Farm Management Information System for The McGee’s Christmas Tree Farm

A Report to:

(Elena Cherkaev Ph.D., Jonathan Race Ph.D., Alexandra Rivas)

as part of the

Professional Master of Science and Technology Professional Experience Project

Sponsored by:

McGee’s Christmas Tree Farm

Submitted By:

Spencer Smith (u0749201)

Date

TBD

# Executive Summary

This project created a Farm Management Information System (FMIS) for McGee’s Christmas Tree Farm in Placerville, California. The FMIS was built using the Python programming language. Data was collected by taking digital photographs at points of interest around the farm. At each point of interest, a QR code was included in the picture so that the locations can be categorized by the description of the QR code. The pictures were then processed to extract the GPS coordinates from the meta data and read the QR codes. This data was overlayed on an aerial map of the farm to produce a visualization of the distribution of points of interest.

The purpose of the FMIS is to help the farm manager understand the work required on the farm. The descriptions for the points of interest used during data collection were: Dead Tree, 2023 Sapling, Pest Issue, and Open Space. The data collection occurred in June 2023 with the other work for the project also occurring in Summer of 2023. The project was completed without cost for McGee’s Christmas Tree Farm.

Python was chosen for this project to minimize the labor required for software development. The packages used for this project include pandas, matplotlib, plotly, exif, os, re, datetime, tkinter, and cv2. The python file was converted to an executable file so the sponsor does not have to download and install python. A Standard Operating Procedure was created for the sponsor along with training slides and a training video.

# Project Description

## Introduction

Harvesting a Christmas tree is an important winter tradition for many American families. The harvest and decoration of an evergreen tree near the winter solstice is a tradition that dates long before the founding of the United States of America or even the Christian Religion. The practice may be over 6000 years old1.

The two categories of Christmas trees are natural and artificial. There are three main ways to acquire a Natural Christmas tree: harvest from public or owned land, purchase a pre-cut tree, or purchase of a Choose and Cut tree. In 2017 there were 295,162 Acres of cultivated Christmas trees across 10,095 farms, with sales exceeding $375 million2.

McGee’s Christmas Tree Farm sells pre-cut and Choose and Cut trees. The Choose and Cut trees are sourced from approximately 12 acres of cultivated trees. The property also contains a building for processing transactions and loading trees onto vehicles. Pre-cut trees are sourced from other lots and transported to the main lot for sale. The farm is open for 1 or 2 weekends per year, typically starting the weekend after the Thanksgiving holiday in November.

Choose and cut Christmas tree farming presents challenges not experienced in many types of agriculture. In most farming, there is an ideal size at which each variety of plant should be harvested. This uniformity allows farmers to estimate when and where harvesting will be required. Typical farming also involves harvesting the plants in an organized fashion, with entire fields or parts of fields harvested at one time. In choose and cut Christmas free farming, each customer has an idea of the ideal tree size, often depending on the ceiling height or available floor space at their residence. Choose and cut also results in sporadic harvesting because the customers can walk the farm to choose their ideal tree. To solve the issue of knowing where work is required on the farm, this project aims to create a task management and visualization system for the McGee Christmas Tree Farm.

The McGee’s Christmas Tree Farm is located in Placerville, California. The farm is about 600 miles from the University of Utah in the foothills of the Sierra mountains. The farm is located in an area locally known as apple hill. There are many orchards, vineyards, and Christmas tree farms in the area. The area is a popular destination for tourism in the fall for wine tasting, eating as restaurants, buying baked goods, and choosing a Christmas tree.

During this project I traveled to the farm from Utah to do the data collection. I spent 2 weeks there because I wanted to be sure I had enough time. While I was there, I was able to walk the 12-acre farm and collect the data with Eli McGee, the farm manager. This is a beautiful area and I would recommend anyone visit to experience the evergreen forests.

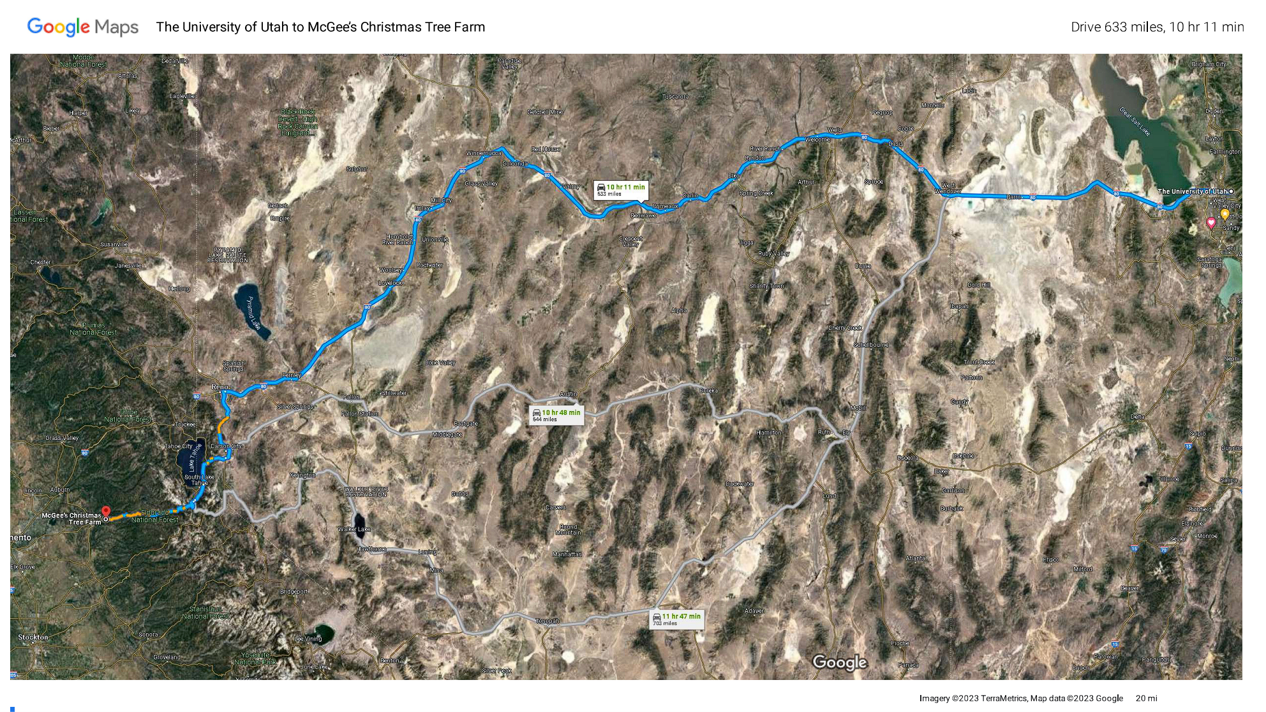


Figure X

## Project Objectives

STEM Objective 1: Collect and store the data for the type and location of work required on McGee’s Christmas Tree Farm. This objective is to be completed by June 15, 2023.

STEM Objective 2: Create a user interface to visualize and update the task data. This objective is to be completed by July 31, 2023.

Non-STEM Objective 1: Create a Standard Operating Procedure to document how to use and upkeep the Farm Management Information System. This objective is to be completed by August 15, 2023.

Non-STEM Objective 2: Train the Sponsor on how to use the Farm Management Information System using the Standard Operating Procedure. This objective is to be completed by August 31, 2023.

There were no changes to the project objectives during the project. There were changes to the methods as the specific challenges of the project presented themselves. Originally, the tasks were intended to be sorted by the color of each picture. After testing, this turned out to be unreliable. The colors would vary drastically depending on the lighting and the colors of nearby objects. To resolve this issue, I pivoted the project to using barcodes. The barcodes were more reliable but they are difficult to use and they got very long if the task description was long. I figured out a way to generate the barcodes in excel but it required complicated VBA macros. While the barcodes were much better than my initial method, they were not satisfactory for this project. The last option I tried was QR codes. This ended up being the best option. The QR can be easily created using an excel add-on which eliminated the need for complicated python or VBA code to generate the codes. Smart phone cameras also provide a preview of the QR code description while taking the picture which is a good indicator that the QR code will be readable in post processing.

Another difficult aspect of this project that was not discussed in the proposal was mapping the coordinates over a map of the farm. I has assumed this part of the project would be simple but it turned out to be difficult, mostly due to the way I approached the problem at first. At first, I tried converting the coordinates to pixel locations based on the coordinates on each corner of the map and the pixel size of the map. This method provided inaccurate results and after many attempts to get it to work I searched for a new method. I ended up finding mpimg from the matplotlib library. This module allowed me to use the arial photo as a map my redefining the edges of the map using its coordinates. Then I was able to map the task data on the map because they were now defined using the same scale and units.

The proposal for this project discussed how an iPhone 14 Pro would be used for data collection. This iPhone has dual band GPS which increases the accuracy of the GPS data. Before data collection we took a sample of data at one corner of the farm. We collected the same data using an iPhone 14 Pro and an iPhone 13. The data was then mapped so we could compare the data. We only had access to one iPhone 14 Pro. If we could only use the iPhone 14 Pro to collect data, then we would not be able to have 2 people collecting data. After comparing the maps of the sampled data, we determined that the increased accuracy of the iPhone 14 Pro was not worth the increased time required to collect the data. About half of the data was collected using an iPhone 14 Pro and the other half with an iPhone 13.

This project was written in the Python programming language. Python was chosen because of my prior experience with it, and the extensive libraries available for use. Any other programming language for this project would have taken prohibitively long to write the software. Python is notoriously slow at processing data but this project values speed of software development over performance of the software. Processing all 2200 pictures takes about a half hour. While this is not ideal, it is simple enough to allow the program to run while working on something else.

## Methods

**STEM Objective 1**

This was most difficult from a methods perspective. Three methods were considered for collecting the location data for the tasks. The first method considered was directly numbering trees. If executed correctly, this method could provide an extremely accurate location, but setting up and using the numbering system would be very time-consuming. It might require physical numbering, which would be prohibitively expensive. The second method considered was using the Label feature within Google Maps. This method would require pulling the location data from Google Maps and using the Google Maps Application Programming Interface. Between the risk of changes in Google Maps and the difficulty of using Labels, this method was determined to be not viable.

The third method, which is the selected method, consists of taking pictures using a smartphone at the location of each task. There could be thousands of pictures required for data collection, so the sponsor has been instructed to allow 50 Gb of storage for the data collection. During data collection of this project there were 2200 photos that required 5.5 Gb of storage. The pictures were uploaded to a Windows PC for processing. The software reads the picture’s Exchangeable Image File Format (EXIF) data to obtain the GPS coordinates. This method was shown to be viable in an article published by the Journal of Physics: Conference Series3. This project was written in the Python programming language.

#Convert to decimal coodinates

def decimal\_coords(coords, ref):

    decimal\_degrees = coords[0] + coords[1] / 60 + coords[2] / 3600

    if ref == 'S' or ref == 'W':

        decimal\_degrees = -decimal\_degrees

    return decimal\_degrees

#extracts and prints GPS coordinates

def img\_coords(f\_name):

    with open(f\_name, 'rb') as src:

        img = Image(src)

        print(src.name, img)

    if img.has\_exif:

        try:

            img.gps\_longitude

            return [decimal\_coords(img.gps\_latitude, img.gps\_latitude\_ref),\

                    decimal\_coords(img.gps\_longitude, img.gps\_longitude\_ref)\

                        , img.datetime\_original]

        except Exception as e:

            return ['error during exif read', e, None]

    else:

        return ['error', 'no exif', None]

Figure X GPS Extract functions

Figure X shows the functions that are responsible for getting the GPS coordinates from the images. These functions are based on an example provided by medium.com5. Besides getting the coordinates. These functions also provide the user with useful error messages if the pictures do not contain EXIF data for if there is an issue with the data. During development we discovered a situation where the file can contain EXIF but missing GPS data. If the pictures are sent over email or text using an iPhone to a person you have not authorized location sharing with, then the iPhone will strip out the GPS coordinates while leaving the rest of the EXIF data.

This was difficult to figure out because the iPhone gives no warning that it is altering the data from the picture. The GPS coordinates in the EXIF data are stored in a Degrees, Minutes, Seconds format. To make the data easier to work with, these coordinates are converted to decimal degrees using the decimal\_coords function. The Image function from the exif package is what allows EXIF data to be read4. The img\_coords function returns the coordinates or errors as a list to be used in later steps

Different types of tasks can be required on the farm. To distinguish between each task, each picture contains a QR code with the description of the task. These QR codes are prepared prior to data collection. During data collection in June, we chose 2023 Sapling, Dead Tree, Open Space, and Pest Issue as the task descriptions. The QR codes were printed out so we could take pictures of the QR codes while walking the farm. The QR codes need to be in full light or full shade when taking the picture or the program will not be able to read the QR code. There can also only be one QR code in the frame at once because the program will reject the input if it detects more than one QR code. These errors can be fixed in post processing, but it is easier to take a proper picture the first time.

def barcode\_error\_fixing(picture\_path,error\_code):

    if '(' in os.path.basename(picture\_path):

        name = os.path.basename(picture\_path)

        return name[name.find('(')+1:name.find(')')]

    else:

        return error\_code

def read\_barcode(picture\_path):

    im = cv2.imread(picture\_path, cv2.IMREAD\_GRAYSCALE)

    blur = cv2.GaussianBlur(im, (5, 5), 0)

    ret, bw\_im = cv2.threshold(blur, 120, 255,\

                                cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)

    barcode\_info = decode(bw\_im)#, symbols=[ZBarSymbol.CODE128])

    if barcode\_info == []:

        return barcode\_error\_fixing(picture\_path,"error: no barcode found")

    elif len(barcode\_info) > 1:

        return barcode\_error\_fixing(picture\_path,"error: multiple barcodes")

    else:

        return barcode\_info[0].data.decode("utf-8")

Figure X: QR Code Functions

Figure X contains the functions that read the barcodes. The decode function from the pyzbar package is used to read the QR code. While testing this function I learned that the decode function can perfectly read screenshots of QR codes, but could not read pictures of QR codes printed on paper. After reading multiple Stack Overflow posts I learned that pictures of QR codes need image processing so they can be read properly. Since the CV2 package was already being used for loading the images, I used the image processing functions that are available within CV2. The details behind each function are not particularly important for this project but can be reviewed in detail in the CV2 documentation and other image processing literature. This particular combination of image processing produces acceptable results, but could be improved because it struggles to read the QR codes when they are partially covered in shadow.

The read\_barcode function returns the description of the QR code as a string. If there is an error reading the QR code or if there are multiple QR codes in the picture, an error message is returned as a string. If there is an error reading the barcode, the barcode\_error\_fixing function is called. This function overrides anything read from the image and searches the name of the image file for a replacement. It searches for anything in parentheses and returns whatever it finds. If there are no parentheses in the name of the file it will return a string that describes the issue.

class task:

    def \_\_init\_\_(self, file\_path, task\_type, latitude,\

                  longitude, date\_picture\_taken, date\_picture\_processed):

        self.file\_path = file\_path

        self.task\_type = task\_type

        self.latitude = latitude

        self.longitude = longitude

        self.date\_picture\_taken = date\_picture\_taken

        self.date\_picture\_processed = date\_picture\_processed

    def valid(self):

        #check for error message or NoneType

        e = 'error'

        if self.latitude is not None and self.task\_type is not None:

            if e in str(self.latitude) or e in str(self.task\_type):

                return False

            else:

                return True

        else:

            return False

Figure X: task class

To check for error messages in the GPS or task data I created a class that contains all the data that will later be saved. It was not necessary to create a class for this purpose, but when the class was written I wasn’t sure if it would be useful for other purposes. It could be useful for future improvements

def new\_data():

    columns\_upload = ['Picture File Path', 'Task Type', 'Latitude',\

                       'Longitude', 'Date Picture Taken',\

                          'Date Picture Processed']

    columns\_errors = \

        ['Picture File Path', 'Barcode', 'exif', 'exif\_detail']

    picture\_df = pd.DataFrame(columns = columns\_upload)

    errors\_df = pd.DataFrame(columns = columns\_errors)

    root = input\_data\_path

    image\_list = os.listdir(root)

    image\_list = \

        [root + '\\' + a  for a in image\_list if a.upper().endswith('JPG')]

    #print(image\_list)

    now = datetime.now().timestamp()

    for a in image\_list:

        barcode\_string = read\_barcode(a)

        exif\_list = img\_coords(a)

        current\_pic = task(a,barcode\_string, exif\_list[0],\

                            exif\_list[1], exif\_list[2], now)

        if current\_pic.valid():

            picture\_df.loc[picture\_df.shape[0]] = \

                [current\_pic.file\_path, current\_pic.task\_type,\

                  current\_pic.latitude, current\_pic.longitude, \

                    current\_pic.date\_picture\_taken, now]

        else:

            errors\_df.loc[errors\_df.shape[0]] =\

                  [current\_pic.file\_path, current\_pic.task\_type,\

                    current\_pic.latitude, current\_pic.longitude]

        del current\_pic

    folder = data\_save\_location

    picture\_df.to\_csv(folder + 'picturedata' + str(now) + '.csv' )

    errors\_df.to\_csv(folder + 'errors' + str(now) + '.csv')

Figure X: new\_data function

The new\_data function combines all the previous functions and saves the data into 2 separate DataFrames. DataFrames are a commonly used data structure from the pandas package. If any errors are detected in the data, it is saved in a separate DataFrame to make the errors easier to fix and to allow the map to populate. The DataFrames are saved as csv files with a timestamp in the name of the file that is used later to determine the newest file

def load\_newest\_file(path):

    full\_names = []

    file\_times = []

    for filename in os.listdir(path):

        full\_name = os.path.join(path,filename)

        if filename.endswith(".csv") and "picturedata"\

              in filename and os.path.isfile(full\_name):

            ts\_list = re.findall(r'\d+',filename)

            ts = ts\_list[0] + "." + ts\_list[1]

            file\_time = float(ts)

            full\_names.append(full\_name)

            file\_times.append(file\_time)

    current\_file = full\_names[file\_times.index(max(file\_times))]

    current\_df = pd.read\_csv(current\_file,index\_col=0)

    return current\_df

Figure X: load\_newest\_file function

The load\_newest\_file function takes the previously saved csv file and converts it back into a dataframe. This is necessary because the it allows the map to be generated from data that was saved in a previous instance of this program. If this function did not exist, the pictures would have to be processed every time the map was generated. This would be undesirable because processing the pictures is the most time intensive part of the process. The function uses the regular expression package to read only the numbers from the filename. There are 2 numbers in the file name because a timestamp is a decimal number. The period in the number is not considered a number by the regular expression, so 2 numbers are returned by it. These 2 numbers are then turned into a float data type so they can be properly ordered to find the largest timestamp. Once the file with the largest timestamp is found it is loaded into a DataFrame

Figure X: load\_data\_func

def load\_data\_func():

    df = load\_newest\_file(data\_save\_location)

    display\_data(df.values.tolist(),task\_subset)

    create\_bar\_chart(df)

The load\_data\_func function is a simple connector function that loads the data then calls functions to generate the visualizations. This function has no input because it is later called by the user interface.

def display\_data(data\_list, subset):

    temp\_list = []

    if len(subset) != 0:

        for i in data\_list:

            if i[1] in subset:

                temp\_list.append(i)

        data\_list = temp\_list

    fig, ax = plt.subplots()

    def add\_scatter(x,y, color):

        ax.scatter(x, y, edgecolor='yellow', zorder=2,)

    descriptions = []

    x = []

    y = []

    for i in data\_list:

        x.append(i[3])

        y.append(i[2])

        descriptions.append(i[1])

    unique\_descriptions = []

    for i in descriptions:

        if i not in unique\_descriptions:

            unique\_descriptions.append(i)

    colors = []

    for i in range(len(unique\_descriptions)):

        colors.append(i/len(unique\_descriptions))

Figure X: display\_data part 1

The display\_data function is too long to be displayed on one page of this report so it will be split into 2 figures. The first part of this function removes any unwanted data from the dataset. For example, if the user only wants to display only the dead trees on the map so they can work on removing them, they simply enter Dead Trees into the subset. Then the x and y coordinates and task descriptions a separated from the input list. With the list of descriptions, all of the unique values are found so they can later be plotted separately. The colors list is not necessary anymore. The display\_data function has much room for improvement because it contains many features with are unused. These features have not been removed out of caution against premature optimization.

Figure X: display\_data part 2

    for i in range(len(colors)):

        current\_x =[]

        current\_y =[]

        current\_color = []

        for j in range(len(descriptions)):

            if unique\_descriptions[i] == descriptions[j]:

                current\_x.append(x[j])

                current\_y.append(y[j])

                current\_color.append(colors[i])

        add\_scatter(current\_x, current\_y, current\_color)

    #ax.scatter(x, y, c=colors, edgecolor = 'red', zorder=2,)

    ax.imshow(mpimg.imread(aerial\_photo\_path), \

              extent=(-120.717566420228, -120.71407,\

                       38.7385559849825, 38.74139), zorder=1)

    plt.legend(unique\_descriptions)

    save\_file\_name = data\_save\_location +\

          'map' +str(datetime.now().timestamp())+ '.png'

    plt.savefig(save\_file\_name,dpi = 800)

    plt.show()

One feature that we decided to not implement that was discussed during the proposal was a method to remove tasks from the database. Some of the code to was written to do this but it was not fully implemented. It was not a necessary feature to get a minimum viable product so it was removed to speed up the project. The current implementation is designed to be used once per year during which the entire database will be rewritten.

The task data must be stored so the program can be run multiple times without reprocessing the images. This is a small project with limited resources so a simple database will suffice. The data will be stored in a .csv file and backed up on a cloud storage service such as Dropbox, OneDrive, or Google Drive. The images can be stored on the local Windows PC until the data is processed and saved to cloud storage. After the data is saved, the images can be deleted.

**STEM Objective 2**

This objective will be accomplished by creating a user interface with the tkinter package in Python7. An aerial photograph of the farm will be acquired from an open source such as EarthExplorer (see Figure 2)8. The task data will then be color coded and overlayed onto the aerial photograph. The GPS coordinates will be converted to pixel locations on the aerial photograph so they are placed correctly.

**Non-STEM Objective 1**

The Standard Operating Procedure (SOP) will be the documentation for the software. It will detail how to gather and, upload the data, generate the data visualization, and interpret it. The SOP will be created in Microsoft Office and be delivered in .docx and .pdf formats.

**Non-STEM Objective 2**

The sponsor will be trained on the software using a PowerPoint presentation along with a demonstration of the software. The PowerPoint presentation will be a summary of the SOP with references to the corresponding sections of the SOP. The presentation to the Sponsor will occur over a Zoom call. During the initial on-site data collection visit, the Sponsor will be trained on the data collection and upload process in person. Training the Sponsor during data collection should prevent the need for a second on-site visit, but a second visit could be done if needed.



Figure 2: Example EarthExplorer screenshot

## Expected Outcomes

At the end of the project, the Farm Management Information System (FMIS) and supporting documentation will be delivered to the Sponsor. The Sponsor will also be trained on the SOP, so they understand how to use it in their daily operations. The FMIS will have the capability to store and display task data relevant to the operation of the farm. The data will be displayed in a way that allows the user to immediately know which areas need the most work and where their time would be best spent.

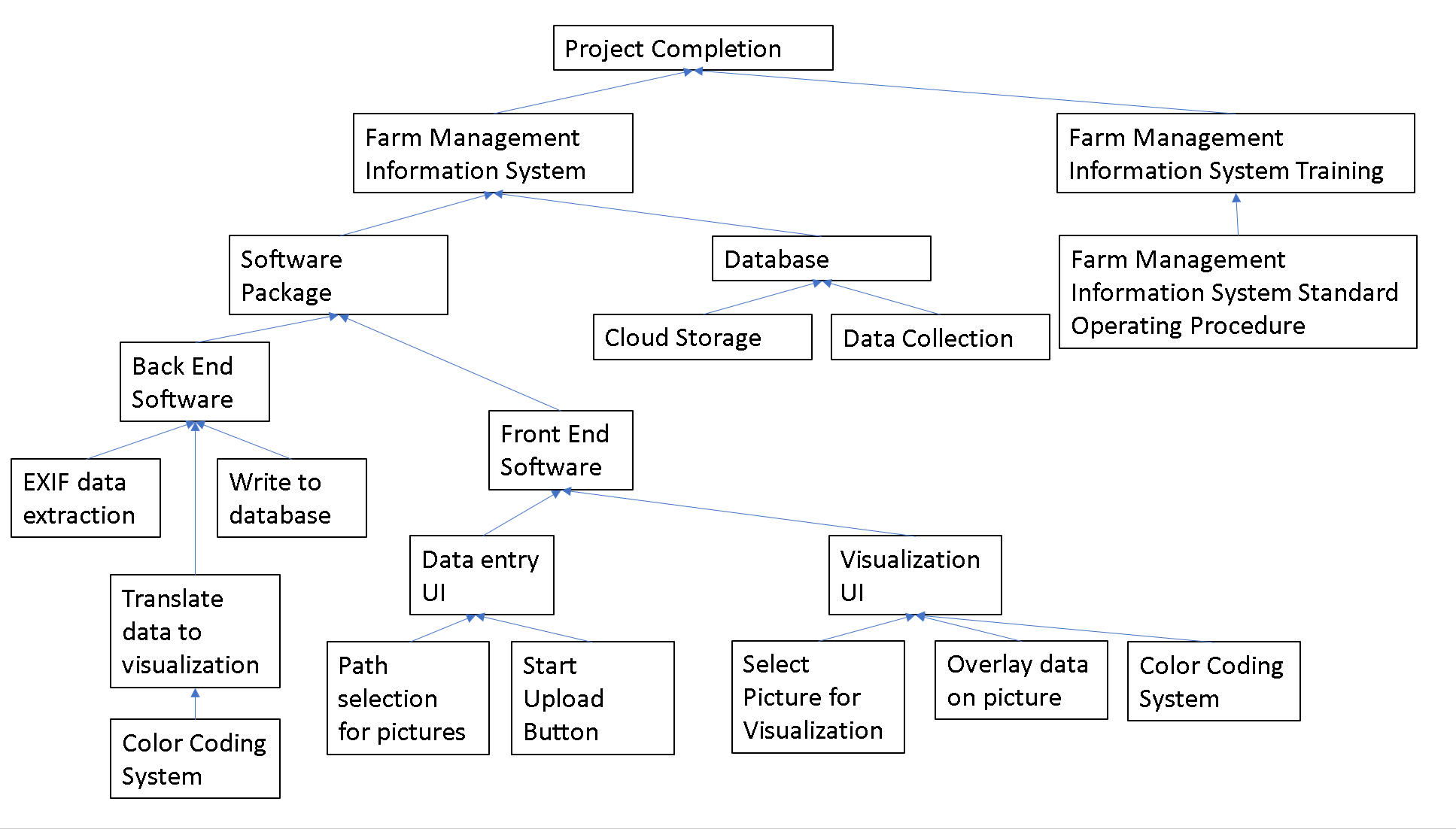


Figure 3: Project WBS

## Resource Requirements

This project will be completed with open-source software. Python and associated packages can be obtained from python.org. The part of this project that will require ongoing funding from the Sponsor is a cloud storage service which varies in cost based on the features provided and the storage required. This part could potentially be cut from the project if needed, but relying on only local storage is risky. The Sponsor will assist with the data collection through the iPhone 14 Pro that will be used for data collection. At the end of the project, the Sponsor will provide a Windows PC to install the software on. The rest of the project will solely be completed by Spencer Smith.

## Timeline

The project is split into one-month sprints based on an agile framework. All dates are for the year 2023.

May: Prepare for data collection. All software related to the extraction and storage of data will be completed. The color-coding system will be finalized. The software will be tested with fabricated data

June: The data from the farm will be collected and stored in the database

July: The user Interface and Visualization software will be created and tested.

August: The Standard Operating Procedure will be drafted, and the Sponsor will be trained on the documentation.

September: Final report written and presented to the Supervisory Committee.

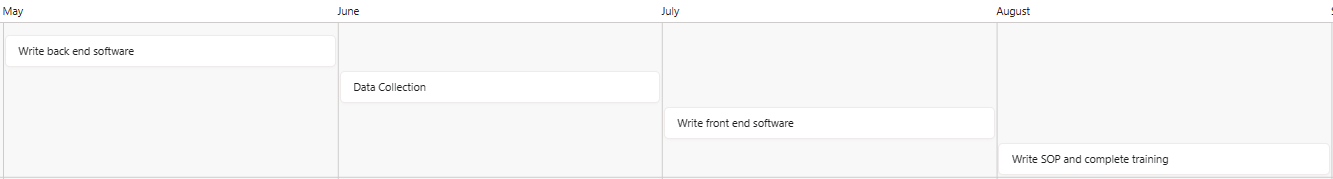


Figure 4: Project Schedule (all dates in 2023)

# Sponsor Information

McGee’s Christmas Tree Farm in Placerville, CA is the end customer for this project. Spencer Smith is not employed by the McGee’s Christmas Tree Farm. The Sponsor’s contact information is as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Phone | Email | Address |  |
| Eli McGee | (530) 644-4731 | michaelelimcgee@gmail.com | 3131 Carson Rd, Placerville , CA 95667 |  |

# References

1 Karas, Sheryl. 1998. *The Solstice Evergreen The History, Folklore and Origins of the Christmas Tree*. Revised and Expanded Edition. Aslan Publishing. <https://books.google.com/books?hl=en&lr=&id=-iwNCdt8kUMC&oi=fnd&pg=PA1&dq=christmas+tree+history&ots=PR2mBhCtfL&sig=alHdsGR9_gc5N4rbNIvEGVaMqYQ#v=onepage&q&f=false>.

2 Perdue, Sonny, and Hubert Hamer. 2019. “2017 Census of Agriculture.” <https://www.nass.usda.gov/Publications/AgCensus/2017/Full_Report/Volume_1,_Chapter_1_US/usv1.pdf>.

3 Prasetiyo, B, and al et. 2021. “Automatic Geotagging Using GPS EXIF Metadata of Smartphone Digital Photos in Tree Planting Location Mapping.” In . Vol. 1918. Journal of Physics: Conference Series. <https://doi.org/10.1088/1742-6596/1918/4/042001>.

4 Thieding, Tyler. n.d. “Exif 1.6.0.” Python. Accessed March 12, 2023. <https://pypi.org/project/exif/>.

5 Abdishakur. 2021. “How to Extract GPS Coordinates from Images in Python.” July 29, 2021. <https://medium.com/spatial-data-science/how-to-extract-gps-coordinates-from-images-in-python-e66e542af354#:~:text=The%20first%20function%20converts%20the%20coordinates%20into%20Decimal,if%20ref%20%3D%3D%20%E2%80%9CS%E2%80%9D%20or%20ref%20%3D%3D%20%E2%80%9CW%E2%80%9D%3A>.

6 Apple Inc. n.d. “IPhone 14 Pro Specifications.” Accessed March 12, 2023. <https://www.apple.com/iphone-14-pro/specs/>.

7 Python Software Foundation. n.d. “Tkinter — Python Interface to Tcl/Tk.” Accessed March 12, 2023. <https://docs.python.org/3/library/tkinter.html>.

8 US Department pf the Interior. n.d. “EarthExplorer.” Accessed March 12, 2023. <https://earthexplorer.usgs.gov/>.