App dev: Home assignment

# Background

All images from microscopy scans in the company undergo a QA procedure by a Python script named IMV (for Image Validation). The result of IMV is a csv file that stores quality measurements in tabular form.

The assignment is to provide a viewer app for biologists to analyze the outputs of IMV.

# Plate scan – data organization

A plate is a matrix of plastic wells (small tubes) organized as 16 rows by 24 columns.

To avoid edge effects, the outer rows 1,16 and outer columns 1,24 are left empty.

Imaging of a well in high magnification requires the microscope to split the scanned area into smaller 2D area patches called **fields.**  Therefore, in the data there will be multiple data samples for each well – each sample bears the statistics of one field within the well.

## QA data file

The file ‘image\_quality.csv’ is the main data table of the image Quality check.

The last row in the file should be ignored (or removed) before reading in the data-frame.

Each row in the table includes meta-data columns ‘r’,’c’,’f’ that identify the row, column and field coordinates of the image that has been analyzed.

The ‘message’ column should be ignored for the purpose of this assignment.

The other columns are numeric features extracted from the images.

## Assay layout file

The file ‘Assay\_layout.csv’ provides a biological description of the assay. It details the compounds, treatment time and other conditions of the samples placed in each well.

The ‘Column’, ‘Row’ columns are the wells coordinates (notice the difference in column names w.r.t the corresponding ‘r’, ’c’ columns of the qa data file)

For the purposes of this assignment, use only the column named ‘Compound’. It carries the role of the sample in the well as follows:

* DMSO represent the control group (the untreated condition)
* Any other symbol identifies a compound with which the sample is treated

# Requirements & expected behavior

The app is based on Streamlit framework. Refer to <https://docs.streamlit.io/>

1. the user is expected to perform the following steps:
   1. Upload the assay-layout file from her local storage
   2. Upload the QA data file from her local storage
   3. Once data is available, select a feature of interest from a drop-down list of features available from the QA data table
2. Compute the feature **variation** **V** for every combination of well & feature as follows:
   1. Define **M(W,fet)** as the median of the feature **fet**, taken over all the fields (data lines in the input) that belong to the set of wells designated by **W** (single or multiple wells).
   2. Calculate Mdmso,fet = M(Wdmso,fet) - the median of the feature over those wells annotated as ‘DMSO’ in the assay layout.
   3. Then, for every single well, calculate **V(w,fet)** = M(w,fet)/ Mdmso,fet
3. Selected feature display
   1. Display a 2d heat-map of the selected feature **median** values. The map has the Row X Col labels of the plate (16X24) as described in the introduction.
   2. Display a histogram of the selected feature values, taken over all the wells in the data.
4. Brush selection
   1. Enable the user to graphically select (A.K.A “brushing”) a set of wells from the heat-map in 3.a. Call this graphically selected set the “**test-group**”
   2. Create (or update) a box-plot with 2 boxes. One for each set of wells:
      1. DMSO box – made from the values V(wdmso,fet) for wells in the DMSO group
      2. Selected box - from V(wtest-group,fet) for wells from the test-group in 4(a) above
5. SQL search
   1. Store the data uploaded by the user to a local SQLite database with two tables
      1. The data table – store records for the numeric features from the QA data file, plus the value V(w,fet) you computed for each well in steps 2a-2c
      2. The assay table – store the records from the assay layout file. For this assignment, it is sufficient to store the association between compound identifiers and well coordinates; please ignore other features of the assay file.
   2. In the GUI, display a slider label “Variance threshold”. The slider enables the user to pick values in the range [0..100), in steps of 5
   3. Upon user changes to the slider value, perform an SQL query and display the results of every well-feature combination in which the variation V() (as defined in 2 above) has larger value than the threshold set by the slider. For this requirement – use SQL query and do **NOT (**!!) rely on the dataframe you might already have stored in memory from previous stages
6. Bonus question – no coding required

Most Streamlit examples are single-page applications. Packing many views on a single-page will result in a cluttered display. We would often prefer to split the display into **pages.**

Suggest a software architecture around Streamlit that will ease the splitting of the visual display into several pages that share the same data (or “state” in Streamlit terminology)

Assignment notes: Read before starting execution

* Watch the example provided in : <https://youtu.be/htXgwEXwmNs>
* Streamlit use script re-run upon GUI interaction. Pay extra attention to the usage of **session state** for storing context across changes.
* Prefer Pandas & Numpy operations over loops whenever possible
* Using public solutions from the net is encouraged. However, when you do use a public source or part of it – you are expected to explain the solution and protect your reasons to use it.
* Your code is not expected to protect against all scenarios of ill-formed data (bad file format, missing or mis-spelled columns). It should protect against exceptions due to arithmetic errors (division by zero, imputation of Nan)