can try to get the data (in JSON format) at once with the following codes:

|  |
| --- |
| Python codes |
| import os.path  import traceback  def do\_something\_wrong():  cc = int("come on!")  return cc  def capture\_Error():  try: # first catch  do\_something\_wrong()  except Exception as ex:  print(f"The Exception is here:\n{ex}")  try: # second catch  do\_something\_wrong()  except:  print(f"Use traceback.format\_exc() instead:\n{traceback.format\_exc()}")  ##  # The first catch would only display  # The Exception is here:  # invalid literal for int() with base 10: 'come on!'  #while the second catch includes not only the error but where it occurs  ## the second catch includes not only the error but where it occurs  ## In a large software project, the second example would be way more helpful than the first.  import requests  import csv  # The URL to send the request to  # Cron: 2019  #url = "https://dnn.beckshybrids.com/YieldDataHandler16.ashx?zipcode=46031&mileRadius=1500&family=1&product=&year=2019&challenge=0&method=getmapmarkers"  # Soybean: 2019  # url = https://dnn.beckshybrids.com/YieldDataHandler16.ashx?zipcode=46031&mileRadius=1500&family=1&product=&year=2019&challenge=0&method=getmapmarkers  def parstItemsFromURL(url,yearStr):  # Send a GET request to the URL  response = requests.get(url)  # Check if the request was successful  if response.status\_code == 200:  # Parse the JSON response  data = response.json()  # Define the CSV file name  csv\_file = "TestPlotId\_"+yearStr+".csv"  # if os.path.exists(csv\_file):  # os.remove(csv\_file)  # else:  # pass  # Open the CSV file in write mode  with open(csv\_file, mode='a', newline='') as file:  writer = csv.writer(file)  # Write the header row  writer.writerow(["TestPlotId", "Latitude", "Longitude", "Address", "Name", "YearCode"])  # Write the data rows  for item in data:  test\_plot\_id = item.get('TestPlotId')  latitude = item.get('Latitude')  longitude = item.get('Longitude')  address = item.get('Address')  name = item.get('Name')  year\_code = item.get('YearCode')  # Write the data to the CSV file  writer.writerow([test\_plot\_id, latitude, longitude, address, name, year\_code])  print(f"Data saved successfully to {csv\_file}")  else:  print(f"Failed to retrieve data. Status code: {response.status\_code}")  return 1  # if \_\_name\_\_ == "\_\_main\_\_":  # capture\_Error()  def deletePreviousCSV(yearStr):  csv\_file = "test\_plots\_data\_" + yearStr + ".csv"  if os.path.exists(csv\_file):  os.remove(csv\_file)  else:  pass  if \_\_name\_\_ == "\_\_main\_\_":  print("Start downloading pdf files")  # url = "https://dnn.beckshybrids.com/YieldDataHandler16.ashx?zipcode=46031&mileRadius=1500&family=1&product=&year=2019&challenge=0&method=getmapmarkers"  urlBasicStr = "https://dnn.beckshybrids.com/YieldDataHandler16.ashx?zipcode=46031&mileRadius=1500&family=1&product=&year=2019&challenge=0&method=getmapmarkers"  familyIDLis = ["1","2","3"]  yearList = ["2019","2020","2021","2022"]  # familyID = "1"  # yearStr = "2020"  for yearStr in yearList:  deletePreviousCSV(yearStr)  for yearStr in yearList:  for familyID in familyIDLis:  url = "https://dnn.beckshybrids.com/YieldDataHandler16.ashx?zipcode=46031&mileRadius=1500&family=" + familyID + "&product=&year=" + yearStr + "&challenge=0&method=getmapmarkers"  print("it is processing:{}".format(url))  parstItemsFromURL(url, yearStr) |

# **Digitalize PDF files**

We can write python codes to extract useful information from pdf files according to the structure. For instance, we want to get the planting and harvesting dates from a pdf file.

A screenshot of a computer

Description automatically generated

A screenshot of a document

Description automatically generated

|  |
| --- |
| **Python codes: to extract specific items** |
| import pdfplumber  import re  with pdfplumber.open(pdf\_path) as pdf:  # Assuming the information is on the first page  page = pdf.pages[0]  text = page.extract\_text()  print(text)  # Regular expressions to find PLANTED and HARVESTED dates  planted\_match = re.search(r'PLANTED:\s\*(\d{2}/\d{2}/\d{4})', text)  harvested\_match = re.search(r'HARVESTED:\s\*(\d{2}/\d{2}/\d{4})', text)  planted\_date = planted\_match.group(1).strip() if planted\_match else None  harvested\_date = harvested\_match.group(1).strip() if harvested\_match else None  # Use regular expression to find the content between start\_keyword and end\_keyword  pattern = rf'{"SOIL TEXTURE"}:\s\*(.\*?)\s\*{"HERBICIDE"}:'  soil\_texture\_match = re.search(pattern, text, re.DOTALL)  soil\_texture = soil\_texture\_match.group(1).strip() if soil\_texture\_match else "Content not found." |

When we use python to digitize a pdf file, we will get the data organized irregularly. To solve this problem, for the table, we can read line by line and merge elements if necessary. More importantly, we need to carefully check the data structure and then write codes to organize it. An example is offered as follows:

|  |
| --- |
| **Python codes: iterate through pdf files and extract information and save it into csv files.** |
| ###Creator: Xinghua Cheng  ###Date: August 29, 2024  ###  import glob  import pdfplumber  import re  import pandas as pd  import os  def merge\_elements(lst):  # Step 1: Find the index of '--'  # try:  # o\_index = lst.index('--')  # except ValueError:  # raise ValueError("'--' not found in the list")  #  #  o\_index = 0  try:  # Find the index of the last '--'  o\_index = len(lst) - 1 - lst[::-1].index('--')  print(f"The index of the last '--' is: {lst}")  except ValueError:  print("The list does not contain '--'.")  # Step 2: Determine the range to merge (from the start to O-5)  merge\_start = 0  merge\_end = o\_index - 6  # Step 3: Merge the elements from merge\_start to merge\_end into a single string  merged\_element = '\_'.join(lst[merge\_start:merge\_end + 1])  # Step 4: Construct the new list  new\_list = [merged\_element] + lst[merge\_end + 1:]  return new\_list  def merge\_companyBrand(lst):  # Determine the range to merge (from the start to O-5)  merge\_start = 0  merge\_end = len(lst)-11  # Step 3: Merge the elements from merge\_start to merge\_end into a single string  merged\_element = '\_'.join(lst[merge\_start:merge\_end + 1])  # Step 4: Construct the new list  new\_list = [merged\_element] + lst[merge\_end + 1:]  return new\_list  def getCropType(dfCropYield):  cropType = "Corn"  #get the column of Yield1 and get the average and take it as aveYield  # Convert the 'yield1' column from strings to numbers  Yield1 = dfCropYield['actual\_yield1']  Yield1  dfCropYield['actual\_yield1\_numeric'] = pd.to\_numeric(dfCropYield['actual\_yield1'], errors='coerce')  # Calculate the average of the 'yield1' column  average\_yield = dfCropYield['actual\_yield1\_numeric'].mean()  print("the average\_yield is:{}".format(average\_yield))  #if aveYield >=100, it is Soybean  #if aveYield >, it is Corn  if average\_yield>=100:  cropType = "Corn"  elif average\_yield<100: # I just open pdf files and determine the value with empirical experience  cropType = "Soybean"  else:  cropType = "TBD\_type"  print("The Crop Type is:{}".format(cropType))  print("To ensure the crop type is correct, please also go to check the planting date.")  return cropType  #The function that aims to extract information from Beck PDF Files.  def extract\_information\_from\_pdf(pdf\_path):  def count\_items\_in\_line(line):  # Split the line by spaces and count non-empty items  items = [item for item in line.split() if item]  return len(items)  with pdfplumber.open(pdf\_path) as pdf:  # Assuming the information is on the first page  page = pdf.pages[0]  text = page.extract\_text()  print(text)  # file1 = open("myfile.txt", "w")  # file1.write(text)  # file1.close()  # Extract the table (assuming there's only one table on the page)  # Extract the first page  first\_page = pdf.pages[0]  # Split the extracted text into lines  lines = text.splitlines()  # Find the line where "Company" is included  # company\_line = next((line for line in lines if "Company" in line), "Company not found.")  # print(f"Line containing 'Company': {company\_line}")  # Find the index of the line where "Company" is included  # We will got the case "Fertilzer Service Company" in the first line  company\_line\_index = next((index for index, line in enumerate(lines) if "Brand" and "Company" in line and count\_items\_in\_line(line)>=5), -1)  header\_items\_count = count\_items\_in\_line(lines[company\_line\_index])  value\_lines = []  if company\_line\_index != -1:  # Get the number of items in the header line  header\_items\_count = count\_items\_in\_line(lines[company\_line\_index])  # Extract all subsequent lines that correspond to the values for those fields  for line in lines[company\_line\_index + 1:]:  # Check if the line has the same number of items as the header line  if count\_items\_in\_line(line) >= header\_items\_count and "--" in line:  value\_lines.append(line)  else:  # Break if the line does not match the expected number of items  break  # Output the header and the corresponding value lines  print(f"Header Line: {lines[company\_line\_index]}")  print("Value Lines:")  for value\_line in value\_lines:  print(value\_line)  else:  print("Line containing 'Company' not found.")  # Split the header line into columns  columns = [item for item in lines[company\_line\_index].split() if item]  lenthColumns = len(columns)  columnsNew = merge\_companyBrand(columns)  pathPDF, fileNamePDF = os.path.split(pdf\_path)  columnsNew.append('PDF\_file')  # Split each value line into a list of values  data = []  for value\_line in value\_lines:  if True: #count\_items\_in\_line(value\_line) >= header\_items\_count:  #we need to process it  valueLis = [item for item in value\_line.split() if item]  lenthvalueLis = len(valueLis)  # for instance:  # BECK'S EX 2909 (5507 genetics)\*  # we need to change it to [BECK'S,EX\_2909\_(5507 genetics)\*,16, 8 ,556, 20.2, 56.3, --, 214.5, -6.1, 208.4, 14]  valueLisNew = []  # if lenthvalueLis == lenthColumns:  # #we do not to need the first two column  # valueLisNew = valueLis  #  # else:  valueLisNew = merge\_companyBrand(valueLis)  valueLisNew.append(pdf\_path)  data.append(valueLisNew)  # we can take the maximum number of items as the number of cloumn, so we can save result as csv  # Create a DataFrame using the columns and data  dfCropYield = pd.DataFrame(data, columns=columnsNew)  ###############################################################################  # Check with df and see if it is either Crop or Soybean  ###############################################################################  # Rename the columns  columns = dfCropYield.columns.tolist()  first\_yield1\_index = columns.index('Yield1')  second\_yield1\_index = first\_yield1\_index + columns[first\_yield1\_index + 1:].index('Yield1') + 1  dfCropYield.columns.values[first\_yield1\_index] = 'actual\_yield1'  dfCropYield.columns.values[second\_yield1\_index] = 'adj\_yield1'  # # Display the DataFrame  print("data frame for the pdfFile:{}".format(pdf\_path))  print(dfCropYield)  crop\_type = getCropType(dfCropYield) #it is either Corn or Soybean in the U.S. Corn belt  crop\_type  ###############################################################################  # Save the information of crop yield and brand  ###############################################################################  crop\_yield\_brand\_csv = pathPDF+"\\"+"Beck\_plot\_"+fileNamePDF[:-4] +"\_crop\_yield\_brand"+ "\_"+ crop\_type + ".csv"  if os.path.exists(crop\_yield\_brand\_csv):  os.remove(crop\_yield\_brand\_csv)  else:  pass  dfCropYield.to\_csv(crop\_yield\_brand\_csv, index=False)  print("the information of crop yield and brand is saved in :{}".format(crop\_yield\_brand\_csv))  ###############################################################################  # Extract information from the first "table" from a pdf file  ###############################################################################  # Regular expression to find the "County, State" pattern  county\_state\_match = re.search(r'(\w+),\s+(\w{2})', text)  county\_state = county\_state\_match.group(0).strip() if county\_state\_match else None  # Regular expressions to find PLANTED and HARVESTED dates  planted\_match = re.search(r'PLANTED:\s\*(\d{2}/\d{2}/\d{4})', text)  harvested\_match = re.search(r'HARVESTED:\s\*(\d{2}/\d{2}/\d{4})', text)  planted\_date = planted\_match.group(1).strip() if planted\_match else None  harvested\_date = harvested\_match.group(1).strip() if harvested\_match else None  # Correct regular expression for PREVIOUS CROP  previous\_crop\_match = re.search(r'PREVIOUS CROP:\s\*(\w+)', text)  previous\_crop = previous\_crop\_match.group(1).strip() if previous\_crop\_match else None  # Correct regular expression for tillage  # tillage\_match = re.search(r'TILLAGE:\s\*(\w+)', text)  # tillage = tillage\_match.group(1).strip() if tillage\_match else None  # Use regular expression to find the content after "TILLAGE:"  tillage\_match = re.search(r'TILLAGE:\s\*(.\*)', text)  tillage = tillage\_match.group(1).strip() if tillage\_match else "Content not found."  insecticide\_match = re.search(r'INSECTICIDE:\s\*(.\*)', text)  insecticide = insecticide\_match.group(1).strip() if insecticide\_match else "Content not found."  # Use regular expression to find the content between start\_keyword and end\_keyword  pattern = rf'{"SOIL TEXTURE"}:\s\*(.\*?)\s\*{"HERBICIDE"}:'  soil\_texture\_match = re.search(pattern, text, re.DOTALL)  soil\_texture = soil\_texture\_match.group(1).strip() if soil\_texture\_match else "Content not found."  pattern2 = rf'{"ROW WIDTH"}:\s\*(.\*?)\s\*{"PREVIOUS CROP"}:'  row\_width\_match = re.search(pattern2, text, re.DOTALL)  row\_width = row\_width\_match.group(1).strip() if row\_width\_match else "Content not found."  return county\_state, planted\_date, harvested\_date,previous\_crop,tillage,soil\_texture,insecticide,row\_width,crop\_type  ###Goal: To determine whether this file was generated for Corn or Soybean.  def iterate\_through\_pdf\_files(pdfFileInter):  #Step 1: read pdf files and get the items of interest  pdfFileInterLis = glob.glob(pdfFileInter+"\\"+"\*.pdf")  #pdfFileInterLis = [pdfFileInter+"\\"+"94338.pdf"]  for pdfFile in pdfFileInterLis:  path,fileName = os.path.split(pdfFile)  print("The file: {}".format(pdfFile)+"is being processed:\n")  # Example usage  pdf\_path = pdfFile #'pdf\_downloads\_2019/94456.pdf'  County\_State, PLANTED\_DATE, HARVESTED\_DATE, PREVIOUS\_CROP, TILLAGE, SOIL\_TEXTURE, INSECTICIDE, ROW\_WIDTH ,CROP\_TYPE = extract\_information\_from\_pdf(pdf\_path)  print(f"County, State(abbreviation): {County\_State}")  print(f"PLANTED Date: {PLANTED\_DATE}")  print(f"HARVESTED Date: {HARVESTED\_DATE}")  print(f"PREVIOUS CROP: {PREVIOUS\_CROP}")  print(f"TILLAGE: {TILLAGE}")  print(f"SOIL TEXTURE: {SOIL\_TEXTURE}")  print(f"INSECTICIDE: {INSECTICIDE}")  print(f"ROW WIDTH: {ROW\_WIDTH}")  PDFCSV = pdfFile  plotID = fileName[:-4] #94456.pdf -> 94456  columnsCSV = ["County\_State", "PLANTED\_DATE", "HARVESTED\_DATE", "PREVIOUS\_CROP", "TILLAGE", "SOIL\_TEXTURE", "INSECTICIDE", "ROW\_WIDTH" ,"CROP\_TYPE","plotID"]  data = [County\_State, PLANTED\_DATE, HARVESTED\_DATE, PREVIOUS\_CROP, TILLAGE, SOIL\_TEXTURE, INSECTICIDE, ROW\_WIDTH ,CROP\_TYPE,plotID]  df = pd.DataFrame([data], columns = columnsCSV)  savedResultsCSV = pdfFileInter + "\\" + "Beck\_" + plotID + "\_" + "plan\_harvest\_managment\_"+ CROP\_TYPE + ".csv"  if df.empty:  print("")  else:  if os.path.exists(savedResultsCSV):  os.remove(savedResultsCSV)  else:  pass  #save results  df.to\_csv(savedResultsCSV, index=False)  print("the information of management (planting, harvesting, and tillage etc)is saved in :{}".format(savedResultsCSV))  ###csv records  ## pdf\_plotid\_#\_pdf\_crop\_type.csv  ## for example: Year\_2019\_Plotid\_94141\_Corn.csv  ## County\_State,PLANTED\_DATE, HARVESTED\_DATE, PREVIOUS\_CROP, TILLAGE, SOIL\_TEXTURE, INSECTICIDE, ROW\_WIDTH, PDFCSV, CROP\_TYPE,  #save results as a dataframe  return 1  if \_\_name\_\_ == "\_\_main\_\_":  mainPath = "C:\\Users\\xic23015\\OneDrive - University of Connecticut\\pythonProject\\crop\_phenology\_500m\_merge\\process\_SG-FG-images\\Postprocessing"  yearLis = ["2019","2020","2021","2022"] #["2019","2020","2021","2022"]  pdfFileFolder = "pdf\_downloads\_" #For example, "pdf\_downloads\_2019"  for yearStr in yearLis:  pdfFileFolder = "pdf\_downloads\_" + yearStr  pdfFileInter = mainPath + "\\" + pdfFileFolder  iterate\_through\_pdf\_files(pdfFileInter)  print("\n")  print("Finished! no errors!")  print("please go forward and synthesize all csv files!") |