Award No. 2102120

Scientific Computing & Data Analytics*

A Comprehensive Toolkit for Research

Emmanuel Miguel Gonzalez, Jeffrey Demieville, Brenda Huppenthal,
Emily Cawley, Aditya Kumar, Bella Salter, Duke Pauli

2024-04-12

*The PhytoOracle project is supported by the following grants: U.S. Department of Energy Biological and Environmental Research (DE-SC0020401) and Advanced Research Projects Agency - Energy OPEN (DE-AR0001101); National Science Foundation Plant Genome Research Program (IOS-2102120, IOS-2023310, and IOS-1849708), Division of Biological Infrastructure (2019674 and 1743442), and CyVerse project (DBI-1743442); Cotton Incorporated (18-384, 20-720, 21-830, and 23-890); and U.S. Department of Agriculture National Institute of Food and Agriculture Specialty Crop Research Initiative (2021-51181-35903). We thank Agricultural Genome to Phenome Initiative (AG2PI) for hosting our workshop.

Table of Contents

Workshop Overview

The surge in data across different fields, largely due to technological progress, data analytics, and increased storage capacity, necessitates efficient scientific computing. These extensive data sets present opportunities for uncovering insights and knowledge. Yet, to fully harness this potential, it is essential to process the raw data using methods like information extraction, data mining, and knowledge discovery.

This workshop series aims to equip you with a comprehensive computational toolkit for your research. This toolkit will enable you to effectively manage and analyze the increasing volumes of data in your field, thereby enhancing your research outcomes. The workshop series will cover:

- Fundamental concepts of Python, Jupyter notebooks, and GitHub
- Overview of machine learning and interactive visualization
- Real-world applications in the field of plant phenotyping

We will use Google Colab notebooks, a web-based computing environment, throughout the series.

By the end of the workshop series, you'll be able to:

- Understand the fundamentals of scientific computing, including data preprocessing, statistical analysis, machine learning, and data visualization
- Incorporate a variety of scientific computing techniques into your research workflow
- Utilize established software to identify patterns within data sets
- Perform analyses with common machine learning tools, including neural networks and dimensionality reduction methods

Pre-Workshop Preparation

Google account

Create a Google account at google.com. This account will be used to access the Google Colab notebooks during the workshop

GitHub account

Create a GitHub account at github.com. This account will be used to access GitHub and create a repository.

About Presenters

Emmanuel Gonzalez is a PhD candidate in Dr. Duke Pauli's lab at the University of Arizona whose work focuses on leveraging plant phenomics, data science, and machine learning to investigate how crops respond to both abiotic and biotic stress in field conditions.

Jeffrey Demieville is an interdisciplinary R&D Engineer at the University of Arizona whose work focuses on applying biological, agricultural, and systems engineering practices to the field of phenomics, with particular emphasis on the University of Arizona Field Scanner system and its data outputs.

Brenda Huppenthal is a computer science graduate student at the University of Arizona whose work focuses on using computer vision and specifically deep learning approaches to obtain underlying structural information from point clouds.

Emily Cawley is an undergraduate Computer Science student at the University of Arizona. As an undergraduate researcher in the Pauli Lab, she works on predicting transformations of 3D data with neural networks.

Aditya Kumar is an undergraduate researcher at Dr. Duke Pauli's Lab at The University of Arizona. He specializes in software development and data visualization and is currently focusing on refining object detection models for Sorghum Panicle Detection.

Bella Salter is an undergraduate researcher at the Pauli lab whose work focuses on predicting late season lettuce growth based on early metrics of success using machine learning techniques such as long-short term memory.

Session 1: Fundamental Computational Toolkit

Session 1 Introduction

Scientific Computing & Data Analytics: A Comprehensive Toolkit for Research

Emmanuel Miguel Gonzalez, Jeffrey Demieville, Brenda Huppenthal, Emily Cawley, Aditya Kumar, Bella Salter, Duke Pauli

Award No. 2102120

Pauli Lab

About the Presenters

Emmanuel Gonzalez

Brenda Huppenthal

Jeffrey Demieville

Emily Cawley

Bella Salter

Aditya Kumar

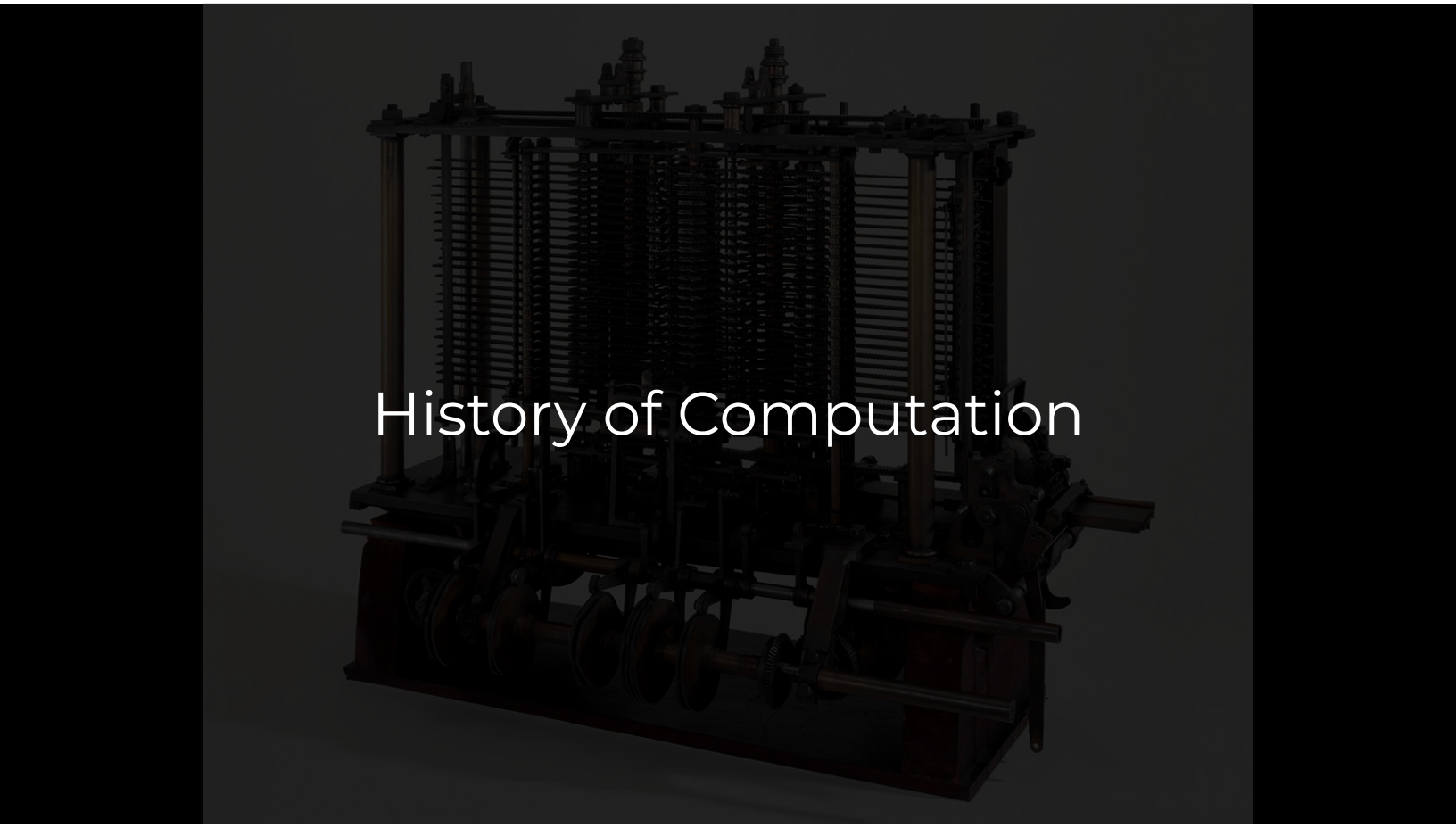
Pauli Lab

THE UNIVERSITY OF ARIZONA

PhytoOracle

About Me

- Role:
  - Plant Science PhD Candidate in Dr. Duke Pauli's lab
- Research Interests:
  - Leveraging plant phenomics to study abiotic and biotic stress responses in field conditions
  - Developing high-throughput phenotyping software
- Fun Fact:
  - My partner and I enjoy weekend road trips with our two dogs, Droopy and Manuela


History of Computation

The Dawn of Computing

Abacus

2700 B.C.

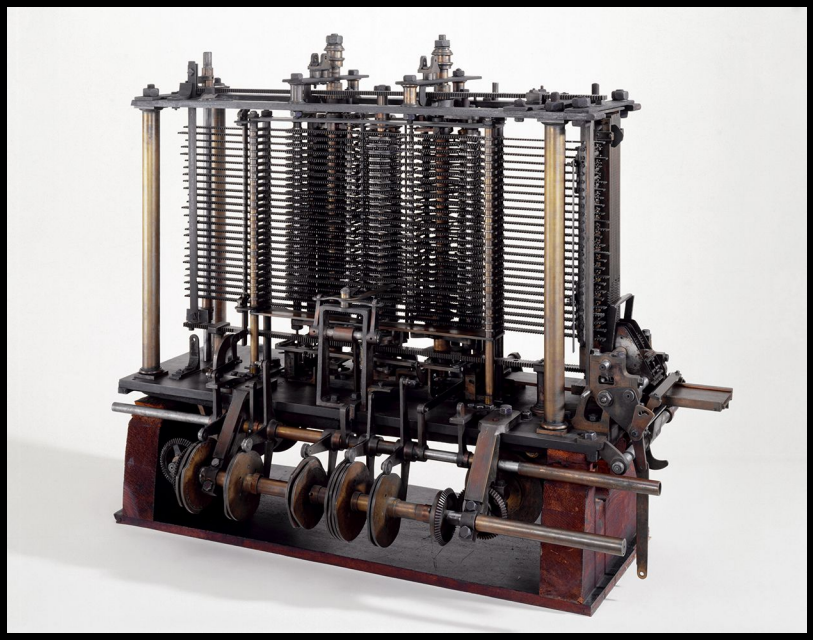
Pascaline

1640s

Arithmometer

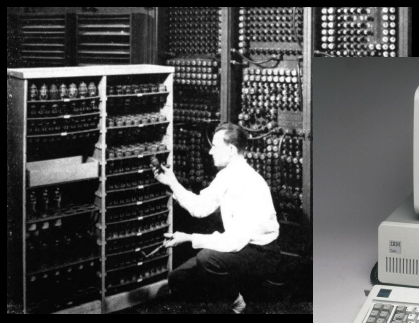
1820s

The Analytical Engine was the First Computer



Designed by Charles Babbage in the 19th century

The Evolution of Modern Computers

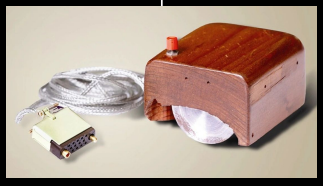


1940s: ENIAC

1980s: IBM PC

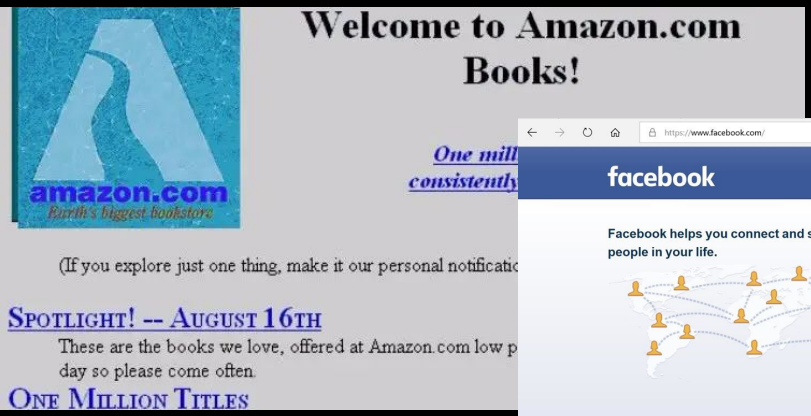
2000s: Apple iMac G3

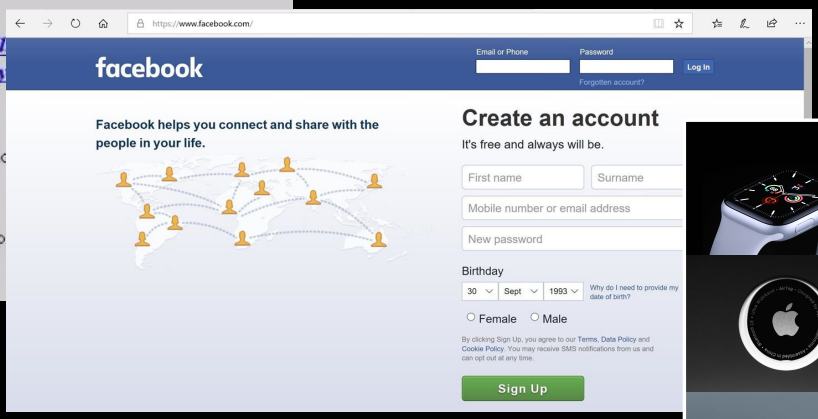
Current: Laptops



1960s

Internet Age: From Consumers to Producers of Data



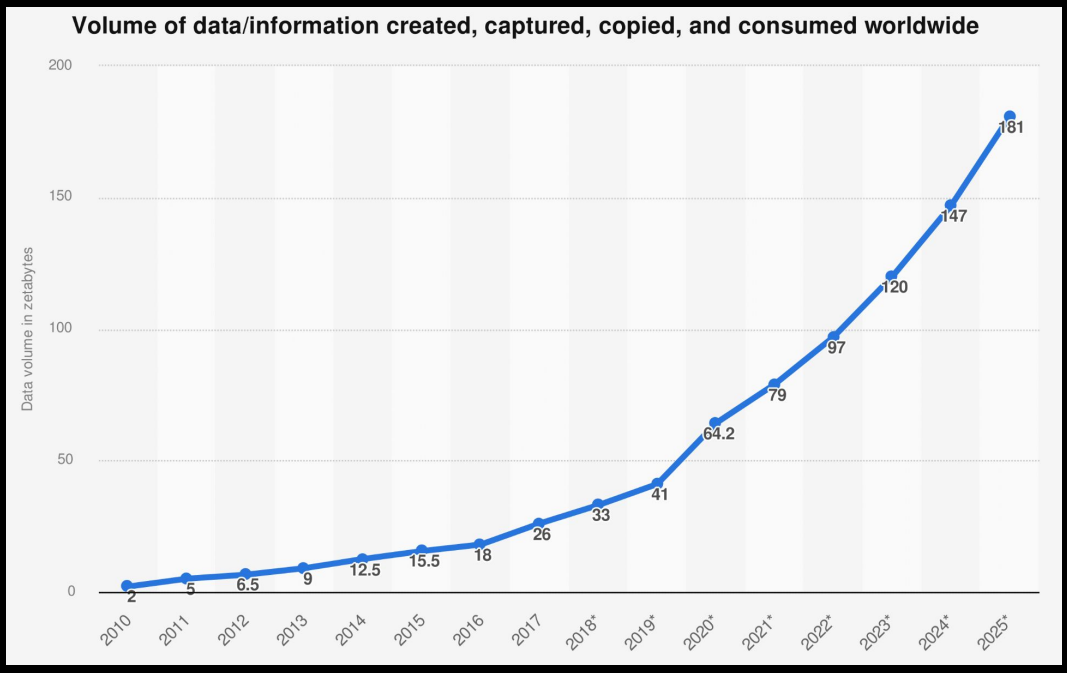


1990s:
Consumers

2000s:
Producers

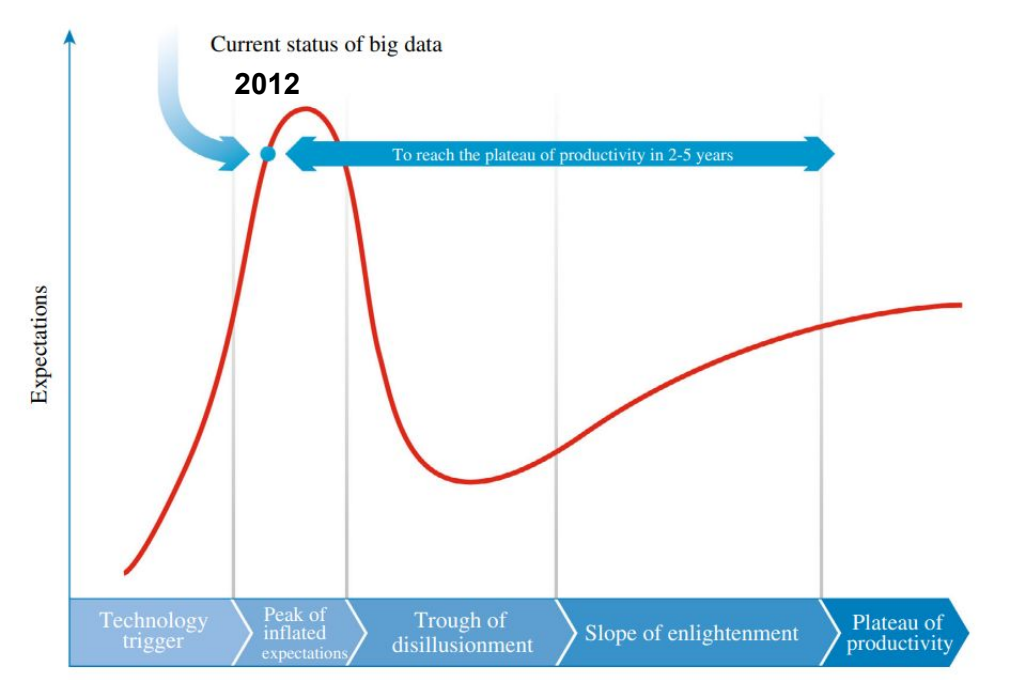
Sensors

Data Volumes Are Growing Exponentially



1 zettabyte (ZB) = 1 trillion GB

Big Data Creates Hype Cycles



Guo et al., 2014

Hype Does Not Always Materialize

CHRIS ANDERSON SCIENCE JUN 23, 2008 12:00 PM

The End of Theory: The Data Deluge Makes the Scientific Method Obsolete

Illustration: Marian Bantjes "All models are wrong, but some are useful." So proclaimed statistician George Box 30 years ago, and he was right. But what choice did we have? Only models, from cosmological equations to theories of human behavior, seemed to be able to consistently, if imperfectly, explain the world around us. Until now. Today companies [...]

Harvard Business Review

You May Not Need Big Data After All

by Jeanne W. Ross, Cynthia M. Beath, and Anne Quaadgras

From the Magazine (December 2013)

nature portfolio

ADVERTISEMENT FEATURE Advertiser retains sole responsibility for the content of this article

Advancing precision medicine using AI and big data

A data-driven approach to medical research is helping to shed new light on cancer and other heterogeneous diseases

Harvard Business Review

Use Big Data to Create Value for Customers, Not Just Target Them

by Niraj Dawar

August 16, 2016

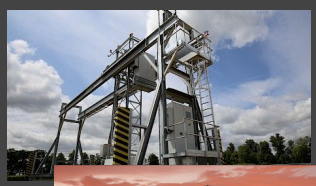
Importance of Data Analytics

Big Data Necessitates Scientific Computing

Big Data Necessitates Scientific Computing

Scientific Computing Case Study

Robots

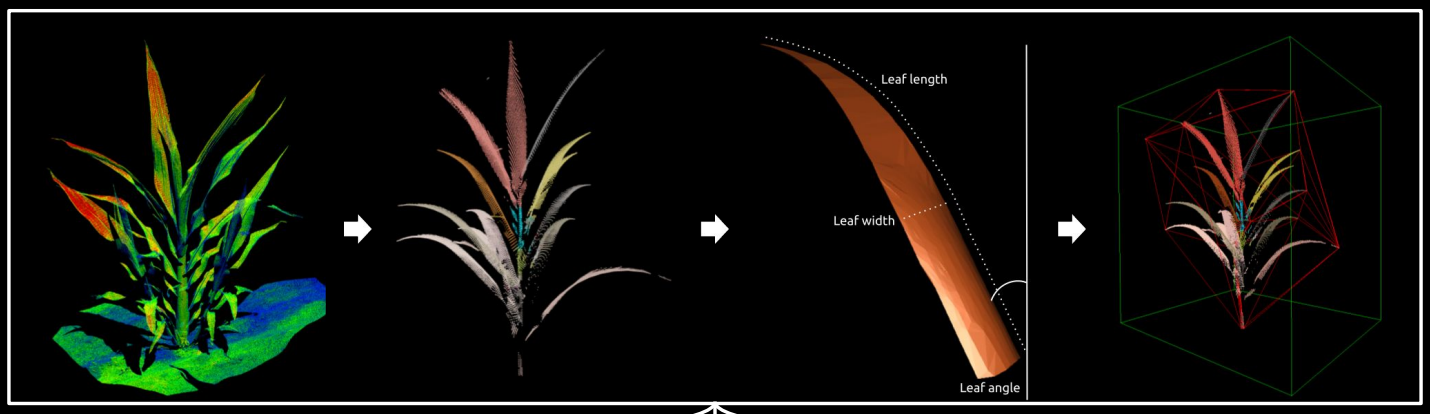


Carts

Drones

Phones

Scientific Computing Case Study



Python

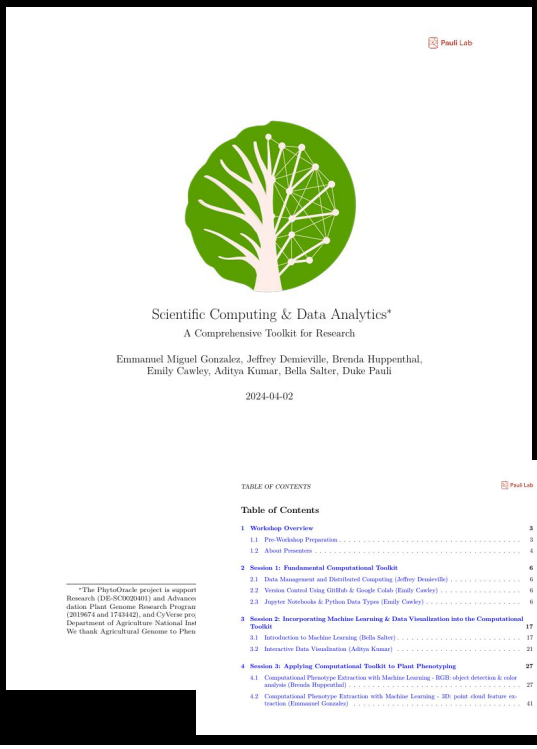
High Performance Computer

GitHub

Access to Workshop Materials

Contains workshop overview, learning objectives, and code

bit.ly/AG2PI_SciComp



Pauli Lab

Scientific Computing & Data Analytics*

A Comprehensive Toolkit for Research

Emmanuel Miguel Gonzalez, Jeffrey Demieville, Brenda Huppenthal, Emily Cawley, Aditya Kumar, Bella Salter, Duke Pauli

2024-04-02

*The PhytoOracle project is support
Research (DE-SC0020401) and Advances
dation Plant Genome Research Progran
(2019674 and 1743442), and CyVerse pro
Department of Agriculture National Inst
We thank Agricultural Genome to Phen

Pauli Lab

Table of Contents

Building Your Computational Toolkit

Session 1:

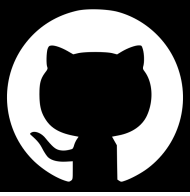
- Fundamental Computational Toolkit

Session 2:

- Machine Learning & Data Visualization

Session 3:

- Applying Computational Toolkit to Plant Phenotyping



Data Management and Distributed Computing (Jeffrey Demieville)

Learning Objectives

1. Describe scenarios where specific file formats may be preferred.
2. Explain the benefits of automated, consistent data handling in projects involving large datasets.
3. Identify scenarios where distributed computing processes are appropriate.
4. Evaluate the advantages of accessing high-performance distributed computing systems.

DATA MANAGEMENT AND DISTRIBUTED COMPUTING

JEFFREY DEMIEVILLE, M.S.
R&D ENGINEER/SCIENTIST III

Pauli Lab

THE UNIVERSITY OF ARIZONA

APRIL 10, 2024

Jeffrey Demieville

Interdisciplinary R&D Engineer

M.S. Systems Engineering
University of Arizona

B.S. Biological/Agricultural Engineering
Texas A&M University

Lead engineer for Maricopa Field Scanner (>5 years)
- Maintenance, Repair, and Operation
- Upgrade and Adaptation
- Efficiency Improvements

Work with the PhytoOracle computing pipelines that process Maricopa Field Scanner data (1 year)

THE UNIVERSITY OF ARIZONA

What is Phenomics?

BIOLOGY

What traits make up a phenotype?

How do genotypes translate to phenotypes?

DATA SCIENCES

What methods are needed to store, catalog, and manage large scale datasets?

Data Management

How are large scale datasets processed to permit analysis?

Distributed Computing

ENGINEERING

What tools and methods need to be developed to measure phenotypes?

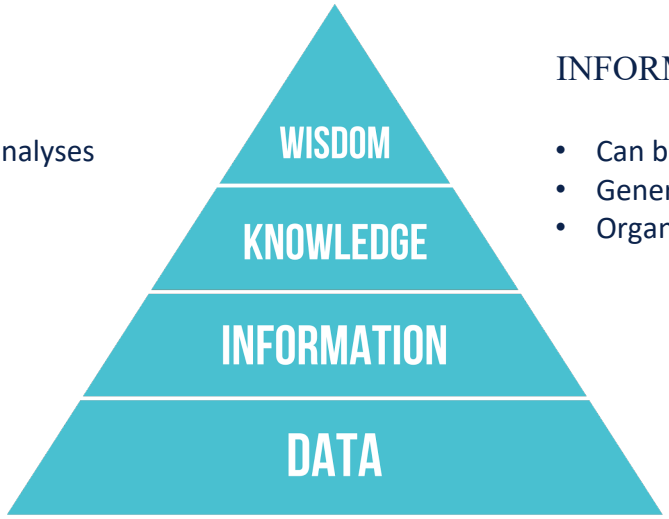
What instrumentation is needed to capture desired datasets?

THE UNIVERSITY OF ARIZONA

WHAT'S THE DIFFERENCE?

DATA:

- Not useful in analyses
- Direct outputs
- Unorganized



INFORMATION:

- Can be used in analyses
- Generated by some form of processing
- Organized

Questions are answered by **analysis** of **information** generated from **data**

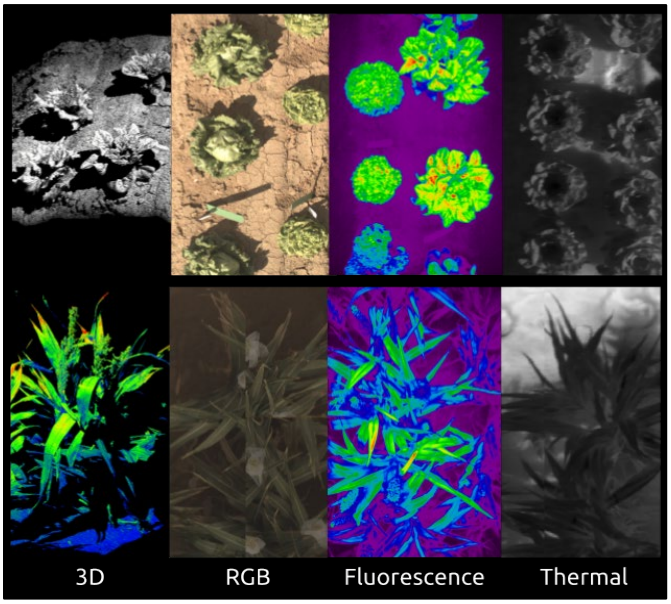
THE UNIVERSITY OF ARIZONA

How has the data from the Maricopa Field Scanner been used to generate usable information?

THE UNIVERSITY OF ARIZONA

The Maricopa Field Scanner has several cameras and sensors

- 3D Laser Scanners
  - ❑ Height
  - ❑ Leaf area
  - ❑ Surface Normal Angles
- RGB Cameras
  - ❑ Triangular Greenness Index
  - ❑ Bounding Area
- PSII Chlorophyll Fluorescence Imager
  - ❑ Status of Photosystem II
- Thermal Infrared Cameras
  - ❑ Transpiration
  - ❑ Heat Stress
  - ❑ Leaf Senescence
  - ❑ Canopy Temperature
- Shortwave Infrared (SWIR) and Visual-Near Infrared (VNIR) Hyperspectral Imagers
  - ❑ Chlorophyll content
  - ❑ Leaf water content
  - ❑ Specific leaf area
  - ❑ Nutrients (Nitrogen, Phosphorus, Potassium)



THE UNIVERSITY OF ARIZONA

Types of Data

Geospatial Data

- Where are targets oriented in space?
- CSVs, Shapefiles

Fieldbooks

- Where are our targets located with respect to each other?
- What additional information (e.g., genotypes) is necessary to tie into these targets?
- CSVs, Excel Workbooks

	E	F	G	H	I	J	
48	02	25	0225	1	border	NA	BTx_623
49	02	26	0226	1	border	NA	BTx_623
50	02	27	0227	1	border	NA	BTx_623
51	02	28	0228	1	border	NA	BTx_623
52	02	29	0229	1	border	NA	BTx_623
53	02	30	0230	1	border	NA	BTx_623
54	02	31	0231	1	border	NA	BTx_623
55	02	32	0232	1	border	NA	BTx_623
56	03	01	0301	1	border	NA	BTx_623
57	03	02	0302	1	experimental	511	PI 67792
58	03	03	0303	1	experimental	317	PI 67791
59	03	04	0304	1	experimental	3762	PI 67826
70	03	05	0305	1	experimental	2313	PI 67811
71	03	06	0306	1	experimental	321	PI 67791
72	03	07	0307	1	experimental	3221	PI 67821

Metadata

- Data informing on other data
- Assists in analysis
- JSON, TXT, HDF5

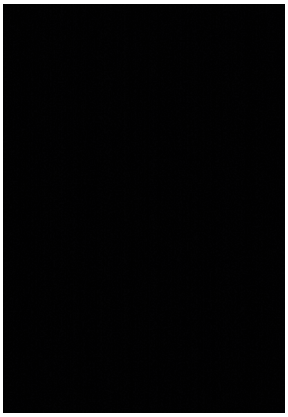

```
    },
    "sensor_variable_metadata": {
      "Rotate flip type - left": "0",
      "Crosshairs - left": "0",
      "exposure - left": "1850",
      "autoexposure - left": "0",
      "gain - left": "1300",
      "autogain - left": "0",
      "gamma - left": "50",
      "rwhitebalanceratio - left": "170",
      "bwhitebalanceratio - left": "103",
      "Rotate flip type - right": "0",
      "Crosshairs - right": "0",
      "exposure - right": "1850",
      "autoexposure - right": "0",
      "gain - right": "1300",
      "autogain - right": "0",
      "gamma - right": "50",
      "rwhitebalanceratio - right": "155",
      "bwhitebalanceratio - right": "110",
      "height left image [pixel]": "2472",
      "width left image [pixel]": "3296",
      "image format left image": "BayerGR8",
      "height right image [pixel]": "2472",
      "width right image [pixel]": "3296",
      "image format right image": "BayerGR8"
    }
  }
```


THE UNIVERSITY OF ARIZONA

Types of Data

Image Data

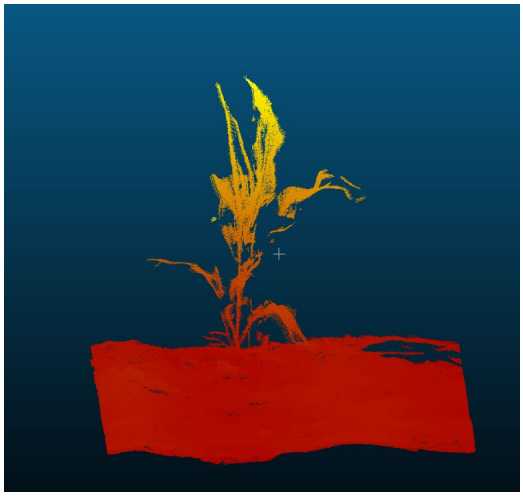
- Output from physical sensors
- Thermal, PSII Fluorescence, RGB
- Variety of wavelengths, number of bands, fields of view
- BIN, JPEG, PNG, HDF5



Types of Data

3D Point Cloud Data

- Output by LIDAR or structured-light 3D scanners
- Varying density, fields of view, methodology
- LAS, PLY, XYZ, PCD

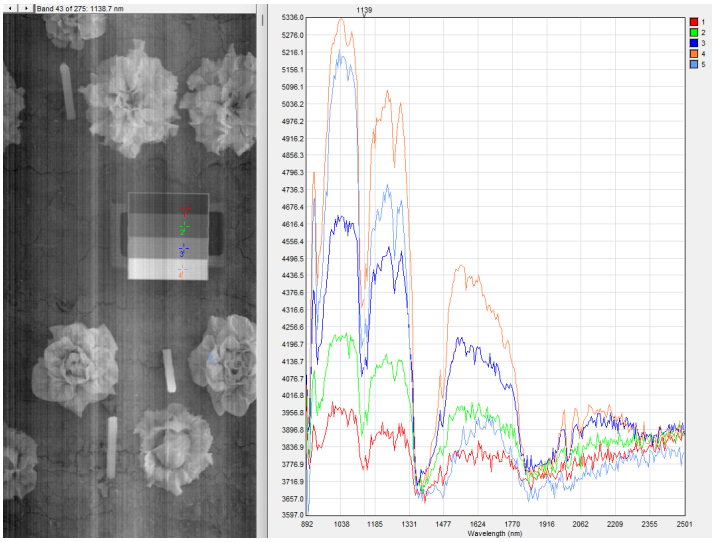


- The field scanner has two structured-light 3D laser scanners capturing from opposite directions to provide a more complete point cloud.
- Full-field scan = two point clouds/measurement, hundreds of measurements

Types of Data

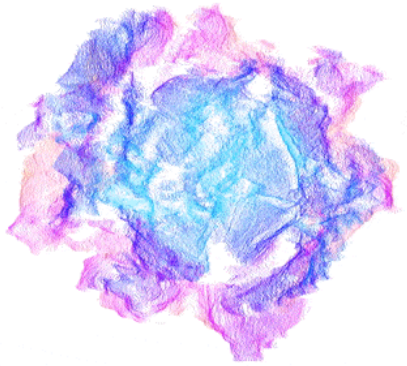
Hyperspectral Sensor Data

- Point measurement or imaging sensor
- Variety of bandwidths, spectral range, and spatial range
- Captures a wide range of data not typically available with normal image sensors
- ENVI, HDF5



THE UNIVERSITY OF ARIZONA

Why so many different formats?



Application dictates suitability.

Data formats, like any standard, have evolved over time. In attempts to solve shortcomings of other standards, new standards are developed and sometimes adopted, but may have their own shortcomings.

- Some analyses more efficiently use data or retain more precision when the data is kept in a raw format (e.g., CSV, BIN, TXT).
- Some data is more efficient to use when additional metadata is provided (e.g., ENVI, HDF5) or when raw data is pre-processed (e.g., JPG, PNG).
- Some data is of a format that it doesn't make sense to use certain data formats. (2D data in 3D format, 4D data in 3D format).

Does a one-size-fits-all solution exist?

NO.

Choices can be made to unify data types for ease of use, efficiency, and consistency in data management.

HDF5: a structured format that can store large, complex data of varying types into a single object.

Beyond collecting and storing data, what else is needed to use and share information?

Data Management

Implementing automated and consistent data handling is essential to maintain data security, integrity, and accessibility throughout its lifecycle.

Data Management

What are the benefits of implementing automated, consistent data handling in projects involving large-scale data collection?

Reduced risks

Automatable

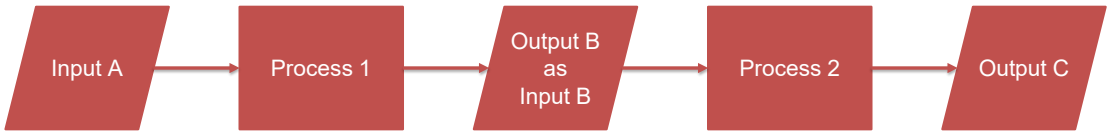
Known locations

You have structured data. Now what?

Use pipelines to get the outputs you need.

A set of interconnected processes taking the outputs of a process as the inputs to the next

Like a factory building a car, there are often several complex steps needed to take raw inputs (data) to a finished product (a complete analysis).



Our team leverages CyVerse for storing and organizing several petabytes of data from the Maricopa Field Scanner.

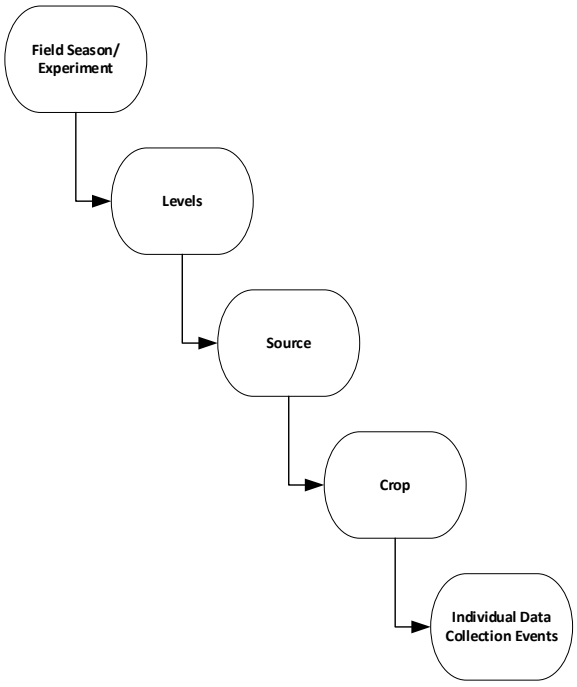
CyVerse allows us to define fixed paths for data, share data with collaborators, and store the products of our computing pipelines, as well as take advantage of the University of Arizona High Performance Computer to distribute our computing.

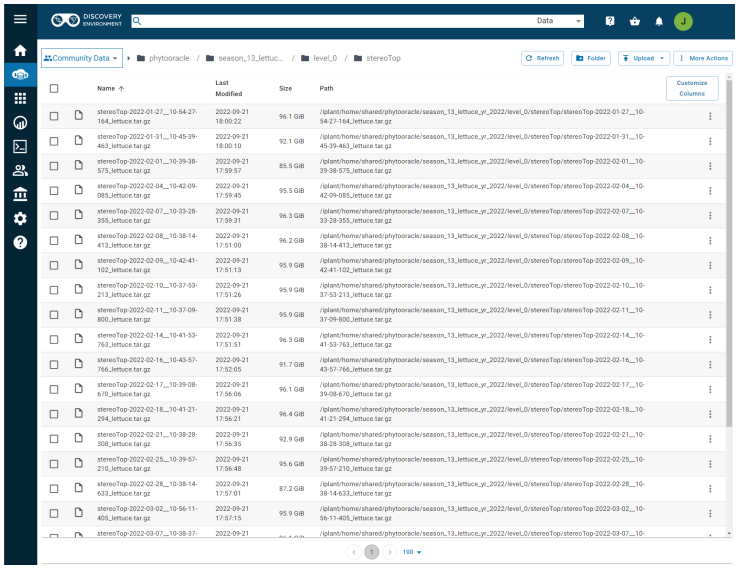
CYVERSE®

cyverse.org

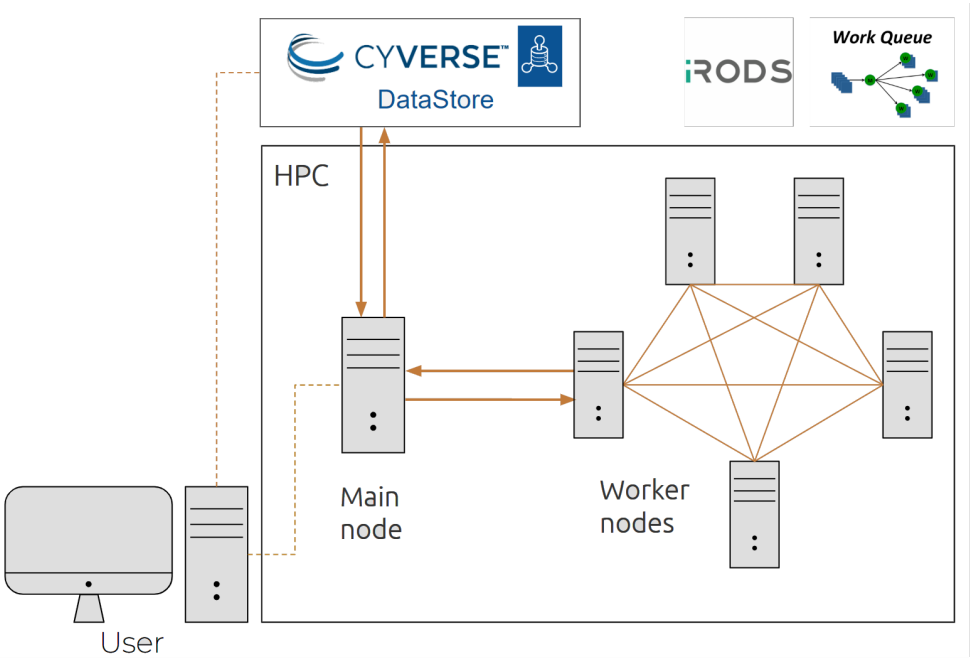
1 Petabyte (PB) = 1,000,000 Gigabytes (GB)

THE UNIVERSITY OF ARIZONA





THE UNIVERSITY OF ARIZONA



THE UNIVERSITY OF ARIZONA

Distributed Computing

What are the advantages of utilizing distributed computing in projects involving large-scale data collection?

Improved scaling

Increased reliability

Increased efficiency

DATA MANAGEMENT AND DISTRIBUTED COMPUTING

github.com/phytooracle

PhytoOracle

What is it?

PhytoOracle is a scalable, modular, and distributed workflow manager. It is intended for use in analyzing high-throughput phenotyping data from a variety of platforms.

What does it do?

PhytoOracle takes our known data types, known data locations, and known processes and integrates them together into a collection of free, open-source processing pipelines.

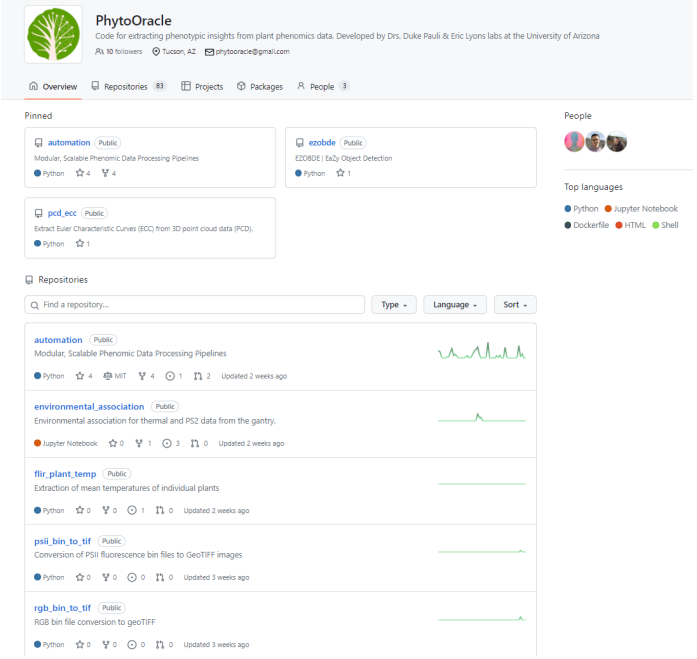
How do we use it?

We develop a set of instructions corresponding to our pipelines. PhytoOracle interprets these instructions and implements our pipelines to provide us usable processed data and information.

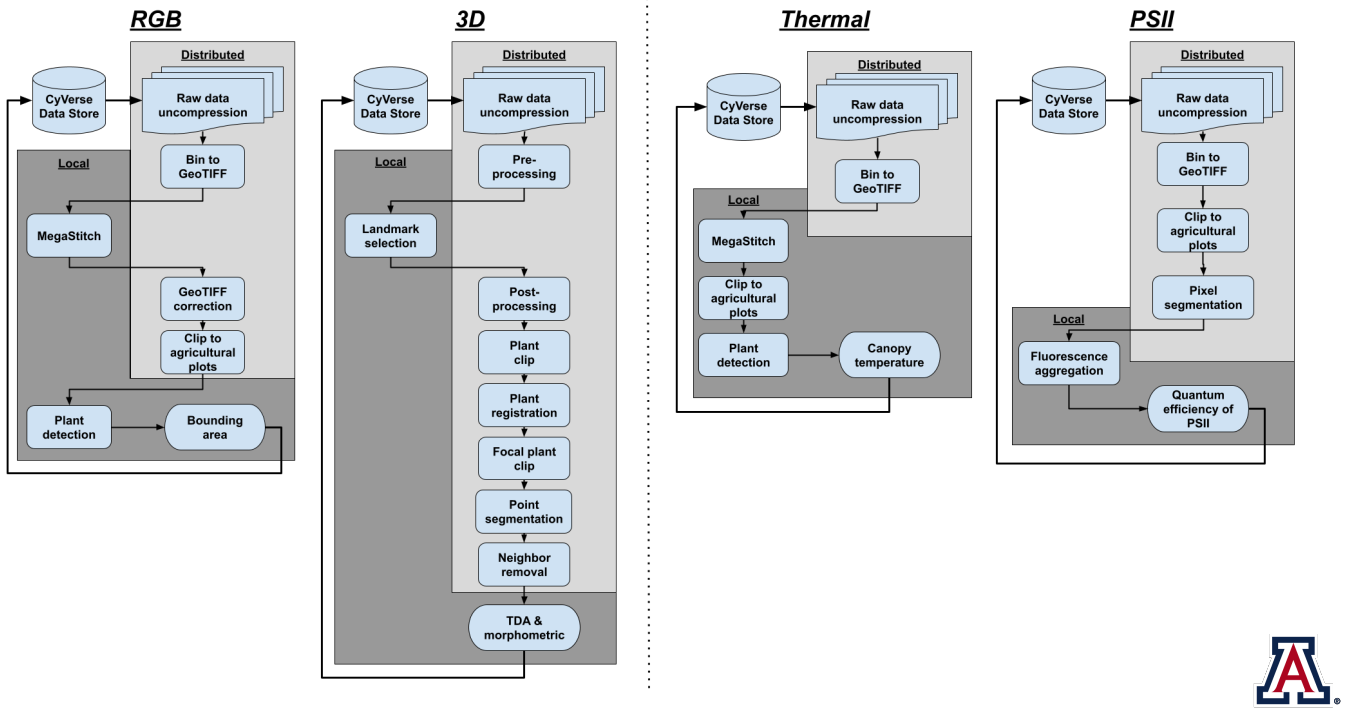
THE UNIVERSITY OF ARIZONA

To generate useful information from the Maricopa Field Scanner, we synthesize several types of raw data together using PhytoOracle processing pipelines.

PhytoOracle contains over 80 publicly-available repositories that can be used, combined, and expanded upon for processing of data.



THE UNIVERSITY OF ARIZONA



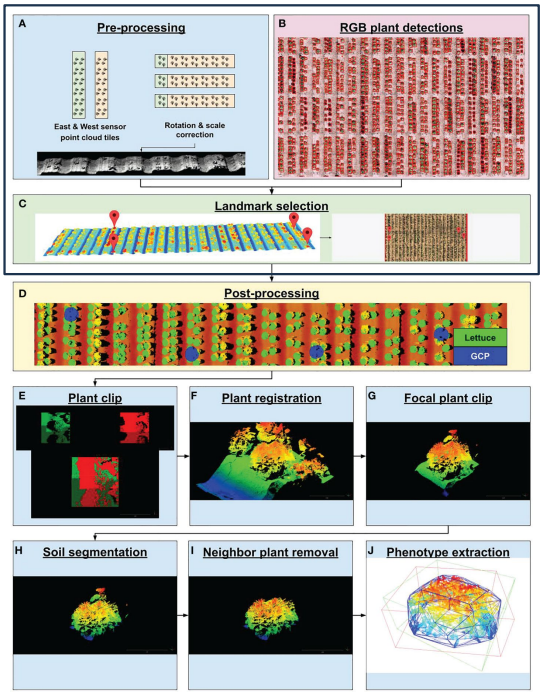
THE UNIVERSITY OF ARIZONA

For quality 3D data, several preprocessing steps are required.

In this stage, we implement:

- Alignment between scanner outputs
- Orientation
- Scaling
- Geospatial referencing

RGB Outputs → Landmark Selection
Allows georeferencing of 3D points



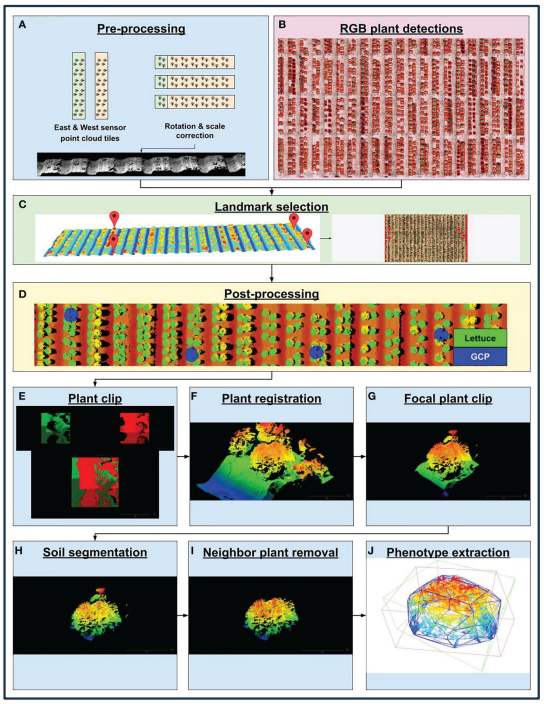
THE UNIVERSITY OF ARIZONA

RGB plant detections and georeferencing
→ Use to clip individual plants from full-field point cloud

Allows tracking, segmentation, further phenotype extraction

Intermediary data products
- → Inputs for additional pipelines
- → Progress review
- → Method evaluation

End-to-end pipelines allow our team to leverage powerful processing tools for higher-throughput phenotyping.



THE UNIVERSITY OF ARIZONA

Thank you

Jeffrey Demieville jdemieville@arizona.edu

Pauli Lab

thepaulilab.com


Dr. Duke Pauli


Emmanuel Gonzalez


Brenda Huppenthal


Aditya Kumar


Emily Cawley


Bella Salter

THE UNIVERSITY OF ARIZONA

Version Control Using GitHub & Google Colab (Emily Cawley)

Learning Objectives

1. Create a GitHub repository and demonstrate basic proficiency by setting up a repository, adding a basic program, and documenting a committed change to the repository.
2. Explore well-documented public projects and repositories to gain insights into naming conventions, commenting style, project documentation, and project development best practices.

Introduction to Python & GitHub

Emily Cawley

Pauli Lab

The University of Arizona

Who am I?

Emily Cawley-Research Intern

I am a photographer, and a senior Computer Science and Global Media Studies major at The University of Arizona. I joined the Pauli Lab in 2023 and previously worked with the PACT Consortium. My area of interest is in AI/Machine Learning and after I graduate, I hope to work within the industry to combine my two areas of study.

Pauli Lab

https://thepaulilab.com/


Dr. Duke Pauli


Emmanuel Gonzalez


Brenda Huppenthal


Aditya Kumar


Jeffery Demieville


Bella Salter

THE UNIVERSITY OF ARIZONA

Overview Python

- Math
- Numbers
- Arrays
- Text
- Tuples
- Booleans

- Loops
- Functions
- Scope
- Numpy
- Google Colab

Overview GitHub

- Online Interface
- Ins and Outs of a Repository
- Git on the Command Line
- Good Coding Practices

In their own words GitHub is " a code hosting platform for version control and collaboration. It lets you and others work together on projects from anywhere."
Let's go over to their website:
https://github.com/

Command Line

Sometimes you will be unable to use a GUI to access a repository
For example, when you are using UA's HPC
This is where the command line comes in

- https://github.com/cli/cli#installation

- https://education.github.com/git-cheat-sheet-education.pdf
- https://docs.github.com/en/get-started/using-git/about-git

THE UNIVERSITY OF ARIZONA

Jupyter Notebooks & Python Data Types (Emily Cawley)

Learning Objectives

1. Create your own Jupyter/Colab notebook and execute basic lines of code.
2. Gain proficiency in Python fundamentals, including data types, conditionals, operators, loops, and functions.
3. Understand and apply best coding practices.
4. Learn to manipulate Numpy arrays effectively.

Workshop_Presentation_Python

April 12, 2024

```
[ ]: import sys
```

1 Python-an overview

Items of note - Runtime Checked - Indent based code blocks

1.1 Math

- Also know as Arithmetic Operations
- Performed with arithmetic operators
  - Addition: **x + y**
  - Subtraction: **x - y**
  - Multiplication: **x * y**
  - Division: **x / y**
  - Modulus: **x % y**
  - Exponents: **x** y**
  - Floor Division: **x // y**

1.2 Numbers

1.2.1 Integers

```
[ ]: sys.int_info
```

```
[ ]: sys.int_info(bits_per_digit=30, sizeof_digit=4, default_max_str_digits=4300,
     str_digits_check_threshold=640)
```

```
[ ]: x = 17
     x
```

```
[ ]: 17
```

```
[ ]: x = sys.maxsize
     y = -x
     print("x:", x)
     print("y:", y)
```

```
x: 9223372036854775807
y: -9223372036854775807
```

1.2.2 Floats

[ ]:
```
sys.float_info
```

[ ]:
```
sys.float_info(max=1.7976931348623157e+308, max_exp=1024, max_10_exp=308,
min=2.2250738585072014e-308, min_exp=-1021, min_10_exp=-307, dig=15,
mant_dig=53, epsilon=2.220446049250313e-16, radix=2, rounds=1)
```

[ ]:
```
import math
```

[ ]:
```
pi = math.pi
pi
```

[ ]:
```
3.141592653589793
```

1.3 Arrays

Documentation: https://docs.python.org/3/tutorial/datastructures.html
In python arrays are called lists.

Properties: > - Ordered > - Mutable > - Type Flexible > - Allow Duplicates

[ ]:
```
of_ints = [5, 6, 1 , 20, 64, 33, 85, 22, 27, 3]
print(of_ints)
```

```
[5, 6, 1, 20, 64, 33, 85, 22, 27, 3]
```

[ ]:
```
of_strings = ['hello world', 'this is a sentence.', 'this is how a lot of data␣
↪can be formatted']
print(of_strings)
```

```
['hello world', 'this is a sentence.', 'this is how a lot of data can be
formatted']
```

[ ]:
```
multi_ray = [5, 'a', 'you can have multiple types too', (4, 'four')]
print(multi_ray)
```

```
[5, 'a', 'you can have multiple types too', (4, 'four')]
```

[ ]:
```
array_dim = [of_ints, of_strings, multi_ray]
print(array_dim)
```

```
[[5, 6, 1, 20, 64, 33, 85, 22, 27, 3], ['hello world', 'this is a sentence.',
'this is how a lot of data can be formatted'], [5, 'a', 'you can have multiple
types too', (4, 'four')]]
```

Size and Indexing

```
[ ]: print(of_ints)
     print("Length: ", len(of_ints)  )
```

```
[5, 6, 1, 20, 64, 33, 85, 22, 27, 3]
Length:  10
```

```
[ ]: of_ints[0]
```

```
[ ]: 5
```

Slicing Notation: array[lower_bound : upper_bound : step]

```
[ ]: of_ints
```

```
[ ]: [5, 6, 1, 20, 64, 33, 85, 22, 27, 3]
```

```
[ ]: print(of_ints[:7:-1])
     print(of_ints[:7])
     print(of_ints[3:])
```

```
[3, 27]
[5, 6, 1, 20, 64, 33, 85]
[20, 64, 33, 85, 22, 27, 3]
```

Searching

```
[ ]: of_ints
```

```
[ ]: [5, 6, 1, 20, 64, 33, 85, 22, 27, 3]
```

```
[ ]: print( 3 in of_ints)
     print(44 in of_ints)
```

```
True
False
```

```
[ ]: of_strings
```

```
[ ]: ['hello world',
      'this is a sentence.',
      'this is how a lot of data can be formatted']
```

```
[ ]: of_strings
```

```
[ ]: ['hello world',
      'this is a sentence.',
      'this is how a lot of data can be formatted']
```

[ ]:
```
print('hello world' in of_strings)
print('Hello World' in of_strings)
```

```
True
False
```

[ ]:
```
print(of_strings in array_dim)
```

```
True
```

Changing arrays

[ ]:
```
ray = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

[ ]:
```
print(ray)
print(ray.index(5))
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
4
```

[ ]:
```
ray.remove(4)
print(ray)
```

```
[1, 2, 3, 5, 6, 7, 8, 9, 10]
```

[ ]:
```
ray.append(4)
print(ray)
```

```
[1, 2, 3, 5, 6, 7, 8, 9, 10, 4]
```

[ ]:
```
ray.pop(-1)
```

[ ]:
```
4
```

[ ]:
```
ray.insert(3, 4)
print(ray)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

[ ]:
```
del ray[3]
ray
```

[ ]:
```
[1, 2, 3, 5, 6, 7, 8, 9, 10]
```

[ ]:
```
ray.insert(3,4)
ray
```

[ ]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

[ ]:
```
ray[3]=77
ray
```

[ ]:
```
[1, 2, 3, 77, 5, 6, 7, 8, 9, 10]
```

[ ]:
```
ray=[1,2,3,4,5]
```

[ ]:
```
ray = ray*2
print(ray)
```

```
[1, 2, 3, 4, 5, 1, 2, 3, 4, 5]
```

1.4 Text

1.4.1 Characters

- Technially Python does not have an explicit type char
- Are numbers -Unicode character encodings

1.4.2 Strings

- Are an array of characters
- Can use **either** ” ” or ’ ’
- Formatted strings
- Multiline Strings

[ ]:
```
x = "Hello World!"
y = 'Hello World!'
print(x)
print(y)
print(x[4]) # is a character in other languages
```

```
Hello World!
Hello World!
o
```

[ ]:
```
requests = 5
print(f"The user requested this {requests} many times\n")
```

```
The user requested this 5 many times
```

[ ]:
```
block_quote = """Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed␣
 ↪do eiusmod tempor incididunt ut labore et dolore magna aliqua.
Urna condimentum mattis pellentesque id nibh tortor.
Tortor consequat id porta nibh venenatis cras sed felis eget.
Dictum non consectetur a erat nam at. """

print(block_quote)
```

```
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor
incididunt ut labore et dolore magna aliqua.
Urna condimentum mattis pellentesque id nibh tortor.
Tortor consequat id porta nibh venenatis cras sed felis eget.
Dictum non consectetur a erat nam at.
```

Searching

[ ]:
```
print('world' in x)
print('World' in x)
```

```
False
True
```

Processing

[ ]:
```
bq_stripped = block_quote.strip()
print(bq_stripped, "\n\n")

bq_rem = block_quote.replace(" ", "")
print(bq_rem, "\n\n")

bq_split = block_quote.split(" ")
print(bq_split)
```

```
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor
incididunt ut labore et dolore magna aliqua.
Urna condimentum mattis pellentesque id nibh tortor.
Tortor consequat id porta nibh venenatis cras sed felis eget.
Dictum non consectetur a erat nam at.


Loremipsumdolorsitamet,consecteturadipiscingelit,seddoeiusmodtemporincididuntutl
aboreetdoloremagnaaliqua.
Urnacondimentummattispellentesqueidnibhtortor.
Tortorconsequatidportanibhvenenatiscrassedfeliseget.
Dictumnonconsecteturaeratnamat.


['Lorem', 'ipsum', 'dolor', 'sit', 'amet,', 'consectetur', 'adipiscing',
'elit,', 'sed', 'do', 'eiusmod', 'tempor', 'incididunt', 'ut', 'labore', 'et',
'dolore', 'magna', 'aliqua.\nUrna', 'condimentum', 'mattis', 'pellentesque',
'id', 'nibh', 'tortor.\nTortor', 'consequat', 'id', 'porta', 'nibh',
'venenatis', 'cras', 'sed', 'felis', 'eget.\nDictum', 'non', 'consectetur', 'a',
'erat', 'nam', 'at.', '']
```

Concatenating

[ ]:
```
x = 'Tortor consequat id porta nibh venenatis cras sed felis eget. '
y = "Dictum non consectetur a erat nam at."

z = x+y
print(z)
print("\n", x, "\n", y)
```

```
Tortor consequat id porta nibh venenatis cras sed felis eget. Dictum non
consectetur a erat nam at.

 Tortor consequat id porta nibh venenatis cras sed felis eget.
 Dictum non consectetur a erat nam at.
```

1.5 Tuples

- Ordered
- Immutable
- Type Flexible
- Allow Duplicates

[ ]:
```
tup = ('red', 'pink', 'blue', 'yellow', 'yellow')
print(tup)
x = tup[0]
x
```

```
('red', 'pink', 'blue', 'yellow', 'yellow')
```

[ ]:
```
'red'
```

Packing and Unpacking

[ ]:
```
names = ( "Astarion", "Gale", "Lae'zel", "Wyll")
(c1, c2, c3, c4) = names

print(c1)
print(c2)
print(c3)
print(c4)
```

```
Astarion
Gale
Lae'zel
Wyll
```

[ ]:
```
names = ( "Astarion", "Gale", "Lae'zel", "Wyll")
v1, v2, _, _ = names

print(v1)
```

```
print(v2)
```

```
Astarion
Gale
```

1.6 Dictionaries

- Maps Key-Value pairs
- Visualize as a table
- Type flexible
- Mutable
- No duplicate keys allowed

[ ]:
```
dict = {
    "RSVP" : 'Répondez s'il vous plait',
    'ASAP' : 'as soon as possible',
    "BBEG" : 'big bad evil guy',
    "SMH" : 'shaking my head',
    'AWOL' : 'absent without leave',
    'UFO' : 'unidentified flying object',
    "SCUBA" : "Self-contained underwater breathing apparatus"
}
```

[ ]:
```
print(dict)
```

```
{'RSVP': 'Répondez s'il vous plait', 'ASAP': 'as soon as possible', 'BBEG': 'big
bad evil guy', 'SMH': 'shaking my head', 'AWOL': 'absent without leave', 'UFO':
'unidentified flying object', 'SCUBA': 'Self-contained underwater breathing
apparatus'}
```

[ ]:
```
k=dict.keys()
v = dict.values()
print(k)
print(v)
```

```
dict_keys(['RSVP', 'ASAP', 'BBEG', 'SMH', 'AWOL', 'UFO', 'SCUBA'])
dict_values(['Répondez s'il vous plait', 'as soon as possible', 'big bad evil
guy', 'shaking my head', 'absent without leave', 'unidentified flying object',
'Self-contained underwater breathing apparatus'])
```

Changing

[ ]:
```
dict["CAD"] = "comupter aided design"
print(k)
print(v)
print(dict)
```

```
dict_keys(['RSVP', 'ASAP', 'BBEG', 'SMH', 'AWOL', 'UFO', 'SCUBA', 'CAD'])
dict_values(['Répondez s'il vous plait', 'as soon as possible', 'big bad evil
```

```
guy', 'shaking my head', 'absent without leave', 'unidentified flying object',
'Self-contained underwater breathing apparatus', 'comupter aided design'])
{'RSVP': 'Répondez s'il vous plait', 'ASAP': 'as soon as possible', 'BBEG': 'big
bad evil guy', 'SMH': 'shaking my head', 'AWOL': 'absent without leave', 'UFO':
'unidentified flying object', 'SCUBA': 'Self-contained underwater breathing
apparatus', 'CAD': 'comupter aided design'}
```

[ ]:
```
dict["SMH"] = "shrimp must have"
print(dict["SMH"])
```

```
shrimp must have
```

[ ]:
```
dict.pop("SMH")
print(dict)
```

```
{'RSVP': 'Répondez s'il vous plait', 'ASAP': 'as soon as possible', 'BBEG': 'big
bad evil guy', 'AWOL': 'absent without leave', 'UFO': 'unidentified flying
object', 'SCUBA': 'Self-contained underwater breathing apparatus', 'CAD':
'comupter aided design'}
```

1.7 Booleans

Only two choices: True or False
If you are doing discrete math you are using booleans

[ ]:
```
x= True
y=False
```

[ ]:
```
True == 1
```

[ ]:
```
True
```

Logical Operators

[ ]:
```
a = x or y
b = x and y
c = not x

print(a, b, c)
```

```
True False False
```

[ ]:
```
y or not x and y
```

[ ]:
```
False
```

[ ]:
```
y or not (x and y)
```

[ ]:
```
True
```

Comparison Operators

- ==
- !=
- >
- <
- >=
- <=

Identity Operators Identity operators test whether the things being compared are the same object > - is > - is not

[ ]:
```
ray1 = ["one", 4, "two"]
ray2 = ray1.copy()
ray3 = ray1

print(ray1)
print(ray2)
print(ray3)
```

```
['one', 4, 'two']
['one', 4, 'two']
['one', 4, 'two']
```

[ ]:
```
ray1 is ray3
```

[ ]:
```
True
```

[ ]:
```
ray1 is ray2
```

[ ]:
```
False
```

Membership Operators

[ ]:
```
"one" in ray1
```

[ ]:
```
True
```

[ ]:
```
5 in ray1
```

[ ]:
```
False
```

Type Identification and Manipulation

1.8 Loops

-

1.8.1 For

-

1.8.2 While

[ ]:
```
for i in range(6):
  print(i)
  if i==3:
    break
```

```
0
1
2
3
```

[ ]:
```
for x in of_ints:
    print(x)
```

```
5
6
1
20
64
33
85
22
27
3
```

[ ]:
```
for key in dict:
    print(key + ": "+ dict[key])
```

```
RSVP: Répondez s'il vous plait
ASAP: as soon as possible
BBEG: big bad evil guy
AWOL: absent without leave
UFO: unidentified flying object
SCUBA: Self-contained underwater breathing apparatus
CAD: comupter aided design
```

[ ]:
```
i=0;
while(i<5):
  print(of_ints[i]);
  i+=1;
```

```
5
6
1
```

20
64

1.9 Functions

[ ]:
```
def randfunction(dictionary, entry, x, y):
  """ RandFunction: adds a key-value pair into a dictionary
  parameters: dictionary-dictionary containing acronyms
  entry-new entry into the dictionary
  x: integer for condition
  y: integer for condition
  return: 'completed' """
  if x > y:
    dictionary[entry[0]]= entry[1]

  return "completed"
```

[ ]:
```
x= randfunction(dict, ("STEM", "science, technology, engineering, and math"),␣
 ↪7, 5)
```

[ ]:
```
'completed'
```

[ ]:
```
dict
```

[ ]:
```
{'RSVP': 'Répondez s'il vous plait',
 'ASAP': 'as soon as possible',
 'BBEG': 'big bad evil guy',
 'AWOL': 'absent without leave',
 'UFO': 'unidentified flying object',
 'SCUBA': 'Self-contained underwater breathing apparatus',
 'CAD': 'comupter aided design',
 'STEM': 'science, technology, engineering, and math'}
```

[ ]:
```
print(x)
```

```
completed
```

1.9.1 Variable Scope

Variables declared outside a function are always global
Variables declared inside a function can be denoted as global with the global keyword
If you are changing a global variable inside a function invoke global

[ ]:
```
x=2
def randfunction2(entry, x, y):
    print(x)
    global dictionary
    dictionary = {}
```

```
    if x > y:
        dictionary[entry[0]]= entry[1]
```

[ ]:
```
randfunction2(("EDA", "exploratory data analysis"), 7,5)
```

```
7
```

[ ]:
```
dictionary
```

[ ]:
```
{'EDA': 'exploratory data analysis'}
```

2 Numpy

References: https://numpy.org/devdocs/user/absolute_beginners.html

[ ]:
```
import numpy as np
```

[ ]:
```
np_array = np.random.randint(100, size=(4,3))
```

[ ]:
```
np_array
```

[ ]:
```
array([[39, 14, 28],
       [46, 40, 17],
       [20, 74, 56],
       [28, 19, 77]])
```

[ ]:
```
np_array.size
```

[ ]:
```
12
```

[ ]:
```
np_array.shape
```

[ ]:
```
(4, 3)
```

So what's the difference between a python arraylist and a numpy array?

- All elements must be the same type
- Total size of the array is immutable
- Each row must have the same number of columns
- Slicing does not return a copy but returns a view which is linked to the original object in memory

[ ]:
```
ray = [1,2,3,4,5,6,7,8,9,10]
nray = np.array(ray)
```

[ ]:
```
nray
```

[ ]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

The standard numpy array generators

[ ]:
```
np.zeros(10)
```

[ ]:
```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

[ ]:
```
np.ones(10)
```

[ ]:
```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

[ ]:
```
np.empty(10)
```

[ ]:
```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

[ ]:
```
np.arange(10)
```

[ ]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

[ ]:
```
np.arange(4,10,2)
```

[ ]:
```
array([4, 6, 8])
```

[ ]:
```
np.linspace(0, 10, 7)
```

[ ]:
```
array([ 0.        ,  1.66666667,  3.33333333,  5.        ,  6.66666667,
        8.33333333, 10.        ])
```

[ ]:
```
a = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
b = np.array([[5, 6, 7, 8], [1, 2,3, 4]])
```

[ ]:
```
c = np.concatenate((a,b), axis = 1)
```

[ ]:
```
c
```

[ ]:
```
array([[1, 2, 3, 4, 5, 6, 7, 8],
       [5, 6, 7, 8, 1, 2, 3, 4]])
```

[ ]:
```
x = np.array([[1, 2], [3, 4]])
y = np.array([[5, 6]])
```

[ ]:
```
z = np.concatenate((y,x), axis=0)
```

[ ]:
```
z
```

[ ]:
```
array([[5, 6],
       [1, 2],
       [3, 4]])
```

[ ]:
```
new_ray = nray*z
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-142-735e44523155> in <cell line: 1>()
----> 1 new_ray = nray*z

ValueError: operands could not be broadcast together with shapes (10,) (3,2)
```

Reshaping arrays

[ ]:
```
np_array
```

[ ]:
```
array([[39, 14, 28],
       [46, 40, 17],
       [20, 74, 56],
       [28, 19, 77]])
```

[ ]:
```
np_array.shape
```

[ ]:
```
(4, 3)
```

[ ]:
```
np_array = np_array.reshape((2,6))
```

[ ]:
```
np_array
```

[ ]:
```
array([[39, 14, 28, 46, 40, 17],
       [20, 74, 56, 28, 19, 77]])
```

[ ]:
```
np_array.T
```

[ ]:
```
array([[39, 20],
       [14, 74],
       [28, 56],
       [46, 28],
       [40, 19],
       [17, 77]])
```

[ ]:
```
np_array.flatten()
```

[ ]:
```
array([39, 14, 28, 46, 40, 17, 20, 74, 56, 28, 19, 77])
```

[ ]:
```
np_array.reshape((1, 12))
```

```
[ ]: array([[39, 14, 28, 46, 40, 17, 20, 74, 56, 28, 19, 77]])
```

3 Packages

- Scipy: https://pypi.org/project/scipy/

  SciPy (pronounced "Sigh Pie") is an open-source software for mathematics, science, and engineering. It includes modules for statistics, optimization, integration, linear algebra, Fourier transforms, signal and image processing, ODE solvers, and more.

- Pingouin: https://pingouin-stats.org/build/html/index.html

  Pingouin is an open-source statistical package written in Python 3 and based mostly on Pandas and NumPy. Some of its main features include: ANOVAs, Pairwise post-hocs tests and pairwise correlations, Robust, partial, distance and repeated measures correlations, Linear/logistic regression and mediation analysis, Bayes Factors, Multivariate tests, Reliability and consistency, Effect sizes and power analysis, Parametric/bootstrapped confidence intervals around an effect size or a correlation coefficient, Circular statistics, Chi-squared tests, Plotting: Bland-Altman plot, Q-Q plot, paired plot, robust correlation…

- Sklearn: https://scikit-learn.org/stable/

  Machine learning library with classification, regression, clustering algorithms as well as dimensionality reduction, model selection, and preprocessing.

*Open3D: https://www.open3d.org/

  Open3D is a open-source libray that supports 3D data. It features 3D data structures, 3D data processign algorithms, scene reconstruction, surface alignment, 3D visualizations, machine learning support, and GPU acceleation for core 3D operations.

Session 2: Incorporating Machine Learning & Data Visualization into the Computational Toolkit

Session 2 Introduction

Scientific Computing & Data Analytics: A Comprehensive Toolkit for Research

Emmanuel Miguel Gonzalez, Jeffrey Demieville, Brenda Huppenthal, Emily Cawley, Aditya Kumar, Bella Salter, Duke Pauli

Award No. 2102120

Pauli Lab

About the Presenters

Emmanuel Gonzalez

Brenda Huppenthal

Jeffrey Demieville

Emily Cawley

Bella Salter

Aditya Kumar

Pauli Lab

THE UNIVERSITY OF ARIZONA

PhytoOracle

Building Your Computational Toolkit

Session 1:

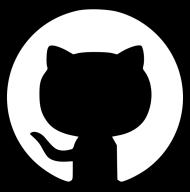
- Fundamental Computational Toolkit

Session 2:

- Machine Learning & Data Visualization

Session 3:

- Applying Computational Toolkit to Plant Phenotyping



Building Your Computational Toolkit

Session 1:

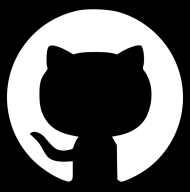
- Fundamental Computational Toolkit

Session 2:

- Machine Learning & Data Visualization

Session 3:

- Applying Computational Toolkit to Plant Phenotyping

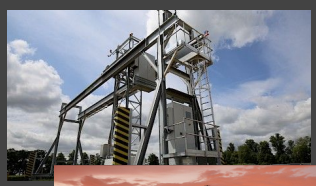



Machine Learning in Scientific Computing

Big Data Necessitates Machine Learning

Many Options to Collect Data Exist

Robots

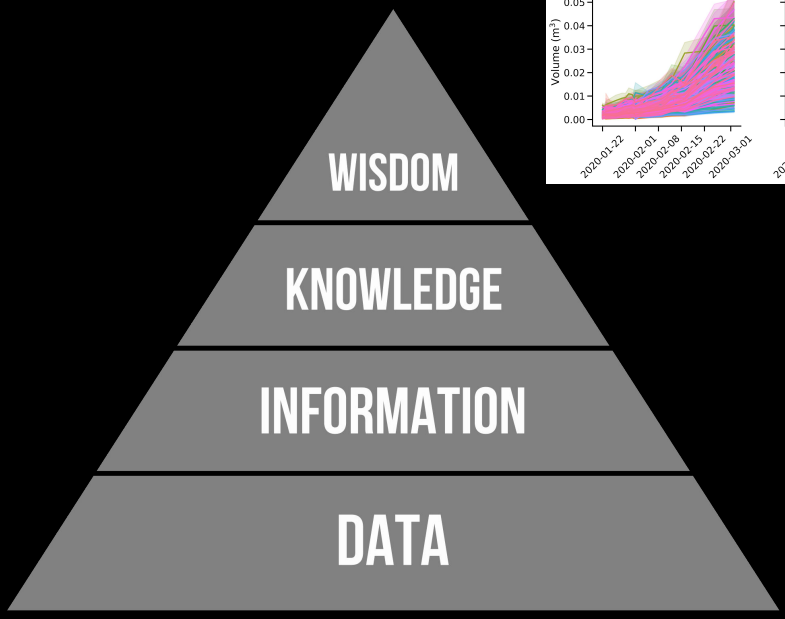


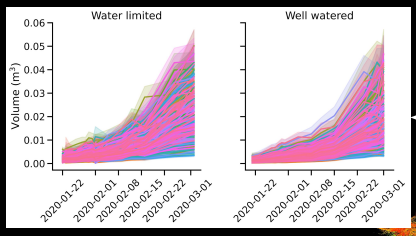
Carts

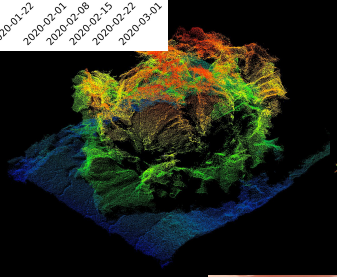
Drones

Phones

Machine Learning Helps Extract to Extract Insights







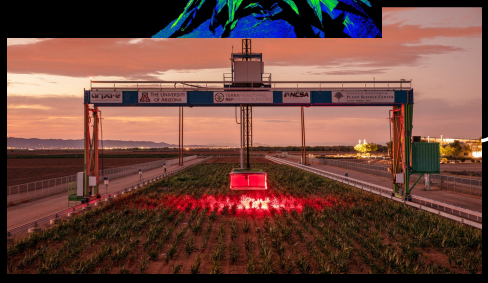


Image credit: Longlivetheux via Wikimedia Commons

Machine Learning Helps Extract to Extract Insights

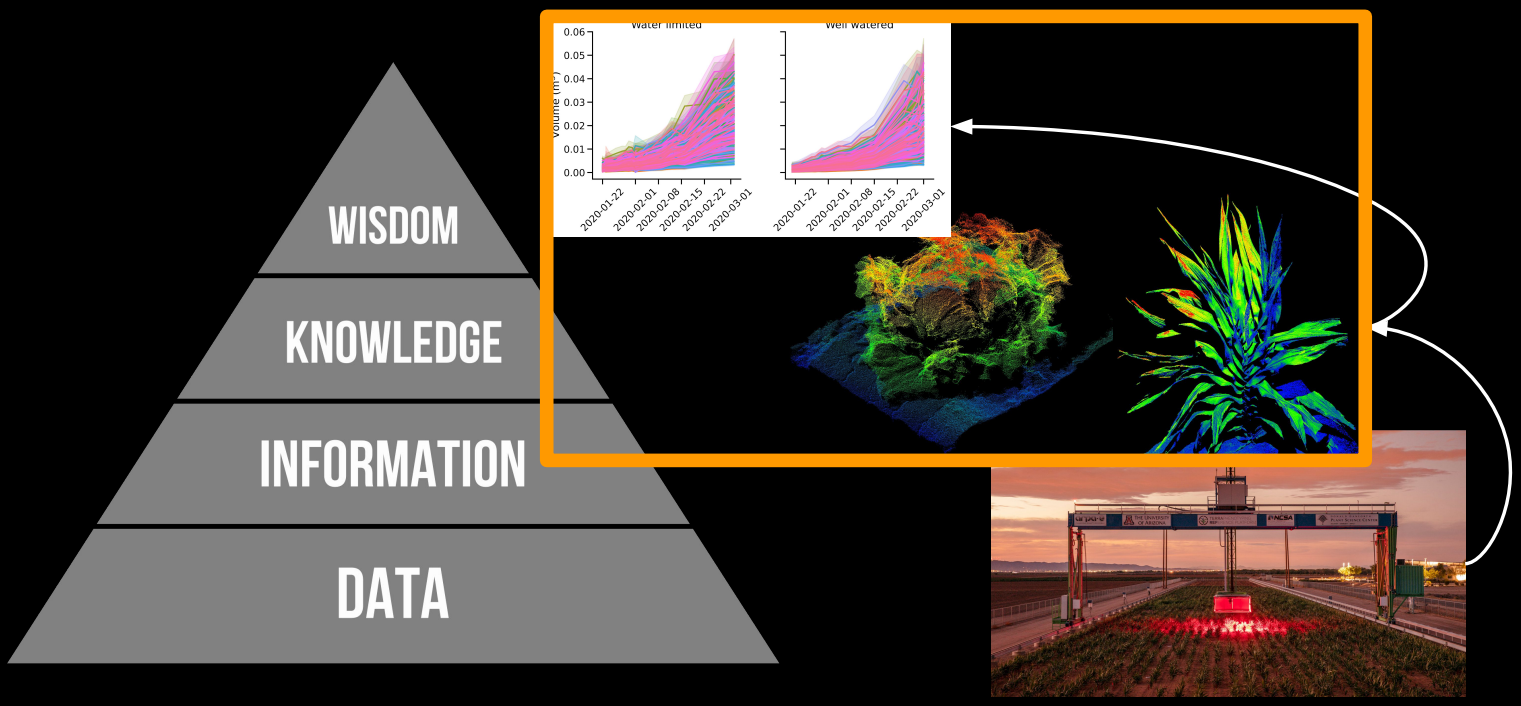
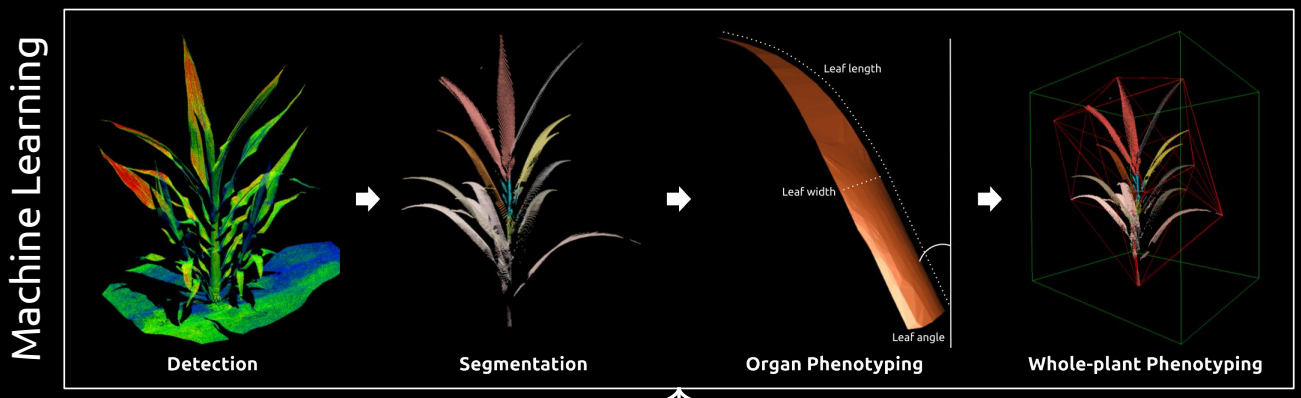
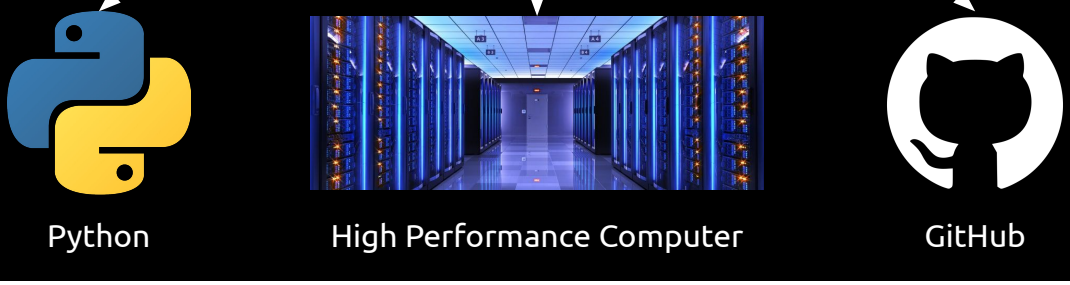


Image credit: Longlivetheux via Wikimedia Commons

Machine Learning Is Vital in This Process

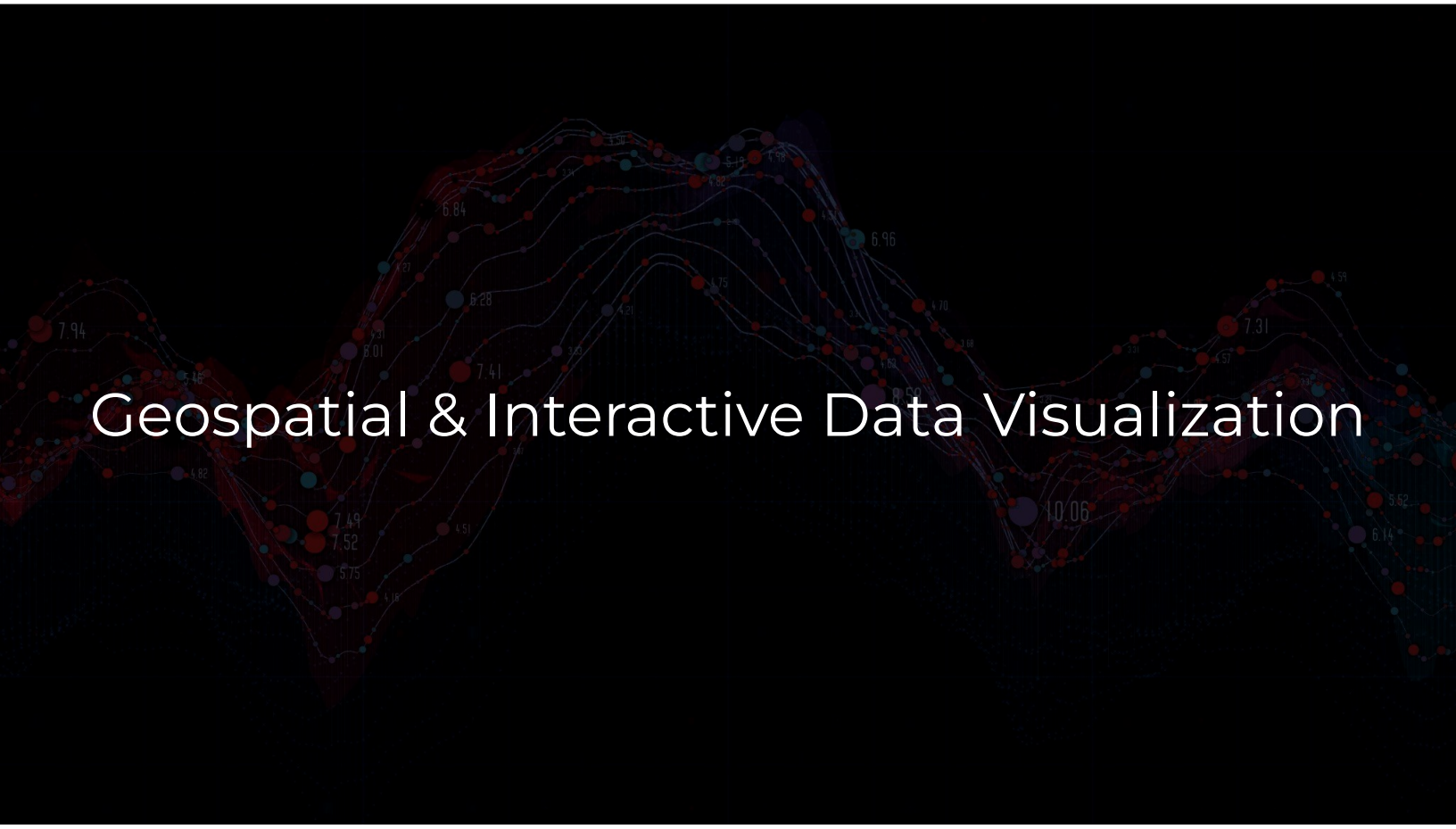




Python

High Performance Computer

GitHub


Geospatial & Interactive Data Visualization

Finding Patterns in Data Using Interactive Visualizations

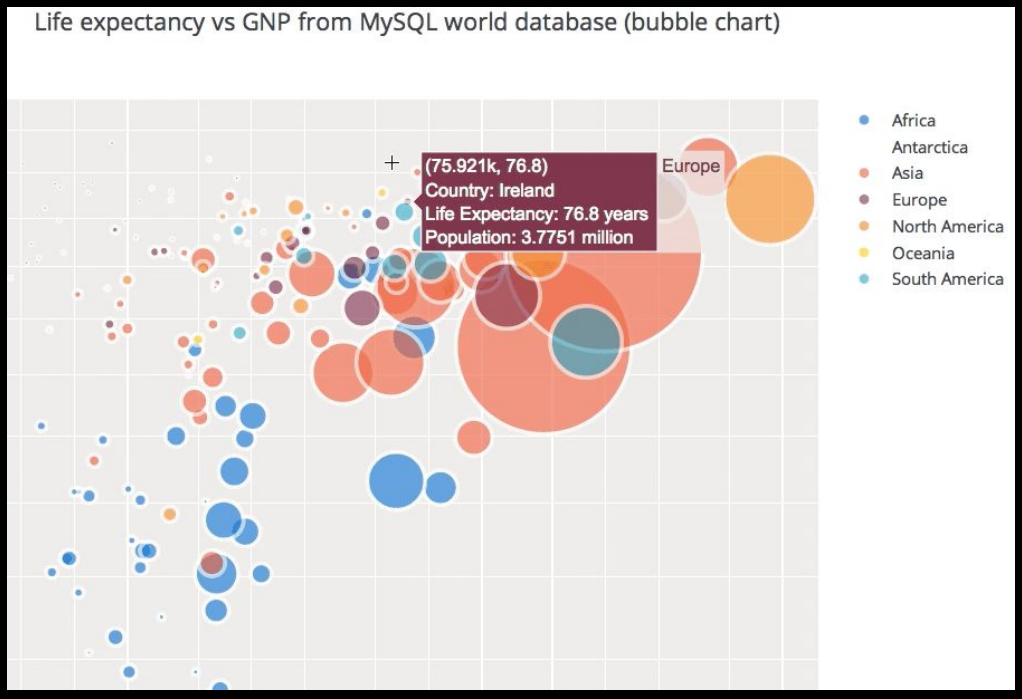
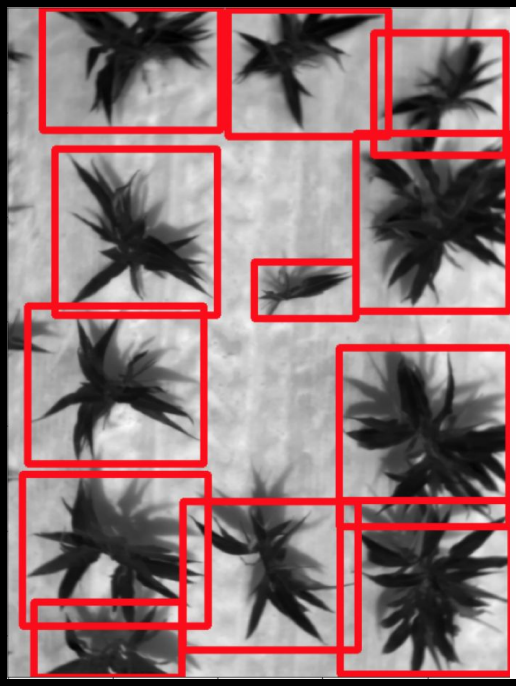
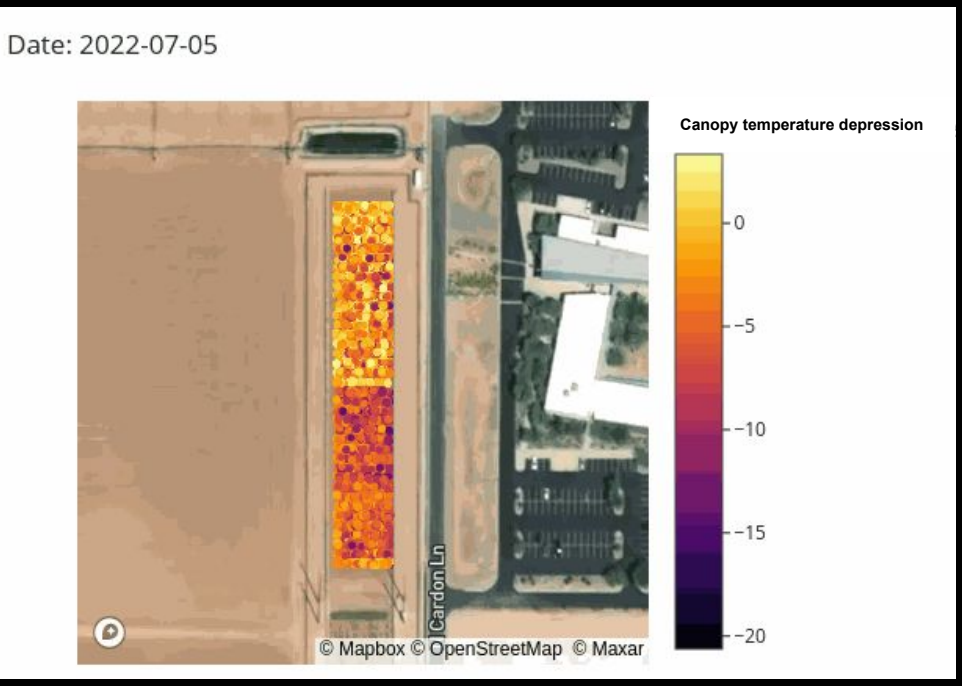


Image credit: Analytics Vidhya

Geospatial Visualizations Highlight Spatial Trends





Access to Workshop Materials

Contains workshop overview, learning objectives, and code

bit.ly/AG2PI_SciComp

Pauli Lab

Scientific Computing & Data Analytics*

A Comprehensive Toolkit for Research

Emmanuel Miguel Gonzalez, Jeffrey Demieville, Brenda Huppenthal,
Emily Cawley, Aditya Kumar, Bella Salter, Duke Pauli

2024-04-02

*The PhytoOracle project is support
Research (DE-SC0020401) and Advance
dation Plant Genome Research Progran
(2019674 and 1743442), and CyVerse pro
Department of Agriculture National Inst
We thank Agricultural Genome to Phen

Pauli Lab

Table of Contents

Introduction to Machine Learning (Bella Salter)

Learning Objectives

1. Distinguish between machine learning algorithms, such as regression and neural networks.
2. Develop a comprehension of machine learning, encompassing aspects such as model training, performance evaluation, and interpretation.

Machine Learning

Bella Salter

Pauli Lab

THE UNIVERSITY OF ARIZONA

April 11, 2024

Bella Salter

Undergraduate Research Assistant

Majoring in Mathematics and Computer Science and minoring in Statistics and Biology at the University of Arizona

I have worked with the Pauli lab since March 2023.

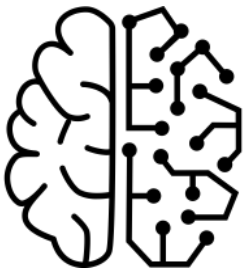
Currently working on a project to study late season lettuce growth based on their performance early in the growing cycle using a long short term memory model.

THE UNIVERSITY OF ARIZONA

Session Overview

- Basic principles of machine learning and its implementation
- Types of machine learning models
- Machine learning example

What is machine learning?



- Applied statistics in computing problems allows machines to imitate the way humans process information and ultimately make decisions.
- It is used for computer visualization, image recognition, language processing, and more.
- The goal is to give a program some data and parameters and end up with a useful prediction.
- Think of trying to create a general best-fit line through data by hand. You would probably look at the data and try to split the difference to find a close enough function to describe it. This is what machine learning does, only by using algorithms and statistics.

Components of Machine Learning Code

COLLECT DATA

Collecting sufficient data is essential to training the model.

A large amount of data will make your model more accurate.

SPLIT DATA

Splitting data into testing and training sets is critical to fitting.

This function must be random to reduce bias.

TRAIN MODEL

There are many different models designed for different tasks.

Selecting the right model is an art that is vital to success.

EVALUATE

Various statistical measures can be used to analyze predictions.

Metrics of evaluation depend on the type of model as well as its goal.

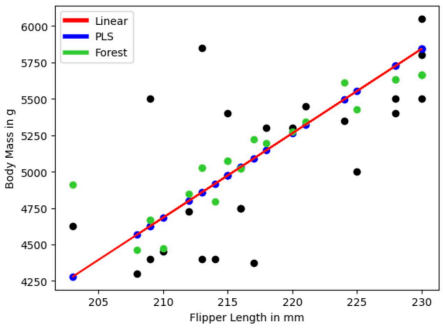
What are some different machine learning models?

- Different models need to be used to analyze different data types, such as image data, 3D data, and more. We will primarily focus on numerical data for this portion of the workshop.
- Linear regression is the simplest type of model. Put simply, it assumes the data is linear, and predicts outcomes based on a best-fit line.
- Neural networks, on the other hand, could be thought of as the most complex. These rely on intricate pieces of computer software that more closely resemble human neurons.

What are some models we can implement today?

- Linear Regression is often used for simpler predictions based on one set of observations.
- Partial Least Squares Regression is used for complex data(hyperspectral) with many types of input.
- RandomForestRegression is often used for predicting future cost.
- Classification models, like KMeans, can be used to show similarities between groups.

How do we measure a model's success?



- A model's success metric should depend on the type of data analyzed, as well as what we, as the programmers, want the outcome to be.
- A commonly used metric of success is the RMSE, or the root mean square error. In essence, it measures the average difference of the predicted values from the actual values.
- Additionally, predictions from different models can be compared to determine which is the most successful.

Overfitting vs. Underfitting

- Two common problems with a trained model are overfitting and underfitting.
- Overfitting refers to the phenomenon where a model too closely resembles its training data. This means that when the information needs to be extrapolated to other data points, its performance suffers.
- On the other hand, underfitting refers to a model that does not describe its data well enough.
- Both these problems can be minimized by properly tuning parameters and splitting up training and testing data in the correct proportions.

Overfitting vs. Underfitting

- These errors are why we split up the data into testing and training sets. We can see how the model uses its prior information gathered from the training set by analyzing its performance on our testing set.
- The typical split for training and testing is 80/20, when our model has no extra parameters.
- Some models have extra inputs, called hyperparameters, such as model depth, learning rate, etc. These require a validation set, so that you can seek out the best parameter. The typical split for train/test/validation in this case is 80/10/10.

Thank you

Bella Salter msalter@arizona.edu

Pauli Lab

https://thepaulilab.com/

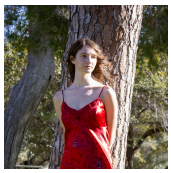

Dr. Duke Pauli


Emmanuel Gonzalez


Jeffrey Demieville


Brenda Huppenthal


Aditya Kumar


Emily Cawley

MACHINE LEARNING

THE UNIVERSITY OF ARIZONA

introMachineLearning

April 12, 2024

1 Introduction to Machine Learning

This is an example of a simple linear model using a dataset about penguins.

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import plotly.express as px

from sklearn import preprocessing, svm
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.cross_decomposition import PLSRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn import metrics

dest = "https://data.cyverse.org/dav-anon/iplant/home/msalter/penguins_size.csv"

col_names = ["culmen_length_mm", "culmen_depth_mm", "flipper_length_mm",␣
 ↪"body_mass_g", "sex"]

data = pd.read_csv(dest, on_bad_lines='skip')
data.shape # this allows us to view the shape of the data
print(data) # here, you can see that we have a few different data points about␣
 ↪the plants
```

```
    species     island  culmen_length_mm  culmen_depth_mm  flipper_length_mm  \
0    Adelie  Torgersen              39.1             18.7              181.0
1    Adelie  Torgersen              39.5             17.4              186.0
2    Adelie  Torgersen              40.3             18.0              195.0
3    Adelie  Torgersen               NaN              NaN                NaN
4    Adelie  Torgersen              36.7             19.3              193.0
..      ...        ...               ...              ...                ...
339  Gentoo     Biscoe               NaN              NaN                NaN
340  Gentoo     Biscoe              46.8             14.3              215.0
341  Gentoo     Biscoe              50.4             15.7              222.0
342  Gentoo     Biscoe              45.2             14.8              212.0
343  Gentoo     Biscoe              49.9             16.1              213.0
```

```
     body_mass_g     sex
0         3750.0    MALE
1         3800.0  FEMALE
2         3250.0  FEMALE
3            NaN     NaN
4         3450.0  FEMALE
..           …      …
339          NaN     NaN
340       4850.0  FEMALE
341       5750.0    MALE
342       5200.0  FEMALE
343       5400.0    MALE

[344 rows x 7 columns]
```

[ ]:
```
# Let's take a closer look at our data. What does it look like? Are there any␣
↪problematic data points?
data = data.dropna()
print(data)
```

```
    species     island  culmen_length_mm  culmen_depth_mm  flipper_length_mm  \
0    Adelie  Torgersen              39.1             18.7              181.0
1    Adelie  Torgersen              39.5             17.4              186.0
2    Adelie  Torgersen              40.3             18.0              195.0
4    Adelie  Torgersen              36.7             19.3              193.0
5    Adelie  Torgersen              39.3             20.6              190.0
..      …        …                  …                …                  …
338  Gentoo     Biscoe              47.2             13.7              214.0
340  Gentoo     Biscoe              46.8             14.3              215.0
341  Gentoo     Biscoe              50.4             15.7              222.0
342  Gentoo     Biscoe              45.2             14.8              212.0
343  Gentoo     Biscoe              49.9             16.1              213.0

     body_mass_g     sex
0         3750.0    MALE
1         3800.0  FEMALE
2         3250.0  FEMALE
4         3450.0  FEMALE
5         3650.0    MALE
..           …      …
338       4925.0  FEMALE
340       4850.0  FEMALE
341       5750.0    MALE
342       5200.0  FEMALE
343       5400.0    MALE

[334 rows x 7 columns]
```

[ ]:
```
# Let's make a plot of our data.
plot = px.scatter_matrix(data_frame = data, dimensions=["flipper_length_mm",␣
 ↪"body_mass_g", "culmen_length_mm"], color="species")
plot.show()
```

[ ]:
```
# We don't need all of the data, so let's choose one species.
data_gentoo = data[data["species"] == "Gentoo"]
df = data_gentoo[["flipper_length_mm", "body_mass_g"]]
df.head()
```

[ ]:

	flipper_length_mm	body_mass_g
220	211.0	4500.0
221	230.0	5700.0
222	210.0	4450.0
223	218.0	5700.0
224	215.0	5400.0

Now that we have a dataframe containing the flipper lengths and corresponding body mass of 342 penguins, let's try to predict the body mass based on the flipper length.

2 Splitting our Test and Train Data

Remember that one of the most important parts of machine learning is preventing overfitting and underfitting. Conveniently, scikit-learn can randomly split our data for us. The standard for splitting training and testing data is typically 80/20.

[ ]:
```
# First, we create NumPy arrays for our input and output
flip_length = np.array(df["flipper_length_mm"]).reshape(-1,1)
body_mass = np.array(df["body_mass_g"]).reshape(-1,1)

# Now, let's split our data
flip_train, flip_test, body_train, body_test = train_test_split(flip_length,␣
 ↪body_mass, test_size=0.20)
```

3 Linear Regression Model

Now, in a few simple lines of code, we can create a linear regression model.

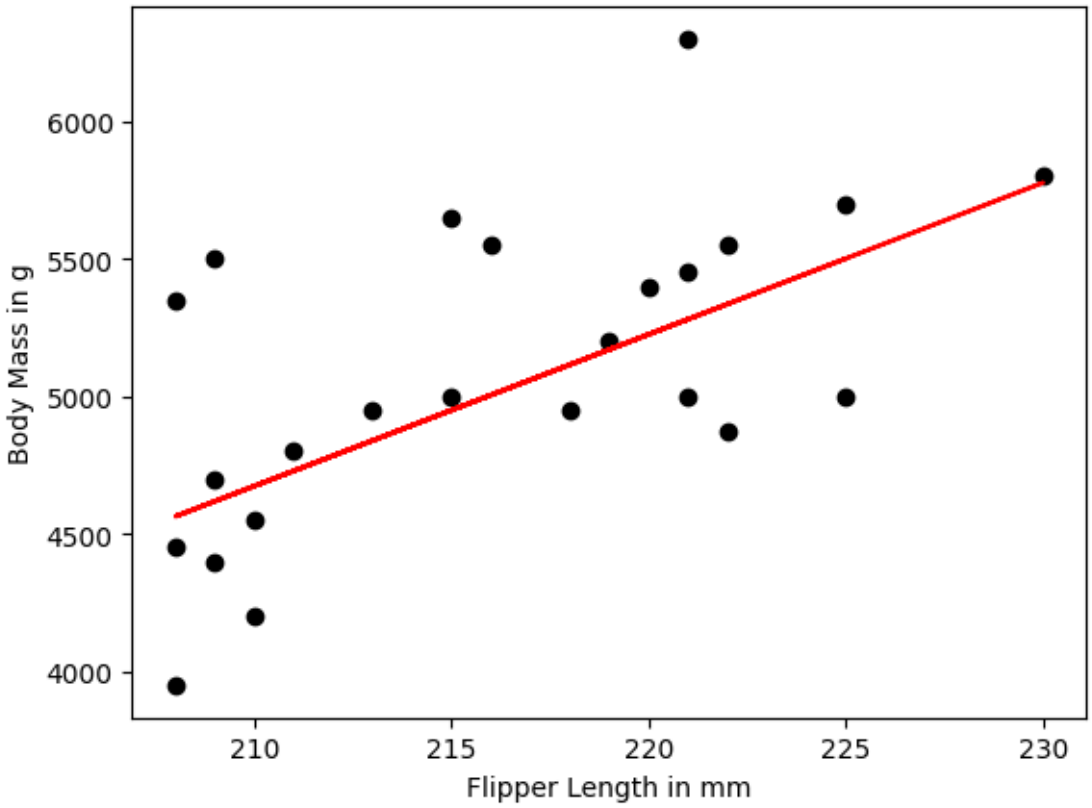
[ ]:
```
# We need to create a model and fit it to our data
linModel = LinearRegression()
linModel.fit(flip_train, body_train)
lin_predictions = linModel.predict(flip_test)
print(lin_predictions)
```

```
[[5005.13402796]
 [4563.92949379]
 [4619.08006056]
```

```
[5280.88686182]
[4729.3811941 ]
[4839.68232765]
[4949.98346119]
[5170.58572828]
[5280.88686182]
[4563.92949379]
[5501.48912891]
[5280.88686182]
[4619.08006056]
[4949.98346119]
[5336.0374286 ]
[5225.73629505]
[4674.23062733]
[5501.48912891]
[5777.24196277]
[5336.0374286 ]
[4563.92949379]
[4619.08006056]
[4674.23062733]
[5115.43516151]]
```

Let's create a graph to analyze the results of our model.

[ ]:
```
# Let's find a way to display the outputs
plt.scatter(flip_test, body_test, color = "black")
plt.plot(flip_test, lin_predictions, color = "red")
plt.xlabel("Flipper Length in mm")
plt.ylabel("Body Mass in g")
plt.show()
#This isn't a very good prediction, so let's try to change the amount of data␣
 ↪used for testing and training
```



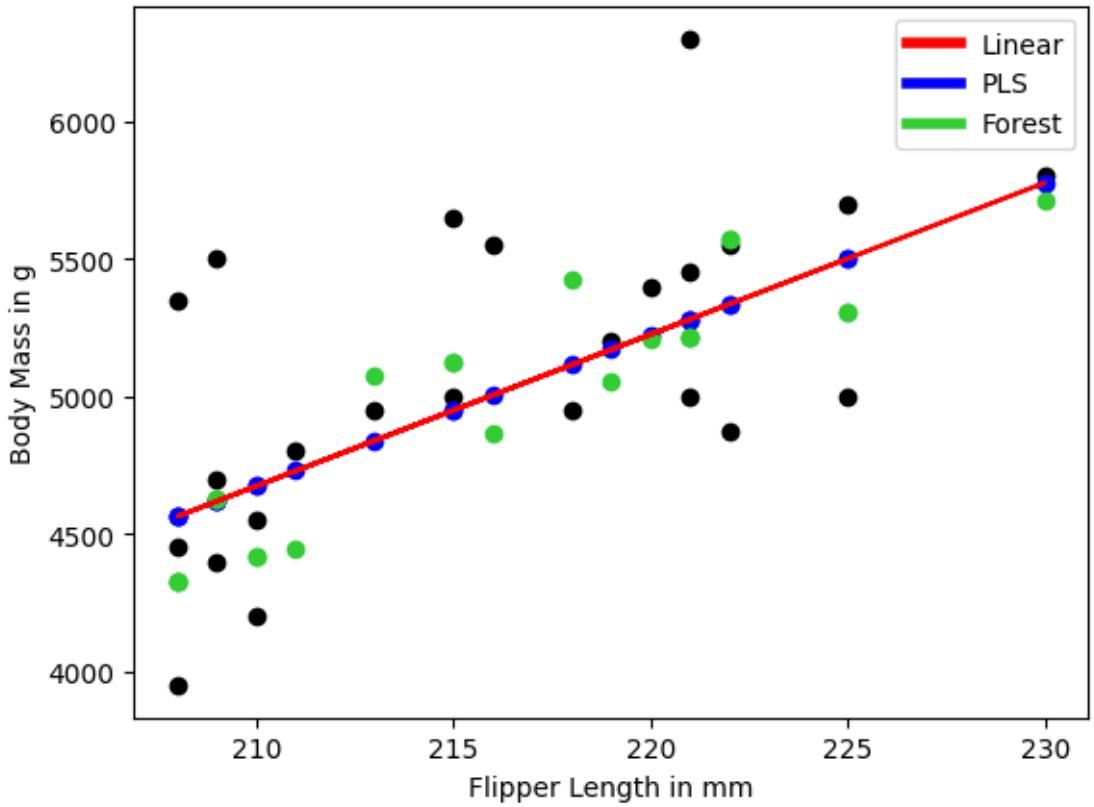
[ ]:
```
# Let's attempt to use another model to see if Linear Regression is truly␣
↪suitable for this task.
body_train = np.ravel(body_train)
reg_model = RandomForestRegressor(n_estimators=10, random_state=0)
reg_model.fit(flip_train, body_train)
reg_predictions = reg_model.predict(flip_test)

pls_model = PLSRegression(n_components=1)
pls_model.fit(flip_train, body_train)
pls_predictions = pls_model.predict(flip_test)
```

[ ]:
```
from matplotlib.lines import Line2D
custom_legend = [Line2D([0], [0], color="red", lw=4),
                 Line2D([0], [0], color="blue", lw=4),
                 Line2D([0], [0], color="limegreen", lw=4)]
```

[ ]:
```
fig, ax = plt.subplots()
ax.legend(custom_legend, ['Linear', 'PLS', 'Forest'])

plt.scatter(flip_test, body_test, color = "black")
plt.plot(flip_test, lin_predictions, color = "red")
```

```
plt.scatter(flip_test, pls_predictions, color="blue")
plt.scatter(flip_test, reg_predictions, color="limegreen")
plt.xlabel("Flipper Length in mm")
plt.ylabel("Body Mass in g")
```

[ ]: Text(0, 0.5, 'Body Mass in g')



4 Analyzing Our Output

Now, we have outputs from three different models. How can we determine which one is best?

[ ]:
```
lin_rmse = metrics.mean_squared_error(body_test, lin_predictions, squared=False)
pls_rmse = metrics.mean_squared_error(body_test, pls_predictions, squared=False)
reg_rmse = metrics.mean_squared_error(body_test, reg_predictions, squared=False)

# Goal is to have lowest root mean square error.
print(f'RMSE of Linear Regression: {lin_rmse}.')
print(f"RMSE of PLS Regression: {pls_rmse}.")
print(f"RMSE of Random Forest Regression: {reg_rmse}.")
```

```
RMSE of Linear Regression: 439.68219372373176.
RMSE of PLS Regression: 439.6821937237318.
RMSE of Random Forest Regression: 469.3992670072348.
```

5 Classification

This is a simple classification model trained live during the workshop on our penguin data.

[ ]:
```python
from sklearn.cluster import KMeans

features = ["flipper_length_mm", "culmen_length_mm"]

X_data = data[features]
Y_data = data["species"]

X_train, X_test, Y_train, Y_test = train_test_split(X_data, Y_data, test_size=0.
↪3)

X_train_n = preprocessing.normalize(X_train)
X_test_n = preprocessing.normalize(X_test)

cluster_model = KMeans(n_clusters=3, n_init='auto')
cluster_model.fit(X_train)

string_lbls = [str(x) for x in cluster_model.labels_]

px.scatter(X_train, "culmen_length_mm", "flipper_length_mm", color=string_lbls)
```

[ ]:
```python
px.scatter(data, "culmen_length_mm", "flipper_length_mm", color="species")
```

Interactive Data Visualization (Aditya Kumar)

Learning Objectives

1. Gain knowledge about efficient visualizations for different kinds of data.
2. Create interactive visualizations of numerical data.
3. Learn to visualize geospatial data on interactive map visualizations.

PhytoOracleVisWorkshop

April 12, 2024

1 Data Visualisation

Hello, I am Aditya Kumar. I am senior at the University of Arizona, and today I will be introducing you to interactive data visualization in Python using Plotly. (https://plotly.com/python/)

1.1 Motivation

Why do we need visuals?

- They make data readable,
- Understandable, and
- Interesting!

What makes a good visualization? * Informative * Fun to look at * Easy to recreate * Appropriate * and most importantly, accurate!

Here's a quick example: https://images.app.goo.gl/EiNsCGpDhrs1Knbd7

Why do we want to make visuals with Plotly? * It is quick * Visuals are easy to create and interactive * Less code to write * Fun to create!

Here's the simplest way to make a bargraph with just more or less vanilla python

```
# Doesn't work for google cloud
import tkinter as tk

class BarGraph(tk.Tk):
    def __init__(self, data, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.title("Simple Bar Graph")

        # Calculate maximum value for scaling
        max_value = max(data.values())

        # Create canvas for drawing
        canvas_width = 400
        canvas_height = 300
        self.canvas = tk.Canvas(self, width=canvas_width, height=canvas_height)
        self.canvas.pack()

        # Draw bars
```

```
        bar_width = 50
        spacing = 20
        x = 50
        for label, value in data.items():
            bar_height = (value / max_value) * canvas_height
            self.canvas.create_rectangle(x, canvas_height - bar_height, x + bar_width, canvas_
            self.canvas.create_text(x + bar_width / 2, canvas_height - bar_height - 10, text=la
            self.canvas.create_text(x + bar_width / 2, canvas_height + 10, text=value)
            x += bar_width + spacing

if __name__ == "__main__":
    # Example data
    data = {
        "A": 100,
        "B": 200,
        "C": 150,
        "D": 300
    }

    # Create and run the application
    app = BarGraph(data)
    app.mainloop()
```

1.2 Basic Visualisation Techniques

For this session, we plan to use the following libraries. Click on the tiny button in the next code block with the play icon on it to import all the dependencies.

[ ]:
```
import plotly.express as px
import pandas as pd
```

Now we will get our dataset. For the first section of the workshop, we are going to go with the thermal dataset that has been generated by our lab.

[ ]:
```
data = pd.read_csv('https://data.cyverse.org/dav-anon/iplant/projects/
 ↪phytooracle/season_14_sorghum_yr_2022/season14_2022/flirIrCamera/
 ↪environmental_association/2022-07-19_environmental_association.csv')
data.head(10)
```

[ ]:

	year	field	range	row	plot	species	experiment	treatment	\
0	2022	north	55	1	5501	sorghum bicolor	NaN	WL	
1	2022	north	55	1	5501	sorghum bicolor	NaN	WL	
2	2022	north	55	1	5501	sorghum bicolor	NaN	WL	
3	2022	north	55	1	5501	sorghum bicolor	NaN	WL	
4	2022	north	55	1	5501	sorghum bicolor	NaN	WL	
5	2022	north	55	1	5501	sorghum bicolor	NaN	WL	
6	2022	north	55	3	5503	sorghum bicolor	NaN	WL	

```
7  2022  north     55    3  5503  sorghum bicolor         NaN        WL
8  2022  north     55    3  5503  sorghum bicolor         NaN        WL
9  2022  north     55    3  5503  sorghum bicolor         NaN        WL

     type  rep  …      time sunDirection  airPressure  relHumidity  \
0  border  NaN  …  14:39:55        360.0  1009.288614    25.287637
1  border  NaN  …  14:39:55        360.0  1009.288614    25.287637
2  border  NaN  …  14:42:33        360.0  1009.241005    26.367992
3  border  NaN  …  14:42:33        360.0  1009.241005    26.367992
4  border  NaN  …  14:42:33        360.0  1009.241005    26.367992
5  border  NaN  …  14:42:33        360.0  1009.241005    26.367992
6  border  NaN  …  14:58:54        360.0  1008.923612    25.476852
7  border  NaN  …  14:58:54        360.0  1008.923612    25.476852
8  border  NaN  …  14:58:54        360.0  1008.923612    25.476852
9  border  NaN  …  14:58:54        360.0  1008.923612    25.476852

   temperature  windDirection  precipitation  windVelocity          par  \
0    41.011383     256.791284          100.0      6.269723  1508.973263
1    41.011383     256.791284          100.0      6.269723  1508.973263
2    40.446791     268.887600          100.0      8.650166  1430.838725
3    40.446791     268.887600          100.0      8.650166  1430.838725
4    40.446791     268.887600          100.0      8.650166  1430.838725
5    40.446791     268.887600          100.0      8.650166  1430.838725
6    40.932035     247.485580          100.0      9.886166  1602.246368
7    40.932035     247.485580          100.0      9.886166  1602.246368
8    40.932035     247.485580          100.0      9.886166  1602.246368
9    40.932035     247.485580          100.0      9.886166  1602.246368

  canopy_temperature_depression
0                     -8.879169
1                     -8.649902
2                    -13.432901
3                    -12.490755
4                    -11.494569
5                     -8.818969
6                     -4.717371
7                     -4.312933
8                     -4.128672
9                     -4.858011

[10 rows x 29 columns]
```

[ ]:
```
list(data.columns)
```

[ ]:
```
['year',
 'field',
 'range',
```

```
 'row',
 'plot',
 'species',
 'experiment',
 'treatment',
 'type',
 'rep',
 'accession',
 'date',
 'roi_temp',
 'quartile_1',
 'mean',
 'median',
 'quartile_3',
 'variance',
 'std_dev',
 'time',
 'sunDirection',
 'airPressure',
 'relHumidity',
 'temperature',
 'windDirection',
 'precipitation',
 'windVelocity',
 'par',
 'canopy_temperature_depression']
```

First, we go with the basics.

Bar Charts

```
[ ]: fig = px.bar(data, x='type', title='Bar Chart - Type')
     fig.update_layout(plot_bgcolor='white')
     fig.show()
```

Violin Plots

```
[ ]: fig = px.violin(data, x='treatment', y='roi_temp', box = True, title='Violin␣
      ↪Plot -  ROI Temperature Distribution')
     fig.show()
```

Scatter Plots

```
[ ]: fig = px.scatter(data, x='windVelocity', y='relHumidity', color =␣
      ↪'temperature', title='Scatter Plot - Wind Velocity vs Relative Humidity')
     fig.show()
```

SPLOM (Scatter Plot Matrices)

[ ]:
```
# gets rid of the need for a for loop
fig = px.scatter_matrix(data, dimensions=[
 'mean',
 'relHumidity',
 'temperature',
 'median',
 'windVelocity'
], color='type', title='Pair Plot -  Thermal Dataset')
fig.show()
```

1.3 Plant Data Visualisation

1.3.1 3D plant data visualisation

Open3D is a tool for 3D data. (Thank you, Captain Obvious) (https://www.open3d.org/docs/release/introduction.html)

[ ]:
```
!python3 -m pip install open3d

import open3d as o3d
```

```
Collecting open3d
  Downloading open3d-0.18.0-cp310-cp310-manylinux_2_27_x86_64.whl (399.7 MB)
                             399.7/399.7

MB 1.5 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.18.0 in
/usr/local/lib/python3.10/dist-packages (from open3d) (1.25.2)
Collecting dash>=2.6.0 (from open3d)
  Downloading dash-2.16.1-py3-none-any.whl (10.2 MB)
                             10.2/10.2 MB
30.8 MB/s eta 0:00:00
Requirement already satisfied: werkzeug>=2.2.3 in
/usr/local/lib/python3.10/dist-packages (from open3d) (3.0.2)
Requirement already satisfied: nbformat>=5.7.0 in
/usr/local/lib/python3.10/dist-packages (from open3d) (5.10.3)
Collecting configargparse (from open3d)
  Downloading ConfigArgParse-1.7-py3-none-any.whl (25 kB)
Collecting ipywidgets>=8.0.4 (from open3d)
  Downloading ipywidgets-8.1.2-py3-none-any.whl (139 kB)
                             139.4/139.4

kB 18.2 MB/s eta 0:00:00
Collecting addict (from open3d)
  Downloading addict-2.4.0-py3-none-any.whl (3.8 kB)
Requirement already satisfied: pillow>=9.3.0 in /usr/local/lib/python3.10/dist-
packages (from open3d) (9.4.0)
Requirement already satisfied: matplotlib>=3 in /usr/local/lib/python3.10/dist-
packages (from open3d) (3.7.1)
```

```
Requirement already satisfied: pandas>=1.0 in /usr/local/lib/python3.10/dist-
packages (from open3d) (2.0.3)
Requirement already satisfied: pyyaml>=5.4.1 in /usr/local/lib/python3.10/dist-
packages (from open3d) (6.0.1)
Requirement already satisfied: scikit-learn>=0.21 in
/usr/local/lib/python3.10/dist-packages (from open3d) (1.2.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages
(from open3d) (4.66.2)
Collecting pyquaternion (from open3d)
  Downloading pyquaternion-0.9.9-py3-none-any.whl (14 kB)
Requirement already satisfied: Flask<3.1,>=1.0.4 in
/usr/local/lib/python3.10/dist-packages (from dash>=2.6.0->open3d) (2.2.5)
Requirement already satisfied: plotly>=5.0.0 in /usr/local/lib/python3.10/dist-
packages (from dash>=2.6.0->open3d) (5.15.0)
Collecting dash-html-components==2.0.0 (from dash>=2.6.0->open3d)
  Downloading dash_html_components-2.0.0-py3-none-any.whl (4.1 kB)
Collecting dash-core-components==2.0.0 (from dash>=2.6.0->open3d)
  Downloading dash_core_components-2.0.0-py3-none-any.whl (3.8 kB)
Collecting dash-table==5.0.0 (from dash>=2.6.0->open3d)
  Downloading dash_table-5.0.0-py3-none-any.whl (3.9 kB)
Requirement already satisfied: importlib-metadata in
/usr/local/lib/python3.10/dist-packages (from dash>=2.6.0->open3d) (7.1.0)
Requirement already satisfied: typing-extensions>=4.1.1 in
/usr/local/lib/python3.10/dist-packages (from dash>=2.6.0->open3d) (4.10.0)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-
packages (from dash>=2.6.0->open3d) (2.31.0)
Collecting retrying (from dash>=2.6.0->open3d)
  Downloading retrying-1.3.4-py3-none-any.whl (11 kB)
Requirement already satisfied: nest-asyncio in /usr/local/lib/python3.10/dist-
packages (from dash>=2.6.0->open3d) (1.6.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-
packages (from dash>=2.6.0->open3d) (67.7.2)
Collecting comm>=0.1.3 (from ipywidgets>=8.0.4->open3d)
  Downloading comm-0.2.2-py3-none-any.whl (7.2 kB)
Requirement already satisfied: ipython>=6.1.0 in /usr/local/lib/python3.10/dist-
packages (from ipywidgets>=8.0.4->open3d) (7.34.0)
Requirement already satisfied: traitlets>=4.3.1 in
/usr/local/lib/python3.10/dist-packages (from ipywidgets>=8.0.4->open3d) (5.7.1)
Collecting widgetsnbextension~=4.0.10 (from ipywidgets>=8.0.4->open3d)
  Downloading widgetsnbextension-4.0.10-py3-none-any.whl (2.3 MB)
                           2.3/2.3 MB
62.5 MB/s eta 0:00:00
Requirement already satisfied: jupyterlab-widgets~=3.0.10 in
/usr/local/lib/python3.10/dist-packages (from ipywidgets>=8.0.4->open3d)
(3.0.10)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3->open3d) (1.2.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-
```

```
packages (from matplotlib>=3->open3d) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3->open3d) (4.50.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3->open3d) (1.4.5)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3->open3d) (24.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3->open3d) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=3->open3d) (2.8.2)
Requirement already satisfied: fastjsonschema in /usr/local/lib/python3.10/dist-
packages (from nbformat>=5.7.0->open3d) (2.19.1)
Requirement already satisfied: jsonschema>=2.6 in
/usr/local/lib/python3.10/dist-packages (from nbformat>=5.7.0->open3d) (4.19.2)
Requirement already satisfied: jupyter-core in /usr/local/lib/python3.10/dist-
packages (from nbformat>=5.7.0->open3d) (5.7.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas>=1.0->open3d) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-
packages (from pandas>=1.0->open3d) (2024.1)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-
packages (from scikit-learn>=0.21->open3d) (1.11.4)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-
packages (from scikit-learn>=0.21->open3d) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.21->open3d)
(3.4.0)
Requirement already satisfied: MarkupSafe>=2.1.1 in
/usr/local/lib/python3.10/dist-packages (from werkzeug>=2.2.3->open3d) (2.1.5)
Requirement already satisfied: Jinja2>=3.0 in /usr/local/lib/python3.10/dist-
packages (from Flask<3.1,>=1.0.4->dash>=2.6.0->open3d) (3.1.3)
Requirement already satisfied: itsdangerous>=2.0 in
/usr/local/lib/python3.10/dist-packages (from
Flask<3.1,>=1.0.4->dash>=2.6.0->open3d) (2.1.2)
Requirement already satisfied: click>=8.0 in /usr/local/lib/python3.10/dist-
packages (from Flask<3.1,>=1.0.4->dash>=2.6.0->open3d) (8.1.7)
Collecting jedi>=0.16 (from ipython>=6.1.0->ipywidgets>=8.0.4->open3d)
  Downloading jedi-0.19.1-py2.py3-none-any.whl (1.6 MB)
                           1.6/1.6 MB
60.3 MB/s eta 0:00:00
Requirement already satisfied: decorator in
/usr/local/lib/python3.10/dist-packages (from
ipython>=6.1.0->ipywidgets>=8.0.4->open3d) (4.4.2)
Requirement already satisfied: pickleshare in /usr/local/lib/python3.10/dist-
packages (from ipython>=6.1.0->ipywidgets>=8.0.4->open3d) (0.7.5)
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from
```

```
ipython>=6.1.0->ipywidgets>=8.0.4->open3d) (3.0.43)
Requirement already satisfied: pygments in /usr/local/lib/python3.10/dist-
packages (from ipython>=6.1.0->ipywidgets>=8.0.4->open3d) (2.16.1)
Requirement already satisfied: backcall in /usr/local/lib/python3.10/dist-
packages (from ipython>=6.1.0->ipywidgets>=8.0.4->open3d) (0.2.0)
Requirement already satisfied: matplotlib-inline in
/usr/local/lib/python3.10/dist-packages (from
ipython>=6.1.0->ipywidgets>=8.0.4->open3d) (0.1.6)
Requirement already satisfied: pexpect>4.3 in /usr/local/lib/python3.10/dist-
packages (from ipython>=6.1.0->ipywidgets>=8.0.4->open3d) (4.9.0)
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.10/dist-
packages (from jsonschema>=2.6->nbformat>=5.7.0->open3d) (23.2.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in
/usr/local/lib/python3.10/dist-packages (from
jsonschema>=2.6->nbformat>=5.7.0->open3d) (2023.12.1)
Requirement already satisfied: referencing>=0.28.4 in
/usr/local/lib/python3.10/dist-packages (from
jsonschema>=2.6->nbformat>=5.7.0->open3d) (0.34.0)
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.10/dist-
packages (from jsonschema>=2.6->nbformat>=5.7.0->open3d) (0.18.0)
Requirement already satisfied: tenacity>=6.2.0 in
/usr/local/lib/python3.10/dist-packages (from
plotly>=5.0.0->dash>=2.6.0->open3d) (8.2.3)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-
packages (from python-dateutil>=2.7->matplotlib>=3->open3d) (1.16.0)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.10/dist-
packages (from importlib-metadata->dash>=2.6.0->open3d) (3.18.1)
Requirement already satisfied: platformdirs>=2.5 in
/usr/local/lib/python3.10/dist-packages (from jupyter-
core->nbformat>=5.7.0->open3d) (4.2.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests->dash>=2.6.0->open3d)
(3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests->dash>=2.6.0->open3d) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests->dash>=2.6.0->open3d)
(2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests->dash>=2.6.0->open3d)
(2024.2.2)
Requirement already satisfied: parso<0.9.0,>=0.8.3 in
/usr/local/lib/python3.10/dist-packages (from
jedi>=0.16->ipython>=6.1.0->ipywidgets>=8.0.4->open3d) (0.8.3)
Requirement already satisfied: ptyprocess>=0.5 in
/usr/local/lib/python3.10/dist-packages (from
pexpect>4.3->ipython>=6.1.0->ipywidgets>=8.0.4->open3d) (0.7.0)
Requirement already satisfied: wcwidth in /usr/local/lib/python3.10/dist-
```

```
packages (from prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0->ipython>=6.1.0-
>ipywidgets>=8.0.4->open3d) (0.2.13)
Installing collected packages: dash-table, dash-html-components, dash-core-
components, addict, widgetsnbextension, retrying, pyquaternion, jedi,
configargparse, comm, ipywidgets, dash, open3d
  Attempting uninstall: widgetsnbextension
    Found existing installation: widgetsnbextension 3.6.6
    Uninstalling widgetsnbextension-3.6.6:
      Successfully uninstalled widgetsnbextension-3.6.6
  Attempting uninstall: ipywidgets
    Found existing installation: ipywidgets 7.7.1
    Uninstalling ipywidgets-7.7.1:
      Successfully uninstalled ipywidgets-7.7.1
Successfully installed addict-2.4.0 comm-0.2.2 configargparse-1.7 dash-2.16.1
dash-core-components-2.0.0 dash-html-components-2.0.0 dash-table-5.0.0
ipywidgets-8.1.2 jedi-0.19.1 open3d-0.18.0 pyquaternion-0.9.9 retrying-1.3.4
widgetsnbextension-4.0.10
```

[ ]:
```python
import requests # for downloading stuff from the internet
import numpy as np
```

Load the data from an online source and use open3D to read it in. Voxel downsampling is a technique used to reduce the number of points in a point cloud by dividing the 3D space into small voxels and representing all the points within each voxel by their centroid.

[ ]:
```python
# URL to the point cloud file
url = "https://data.cyverse.org/dav-anon/iplant/commons/community_released/
 ↪phytooracle/TempFolder_bhuppenthal_SPN_Outputs/2022-06-08/
 ↪BTx623_7401_330773026341/combined_multiway_registered.ply"
response = requests.get(url)

file_path = "SAMPLE_NAME.PLY"
with open(file_path, 'wb') as f:
    f.write(response.content)

pcd = o3d.io.read_point_cloud(file_path)
pcd = pcd.voxel_down_sample(voxel_size=0.002)
```

Center and modify the data to make it fit for visualisation

[ ]:
```python
points = np.asarray(pcd.points)
print(points[:10])

x_offset = np.mean(points[:, 0])
y_offset = np.mean(points[:, 1])

# to make data manageable/usable
xs = points[:, 0] - x_offset
```

```
ys = points[:, 1] - y_offset
zs = points[:, 2]

df_dict = {"x": xs, "y": ys, "z": zs}
df = pd.DataFrame(df_dict)
df.size
```

```
[[4.08992542e+05 3.66027929e+06 7.16000946e-01]
 [4.08992743e+05 3.66027978e+06 1.02505309e+00]
 [4.08992700e+05 3.66027956e+06 7.28060120e-01]
 [4.08992876e+05 3.66027963e+06 7.16073669e-01]
 [4.08992544e+05 3.66027935e+06 7.16105530e-01]
 [4.08992616e+05 3.66027939e+06 8.21440735e-01]
 [4.08992634e+05 3.66027943e+06 8.47185364e-01]
 [4.08992617e+05 3.66027946e+06 8.19393433e-01]
 [4.08992755e+05 3.66027960e+06 9.69531372e-01]
 [4.08992910e+05 3.66027963e+06 1.01007013e+00]]
```

[ ]:
```
1101696
```

Visualise it

[ ]:
```
color_scale = [[0.0, "sandybrown"], [0.5, "limegreen"], [1.0, "green"]]
fig = px.scatter_3d(
  df,
  title="Sorghum",
  x="x",
  y="y",
  z="z",
  color="z",
  color_continuous_scale=color_scale,
)

fig.update_traces(marker=dict(size=1.5))
fig.show()
```

1.3.2 Scatter Map Plots for Visualising Geospatial data

Download data

[ ]:
```
url = "https://data.cyverse.org/dav-anon/iplant/commons/community_released/
 ↪phytooracle/dashboard_cache/scanner3DTop/combined_data/
 ↪season_15_lettuce_yr_2022_2023-01-09__19-33-08-066_lettuce_all.csv"
response = requests.get(url)

file_path = "sampledata.csv"
with open(file_path, 'wb') as f:
    f.write(response.content)
```

```
df = pd.read_csv("sampledata.csv")
df.head(10)
```

[ ]:

	date	pred_conf	plot	genotype	lon	\
0	2023-01-09__10-30-14-158_lettuce	0.467930	2639	Emperor	-111.974845	
1	2023-01-09__10-30-14-158_lettuce	0.443987	2639	Emperor	-111.974845	
2	2023-01-09__10-30-14-158_lettuce	0.440178	2639	Emperor	-111.974844	
3	2023-01-09__10-30-14-158_lettuce	0.427877	2639	Emperor	-111.974845	
4	2023-01-09__10-30-14-158_lettuce	0.419393	2639	Emperor	-111.974845	
5	2023-01-09__10-30-14-158_lettuce	0.377047	2639	Emperor	-111.974844	
6	2023-01-09__10-30-14-158_lettuce	0.357165	2639	Emperor	-111.974845	
7	2023-01-09__10-30-14-158_lettuce	0.339829	2639	Emperor	-111.974845	
8	2023-01-09__10-30-14-158_lettuce	0.297797	2639	Emperor	-111.974845	
9	2023-01-09__10-30-14-158_lettuce	0.292994	2639	Emperor	-111.974845	

	lat	nw_lat	nw_lon	se_lat	se_lon	…	\
0	33.075454	33.075453	-111.974845	33.075455	-111.974844	…	
1	33.075441	33.075441	-111.974845	33.075442	-111.974844	…	
2	33.075446	33.075446	-111.974845	33.075447	-111.974844	…	
3	33.075458	33.075457	-111.974845	33.075459	-111.974844	…	
4	33.075457	33.075456	-111.974845	33.075457	-111.974844	…	
5	33.075443	33.075443	-111.974845	33.075443	-111.974844	…	
6	33.075430	33.075430	-111.974845	33.075431	-111.974844	…	
7	33.075432	33.075431	-111.974845	33.075432	-111.974844	…	
8	33.075435	33.075435	-111.974845	33.075435	-111.974844	…	
9	33.075433	33.075432	-111.974845	33.075433	-111.974844	…	

	amplitude_betti_2	amplitude_silhouette_0	amplitude_silhouette_1	\
0	0.0	0.004483	0.000000	
1	0.0	0.003375	0.003009	
2	0.0	0.002682	0.000000	
3	0.0	0.003619	0.000000	
4	0.0	0.002682	0.000000	
5	0.0	0.000000	0.004510	
6	0.0	0.002682	0.000000	
7	0.0	0.003619	0.000000	
8	0.0	0.004510	0.000000	
9	0.0	0.002682	0.000000	

	amplitude_silhouette_2	amplitude_heat_0	amplitude_heat_1	\
0	0.0	1.519042	0.000000	
1	0.0	0.934998	0.314436	
2	0.0	0.053287	0.000000	
3	0.0	0.242755	0.000000	
4	0.0	0.106574	0.000000	
5	0.0	0.000000	0.139391	

```
6                    0.0          0.053287          0.000000
7                    0.0          0.242755          0.000000
8                    0.0          0.139391          0.000000
9                    0.0          0.053287          0.000000

   amplitude_heat_2  amplitude_persistence_image_0  \
0               0.0                       0.000000
1               0.0                       4.066118
2               0.0                       0.000000
3               0.0                       0.000000
4               0.0                       0.000000
5               0.0                       0.000000
6               0.0                       0.000000
7               0.0                       0.000000
8               0.0                       0.000000
9               0.0                       0.000000

  amplitude_persistence_image_1  amplitude_persistence_image_2
0                      0.000000                            0.0
1                      2.437894                            0.0
2                      0.000000                            0.0
3                      0.000000                            0.0
4                      0.000000                            0.0
5                      0.000000                            0.0
6                      0.000000                            0.0
7                      0.000000                            0.0
8                      0.000000                            0.0
9                      0.000000                            0.0

[10 rows x 50 columns]
```

MapBox Generation

[ ]:
```
from google.colab import userdata

px.set_mapbox_access_token(userdata.get("your_mapbox_api_key"))
fig = px.scatter_mapbox(
            df,
            lat="lat",
            lon="lon",
            color="genotype",
            zoom=16.6,
            opacity=1,
            mapbox_style="satellite",
            hover_data=["lat", "lon", "genotype", "plant_name"],
        )
fig.show()
```

1.4 Appendix

Matplotlib offers extensive customization but demands more code, Seaborn simplifies statistical plots with built-in themes, and Plotly excels at creating dynamic and interactive visualizations

1. Additional Reference Materials:

- https://youtu.be/GGL6U0k8WYA?si=8QZA4dr8f9HVK3Ll
- https://plotly.com/python/

2. Some interesting alternatives to Plotly

- Seaborn: https://seaborn.pydata.org/
- Matplotlib: https://matplotlib.org/stable/plot_types/index.html

Session 3: Applying Computational Toolkit to Plant Phenotyping

Session 3 Introduction

Scientific Computing & Data Analytics: A Comprehensive Toolkit for Research

Emmanuel Miguel Gonzalez, Jeffrey Demieville, Brenda Huppenthal, Emily Cawley, Aditya Kumar, Bella Salter, Duke Pauli

Award No. 2102120

Pauli Lab

About the Presenters

Emmanuel Gonzalez

Brenda Huppenthal

Jeffrey Demieville

Emily Cawley

Bella Salter

Aditya Kumar

Pauli Lab

THE UNIVERSITY OF ARIZONA

PhytoOracle

Building Your Computational Toolkit

Session 1:

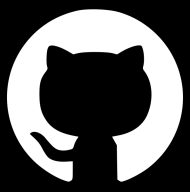
- Fundamental Computational Toolkit

Session 2:

- Machine Learning & Data Visualization

Session 3:

- Applying Computational Toolkit to Plant Phenotyping



Building Your Computational Toolkit

Session 1:

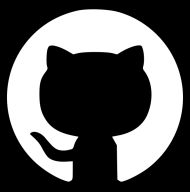
- Fundamental Computational Toolkit

Session 2:

- Machine Learning & Data Visualization

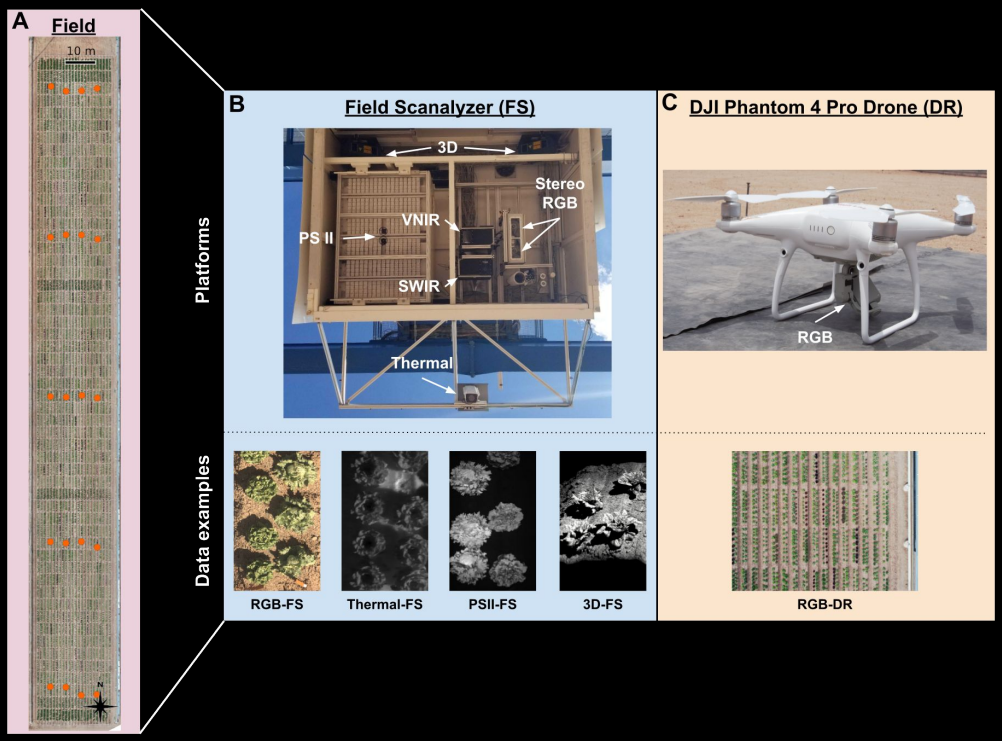
Session 3:

- Applying Computational Toolkit to Plant Phenotyping

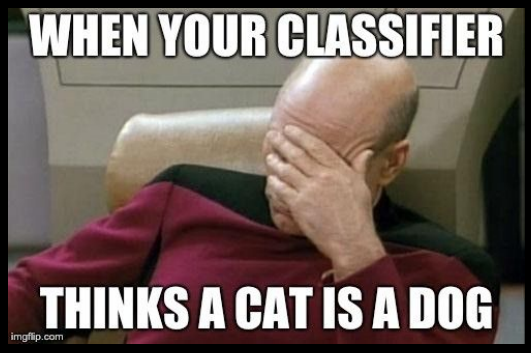


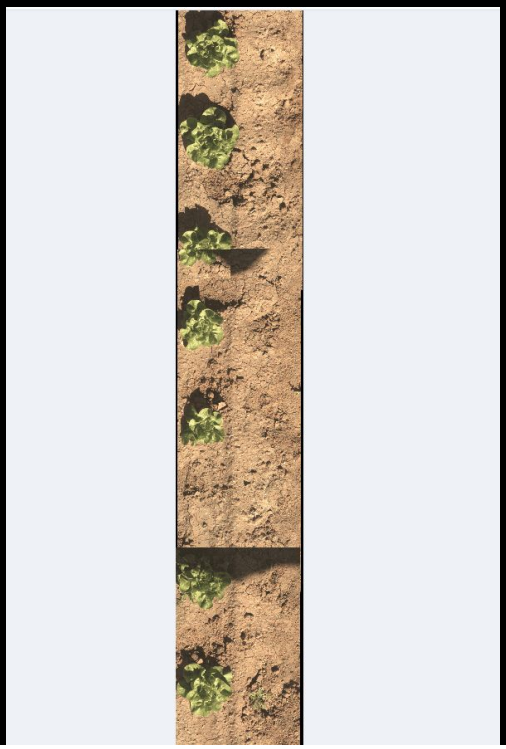

Object Detection in RGB Images

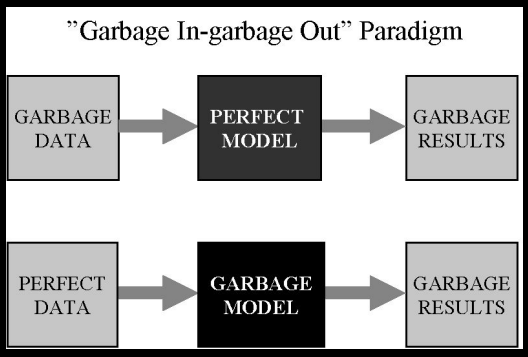
Data Collection is Essential in Machine Learning



Labeling: A Crucial Step With Many Common Pitfalls

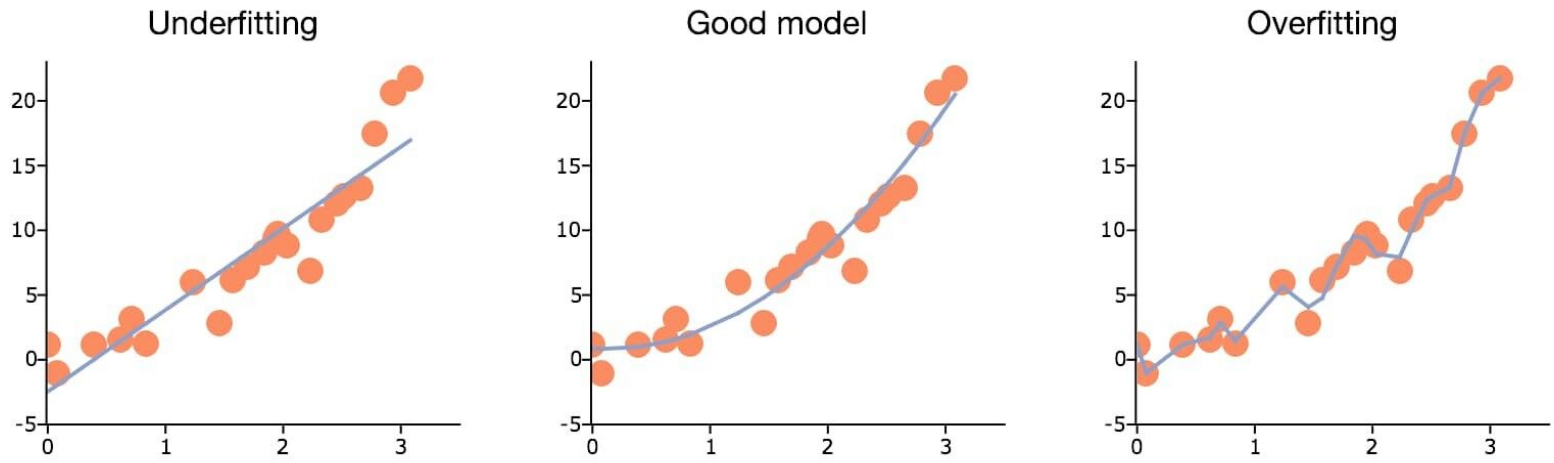






Visualizing Model Outputs

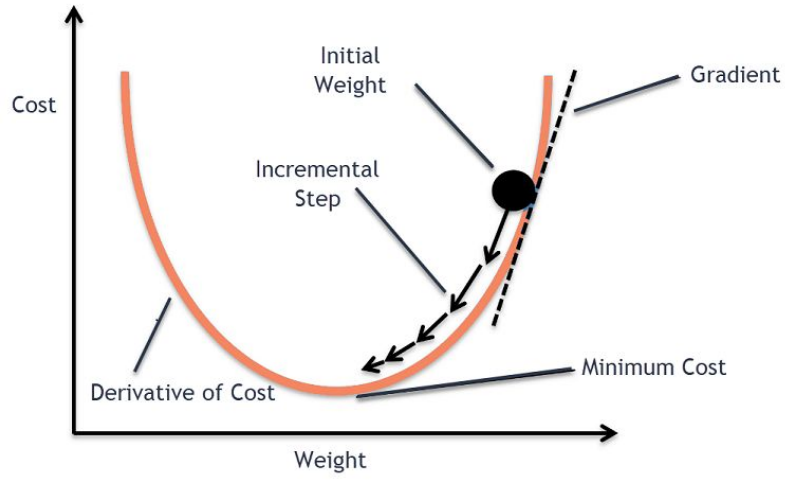
Investigating Model Performance

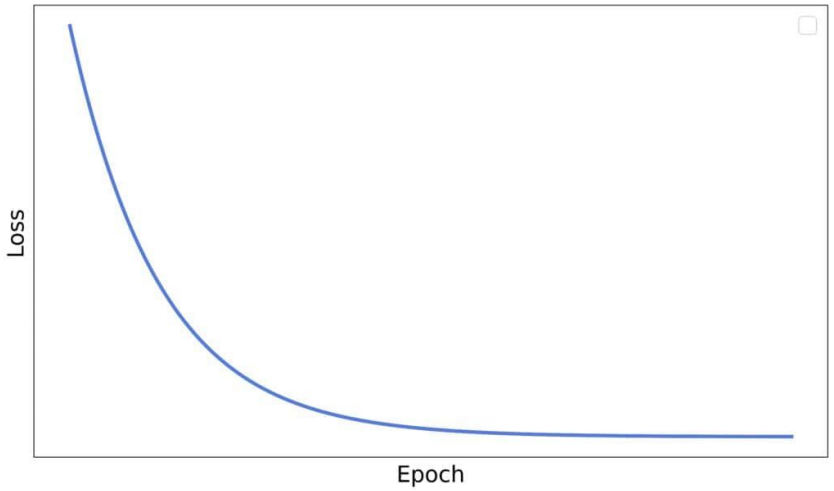


Underfitting is when a model fails to capture data patterns, leading to poor performance.
Overfitting is when a model learns data noise, resulting in poor performance on new data.

Image Credit: APTECH

Investigating Model Performance

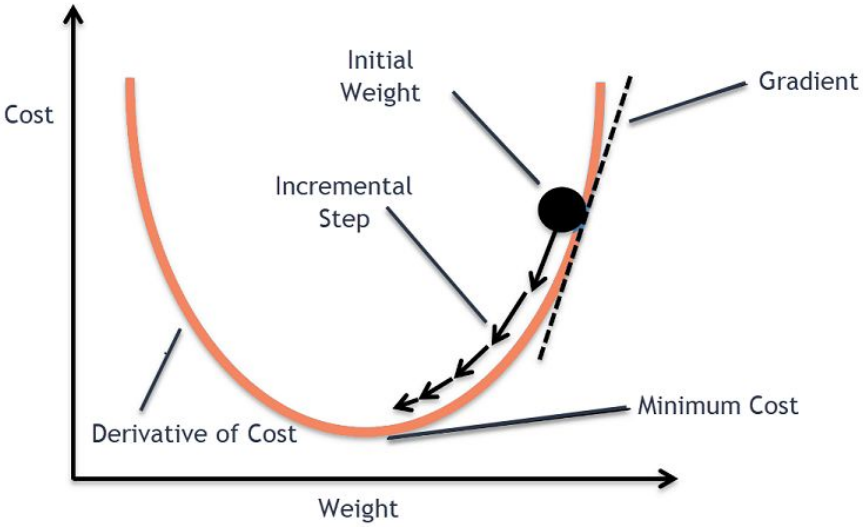


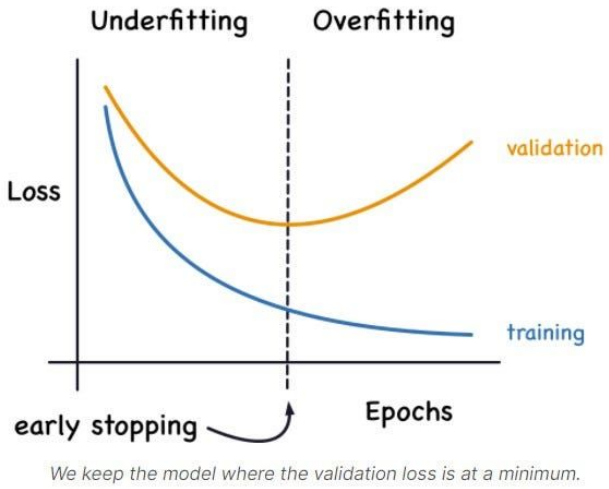


Loss is the error of our model and we use gradient descent to minimize this error. The model's parameters are iteratively adjusted in the direction that reduces the loss the most.

Image Credit: Rhnyewale

Investigating Model Performance



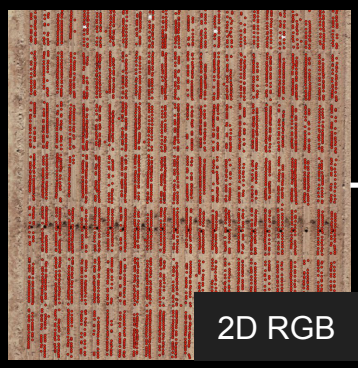


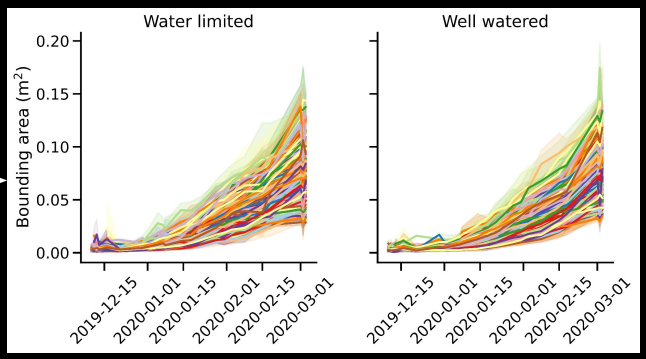
We keep the model where the validation loss is at a minimum.

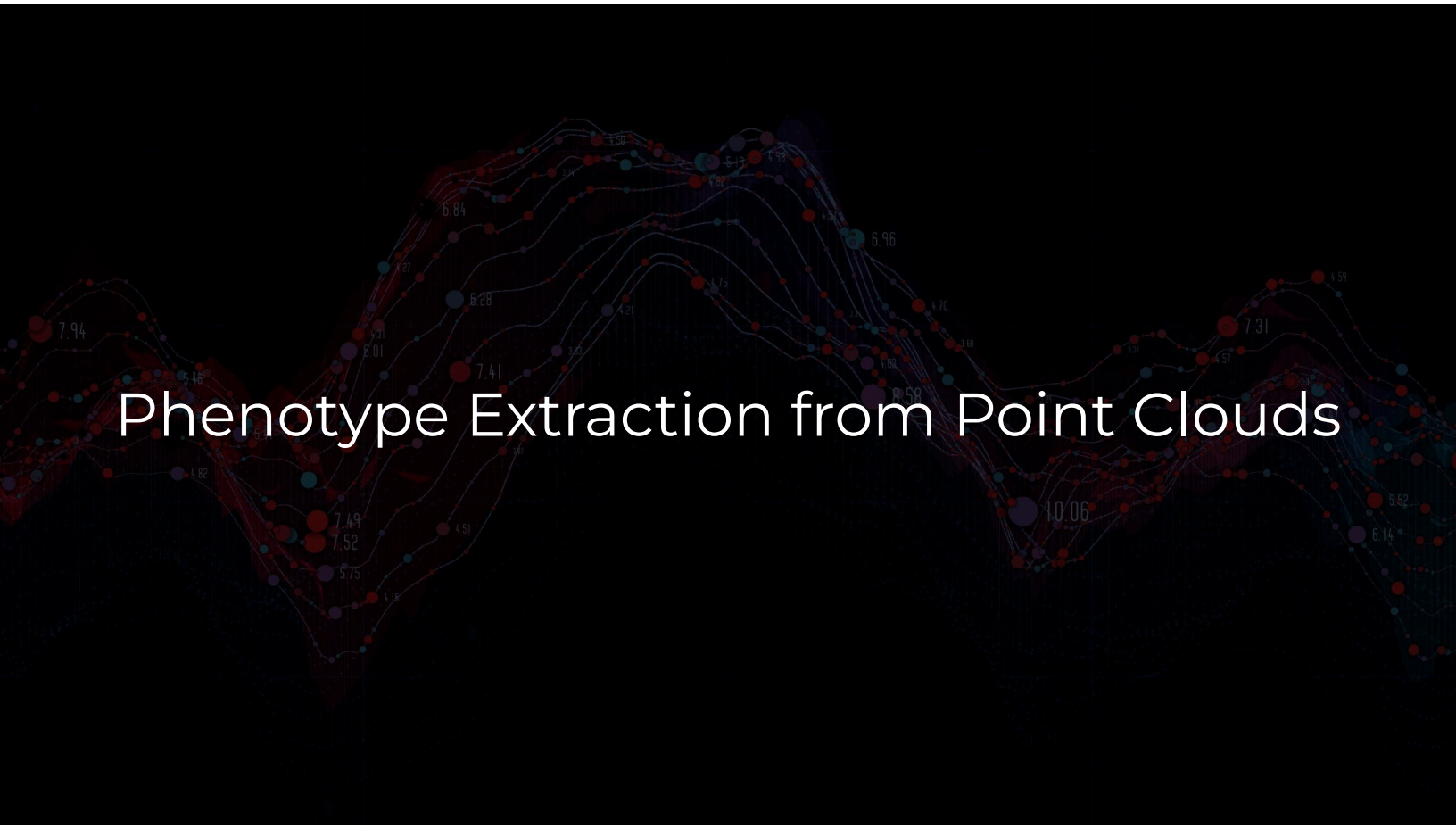
The goal is to reach the lowest point of a valley by consistently moving in a downward direction. However, if we go too far, it could lead to a model that is either overfitted or underfitted.

Image Credit: Analytics Vidhya & Rhnyewale

Leveraging Your Model for Plant Phenotyping

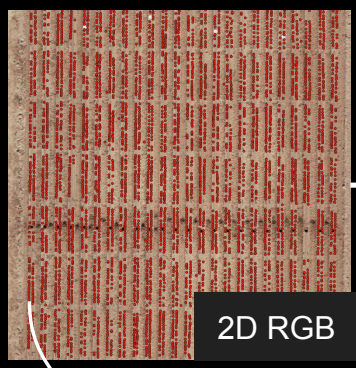


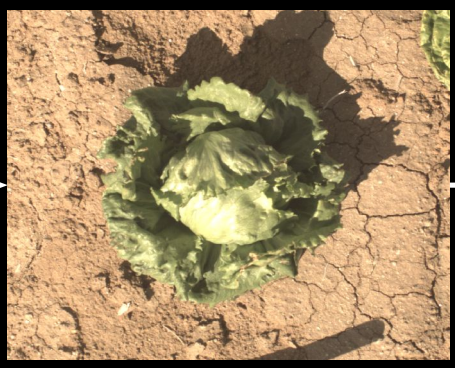


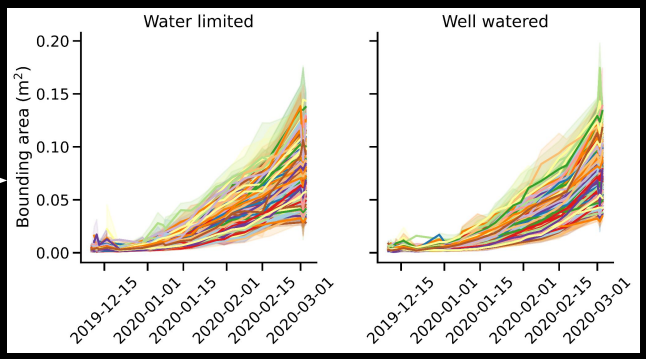


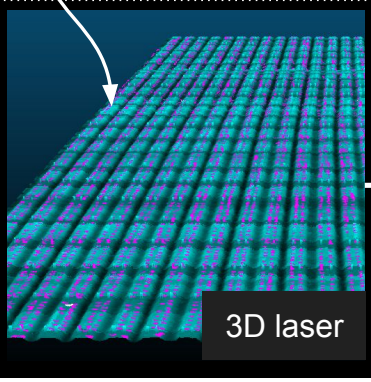
Phenotype Extraction from Point Clouds

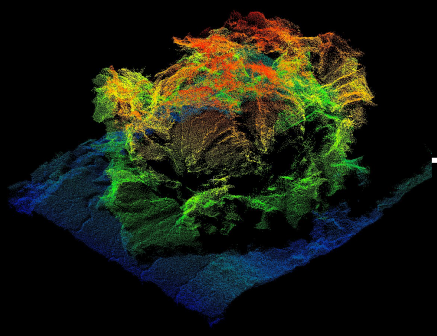
Large, complex datasets are generated

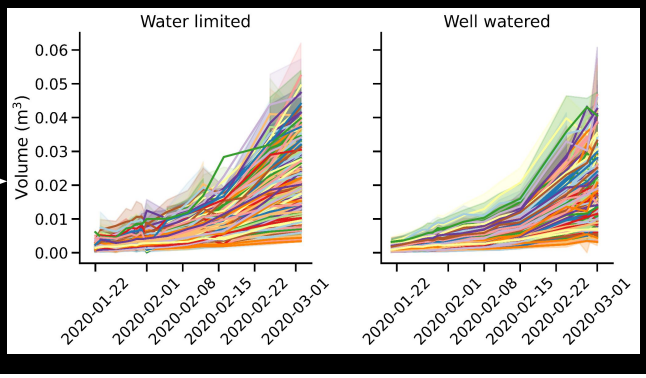




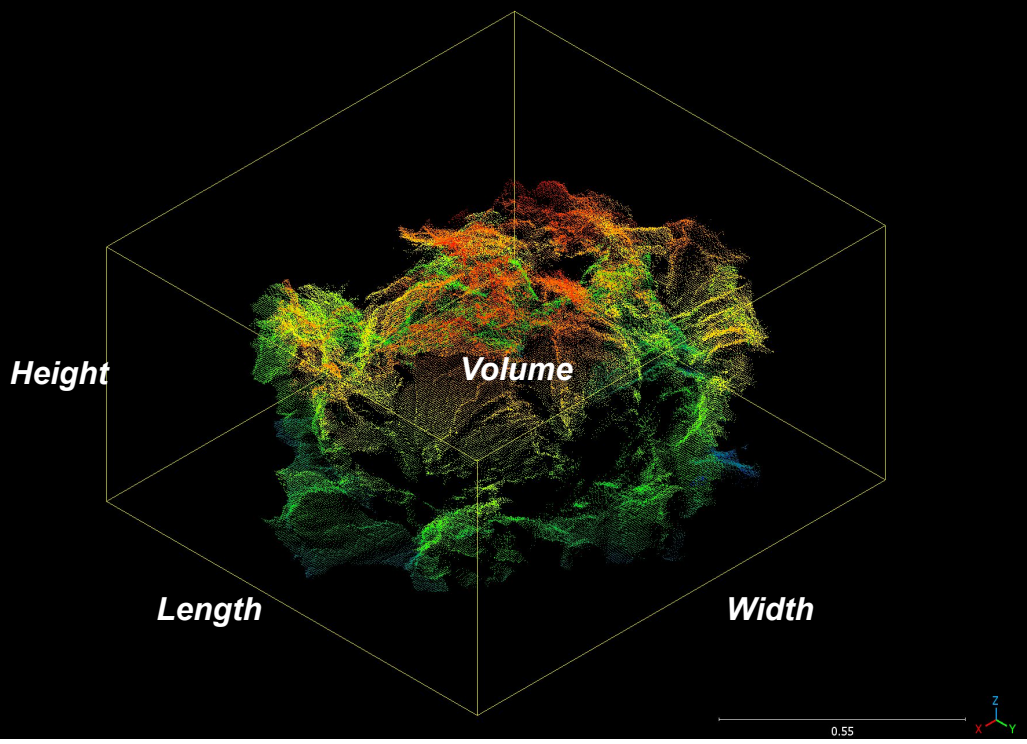






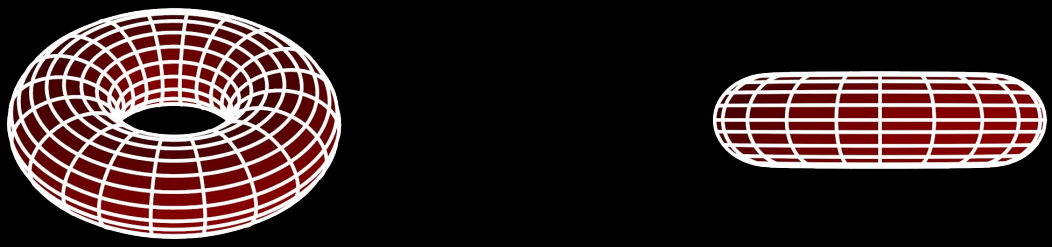


Traditional Shape Descriptors Are Useful

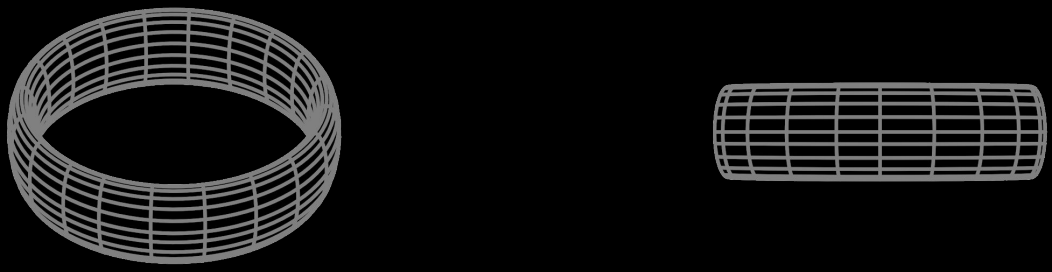


Nuances May Be Missed by Traditional Descriptors

Shape 1

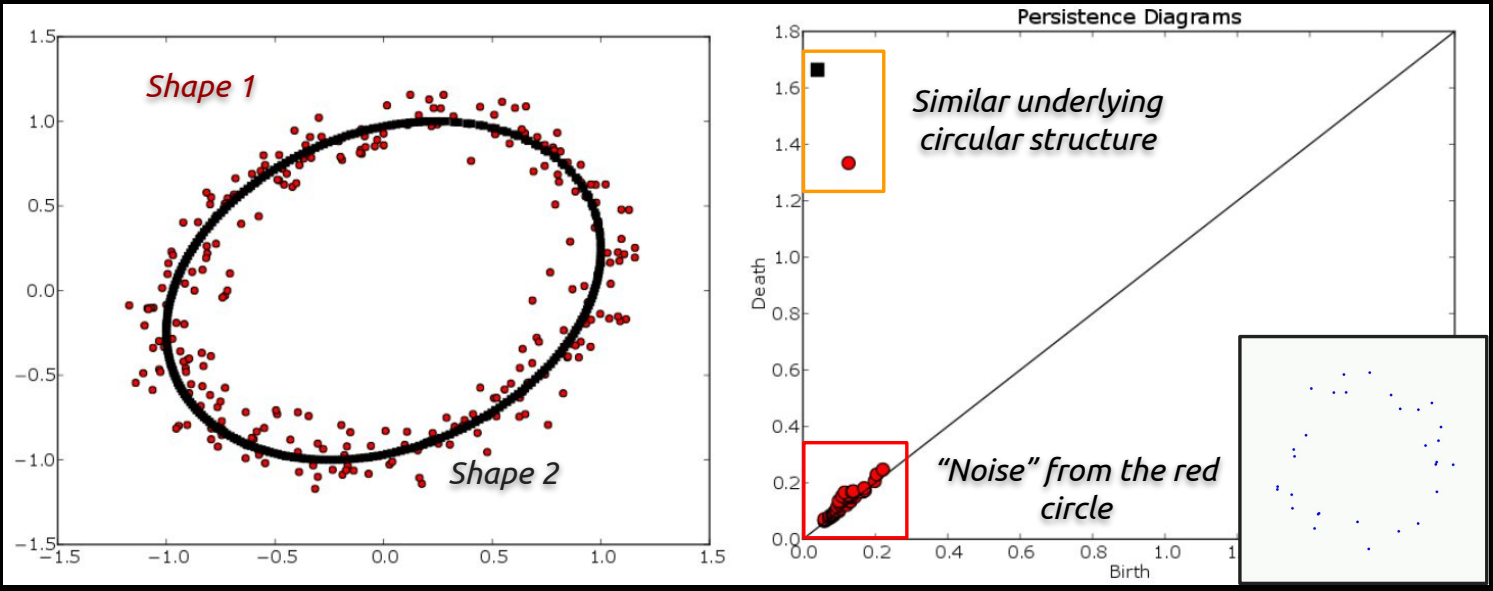


Shape 2



Same shape?

Persistence Diagram



Adapted from: A User's Guide to Topological Data Analysis (Munch 2017)

Persistence Diagram

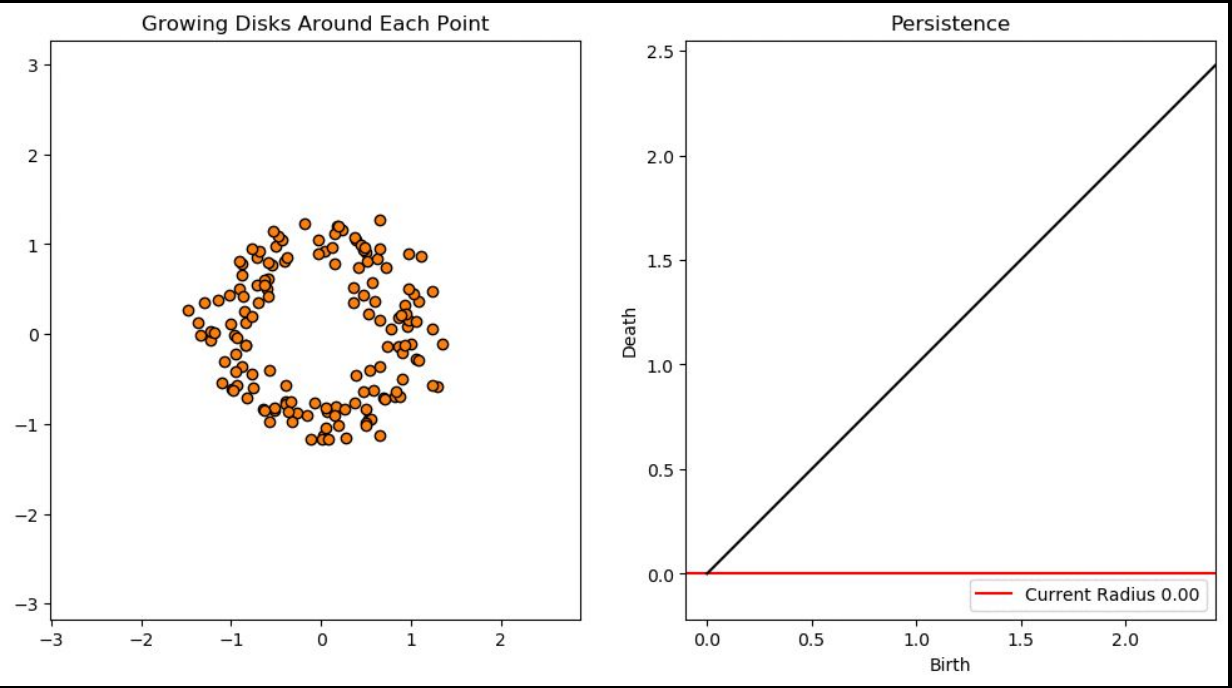
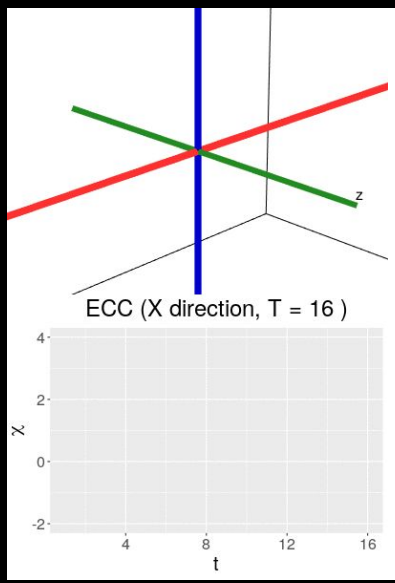


Image Credit: Gary Koplik

Euler Characteristic Curve

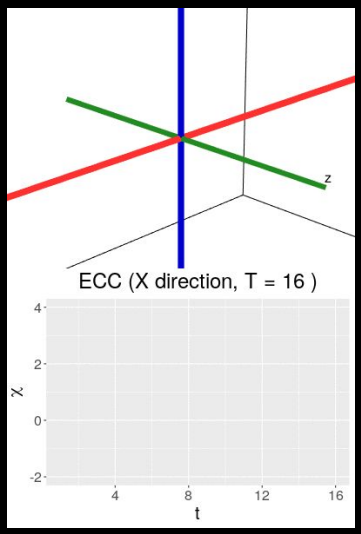


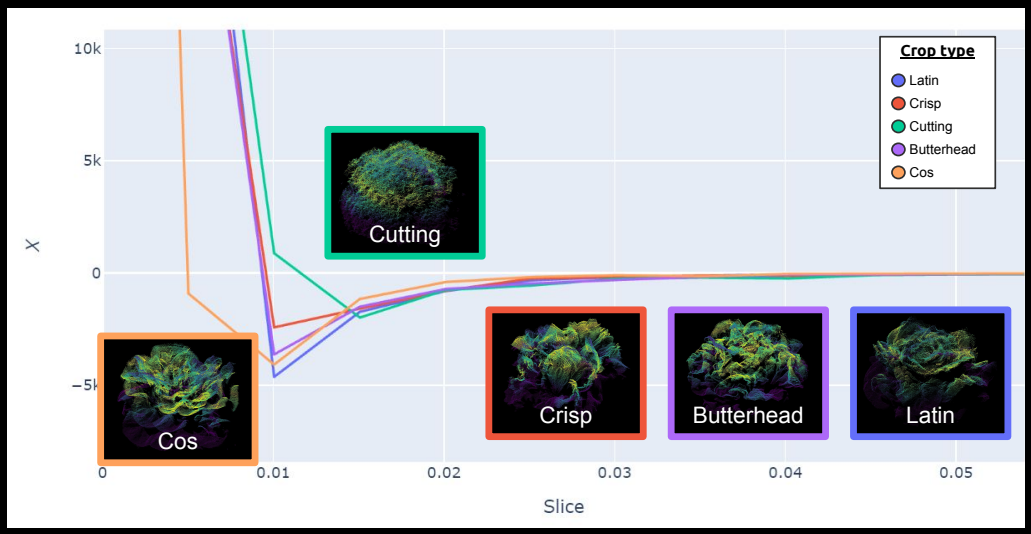
Name	Image	Vertices *V*	Edges *E*	Faces *F*	Euler characteristic: *V* − *E* + *F*
Tetrahedron		4	6	4	**2**
Hexahedron or cube		8	12	6	**2**
Octahedron		6	12	8	**2**
Dodecahedron		20	30	12	**2**
Icosahedron		12	30	20	**2**

Euler characteristic ($\mathscr{X}$) = Vertices – Edges + Faces

Adapted from: Measuring hidden phenotype: quantifying the shape of barley seeds using the Euler characteristic transform (Amezquita et al. 2022)

Euler Characteristic Curve



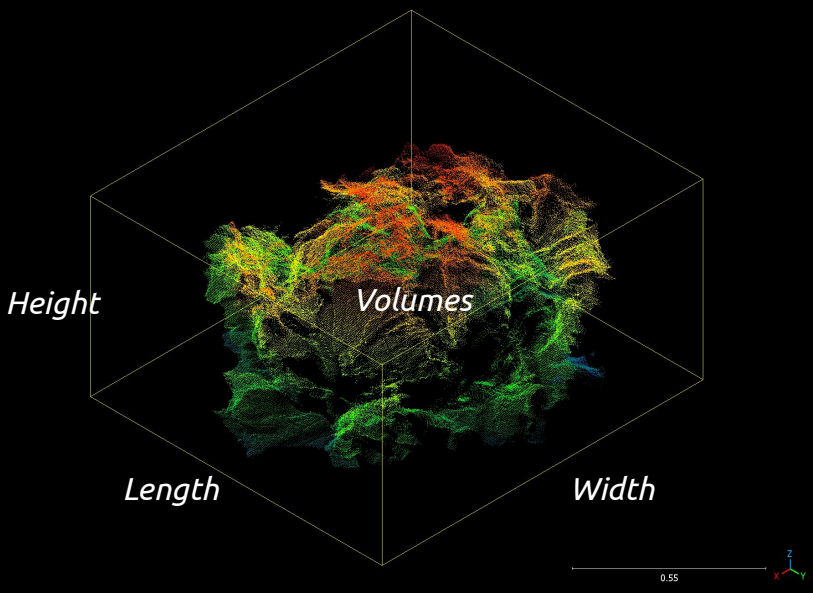


Euler characteristic ($\mathscr{X}$) = #(Connected Components) – #(Loops) + #(Voids)

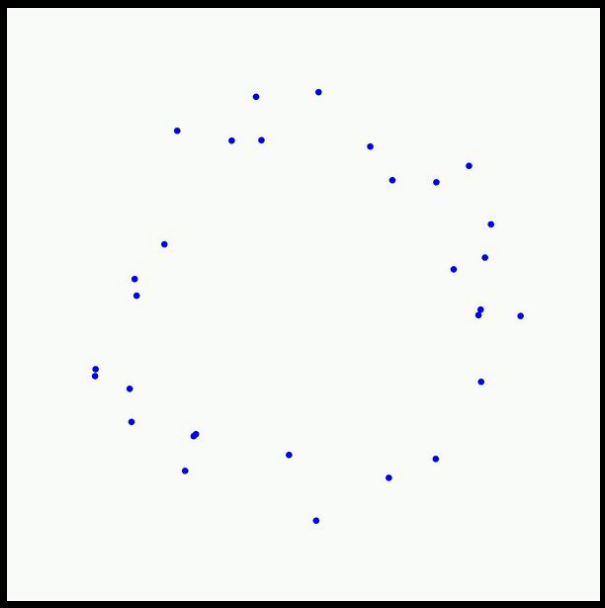
Adapted from: Measuring hidden phenotype: quantifying the shape of barley seeds using the Euler characteristic transform (Amezquita et al. 2022)

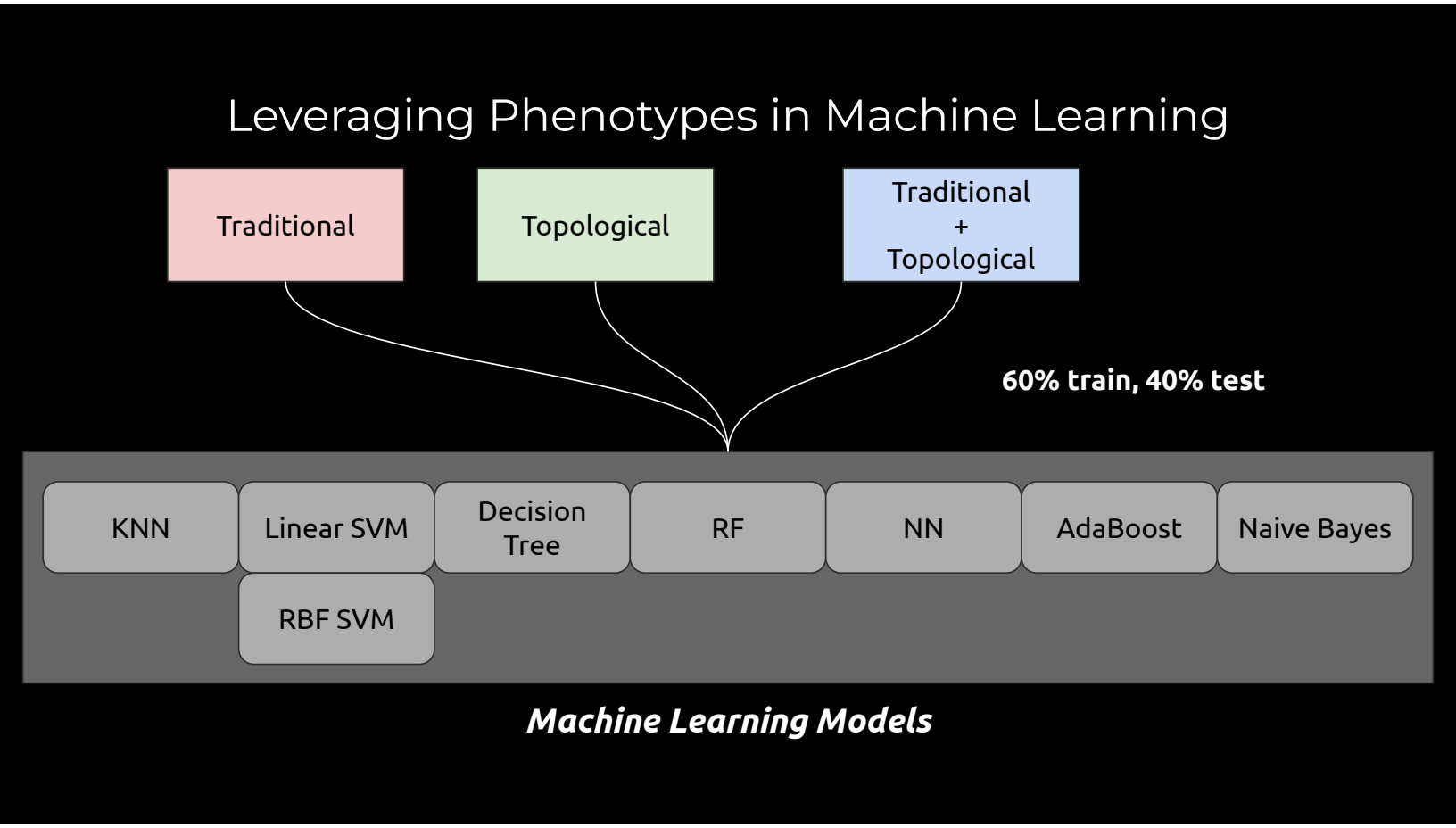
Combining Traditional and Topological Shape Descriptors

Traditional



Topological




Leveraging Phenotypes in Machine Learning
Traditional
Topological
Traditional + Topological
60% train, 40% test
KNN
Linear SVM
Decision Tree
RF
NN
AdaBoost
Naive Bayes
RBF SVM
Machine Learning Models

Access to Workshop Materials

Contains workshop overview, learning objectives, and code

bit.ly/AG2PI_SciComp

Pauli Lab

Scientific Computing & Data Analytics*

A Comprehensive Toolkit for Research

Emmanuel Miguel Gonzalez, Jeffrey Demieville, Brenda Huppenthal,
Emily Cawley, Aditya Kumar, Bella Salter, Duke Pauli

2024-04-02

*The PhytoOracle project is support
Research (DE-SC0020401) and Advance
dation Plant Genome Research Progran
(2019674 and 1743442), and CyVerse pro
Department of Agriculture National Inst
We thank Agricultural Genome to Phen

Pauli Lab

Table of Contents

Computational Phenotype Extraction with Machine Learning - RGB: object detection & color analysis (Brenda Huppenthal)

Learning Objectives

1. Acquire skills to download datasets from remote sources, decompress them if necessary, and utilize scikit-learn to segregate the data into training, validation, and testing sets.
2. Gain experience in viewing annotated training images and outputs from the Detecto model.
3. Understand the fundamental aspects of image data format and acquire the ability to convert between pixel and geographical coordinates within your datasets.
4. Master the process of loading, training, and applying inference with object detection models using Detecto.
5. Cultivate a basic knowledge of machine learning to explore other Python libraries that offer computer vision models.
6. Learn to use popular libraries like matplotlib and seaborn to visualize accuracy and loss on training, validation, and testing datasets.

object_detection_with_deteco

April 12, 2024

**

Object Detection for Plant Phenotyping: A Hands-On Workshop

**

Workflow: - Use Labelbox to annotate an image dataset. - Download the annotated images and prepare them for training the model. - Train an object detection model using detecto. - Evaluate the model performance. - Detect plants in drone images. - With these bounding boxes, we can perform further calculations on specific plants, such as the Triangular Greenness Index.

[ ]:
```
# Install Python libraries
!python3 -m pip install detecto
!python3 -m pip install labelbox[data]
!python3 -m pip install pascal-voc-writer
!python3 -m pip install utm
!python3 -m pip install rasterio
!python3 -m pip install geopandas
```

```
Collecting detecto
  Downloading detecto-1.2.2-py3-none-any.whl (25 kB)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-
packages (from detecto) (3.7.1)
Requirement already satisfied: opencv-python in /usr/local/lib/python3.10/dist-
packages (from detecto) (4.8.0.76)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages
(from detecto) (2.0.3)
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages
(from detecto) (2.2.1+cu121)
Requirement already satisfied: torchvision in /usr/local/lib/python3.10/dist-
packages (from detecto) (0.17.1+cu121)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages
(from detecto) (4.66.2)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->detecto) (1.2.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-
packages (from matplotlib->detecto) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->detecto) (4.51.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
```

```
/usr/local/lib/python3.10/dist-packages (from matplotlib->detecto) (1.4.5)
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.10/dist-
packages (from matplotlib->detecto) (1.25.2)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->detecto) (24.0)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-
packages (from matplotlib->detecto) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->detecto) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->detecto) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas->detecto) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-
packages (from pandas->detecto) (2024.1)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-
packages (from torch->detecto) (3.13.4)
Requirement already satisfied: typing-extensions>=4.8.0 in
/usr/local/lib/python3.10/dist-packages (from torch->detecto) (4.11.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages
(from torch->detecto) (1.12)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-
packages (from torch->detecto) (3.3)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages
(from torch->detecto) (3.1.3)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages
(from torch->detecto) (2023.6.0)
Collecting nvidia-cuda-nvrtc-cu12==12.1.105 (from torch->detecto)
  Using cached nvidia_cuda_nvrtc_cu12-12.1.105-py3-none-manylinux1_x86_64.whl
(23.7 MB)
Collecting nvidia-cuda-runtime-cu12==12.1.105 (from torch->detecto)
  Using cached nvidia_cuda_runtime_cu12-12.1.105-py3-none-manylinux1_x86_64.whl
(823 kB)
Collecting nvidia-cuda-cupti-cu12==12.1.105 (from torch->detecto)
  Using cached nvidia_cuda_cupti_cu12-12.1.105-py3-none-manylinux1_x86_64.whl
(14.1 MB)
Collecting nvidia-cudnn-cu12==8.9.2.26 (from torch->detecto)
  Using cached nvidia_cudnn_cu12-8.9.2.26-py3-none-manylinux1_x86_64.whl (731.7
MB)
Collecting nvidia-cublas-cu12==12.1.3.1 (from torch->detecto)
  Using cached nvidia_cublas_cu12-12.1.3.1-py3-none-manylinux1_x86_64.whl (410.6
MB)
Collecting nvidia-cufft-cu12==11.0.2.54 (from torch->detecto)
  Using cached nvidia_cufft_cu12-11.0.2.54-py3-none-manylinux1_x86_64.whl (121.6
MB)
Collecting nvidia-curand-cu12==10.3.2.106 (from torch->detecto)
  Using cached nvidia_curand_cu12-10.3.2.106-py3-none-manylinux1_x86_64.whl
(56.5 MB)
```

```
Collecting nvidia-cusolver-cu12==11.4.5.107 (from torch->detecto)
  Using cached nvidia_cusolver_cu12-11.4.5.107-py3-none-manylinux1_x86_64.whl
(124.2 MB)
Collecting nvidia-cusparse-cu12==12.1.0.106 (from torch->detecto)
  Using cached nvidia_cusparse_cu12-12.1.0.106-py3-none-manylinux1_x86_64.whl
(196.0 MB)
Collecting nvidia-nccl-cu12==2.19.3 (from torch->detecto)
  Using cached nvidia_nccl_cu12-2.19.3-py3-none-manylinux1_x86_64.whl (166.0 MB)
Collecting nvidia-nvtx-cu12==12.1.105 (from torch->detecto)
  Using cached nvidia_nvtx_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (99 kB)
Requirement already satisfied: triton==2.2.0 in /usr/local/lib/python3.10/dist-
packages (from torch->detecto) (2.2.0)
Collecting nvidia-nvjitlink-cu12 (from nvidia-cusolver-
cu12==11.4.5.107->torch->detecto)
  Using cached nvidia_nvjitlink_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl
(21.1 MB)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-
packages (from python-dateutil>=2.7->matplotlib->detecto) (1.16.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from jinja2->torch->detecto) (2.1.5)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-
packages (from sympy->torch->detecto) (1.3.0)
Installing collected packages: nvidia-nvtx-cu12, nvidia-nvjitlink-cu12, nvidia-
nccl-cu12, nvidia-curand-cu12, nvidia-cufft-cu12, nvidia-cuda-runtime-cu12,
nvidia-cuda-nvrtc-cu12, nvidia-cuda-cupti-cu12, nvidia-cublas-cu12, nvidia-
cusparse-cu12, nvidia-cudnn-cu12, nvidia-cusolver-cu12, detecto
Successfully installed detecto-1.2.2 nvidia-cublas-cu12-12.1.3.1 nvidia-cuda-
cupti-cu12-12.1.105 nvidia-cuda-nvrtc-cu12-12.1.105 nvidia-cuda-runtime-
cu12-12.1.105 nvidia-cudnn-cu12-8.9.2.26 nvidia-cufft-cu12-11.0.2.54 nvidia-
curand-cu12-10.3.2.106 nvidia-cusolver-cu12-11.4.5.107 nvidia-cusparse-
cu12-12.1.0.106 nvidia-nccl-cu12-2.19.3 nvidia-nvjitlink-cu12-12.4.127 nvidia-
nvtx-cu12-12.1.105
Collecting labelbox[data]
  Downloading labelbox-3.67.0-py3-none-any.whl (238 kB)
                             238.4/238.4
kB 6.4 MB/s eta 0:00:00
Requirement already satisfied: requests>=2.22.0 in
/usr/local/lib/python3.10/dist-packages (from labelbox[data]) (2.31.0)
Requirement already satisfied: google-api-core>=1.22.1 in
/usr/local/lib/python3.10/dist-packages (from labelbox[data]) (2.11.1)
Requirement already satisfied: pydantic>=1.8 in /usr/local/lib/python3.10/dist-
packages (from labelbox[data]) (2.6.4)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages
(from labelbox[data]) (4.66.2)
Requirement already satisfied: python-dateutil<2.9.0,>=2.8.2 in
/usr/local/lib/python3.10/dist-packages (from labelbox[data]) (2.8.2)
Collecting strenum>=0.4.15 (from labelbox[data])
```

```
  Downloading StrEnum-0.4.15-py3-none-any.whl (8.9 kB)
Requirement already satisfied: shapely in /usr/local/lib/python3.10/dist-
packages (from labelbox[data]) (2.0.3)
Collecting geojson (from labelbox[data])
  Downloading geojson-3.1.0-py3-none-any.whl (15 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages
(from labelbox[data]) (1.25.2)
Requirement already satisfied: PILLOW in /usr/local/lib/python3.10/dist-packages
(from labelbox[data]) (9.4.0)
Requirement already satisfied: opencv-python in /usr/local/lib/python3.10/dist-
packages (from labelbox[data]) (4.8.0.76)
Collecting typeguard (from labelbox[data])
  Downloading typeguard-4.2.1-py3-none-any.whl (34 kB)
Requirement already satisfied: imagesize in /usr/local/lib/python3.10/dist-
packages (from labelbox[data]) (1.4.1)
Requirement already satisfied: pyproj in /usr/local/lib/python3.10/dist-packages
(from labelbox[data]) (3.6.1)
Collecting pygeotile (from labelbox[data])
  Downloading pyGeoTile-1.0.6.tar.gz (3.8 kB)
  Preparing metadata (setup.py) … done
Requirement already satisfied: typing-extensions in
/usr/local/lib/python3.10/dist-packages (from labelbox[data]) (4.11.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-
packages (from labelbox[data]) (24.0)
Requirement already satisfied: googleapis-common-protos<2.0.dev0,>=1.56.2 in
/usr/local/lib/python3.10/dist-packages (from google-api-
core>=1.22.1->labelbox[data]) (1.63.0)
Requirement already satisfied: protobuf!=3.20.0,!=3.20.1,!=4.21.0,!=4.21.1,!=4.2
1.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0.dev0,>=3.19.5 in
/usr/local/lib/python3.10/dist-packages (from google-api-
core>=1.22.1->labelbox[data]) (3.20.3)
Requirement already satisfied: google-auth<3.0.dev0,>=2.14.1 in
/usr/local/lib/python3.10/dist-packages (from google-api-
core>=1.22.1->labelbox[data]) (2.27.0)
Requirement already satisfied: annotated-types>=0.4.0 in
/usr/local/lib/python3.10/dist-packages (from pydantic>=1.8->labelbox[data])
(0.6.0)
Requirement already satisfied: pydantic-core==2.16.3 in
/usr/local/lib/python3.10/dist-packages (from pydantic>=1.8->labelbox[data])
(2.16.3)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-
packages (from python-dateutil<2.9.0,>=2.8.2->labelbox[data]) (1.16.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests>=2.22.0->labelbox[data])
(3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests>=2.22.0->labelbox[data]) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in
```

```
/usr/local/lib/python3.10/dist-packages (from requests>=2.22.0->labelbox[data])
(2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests>=2.22.0->labelbox[data])
(2024.2.2)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from google-
auth<3.0.dev0,>=2.14.1->google-api-core>=1.22.1->labelbox[data]) (5.3.3)
Requirement already satisfied: pyasn1-modules>=0.2.1 in
/usr/local/lib/python3.10/dist-packages (from google-
auth<3.0.dev0,>=2.14.1->google-api-core>=1.22.1->labelbox[data]) (0.4.0)
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-
packages (from google-auth<3.0.dev0,>=2.14.1->google-api-
core>=1.22.1->labelbox[data]) (4.9)
Requirement already satisfied: pyasn1<0.7.0,>=0.4.6 in
/usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.1->google-
auth<3.0.dev0,>=2.14.1->google-api-core>=1.22.1->labelbox[data]) (0.6.0)
Building wheels for collected packages: pygeotile
  Building wheel for pygeotile (setup.py) … done
  Created wheel for pygeotile: filename=pyGeoTile-1.0.6-py3-none-any.whl
size=4873
sha256=20afd020de406df91c52a0d8f7d941bc63b3075c67f57ea2deed7864ad45cd69
  Stored in directory: /root/.cache/pip/wheels/02/4d/7c/e01f952fbc94a1c610404897
fe4ff7c3403e61597bd8d85813
Successfully built pygeotile
Installing collected packages: strenum, pygeotile, typeguard, geojson, labelbox
Successfully installed geojson-3.1.0 labelbox-3.67.0 pygeotile-1.0.6
strenum-0.4.15 typeguard-4.2.1
Collecting pascal-voc-writer
  Downloading pascal_voc_writer-0.1.4-py2.py3-none-any.whl (4.0 kB)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages
(from pascal-voc-writer) (3.1.3)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from jinja2->pascal-voc-writer) (2.1.5)
Installing collected packages: pascal-voc-writer
Successfully installed pascal-voc-writer-0.1.4
Collecting utm
  Downloading utm-0.7.0.tar.gz (8.7 kB)
  Preparing metadata (setup.py) … done
Building wheels for collected packages: utm
  Building wheel for utm (setup.py) … done
  Created wheel for utm: filename=utm-0.7.0-py3-none-any.whl size=6084
sha256=b14b271973985372283eb2e8c4de7c7b1fbc3c33ff5440ab6b68dbe9ae183f86
  Stored in directory: /root/.cache/pip/wheels/2f/a1/c8/543df0e8f5e824c3e92a432e
32deb9cd89ae686095ee8cfcbe
Successfully built utm
Installing collected packages: utm
Successfully installed utm-0.7.0
```

```
Collecting rasterio
  Downloading rasterio-1.3.9-cp310-cp310-manylinux2014_x86_64.whl (20.6 MB)
                     20.6/20.6 MB
51.0 MB/s eta 0:00:00
Collecting affine (from rasterio)
  Downloading affine-2.4.0-py3-none-any.whl (15 kB)
Requirement already satisfied: attrs in /usr/local/lib/python3.10/dist-packages
(from rasterio) (23.2.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-
packages (from rasterio) (2024.2.2)
Requirement already satisfied: click>=4.0 in /usr/local/lib/python3.10/dist-
packages (from rasterio) (8.1.7)
Requirement already satisfied: cligj>=0.5 in /usr/local/lib/python3.10/dist-
packages (from rasterio) (0.7.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages
(from rasterio) (1.25.2)
Collecting snuggs>=1.4.1 (from rasterio)
  Downloading snuggs-1.4.7-py3-none-any.whl (5.4 kB)
Requirement already satisfied: click-plugins in /usr/local/lib/python3.10/dist-
packages (from rasterio) (1.1.1)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-
packages (from rasterio) (67.7.2)
Requirement already satisfied: pyparsing>=2.1.6 in
/usr/local/lib/python3.10/dist-packages (from snuggs>=1.4.1->rasterio) (3.1.2)
Installing collected packages: snuggs, affine, rasterio
Successfully installed affine-2.4.0 rasterio-1.3.9 snuggs-1.4.7
Requirement already satisfied: geopandas in /usr/local/lib/python3.10/dist-
packages (0.13.2)
Requirement already satisfied: fiona>=1.8.19 in /usr/local/lib/python3.10/dist-
packages (from geopandas) (1.9.6)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-
packages (from geopandas) (24.0)
Requirement already satisfied: pandas>=1.1.0 in /usr/local/lib/python3.10/dist-
packages (from geopandas) (2.0.3)
Requirement already satisfied: pyproj>=3.0.1 in /usr/local/lib/python3.10/dist-
packages (from geopandas) (3.6.1)
Requirement already satisfied: shapely>=1.7.1 in /usr/local/lib/python3.10/dist-
packages (from geopandas) (2.0.3)
Requirement already satisfied: attrs>=19.2.0 in /usr/local/lib/python3.10/dist-
packages (from fiona>=1.8.19->geopandas) (23.2.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-
packages (from fiona>=1.8.19->geopandas) (2024.2.2)
Requirement already satisfied: click~=8.0 in /usr/local/lib/python3.10/dist-
packages (from fiona>=1.8.19->geopandas) (8.1.7)
Requirement already satisfied: click-plugins>=1.0 in
/usr/local/lib/python3.10/dist-packages (from fiona>=1.8.19->geopandas) (1.1.1)
Requirement already satisfied: cligj>=0.5 in /usr/local/lib/python3.10/dist-
packages (from fiona>=1.8.19->geopandas) (0.7.2)
```

```
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages
(from fiona>=1.8.19->geopandas) (1.16.0)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.10/dist-packages (from pandas>=1.1.0->geopandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas>=1.1.0->geopandas) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-
packages (from pandas>=1.1.0->geopandas) (2024.1)
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.10/dist-
packages (from pandas>=1.1.0->geopandas) (1.25.2)
```

[ ]:
```python
# Import Python libraries
import os
import sys
import glob
import shutil
import subprocess as sp
import warnings
import multiprocessing
import re
from tqdm import tqdm
import json
import tarfile

import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import geopandas as gpd
from shapely.geometry import Polygon

import rasterio
from rasterio.io import MemoryFile
from rasterio.merge import merge
from rasterio.plot import show
from rasterio.windows import from_bounds
from rasterio.enums import Resampling
from rasterio.warp import transform_bounds

import matplotlib
import matplotlib.pyplot as plt
import plotly.express as px
import matplotlib.pyplot as plt
import seaborn as sns

from PIL import Image

import pascal_voc_writer # common interchange format for object detection labels
```

```
import cv2

import detecto
from detecto import core, utils, visualize
from detecto.core import Dataset, Model
from detecto.visualize import show_labeled_image
from detecto.utils import normalize_transform

from xml.dom import minidom

import tifffile as tifi
import utm
from osgeo import gdal    # GDAL is used to work with the GeoTIFF file format

import pyproj
from pyproj import Proj

import labelbox
from labelbox import Client, OntologyBuilder
from labelbox.data.annotation_types import Geometry

from urllib.request import urlretrieve

import folium
from folium.plugins import MarkerCluster
from folium import plugins

from IPython.display import display

# Ignore all warnings
warnings.filterwarnings("ignore")
```

#1 | GeoTIFF

TIFF: Tagged Image File Format. This format embeds metadata tags alongside the image.

GeoTIFF extends this format to include additional tags that include georeferencing information. Some instances of the information that can be included in the tags: what are the bounds of the image with respect to earth, what projection was used to derive these?

In this code, we use GDAL, an open source GeoTIFF reader and writer.

#2 | Labelbox

This project uses Labelbox to annotate large amounts of image data for model training. The functions below support downloading the annotated images from Labelbox.

https://labelbox.com/product/annotate/

0.0.1 Import Labelbox labels

[ ]:
```
#-----------------------------------------------------------------------------
def get_labels(project_id):
    '''
    Download the labels from LabelBox.
    '''

    # Enter your Labelbox API key here
    LB_API_KEY = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
 ↪eyJ1c2VySWQiOiJjazdtNWVrZmY5Z3p4MDg2NmN4dXpoMWpsIiwib3JnYW5pemF0aW9uSWQiOiJjazdtNWVrZXh4aGF
 ↪TNn_GaAv1WGwjo4xJB2SJAe3gyMwDbapZq4vu8EJhcI"

    # Create Labelbox client
    lb = labelbox.Client(api_key=LB_API_KEY)

    # Get project by ID
    project = lb.get_project(project_id)

    # Export image and text data as an annotation generator:
    labels_annotation = project.label_generator()

    # Export labels as a json:
    labels = project.export_labels(download = True)

    return project, labels, labels_annotation
```

[ ]:
```
project_id = 'ckrvtzsgc1nyt0ybna91901rp'

project, labels, labels_annotation = get_labels(project_id)
```

0.0.2 Split data into train/validation/test sets

[ ]:
```
#-----------------------------------------------------------------------------
def split_data(labels):
  # Create list of images, names, and ids for labeled data, i.e. was not skipped
  img_list = [item['Labeled Data'] for item in labels if item['Skipped']==False]
  name_list = [item['External ID'] for item in labels if item['Skipped']==False]
  id_list = [item['ID'] for item in labels if item['Skipped']==False]

  # Create dictionary associating the image name with the url to the image
  img_dict = dict(zip(name_list, img_list))

  # Split the data: 80% train, 10% validation, 10% test
  train, val, test = np.split(name_list, [int(.8*len(name_list)), int(.
 ↪9*len(name_list))])
  return train, val, test, img_dict
```

[ ]:
```
train, val, test, img_dict = split_data(labels)
```

0.0.3 Download images

[ ]:
```
#-------------------------------------------------------------------------------
def download_set(work_path, set_list, img_dict):
    '''
    Download train, validation, and test set images into work_path.
    '''
    test_type = work_path.split('/')[-1]

    if not os.path.isdir(work_path):
        os.makedirs(work_path)
    else:
        print(f'{test_type.capitalize()} set already exists.')
        return

    # Create a progress bar
    pbar = tqdm(total=len(set_list), desc=f'Downloading {test_type} images',␣
 ↪unit='image')

    for item in set_list:
        url = img_dict.get(item)

        if not os.path.isfile(f'{os.path.join(os.getcwd(), work_path, item)}'):
            sp.call(f'wget "{url}" -O {os.path.join(work_path, item)}',␣
 ↪shell=True)

            # Update the progress bar
            pbar.update(1)

    # Close the progress bar
    pbar.close()

    print('Download complete.')
```

[ ]:
```
download_set('lettuce_object_detection/train', train, img_dict)

download_set('lettuce_object_detection/val', val, img_dict)

download_set('lettuce_object_detection/test', test, img_dict)
```

```
Downloading train images: 100%|      | 201/201 [05:49<00:00,  1.74s/image]
Download complete.
Downloading val images: 100%|      | 25/25 [00:42<00:00,  1.68s/image]
```

```
Download complete.
Downloading test images: 100%|      | 26/26 [00:46<00:00,  1.79s/image]
Download complete.
```

0.0.4 Rewrite labels in Pascal VOC XML Format

Pascal VOC XML is a common interchange format for object detection labels, and is used by detecto. These functions convert the labels into Pascal VOC XML for training the model.

https://roboflow.com/formats/pascal-voc-xml

[ ]:
```
#-------------------------------------------------------------------------------
def create_labels(data, train, val, test):
    '''
    For each image, creates a corresponding .xml with all object detections.
    '''

    # Create a progress bar
    total = len(train) + len(val) + len(test)
    pbar = tqdm(total=total, desc='Creating labels', unit='label')

    for i in range(len(data)):
        # Wrap the code in a try/except block so that we don't crash the program
        # if we run into an error.
        try:
            # Look up the name of the image and load the image
            file_name = data[i]['External ID'].replace('.png', '.txt')
            name = data[i]['External ID']
            out_name = name.replace('.png', '.xml')

            if name in test:
                file_type = 'test'
            elif name in train:
                file_type = 'train'
            else:
                file_type = 'val'

            img = cv2.imread(os.path.join('lettuce_object_detection',␣
 ↪file_type, name))

            # For every labeled object in the image, save the bounding box and
            # specific label 'plant'
            h, w, _ = img.shape
            label_list, x, y = [], [], []
            for a in range(len(data[i]['Label']['objects'])):
```

```
                points = data[i]['Label']['objects'][a]['bbox']
                label = data[i]['Label']['objects'][a]['value']

                label_list.append(label)
                x.append([points['left'], (points['left'] + points['width'])])
                y.append([points['top'], (points['top'] + points['height'])])

            final = list(zip(label_list, x, y))
            if not final:
                print('empty')

            # Write the bounding boxes and labels to the xml file
            name = os.path.join('lettuce_object_detection', file_type, name)
            writer = pascal_voc_writer.Writer(name, w, h)
            for item in final:
                min_x, max_x = item[1]
                min_y, max_y = item[2]
                writer.addObject(item[0], min_x, min_y, max_x, max_y)
            writer.save(os.path.join('lettuce_object_detection', file_type,␣
↪out_name))

            # Update the progress bar
            pbar.update(1)

        except:
            # We could have it print out a message or raise an error instead
            pass

    # Close the progress bar
    pbar.close()

    print('\nDone creating labels.')
```

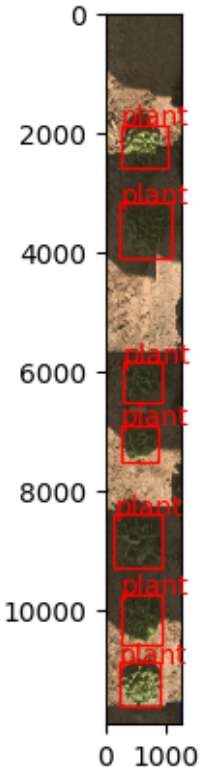
[ ]:
```
create_labels(labels, train, val, test)
```

```
Creating labels: 100%|      | 252/252 [01:49<00:00,  2.30label/s]


Done creating labels.
```

0.0.5 Visualize labels

Now that we have the object detections in the correct format, we can use detecto to visualize the detections on top of the images.

[ ]:
```
# detecto.core.Dataset
# Path to a folder that contains the images and XML files that describe the
# object detections
dataset = Dataset('lettuce_object_detection/train')

image, targets = dataset[4]

# detecto.visualize.show_labeled_image
# Show the images with the object detections
show_labeled_image(image, targets['boxes'], targets['labels'])
```



#3 | detecto

Detecto is a lightweight Python library that allows you to build an object detection model in 5 lines of code.

Detecto downloads a pretrained Faster R-CNN model from Pytorch and fine tunes it to your dataset. There are a few different models that you can choose from as a starting point. Here, we just use the default.

The feature set is also somewhat limited. You can change the hyperparameter tuning, but you don't have direct access to the accuracy, and you only get the loss of the validation set, not the training set.

https://detecto.readthedocs.io/en/latest/

0.0.6 Train the model

[ ]:
```
# Load the training dataset and create a DataLoader
train_dataset = core.Dataset('lettuce_object_detection/train')
loader = core.DataLoader(train_dataset, batch_size=2, shuffle=True)

val_dataset = core.Dataset('lettuce_object_detection/val')

# Load the model and tell it that we want it to detect the label 'plant'
model = core.Model(['plant'])
losses = model.fit(loader, val_dataset, epochs=5, learning_rate=0.001,␣
 ↪verbose=True)
```

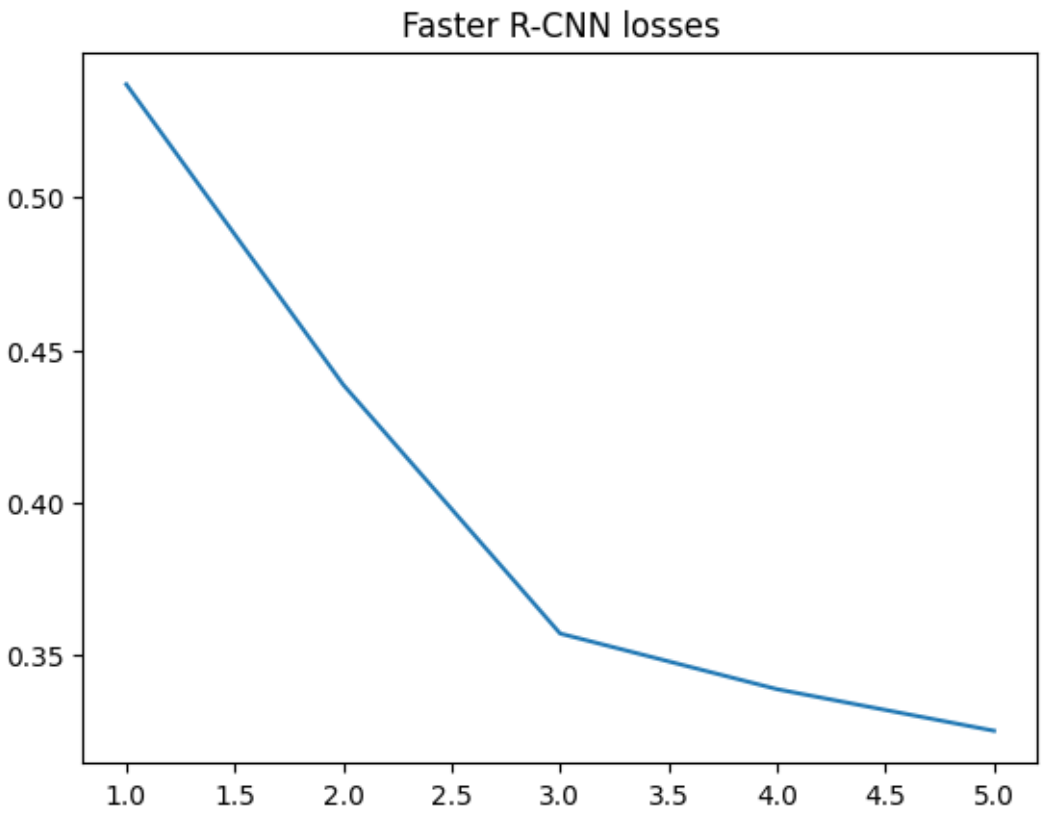
```
Downloading:
"https://download.pytorch.org/models/fasterrcnn_resnet50_fpn_coco-258fb6c6.pth"
to /root/.cache/torch/hub/checkpoints/fasterrcnn_resnet50_fpn_coco-258fb6c6.pth
100%|      | 160M/160M [00:01<00:00, 150MB/s]

Epoch 1 of 5
Begin iterating over training dataset

100%|      | 101/101 [03:22<00:00,  2.00s/it]

Begin iterating over validation dataset

100%|      | 25/25 [00:23<00:00,  1.05it/s]

Loss: 0.5370809018611908
Epoch 2 of 5
Begin iterating over training dataset

100%|      | 101/101 [03:12<00:00,  1.91s/it]

Begin iterating over validation dataset

100%|      | 25/25 [00:23<00:00,  1.05it/s]

Loss: 0.4386139464378357
Epoch 3 of 5
Begin iterating over training dataset

100%|      | 101/101 [03:10<00:00,  1.88s/it]

Begin iterating over validation dataset

100%|      | 25/25 [00:24<00:00,  1.02it/s]

Loss: 0.35703924715518953
Epoch 4 of 5
Begin iterating over training dataset

100%|      | 101/101 [03:09<00:00,  1.88s/it]
```

```
Begin iterating over validation dataset

100%|      | 25/25 [00:24<00:00,  1.01it/s]

Loss: 0.33880165219306946
Epoch 5 of 5
Begin iterating over training dataset

100%|      | 101/101 [03:09<00:00,  1.88s/it]

Begin iterating over validation dataset

100%|      | 25/25 [00:24<00:00,  1.03it/s]

Loss: 0.32517029404640196
```

A note on the training, we only set the learning rate, and we stop after 5 epochs. You may have to play with these settings to achieve good performance on your detection task, and detecto lets you set other hyperparameters like momentum, weight decay, etc.

0.0.7 Visualize loss

[ ]:
```python
# Plot with matplotlib
plt.plot(range(1,len(losses)+1), losses)

# Add a title
plt.title('Faster R-CNN losses')

# Show the plot
plt.show()
```



1 4 | Run model on data

1.0.1 Download drone image data

[ ]:
```
#----------------------------------------------------------------------------
def download_and_extract_tarball(url, extract_path='.'):
    '''
    Downloads and extracts a tarball from the url.
    '''

    # Download the tarball file
    file_name = os.path.basename(url)
    tarball_path = os.path.join(extract_path, file_name)
    urlretrieve(url, tarball_path)

    # Extract the contents of the tarball
    with tarfile.open(tarball_path) as tar:
        tar.extractall(path=extract_path)
        members = tar.getmembers()
        # Get the name of the first member in the archive
```

```
        top_level_dir = members[0].name.split('/')[0]

    # Return the path to the extracted data
    return os.path.join(extract_path, top_level_dir)
```

[ ]:
```
data_path = download_and_extract_tarball(
    'https://data.cyverse.org/dav-anon/iplant/projects/phytooracle/
 ↪season_10_lettuce_yr_2020/level_0/drone/RGB/2020-02-24_ortho.tar')
```

1.0.2 Get file paths

[ ]:
```
#-------------------------------------------------------------------------------
def filter_image_list(data_path, sequence, range_min, range_max):

    # Get list of all images
    img_list = glob.glob(os.path.join(data_path, sequence))

    # Regular expression pattern to match two integers at the start of the␣
 ↪basename
    pattern = re.compile(r'^(\d{2})(\d{2})')

    # Filter img_list using a list comprehension with a conditional expression
    filtered_img_list = [img for img in img_list if (match := pattern.search(os.
 ↪path.basename(img))) and range_min <= int(match.group(1)) <= range_max]

    return filtered_img_list
```

[ ]:
```
filtered_img_list = filter_image_list(data_path=data_path,
                                      sequence='*/*.tif',
                                      range_min=18,
                                      range_max=18)
```

1.0.3 Run inference on images

[ ]:
```
#-------------------------------------------------------------------------------
def open_image(img_path):
    '''
    Read a .tif image from img_path, and return as an np.array.
    '''

    img = tifi.imread(img_path)
    img = np.array(img)

    if img.shape[2]==4:
        img = np.array(img)[:,:,:3]
```

```
    return img

#--------------------------------------------------------------------------------
def pixel2geocoord(one_img, x_pix, y_pix):
    ds = gdal.Open(one_img)
    c, a, b, f, d, e = ds.GetGeoTransform()
    lon = a * int(x_pix) + b * int(y_pix) + a * 0.5 + b * 0.5 + c
    lat = d * int(x_pix) + e * int(y_pix) + d * 0.5 + e * 0.5 + f

    return (lat, lon)

#--------------------------------------------------------------------------------
def get_min_max(box):
    min_x, min_y, max_x, max_y = int(box[0]), int(box[1]), int(box[2]),␣
 ↪int(box[3])

    return min_x, min_y, max_x, max_y

#--------------------------------------------------------------------------------
def process_image(img, model, date):
    cont_cnt = 0
    lett_dict = {}

    plot = img.split('/')[-1].replace('_ortho.tif', '').replace('_plotclip.
 ↪tif', '')
    plot_name = plot.replace('_', ' ')
    print(f'Image: {plot_name}')
    a_img = open_image(img)
    df = pd.DataFrame()
    myProj = Proj("+proj=utm +zone=12 +north +ellps=WGS84 +datum=WGS84 +units=m␣
 ↪+no_defs")

    try:
        predictions = model.predict(a_img)
        labels, boxes, scores = predictions
        print(scores)
        copy = a_img.copy()

        for i, box in enumerate(boxes):

            cont_cnt += 1

            min_x, min_y, max_x, max_y = get_min_max(box)
            center_x, center_y = ((max_x+min_x)/2, (max_y+min_y)/2)
            northing, easting = pixel2geocoord(img, center_x, center_y)
            lon, lat = myProj(easting, northing, inverse=True)
            nw_n, nw_e = pixel2geocoord(img, min_x, max_y)
```

```
            se_n, se_e = pixel2geocoord(img, max_x, min_y)
            nw_lon, nw_lat = myProj(nw_e, nw_n, inverse=True)
            se_lon, se_lat = myProj(se_e, se_n, inverse=True)

            area_sq = (se_e - nw_e) * (se_n - nw_n)
            lett_dict[cont_cnt] = {
                'date': date,
                'pred_conf': scores[i].detach().numpy(),
                'plot': plot,
                'lon': lon,
                'lat': lat,
                'min_x': min_x,
                'max_x': max_x,
                'min_y': min_y,
                'max_y': max_y,
                'nw_lat': nw_lat,
                'nw_lon': nw_lon,
                'se_lat': se_lat,
                'se_lon': se_lon,
                'bounding_area_m2': area_sq
            }

        df = pd.DataFrame.from_dict(lett_dict, orient='index', columns=['date',
                                                                        'pred_conf',
                                                                        'plot',
                                                                        'lon',
                                                                        'lat',
                                                                        'min_x',
                                                                        'max_x',
                                                                        'min_y',
                                                                        'max_y',
                                                                        'nw_lat',
                                                                        'nw_lon',
                                                                        'se_lat',
                                                                        'se_lon',
                                                                        ␣
 ↪'bounding_area_m2']).set_index('date')
    except:
        pass

    return df
```

[ ]:
```
date = os.path.basename(data_path).split('_')[0]

major_df = pd.DataFrame()

for img in filtered_img_list:
```

```
    df = process_image(img, model, date)
    major_df = pd.concat([major_df, df], ignore_index=True)
```

```
Image: 1846
tensor([0.9946, 0.9917, 0.9887, 0.9867, 0.9832, 0.9811, 0.9805, 0.9764, 0.9642,
        0.9562, 0.9341, 0.9171, 0.9108, 0.8653, 0.7953, 0.7809, 0.3775, 0.3134,
        0.2543, 0.2242, 0.2031, 0.1861, 0.1720, 0.1705, 0.1373, 0.1191, 0.1076,
        0.1031, 0.0793, 0.0783, 0.0692, 0.0662, 0.0622, 0.0620, 0.0563, 0.0555])
Image: 1830
tensor([0.9989, 0.9989, 0.9988, 0.9985, 0.9983, 0.9980, 0.9957, 0.9944, 0.9011,
        0.8782, 0.7854, 0.6093, 0.6008, 0.5906, 0.5529, 0.4479, 0.4202, 0.3693,
        0.3507, 0.3274, 0.2840, 0.2466, 0.2371, 0.2215, 0.2108, 0.2078, 0.1567,
        0.1543, 0.1528, 0.1442, 0.1431, 0.1357, 0.1294, 0.1204, 0.1186, 0.1116,
        0.0989, 0.0968, 0.0930, 0.0921, 0.0907, 0.0818, 0.0815, 0.0803, 0.0782,
        0.0756, 0.0755, 0.0701, 0.0689, 0.0526, 0.0517])
Image: 1804
tensor([0.9994, 0.9994, 0.9989, 0.9987, 0.9981, 0.9979, 0.9737, 0.9010, 0.3545,
        0.3479, 0.2811, 0.2223, 0.0956, 0.0921, 0.0746, 0.0738, 0.0704, 0.0620,
        0.0569, 0.0518, 0.0511, 0.0509])
Image: 1844
tensor([0.9980, 0.9977, 0.9973, 0.9971, 0.9968, 0.9956, 0.9949, 0.9884, 0.5617,
        0.1453, 0.1122, 0.0769, 0.0695, 0.0683, 0.0652, 0.0637, 0.0523])
Image: 1808
tensor([0.9993, 0.9991, 0.9990, 0.9987, 0.9985, 0.9984, 0.9981, 0.9961, 0.7089,
        0.5476, 0.4644, 0.4195, 0.3992, 0.3194, 0.2942, 0.2529, 0.2475, 0.1626,
        0.1514, 0.1465, 0.1228, 0.1215, 0.1142, 0.1112, 0.0863, 0.0730, 0.0640,
        0.0612, 0.0561, 0.0516, 0.0514])
Image: 1838
tensor([0.9987, 0.9980, 0.9978, 0.9968, 0.9965, 0.9960, 0.9899, 0.9473, 0.9361,
        0.8255, 0.6476, 0.6187, 0.5796, 0.2843, 0.2396, 0.2247, 0.1811, 0.1092,
        0.0991, 0.0868, 0.0607, 0.0571, 0.0547])
Image: 1805
tensor([0.9993, 0.9993, 0.9992, 0.9992, 0.9989, 0.9985, 0.9983, 0.9981, 0.9816,
        0.5611, 0.2724, 0.1912, 0.1564, 0.0943, 0.0902, 0.0617, 0.0611, 0.0561])
Image: 1833
tensor([0.9993, 0.9993, 0.9993, 0.9991, 0.9987, 0.9985, 0.9959, 0.4006, 0.3373,
        0.3031, 0.2732, 0.1944, 0.1840, 0.1726, 0.1725, 0.1519, 0.1110, 0.1016,
        0.0856, 0.0817, 0.0793, 0.0786, 0.0593, 0.0569, 0.0567])
Image: 1818
tensor([0.9986, 0.9973, 0.9961, 0.9953, 0.9933, 0.9928, 0.9924, 0.8855, 0.7369,
        0.7192, 0.6646, 0.5864, 0.4428, 0.3862, 0.3263, 0.2970, 0.2349, 0.1897,
        0.1371, 0.1233, 0.1120, 0.1016, 0.1015, 0.1003, 0.0952, 0.0914, 0.0887,
        0.0788, 0.0775, 0.0743, 0.0737, 0.0697, 0.0658, 0.0653, 0.0650, 0.0566,
        0.0529, 0.0512])
Image: 1819
tensor([0.9991, 0.9989, 0.9987, 0.9985, 0.9984, 0.9983, 0.9980, 0.9976, 0.8749,
        0.7357, 0.5836, 0.4488, 0.4256, 0.3574, 0.3226, 0.3088, 0.3004, 0.2863,
```

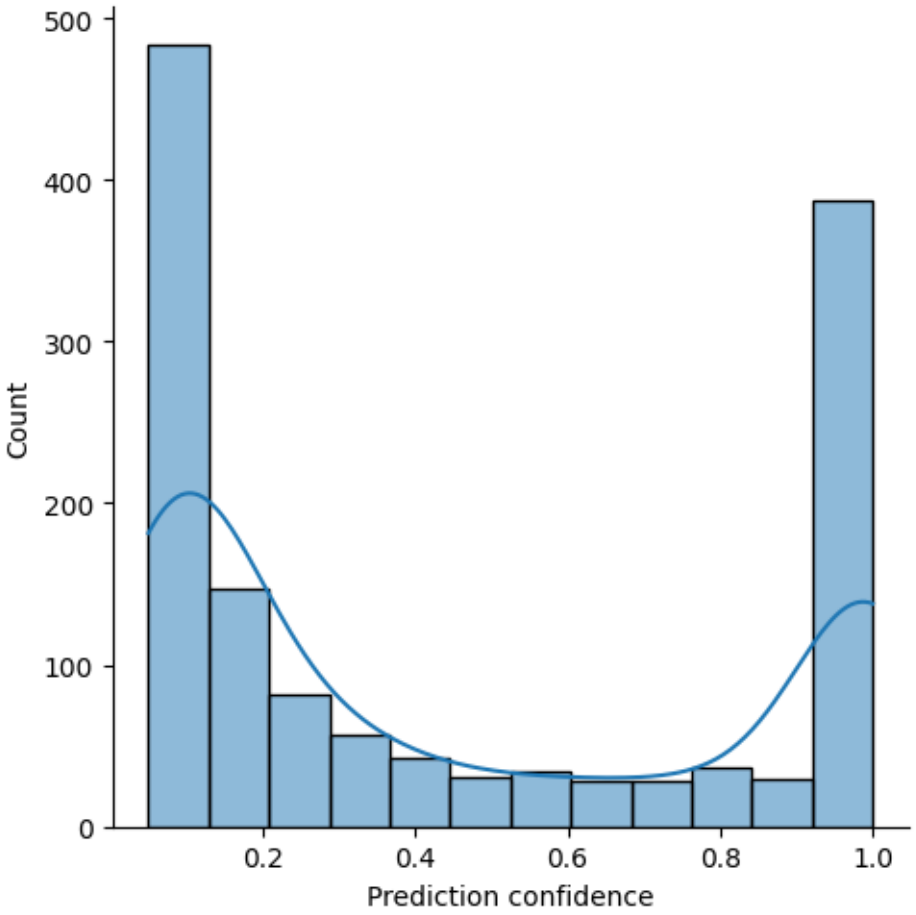
```
        0.2214, 0.1750, 0.1439, 0.0904, 0.0890, 0.0828, 0.0763, 0.0758, 0.0691,
        0.0667, 0.0585, 0.0533, 0.0509])
Image: 1810
tensor([0.9993, 0.9987, 0.9986, 0.9984, 0.9981, 0.9980, 0.9978, 0.9978, 0.9952,
        0.8032, 0.7715, 0.7322, 0.6501, 0.5624, 0.5538, 0.4310, 0.3456, 0.2729,
        0.1867, 0.1213, 0.1201, 0.1144, 0.1139, 0.1109, 0.0849, 0.0794, 0.0630,
        0.0538])
Image: 1847
tensor([0.9976, 0.9961, 0.9956, 0.9954, 0.9950, 0.9945, 0.9941, 0.9895, 0.9208,
        0.7192, 0.2577, 0.0928, 0.0760, 0.0734, 0.0709, 0.0671, 0.0666, 0.0642,
        0.0561])
Image: 1840
tensor([0.9992, 0.9989, 0.9986, 0.9984, 0.9983, 0.9982, 0.9980, 0.9980, 0.7025,
        0.6108, 0.4909, 0.4652, 0.3717, 0.3535, 0.2454, 0.2225, 0.2049, 0.1667,
        0.1521, 0.1506, 0.1388, 0.1157, 0.1140, 0.1139, 0.1097, 0.1027, 0.0938,
        0.0919, 0.0780, 0.0758, 0.0725, 0.0715, 0.0669, 0.0650, 0.0535])
Image: 1821
tensor([0.9992, 0.9987, 0.9987, 0.9987, 0.9985, 0.9976, 0.9972, 0.9916, 0.8184,
        0.7702, 0.6292, 0.2313, 0.1972, 0.1562, 0.0948, 0.0927, 0.0833, 0.0800,
        0.0714, 0.0691, 0.0678, 0.0660, 0.0590, 0.0578, 0.0561, 0.0553, 0.0549])
Image: 1835
tensor([0.9989, 0.9983, 0.9982, 0.9982, 0.9981, 0.9978, 0.9975, 0.9747, 0.6507,
        0.4092, 0.3505, 0.3168, 0.2538, 0.1802, 0.1601, 0.1361, 0.1319, 0.1137,
        0.1076, 0.1022, 0.0753, 0.0724, 0.0639, 0.0548, 0.0505])
Image: 1817
tensor([0.9995, 0.9994, 0.9994, 0.9993, 0.9992, 0.9988, 0.9988, 0.9987, 0.6358,
        0.6163, 0.5367, 0.4986, 0.4576, 0.4552, 0.3888, 0.2172, 0.1544, 0.1330,
        0.1182, 0.1175, 0.0872, 0.0833, 0.0820, 0.0815, 0.0778, 0.0759, 0.0716,
        0.0644, 0.0631, 0.0522])
Image: 1827
tensor([0.9993, 0.9991, 0.9984, 0.9977, 0.9972, 0.9962, 0.6479, 0.6286, 0.6223,
        0.3866, 0.3554, 0.3386, 0.3049, 0.2719, 0.2218, 0.1979, 0.1723, 0.1645,
        0.1561, 0.1207, 0.1184, 0.1059, 0.0719, 0.0687])
Image: 1832
tensor([0.9991, 0.9986, 0.9986, 0.9978, 0.9976, 0.9972, 0.9965, 0.9292, 0.9105,
        0.8991, 0.8907, 0.8344, 0.7166, 0.7138, 0.6639, 0.6240, 0.5808, 0.3768,
        0.3247, 0.3067, 0.2407, 0.2139, 0.1775, 0.1686, 0.1679, 0.1262, 0.1024,
        0.0903, 0.0734, 0.0715, 0.0707, 0.0683, 0.0659, 0.0528])
Image: 1816
tensor([0.9994, 0.9992, 0.9992, 0.9990, 0.9990, 0.9989, 0.9986, 0.9986, 0.9982,
        0.9406, 0.9096, 0.8990, 0.8875, 0.8692, 0.8102, 0.7965, 0.7343, 0.7108,
        0.6983, 0.6334, 0.5617, 0.5125, 0.3329, 0.2786, 0.2686, 0.2046, 0.2000,
        0.1939, 0.1326, 0.1123, 0.1084, 0.0909, 0.0907, 0.0755, 0.0649, 0.0580,
        0.0568, 0.0553, 0.0515, 0.0513, 0.0501])
Image: 1803
tensor([0.9995, 0.9993, 0.9984, 0.9982, 0.9982, 0.9981, 0.9979, 0.8643, 0.8233,
        0.7397, 0.5836, 0.5684, 0.5593, 0.5407, 0.2910, 0.1921, 0.1619, 0.1312,
        0.0678, 0.0518])
```

```
Image: 1824
tensor([0.9989, 0.9985, 0.9983, 0.9983, 0.9979, 0.9979, 0.9953, 0.9588, 0.8807,
        0.7504, 0.7272, 0.6808, 0.6418, 0.5266, 0.5043, 0.4811, 0.4350, 0.3833,
        0.3761, 0.2990, 0.2793, 0.2449, 0.1763, 0.1714, 0.1678, 0.1650, 0.1521,
        0.1444, 0.0982, 0.0857, 0.0839, 0.0838, 0.0783, 0.0781, 0.0776, 0.0718,
        0.0704, 0.0703, 0.0658, 0.0562, 0.0512])
Image: 1843
tensor([0.9992, 0.9986, 0.9984, 0.9983, 0.9982, 0.9981, 0.9978, 0.9977, 0.2632,
        0.2408, 0.1392, 0.1039, 0.0761, 0.0730, 0.0622, 0.0570, 0.0502])
Image: 1822
tensor([0.9985, 0.9976, 0.9975, 0.9973, 0.9962, 0.9961, 0.9951, 0.9949, 0.9934,
        0.9131, 0.8127, 0.7707, 0.7021, 0.6952, 0.5227, 0.4589, 0.4163, 0.2677,
        0.1377, 0.1346, 0.1188, 0.1054, 0.0936, 0.0903, 0.0845, 0.0805, 0.0794,
        0.0786, 0.0742, 0.0731, 0.0694, 0.0657, 0.0594, 0.0566, 0.0564, 0.0517,
        0.0514, 0.0508])
Image: 1834
tensor([0.9992, 0.9990, 0.9985, 0.9984, 0.9981, 0.9979, 0.9976, 0.9972, 0.9968,
        0.9879, 0.9306, 0.9270, 0.9153, 0.9129, 0.8383, 0.6544, 0.4650, 0.4489,
        0.4421, 0.2031, 0.1761, 0.1611, 0.1501, 0.1226, 0.1184, 0.1171, 0.1147,
        0.0888, 0.0833, 0.0759, 0.0737, 0.0707, 0.0701, 0.0652, 0.0617, 0.0518])
Image: 1815
tensor([0.9987, 0.9978, 0.9965, 0.9964, 0.9956, 0.9956, 0.9953, 0.9951, 0.9930,
        0.9923, 0.7569, 0.6048, 0.5006, 0.4032, 0.2963, 0.2872, 0.2826, 0.2416,
        0.2184, 0.2045, 0.2025, 0.2014, 0.1800, 0.1788, 0.1528, 0.1458, 0.0986,
        0.0833, 0.0799, 0.0786, 0.0734, 0.0680, 0.0661, 0.0658, 0.0566, 0.0501])
Image: 1801
tensor([0.9991, 0.9991, 0.9984, 0.9983, 0.9971, 0.9941, 0.6959, 0.6745, 0.6034,
        0.5274, 0.5105, 0.4314, 0.2561, 0.2507, 0.2395, 0.2193, 0.2047, 0.1777,
        0.1740, 0.1598, 0.1145, 0.0921, 0.0885, 0.0773, 0.0742, 0.0742, 0.0685,
        0.0674, 0.0590, 0.0584, 0.0528, 0.0523])
Image: 1828
tensor([0.9991, 0.9989, 0.9986, 0.9980, 0.9971, 0.9958, 0.9684, 0.7610, 0.6530,
        0.4621, 0.3988, 0.3747, 0.3589, 0.3444, 0.3442, 0.3351, 0.3073, 0.2513,
        0.2310, 0.2132, 0.1767, 0.1604, 0.1575, 0.1148, 0.1039, 0.0854, 0.0826,
        0.0777, 0.0764, 0.0673, 0.0595, 0.0505, 0.0503])
Image: 1842
tensor([0.9990, 0.9986, 0.9984, 0.9979, 0.9974, 0.9972, 0.9964, 0.9961, 0.9623,
        0.9509, 0.9277, 0.8098, 0.7483, 0.6479, 0.5926, 0.4625, 0.4466, 0.3753,
        0.3730, 0.3130, 0.2221, 0.1727, 0.1644, 0.1407, 0.1323, 0.1304, 0.1076,
        0.0929, 0.0890, 0.0871, 0.0712, 0.0616, 0.0593])
Image: 1823
tensor([0.9991, 0.9985, 0.9985, 0.9983, 0.9981, 0.9971, 0.9969, 0.1452, 0.1216,
        0.1122, 0.1042, 0.0873, 0.0786, 0.0776, 0.0648, 0.0576, 0.0561, 0.0553])
Image: 1812
tensor([0.9985, 0.9984, 0.9983, 0.9982, 0.9979, 0.9975, 0.9964, 0.9943, 0.6011,
        0.5402, 0.4272, 0.4182, 0.3884, 0.2151, 0.2017, 0.2010, 0.1838, 0.1518,
        0.1517, 0.1360, 0.0852, 0.0808, 0.0733, 0.0719, 0.0718, 0.0660, 0.0607,
        0.0606, 0.0515])
```

```
Image: 1813
tensor([0.9995, 0.9992, 0.9992, 0.9991, 0.9987, 0.9984, 0.9914, 0.8908, 0.5998,
        0.2881, 0.2727, 0.2540, 0.2156, 0.1714, 0.1593, 0.1391, 0.1152, 0.1123,
        0.1067, 0.0921, 0.0906, 0.0863, 0.0706, 0.0664, 0.0619, 0.0616, 0.0531,
        0.0512, 0.0505])
Image: 1845
tensor([0.9991, 0.9990, 0.9988, 0.9984, 0.9977, 0.9974, 0.9974, 0.9950, 0.4004,
        0.3792, 0.1803, 0.1353, 0.1039, 0.0680])
Image: 1831
tensor([0.9994, 0.9992, 0.9991, 0.9990, 0.9990, 0.9988, 0.9987, 0.9971, 0.8784,
        0.8236, 0.8117, 0.8050, 0.4381, 0.3082, 0.2545, 0.2443, 0.1828, 0.1139,
        0.0972, 0.0951, 0.0904, 0.0832, 0.0703, 0.0686])
Image: 1837
tensor([0.9993, 0.9991, 0.9991, 0.9984, 0.9979, 0.9971, 0.9970, 0.5478, 0.4749,
        0.3326, 0.2569, 0.2046, 0.1839, 0.1450, 0.1415, 0.1322, 0.1078, 0.0950,
        0.0939, 0.0777, 0.0743, 0.0710, 0.0673, 0.0640, 0.0633, 0.0555, 0.0526])
Image: 1825
tensor([0.9992, 0.9982, 0.9974, 0.9968, 0.7486, 0.6027, 0.4806, 0.2915, 0.2010,
        0.1826, 0.1746, 0.1379, 0.1372, 0.0867, 0.0777, 0.0776, 0.0732, 0.0580,
        0.0572, 0.0563, 0.0545])
Image: 1848
tensor([0.9994, 0.9993, 0.9991, 0.9990, 0.9987, 0.9985, 0.9983, 0.9982, 0.8887,
        0.8502, 0.8279, 0.8129, 0.7772, 0.7526, 0.7471, 0.7343, 0.5177, 0.4757,
        0.4716, 0.4446, 0.4248, 0.3251, 0.3211, 0.2930, 0.2790, 0.2763, 0.2173,
        0.1784, 0.1360, 0.1285, 0.1255, 0.1241, 0.1117, 0.1088, 0.1073, 0.1027,
        0.1021, 0.0699, 0.0695, 0.0546, 0.0521, 0.0521])
Image: 1826
tensor([0.9982, 0.9976, 0.9974, 0.9969, 0.9964, 0.9963, 0.9938, 0.9018, 0.8108,
        0.6740, 0.6040, 0.3925, 0.3212, 0.2898, 0.2505, 0.2361, 0.2181, 0.1927,
        0.1888, 0.1836, 0.1667, 0.1452, 0.1406, 0.1229, 0.1215, 0.1207, 0.1105,
        0.1036, 0.0922, 0.0879, 0.0849, 0.0686, 0.0660, 0.0623, 0.0616, 0.0611,
        0.0548, 0.0541])
Image: 1820
tensor([0.9986, 0.9981, 0.9976, 0.9967, 0.9967, 0.9948, 0.9508, 0.9315, 0.9181,
        0.8571, 0.8394, 0.8362, 0.7833, 0.7645, 0.6095, 0.5219, 0.4834, 0.3031,
        0.2986, 0.2486, 0.2459, 0.1880, 0.1260, 0.1222, 0.1080, 0.1044, 0.0939,
        0.0868, 0.0734, 0.0652, 0.0620, 0.0603, 0.0585, 0.0577, 0.0543, 0.0537,
        0.0521])
Image: 1802
tensor([0.9997, 0.9990, 0.9990, 0.9989, 0.9985, 0.9961, 0.9544, 0.9310, 0.8774,
        0.8368, 0.6762, 0.6717, 0.6281, 0.5870, 0.4790, 0.3855, 0.3224, 0.2979,
        0.1887, 0.1670, 0.1625, 0.1577, 0.1526, 0.1327, 0.1231, 0.1225, 0.1213,
        0.1036, 0.0945, 0.0867, 0.0717, 0.0573])
Image: 1807
tensor([0.9985, 0.9973, 0.9973, 0.9973, 0.9967, 0.9964, 0.9961, 0.9952, 0.9945,
        0.9930, 0.8930, 0.5526, 0.5289, 0.3587, 0.2674, 0.2669, 0.2547, 0.2316,
        0.1457, 0.1170, 0.1098, 0.0976, 0.0876, 0.0847, 0.0769, 0.0707, 0.0701,
        0.0641, 0.0540, 0.0520])
```

```
Image: 1839
tensor([0.9991, 0.9990, 0.9990, 0.9989, 0.9984, 0.9980, 0.9977, 0.9897, 0.7492,
        0.5223, 0.4997, 0.3803, 0.3796, 0.3606, 0.3471, 0.2322, 0.2278, 0.2203,
        0.1939, 0.1725, 0.1416, 0.1334, 0.1157, 0.1141, 0.1104, 0.0703, 0.0693,
        0.0664, 0.0663, 0.0501])
Image: 1809
tensor([0.9991, 0.9982, 0.9982, 0.9981, 0.9964, 0.9963, 0.9954, 0.9941, 0.8028,
        0.5927, 0.2311, 0.1614, 0.1492, 0.0700, 0.0696, 0.0660, 0.0610, 0.0586,
        0.0561, 0.0550, 0.0537, 0.0536, 0.0509])
Image: 1841
tensor([0.9992, 0.9989, 0.9989, 0.9986, 0.9971, 0.9870, 0.9616, 0.8065, 0.2149,
        0.0914, 0.0880, 0.0620, 0.0504])
Image: 1811
tensor([0.9990, 0.9986, 0.9985, 0.9974, 0.9969, 0.9967, 0.9965, 0.9924, 0.7499,
        0.5378, 0.3815, 0.2871, 0.2147, 0.1651, 0.1549, 0.1131, 0.1078, 0.0938,
        0.0885, 0.0836, 0.0763, 0.0762, 0.0713, 0.0692, 0.0690, 0.0601, 0.0599,
        0.0564])
Image: 1836
tensor([0.9976, 0.9970, 0.9951, 0.9947, 0.9930, 0.9915, 0.9870, 0.9869, 0.7854,
        0.5108, 0.3137, 0.2439, 0.1934, 0.1608, 0.1392, 0.1340, 0.1203, 0.1201,
        0.1039, 0.0925, 0.0812, 0.0766, 0.0655, 0.0636, 0.0627])
Image: 1814
tensor([0.9985, 0.9984, 0.9983, 0.9982, 0.9980, 0.9979, 0.9774, 0.9602, 0.9399,
        0.8257, 0.8239, 0.7904, 0.7786, 0.5380, 0.4060, 0.3369, 0.2938, 0.2920,
        0.2574, 0.2438, 0.2100, 0.2062, 0.1801, 0.1593, 0.1408, 0.1270, 0.1220,
        0.0899, 0.0865, 0.0799, 0.0748, 0.0738, 0.0736, 0.0688, 0.0582, 0.0580,
        0.0547, 0.0515])
Image: 1829
tensor([0.9997, 0.9993, 0.9993, 0.9993, 0.9990, 0.9990, 0.9989, 0.3735, 0.3269,
        0.2714, 0.1323, 0.1105, 0.1043, 0.0514])
Image: 1806
tensor([0.9974, 0.9972, 0.9946, 0.9935, 0.9919, 0.9917, 0.9916, 0.9912, 0.7874,
        0.7759, 0.7031, 0.4244, 0.3738, 0.3590, 0.3054, 0.2273, 0.2125, 0.1624,
        0.1326, 0.1181, 0.1028, 0.0960, 0.0887, 0.0856, 0.0810, 0.0806, 0.0760,
        0.0729, 0.0649, 0.0634, 0.0605, 0.0588, 0.0505])
```

[ ]:
```
major_df['pred_conf'] = major_df['pred_conf'].astype(float)
sns.displot(x='pred_conf', kde=True, data=major_df)
plt.xlabel('Prediction confidence')
```



[ ]:
```
# filter dataframe for predictions with >= 95% confidence
major_df = major_df[major_df['pred_conf']>=0.95]
```

[ ]:
```
major_df
```

[ ]:

	pred_conf	plot	lon	lat	min_x	max_x	min_y	max_y	\
0	0.994561	1846	-111.974808	33.075147	30	103	821	895	
1	0.991678	1846	-111.974808	33.075143	27	98	962	1036	
2	0.988689	1846	-111.974808	33.075157	32	100	523	585	
3	0.986702	1846	-111.974808	33.075166	36	121	231	306	
4	0.983218	1846	-111.974808	33.075150	35	93	727	784	
…	…	…	…	…	…	…	…	…	
1356	0.993456	1806	-111.975026	33.075151	31	90	698	757	
1357	0.991883	1806	-111.975026	33.075169	28	84	156	208	
1358	0.991722	1806	-111.975026	33.075145	24	95	894	964	
1359	0.991559	1806	-111.975026	33.075154	23	85	609	659	
1360	0.991167	1806	-111.975026	33.075158	33	84	495	553	

```
         nw_lat      nw_lon     se_lat      se_lon  bounding_area_m2
0     33.075146 -111.974809  33.075148 -111.974807          0.068785
1     33.075141 -111.974809  33.075144 -111.974807          0.066900
2     33.075156 -111.974809  33.075158 -111.974807          0.053683
3     33.075165 -111.974809  33.075167 -111.974806          0.081174
4     33.075150 -111.974809  33.075151 -111.974807          0.042096
…           …          …          …          …                 …
1356  33.075150 -111.975027  33.075152 -111.975025          0.044324
1357  33.075168 -111.975027  33.075170 -111.975025          0.037079
1358  33.075144 -111.975027  33.075146 -111.975024          0.063284
1359  33.075154 -111.975027  33.075155 -111.975025          0.039473
1360  33.075157 -111.975027  33.075159 -111.975025          0.037665

[375 rows x 13 columns]
```

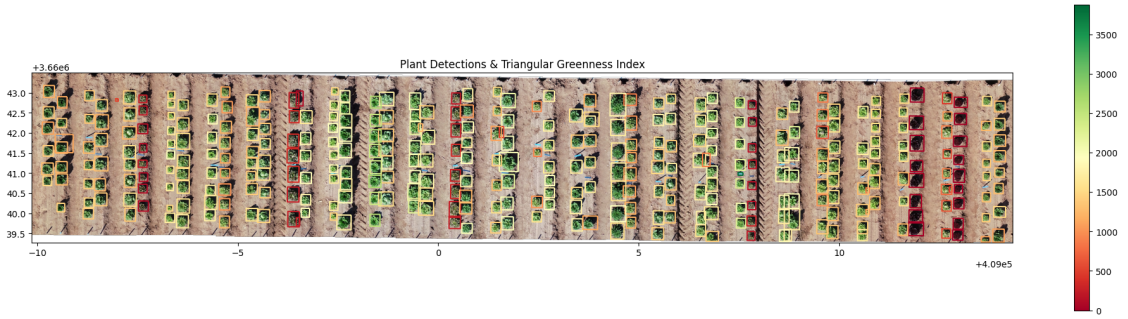
1.0.4 Visualize plant detections and calculated phenotypes

[ ]:
```
#-----------------------------------------------------------------------------
def merge_geotiffs(file_list, out_file):
    src_files_to_mosaic = []
    for file in file_list:
        src = rasterio.open(file)
        src_files_to_mosaic.append(src)
    mosaic, out_trans = merge(src_files_to_mosaic)
    out_meta = src.meta.copy()
    out_meta.update({"driver": "GTiff",
                     "height": mosaic.shape[1],
                     "width": mosaic.shape[2],
                     "transform": out_trans})
    with rasterio.open(out_file, "w", **out_meta) as dest:
        dest.write(mosaic)

#-----------------------------------------------------------------------------
def visualize_boxes_on_mosaic(mosaic_file, df, se_lat_col='se_lat',␣
 ↪se_lon_col='se_lon', nw_lat_col='nw_lat', nw_lon_col='nw_lon'):
    # Convert lat and lon points to easting and northing
    geometry = [Polygon([(se_lon, se_lat), (nw_lon, se_lat), (nw_lon, nw_lat),␣
 ↪(se_lon, nw_lat)]) for se_lat, se_lon, nw_lat, nw_lon in zip(df[se_lat_col],␣
 ↪df[se_lon_col], df[nw_lat_col], df[nw_lon_col])]
    gdf = gpd.GeoDataFrame(df, geometry=geometry)
    gdf.crs = 'EPSG:4326'
    gdf = gdf.to_crs('EPSG:32612')

    # Read mosaic file
    with rasterio.open(mosaic_file) as src:
        mosaic = src.read()
        bounds = src.bounds
```

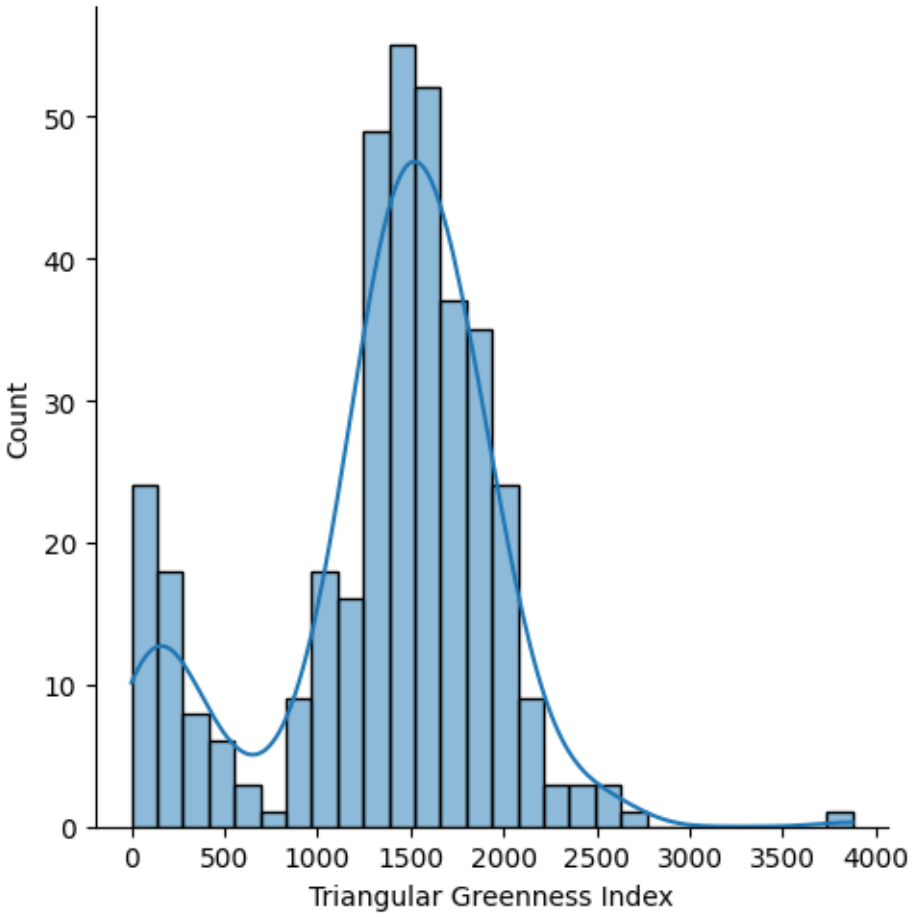
```
        extent = [bounds.left, bounds.right, bounds.bottom, bounds.top]

    # Calculate TGI for each detection
    tgi_values = []
    for index, row in gdf.iterrows():
        # Clip out plant using bounding box
        minx, miny, maxx, maxy = row.geometry.bounds
        window = from_bounds(minx, miny, maxx, maxy, transform=src.transform)
        clipped_mosaic = mosaic[:, int(window.row_off):int(window.
↪row_off+window.height), int(window.col_off):int(window.col_off+window.width)]

        # Calculate TGI
        r_band = clipped_mosaic[0]
        g_band = clipped_mosaic[1]
        b_band = clipped_mosaic[2]
        # tgi = (g_band.astype(float)-(0.39*r_band.astype(float))-(0.61*b_band.
↪astype(float)))
        tgi = (-1) * 0.5 * ((200*(r_band.astype(float) - g_band.
↪astype(float)))-(100 * (r_band.astype(float) - g_band.astype(float))))
        tgi[tgi < 0] = np.nan
        median = np.nanmedian(tgi)
        tgi_values.append(median)

    # Add TGI values to dataframe
    gdf['TGI'] = tgi_values

    # Create colormap and normalize TGI values
    cmap = plt.cm.get_cmap('RdYlGn')
    norm = matplotlib.colors.Normalize(vmin=gdf['TGI'].min(), vmax=gdf['TGI'].
↪max())

    # Plot mosaic and boxes
    fig, ax = plt.subplots(figsize=(25, 25))
    ax.imshow(mosaic.transpose(1, 2, 0), extent=extent)

    for index, row in gdf.iterrows():
        gpd.GeoSeries(row.geometry.boundary).plot(ax=ax,␣
↪color=cmap(norm(row['TGI'])))

    # Add colorbar
    sm = plt.cm.ScalarMappable(cmap=cmap, norm=norm)
    sm.set_array([])
    cbar = fig.colorbar(sm, ax=ax, shrink=0.25)

    # Make colorbar smaller
    # cbar.ax.set_position([0.85, 0.15, 0.03, 0.7])
```

```
    # Add title to plot
    ax.set_title("Plant Detections & Triangular Greenness Index")

    plt.show()

    return gdf
```

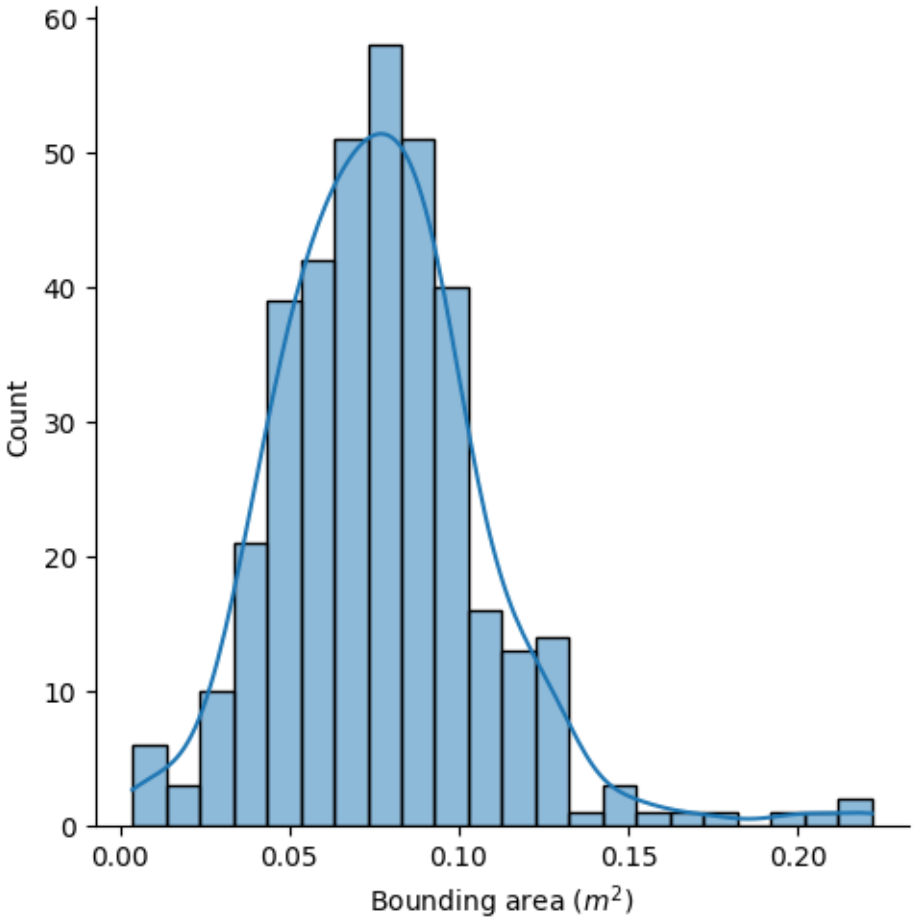
[ ]:
```
import matplotlib.colors as colors

merge_geotiffs(file_list = filtered_img_list, out_file = 'mosaic.tif')
out = visualize_boxes_on_mosaic(mosaic_file = 'mosaic.tif', df = major_df)
```



[ ]:
```
sns.displot(x='TGI', kde=True, data=out)
plt.xlabel('Triangular Greenness Index');
```



[ ]:
```
sns.displot(x='bounding_area_m2', kde=True, data=out)
plt.xlabel('Bounding area ($m^2$)');
```


60
50
40
30
20
10
0
Count
0.00
0.05
0.10
0.15
0.20
Bounding area (m^2)

Computational Phenotype Extraction with Machine Learning - 3D: point cloud feature extraction (Emmanuel Gonzalez)

Learning Objectives

1. Gain the ability to visualize and manipulate point cloud datasets programmatically, irrespective of previous exposure to point cloud data.
2. Learn to extract both traditional phenotypic traits and topological traits from point cloud datasets.
3. Understand how to distinguish between traditional and topological phenotypic traits using visualization techniques and by assessing their ability to encapsulate phenotypic variation.
4. Apply techniques of topological data analysis to perform routine plant phenotyping tasks, such as examining temporal growth patterns and alterations in plant architecture.

point_cloud_feature_extraction_trad_live

April 12, 2024

**

Machine Learning-Based Computational Extraction of Phenotypes from Point Clouds

**

1 Import libraries

[ ]:
```
#@title
%%capture
# Install packages
!python3 -m pip install --upgrade pip
!python3 -m pip install pingouin
!python3 -m pip install umap-learn
!python3 -m pip install open3d
!python3 -m pip install "pybind11[global]"
!python3 -m pip install euchar
# !python3 -m pip install matplotlib==3.6.2
!python3 -m pip install --upgrade umap-learn

# Genearl packages
import os
import sys
import pandas as pd
import numpy as np
import glob
import multiprocessing
import re
from datetime import datetime
import random
import warnings
import subprocess as subp
import seaborn as sns
import plotly.express as px
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import statistics
import pingouin as pg
```

```
# Classification packages
import scipy.stats as stats
from sklearn import preprocessing
from sklearn.decomposition import KernelPCA
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.svm import LinearSVC
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.tree import ExtraTreeClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neighbors import RadiusNeighborsClassifier
from sklearn.neighbors import NearestCentroid
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV

# import umap
from sklearn.metrics import classification_report
import scipy as sp
from scipy.stats import pearsonr

# Topological data analysis packages
import euchar.utils
from euchar.curve import image_2D, image_3D, filtration
from euchar.filtrations import alpha_filtration_2D, alpha_filtration_3D,␣
 ↪inverse_density_filtration
from euchar.display import piecewise_constant_curve
from seaborn import distplot,displot,histplot
import open3d as o3d


warnings.filterwarnings('ignore')
sns.set_context("poster", font_scale=0.7)
sns.set_palette("colorblind")
```

```
sns.set_style("whitegrid")
```

2 1 | Extraction of Topological Data Analysis (TDA) shape descriptors

Click on “Show code” to see all functions for this section. Otherwise, click the Run Cell icon.

[ ]:
```
#@title
def visualize(obj):
    '''Visualize open3d object'''

    o3d.visualization.draw_geometries([obj])


#-------------------------------------------------------------------------------
def voxelization(filename, voxel_size):
    '''converts a pcd file and returns PCD and VoxelGrid object'''

    pcd = o3d.io.read_point_cloud(filename)
    pcd = pcd.voxel_down_sample(voxel_size=0.02) # 0.01 ~ 5 mins; 0.009
    # Normalize point cloud points
    xyz = np.array(pcd.points)
    points = xyz - np.expand_dims(np.min(xyz, axis=0), 0)

    # create new point cloud object
    pcd = o3d.geometry.PointCloud()
    pcd.points = o3d.utility.Vector3dVector(points)
    N = len(pcd.points)

    # voxelization
    pcd.colors = o3d.utility.Vector3dVector(np.random.uniform(0, 1, size=(N,␣
 ↪3)))
    voxel_grid = o3d.geometry.VoxelGrid.create_from_point_cloud(pcd,␣
 ↪voxel_size=voxel_size)

    return pcd,voxel_grid


#-------------------------------------------------------------------------------
def voxels_to_img3d(voxel):
    '''return 3d array of pixel values from voxel object'''

    voxels = voxel.get_voxels()
    indices = np.stack(list(vx.grid_index for vx in voxels))
    colors = np.stack(list(vx.color for vx in voxels))
```

```
    min_, max_ = 1000000, -100000
    for index in indices:
        min_ = min(min_, index[0], index[1], index[2])
        max_ = max(max_, index[0], index[1], index[2])

    max_ += 1

    img3d = np.ones((max_, max_, max_))
    for i in range(len(indices)):
        x, y, z = indices[i]
        img3d[x,y,z] = 0

    return img3d


#-------------------------------------------------------------------------------
def save_array(array, output, plant_name, tag):

    outpath = os.path.join(os.getcwd(), output, "arrays", plant_name)

    if not os.path.isdir(outpath):
        os.makedirs(outpath)

    np.save(os.path.join(outpath, '_'.join([plant_name, tag])), array)


#-------------------------------------------------------------------------------
def normalize_pc(points):

    min_max_scaler = preprocessing.MinMaxScaler()
    minmax = min_max_scaler.fit_transform(points)

    return minmax


#-------------------------------------------------------------------------------
def get_market_plant(data_path, ecc_path):

    result_dict = {}
    cnt = 0

    for item in glob.glob(data_path):
        cnt += 1
        market_type, plant_name = item.split('/')[-2:]

        result_dict[cnt] = {
            'market_type': market_type,
```

```
            'plant_name': plant_name
        }

    map_df = pd.DataFrame.from_dict(result_dict, orient='index')

    csv_list = glob.glob(ecc_path)
    df = pd.concat([pd.read_csv(csv) for csv in csv_list]).reset_index()
    df = map_df.merge(df, on='plant_name', how='inner')

    return df


#-------------------------------------------------------------------------------
def z_score_outlier(df):
    # Get unique genotypes and treatments
    genotypes = df['genotype'].unique()
    treatments = df['treatment'].unique()

    # Create an empty dataframe to store the filtered data
    filt_df_no_outliers = [] #pd.DataFrame()

    # Loop through each genotype and treatment
    for genotype in genotypes:
        for treatment in treatments:
            # Get data for the current genotype and treatment
            data = df[(df['genotype'] == genotype) & (df['treatment'] ==␣
 ↪treatment)]

            # Calculate Z-scores
            z_scores = stats.zscore(data['hull_volume'])

            # Check for NaN values in z_scores
            if np.isnan(z_scores).any():
                print(f"Warning: NaN values found in z_scores for genotype␣
 ↪{genotype} and treatment {treatment}")
                continue

            # Identify outliers
            outliers = (z_scores > 3) | (z_scores < -3)

            # Remove outliers
            data_no_outliers = data[~outliers]

            # Append the filtered data to the new dataframe
            filt_df_no_outliers.append(data_no_outliers)

    filt_df_no_outliers = pd.concat(filt_df_no_outliers)
```

```
    return filt_df_no_outliers


#-------------------------------------------------------------------------------
def calculate_dap(df, planting_date='2019-11-13'):
    planting_date = datetime.strptime(planting_date, '%Y-%m-%d')
    df['date'] = pd.to_datetime(df['date'])
    df['DAP'] = (df['date'] - planting_date).dt.days
    return df


#-------------------------------------------------------------------------------
def get_ecc_pd_csv():

    df = pd.read_csv('https://data.cyverse.org/dav-anon/iplant/projects/
 ↪phytooracle/season_10_lettuce_yr_2020/level_4/scanner3DTop/ecc_pd.csv')
    desc_columns, wg_columns, morph_columns, pd_columns, ecc_columns =␣
 ↪get_ecc_column_names(df=df)
    # df = df.dropna(subset=['genotype']).reset_index()

    genotype_list = [
        'Emperor',
        'Grand Rapids',
        'Green Towers',
        'Iceberg',
        'La Brillante',
        'Lolla Rosa',
        'Margarita',
        'Merlot',
        'Ninja',
        'Salad Bowl',
        'Salinas',
        'Valmaine'
    ]
    df['genotype'] = df['genotype'].str.strip()
    df['genotype'] = df['genotype'].replace('_', ' ')
    df = df[df['genotype'].isin(genotype_list)]

    return df


#-------------------------------------------------------------------------------
def get_ecc_column_names(df):

    desc_columns = [
```

```
        'double_lettuce', 'treatment', 'plot', 'plant_name', 'crop_type',␣
 ↪'origin_country', 'population_type', 'Cultivar type', 'genotype', 'date'
    ]

    wg_columns = [
        'Leaf thickness', 'Leaf blistering',
        'Leaf margin undulation', 'Leaf venation', 'Leaf division',
        'Leaf tip shape', 'Leaf color', 'Leaf color intensity',
        'Leaf anthocyanin content', 'Leaf anthocyanin distribution',
        'Leaf anthocyanin pattern', 'Plant diameter', 'Head shape',
        'Head leafs overlap', 'Head height', 'Heart formation',
        'Side shoot formation tendency'
    ]

    morph_columns = [
#         'min_x', 'min_y', 'min_z',
#         'max_x', 'max_y', 'max_z',
        'num_points', 'length', 'width', 'height',
        'hull_volume', 'oriented_bounding_box', 'axis_aligned_bounding_box',
    ]

    pd_columns = [
        'persistence_entropy_0', 'persistence_entropy_1',␣
 ↪'persistence_entropy_2',
        'number_points_0', 'number_points_1', 'number_points_2',
        'amplitude_landscape_0', 'amplitude_landscape_1',␣
 ↪'amplitude_landscape_2',
        'amplitude_bottleneck_0', 'amplitude_bottleneck_1',␣
 ↪'amplitude_bottleneck_2',
        'amplitude_wasserstein_0', 'amplitude_wasserstein_1',␣
 ↪'amplitude_wasserstein_2',
        'amplitude_betti_0', 'amplitude_betti_1', 'amplitude_betti_2',
        'amplitude_silhouette_0', 'amplitude_silhouette_1',␣
 ↪'amplitude_silhouette_2',
        'amplitude_heat_0', 'amplitude_heat_1', 'amplitude_heat_2',
        'amplitude_persistence_image_0', 'amplitude_persistence_image_1',␣
 ↪'amplitude_persistence_image_2'
      ]

    ecc_columns = []

    for column in df.columns:

        try:
            column_float = float(column)
            ecc_columns.append(column)
```

```
        except:
            pass

    return desc_columns, wg_columns, morph_columns, pd_columns, ecc_columns


#-----------------------------------------------------------------------------
```

2.0.1 Create output directory

[ ]:
```
plot_outpath = os.path.join("kpca", "combined")

if not os.path.isdir(plot_outpath):
    os.makedirs(plot_outpath)
```

2.0.2 Download example point cloud data

[ ]:
```
%%capture
!wget https://data.cyverse.org/dav-anon/iplant/projects/phytooracle/
  ↪season_10_lettuce_yr_2020/level_4/scanner3DTop/market_types_examples.tar.gz
!tar -xvf market_types_examples.tar.gz && rm market_types_examples.tar.gz

pcd_list = glob.glob('./market_types/*/*/final.ply')
```

[ ]:
```
pcd_list
```

[ ]:
```
['./market_types/Butterhead/Titan_7/final.ply',
 './market_types/Butterhead/Blondine_109/final.ply',
 './market_types/Crisp/Bedford_36/final.ply',
 './market_types/Crisp/Angie_110/final.ply',
 './market_types/Crisp/Pacific_100/final.ply',
 './market_types/Crisp/Argeles_21/final.ply',
 './market_types/Crisp/Batavia_Rouge_Grenobloise_130/final.ply',
 './market_types/Cutting/Bartoli_38/final.ply',
 './market_types/Cutting/Grenadine_86/final.ply',
 './market_types/Cutting/Colorado_147/final.ply',
 './market_types/Cutting/Ruby_239/final.ply',
 './market_types/Cutting/Slobolt_73/final.ply',
 './market_types/Latin/Sucrine_95/final.ply',
 './market_types/Latin/Fordhook_70/final.ply',
 './market_types/Latin/Aido_95/final.ply',
 './market_types/Cos/Odra_57/final.ply',
 './market_types/Cos/Green_Towers_3/final.ply',
 './market_types/Cos/Green_Towers_98/final.ply',
```

```
 './market_types/Cos/Valmaine_73/final.ply']
```

2.0.3 Visualize point cloud

[ ]:
```python
# Load the point cloud using Open3D
# pcd = o3d.io.read_point_cloud('./market_types/Crisp/Angie_110/final.ply')
pcd = o3d.io.read_point_cloud('./market_types/Butterhead/Blondine_109/ml_crop.
 ↪ply')

# Downsample the point cloud
pcd = pcd.voxel_down_sample(voxel_size=0.001)

# Normalize point cloud points
xyz = np.array(pcd.points)
points = xyz - np.expand_dims(np.min(xyz, axis=0), 0)

# Create new point cloud object
pcd = o3d.geometry.PointCloud()
pcd.points = o3d.utility.Vector3dVector(points)

# Convert the point cloud to a NumPy array
pcd_np = np.asarray(pcd.points)

# Create a DataFrame from the NumPy array
df = pd.DataFrame(pcd_np, columns=['x', 'y', 'z'])

# Add a column to the DataFrame to use for coloring the points
df['color'] = df['z']

# Create a 3D scatter plot using Plotly Express with smaller point sizes and a␣
 ↪colormap
fig = px.scatter_3d(df, x='x', y='y', z='z', color='color',␣
 ↪color_continuous_scale='Inferno') #Viridis, Magma, Plasma, Cividis, Inferno
fig.update_traces(marker=dict(size=3))
fig.show()
```

2.0.4 Extraction of Euler characteristic curves

[ ]:
```python
#-------------------------------------------------------------------------------
def euler_char_curves(points, img3d, save, output, plant_name, visualize_ecc):
    '''extract ecc from set of points and array of pixel values of 3d image'''

    outpath = os.path.join(os.getcwd(), output, "figures", plant_name)

    if not os.path.isdir(outpath):
        os.makedirs(outpath)
```

```
    simplices_3D, alpha_3D = alpha_filtration_3D(points)
    bins_3D = np.linspace(0.0, 1, num=200)

    filt_3D = filtration(simplices_3D, alpha_3D, bins_3D)

    if visualize_ecc:
        plt.show()

    if save:
        outpath = os.path.join(os.getcwd(), output, "dataframes", plant_name)

        if not os.path.isdir(outpath):
            os.makedirs(outpath)

        df = pd.DataFrame({'bin': bins_3D, 'filter': filt_3D})
        df['plant_name'] = plant_name
        df = df.set_index('plant_name')
        df.to_csv(os.path.join(outpath, '_'.join([plant_name, "ecc_long.
 ↪csv"])), index=True)

        df = df.pivot(columns='bin')
        df.columns = df.columns.droplevel(0)
        df.to_csv(os.path.join(outpath, '_'.join([plant_name, "ecc_wide.
 ↪csv"])), index=True)


#-------------------------------------------------------------------------------
def run(file, p, v, voxel_size, save, output, plant_name, visualize_ecc):
    '''visualize 3d image file as pcd and voxels, extract and display ecc␣
 ↪plots'''

    pcd,voxel = voxelization(file, voxel_size)

    if p:
        visualize(pcd)
    if v:
        visualize(voxel)
    img3d = voxels_to_img3d(voxel)
    points = np.asarray(pcd.points)
    points = normalize_pc(points)
    euler_char_curves(points=points, img3d=img3d, save=save, output=output,␣
 ↪plant_name=plant_name, visualize_ecc=visualize_ecc)


#-------------------------------------------------------------------------------
```

[ ]:
```
for pcd in pcd_list:
    run(file=pcd, p=False, v=False, voxel_size=0.001, save=True,␣
 ↪output="euler_characteristic_curves", plant_name=os.path.dirname(pcd).
 ↪split('/')[-1], visualize_ecc=False)
```

[ ]:
```
df = get_market_plant(data_path='./market_types/*/*', ecc_path='./
 ↪euler_characteristic_curves/dataframes/*/*_long.csv')
```

2.0.5 Extraction of traditional phenotypes

[ ]:
```
# ----------------------------------------------------
def calculate_convex_hull_volume(pcd):
    hull, _ = pcd.compute_convex_hull()
    hull_volume = hull.get_volume()
    # print(f'Volume: {hull_volume}')
    return hull_volume


# ----------------------------------------------------
def calculate_oriented_bb_volume(pcd):

    obb_vol = pcd.get_oriented_bounding_box().volume()

    return obb_vol


# ----------------------------------------------------
def calculate_axis_aligned_bb_volume(pcd):

    abb_vol = pcd.get_axis_aligned_bounding_box().volume()

    return abb_vol


# ----------------------------------------------------
def get_min_max(pcd):

    max_x, max_y, max_z = pcd.get_max_bound()
    min_x, min_y, min_z = pcd.get_min_bound()

    return max_x, max_y, max_z, min_x, min_y, min_z


# ----------------------------------------------------
```

[ ]:
```
# Open point cloud
pcd = o3d.io.read_point_cloud('./market_types/Butterhead/Blondine_109/ml_crop.
 ↪ply')

# Downsample the point cloud
pcd = pcd.voxel_down_sample(voxel_size=0.001)

# Get min, max coordinates
max_x, max_y, max_z, min_x, min_y, min_z = get_min_max(pcd)

# Calculate plant hull volumes and bounding box volumes
hull_vol = calculate_convex_hull_volume(pcd)

obb_vol = calculate_oriented_bb_volume(pcd)

abb_vol = calculate_axis_aligned_bb_volume(pcd)
```

[ ]:
```
np.array(pcd.points)
```

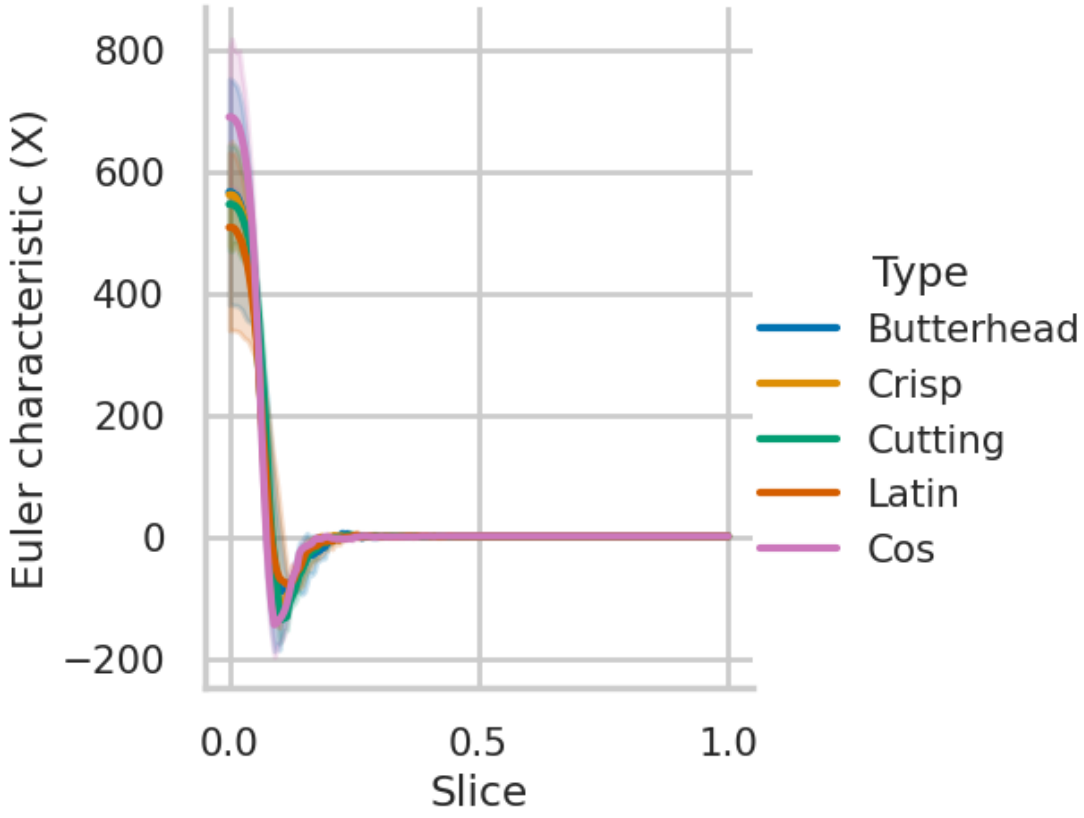
[ ]:
```
array([[4.08998044e+05, 3.65998595e+06, 1.04977991e+00],
       [4.08998297e+05, 3.65998609e+06, 1.02153652e+00],
       [4.08998066e+05, 3.65998600e+06, 1.08321960e+00],
       …,
       [4.08998095e+05, 3.65998591e+06, 1.10778027e+00],
       [4.08998095e+05, 3.65998591e+06, 1.10792871e+00],
       [4.08998265e+05, 3.65998590e+06, 1.07598751e+00]])
```

[ ]:
```
print(f'Convex hull volume (CH): {hull_vol}\nOriented bounding box volume (OBB):
 ↪ {obb_vol}\nAxis-aligned bounding box volume (AABB): {abb_vol}')
```

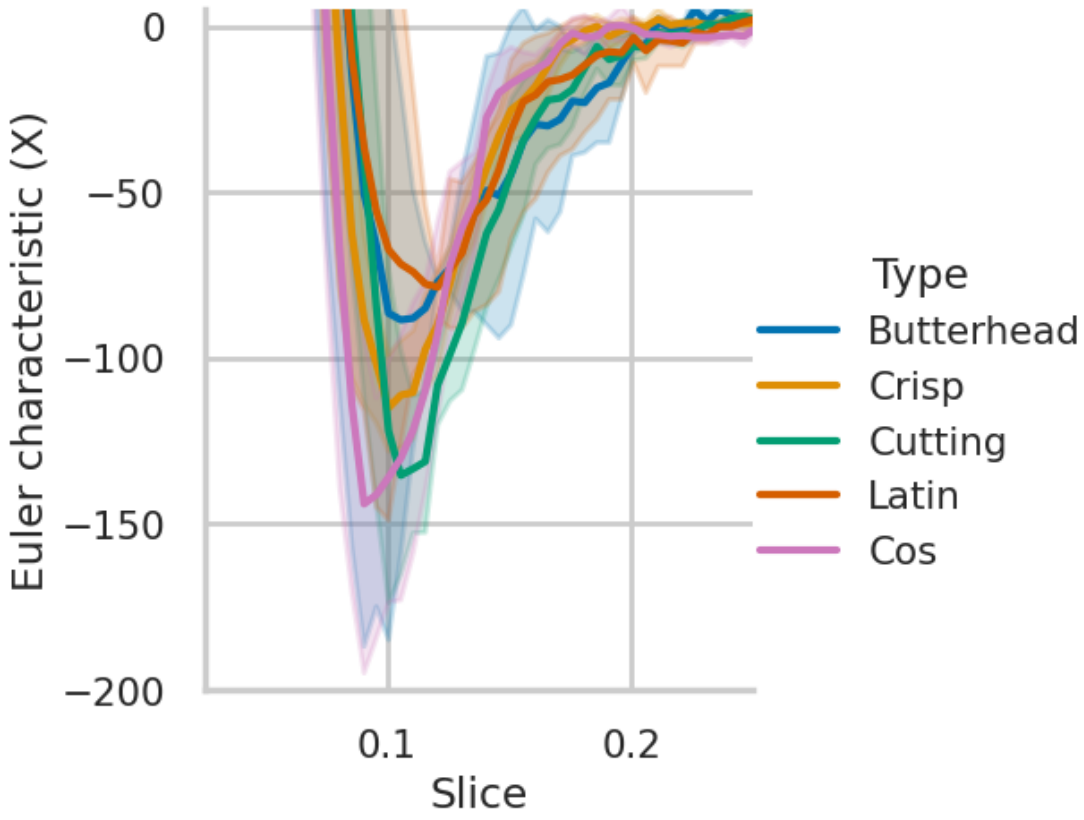
```
Convex hull volume (CH): 0.011479751494618995
Oriented bounding box volume (OBB): 0.027956235319766855
Axis-aligned bounding box volume (AABB): 0.01850026925312912
```

2.0.6 Visualization of Euler characteristic curves

[ ]:
```
g = sns.relplot(x='bin', y='filter', hue='market_type', kind='line', data=df)
g.set(xlabel='Slice', ylabel='Euler characteristic (X)')
g._legend.set_title('Type')
```



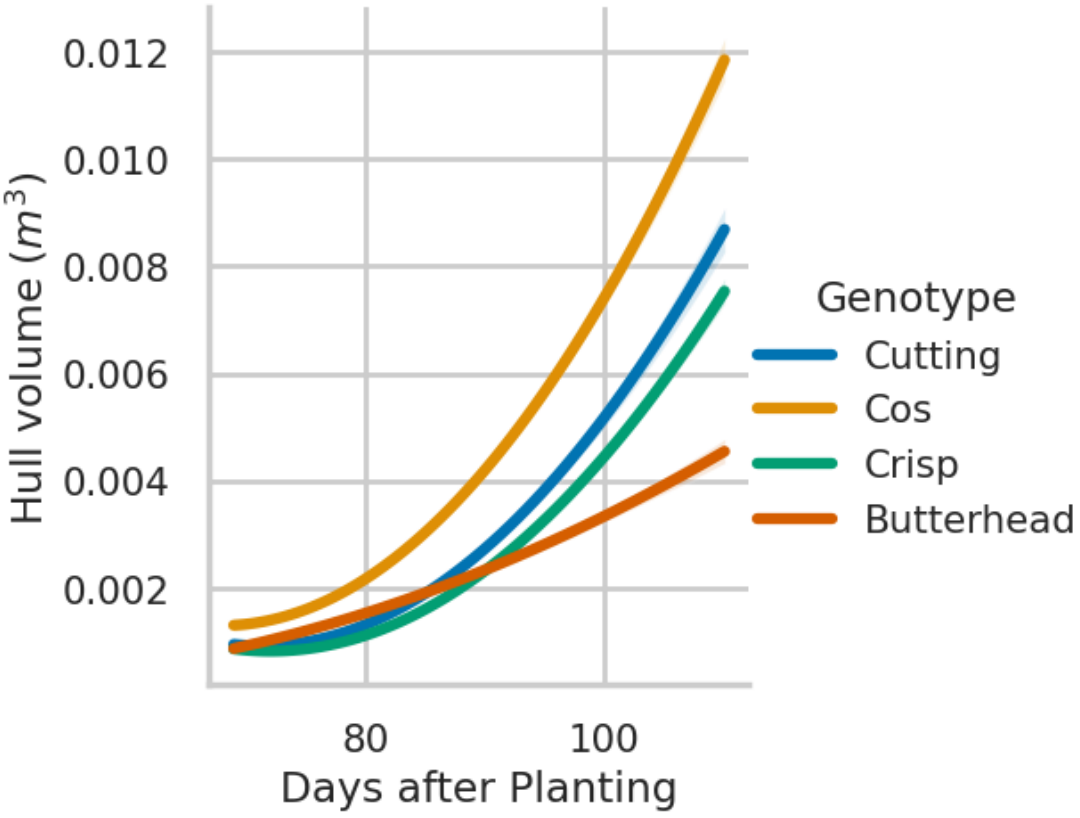
[ ]:
```
g = sns.relplot(x='bin', y='filter', hue='market_type', kind='line', data=df)
g.set(xlim=(0.025, 0.25), ylim=(-200, 5), xlabel='Slice', ylabel='Euler␣
↪characteristic (X)')
g._legend.set_title('Type')
```



2.0.7 Visualization of traditional *phenotypes*

[ ]:
```
df = get_ecc_pd_csv()
desc_columns, wg_columns, morph_columns, pd_columns, ecc_columns =␣
↪get_ecc_column_names(df=df)
df = z_score_outlier(df = df)
df = calculate_dap(df = df)
```

[ ]:
```
g = sns.lmplot(x='DAP', y='hull_volume', hue='crop_type', scatter=False,␣
↪order=2, data=df)
g.set_axis_labels('Days after Planting', 'Hull volume ($m^3$)')
g._legend.set_title('Genotype')
```



3 2 | Principal component analysis (PCA)

In this section we will run PCA to assess the feasibility of leveraging traditional and TDA shape descriptors for classification.

Click on "Show code" to see all functions for this section. Otherwise, click the Run Cell icon.

[ ]:
```
#@title
#-------------------------------------------------------------------------------
def run_pca(df, features):
  # Separating out the features
  x = df.loc[:, features].values

  # Separating out the target
  y = df.loc[:,['genotype', 'treatment', 'crop_type']].values

  # Standardizing the features
  x = StandardScaler().fit_transform(x)

  # Run PCA
  pca = PCA(n_components=2)
```

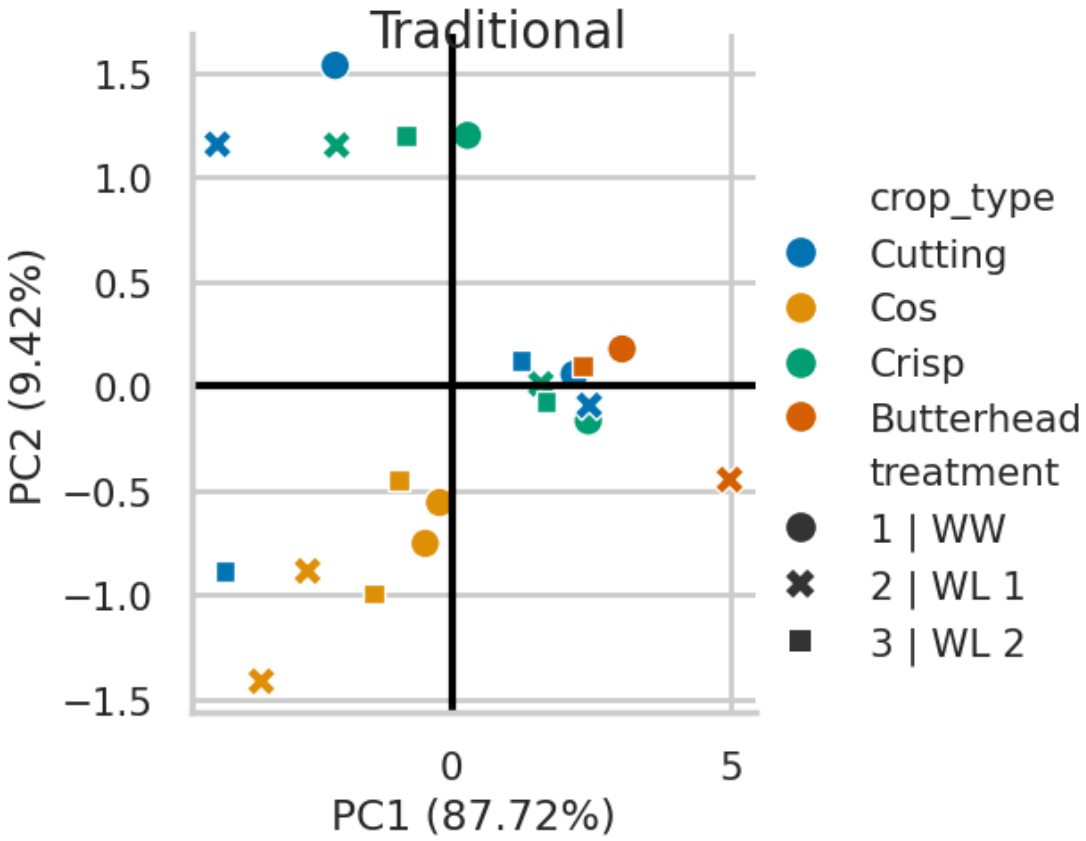
```
  principalComponents = pca.fit_transform(x)
  pc1_ev, pc2_ev = pca.explained_variance_ratio_
  pc1_ev = round(pc1_ev*100, 2)
  pc2_ev = round(pc2_ev*100, 2)
  principalDf = pd.DataFrame(data = principalComponents
             , columns = ['principal component 1', 'principal component 2'])
  result_df = pd.concat([pd.DataFrame(y, columns=['genotype', 'treatment',␣
 ↪'crop_type']), principalDf], axis=1)

  return result_df, pc1_ev, pc2_ev


#------------------------------------------------------------------------------
```

[ ]:
```
temp_df = df.dropna(subset=['genotype'])
temp_df = df[df['date'].isin(['2020-03-01', '2020-03-02'])]
temp_df = temp_df[morph_columns + ecc_columns + ['treatment', 'genotype',␣
 ↪'crop_type']]
temp_df = temp_df.groupby(by=['treatment', 'genotype', 'crop_type']).median().
 ↪reset_index()
```

3.0.1 Analysis of traditional phenotypes

[ ]:
```
result_df, pc1_ev, pc2_ev = run_pca(df=temp_df, features=morph_columns)

g = sns.relplot(x=f'principal component 1', y='principal component 2',␣
 ↪hue='crop_type', style='treatment', data=result_df, kind='scatter')
g.set_xlabels(f'PC1 ({pc1_ev}%)')
g.set_ylabels(f'PC2 ({pc2_ev}%)')
g.fig.suptitle('Traditional')
plt.axhline(0, ls='-', c='black')
plt.axvline(0, ls='-', c='black')
```

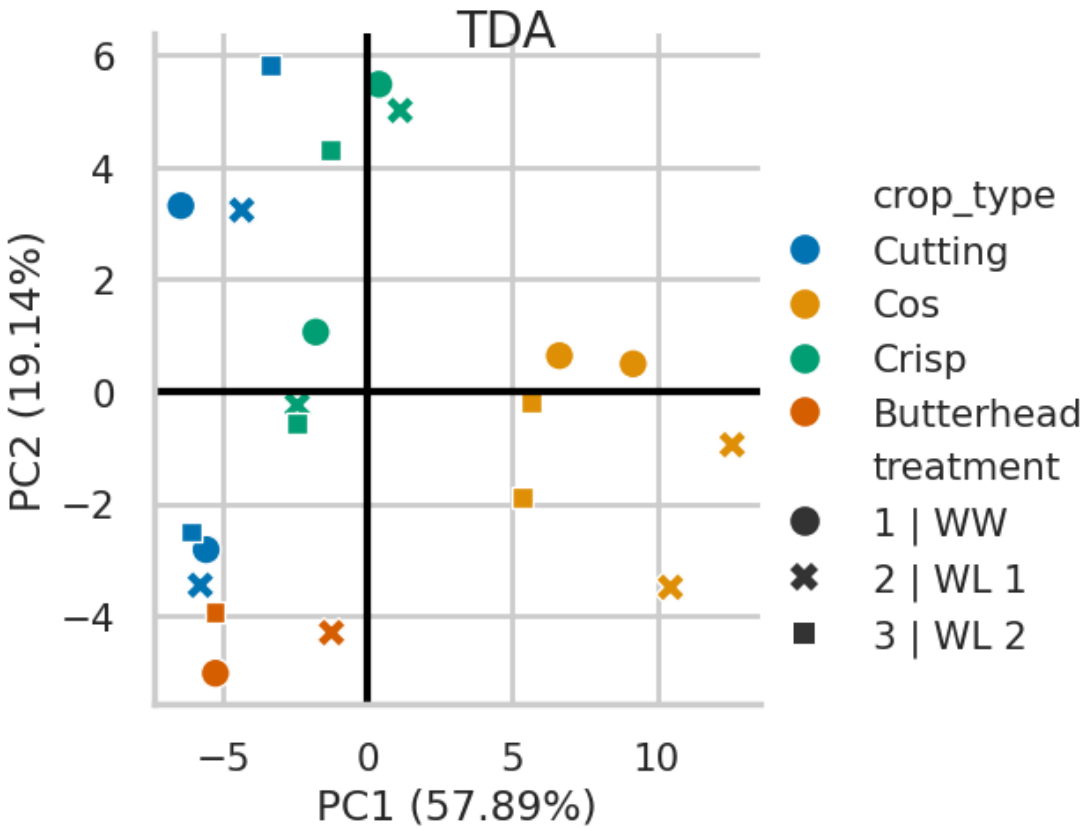
[ ]:
```
<matplotlib.lines.Line2D at 0x7ad7c1eedcc0>
```



3.0.2 Analysis of Euler characteristic curves

[ ]:
```
result_df, pc1_ev, pc2_ev = run_pca(df=temp_df, features=ecc_columns)

g = sns.relplot(x=f'principal component 1', y='principal component 2',␣
↪hue='crop_type', style='treatment', data=result_df, kind='scatter')
g.set_xlabels(f'PC1 ({pc1_ev}%)')
g.set_ylabels(f'PC2 ({pc2_ev}%)')
g.fig.suptitle('TDA')
plt.axhline(0, ls='-', c='black')
plt.axvline(0, ls='-', c='black')
```

[ ]:
```
<matplotlib.lines.Line2D at 0x7ad7bf4899c0>
```



Traditional PCA * Phenotypes: height, volumes * Results: Much stronger first principal component (PC1) that explains 87.72% of the variance in the data. This suggests that there is a dominant pattern in the data that can be captured by this first principal component.

Topological data analysis PCA * Phenotypes: Euler characteristic * Results: More evenly distributed variance explained by its first two principal components (PC1 and PC2), with 57.89% and 19.14% respectively. This suggests that there are multiple patterns in the data that are being captured by these two principal components.

4 3 | Classification

Click on "Show code" to see all functions for this section. Otherwise, click the Run Cell icon.

[ ]:
```
#@title
#-------------------------------------------------------------------------------
def create_df(csv):

    date = get_date_from_string(string=csv)

    df = pd.read_csv(csv)
    df['date'] = date
```

```
    return df

#-------------------------------------------------------------------------------
def get_date_from_string(string):

    match_str = re.search(r'\d{4}-\d{2}-\d{2}', string)

    # computed date
    # feeding format
    res = datetime.strptime(match_str.group(), '%Y-%m-%d').date()

    # printing result
    date = str(res)

    return date

#-------------------------------------------------------------------------------
def datetime_column(df):

    df['date'] = pd.to_datetime(df['date'])

    return df

#-------------------------------------------------------------------------------
def get_df(data_path, out_file):

    if not os.path.isfile(out_file):

        csv_list = glob.glob(data_path)

        df = pd.DataFrame()

        with multiprocessing.Pool(int(multiprocessing.cpu_count()*0.80)) as p:

            temp_df = p.map(create_df, csv_list)
            df = df.append(temp_df)

        df.to_csv(out_file, index=False)

    else:
        df = pd.read_csv(out_file)

    return df

#-------------------------------------------------------------------------------
def add_passport_data(df):
```

```
   # Open passport data
   ps = pd.read_excel('https://cgn.websites.wur.nl/Website/downloads/download/
↪Cnr06Passport.xlsx', sheet_name='data')\
                .rename(columns={'ACCNAME': 'genotype',
                                 'SUBTAXA': 'subtaxa',
                                 'ORIGCTY': 'origin_country'})\
                [['genotype', 'subtaxa', 'origin_country', 'SAMPSTAT']]
   ps['SAMPSTAT'] = ps['SAMPSTAT'].astype(str)
   ps = ps.set_index('SAMPSTAT')
   ps['subtaxa'] = ps['subtaxa'].str.replace('group ', '')
   ps['subtaxa'] = ps['subtaxa'].str.replace(' Lettuce', '')
   ps['subtaxa'] = ps['subtaxa'].str.strip()
   ps['genotype'] = ps['genotype'].str.strip()
   #ps = ps.dropna(subset=['genotype'])


   # Get sample status
   sub = pd.read_excel('https://cgn.websites.wur.nl/Website/downloads/download/
↪Cnr06Passport.xlsx', sheet_name='sample status')
   sub['Code'] = sub['Code'].astype(float).astype(str)
   sub = sub.rename(columns={'Code': 'genotype'})
   #sub = sub.set_index('Code')

   # Combine the two dataframes
   result = ps.merge(sub, on='genotype', how='outer')#, left_index=True,␣
↪right_index=True)#.dropna()
   result = result.rename(columns={'Description': 'population_type',
                           'subtaxa': 'crop_type'}).reset_index()

   # Get additional data
   wg = pd.read_excel('https://www.wur.nl/upload_mm/d/d/7/
↪e1651268-a8a6-4d52-aea6-54ad103dee4d_Data_CGNSC002%20%282022-08-23%29.xlsx',
                      sheet_name='Accession data')\
                       [['Population type', 'Name', 'Crop type', 'Country',␣
↪'Cultivar type',
                         'Leaf thickness', 'Leaf blistering', 'Leaf margin␣
↪undulation', 'Leaf venation', 'Leaf division', 'Leaf tip shape',
                         'Leaf color', 'Leaf color intensity', 'Leaf␣
↪anthocyanin content', 'Leaf anthocyanin distribution', 'Leaf anthocyanin␣
↪pattern',
                         'Plant diameter', 'Head shape', 'Head leafs overlap',␣
↪'Head height', 'Heart formation', 'Side shoot formation tendency']]\
                      .rename(columns={'Population type': 'population_type',
                                       'Crop type': 'crop_type',
                                       'Country': 'origin_country',
```

```
                                        'Name': 'genotype'}).reset_index()
    wg['crop_type'] = wg['crop_type'].str.strip()
    wg['genotype'] = wg['genotype'].str.strip()
    info_df = pd.concat([result, wg], ignore_index=False)

    combined = info_df.merge(df, on='genotype', how='outer')#, left_index=True,␣
 ↪right_index=True)

    return combined

#-------------------------------------------------------------------------------
def drop_double_plants(df):

    rgb = pd.read_csv('https://data.cyverse.org/dav-anon/iplant/projects/
 ↪phytooracle/season_10_lettuce_yr_2020/level_3/stereoTop/
 ↪stereoTop_full_season_clustering.csv', parse_dates=['date'])
    rgb = rgb[rgb['date']=='2020-03-03']
    rgb = rgb.drop_duplicates(subset=['plant_name'])
    rgb = rgb[['double_lettuce', 'treatment', 'plot', 'plant_name', 'genotype']]
    df = df.set_index('plant_name').merge(rgb.set_index('plant_name'),␣
 ↪left_index=True, right_index=True)#.dropna()
    df = df[df['double_lettuce']==0]
    df = df.reset_index()

    return df

#-------------------------------------------------------------------------------
def clean_df(df, drop_mapping_population, drop_diversity_panel,␣
 ↪genotype_column, passport_data, drop_doubles, treatment_column,␣
 ↪length_width_height):

    if drop_diversity_panel:
        df = df[df['plant_name'].str.contains('GRxI')]

    if drop_mapping_population:
        df = df[~df['plant_name'].str.contains('GRxI')]

    if genotype_column:
        df['genotype'] = df['plant_name'].apply(lambda x: x.split('_')[:-1]).
 ↪str.join(' ')

    if passport_data:
        df = add_passport_data(df)

    if drop_doubles:
        df = drop_double_plants(df)
```

```
    if treatment_column:
        df = fix_treatment_column(df)

    if length_width_height:
        df = add_lwh(df)

    return df

#-------------------------------------------------------------------------------
def fix_treatment_column(df):

    df['treatment'] = df['treatment'].map({
        'treatment 3': '3 | WL 2',
        'treatment 2': '2 | WL 1',
        'treatment 1': '1 | WW'
    })

    df['plot'] = df['plot'].str.split('_', expand=True)[6].str.zfill(2) +␣
 ↪df['plot'].str.split('_', expand=True)[8].str.zfill(2)

    return df

#-------------------------------------------------------------------------------
def add_lwh(df):

    df['length'] = df['max_x'] - df['min_x']
    df['width'] = df['max_y'] - df['min_y']
    df['height'] = df['max_z'] - df['min_z']

    df['genotype'] = df['plant_name'].apply(lambda x: x.split('_')[:-1]).str.
 ↪join(' ')

    return df


#-------------------------------------------------------------------------------
def extract_columns(df, feature_type, kpca_n_comp):

    desc_columns, wg_columns, morph_columns, pd_columns, ecc_columns =␣
 ↪get_ecc_column_names(df=df)

    data_type = {
        'mor': ['length', 'width', 'height',
#                'min_x', 'min_y', 'min_z', 'max_x', 'max_y', 'max_z',
                'num_points', 'oriented_bounding_box',␣
 ↪'axis_aligned_bounding_box', 'hull_volume'],
```

```
        'pd': ['persistence_entropy_0', 'persistence_entropy_1',␣
↪'persistence_entropy_2',
                'number_points_0', 'number_points_1', 'number_points_2',
                'amplitude_landscape_0', 'amplitude_landscape_1',␣
↪'amplitude_landscape_2',
                'amplitude_bottleneck_0', 'amplitude_bottleneck_1',␣
↪'amplitude_bottleneck_2',
                'amplitude_wasserstein_0', 'amplitude_wasserstein_1',␣
↪'amplitude_wasserstein_2',
                'amplitude_betti_0', 'amplitude_betti_1', 'amplitude_betti_2',
                'amplitude_silhouette_0', 'amplitude_silhouette_1',␣
↪'amplitude_silhouette_2',
                'amplitude_heat_0', 'amplitude_heat_1', 'amplitude_heat_2',
                'amplitude_persistence_image_0',␣
↪'amplitude_persistence_image_1', 'amplitude_persistence_image_2'],

        'ecc': ecc_columns

    }

    if feature_type=='mor + pd + ecc':
        df = df[['genotype'] +␣
↪data_type['mor']+data_type['pd']+data_type['ecc']].dropna()
        geno = df[['genotype']]

        mor = df[data_type['mor']]
        #mor = run_kernel_pca(mor, n_comp=kpca_n_comp)

        pdi = df[data_type['pd']]
        pdi = run_kernel_pca(pdi, n_comp=kpca_n_comp)

        ecc = df[data_type['ecc']]
        ecc = run_kernel_pca(ecc, n_comp=kpca_n_comp)

        X = pd.concat([mor.reset_index(), pd.DataFrame(pdi), pd.
↪DataFrame(ecc)], axis=1)
        X = np.asarray(X)

    elif feature_type=='mor + pd':
        df = df[['genotype'] + data_type['mor']+data_type['pd']].dropna()
        geno = df[['genotype']]

        mor = df[data_type['mor']]
        #mor = run_kernel_pca(mor, n_comp=kpca_n_comp)

        pdi = df[data_type['pd']]
```

```
        pdi = run_kernel_pca(pdi, n_comp=kpca_n_comp)

        X = pd.concat([mor.reset_index(), pd.DataFrame(pdi)], axis=1)
        X = np.asarray(X)

    elif feature_type=='mor + ecc':
        df = df[['genotype'] + data_type['mor']+data_type['ecc']].dropna()
        geno = df[['genotype']]

        mor = df[data_type['mor']]
        #mor = run_kernel_pca(mor, n_comp=kpca_n_comp)

        ecc = df[data_type['ecc']]
        ecc = run_kernel_pca(ecc, n_comp=kpca_n_comp)

        X = pd.concat([mor.reset_index(), pd.DataFrame(ecc)], axis=1)
        X = np.asarray(X)

    elif feature_type=='pd + ecc':
        df = df[['genotype'] + data_type['pd']+data_type['ecc']].dropna()
        geno = df[['genotype']]

        pdi = df[data_type['pd']]
        pdi = run_kernel_pca(pdi, n_comp=kpca_n_comp)

        ecc = df[data_type['ecc']]
        ecc = run_kernel_pca(ecc, n_comp=kpca_n_comp)

        X = pd.concat([pd.DataFrame(pdi), pd.DataFrame(ecc)], axis=1)
        X = np.asarray(X)

    elif feature_type=='pd':
        df = df[['genotype'] + data_type['pd']].dropna()
        geno = df[['genotype']]

        pdi = df[data_type['pd']]
        X = run_kernel_pca(pdi, n_comp=kpca_n_comp)

    elif feature_type=='ecc':
        df = df[['genotype'] + data_type['ecc']].dropna()
        geno = df[['genotype']]

        ecc = df[data_type['ecc']]
        X = run_kernel_pca(ecc, n_comp=kpca_n_comp)

    elif feature_type=='mor':
        df = df[['genotype'] + data_type['mor']].dropna()
```

```
        geno = df[['genotype']]

        mor = df[data_type['mor']]
        #X = run_kernel_pca(mor, n_comp=kpca_n_comp)
        X = np.asarray(mor)

    return X, geno.values.ravel()

#-------------------------------------------------------------------------------
def run_kernel_pca(X, n_comp=2):

    X = StandardScaler(with_std=True).fit_transform(X)
    transformer = KernelPCA(n_components=n_comp, kernel='rbf')
    #transformer = umap.UMAP(n_components=n_comp, n_neighbors=50, min_dist=0.1,␣
 ↪metric='manhattan', n_jobs=-1)
    X_transformed = transformer.fit_transform(X)

    return X_transformed

#-------------------------------------------------------------------------------
def annotate(data, **kws):
    r, p = sp.stats.pearsonr(data[X], data["value"])
    R2 = r**2
    ax = plt.gca()
    ax.text(.05, .8, 'r={:.2f}, r$^2$={:.2f}, p={:.2g}'.format(r, R2, p),
            transform=ax.transAxes)

#-------------------------------------------------------------------------------
def get_shared_columns(df1, df2):

    shared_columns = list(set(df1.columns.tolist()).intersection(df2.columns.
 ↪tolist()))
    shared_columns.remove('plot')
    return shared_columns

#-------------------------------------------------------------------------------
def get_phenotype_data(cyverse_path, season):

    if season==10:
        df = pd.read_csv(cyverse_path)
        df['plot'] = df['plot'].str.split('_', expand=True)[6].str.zfill(2).
 ↪astype(str) + df['plot'].str.split('_', expand=True)[8].str.zfill(2).
 ↪astype(str)
#         df = df[df['double_lettuce']==0]
#         df["bounding_area_m2"] = preprocessing.scale(df['bounding_area_m2'].
 ↪values)
```

```
    return df

#-------------------------------------------------------------------------------
def clean_report(report_df):
    report_df['data_type'] = report_df['data_type'].replace({
        'mor + pd + ecc': 'ECC-PD-Morphometric', #'KPCA(ECC-TDA) + KPCA(PD-TDA)␣
 ↪+ Morphometric',
        'mor + pd': 'PD-Morphometric', #'KPCA(PD-TDA) + Morphometric',
        'mor + ecc': 'ECC-Morphometric', #'KPCA(ECC-TDA) + Morphometric',
        'pd + ecc': 'ECC-PD', #'KPCA(ECC-TDA) + KPCA(PD-TDA)',
        'pd': 'PD', #'KPCA(PD-TDA)',
        'ecc': 'ECC', #'KPCA(ECC-TDA)',
        'mor': 'Morphometric'
    })

    report_df = report_df.rename(columns={
        'number_components': 'Number of components',
        'name': 'Classifier'
    })
    report_df = report_df[~report_df['date'].isin(['2020-02-01', '2020-02-29',␣
 ↪'2020-01-22', '2020-02-29'])]

    return report_df


#-------------------------------------------------------------------------------
```

4.0.1 Classify point clouds based on Euler characteristic curves and traditional phenotypes

[ ]:
```
#-------------------------------------------------------------------------------
def run_classifiers(X, y):
    names = [
        "Decision Tree",
        "Random Forest",
        "Naive Bayes",
        "Linear Discriminant Analysis",
        "k-Nearest Neighbors",
        "Support Vector Machines",
        "Logistic Regression",
        "Gradient Boosting"
    ]

    classifiers = [
        DecisionTreeClassifier(),
        RandomForestClassifier(oob_score=True, random_state=42, n_jobs=-1),
```

```
        GaussianNB(),
        LinearDiscriminantAnalysis(),
        KNeighborsClassifier(),
        SVC(),
        LogisticRegression(),
        GradientBoostingClassifier()
    ]

    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42
    ) #0.4

    name_list = []
    score_list = []
    classification_reports = []
    cnt = 0

    for name, clf in zip(names, classifiers):
        cnt += 1

        clf.fit(X_train, y_train)
        score = clf.score(X_test, y_test)
        print(f'Accuracy of {name}: {score}')

        y_pred = clf.predict(X_test)
        report_df = classification_report(y_test, y_pred, output_dict=True)
        report_df = pd.DataFrame(report_df).transpose()
        report_df['name'] = name
        report_df['score'] = score

        name_list.append(name)
        score_list.append(score)
        classification_reports.append(report_df)

    score_dict = dict(zip(name_list, score_list))
    report = pd.concat(classification_reports)

    return score_dict, report


#-------------------------------------------------------------------------------
```

[ ]:
```
if not os.path.isfile(os.path.join(plot_outpath, 'classification_results_kpca.
↪csv')):

    cnt = 0
    result_dict = {}
```

```
    report_list = []

    # Iterate through each collection date
    for date in df['date'].unique():

        # Process each data type
        for item in ['mor + ecc', 'mor', 'ecc']:
            for n in [2, 3, 4]:
                print(f'Processing {date} | {item} | {n}' )

                try:

                    cnt += 1
                    # Isolate a single dat
                    date_df = df[df['date']==date]

                    # Extract specific columns (morphometric, TDA, or combined)␣
↪and run KPCA on TDA columns
                    X, geno = extract_columns(df=date_df, feature_type=item,␣
↪kpca_n_comp=n)

                    # Run various classifiers
                    score_dict, report = run_classifiers(X=X, y=geno)
                    report['date'] = date
                    report['number_components'] = n
                    report['number_plants'] = len(X)
                    report['data_type'] = item
                    report_list.append(report)

                    # Collect results
                    result_dict[cnt] = {
                        'date': pd.to_datetime(date),
                        'number_components': n,
                        'number_plants': len(X),
                        'data_type': item
                    }
                    result_dict[cnt].update(score_dict)
                    print('\n')
                except:
                    pass

    report_df = pd.concat(report_list)
    report_df.index.name = 'genotype'
    report_df = report_df.reset_index()
    report_df = report_df[~report_df['genotype'].isin(['accuracy', 'macro avg',␣
↪'weighted avg'])]
```

```
    report_df.to_csv(os.path.join(plot_outpath,␣
 ↪'classification_report_genotype_kpca.csv'), index=False)
    report_df['date'] = pd.to_datetime(report_df['date'])

else:

    report_df = pd.read_csv(os.path.join(plot_outpath,␣
 ↪'classification_report_genotype_kpca.csv'))
    report_df['date'] = pd.to_datetime(report_df['date'])

# Clean the results
report_df = clean_report(report_df)
```

[ ]:
```
report_df.tail()
```

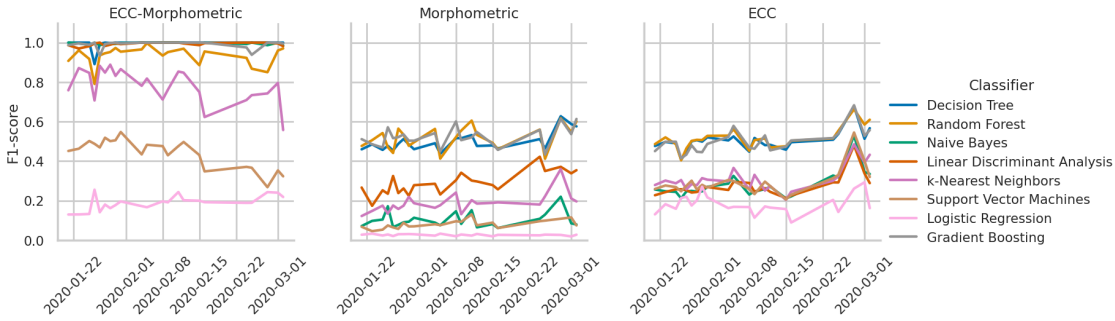
[ ]:

	genotype	precision	recall	f1-score	support	Classifier	\
24510	Merlot	0.250000	1.0	0.400000	1.0	Gradient Boosting	
24511	Ninja	0.625000	1.0	0.769231	10.0	Gradient Boosting	
24512	Salad Bowl	0.875000	1.0	0.933333	7.0	Gradient Boosting	
24513	Salinas	0.736842	1.0	0.848485	14.0	Gradient Boosting	
24514	Valmaine	0.875000	0.7	0.777778	20.0	Gradient Boosting	

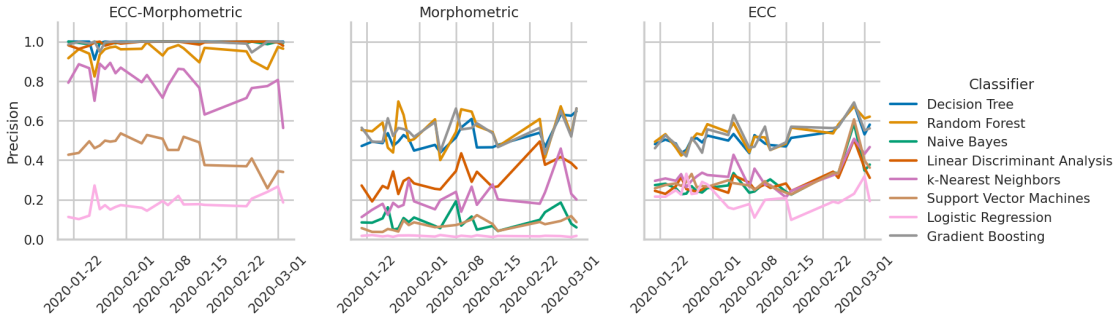
	score	date	Number of components	number_plants	data_type
24510	0.710526	2020-03-01	4	569	ECC
24511	0.710526	2020-03-01	4	569	ECC
24512	0.710526	2020-03-01	4	569	ECC
24513	0.710526	2020-03-01	4	569	ECC
24514	0.710526	2020-03-01	4	569	ECC

4.0.2 Visualize classifier performance

[ ]:
```
g = sns.relplot(x='date', y='f1-score', hue='Classifier', col='data_type',␣
 ↪ci=None, kind='line', data=report_df, facet_kws={'sharey': True, 'sharex':␣
 ↪True})
g.set_xticklabels(rotation=45)
g.set_titles(row_template = '{row_name}', col_template = '{col_name}')
g.set_xlabels('')
g.set_ylabels('F1-score')
g.set(ylim=(0, 1.1));
```



[ ]:
```
g = sns.relplot(x='date', y='precision', hue='Classifier', col='data_type',␣
↪ci=None, kind='line', data=report_df, facet_kws={'sharey': True, 'sharex':␣
↪True})
g.set_xticklabels(rotation=45)
g.set_titles(row_template = '{row_name}', col_template = '{col_name}')
g.set_xlabels('')
g.set_ylabels('Precision')
g.set(ylim=(0, 1.1));
```



[ ]:
```

```