

# Customizable Analysis and Visualization Tool for COVID Cases

## Milestone 5

# Team Members

- ▶ Calvin Burns, [cburns2017@my.fit.edu](mailto:cburns2017@my.fit.edu) (Team Lead)
- ▶ Sam Hartle, [shartle2017@my.fit.edu](mailto:shartle2017@my.fit.edu)
- ▶ Nicole Wright, [nwright2017@my.fit.edu](mailto:nwright2017@my.fit.edu)
- ▶ Stian Olsen, [shagboeolsen2017@my.fit.edu](mailto:shagboeolsen2017@my.fit.edu)

# Faculty Advisor/Client

- ▶ Dr. Philip Chan, [pkc@cs.fit.edu](mailto:pkc@cs.fit.edu)

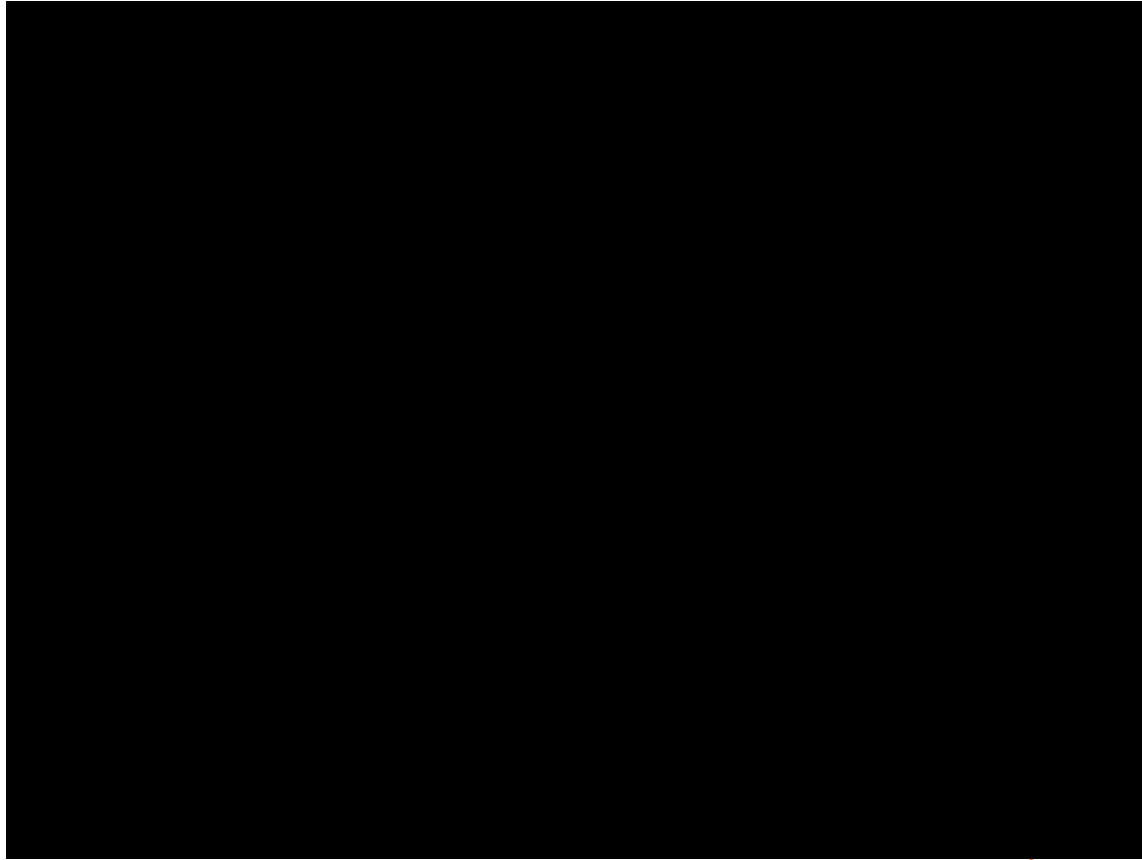
# Progress Matrix

Task	Completion %	Stian	Sam	Nicol e	CJ	To do
1) Continue work on scatter plot	100%	100%	-	-	-	None
2) Operations card update to be more intuitive	80%	-	-	-	80%	
3) Save unique workspaces	0%	-	-	-	-	All
4) Layering plots	80%	80%	-	-	-	Change UI so user can create multiple plots
5) API auto updates for datasets	50%	-	50%	-	-	Convert/append JSON object received from API endpoint to a current CSV
6) Finish Application Feature	100%	-	-	80%	20%	None

# Task 1 - Continue work on scatter plot

- Finding a good use for the scatter plot and finishing up the plot utility.
- Scatter plot to see if there is a correlation between positivity rate and mobility during the pandemic in Florida and in the USA.
- We also encountered a problem with partial weeks when we were resampling the data.

# Demo Task 1



## Task 2 - Update Operations Card

- Suggestion by Dr. Chan to use a similar format to SQL for querying and manipulating a dataset
- To move closer to the SQL method, we removed our filters and operations card and replaced them with a single “SQL Query” card
  - Has fields for SELECT, FROM, WHERE, GROUPBY, and ORDERBY
- We found a pandas plugin called “dataframe\_sql”
  - Allows us to pass a SQL query statement and it will return a filtered and manipulated dataframe

# Visual for Task 2

## SQL Statment

Select

From

Where

Groupby

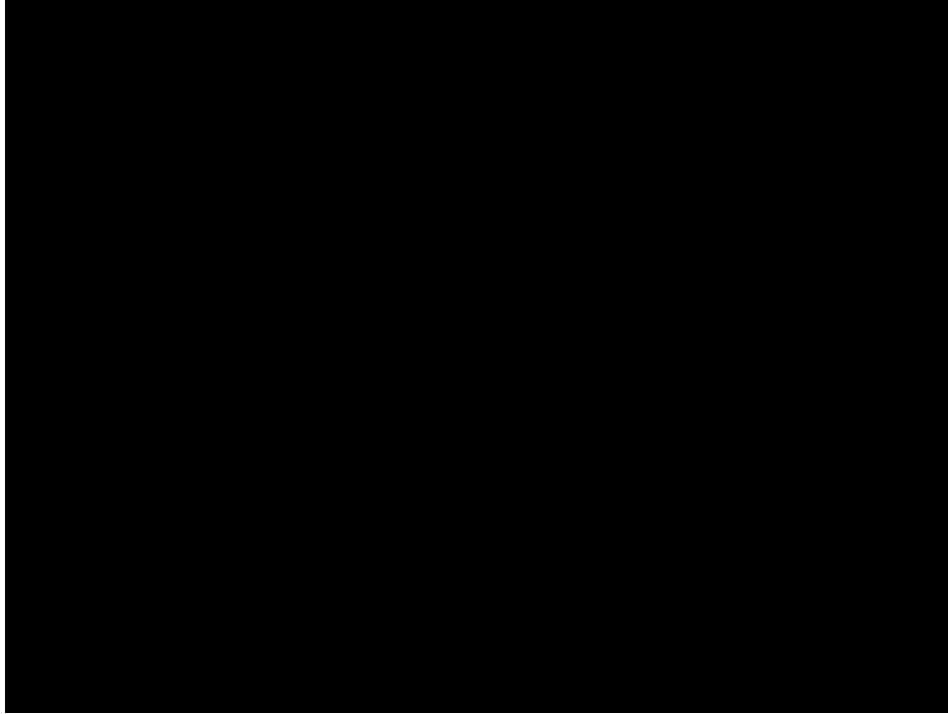
Orderby



# Task 4 - Layer plots

- Figure out which plots can be layered.
- We can layer pie charts, bar charts, line charts, timelines. We still need to figure out if we can layer scatter plots.

# Demo Task 4



# Task 5 - Auto Update Datasets

- User uploads a dataset and sets a URL field that points to an API endpoint
- Then a file is fetched from that endpoint and processed to update the dataset
- The research phase of this task is nearly complete
- Backend implementation will include querying the API endpoint given by the user and converting the returned JSON object to a CSV object
  - Append/Reprocess the dataset
- Frontend changes include changing the “Upload Dataset” page to add a text field for setting the URL that points to an API endpoint

# Task 6 - Dataset Application

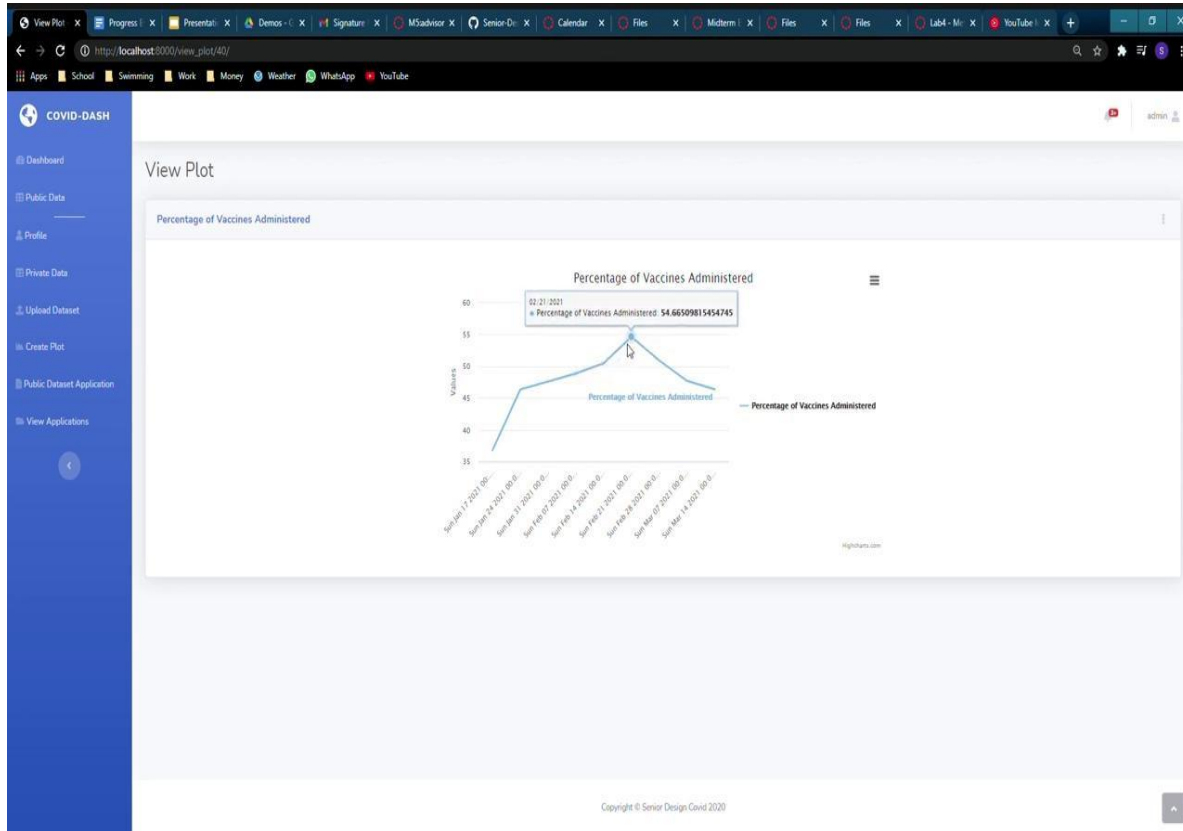
- Updated public data to include both shared and curated data
  - Shared: Data made public by a user but sent through approval process. *Use at your own risk.*
  - Curated: Data that has been submitted via an application and approved by the admin. *Verified, Safe*

# Demo Task 6

The screenshot shows a web browser window with the following elements:

- Browser Tabs:** Senior-De, Update a, Public De, Select ap, Senior De, Launch M, Milestone, Presentat, Progress.
- Address Bar:** 127.0.0.1:8000/public\_data/
- Page Title:** COVID-DASH
- Sidebar Menu:**
  - Dashboard
  - Public Data (selected)
  - Profile
  - Private Data
  - Upload Dataset
  - Create Plot
  - Public Dataset Application
  - View Applications
- Main Content Area:**
  - Curated Data:** This data has been reviewed and approved by the site admin.
  - Curated Dashboards:**
    - Show: 10
    - Search: [input field]
    - Table with columns: Name, User, Date Created, Last Modified.
    - Showing 1 to of
    - Page navigation: « 1 2 3 »
  - Curated Datasets:** [Up arrow button]

# Additional Demo for Vaccines



The diagram illustrates the architecture of a Heroku Python Virtual Environment. It is built on a Linux OS and contains a Python Virtual Environment. The Django Web Server is the central component, which is only accessible by admin accounts. It connects to an Admin Portal (Manage Postgres Database Tables) and a Postgres Database. The Django Web Server also interacts with a Database Schema and an Object Relation Manager. The Views layer includes Create Account/Sign-in, Upload Dataset, Build Plot, and Create Dashboard. The Utilities layer includes Process Dataset (CSV processed and compressed using Pandas, then exported as a Pickled object file) and Render Plot (Opens the Pickled object as a Pandas DataFrame, applies filters and operations using Pandas, and produces a HTML ready representation of the DataFrame). The Pandas library is utilized by both the Process Dataset and Render Plot utilities. The Postgres Database contains tables for user, dataset, location, nation, fuel, and dashboard, and is managed by the Object Relation Manager.

```

graph TD
    subgraph Linux_OS [Linux OS]
        subgraph Python_Virtual_Environment [Python Virtual Environment]
            subgraph Django_Web_Server [Django Web Server]
                URL_Request_Handler[URL Request Handler]
                Database_Schema[Database Schema]
                ORM[Object Relation Manager]
            end
            Admin_Portal[Admin Portal  
• Manage Postgres Database Tables]
            subgraph Views
                Create_Account[Create Account/Sign-in]
                Upload_Dataset[Upload Dataset]
                Build_Plot[Build Plot]
                Create_Dashboard[Create Dashboard]
            end
            subgraph Utilities
                Process_Dataset[Process Dataset  
CSV processed and compressed using Pandas,  
then exported as a Pickled object file.]
                Render_Plot[Render Plot  
Opens the Pickled object as a Pandas DataFrame,  
applies filters and operations using Pandas,  
and produces a HTML ready representation of the DataFrame.]
            end
            Pandas[Pandas]
        end

        Django_Web_Server -- "Only Accessible by Admin Accounts" --> Admin_Portal
        Admin_Portal --> Django_Web_Server
        Django_Web_Server --> Database_Schema
        Database_Schema --> ORM
        ORM --> Django_Web_Server
        Django_Web_Server --> Views
        Views --> Django_Web_Server
        Django_Web_Server --> Utilities
        Utilities --> Django_Web_Server
        Process_Dataset -- "Utilizes" --> Pandas
        Pandas -- "Utilizes" --> Render_Plot
        Django_Web_Server -- "Interacts with" --> Postgres_DB
        Postgres_DB -- "Utilizes" --> Pandas
    end

```

**Postgres Database**

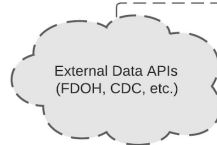
Contains tables for all models defined in the database schema.

Table operations (INSERT, SELECT, etc) are managed by the Object Relation Manager.

The actual data from an uploaded dataset is stored in a pickled object on the AWS server. This database stores a file path to the pickled object file.

**Database Schema:**

- user**
  - id
  - email
  - password
- dataset**
  - name
  - description
  - location
- location**
  - name
  - lat
  - lon
- nation**
  - name
  - continent
  - area
  - pop
- fuel**
  - name
  - type
  - unit
- dashboard**
  - id
  - name
  - description
  - filters
  - operations
  - html



# Task Matrix for Milestone 6

Task	Stian	Sam	Nicole	CJ
1) Ensure that all showcase materials are completed and submitted	Demo Video (25%)	User/Developer manual (25%)	Clean up ebook page and assist Sam and/or Stian as needed (25%)	Clean up poster and assist Sam and/or Stian as needed (25%)
2) Update Build Plot to use SQL statements	-	-	-	Implement evaluation of SQL queries on Pandas df (100%)
3) Auto updates for datasets	-	Implement automatic dataset updates via API endpoints (80%)	-	Assist Sam as needed with implementation (20%)
4) Update application process to work for plots & dashboards	-	-	Implement application flow for Plots and Dashboards (80%)	Assist Nicole as needed (20%)
5) Test/demo of system for evaluation	Speed (25%)	Reliability (25%)	User survey (25%)	Accuracy (25%)



Questions?