
Lanceotron Documentation

Martin Sergeant

Nov 24, 2021

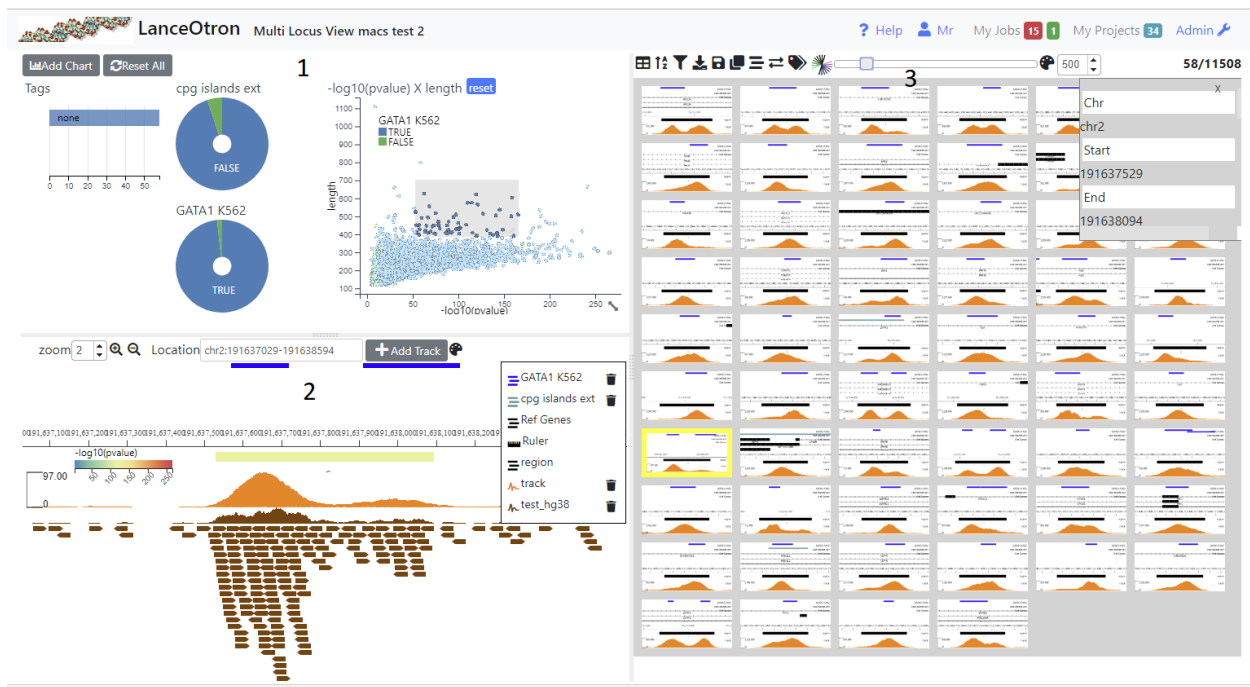
CONTENTS:

1	Multi Locus View	1
1.1	Summary	1
1.2	Creating a Project	2
1.3	Adding Graphs/Charts	3
1.4	The Genome Browser	6
1.5	The Table/Images	7
1.6	Tagging Locations	9
1.7	Adding Images	10
1.8	Running Analysis Jobs	11
1.9	Creating Subsets	14
1.10	Exporting Data	14
1.11	Project History	15
1.12	Permissions	15
1.13	Submitting an Issue	16
1.14	Frequently Asked Questions	16
2	Combat MLV	17
2.1	Summary	17
2.2	Features	17
2.3	Getting Started	18
2.4	Left Hand (Summary) Panel	18
2.5	Right Hand (Feature) Panel	22
3	MLV Developer Documentation	25
3.1	Installing the Application	25
3.2	Config Settings	29
3.3	Executing Scripts	30
3.4	Modules	32
3.5	Projects	34
3.6	Jobs	37
4	Indices and tables	41

MULTI LOCUS VIEW

1.1 Summary

Multi Locus View (MLV) enables the user to visually inspect the underlying data from NGS experiments e.g. RNA-seq, ChIP-seq. Initially, the user uploads a list of genomic locations and associated data (*Uploading a File*), which can then intuitively filtered, sorted and viewed to drill down to regions of interest.



The view is divided into three sections. The top left shows charts (1), the bottom left a simple genome browser (2) and the right section shows a table containing all the genomic locations (3). Initially no charts are present and the browser only shows a track of the uploaded genomic regions (and possibly a RefGene track if a genome was selected). In order to make it easier to view and filter your data you can add charts (*Adding Graphs/Charts*) and browser tracks (*Adding Tracks*). In addition analysis jobs such as finding intersections, calculating signal at each location from a bigWig track and dimension reduction can be carried out (*Running Analysis Jobs*). Locations can be annotated (*Tagging Locations*) by the user and downloaded or exported to the data visualisation tool *Zegami*

1.2 Creating a Project

The screenshot illustrates the initial steps of creating a project in the Lanceotron application. The top navigation bar features a 'MLV' logo, a 'Help' icon, a user profile 'Mr', and links for 'My Jobs' (15), 'My Projects' (15), and 'Admin'. A red '1' is positioned above the 'My Projects' link. The 'Project Types' section offers two choices: 'Annotations' and 'Multi Locus View'. The 'Multi Locus View' option is selected and highlighted with a red '2'. Below this, a form titled 'Enter name and description to start' contains three input fields: 'Name:', 'Description:', and 'Genome:' (a dropdown menu with '--Select--'). A blue 'Create' button is located at the bottom right of the form. A red '3' is placed above the 'Description:' field.

To create a project click on 'My Projects' (1) on the top navigation bar and then the Multi Locus View panel (2). This will take you to a page where you have to fill in the name and description of the project (3). You also have to select the genome required. If the genome you want is not available select 'other'. In this case gene information and other annotations will not be available. When you press 'Create' you will be taken to a page where you can upload your file (see below)

1.2.1 Uploading Data

Input File Required

The file format is quite flexible and can be either tab(.tsv) or comma(.csv) delimited and can also be gzipped (.gz). The only requirement is that the first three columns (in order) specify the genomic location i.e. chromosome, start and finish. Normal bed files fulfill these criteria as well as excel data that has been saved as a .csv or .tsv file.

Other file types apart from bed-like files are not supported in the initial upload but can be added later to the browser (see [Adding Tracks](#)). In addition, in subsequent analysis bigWig files can be uploaded in order to calculate peak stats at each genomic location (see [Calculate Peak Stats](#))

Chromosome names can either be the UCSC e.g. chr1, chr2, chrX etc or Ensembl style 1,2,X. However, if subsequent tracks are added to the browser or biWig files are used to calculate peak stats, they have to be of the same format as the initial upload. This is not the case for calculating intersections, where you can mix and match between projects/annotation sets with Ensembl and UCSC style chromosome names.

Uploading a File

Press Choose in the displayed dialog and select your file containing genomic locations. The file will be parsed and the column (feild) types will be ascertained and displayed. Ensure the correct datatype has been deduced and change if necessary using the dropdowns (2). The header names are taken from the file (1), but you can change these and you can also delete any columns that you do not want (3). If no headers were detected in the file, enter the name of the column - the value of the first row is given to help you (1). If the file contains a header, but it was not detected, check the Has Headers box (4) at the the bottom of the dialog.

Once you are satisfied, press the upload button. There will be a delay as the data is uploaded and processed and if there are no problems you will be presented with an initial view

1.2.2 Saving A Project

When tracks, graphs or columns are added and you have edit permissions (see [Permissions](#)), they are permanantly added to the project. Also, when an analysis job has completed, any graphs,tracks or columns produced will be permanant. However, when you alter a graph or tracks's settings or reize/move them, these are not saved. Similary any changes you make to the table (column resizing/ordering, sorting etc.) will not automatically be saved. In order to save these changes you need to save the layout > Save Layout.

If you wish to make changes to public project then you will have to clone the project > Save As. The project will be copied into your name and you can them make any changes you wish.

1.3 Adding Graphs/Charts

Charts help you get a picture of the data as a whole and also help you filter the data. By selecting regions (dragging with th mouse) on scatter plots and histograms or clicking on sections in pie charts, row charts and box plots, the data can be intuitively filtered. With each filtering step, all charts will update (as well as the table and browser) to reflect the filtered data. Filters on individual charts can be removed by clicking the reset button which appears on the chart's title bar when a filter is applied or filters on all charts can be removed with the 'Reset All' button.

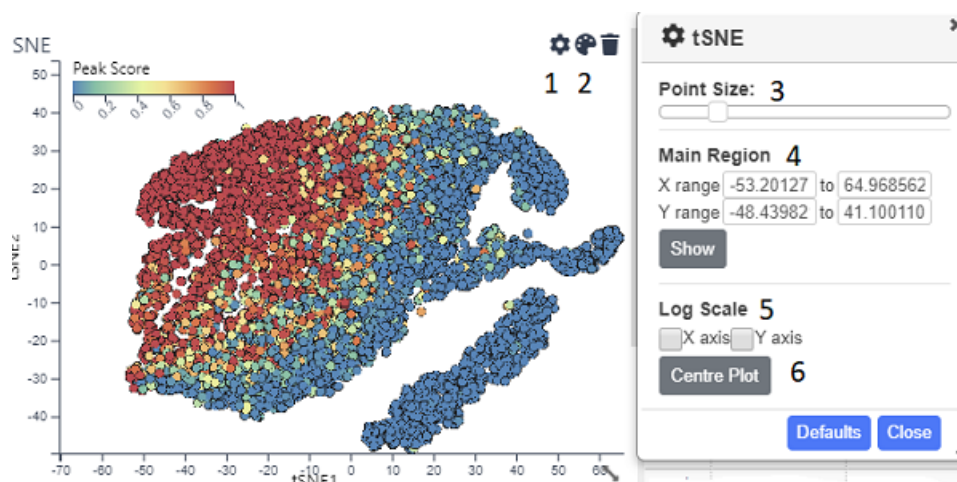
Charts can be moved by dragging on the title bar, and resized by dragging on the resize icon, which appears in the bottom right hand corner of chart when you hover over a chart.

Initially the only chart visible will be a row chart showing Tags (see [Tagging Locations](#)) so you need to add other charts to get a better insight into your data (see below)

1.3.1 Adding a Chart

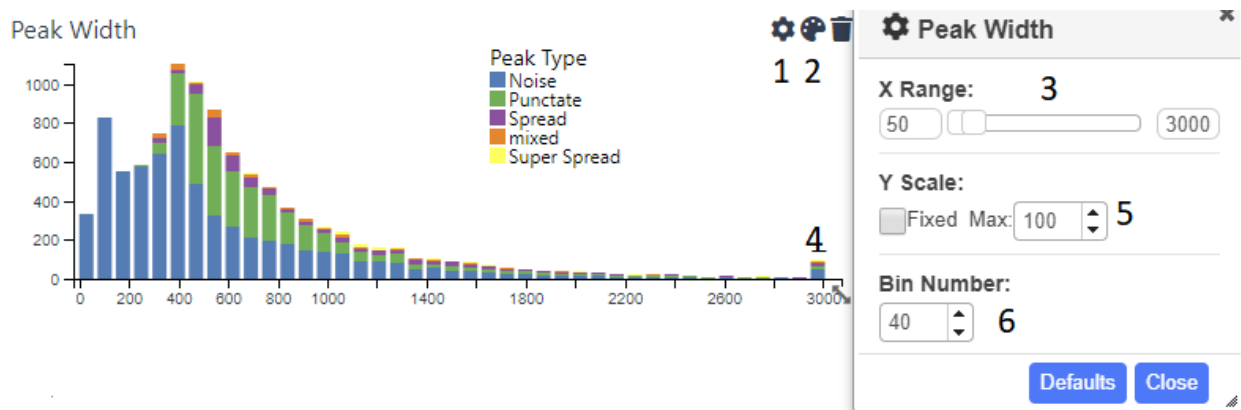
Clicking on the 'Add Chart' button will show a dialog where you have to select the type of chart, the fields to use in the chart and its name. Once created you can change the chart's settings (icon), which differ according to the chart's type and with some charts color it (icon). Charts can be moved by dragging them via the title bar and resized by the resize icon which appears in bottom left hand corner when the mouse is over the chart. The chart can be removed by clicking the trash icon, which appears when you hover over the graph's title. Once charts have been added and the appropriate settings/colors added, they can be saved using the icon above the table. The following chart types are available

Scatter Plot



A standard scatter plot requiring numerical fields for the x and y axis. Once created, the points can be coloured (on the title bar). Also by opening up the settings dialog (icon) you can alter the point size (3). By default the graph will show all the points, but you can zoom in and out using the mouse wheel and pan by pressing shift and dragging with the mouse. However if you want the default view to be a particular region, you can set this using the inputs in the Main Region section (4) and pressing show. The x and/or y axis can also be set to a log scale (5). After zooming and panning, the Centre Plot button (6) will restore the plot to show all the points or the region specified in (4). Normal mouse dragging (without shift) will cause produce a brush that filters the data, once created the brush can be dragged to different regions of the plot.

Histogram



Shows the distribution of a numerical field in the data. The x range is automatically set to include the largest and smallest values. However, this will often lead to the chart looking skewed due to low numbers of extreme outliers. Therefore, you can use the cog icon (1) to open up the settings dialog, where an upper and/or lower limit can be set (3). Values higher or lower than these limits will be lumped together in the outermost bin (4). The y axis can also be capped (5) in order to get a better handle on bins containing fewer counts. The number of bins can be adjusted using the appropriate spinner(6). Each bar can be coloured by categorical data use the icon (2).

Pie Chart

Shows categorical data. By default the maximum number of categories shown are the 8 largest ones, any remaining categories are lumped into 'Others'. This can be changed by opening up the settings dialog (). Clicking on a segment (category) will select that category and clicking on further segments will add these to the filter. To filter again with a single category, use the reset button.

Row Chart

A chart showing categories on the y axis and usually the number of records belonging to this category on the x axis. You can also choose a numerical field for the x axis, in which case the values of this field will be summed for each category. However a boxplot is usually more informative for this kind of information as the average and quartile ranges of the values are shown instead of the sum. As with the pie chart, the maximum number of categories shown are the 8 the largest ones, but this can be changed by opening up the settings dialog ()

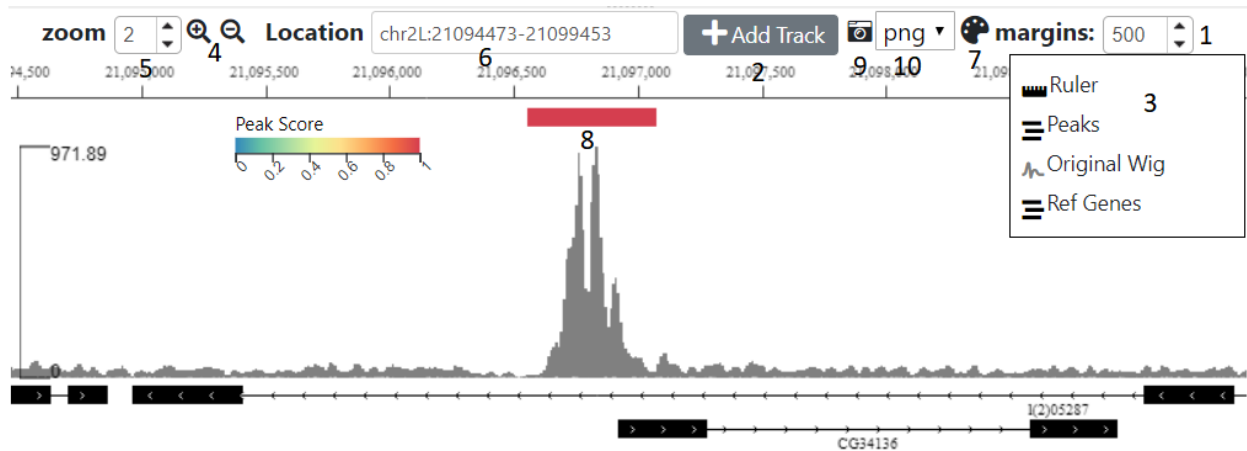
BoxPlot

A chart showing categories and average/quartile ranges of the values of another field for that category. Box plots work best for fields that contain only a small number of categories. They are scaled to include all the datapoints, so if there are extreme outliers, the boxes will appear squashed.

Bar Chart

A Bar Chart showing the column average of any number of supplied fields. Because fields may differ in scale, to ensure differing values can fit on the same scale, the average is scaled between the median $\pm 1.5 \times \text{IQR}$ (the same as the whiskers on a boxplot). The graph changes to only include those datapoints in the current filter. No Selection is possible with this chart, as it would make no sense to filter on a column

1.4 The Genome Browser



The browser shows the genomic location of the currently selected table row (or image). The distance either side of the region to also show can be controlled using the margin spinner (1) above the browser

1.4.1 Adding Tracks

Initially only two tracks will be displayed, the genomic locations you uploaded and if you didn't select 'other' for the genome, a track is displaying the genes. Other tracks can be added with the 'Add Tracks' button (2), which shows a dialog where you need to enter the url of a publically accessible track. The hosting server of the track should allow Cross Origin Resource Sharing (CORS). The type of track will try and be ascertained based on the url, although you can manually override this by clicking on one of the radio buttons

Tracks that can be added are:-

- bed(tabix) - A bed file that has been gzipped and indexed with tabix
- BigBed
- BigWig
- Bam - A bam index is also required
- UCSC session - either cut and paste the url from the UCSC browser or use a session url. The latter will be more stable as the former uses a temporary id, which is only valid for a short period.

1.4.2 Altering Track Appearance

Clicking on the track label in the legend (3) will open a dialog for that track. The contents of the dialog will vary according to the type of track. The track height can be altered from this dialog

1.4.3 Zooming/Panning

There are five ways you can navigate using the browser:-

- You can zoom in and out using the mouse wheel and scroll left and right by dragging the mouse
- Use shift and drag to highlight and zoom into a region on the browser
- Use the magnifying glass icons (4), the zoom amount can be controlled by the adjacent spinner (5)
- Type the location co-ordinates (chr:<start>-<stop>) in the location text box (6)
- Click on a row or image in the right hand table to go to that feature. The margin spinner (1) shows how many bp either side of the feature will be displayed.

1.4.4 Feature track

This shows the uploaded regions(features) displayed in the right hand table. Clicking on the settings icon (7) will bring up a dialog where the following can be adjusted:-

- Set the field you wish to label the features with
- Set the field to color the feature by
- Set the field with which to position the feature on the y axis. By default the feature layout depends on the layout (Collapsed, expanded or squished) but can be a numeric field
- Choose the margins (distance either side of the feature) that will be displayed when you click on an image or a row in the table

1.4.5 Saving the Browser Layout

Use the disk icon above the table to save all settings including the current layout of the browser (tracks and track settings)

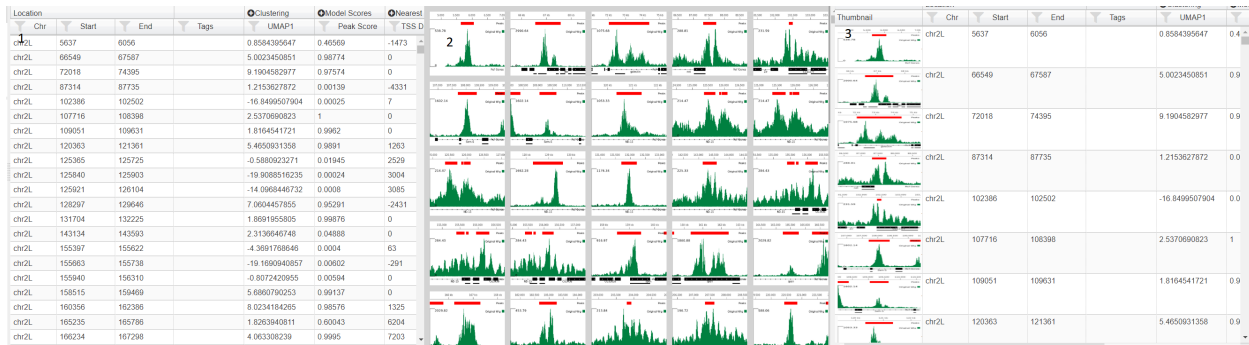
1.4.6 Capturing An Image

Use the icon (9) to download an image of the current browser view. The image format (png, pdf or svg) can be altered using the adjacent dropdown (10)

1.5 The Table/Images

The default table behaves as a typical spread sheet, you can alter the column width by dragging the header's left and right borders and move columns by dragging the column's header. Clicking on the header will sort by that column. Clicking on that row will select it and update the browser.

1.5.1 Table Mode



If your project contains images (see [Adding Images](#)) then then you change how the table is displayed using the table icon (). Three choices are table (1), images (2) and table with thumbnails (3). In image mode, the genomic location can be selected by clicking on the image and using the arrow keys to select the next/previous image. In this mode, the data can be sorted and filtered using the icons () in the menu above the table. Also in image mode you can alter image size using the slider in the table menu and also color the border around the image by a field (). This opens up a dialog where you can choose the field and the color scale to use

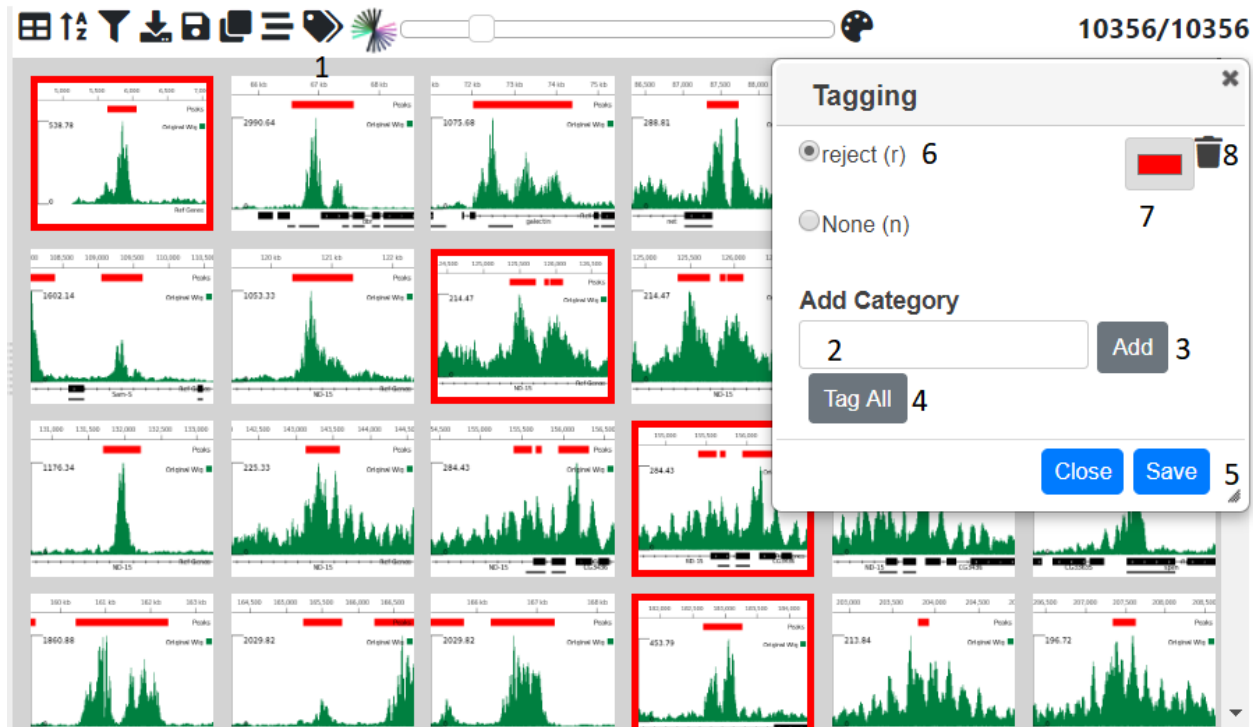
1.5.2 Filtering Data

It is often more intuitive to filter using graphs (see [Adding Graphs/Charts](#)), however data can be filtered by clicking on filter icon in each column header. To filter on multiple columns or when the table is only showing images, press the filter icon on the top table menu. This will bring up a dialog showing filtering options for all fields in the data. Whenever any filters are added or changed, any charts will update accordingly, but the filters are not added to the charts or existing filters on the charts updated as they are completely independant.

1.5.3 Sorting Data

The data can be sorted on columns by clicking the column header (shift click to sort on multiple columns). The data can be also be filtered by clicking the sort icon in the table menu. In the sort dialog, the columns to be sorted on are added using the plus icon and then either Ascending (ASC) or descending (DESC) can be chosen . The sort order can be changed by dragging the labels or columns removed from the sort by clicking on the trash icon

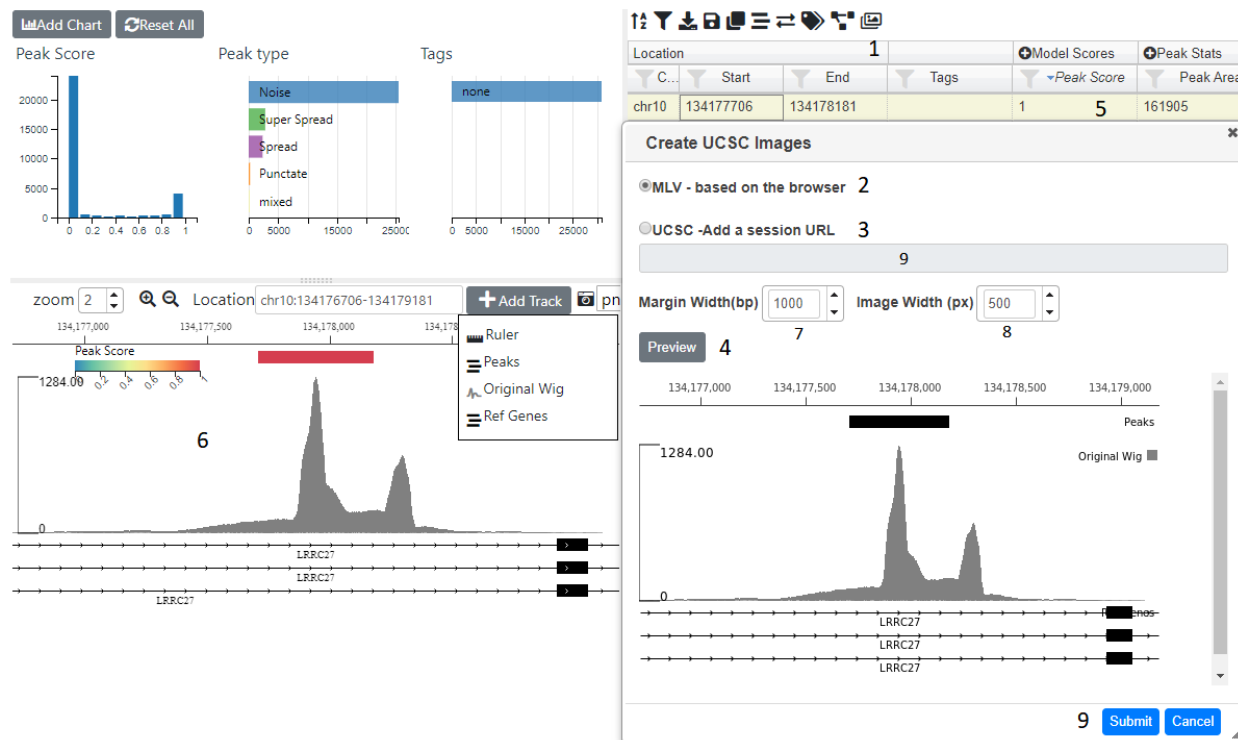
1.6 Tagging Locations



Sometimes it may be useful to categorise or tag the genomic location based on a trend that you have discovered. This can be done by opening up the tagging dialog with tag icon (1) in the menu above the table. Initially only the none category is present. To add other ones type a name in the text box (2) and press the add button (3). The category will then be added to the list at the top of the dialog. By selecting the radio button next to it, then clicking on an image or a cell in the tagging column in the table will tag that genomic location. Multiple locations can be tagged by clicking and image/cell and the shift clicking another one and all the images/rows in between will be tagged. The 'Tag All' will tag all the currently filtered locations with the currently selected category. Another way to tag is to use the arrow keys to go to the next previous image/row and then press the shortcut key shown in brackets next to the category to tag the currently selected items with that category. The category color can be changed by clicking on the appropriate color chooser (7). The category can be removed (which will remove all tags of this category from the data using the trash icon next to the category (8)

N.B. To permanently save the tags press the Save button (5) which will commit the changes to the database.

1.7 Adding Images



Images for every genomic location can be added to the project and then displayed in the table. The icon (1) opens up a dialog where you can choose to either have images created based on the internal browser (2) or by the UCSC browser (3). With the UCSC option, you can have more detailed images, but image generation is much slower and you are limited in the number of images you can create. One option is to create a smaller subset (see [Creating Subsets](#)) and then produce images from this.

1.7.1 MLV Images

Clicking on the Preview button (4) will show a preview of the image for the currently selected row (5). The image is based on the tracks and settings in the browser (6) see [The Genome Browser](#) on how to add tracks and alter their appearance. You can adjust the image width and the width of margins shown either side of the genomic location by using the appropriate spinners (7 and 8). Once you are happy with image you can press submit button (9) and images for all genomic locations will be created. This will take a few minutes (approx 800 images/min).

1.7.2 UCSC Images

Clicking on the UCSC radio (3) will enable the URL input (9) where you can paste a UCSC browser URL or session. Pressing preview will check the url is valid and produce an image based on the margin width (7), image width (8) and selected genomic location (5). If a preview was successfully produced then you can press the submit button (9) to generate images for all genomic locations. This will take quite a while.

An email will be sent when all images have been generated. You can then view the table in image or thumbnail mode (see [Table Mode](#)) and upload the project to Zegami

1.8 Running Analysis Jobs

Analysis jobs are run in the background on the server and the results, in the form of tracks, graphs and extra columns in the table are added to the project once the job is complete. The following types of analysis are possible:-

- **Annotation Intersection** - calculates whether each location overlaps a set of annotations or locations from another project.
- **Find TSS Distances** - calculates whether each location overlaps a Transcription Start Site (TSS) and if not, the distance to nearest site, either upstream (+) or downstream (-).
- **Calculate Peak Stats** - calculates the area and max height of the signal from a bigWig file at each location in the project.
- **Cluster on Columns** - carries out dimension reduction (UMAP,tSNE) on any number of given columns in the project.

Jobs are run in the background and can be viewed on the 'My Jobs' page (link in the top navigation bar) or in the history dialog (), which will automatically open when you send a job. You do not need to stay on the page whilst jobs are running, although if you do when the job is finished, you will be notified and the appropriate results loaded in.

1.8.1 Annotation Intersection

Intersections can be carried out between an Annotation Set, which is basically just a list of genomic locations, or another project. In the simplest case, a single column will be added, with TRUE/FALSE values, depending on whether a region in your project overlaps with a region in the query set.

Annotation Sets

Annotation sets are just lists of genome locations and can be created by clicking on 'Annotations' in the upper panel of my projects page, which will open up the following page:-

The screenshot displays the 'My Annotation Sets' interface. On the left, a table lists existing annotation sets. On the right, the 'Create Annotation Set' form is shown with a 'Next' button. An 'Upload File' dialog is open, showing 'No File Chosen' and a 'Choose' button. The form fields are numbered 1 through 7, corresponding to the instructions below.

Name	Status	Genome	Date Added	Public	Share	Delete
anno ensembl	complete	Human(hg19)	2020-11-20	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
ucsc test anno	complete	Human(hg19)	2020-11-20	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
jhh	failed	Human(hg19)	2020-11-20	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
sdsds	failed	Human(hg19)	2020-11-20	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
dfgg	new	Human(hg18)	2020-11-20	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
test ucsc	failed	Human(hg19)	2020-11-20	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Black List	complete	Mouse(mm10)	2020-05-10	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Hap1 Marks	complete	Human(hg19)	2020-05-07	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Hap1 Histone Marks	complete	Human(hg19)	2020-05-06	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Black List (ENCODE)	complete	Human(hg19)	2020-04-15	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
super enhancer	complete	Mouse(mm9)	2020-03-30	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
T1 Down Peaks	complete	Mouse(mm10)	2019-12-02	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
dfdd	complete	Human(hg38)	2019-10-07	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
GATA1 K562	complete	Human(hg38)	2019-09-03	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
cpg islands ext	complete	Human(hg38)	2019-06-17	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
cpg Islands ext	complete	Human(hg19)	2019-06-14	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Create Annotation Set (Reset)

Name: (1)

Description: (2)

Genome: (3)

☐ Has Headers

(4)

Upload File (X)

No File Chosen

Fields: 3

Fill in the name, description and genome in the right hand panel (1), then press next (2). A dialog will open (3), which allows you to upload a bed like file (see [Uploading a File](#)). In the left hand panel (4) is a list of all the annotation sets that you own. You can make these public (5), share (6) or delete them (7).

Another way to create an annotation set is within a project, > create annotation set. **N.B.** The set will be created from the currently selected (filtered locations), shown the top right hand corner. Again, the dialog allows you to fill in the name and description of the set and you can also check any columns that you want included in the set.

Intersections

> 'Annotation Intersection' will bring up a table with all the Annotation Sets and projects that you are able to intersect with. You can select single or multiple (ctrl or shift) sets/projects. If you select a single project/set then a dialog will ask you whether you want just a TRUE/FALSE column or whether you want extra columns, with information from the intersecting set.

Once an annotation set has run, columns will be added to the table, either a single TRUE/FALSE column for each intersecting set or the data columns you selected in the previous step. In addition, pie charts showing this TRUE/FALSE distribution will be added, along with a track for each intersecting set.

1.8.2 Find TSS Distances

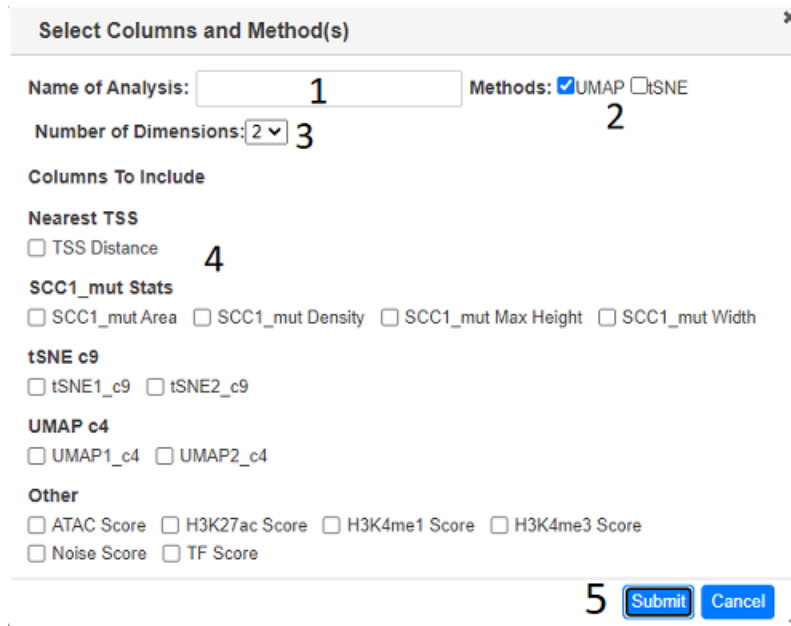
will bring up a simple dialog, with the only choice being whether you want to include Gene Ontology annotations. These are taken from the go-basic.obo file (<http://geneontology.org/>) and collapsed, such there is only one term (the most frequent) at each hierarchical level. If you include GO annotations, you can choose up to which hierarchical level you want.

Once the job has run four columns will be added to the table

- TSS distance - the distance to the nearest TSS, + being upstream and - downstream
- Gene ID - the Genbank gene id of the nearest gene
- Gene Name - the common name of the nearest gene
- Overlaps TSS - either TRUE or FALSE depending on whether the region overlaps the TSS

1.8.3 Cluster on Columns

In order to get a better handle on the data it may be useful to collapse some of the fields into two (or more) dimensions, such that they can be visually clustered in a 2D scatter plot. To do this click on the cluster icon () above the table, which brings up the dialog below.

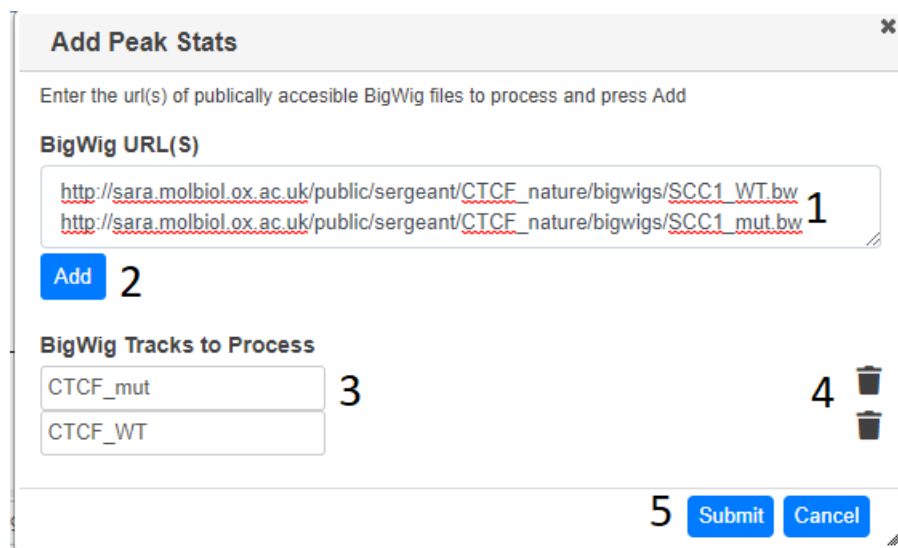


Type a name in the text box (1) and select the dimension reduction methods required (2). The number of dimensions can also be increased (default 2) using the dropdown (3). All numerical fields in the analysis are displayed (4), check all the ones to be used in the analysis and then press submit (5).

The outputs are columns for each dimension for each method named '<method><number>_<analysis_name>' e.g tSNE1_anal1, tSNE2_anal1 etc. For each method, a scatterplot of the first two dimensions will be also be added. You can change these scatter plot e.g color by a specific field (see [Scatter Plot](#)) or use the dimensions to add another graph (see [Adding a Chart](#))

1.8.4 Calculate Peak Stats

If you have bigWig files and you want to find out the peak area/height in each of the genomic locations in your project, use the icon which brings up the dialog below.



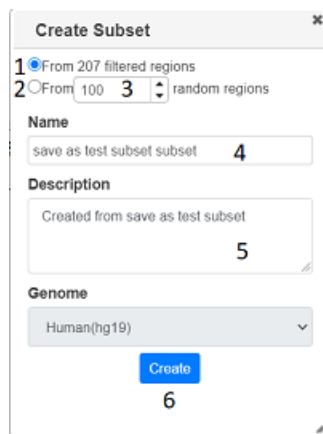
Paste the url (or a list of urls) corresponding to bigwig files you want to analyse in the text box (1) and press Add (2). If the bigWig files can be located and they are the correct format, they will be added to the “bigWig Tracks to Process”

section (3) in the dialog. The name is taken from the file name, but this can be changed. When you have specified all the bigWig files required, press submit. The area, max height, width and density (area/width) in each location will be calculated. You do not have to stay on the page whilst the stats are being calculated.

When complete, columns will be added to the table with the relevant information and each bigWig track will be added to the Browser. The bigWig tracks are added with default settings, so you may need to change them to suit your needs (see *Altering Track Appearance*). Note it just calculates the amount of signal in each region and reports the width of the region, it does not try and call peaks and work out the width of the peak.

1.9 Creating Subsets

Clicking on brings up a dialog, which allows you to create a subset of the currently selected locations



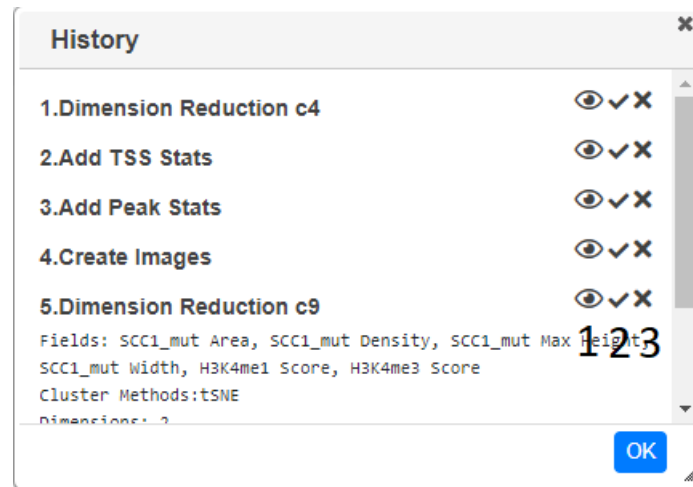
You can choose to create the subset either from the currently filtered locations (1) or from a random subset (2) with a specified number (3). After filling in a name and description (4) and (5), press 'Create' and the subset will be created. Once this has happened, you will get a link to the subset. You can create a subset of any project you have viewing rights too, including public projects, and you will be the owner of the subset. All graphs/tracks/columns are copied, although the graphs may look different as there will be fewer locations in the new project.

1.10 Exporting Data

Click on the download icon to download the currently filtered locations. The data is just downloaded as a text file, although you get a choice to download in either tsv or csv format. Only the currently displayed columns will be downloaded, so expand any column groups by clicking the plus icon if you want these in your file. If you have images in your project, you can export the data to [Zegami](#). Click on the Zegami icon and a dialog will appear, where you have to fill in your zegami username, project id and password. Once the project is created, you will be emailed a link to the project

1.11 Project History

Every action such as adding a graph/track/column is recorded and be viewed by opening the history dialog ()



Clicking on the eye icon (1) will toggle information about the action. The second icon (2) shows the status of the action, a tick means it is complete, a spinning circle shows that it is still processing and an exclamation mark shows there was an error whilst trying to perform the action. If you have edit permissions you are able to undo the action (N.B. there is no redo action). This will remove the action from the dialog and remove any tracks, columns or graphs that the action generated. If columns are removed, then any graphs which use these columns will also be removed, even if they were not added by the action.

1.12 Permissions

There are two types of permission for a project, view and edit.

If you have a view permission for a project (anyone has view permission for a public project), you can open the project and add tracks and charts, as well as edit existing charts and tracks. However, you cannot save any updates or run any jobs such as finding TSS's or creating images. If you want to do this, you will have to copy the project (you need to be logged in) - click the disk icon () and then select 'save as'. This will clone the project in your name and then you can make any changes you wish.

If you have editing rights to a project you can make any changes you want, run any jobs and save the layout. You automatically have editing rights to a project you own it or if you are an administrator. You can also be assigned editing rights to a project (see below)

The icon on the menu allows you to share the project with another user and assign them view or editing rights

1.12.1 Sharing a Project

Click on share icon () above the table and select 'share project'

Start typing the name of user you want to share the project with in the text box (1) and then select the name from the drop down. When you press 'Add' (2), the project will be shared and the name of the user will be added to the dialog (3). You can change the editing rights of the user to view or edit using the dropdown (4) or stop sharing by pressing the trash icon (5)

1.12.2 Making a Project Public

Click on share icon () above the table and select 'share project', you will be prompted to see whether you really want the project to become public. If you click OK then anyone (including non users) will be able to view the project. You can share the project by sharing the link in the browser's address bar.

1.13 Submitting an Issue

An Issue or question, can be asked within MLV (if you are logged in) using the help link in the top navigation bar > 'Send Question'. You can also submit an issue to the [GitHub page](#)

1.14 Frequently Asked Questions

1.14.1 Can MLV be viewed on a mobile/small screen device?

No. The whole idea is to see how each component ie. the graphs, tracks and images change as you filter the data, which would not be possible, if only one component was displayed at once. All panels and individual tracks/graphs/table columns/images can be resized, to get the exact layout that the user requires, rather than relying on adaptive screen size techniques which limit viewing to a single component/panel on small screen sizes

1.14.2 Can I upload a bigWig file?

Not initially. The only files that can be initially uploaded are bed or bed like files with genomic locations. However, bigWig files can be added to the browser and uploaded for processing later (see [Calculate Peak Stats](#)). Another application, [Lanceotron](#) does take bigWig files and identifies peaks based on machine learning.

COMBAT MLV

2.1 Summary

Combat MLV is built on top of [MLV](#) but instead of simultaneously viewing multiple genome locations, it allows the visualization of multiple experimental observations from the same samples as produced by the COMBAT project. The vast quantity and disparate types of data that has been generated by the COMBAT project creates many challenges to visualizing the data as a whole. For example, some experiments used bulk tissue, whereas others used single cell techniques. In addition, gene expression, protein abundance, percentage cell types are quite different types of measurements, having different properties such as scales and thus are difficult to compare. There is also a lot of missing data with a relatively small number of samples having measurements from all modalities. To address these issues the following was implemented in COMBAT-MLV:-

- Data from single cell experiments (modalities) is collapsed to sample/tissue (pseudobulk) to allow comparison with bulk experiments
- Data has been binned into percentile bins so each feature will have a relative value from 1 to 10, which allows easy comparison between modalities. However raw data is also available if required
- Gene/proteins have been matched between modalities so bulk/single cell gene expression can be compared to protein measurements (Luminex/Mass Spectrometry)
- There are many filters that can be applied to the data so that depending on which modalities are being viewed the amount of missing data is kept to a minimum in each view

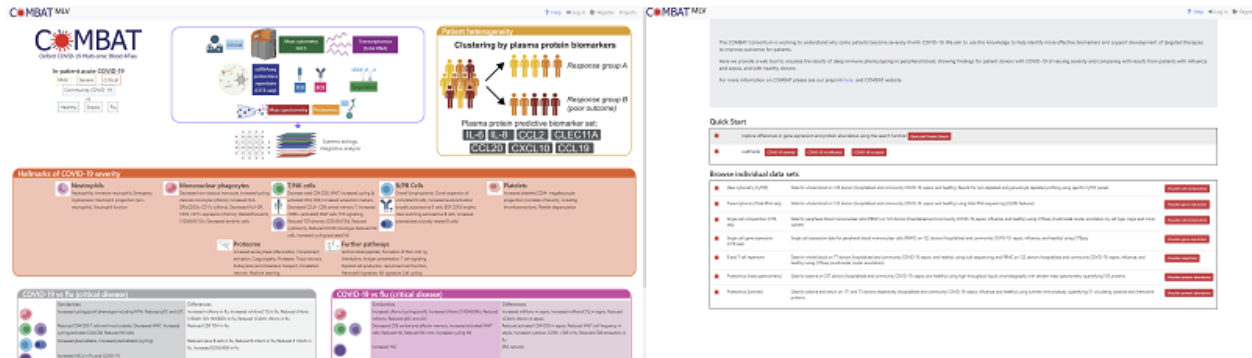
2.2 Features

Note: In the context of COMBAT-MLV, an item of data is known as a feature and is comprised of the following five levels:

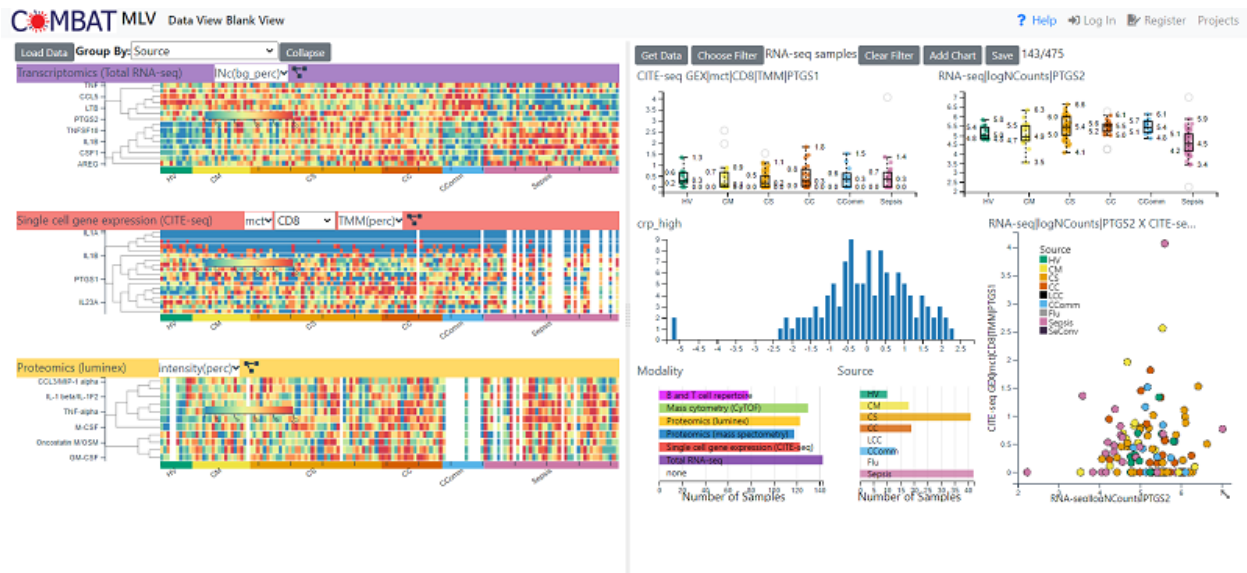
Modality(Cluster Level) | (Cluster) | datatype |name

- **Modality** The actual experiment e.g., CITE-seq GEX, RNA-seq, CYTOF
 - **Cluster Level** The level at which single cell data has been collapsed e.g major/minor cell type (not present in bulk modalities)
 - **Cluster** The actual cluster e.g. cell type (not present in bulk modalities)
 - **DataType** The type of data e.g. raw counts, reads per million, percentile bins
 - **Name** The actual name of the feature being measured e.g gene/protein, % cell type
-

2.3 Getting Started



Specific views can be created by searching for genes/proteins or selecting data from different modalities or loading specific hallmarks. Predefined views can be accessed from the main [home page](#) or from a clickable [summary figure](#). There are two panels in the display. The right-hand panel displays summary of the data heat maps (one per modality) as well as hallmarks. The left hand panel displays individual 'features'



2.4 Left Hand (Summary) Panel

2.4.1 Loading Data

The load data button brings up four choices

- Paste Gene List - (see) Allows you to type to paste gene names and will try and find matches in each modality. The genes will then be shown in a heatmap
- Load Hallmark - loads a text box with a list of distinguishing feature
- Load other data types - Load data sets from modalities that are not protein/gene based such as % cell types
- Load Dendrogram - Loads data that can be displayed hierarchically. At the moment the only option is the cell designation hierarchy derived from the CITE-seq Data

2.4.2 Loading Gene/Protein data

IL1A

IL1B

IL1RN

IL6

IL10

IL23A

IL33

TNF

Find

1

2

Gene and Protein Selection

Input

IL1A

IL1B

IL1RN

IL6

IL10

IL23A

IL33

TNF

LTB

PTGS1

PTGS2

OSM

CSF2

CSF1

AREG

TNFSF10

CCK3L1

CCL3

CCL5

Single cell surface protein abundance (CITE-seq)

Single cell gene expression (CITE-seq)

4

Transcriptomics (Total RNA-seq)

5

Proteomics (mass spectrometry)

Proteomics (luminex)

IL1A

IL1B

IL1RN

IL6

IL10

IL23A

IL33

TNF

LTB

PTGS1

PTGS2

OSM

CSF2

CSF1

AREG

TNFSF10

CCL3

CCL5

IL-1 alpha/IL-1F1

IL-1 beta/IL-1F2

IL-6

IL-10

IL-23

IL-33

TNF-alpha

Oncostatin M/OSM

GM-CSF

M-CSF

CCL3/MIP-1 alpha

6

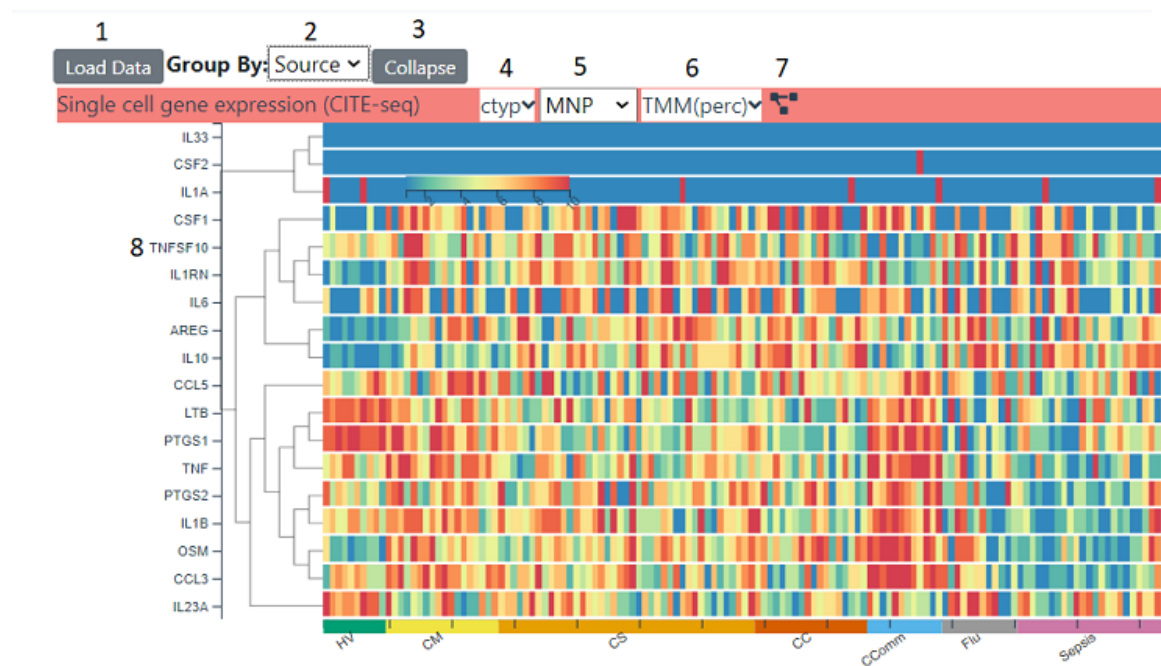
submit

Clicking on Load Data > Paste Gene list will bring up the Gene and Protein Selection dialog. Gene names (or EMBL Ids) need to be typed or pasted in the text box and the the the Find button (2) clicked. After a short search any modality containing data for the gene or corresponding protein will be shown in the main table. Modalities (columns) can be removed by un-checking the appropriate check box (4). Genes (rows) can be removed with the bin icon (3) or individual gene/modalities can be unchecked. Pressing the submit button will load in all the requested data and display appropriate summary heat maps (one per modality)

2.4. Left Hand (Summary) Panel

19

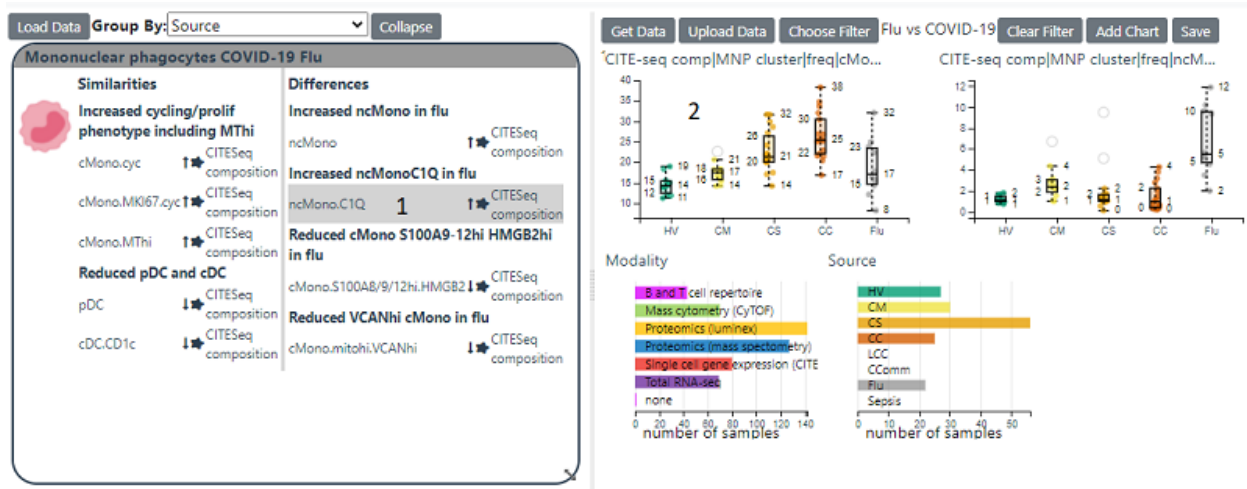
2.4.3 Heat Maps



Heat maps can show a range of data . The samples on the x axis are grouped into categories (initially source) . This can be changed with the source dropdown (2), the choices available depend on which data has been loaded into the right hand panel. For a continuous field categories are split into four equal bins spanning the data range. The heatmap can be collapsed into the displayed categories , whereby an average is shown instead of each individual sample, by using the collapse button (3). Depending on which which modality the heatmap is showing, there will be a number of options in the title bar. For single cell modalities, the cluster level (4) and cluster i.e. cell type (5) can be changed. For all heatmaps, the type of data can be changed (6). Which types are available depends on the modality , but clicking on the dropdown will give you options and explanations about each type. The rows can be re-clustered at any time by clicking the cluster icon (7)

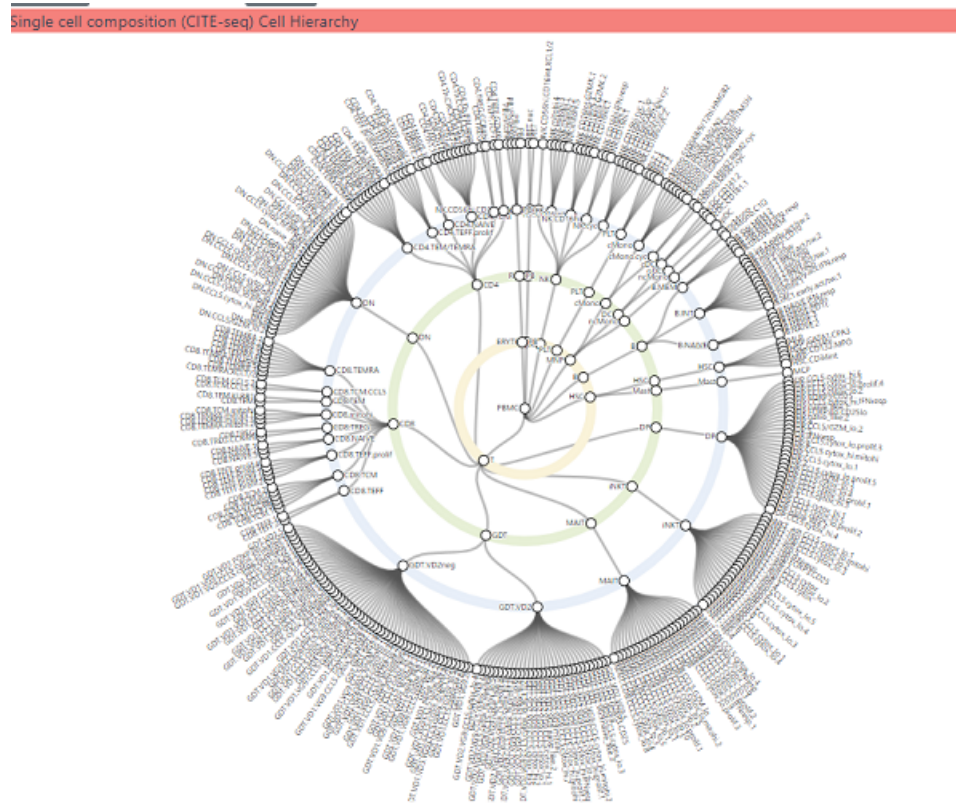
Data can be transferred to the right hand panel by clicking on the x axis labels and that feature will then be transferred across and a box plot will be shown. The data will be whatever cluster/cluster level and data type is currently being displayed in the heat map. The feature can then be combined with other features or sample associated data in a range of charts. For single cell data a heatmap consisting of expression for all cell types of the currently selected cluster level will also be added to the left-hand panel

2.4.4 Hallmarks



Hallmarks show Features which have been shown to differentiate between COVID-19 and Sepsis/Flu or between the different severities of COVID-19 infection. By clicking on a feature in the hallmark will load it into the right hand panel and generate a box plot of that feature

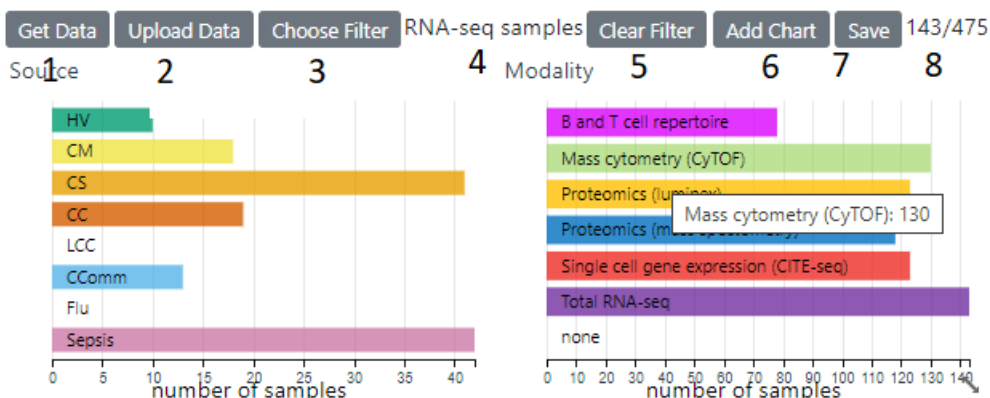
2.4.5 Cell Type Dendrogram



This shows the three hierarchical cell type composition derived from the CITE-seq gene expression analysis. Panning and zooming can be achieved with dragging the mouse and using the mouse wheel respectively. Clicking on a white

circle (cell type) will load data for that cell type into left hand panel. The number of levels can be altered by clicking on the cog icon and changing the level in the dialog which appears.

2.5 Right Hand (Feature) Panel



This panel shows individual feature and other data associated with samples. By default, row charts the sample's source and the modality is shown. A sample can have data from more than one modality To load sample associated data, click on the Get Data button (1) and you can then select from a range of available fields.

2.5.1 Changing Filters

The current filter (4) is displayed and can be changed with the Choose Filter button . The current number of samples and the total are displayed on the right (8) . Changing the filter will update all the charts and the heatmaps to reflect the currently filtered samples. A Filter can also be applied (on top the selected base filter) by clicking on category in a row/pie chart or selecting a range/area in a histogram/scatter plot. Clicking the reset button on a chart will then remove that chart's filter

2.5.2 Adding New Charts

New Charts for any of the current fields can be added with the Add chart button (6) This will bring up a dialog where you select the type of chart and fields to be used in the chart . The type of charts are

- **scatterplot** - two continuous fields can be plotted against each other
- **boxplot** - a continuous field can be chosen for the y axis e.g. a feature and discrete field to the x axis e.g source
- **histogram** - the distribution of continuous field is displayed
- **row chart/pie chart** the categories in a discrete field are displayed

2.5.3 Saving the View

Saving the view will save all the loaded data, current filter and charts. To achieve this click on the save button (7). If you do not own the view you can select 'save as', which will clone the current view and save it under a different name. You can also share the view with another user or make it public. The url will always point to this link or you can access it from my projects in the top nav bar

MLV DEVELOPER DOCUMENTATION

3.1 Installing the Application

3.1.1 Clone from Git

```
git clone https://github.com/Hughes-Genome-Group/mlv.git
```

3.1.2 Setting up a Python Environment

MLV requires python 3.6 or later. One way to manage the python environment is to use virtualenv and virtualenvwrapper.

```
pip install -U virtualenv
pip install -U virtualenvwrapper
```

Then add the following to your .bashrc (paths may vary)

```
export WORKON_HOME=$HOME/envs
export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3.6
. /usr/local/bin/virtualenvwrapper.sh
```

Next create a virtual environment and install all the required python modules:-

```
mkvirtualenv mlv
workon mlv
pip install -r requirements.txt
```

3.1.3 Installing Dependencies

The following programs need to be installed and available in the path:-

- tabix and bgzip (<https://github.com/samtools/htslib>)
- rabbitmq-server (<https://www.rabbitmq.com/download.html>)
- bedtools (<https://github.com/arq5x/bedtools2/releases>)
- bedToBigBed, bigBedToBed and bigWigInfo (<http://hgdownload.cse.ucsc.edu/admin/exe/>)
- nodejs (<https://nodejs.org/en/download/>)

- nodejs modules najax, jquery,xhr2,extend and canvas
- NGINX (<https://www.nginx.com/>) or another http server (for production)
- PostgreSQL 9.5 or above (<https://www.postgresql.org/>)

3.1.4 Creating the Database

MLV requires PostgreSQL 9.5 or above, which can be running on the same or a separate server. The first thing to do is to create the system database and associated tables by running *create_system_db.sql* in *app/databases/*. To do this via the the psql cosole, log in as a user with the correct permissions and run the following:-

```
createdb mlv_user;
\c mlv_user;
\i /path/to/app/databases/create_system_db.sql;
```

If your PostgreSQL instance does not have a suitable user, you need to create one and grant access to tables generated in the previous step.

```
CREATE USER mlv WITH PASSWORD 'pword';
GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA public TO mlv;
GRANT SELECT UPDATE, DELETE ON ALL SEQUENCES IN SCHEMA public TO mlv;
```

You also need to change the DB parameters in settings.py (or a custom config)

```
DB_USER="mlv"
SYSTEM_DATABASE="mlv_user"
DB_HOST="localhost" #or host name
```

Make sure you allow the user to connect to the database in PostgreSQL's *hb_config*:-

```
#allow connection from another host
host mlv_user,generic_genome mlv x.x.x.x/32 md5
#allow local connection
local mlv_user,generic_genom mlv md5
```

If the database is hosted on a different server, you may have to update the firewall settings on this server to allow the mlv server to connect to it on port 5432

Creating The First Users

To create a user you will need to run the appropriate script (see *Executing Scripts*) In order to do this, the app needs to know the location of the scripts module and also the password to the database. These can be stored in a secure file and added to the environment variables:-

```
export FLASK_APP=/path/to/install/app/commands/cli_commands.py
export DB_PASS=pword
```

Then you can add the guest user and an admin:-

```
flask find_or_create_user --firstname John --last_name Doe --email guest@somewhere.com
flask find_or_create_user --first_name Mr --last_name admin --email me@gmailcom \
--password password --admin True
```

Adding Genomes

Genomes are housed in separate databases to the system database. A single databases can hold many genomes or separate databases can be created for each genome (not recommended). A fallback genome called 'other' initially needs to be added. The following code creates a database 'generic_genome' and then adds the fallback 'other' and the C. Elegans(cell) genome:-

```
flask create_new_genome_database -db_name generic_genome
flask add_new_genome --name other --label Other --database generic_genome
flask add_new_genome --name ce11 --label C. Elegans(ce11) --database generic_genome
```

3.1.5 Running and Serving the Application

For testing porposes the application can be run using the *run_app* script.

```
flask run_app --port 5000
```

This will run the application locally on port 5000. For production purposes it should be run through a web server gateway interface such a Gunicorn. For example, the following code will run the app locally on port 5000 using 3 threads.

```
/path/to/virtualenv/gunicorn "app:create_app('lanceotron_config')"\
-b 127.0.0.1:5000 \
--workers 3 \
--error-logfile /path/to/gunicorn.log
```

MLV should also be run through a webserver such as Nginx or Apache. Although static files such as images and tracks can be served through the app via flask, this is not recommended in a production setting. It is more efficient to serve such files from the webserver. An Nginx config to allow this is given below:-

```
#redirect to the flask app
location / {
    proxy_pass http://localhost:5000;
    proxy_read_timeout 180s;
    proxy_set_header    Host                $host;
    proxy_set_header    X-Real-IP           $remote_addr;
    proxy_set_header    X-Forwarded-For     $proxy_add_x_forwarded_for;
    proxy_set_header    X-Forwarded-Proto   $scheme;
}
#directly serve any static files
location /static/ {
    alias /home/sergeant/mlv_dev/app/static/;
}
#temporary images
location /data/temp/ {
    alias /data/mlv/temp/;
}
#static files belonging to a module
location ~ /(.*)/static/(.*) {
    alias /home/sergeant/mlv_dev/app/modules/$1/static/$2;
}
```

(continues on next page)

(continued from previous page)

```
#genome tracks
location /tracks/ {
    alias /path/to/tracks;
}
#any images belonging to projects
location ~ /data/(.+\.?(?:jpg|jpeg|gif|png))$ {
    alias /data/mlv/$1;
}
```

3.1.6 Running the Message Queue

MLV uses celery and rabbit-mq to queue time consuming tasks and run simple pipelines. Two queues need to be initialised, a default queue for jobs which should return quickly and are required for the user to continue and a 'slow queue' for longer tasks and pipelines. To start the queues use the following commands:-

```
/path/to/venv/bin/flask runcelery
/path/to/venv/bin/flask runcelery --queue slow_queue
```

The default number of threads is 3, although this can be changed using the `-threads` parameter. For debugging purposes celery can be disabled by changing the config setting `USE_CELERY=False`, which will cause jobs to run directly in the flask thread and hence would be impracticable for a production environment.

3.1.7 Using Supervisor

Supervisor (<http://supervisord.org/>) can be used to populate environment variables, and run the server and job queues as daemon threads. An example of a suitable config would be:-

```
[unix_http_server]
file=/path/to/supervisor.sock ; (the path to the socket file)

[supervisord]
logfile=/path/to/supervisord.log ; (main log file;default $CWD/supervisord.log)
user=username ; (default is current user, required if root)
directory=/path/to/root_dir ; (default is not to cd during start)
environment=FLASK_APP="/path/to/root_dir/app/commands/cli_commands.py",\
             FLASK_CONFIG="my_config",\
             DATABASE_PASS="pword"

[supervisorctl]
serverurl=unix:///var/run/supervisor/supervisor.sock ; use a unix:// URL for a unix_
↳ socket

[program:mlv]
command=/path/to/venv/bin/gunicorn "app:create_app('lanceotron_config')"\
    -b 127.0.0.1:5000 \
    --workers 3 \
    --error-logfile /path/to/gunicorn.log

autostart=true
autorestart=true
```

(continues on next page)

(continued from previous page)

```
[program:celery]
command=/path/to/venv/bin/flask runcelery
autostart=true
autorestart=true

[program:celery_slow]
command=/path/to/venv/bin/flask runcelery --queue slow_queue
autostart=true
autorestart=true
```

3.2 Config Settings

The main config is *settings.py* in the main app directory. Another config, which can add or override variables in *settings.py* can be specified either in the *create_app* method or in the environment variable `FLASK_CONFIG`. The following shows the config variables which may need to be changed.

3.2.1 Database Settings

- **DB_HOST** The database host, either a server name or an IP address. By default, the environment variable `DB_HOST` will be used or `localhost` if it is not set.
- **DB_USER** The name of the database user, `mlv` by default.
- **SYSTEM_DATABASE** The name of the system/user database, `mlv_user` by default.
- **DB_PASS** The database password, set to the environment variable `DATABASE_PASS` by default (passwords shouldn't be stored in the config)

3.2.2 Folder Locations

- **DATA_FOLDER** The location to store all the data, can be a mapped drive
- **TEMP_FOLDER** The location to store temporary data. Data here can be deleted
- **TRACKS_FOLDER** The location to store and serve genome tracks (BigWigs, BigBed files etc)

3.2.3 Message Queue (Celery) Settings

- **BROKER_URL** The message queue url. By default is the `localhost` but potentially it could be a different server
- **USE_CELERY** The default is `True`, only set to `False` when debugging

3.2.4 App Settings

- **HOME_PAGE** the url of the home page of the application
- **HOST_NAME** the name of the machine hosting the app
- **APPLICATION_NAME** The name of the application
- **APPLICATION_LOGO** The url of the application logo
- **MODULES** A list of modules to load. ["multi_locus_view"] by default

3.2.5 Email Settings

- **MAIL_SERVER** The email server, smtp.gmail.com by default
- **MAIL_PORT** The email server port, 587 by default
- **MAIL_USE_SSL** False by default
- **MAIL_USE_TLS** True by default
- **MAIL_USERNAME** The mail username . The dummy entry `mlv@gmail.com` is the default.
- **MAIL_PASSWORD** password by default
- **HELP_EMAIL_RECIPIENTS** A list of email addressees to which user questions are sent (an empty list by default)
- **MAIL_DEFAULT_SENDER** The name to attach to emails that are sent out. 'The MLV Team' by default.

3.2.6 Misc. Setting

- **SECRET_KEY** Enables secure password hashing, should be sent to large random string
- **JS_VERSION** Should be changed each time a new version is rolled out as this will cause existing js and css caches on the user's computers to be refreshed.

3.3 Executing Scripts

To run any script requires the environment variable `FLASK_APP` to point to the module `cli_commands.py`. Other environment variables that may be required include `FLASK_CONFIG`, which points to a custom config and `DATABASE_PASS`, which holds the database password.

```
export FLASK_APP=/path/to/install/app/commands/cli_commands.py
export FLASK_CONFIG=linux_test_config
export DATABASE_PASS="pword"
```

3.3.1 Writing scripts

To write a script, simply import app from *cli_commands.py* and wrap your code in the app context.

```
from app.commands.cli_commands import app
from app.jobs.jobs import get_job
from app.ngs.project import get_project

with app.app_context():
    j=get_job(1234)
    j.process()
    p=get_project(4321)
    p.delete(True)
```

3.3.2 Built in Scripts

There are a number of utility scripts in *cli_commands.py* that can be run with

```
flask name_of_script --param value
```

create_new_genome_database

creates a new empty genome database.

- - *-db_name* - The name of the database

add_new_genome

Adds a genome to the specified database. If the name matches a public genome in the UCSC genome browser, the RefSeq genes and chromosome file will automatically be added. Otherwise, the chromosome file (tab delimited chromosome to length) can be added manually to *data_root/<genome_name>/<genome_name>/chrom.sizes*.

- - *-name* - The name of the database (required)
- - *-label* - The label e.g. Human(hg19) (required)
- - *-icon* - The url of an icon to represent the database (24px x 24px). If not supplied a default icon will be used
- - *-database* - The name of the database to store the genome in (required)
- - *-connections* - The numner of connections optional - default 5

run_app

Runs the app on the local host.

- - *-port* - The port

runcelery

Runs the message queue

- - *-queue* - The name of the queue . the default is celery, the other oprion is slow_queue
- - *-threads* - The number of threads to give the queue - optional (default is 3)

remove_deleted_projects

Removes all projects (and associated jobs) that are tagged as deleted. All data associated with the project is permanantly deleted.

check_all_jobs

Calls *check_status* on all running jobs. This is not required for jobs in the local queue, only those running on remote servers that have their *check_process* method overwritten. In which case this script should be run at frequent time intervals e.g. in crontab

find_or_create_user

Manually adds a user to the database

- - *-first_name* - The user's first name (required)
- - *-last_name* - The user's last name(required)
- - *-email* - The user's email(required)
- - *-password* - The user's password (required)
- - *-admin* - If True,true or TRUE,the user will habe admin rights (default False)

3.4 Modules

Modules are a way of creating independent applications with discrete templates (html), static files (js,css and images) and python modules. A module can be added to a system by simply adding the module folder to the *app/modules* directory and then adding the name of the folder(module) to the MODULES list in the app's config.

3.4.1 Folder Structure

```
app
|--modules
  |--module_name
    |--jobs
    |--projects
    |--static
    |--templates
    |--__init__.py
    |--config.json
```

3.4.2 Templates

The templates folder should contain subfolders named after each project in the module. Each subfolder should contain *home.html* (see [Project Home Page](#)) as well as any other templates required by the project. Templates are referenced in the normal way. e.g the file *template.html* in the subfolder *project1* of the *templates* directory

```
/app/modules/<module_name>/templates/project1/template.html
```

would be referenced in the view method as:-

```
get_template(self,args):
    return "project1/template.html"
```

3.4.3 Static Files

Any static files (js,css,images) go in the static subfolder of the module and can be referenced from template files by prefixing the module name before static e.g. the js file

```
/app/modules/static/myjsfile.js
```

would be accessed by:-

```
<script src="/<module_name>/static/myjsfile.js?version={{config['JS_VERSION']}}"></
<script>
```

3.4.4 Projects

Any projects need to be specified in a dictionary in the projects list of the module's config see [App Settings](#). This will ensure the project is imported and registered when the app is initialised. The actual project code needs to be in a python file named after the project in the module's project folder - see [Project Class](#).

3.4.5 Jobs

Jobs need to be specified in the 'jobs' list of the module's config. The job's code should then be in a module named after the job in the module's jobs folder - see [Jobs](#).

3.4.6 Config

The config should have three keys:- jobs, projects and config. The jobs and projects specify the jobs and projects that the module contains and the config will update the app's config with any extra variables required. For example:-

```
"jobs": [
    {"name": "peak_search_job"}
],
"projects": [
    {
        "name": "peak_search",
        "label": "Peak Search",
        "large_icon": "/lanceotron/static/img/peak_search.png",
        "can_create": true,
```

(continues on next page)

(continued from previous page)

```
        "is_public":true,
        "main_project":true,
    }
],
"config":{
    "NEW_APP_PARAMETER":"value"
}
```

3.5 Projects

A Project represents an analysis or pipeline and is represented by a JSON config, a python class, html (Jinja) templates and JavaScript files. The metadata for each project is kept in the 'projects' table of the system database.

3.5.1 Config

Each project needs to be described by an entry in the project's list of a module's config. The config should contain the following:-

- **name** - The name of the project type (that will be stored in the database as type)
- **label** - The name shown to user.
- **large_icon** - The url of the icon which is displayed in the panels on the main page.
- **can_create** If True then this type of project can be directly created from the home page.
- **description** A short description which is displayed in the create panel on the main page.
- **is_public** If False, then the user must have the permission 'view_project_type' with the value of the project's type.
- **main_project** If True, then individual projects of this type will be accessible to view from the home page.
- **enter_genome** (optional) - If True then the genome can be entered during the initial creation page, usually only name and description can be entered.
- **anonymous_create** (optional) If True, then projects of this type can be created by an anonymous user that is not logged in.

Users have to be given a specific permission to create a project. If the project type is public, new users will automatically get permission to create that project type.

3.5.2 HTML Templates

Project Home Page

Projects that can be created directly will have the following url:

```
http://<server_name>/projects/<project_type>/home
```

This page allows the user to enter a name, description and genome and then creates an empty project. The actual template file should be located at

```
app/modules/<module_name>/templates/<project_name>/home.html
```

and contain the following html :-

```
{% extends "projects/home_base.html" %}
{% block project_explanation %}
    Information about the project here
{% endblock %}
```

Individual Project Page

A second url points to a specific instance of the project:-

```
http://<server_name>/projects/<project_type>/<project_id>
```

The flask view behind this url checks the user has permission to view/edit the project and calls the project's `get_template` method. The template is then rendered with the following kwargs (plus any extra returned by the `get_template` method)

- `project_id`
- `project_type`
- `project_name`
- `project_description`

The project's `get_template` method receives the args from the request and should return the location of the template and a dictionary containing any key word arguments (in addition to the above) required by the template. Different templates can be returned depending on the state of the project.

```
def get_template(self, args):
    kwgs={}
    template="<project_name>/<template_name>.html"
    #alter template and kwgs based on args and the project's current state
    return template, kwgs
```

The html templates for each project need to be located in the directory

```
app/modules/<module_name>/<project_name>/
```

with the following template:-

```
{% extends "common/page_base.html" %}

{% block stylesheets %}
    {{super() }}
    <!-- extra styles -->
{% endblock %}

{% block outercontent %}
    <!-- main content -->
{% endblock %}

{% block scripts %}
    {{ super() }}
```

(continues on next page)

(continued from previous page)

```
<!-- scripts -->
{% endblock %}
```

3.5.3 Project Class

Projects should inherit from the `GenericObject` in `app.ngs.project`, which supplies methods amongst others for sharing, making public, deleting projects. In addition the `projects` member of `app.ngs.project` should be updated with the class name

To expose a project's method to an HTML Client use the static member 'methods', which is a dictionary with the exposed method as the key and a dictionary containing the following parameters:-

- **permission required, either view or edit**
 - view - allow all users with view permission to access the method. This will include all users if the object is public
 - edit - allow only users with edit permission to access the method
- **async optional, either True or False (False by default)**
 - False - The method will be run in the browser thread
 - True - The method will be processed asynchronously
- **running_flag optional**, The projects's data will have this parameter set to the supplied value immediately before the method is sent to any queue. Should be a list containing the parameter and value to set

As an example a project `new_project` with a single method `get_data` that can be accessed by all users (including anonymous ones) would be written as follows:-

```
from app.ngs.project import GenericObject, projects

class CaptureCompareProject(GenericObject):
    def get_template(self, args):
        return "new_project/temp.html", kwgs
    def get_data(self, param1="default", param2="default"):
        #get the data using params
        return data

projects["new_project"] = NewProject

NewProjects.methods = {
    "filter_peaks": {
        "permission": "view",
        "running_flag": ["filter_status", "filtering"]
    }
}
```

The method can then be called from JavaScript using:-

```
$.ajax({
    url: "/meths/execute_project_action/<project_id>",
```

(continues on next page)

(continued from previous page)

```

type:"POST",
dataType:"json",
contentType:"application/json",
data:JSON.stringify({
    method:"get_data",
    args:{
        param1:"value1",
        param2:"value2"
    }
})
})

```

The ‘args’ parameter contains the key word arguments sent to the project’s method. However, if ‘project_data’ is present in the args then this will be used to update the project’s data immediately and not passed to the method. This is useful if the method is being run asynchronously with the async flag.

3.6 Jobs

Jobs run tasks asynchronously, either locally through celery or remotely on another server. Local jobs should extend *LocalJob* from *app.jobs.jobs*, whilst remote jobs should extend *BaseJob*

3.6.1 Constructor

To construct a job, user_id, inputs (a dictionary of key/value pairs) and genome should be specified although default values of 0, an empty dictionary and ‘other’ will be used. A type, however must be specified. Inputs should contain all the information required to send or resend the job. Once a job is constructed in this way, it will be added to the database and can be retrieved in the future with *app.jobs.jobs.get_job(<job_id>)*.

3.6.2 Registering a job

In a module, each job should be in separate python file in the jobs folder of the module with the name of the job type e.g. new_job.py and registered in the config:-

```

{
    .....,
    "jobs":[
        {"name":"new_job"}
    ],
    .....
}

```

The job’s type needs to be linked to its class using *job_types* from *app.jobs.jobs*. For example new_job.py could contain:-

```

from app.jobs.jobs import BaseJob, job_types
import traceback

class NewJob(LocalJob):
    def __init__(self, job=None, inputs={}, user_id=0, genome="other"):
        if (job):

```

(continues on next page)

(continued from previous page)

```
        super().__init__(job=job)
    else:
        super().__init__(inputs=data,genome=genome,type="new_job",user_id=user_id)

    def process(self):
        try:
            #run the job
        except:
            self.failed(traceback.format_exc())

job_types["type"] = JobClass
```

3.6.3 Methods to Override

send(data)

For local jobs this simply calls *process* asynchronously in the celery thread. For remote jobs, this method needs to be over-ridden in the subclass to call a pipeline on a remote server add it to a remote queue etc. Usually no parameters need to be passed as they should all be stored in the database when the job is created. However, data can be passed for example a password, which you would not want to store.

resend()

The default implementation is just to call *send*. However it should be over-ridden to clean up any mess that was created when the job was first sent.

check_status()

The default is to return the job's status (the field in the underlying database). This is sufficient for local jobs, as the status is updated by the celery thread. For Remote jobs this method should just return the status if it is 'complete', 'new', 'processing' or 'failed'. Otherwise (i.e. the job is still running) it should actually check on the status of the remote script/pipeline (query the remote server, check the queue etc.) and if the job has failed or is complete, call *failed* or *process* respectively.

process()

For local jobs this should be the meat of the job and do all the heavy lifting ,storing the results to the database. For remote jobs this should take the results from the remote pipeline and process/store them in the database. Exceptions should be caught and if catastrophic, call *failed* passing the description of the exception.

delete()

This method just removes the job from the database. Sub-classes should over-ride this method and delete any files or other resources before removing the database entry.

failed(msg)

sets the status and time finished, writing the error message to the database. If there is more cleaning up required then this method should be over-ridden with the relevant code.

kill()

The default implementation of this method is simply to set the job's status as failed and add a message to the outputs. For remote jobs it should send a message to the server/queue to kill the job. For local jobs - if possible the job's status should be checked periodically and processing stopped if the status is 'failed'.

3.6.4 Utility Methods

Job objects have the instance variable 'job', which is just an SQLAlchemy object referring to the database entry. This can be manipulated directly or there are the following convenience methods:-

- **complete** Sets date finished and status fields in the database
- **has_permission(user)** Returns whether the user has permission for this job
- **set_input_parameter(param,value)** Sets the input parameter
- **set_output_parameter(param,value)** Sets the output parameter
- **get_input_parameter(param,value)** Gets the input parameter
- **get_output_parameter(param,value)** Gets the output parameter
- **get_user_name()** Gets the full name of the job's owner
- **set_status(self,value)** Sets the status in the database
- **get_info()** Returns a dictionary containing the job's inputs and outputs

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`