User Documentations

## Application Workflow

### How to use

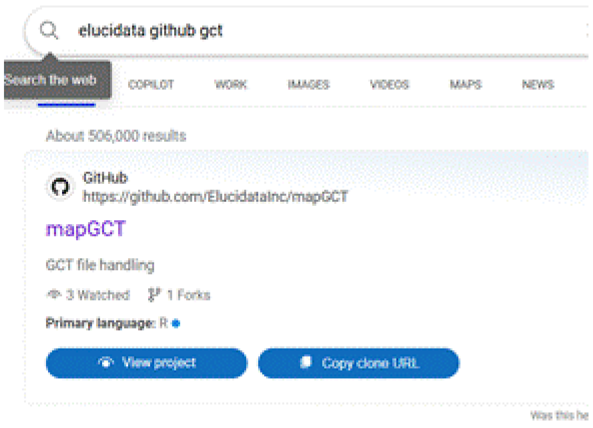
### ui.R

- Contains User interface code

### server.R

- Contains user input and output process, data transformation and some cool stuff to plot your graphs

1) Prerequisite is to have GCT file (Read more about GCT file GitHub - ElucidataInc/mapGCT: GCT file handling)
2) Upload GCT file in upload section and once upload is complete table will come showing the matrix data
3) Then move to normalization and then move to PCA & scree plot



About 506,000 results

GitHub
https://github.com/ElucidataInc/mapGCT

mapGCT

GCT file handling

3 Watched 1 Forks

Primary language: R

## 1. Uploading Gene Expression Data (GCT File)

- The user uploads a gene expression file in the GCT format via the input field `gct_file`. The file should contain gene expression data with rows representing genes and columns representing different samples.
- After the file is uploaded, the application parses the GCT file using the `mapGCT::parse_gct()` function.
- The parsed data (expression matrix) is stored in the `rv$gct_data` reactive value, and metadata about the samples (such as tissue types) is stored in `rv$cdesc`.

**User Interface Behavior:**

- The number of genes, samples, and unique tissue types are displayed in value boxes.
- The user is prompted to select a column from the dataset (from the `cdesc` metadata) for further analysis.

**2. Data Normalization**

The user can choose to normalize the data using one of the following methods:

- **CPM (Counts Per Million)**: Normalizes the gene expression data by scaling each gene's count to a per-million scale.
- **Quantile Normalization**: A statistical normalization technique that adjusts the data so that the distribution of gene expression values across samples is the same.

**Normalization Process:**

- When the "Apply Normalization" button is pressed:
  - The selected normalization method (either "CPM" or "Quantile") is applied to the uploaded gene expression matrix (`rv$gct_data`).
  - If "CPM" is selected, the `normalizeCPM()` function is used.
  - If "Quantile" normalization is selected, the `quantile_normalize()` function is applied.
  - The normalized data is stored in `rv$normalized_counts`.

**User Interface Behavior:**

- The user can choose whether to apply a log transformation to the data by checking the `log_transform` box.

- Once the normalization is complete, users are notified, and the normalized data can be downloaded as a CSV file.

**3. Principal Component Analysis (PCA)**

- PCA is performed on the normalized data (`rv$normalized_counts`) to reduce the dimensionality and visualize the variance explained by the principal components.
- The user can specify how many top genes to include in the PCA analysis (default is 500 genes, which can be adjusted using the `top_counts` input).
- The PCA is computed using the `pca_calc()` function, which performs the PCA on the top genes with the highest variance (using the Median Absolute Deviation (MAD) as a selection criterion).
- A PCA plot is generated using the `pca_plot()` function, where users can select which principal components (PC1, PC2, etc.) to plot on the x and y axes, and a color variable from the metadata to color the data points.

**User Interface Behavior:**

- Once the PCA computation is complete, the user can:
  - Choose the x and y axes for the PCA plot (i.e., which principal components to visualize).
  - Select a metadata column to color the points in the PCA plot (e.g., tissue type).
- The plot is rendered interactively using Plotly, allowing users to zoom and hover for more detailed information.
- After the PCA plot is generated, users can download the plot as a PNG file.

**4. Scree Plot**

- A scree plot is generated to visualize the proportion of variance explained by each principal component (PC). This is done by plotting the explained variance percentages for each PC.
- The scree plot helps users determine how many principal components to retain for further analysis.

**User Interface Behavior:**

- The scree plot is rendered using Plotly, showing the variance explained by each PC.

**5. Downloading Results**

Users can download the following files:

- **Raw Gene Expression Data**: The original gene expression data in CSV format.
- **Normalized Data**: The normalized gene expression data (CPM or Quantile Normalized) in CSV format.
- **PCA Plot**: The PCA plot as a PNG image file.

## Technical & Logical Workflow

**etailed Function Descriptions:**

1. **quantile_normalize()**
    - Purpose: Performs quantile normalization on the provided count matrix.
    - Inputs:
        - `counts`: Gene expression matrix.
        - `log`: Boolean value indicating whether to log-transform the data before normalization.
    - Output: A normalized count matrix.
2. **pca_calc()**
    - Purpose: Performs PCA on the provided input matrix and returns the PCA results along with a summary.
    - Inputs:
        - `input_matrix`: Gene expression data matrix (either raw or normalized).
        - `metadata`: Sample metadata.

        - `top_genes`: Number of top genes to retain for PCA.
    - Output: A list containing PCA scores and summary statistics.

3. **normalizeCPM()**
    - Purpose: Normalizes the gene expression data using the Counts Per Million (CPM) method.
    - Inputs:
        - `counts`: Gene expression matrix.
        - `log`: Boolean value indicating whether to apply a log-transformation.
    - Output: A CPM-normalized gene expression matrix.

4. **pca_plot()**
    - Purpose: Generates an interactive PCA plot using Plotly.
    - Inputs:
        - `pca`: PCA result object.
        - `column`: Metadata column used to color the PCA plot.
        - `pc1`: Principal component for the x-axis.
        - `pc2`: Principal component for the y-axis.
    - Output: A Plotly interactive scatter plot.

5. **Reactive values (`rv`)**:
    - `rv$gct_data`: Stores the gene expression matrix.
    - `rv$cdesc`: Stores the metadata (e.g., sample information).
    - `rv$normalized_counts`: Stores the normalized gene expression matrix.

  - `rv$pca`: Stores the PCA result.
  - `rv$pca_scores_plot`: Stores the PCA plot.

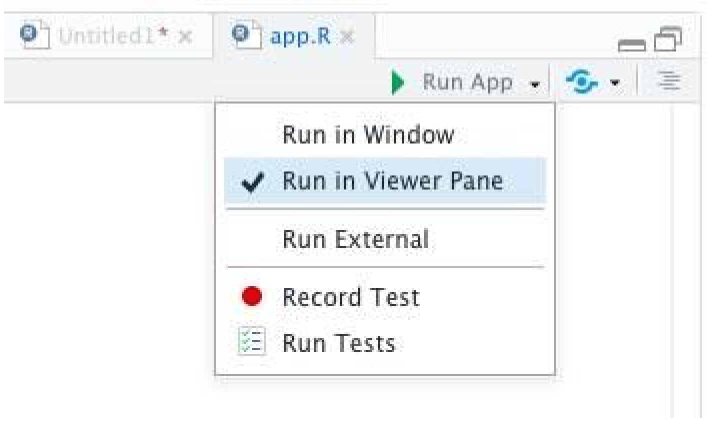
**Error Handling and Notifications:**

- The application uses `tryCatch` to handle errors during file upload, data normalization, and PCA computation. If an error occurs, a notification is displayed to the user indicating the issue.
- Success notifications are displayed when each stage (file upload, normalization, PCA computation) is completed successfully.
- Missing or invalid inputs trigger error or warning messages using `shinyalert()` or `showNotification()`.

## Running An application

Once you statisfies all the dependencies, there are mutliple ways to run the program. But for this tutorial will be using simple RStudio IDE way.

1. Open `ui.R` and `server.R` files in Rstudio
2. You would see `Run App` button (as shown in below figure) click on that
3. Select Run External option so it will open in your system's default browser.



### sessionInfo()

> sessionInfo()

R version 4.2.2 (2022-10-31 ucrt)

Platform: x86_64-w64-mingw32/x64 (64-bit)

```
Running under: Windows 10 x64 (build 22631)

Matrix products: default

locale:
[1] LC_COLLATE=English_India.utf8  LC_CTYPE=English_India.utf8    LC_MONETARY=English_India.utf8 LC_NUMERIC=C
              LC_TIME=English_India.utf8

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
 [1] shinyalert_3.0.0 plotly_4.10.1       reactable_0.4.4     edgeR_3.40.2        limma_3.54.2        shinyhelper_0.3.2
shinyjs_2.1.0
 [8] ggplot2_3.4.1    DT_0.27             mapGCT_0.0.2        data.table_1.14.8   shinydashboard_0.7.2 shiny_1.7.4

loaded via a namespace (and not attached):
 [1] Rcpp_1.0.10      locfit_1.5-9.7      lattice_0.20-45     tidyr_1.3.0         digest_0.6.31       utf8_1.2.3
mime_0.12
 [8] R6_2.5.1         reactR_0.4.4        evaluate_0.20       httr_1.4.5          pillar_1.8.1        rlang_1.1.0
uuid_1.1-0
[15] lazyeval_0.2.2   rstudioapi_0.14     fontawesome_0.5.0   jquerylib_0.1.4     rmarkdown_2.20      textshaping_0.3.6
webshot_0.5.4
[22] htmlwidgets_1.6.2   munsell_0.5.0    xfun_0.37           compiler_4.2.2      httpuv_1.6.9        pkgconfig_2.0.3
systemfonts_1.0.4
[29] htmltools_0.5.4 sourcetools_0.1.7-1 tidyselect_1.2.0    tibble_3.2.1        fansi_1.0.4         viridisLite_0.4.1
crayon_1.5.2
[36] dplyr_1.1.0      withr_2.5.0         later_1.3.0         grid_4.2.2          jsonlite_1.8.4      xtable_1.8-4
gtable_0.3.2
[43] lifecycle_1.0.3  magrittr_2.0.3      scales_1.2.1        cli_3.6.0           cachem_1.0.7        farver_2.1.1
promises_1.2.0.1
[50] bslib_0.4.2      ellipsis_0.3.2      ragg_1.2.5          generics_0.1.3      vctrs_0.6.0         RColorBrewer_1.1-3
tools_4.2.2
[57] glue_1.6.2        purrr_1.0.1      crosstalk_1.2.0    fastmap_1.1.1        yaml_2.3.7          colorspace_2.1-0
memoise_2.0.1
```