PACE DASHBOARD

Technical GUide

2019

Contents

[Overview 2](#_Toc33108331)

[Setup for PACE-RI Dashboard 2](#_Toc33108332)

[Dashboard Layout 3](#_Toc33108333)

[Running the Dashboard 4](#_Toc33108334)

[Booting the Dashboard Locally 4](#_Toc33108335)

[Booting for Production Using Cheroot 4](#_Toc33108336)

[App Framework and File layout 4](#_Toc33108337)

[Starting Folder 4](#_Toc33108338)

[Pacedash Files 5](#_Toc33108339)

[Pages 6](#_Toc33108340)

[Graph Page Creation 6](#_Toc33108341)

[Login Pages – pages used in the login process 6](#_Toc33108342)

[Main Dashboard Pages 6](#_Toc33108343)

[EDA Pages 7](#_Toc33108344)

[Additional Pages 7](#_Toc33108345)

[Assets 8](#_Toc33108346)

[Data 8](#_Toc33108347)

[Note About Related Slapdash Framework 8](#_Toc33108348)

[Users 8](#_Toc33108349)

[Creating or Updating the User Database 9](#_Toc33108350)

Dashboard Technical Guide

# Overview

The ***Pace Dashboard*** is an internal flask web application built in python using the [plotly Dash](https://plot.ly/dash/) framework. It is only accessible on the organization’s internal network at <http://internal.pace.dashboard:8041/>.

Users must login to the dashboard. Login functionality utilizes the werkzeug.security and flask\_login packages.

The application is hosted on a windows server 2008 R2 instance. It uses the cheroot package to establish an HTTP server that calls a WSGI application (which is our flask app). WSGI is a calling convention for the server to forward request to our web application.

The *paceutils* package (custom python package at PACE-RI) is used for most calculations and database queries.

The dashboard pulls data from the PaceDashboard.db and agg.db SQLite databases.

The core python packages used in the dashboard are plotly, dash, werkzeug, flask, flask\_login, and pandas. A full list of dependencies is in the environment file.

*Working knowledge of python is assumed to understand the framework of the dashboard. The* [*plotly dash user guide*](https://dash.plot.ly/) *is a great supplemental guide to understanding the design of the PACE dashboard.*

# Setup for PACE-RI Dashboard

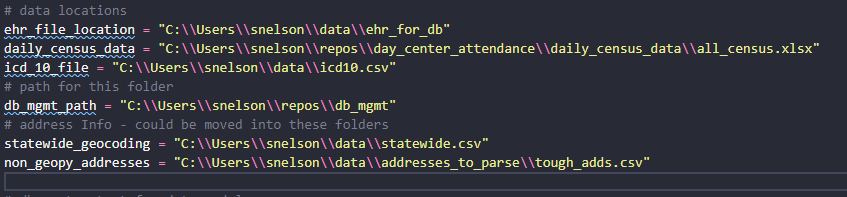
Install python 3.7.5 or greater (Anaconda recommended for personal computers – minconda for servers)

Download all pace\_dash, db\_mgmt, and pace\_utils files

***For the Database***

Create a folder somewhere on your computer name ehr\_for\_db.

In the db\_mgmt/code folder open the filepaths.py file and update the file paths for all data location variables.



Using the anaconda command line prompt navigate to the db\_mgmt folder and run ***conda env create -f environment.yml.***

* Activate the environment by running the ***conda active db\_mgmt\_env*** command
* Next run ***pip install -r requirements.txt*** and then ***pip install -e <path to pace\_utils folder>***

You are now ready to the follow the instructions in this document for updating the database.

**For the Dashboard**

For local computers – navigate to the pace\_dash folder and run ***conda env create -f environment.yml***

* Activate the environment by running the ***conda active pace\_dash\_env*** command
* Once in the environment run;
  + Run ***pip install -r requirements.txt*** in the command line
  + Run ***pip install -e <path to pace\_utils folder>***
  + Run ***python run\_flask.py –debug***
* In your browser (Chrome/Firefox) navigate to http://localhost:8050/ and check that the site is working

This is a local version of the dashboard that can be used to test changes. Any updated files that need to be sent to the server just need to be copied to the V:\Dashboard\pace\_dash folder.

For the server (this is already set up – but in case you need to change servers).

* Run ***pip install -r requirements.txt*** in the command line
* Run ***pip install -e <path to pace\_utils folder>***
* Run python run\_cherry.py to start the dashboard

# Dashboard Layout

Each link on the navbar corresponds to a page of cards, the dashboard user guide has screen shots of the dashboard and more information on navigating these pages.

* These pages have a first row containing a page title and 4 radio/multi-choice buttons. These radio buttons indicate the time period for the values displayed in the cards. These radio buttons are created using dcc.RadioItems from the dash core components and are grouped so that only one can be selected at a time. A dash callback is triggered when one is selected and tells the page to update the card values.
* Each card is a bootstrap card with a title and body. They each contain a combination of columns and rows that indicate the cards layout.
  + the value of the indicator for the selected time period – calculated by a paceutils function.
  + an arrow icon comparing the value to the previous time period’s value for the indicator – arrow direction decided by a function in the helper\_functions file.
  + a sparkline graph – created using plotly.
* Clicking the value will open a new page containing a larger version of the sparkline. On this page the graph can be further filtered by date or grouped by month or quarter. The data filters are dcc.Input components. Theses filters are triggers by dash callbacks. The data for the graph can also be downloaded as a CSV file.

The pages all contain links to “EDA” or Exploratory Analysis pages. These are versions of pages from the first iteration of the dashboard. They provide the user additional abilities to slice, filter, and view the data.

All drop downs, radio buttons, and date fields used to manipulate the data use [dash core components](https://dash.plot.ly/dash-core-components) with [dash callbacks](https://dash.plot.ly/getting-started-part-2) to accomplish these tasks.

# Running the Dashboard

## Booting the Dashboard Locally

Navigate to the ***pace\_dash*** folder in a command prompt window and use the command “python run\_flask.py –debug”. Include –debug so that any changes made to the code will show up on the dashboard without a hard restart. This will run the dashboard locally on <http://localhost:8050/> - enter this address into your browser and you should see the dashboard.

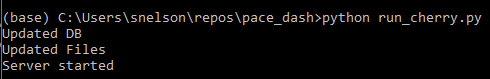


***Note:*** *This should only be used for local testing and creation of the dashboard, it cannot handle multiple request and would fail in production.*

## Booting for Production Using Cheroot

For production the dashboard uses the wsgi module from the cheroot package. Cheroot is a high-performance, pure-Python HTTP server. The script sets up the server to listen to calls from 0.0.0.0 and port 8041. The app is passed to the function as the wsgi callable application and the server name is set to *internal.pace.dashboard.* This results in the dashboard living at <http://internal.pace.dashboard:8041/>.

The server is booted from the command prompt on the server using python run\_cherry.py. The script will print to the command line that the database and dashboard files have updated, and that the server has started.



Command C can be used in the command line to stop the server, this interruption will cause the server to print “Server stopped”.

Whenever the dashboard code or databases on the V:\Drive are updated, the dashboard must be restarted. Navigate to the server, use command c to stop the server, then run the python run\_cherry.py command again.

# App Framework and File layout

There are two main files in the dashboard, those that contain layouts for displaying pages on the dashboard and those that contain utilities and functions that are imported into the layout pages and are used for calculating, slicing, or visualizing the data.

## Starting Folder

The topmost level of the application folder structure contains files for running a server instance or managing the users database.

* *\_\_init\_\_.py* – empty file for indicating this is a python package
* *User\_db\_mgmt.py* – script for creating or updating the users database. Requires an argument specifying the action to perform on the database, the path to the database and then any related arguments for the action function chosen.
* *run\_cherry.py* – script for starting the server instance and calling the app using wsgi.
* *run\_flask.py* – script for running a local testing instance of the dashboard. Should not be used to set up the dashboard for production.

The ***src/pacedash*** folder is where the application code is contained. In the pacedash folder there are various files used either as utilities for the app and its pages or used to set up the application. There are also folders for the dashboard’s static assets, page layouts, and data.

## Pacedash Files

Functions from these files are imported into the pages to ensure the code in the page files remains easy to read. These functions help create charts, calculate values, layout items on the page, and filter the data.

* *\_\_init\_\_.py* – Contains helper functions for creating the Flask and Dash instances.
* *app.py* – Entry point into the app. Creates both the Flask and Dash instances used for the app and then imports the rest of the app through the index module. This is also where the LoginManager from *flask\_login* is imported and set up.
* *index.py* – URL routes and the router are defined here, along with the navbar and its corresponding entries. The router will check to see if a user is authenticated and return the requested page if they are and will return a login redirect page if they are not.
* *settings.py* – Configurable settings for the application. This includes the name of the application, the pages to be shown on the navbar, the file paths for the data, the applications color palette, and plotly graph configuration. Changing these configurations here will change them across the application.
* *components.py* – Convenient Python pseudo-components are defined here. These are used in the layout of the application pages. Contains code for bootstrap rows, columns, cards, headers, and navbar.
* *utils.py* – Utility function for expanding an internal URL and a component function used to allow the components to function as dash components.
* *helper\_functions.py* – Contains some functions that are used in various ways across the dashboard. This file ensures that reusable functions are defined once instead of on every page. This is also where the paceutils classes are imported. These functions create the values in the cards and make the consistent creation of graphs easier.
* *layouts.py* – Contains functions that bundle various dash components and bootstrap elements so that they are reusable and consistent across the dashboard. These include the layout of cards, radio buttons, dropdowns, and various combinations of those items. If a combination of layout elements was to be used on more than one page, it ended up here.
* *demographics\_eda\_utils.py* – contains functions used in the creation of dropdowns and graphs for the demographics-eda page.
* *enrollment\_eda\_utils.py* – Contains functions used in the creation of dropdowns and graphs for the enrollment-eda page.
* *incident\_eda\_utils.py* – Contains functions used in the creation of dropdowns and graphs for the incidents-eda page.
* *utilization\_eda\_utils.py* – Contains functions used in the creation of dropdowns and graphs for the Utilization-eda page.
* *team\_utils.py* – Contains a list of the functions required to create the team page comparison table and a function that creates the actual dash table. Keeps the list of functions from being included in the page code.
* *login\_utils.py* – Contains functions for validating users on the login page and in the router of the index file.
* *users\_mgt.py* – Contains the User class used to log users into the dashboard and track that they are logged in when accessing pages across the dashboard.
* *run\_db\_update.py* – Contains functions for copying updated dashboard code or databases from the V:\Drive. Every time the dashboard is restarted it copies these files down and uploads its log and user database to the V:\Drive.

## Pages

The pages folder contains the python files that contain dash layouts for the pages displayed throughout the dashboard.

### Graph Page Creation

* *default\_graph\_page.py* – This is the page where the main pages’ cards’ indicator values are linked to. A trending graph is created dynamically based on the page’s pathname (URL).
  + If the URL corresponds to a table and column pair in the agg.db database, the default graph page will create a page with a trending graph, data filter, data download link, and a radio button for grouping the data. Otherwise an error page will be created and shown.
  + All the indicator value links are of the format “table-column”, so a function on this page takes this string(text) from the pathname and splits it on the “-“(dash) to create a query of the form “SELECT column FROM table WHERE month BETWEEN ? AND ?” The question marks are filled in with the start and end dates from the data field on the page. If the pathname cannot be split on the “- “, or if the query returns None, an error page is shown.

### Login Pages – pages used in the login process

* *create\_pw.py* –Page where users can create their password for the first time. Contains two dash input components for recording username and password. Uses the *check\_user­* function to check if a user exists and to validate their password. The validate function will throw errors specific to why the users password was not valid.
* *login\_fd.py* – This is the default page layout for a user who is not currently logged in to the dashboard – it asks them to login and can redirect them to the login page. Users cannot login on this page.
* *login.py* – Page where users can enter their username and password to login to the dashboard. Writes to the log file when a user is authenticated. Uses the user class to make sure a username exists and then uses *check\_password\_hash* from *werkzeug.security* and *login\_user* from *flask\_login­* to check the users password and log them into the application.
* *logout.py* – page users are directed to when they click the logout button. Writes to the log that the user has logged out.

### Main Dashboard Pages

These pages are a collection of cards containing values, arrow indicators, and a sparkline. They link to the default graph page where trending graphs are dynamically created. Each card value is calculated using a paceutils function – this function, along with the time period indicated by the radio button, the related table and column pair in aggregate database, and the layout function of the card is passed to the *card\_value* function from the *helpers\_function* file.

Each card has a dash callback that is trigged to update the value when a time range radio button is changed.

***Note:*** *The card values have a corresponding paceutils function and a corresponding column in a table in the aggregate database. This makes creating the trending graphs much easier.*

* demographics.py – Contains demographic related indicators.
* enrollment.py – Contains enrollment related indicators.
* incidents.py – Contains incident related indicators.
* inpatient.py – Contains inpatient (non-nursing facility) related indicators.
* nursing\_facilities.py – Contains nursing facility related indicators.
* operations.py – Contains operations related indicators.

### EDA Pages

These are pages that give the users additional filter options to slice and visualize the related data. Useful for looking at more specific filters or trends – not as useful for viewing single data points.

These pages use functions from the related *\_utils.py* files. Dash callbacks are used to update the data and graphs based on user selected filters. These filters can be in the form of a dropdown, radio button, or data range.

* demographis\_eda.py – Page for further exploring the demographics data.
* enrollment\_eda.py – Page for further exploring the enrollment data.
* incident\_eda.py – Page for further exploring the incident data.
* utilization\_eda.py – Page for further exploring the utilization data.

### Additional Pages

These pages differ from the standard, EDA, or login related pages.

* *teams.py* – This page consists of a large dash table the first column lists all indicators being tracked, and the other columns are the teams here at PACE. Values change using a dash callback that is triggered when the time period radio button is changed.
* *town\_table.py* – Contains a dash table with PACE participants enrolled during the selected time period grouped by city of residence. The table uses a function from paceutils to create the table and a dash callback to update the table based on changing dates.
* *ppts\_map.py* – Creates a map of enrolled and disenrolled ppts indicated by color of dot. The map is created with the plotly Scattermapbox function.

## Assets

The assets folder contains CSS files for styling the application and a folder of font-awesome related files and icons. These are static assets that will be exposed to the web server. Only files that have been modified are describe.

* *pacedash.css* – CSS for the dashboard header – background color of the header/navbar can be changed here as well as the color links.
* *custom.css* – CSS for custom bootstrap rows and for resizing the app based on screen size. Any additional CSS change can be made here.

The font-awesome folder contains font-awesome files included with bootstrap and some Fontello files that have been downloaded separately. This site was used to create the PACE-RI icon in the header.

All other files are related to font awesome or bootstrap and do not need to be modified.

## Data

The data folder contains the PaceDashboard, agg, and users databases as well as a log file that logs when users sign on and out of the application.

* *PaceDashboard.db* – main database for powering the dashboard and includes various tables (mostly from the EHR) in a lightly cleaned format. Additional information is found in the documentation of the *db\_mgmt* folder.
* *agg.db* – This is an aggregate database and contains aggregate values for various data points by month or quarter. Additional information is found in the documentation of the *db\_mgmt* folder.
* *users.db* – Database containing the users table (where passwords are stored) and usernames table (a table for recording who has access to the database, and their related information).
* *log.txt* – Records when a user logs in or out of the dashboard. Does not record if a user has timed out of the database. Formatted as; username, date-timestamp, 1/0 for login/logout.

# Note About Related Slapdash Framework

The original framework for the dashboard came from a package called Slapdash, which provides a layout for quickly building out dash applications. Some of the basic structure is still apparent in the PACE-RI dashboard, but a lot of additional code has been added. Some Slapdash page descriptions are used in this guide. This project would have taken much longer without this package as a jumping off point.

In the past year, the Slapdash layout has also changed, so the projects have really diverged from one another.

More information can be found at <https://github.com/ned2/slapdash>.

# Users

Users are recorded in a csv file on the V:\Drive and in the users database. Usernames are the same as a user’s email, just without the address. I.e., [g antetokounmpo@pace-ri.org](mailto:g%20antetokounmpo@pace-ri.org) will have a dashboard username of gantetokounmpo.

Users are responsible for creating their own passwords, these passwords are hashed using the *werkzeug.security* package and then saved to the users database. The same package is used to check that an entered password is correct. The *flask\_login* package is used to track that the user is logged in across the app.

# Creating or Updating the User Database

The add\_delete\_update\_users.ipynb fie can be used to deal with the users of the dashboard. Navigate to the pace\_dash folder using the Anaconda Prompt and type jupyter notebook as a command. This will launch a jupyter home page, click and open add\_delete\_update\_users.ipynb.

In this notebook there are functions for inserting a new user, deleting a user, update the user database with a list of users in a csv file, creating the user database from scratch from a csv file, and resetting a user’s password. Each function has a docstring explaining what it does.

Choose the function you need and copy it into a new cell. Fill out the requirement parameter and use shift-enter to run the cell.

Once run you have completed the task!