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. A Standard Operating Procedure was created for the sponsor along with training slides and a training video.

The resulting map provides some insights into the status of the farm. One block of trees had almost no new trees planted in 2023 due to muddy conditions. This will be useful information next year during planting season. Another insight is how widespread gall is on the south western side of the farm. This area should be carefully watched and tended to to prevent further spread of the fungal disease.

This project was successful in accomplishing its goals. While far from being an ideal solution, the result of the project was a useable tool that provides insights into the operations of the farm. These insights can be used by the farm manager to make more informed decisions.

# 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 6,000 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. Pre-cut trees are sourced from other lots and transported to the main lot for sale. The property also contains a building for processing transactions and loading trees onto vehicles. 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 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.

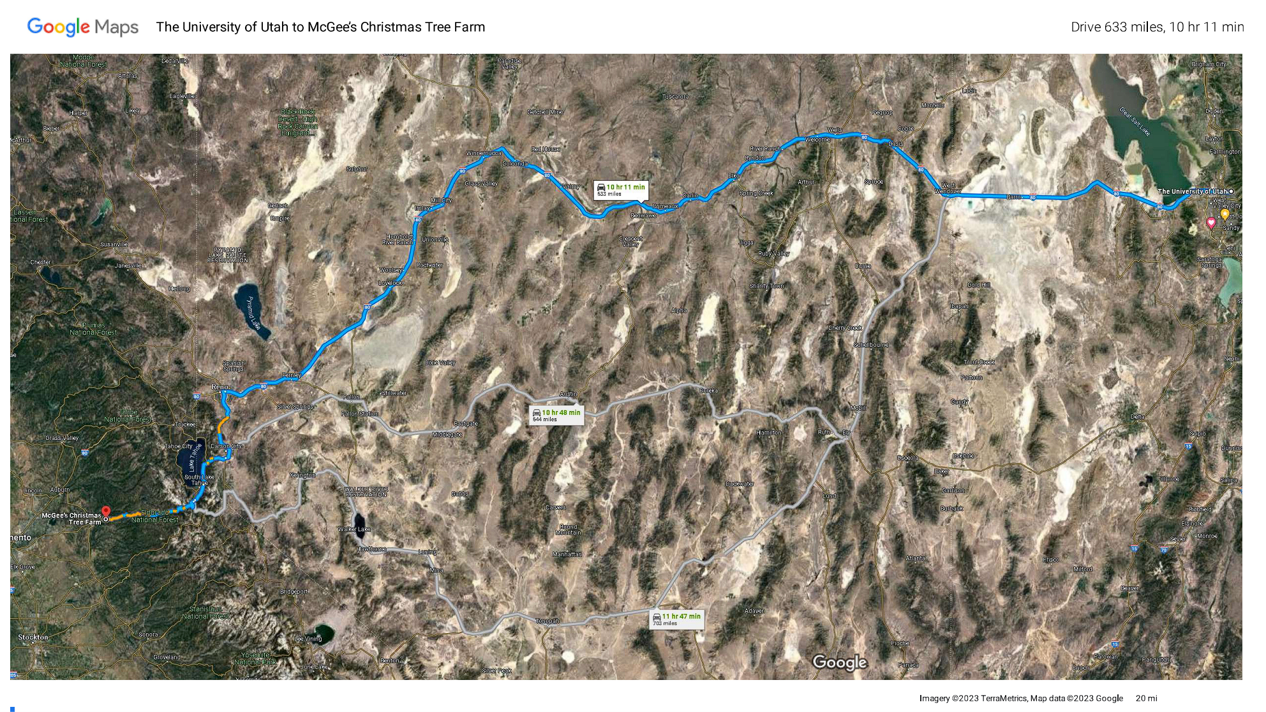


Figure 1: Google Maps, University of Utah to McGee’s Christmas Tree Farm

## 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 Visual Basic for Applications (VBA) code. 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 had 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 library3. This module allowed me to use the arial photo as a map by 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 data4. Before data collection Eli and I 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.

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 was written to do this but it was not fully implemented. It was not a necessary feature to get a minimum viable product5 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.

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 objective 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 Series6. This project was written in the Python programming language. The rest of this section consists of a code snippet followed by an explanation of its function to explain the method

#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 2: GPS Extract functions

Figure 2 shows the functions that are responsible for getting the GPS coordinates from the images. These functions are based on an example provided by medium.com7. Besides getting the coordinates, these functions also provide the user with useful error messages if the pictures do not contain EXIF data or 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 read8. 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 2023 Sapling task was used to document the location of all the trees planted during the 2023 season. Dead Tree was used to document all the locations of dead trees. Eli wanted to find the location of all dead trees so he could come back and remove them later to prevent the spread of pests and disease. The Open Space task was used to document all of the locations that could have a tree planted next season. Pest Issue was used to document any type of pest or disease including aphids, gall, or poison oak.

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 3: QR Code Functions

Figure 3 contains the functions that read the QR codes. The decode function from the pyzbar package is used to read the QR code9. 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 CV210. 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 QR code, 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 4: 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 4: 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 package11. 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 5: 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 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 6: 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 7: 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 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 optimization12.

Figure 8: 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()

The second part of the display\_data function groups each task description and adds a scatter plot for each unique description. After all the scatter plots are created, the aerial photo is added to the plot with the coordinates of the corners defined. The aerial photo was acquired from Google Maps via a screenshot. A legend is put on the plot using the list of unique descriptions. Now that the plot is created it is saved as a .png file and then displayed for the user.

def create\_bar\_chart(df):

    fig = px.bar(df.groupby('Task Type').count().reset\_index(level = 0),\

                  x= 'Task Type', y = 'Latitude')

    fig.update\_layout(barmode='stack',\

                       xaxis={'categoryorder':'total descending'})

    fig.show()

Figure 9: create\_bar\_chart Function

The create\_bar\_chart function uses the plotly package to generate a bar chart that show how many of each task description are in the dataset13. The groupby and count functions from pandas are used to count each task for each task description. After the bar chart is initialized, the bars are reordered so the largest bar is on the left side of the chart. Then plot is displayed which open in the default browser.

**STEM Objective 2**

This objective was accomplished by creating a user interface with the tkinter package in Python14. The user interface takes input from the user for storage locations and file names that are necessary for the program to run. The user can load the data or display the data with a button for each action. The User interface can be seen in Figure 10. The rest of this section consists of a code snippet and an explanation for its function.

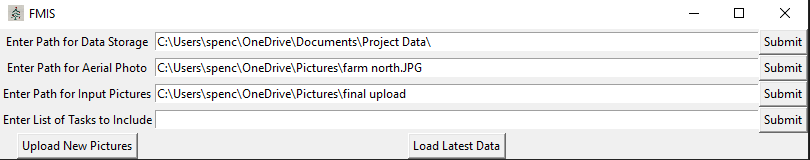


Figure 10: User Interface

root = tk.Tk()

root.iconbitmap(r"C:\Users\spenc\Christmas-Tree-FMIS\tree\_icon.ico")

root.title("FMIS")

Figure 11: tkinter Initialization

When using tkinter, the entire user interface is based on the Tk object that I named root. Every button, label, and entry field is later built upon this root object. In tkinter windows there is an icon in the top left corner of the window. If an icon is not specified then the default tkinter icon is used which look like a feather. To replace the default icon, my mother drew a picture of a Christmas tree that I turned into an icon using an image conversion website. The title that is displayed at the top of the tkinter window is “FMIS”.

save\_path\_label = tk.Label(root,text = "Enter Path for Data Storage")

save\_file\_path = tk.StringVar()

save\_file\_path.set(data\_save\_location)

aerial\_path\_label = tk.Label(root,text = "Enter Path for Aerial Photo")

aerial\_file\_path = tk.StringVar()

aerial\_file\_path.set(aerial\_photo\_path)

task\_subset\_label = tk.Label(root,text = "Enter List of Tasks to Include")

task\_subset\_input = tk.StringVar()

task\_subset\_input.set("")

input\_path\_label = tk.Label(root,text = "Enter Path for Input Pictures")

input\_file\_path = tk.StringVar()

input\_file\_path.set(input\_data\_path)

save\_path\_entry = tk.Entry(root, textvariable= save\_file\_path, width= 100)

aerial\_file\_entry = tk.Entry(root, textvariable= aerial\_file\_path, width= 100)

input\_path\_entry = tk.Entry(root, textvariable= input\_file\_path, width= 100)

task\_subset\_entry = tk.Entry(root, textvariable= task\_subset\_input, width= 100)

Figure 12: Initialize Label and Entry Objects

The user interface contains 4 Entry fields which the use can use to enter information. Each Entry field is a tk.Entry object which has 3 parameters specified each. The first parameter if just the TK object that we created earlier. The second parameter it the text variable which can change based on the user input. The last parameter is the width of the Entry object in the TK window. Each text variable must be initialized as a tk.StringVar object. Each StringVar object is then set with in initial value. A Label is assigned for each Entry object so the user know what each entry field if for.

def set\_aerial\_photo\_path():

    global aerial\_photo\_path

    aerial\_photo\_path = aerial\_file\_path.get()

def set\_task\_subset():

    global task\_subset

    task\_subset = task\_subset\_input.get().split(', ')

def set\_data\_save\_location():

    global data\_save\_location

    data\_save\_location = save\_file\_path.get()

def set\_input\_data\_path():

    global input\_data\_path

    input\_data\_path = input\_file\_path.get()

Figure 13: Functions for Each Entry Field

There is a function for each entry field. These functions take the input into the fields and save them in global variables to be used elsewhere. These functions were used because tkinter buttons cannot call functions that have input parameters. This is just one method to get around the limitation inherent in tkinter buttons. Another method would have been using a lambda function which is arguably a better way to handle the situation. A lambda function would have resulted in less lines of code and a cleaner namespace because the lambda functions are anonymous. While lambda would be a better option, rewriting and testing this section of code was not worth the effort required.

submit\_save = tk.Button(root, text = "Submit",\

                         command = set\_data\_save\_location)

submit\_aerial = tk.Button(root, text = "Submit",\

                           command = set\_aerial\_photo\_path)

submit\_input = tk.Button(root, text = "Submit",\

                          command = set\_input\_data\_path)

submit\_subset = tk.Button(root, text = "Submit",\

                           command = set\_task\_subset)

run\_input = tk.Button(root, text = "Upload New Pictures",\

                       command = new\_data)

load\_data = tk.Button(root, text = "Load Latest Data",\

                       command = load\_data\_func)

Figure 14: tkinter Buttons

There are 6 tkinter buttons used in this user interface. 4 buttons for each entry field, 1 button for loading the data from the pictures, and 1 button for generating the visualizations. The button object has a text parameter which gives a description of the button’s function. Each button has a command assigned to it that runs a previously defined function.

save\_path\_label.grid(row=0,column=0)

save\_path\_entry.grid(row=0,column=1)

submit\_save.grid(row=0,column=2)

aerial\_path\_label.grid(row=1,column=0)

aerial\_file\_entry.grid(row=1,column=1)

submit\_aerial.grid(row=1,column=2)

input\_path\_label.grid(row=2,column=0)

input\_path\_entry.grid(row=2,column=1)

submit\_input.grid(row=2,column=2)

task\_subset\_label.grid(row=3,column=0)

task\_subset\_entry.grid(row=3,column=1)

submit\_subset.grid(row=3,column=2)

run\_input.grid(row=4,column=0)

load\_data.grid(row=4,column=1)

root.mainloop()

Figure 15: Formatting and Mainloop

The final step for the user interface is formatting the objects and displaying. The formatting is accomplished by specifying the row and column parameters. These parameters are zero indexed which is why the first object is located at row 0 and column 0. The mainloop function of tkinter simply run the program and waits for input until the window is closed.

**Non-STEM Objective 1**

A Standard Operating Procedure (SOP) was created for the FMIS. The FMIS was written as step-by-step instructions that follow a flow chart. There are 10 steps in the flow chart with each step having instruction of how to complete the step. The document includes screenshots to help the user understand the steps. The instructions were written with enough detail so that new employees can be trained using the document. The is provided in the appendix as item 1

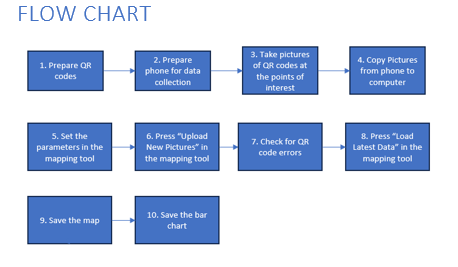


Figure 16: Process Flow Chart

**Non-STEM Objective 2**

A PowerPoint Presentation was created to assist with training if necessary. The PowerPoint is not meant to be a substitute to the SOP. It does not contain all of the details that are in the SOP. Each slide has bullet points with minimal information to give the user an overview of the process before diving into the SOP. The sponsor was also provided with a training video that was recorded using Zoom. This video steps through the training slides and provides a demonstration for each step. With these resources, the sponsor should be able to train any new employees on how to use the tool. The PowerPoint and the training video are provided in the appendix as items 2 and 3.

## Results

The result of this project is a system for collecting and visualizing data at the McGee’s Christmas Tree Farm. The map of the data shows where each data point was collected and provides a legend that describes each point. When discussing the map, we came up with a numbering scheme for the blocks of trees.

Figure 17: Block Numbering

Figure 17 shows some areas which have groups of similar issues. Blocks 3 and 5 have large areas where there is no data. These areas are mostly filled with Douglas Fir trees which are the heartiest type of tree grown on the farm. This would be a good area to direct customers to if they are looking for a Douglas Fir since there is a large area with healthy and mature trees.

Another noticeable area of the map is block 7. This block contains many open spaces. The open spaces in the block were expected because they were unable to plant trees in the area during planting season. The spring was quite wet and this part of the farm lies at the bottom of a large slope and it receives runoff from the neighboring orchard. All this extra moisture made the ground too muddy to work with when they were planting in the spring.

Blocks 8 and 10 have a large grouping of red dots for Pest Issues. This area has a significant number of trees with Gall. These round growths on the branches are cause by a fungal infection. The Gall can eventually kill the branch or tree if enough branches are infected.

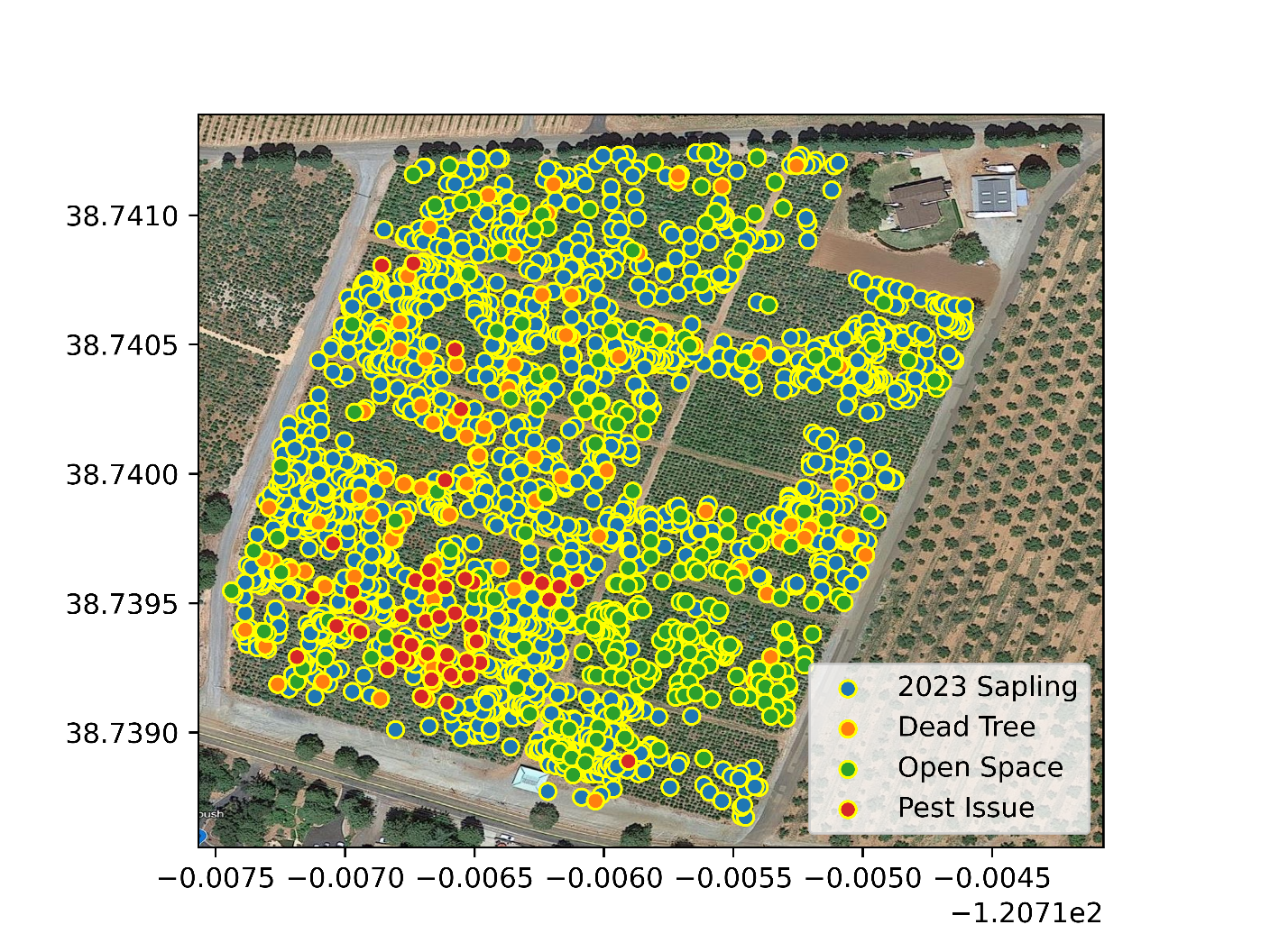


Figure 18: Mapped Data

The bar chart also has some insights. There are 258 open spaces. This provides a lower bound for how many trees should be planted next season. Since many of these open spaces are in an area that can get muddy, then planting could be completed in a break between spring storms when the ground is the driest.

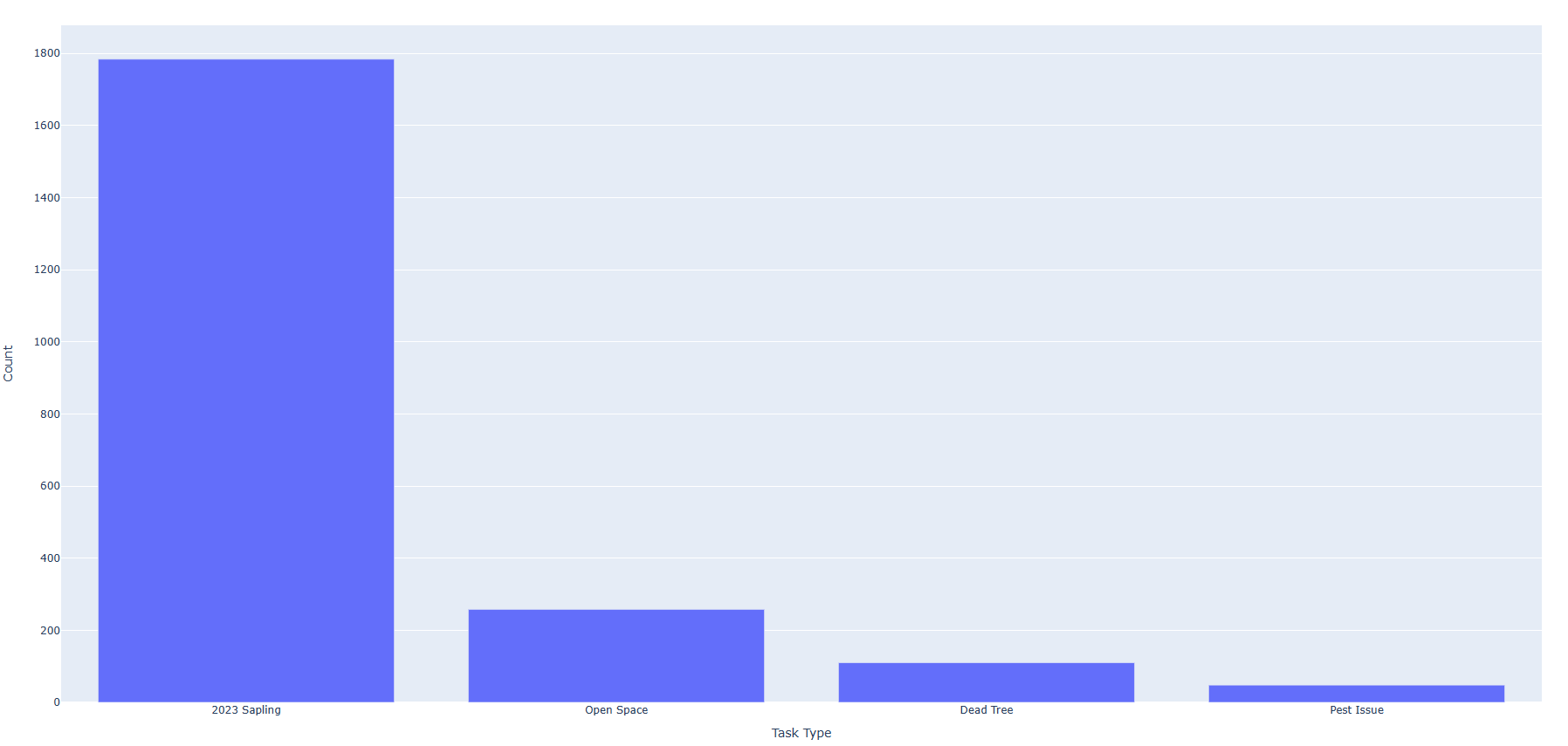


Figure 19: Bar Chart

## Next Steps

Some of the potential upgrades to the software have already been discussed. While this software works for its intended purpose there are many improvements that could be made. The improvements could range from small optimizations to complete overhauls. Whether any of these changes would be worth the effort depends on the ultimate goal for the software. If the software is meant to continue to be used by McGee’s Tree Farm and marginally improve the user experience then some optimizations could be made. If someone wanted to make this software available to other farms with the intent of starting a business, then major changes should be made.

One optimization that should be relatively simple is increasing the reliability the QE code reading. It is time consuming and tedious to fix QR code reading errors. If this could be nearly eliminated it would be a big useability improvement. Another downside of the current software is how long it takes to run. A half hour is a long time to wait for software to run. If this time could be reduced by half or more it would be a significant improvement. Each step of the process could be timed and then improvements should be focused on the longest steps in the process. Since starting this project, I have learned that when using dataframes it is better to create a list and then load the entire list into the dataframe as opposed to filling the dataframe one row at a time. This is one example of a small improvement and I’m sure there are many more that could add up to a significant performance improvement.

If someone wanted to turn this project into a business, then the existing software should only serve as a guide while completely rewriting it in other languages. Ideally, this software would be completely contained in an IOS or Android app. App development is currently not part of my skillset, so that would be a difficult task for me to complete on my own. I have no desire to take on this project but I would have no issue with someone else running with it. One issue with turning this project into a business would be convincing farms that it would be worth the cost. Small businesses are notorious for running with bare minimum technology. I personally know of 2 small business that just started dabbling with Microsoft Excel in 2023, 36 years after it was initially released. This may change as control of family businesses continues to pass to younger generations that grew up using technology from a young age.

## 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 |  |

## Conclusions

In conclusion, this project was successful because it creates useful data visualizations with no software or hardware costs to McGee’s Christmas Tree Farm. With this data they should be able to care for the farm more efficiently and keep the farm running for decades. The farm provides a valuable service to El Dorado County and those who come to visit. It has been rewarding to help a business with the skills that I have learned during my time in the Professional Master of Science and Technology program at the University of Utah.

## Appendix

Item 1 

Item 2 

Item 3 

# 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 The Matplotlib Development Team. 2023. “Matplotlib: Visualization with Python”. Zenodo. <https://doi.org/10.5281/zenodo.8347255>

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

5 Ries, Eric. The Lean Startup : How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses. 1st ed. MRSV. 2011.

6 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>.

7 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>.

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

9 Hudson, Lawrence. “Pyzbar Documentation,” March 15, 2022. <https://pypi.org/project/pyzbar>.

10 “OpenCV Documentation.” OpenCV Team, n.d. <https://pypi.org/project/opencv-python/>.

11 “Pandas Documentation.” The Pandas Project, September 20, 2023. <https://pandas.pydata.org/docs/>.

12 Donald E. Knuth. Art of Computer Programming, The: Volume 1: Fundamental Algorithms. Addison-Wesley Professional, 1997.

13 “Plotly Documentation.” Plotly, n.d. <https://plotly.com/python/>.

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