Creates GUI Component

__init__
- Initialize component and store app_state
- Declares all the GUI component variables
- Calls **create_components**
- Calls **setup_layout**

create_widgets
- Initializes all of the GUI component variables
- Adds bindings to required GUI components
- Adds the styling arguments to GUI components

setup_layout
- Creates a row and column configuration for the GUI
- Adds all of the GUI components to itself

Creates Plots During Runtime

Scree Plot
create_scree_plot
- Shows an error if the df has not been cleaned
- Calls **main.run_analysis**
- Calls **main.create_blank_fig(grid=false)**
- Adds a title and an x and y label to the ax
- Gets the explained variance from pca_results in the app_state
- Uses a try block to add the bar and step chart to the ax
- Calls **main.update_figure**

Biplot
create_biplot
- Calls **validate_biplot_data** to get PCA result data
- Calls **init_biplot_fig**
- Calls **get_color_mapping**
- Adds legend box to the ax using color map
- Adds a confidence ellipse to the ax
- Scales Loadings
- Calls **set_biplot_axis_limits**
- Calls **add_biplot_arrows**
- Adds the scatter plot points to ax
- Calls **main.update_figure**

init_biplot_fig

- Calls **main.create_blank_fig**
- Sets title and x and y label on the ax
- Sets the grid appearance and aspect ratio of the ax
- Sets the background color of the ax

set_biplot_axis_limits

- Calculates the minimum and maximum x and y values from the scaled_loadings
- Calculates a margin based on the max x and y range
- Sets the ax axis limits using the x and y min and max values and the margin

add_biplot_arrows

- Creates a list for holding text objects
- Calls **get_color_mapping** to get the feature map and color
- Loops through the indexes in top_idx
    - Gets the feature and magnitude
    - Skips the feature if the magnitude is less than 0.2
    - Gets the group and color
    - Generates and arrow on the ax
    - Gets the text distance from the app_state
    - Generates a text object and adds it to ax
    - Store the text object in the text object list
- Adjusts the text objects to attempt to minimize overlap

Interactive Biplot

create_interactive_biplot

- Calls **validate_biplot_data** to get the PCA results data
- Initialize function internal figure as a Go figure
- Scales Loadings
- Calls **add_interactive_biplot_groups**
- Updates the figure layout with a menu and sliders
- Updates the figure layout with the top 2 PCA components
- Calls **save_interactive_plot** from file_operations
- Shows a success message if the plot saved properly

add_interactive_biplot_groups

- Calls **get_color_mapping** to get the feature map and color
- Creates a set for groups on the legend
- Loops through the indexes in top_idx
    - Gets the feature and magnitude
    - Skips if the magnitude is less than 0.2
    - Gets the group and color of the feature
    - Determines if the group is already in the legend set
    - Adds the group to the legend set

    - Adds a (arrow) trace to the figure and adds group to the legend if it hasn't already

Top Feature Plot
create_top_n_feat_plot

- Shows an error if the df has not been cleaned
- Calls **main.run_analysis**
- Gets top_n_feat from app_state
- Gets pca_comp_num from app_state
- Calls **init_top_feat_plot**
- Gets the loadings and feat_names from pca_results in app_state
- Sorts the loading and feature names
- Adds a bar chart to the ax
- Calls **main.update_figure**
- Shows a message that the plot was generated successfully

init_top_feat_plot

- Calls **main.create_blank_fig(grid=False)**
- Sets the title and the x and y labels on the ax
- Adjusts figure tick label size and rotation to fit on the ax

Data Functions

validate_biplot_data

- Shows an error if the df has not been cleaned
- Shows an error if custom feature groups are enabled, but have not been uploaded
- Calls **main.run_analysis**
- Gets the pca_results from app_state
- Gets scores, loadings, explained_varience, and feature names from pca_results
- Gets the user input for number of features
- Calculates eigenvalues from variance
- Calculates magnitudes from loadings
- Gets the indexes and features with the top magnitudes
- Returns scores, loadings, variance, eigenvalues, feature names, top indexes, magnitudes, and number of features.

get_color_mapping

- Gets the current color palette selected by the user
- If feature grouping is enabled
    - Shows and raises an error if no feature group has been loaded
    - Gets the groups from the feature group map
    - Splits the groups into predefined and undefined groups. Where predefined is groups that have color assignments from the selected color palette and undefined groups do not.

        - Creates a dictionary and adds the predefined groups with the colors from the color palette
        - Calls **map_generic_colors(undefined_groups)**
        - Adds the results to the dictionary and returns the dictionary
- else
    - Checks that features have been provided
    - Calls and Returns **map_generic_colors(feat)**

map_generic_colors

- Checks that 20 or less features have been provided
- Collects a color blind friendly palette, plt ‘tab10’ if 10 or less features are provided
- Otherwise collects a non-color blind friendly palette, plt ‘tab20
- Creates a dictionary mapping each of the features to its own color
- Returns the dictionary

upload_mapping

- Opens a filedialog asking the user to select a csv file
- Shows an error message and returns if the user doesn’t select a file
- Shows and error message and returns if the user selected file is not a csv file
- Reads the csv file and stores it in a df with lowercase column names
- Shows an error message if the df doesn’t contain a ‘feature’ and ‘group’ column
- Generates a feature mapping using the df and saves it in app_state
- Gets the unique groups from the df
- Creates a group color map using tab20 colors and the unique groups
- Stores the group color map to app_state
- Shows a success message