Front End: Tkinter GUI

The front end provides an intuitive graphical user interface with a multi-page layout. Each page corresponds to a specific task or process and dynamically updates based on user interaction. Key functionalities include:

- **File Uploading**:
  - Users can upload *.csv* or *.zip* files through the GUI.
  - *.csv* files are routed to data analysis processes.
  - *.zip* files are routed to image processing workflows.
- **Process Selection**:

  For *.csv* files, users choose between:

  - Data Filtering & Preprocessing
  - Regression & Classification
  - AI Model-based Analysis
- **Dynamic UI Rendering**:

  UI elements (e.g., sliders, dropdowns, radio buttons) change depending on the selected process or model type.
- **Result Display**:

  Graphs (e.g., regression plots, residuals, confusion matrices), metrics, and predictions are rendered directly within the GUI using embedded plotting libraries such as `matplotlib`.

class **main_sphinx.App** [source]

Bases: `CTk`

Main application class for the Tkinter-based GUI.

This class initializes the GUI, handles page navigation, and manages assets.

> **current_page**
>
> Stores the current active page in the application.
>
> **Type:** ctk.CTkFrame

configure_grid() [source]

Configures the layout grid of the application.

This method sets up the grid structure for proper UI arrangement.

create_sidebar() [source]

Creates the sidebar navigation panel.

This method initializes navigation buttons for different pages.

load_assets() [source]

Loads all image assets used in the application.

This method loads required images for the UI elements.

load_header_image() [source]

Loads and resizes the header image.

This method retrieves the header image, resizes it, and assigns it to the header label.

resize_header_image(*event*) [source]

Resizes the header image dynamically when the window is resized.

Parameters: **event** (*tk.Event*) – The resize event that triggers this function.

show_page(*page_name*, **args*, ***kwargs*) [source]

Handles navigation between different pages.

Parameters:
- **page_name** (*str*) – The name of the page to display.
- ***args** (*tuple*) – Additional arguments passed to the page class.
- ****kwargs** (*dict*) – Keyword arguments passed to the page class.

toggle_mode() [source]

Toggles between Light and Dark mode.

---

class **pages.aimodel_page.AImodelPage**(*parent*, *file_data=None*, *file_name=None*, **args*, ***kwargs*) [source]

Bases: `CTkFrame`

A custom Tkinter frame for configuring and submitting AI models.

This class provides a GUI interface for selecting and configuring machine learning models (RandomForest, CatBoost, Artificial Neural Network, XGBoost), adjusting their hyperparameters, and submitting them to a backend for training or evaluation.

Parameters:
- **parent** – The parent tkinter container.
- **file_data** – The processed data to be used by the models.
- **file_name** – The name of the file associated with the data.

`change_segment`(*segment_name*) [source]

Switch between the different AI model configuration segments.

Parameters: **segment_name** – The selected model name from the segmented button.

`create_ann_frame`() [source]

Create the UI frame for configuring Artificial Neural Network (ANN) hyperparameters.

Returns: A CTkFrame widget containing sliders and dropdowns for ANN.

`create_cb_frame`() [source]

Create the UI frame for configuring CatBoost model hyperparameters.

Returns: A CTkFrame widget containing sliders for CatBoost.

`create_combobox_frame`(*parent*, *label_text*, *options*, *default*, *row*) [source]

Create a dropdown combobox for selecting categorical parameters.

Parameters:
- **parent** – The parent widget to attach this frame.
- **label_text** – The label describing the dropdown.
- **options** – List of available options.
- **default** – Default selected option.
- **row** – Row position inside the parent grid.

`create_info_button`(*parent*, *text*) [source]

Create a small button with an info icon that triggers a popup.

Parameters:
- **parent** – The parent widget to attach this button.
- **text** – The info message to display when clicked.

`create_rf_frame`() [source]

Create the UI frame for configuring RandomForest model hyperparameters.

Returns: A CTkFrame widget containing sliders for RandomForest.

create_slider_frame(*parent*, *model_name*, *label_text*, *from_*, *to*, *default*, *row*) [source]

Create a slider UI element for numeric hyperparameters.

Parameters:
- **parent** – The parent widget to attach this frame.
- **model_name** – Name of the model this parameter belongs to.
- **label_text** – The label describing the slider parameter.
- **from** – Minimum value for the slider.
- **to** – Maximum value for the slider.
- **default** – Default value of the slider.
- **row** – Row position inside the parent grid.

create_xgb_frame() [source]

Create the UI frame for configuring XGBoost model hyperparameters.

Returns: A CTkFrame widget containing sliders for XGBoost.

get_combobox_value(*combobox_name*) [source]

Helper to get the value of a specific combobox.

Parameters: **combobox_name** – The name of the combobox.
Returns: The selected value if found, else None.

get_slider_value(*slider_name*) [source]

Helper to get the value of a specific slider.

Parameters: **slider_name** – The name of the slider.
Returns: Slider value if found, else 0.

messagebox
= <module 'tkinter.messagebox' from 'C:\\Users\\USER\\.conda\\envs\\oopDevEnv\\lib\\tkinter\\messagebox.py'>

preview_data() [source]

Open a popup window displaying the first 50 rows of the processed dataset. Provides a scrollable view using ttk.Treeview.

send_request(*json_data*) [source]

Send a POST request with the selected model data to the Django backend.

Parameters: **json_data** – The full dictionary containing model configuration and dataset.

`show_info_dialog`(*text*) [source]

Show an information popup with a provided text.

Parameters: **text** – The info message to display in the dialog.

`submit_action`() [source]

Handle the submit button click by collecting all model configurations, packaging them into a JSON structure, and sending a request to the backend.

---

class `pages.data_filtering_page.DataFilteringPage`(*parent*, *file_path*, *file_name=''*) [source]

Bases: `CTkFrame`

A Tkinter-based GUI page for performing various data filtering processes including Outlier Detection, Interpolation, Smoothing, and Scaling & Encoding.

Parameters:

- **parent** – The main application container.
- **file_path** – Path to the uploaded CSV file.
- **file_name** – Name of the file (optional).

`add_export_send_buttons`() [source]

Adds export, compare, and send buttons at the bottom of the UI.

`change_segment`(*segment_name*) [source]

Changes the active segment based on user navigation.

Parameters: **segment_name** – Name of the segment to activate.

`create_interpolation_frame`() [source]

Creates the Interpolation segment with radio button method selection.

Returns: CTkFrame with interpolation method options.

`create_process_selection_frame`() [source]

Creates the initial selection frame for choosing filtering vs scaling.

Returns: CTkFrame with radio button options.

`create_scaling_encoding_frame`() [source]

Creates the Scaling & Encoding segment with sliders for test size and random state.

Returns: CTkFrame containing configuration widgets.

create_segment_frame() [source]

Creates the Outlier Detection frame with method options and slider.

Returns: CTkFrame containing radio buttons, slider, and column checkboxes.

create_smoothing_frame() [source]

Creates the Smoothing segment, including SMA and TES options.

Returns: CTkFrame with smoothing controls.

export_data() [source]

Exports processed data to CSV depending on the most completed step.

get_data_by_name(*name*) [source]

Fetches a DataFrame based on the dataset name string.

Parameters: **name** – Dataset name like “Raw Data”, “Interpolated Data” etc.
Returns: Corresponding Pandas DataFrame.

load_csv_columns(*file_path*) [source]

Loads column names from CSV and sets up dropdowns and checkboxes.

Parameters: **file_path** – Path to the CSV file.

lock_segment(*frame*) [source]

Locks a segment UI to prevent changes after completion.

Parameters: **frame** – The segment frame to disable.

move_to_next_segment() [source]

Moves the user to the next logical segment based on progress. Updates visibility of segment buttons.

open_comparison_popup() [source]

Opens a popup window to compare different stages of data processing.

open_send_popup() [source]

Opens a destination selection popup after scaling & encoding.

plot_boxplot(*column_name*, *cleaned=False*) [source]

Generates a boxplot for a selected column with optional cleaned data.

Parameters:
- **column_name** – Name of the column to visualize.
- **cleaned** – Whether to use cleaned data or not.

plot_comparison_graph(*left_data*, *right_data*, *column_name*) [source]

Plots graph comparison of two data stages using boxplot or line chart.

Parameters:
- **left_data** – Name of the left dataset.
- **right_data** – Name of the right dataset.
- **column_name** – Column to plot.

plot_line_graph(*column_name*, *original_data*, *processed_data*, *title*) [source]

Plots line graphs comparing original and processed data.

Parameters:
- **column_name** – Name of the data column.
- **original_data** – The original data series.
- **processed_data** – The processed data series.
- **title** – Plot title.

preview_csv() [source]

Opens a popup window showing the first 50 rows of the uploaded CSV.

preview_scaled_encoded_data() [source]

Displays the scaled and encoded data in a table popup.

run_interpolation() [source]

Executes interpolation on cleaned data and updates graph.

run_outlier_detection() [source]

Triggers outlier detection and updates cleaned data based on user selections.

run_scaling_and_encoding() [source]

Executes scaling and encoding on the latest available dataset (raw or processed).

run_smoothing() [source]

Applies smoothing (SMA or TES) on interpolated data and updates visuals.

send_request(*process_name*, *json_data*) [source]

Sends a request to the backend with specified process name and data.

Parameters:
- **process_name** – The name of the backend API process.
- **json_data** – The data payload as dictionary.

Returns: Parsed JSON response from backend.

show_comparison_data(*left_data*, *right_data*, *column_name*) [source]

Displays side-by-side table comparison of two datasets.

Parameters:
- **left_data** – Name of the first dataset.
- **right_data** – Name of the second dataset.
- **column_name** – Column to compare.

show_info_dialog(*text*) [source]

Displays an information popup with help text.

Parameters: **text** – The text to show in the popup.

show_loading_message() [source]

Displays a loading message in the graph area.

submit_action() [source]

Handles submission logic for each segment and moves to the next step. Validates inputs and triggers processing functions.

toggle_slider(*frame*, *show*) [source]

Show or hide contamination slider based on outlier method.

Parameters:
- **frame** – The parent frame.
- **show** – Boolean to show or hide.

toggle_smoothing_options(*frame*, *method*) [source]

Toggles visibility of smoothing options based on selected method.

Parameters:
- **frame** – The parent frame.
- **method** – Either 'SMA' or 'TES'.

update_boxplot(*column_name*) [source]

Asynchronously updates the boxplot for a selected column.

Parameters: **column_name** – The name of the column to plot.

update_buttons_visibility() [source]

Updates visibility of export, compare, and send buttons.

update_comparison_view() [source]

Handles the action of switching between data and graph views in the compare popup.

update_dropdown(*selected_columns*) [source]

Updates the dropdown values and behavior based on selected columns.

Parameters: **selected_columns** – List of column names to include in dropdown.

---

class **pages.file_upload_page.FileUploadPage**(*parent*) [source]

Bases: `CTkFrame`

File upload interface for CSV and ZIP files. Routes uploaded files to the appropriate next page depending on file type.

Parameters: **parent** – The parent application container.

upload_file() [source]

Handles file upload via a file dialog and routes to the correct page based on file type (.csv or .zip). Stores file path and name in the main application state.

---

class **pages.image_processing_page.ImageProcessingPage**(*parent*, *file_path=None*, *file_name=None*, *data=None*, ***page_state*) [source]

Bases: `CTkFrame`

CustomTkinter page for training an image classification model. Allows user to configure activation, optimizer, and dataset splitting.

Parameters:

- **parent** – Parent widget.
- **file_path** – Path to the dataset ZIP.
- **file_name** – Optional display name.
- **data** – Placeholder for future data passing.
- **page_state** – Page-specific settings (e.g. saved state).

cancel_file() [source]

Handles file cancellation and resets only this page.

change_segment(*segment_name*) [source]

Switches UI to show only the active segment.

Parameters: **segment_name** – Label of the selected segment.

create_image_train_frame() [source]

Builds a small image upload and preview block.

Returns: CTkFrame containing upload and preview buttons.

create_segment_frame() [source]

Creates UI segment to configure activation, epochs, optimizer, test size, and random seed.

Returns: CTkFrame with interactive settings.

plot_confusion_matrix(*cm_data*) [source]

Renders a matplotlib confusion matrix inside a popup.

Parameters: **cm_data** – Dictionary with matrix values and labels.

preview_image() [source]

Opens a new preview window with the uploaded image. Resizes image to fit window if necessary.

show_info_dialog(*text*) [source]

Displays a popup window with explanatory text.

Parameters: **text** – Message content to show.

submit_action() [source]

Sends selected image classification settings to the backend for training. Logs training info and receives metrics for display.

upload_image() [source]

Opens a file picker dialog and stores uploaded image path. Enables preview button after selection.

class **pages.process_selection_page.ProcessSelectionPage**(*parent*, *file_path*, *filename*) [source]

Bases: `CTkFrame`

This module implements the Process Selection Page using CustomTkinter. It allows users to choose between Data Filtering, Regression & Classification, and AI Model training.

A GUI page that allows users to select the next processing step for the uploaded file.

parent

The parent application window.

Type: ctk.CTk

file_path

Path of the uploaded file.

Type: str

filename

Name of the uploaded file.

Type: str

__init__(*parent*, *file_path*, *filename*) [source]

Initializes the Process Selection Page UI.

create_buttons(*parent*, *file_path*, *filename*) [source]

Creates buttons for different processing options.

Parameters:
- **parent** (*ctk.CTk*) – The parent application window.
- **file_path** (*str*) – Path of the uploaded file.
- **filename** (*str*) – Name of the uploaded file.

class **pages.regression_classification.RegressionClassificationPage**(*parent*, *file_path=None*, *file_name=None*, *data=None*, ***page_state*) [source]

Bases: `CTkFrame`

GUI page for configuring and submitting regression and classification models. Supports various regression (Linear, Polynomial, Ridge, Lasso) and classification (RandomForest, SVC, KNN) models.

Parameters:
- **parent** – Main parent application.

- **file_path** – Path to the uploaded file (optional).
- **file_name** – Name of the uploaded file.
- **data** – Preprocessed dataset.
- **page_state** – State dictionary for restoring prior settings (if any).

cancel_file() [source]

Cancels the current file and resets this page state to its default. Clears file paths, sidebar references, and page data.

change_segment(*segment_name*) [source]

Handles switching between Regression and Classification segments.

Parameters: **segment_name** – Either 'Regression' or 'Classification'.

clear_frame(*frame*) [source]

Clears the contents of a given frame.

Parameters: **frame** – The CTkFrame widget to be cleared.

create_classification_frame() [source]

Creates the Classification segment frame with dynamic parameters.

Returns: A CTkFrame containing radio buttons and config for classification models.

create_dropdown(*parent*, *label_text*, *options*) [source]

Creates a labeled dropdown menu.

Parameters:
- **parent** – Container for the dropdown.
- **label_text** – Label to display above the dropdown.
- **options** – List of dropdown options.

create_info_button(*parent*, *text*) [source]

Creates an info button to show contextual information.

Parameters:
- **parent** – The frame to place the button in.
- **text** – Info text to show when clicked.

create_regression_frame() [source]

Creates the Regression segment frame with dynamic parameters.

Returns: A CTkFrame containing radio buttons and config for regression models.

create_slider(*parent*, *label_text*, *min_val*, *max_val*, *default*) [source]

Creates a labeled slider with value label and info button.

Parameters:
- **parent** – The container frame.
- **label_text** – Text to describe the slider.
- **min_val** – Minimum value.
- **max_val** – Maximum value.
- **default** – Default slider position.

create_textbox(*parent*, *label_text*, *mode*) [source]

Creates a labeled textbox for custom input parameters.

Parameters:
- **parent** – The container frame.
- **label_text** – The textbox label.
- **mode** – Input mode ('polynomial' or 'alpha') for validation. - "polynomial": Allows 1-5 single-digit numbers (0-9), separated by commas. - "alpha": Allows 1-5 float values (0-1) with at least 4 decimal places, separated by commas.

lasso_plot(*results_lasso*, *best_params*) [source]

Plots Lasso Regression performance with alpha values against R2 scores.

Parameters:
- **results_lasso** – Dictionary containing cross-validation results.
- **best_params** – Dictionary with best_degree_lasso and best_alpha_lasso.

polynomial_plot(*x_scatter*, *y_scatter*, *y_poly*, *x_label*, *y_label*, *degree*) [source]

Generates and displays a polynomial regression plot overlaying actual vs predicted values.

Parameters:
- **x_scatter** – Input feature values.
- **y_scatter** – Actual target values.
- **y_poly** – Predicted values from polynomial regression.
- **x_label** – Label for x-axis.
- **y_label** – Label for y-axis.
- **degree** – Polynomial degree used in the model.

preview_data() [source]

Opens a new popup window to display the scaled and encoded data.

regression_plot(*x*, *y*, *x_label*, *y_label*, *data=None*, *ax=None*) [source]

Displays a linear regression plot with regression line and scatter points.

Parameters:
- **x** – Feature values (can be list, dict, or array).
- **y** – Target values (can be list, dict, or array).
- **x_label** – Label for x-axis.
- **y_label** – Label for y-axis.
- **data** – Optional dataset (not used).
- **ax** – Matplotlib axis to draw on.

`residual_plot`(*x*, *y*, *ax=None*) [source]

Creates a residual plot showing prediction errors.

Parameters:
- **x** – Predicted values (can be list, dict, or array).
- **y** – Actual target values (can be list, dict, or array).
- **ax** – Optional Matplotlib axis.

`ridge_plot`(*results_ridge*, *best_params*) [source]

Plots Ridge Regression performance with alpha values against R2 scores.

Parameters:
- **results_ridge** – Dictionary containing cross-validation results.
- **best_params** – Dictionary with best_degree_ridge and best_alpha_ridge.

`send_request_classification`(*json_data*) [source]

Sends a POST request with classification model data to the Django backend.

Parameters: **json_data** – JSON-serializable data object to send to the backend.

`send_request_regression`(*json_data*) [source]

Sends a POST request with regression model data to the Django backend. Handles and visualizes results for Linear, Polynomial, Ridge, and Lasso regressions.

Parameters: **json_data** – JSON-serializable data object to send to the backend.

`show_info_dialog`(*text*) [source]

Displays an information popup.

Parameters: **text** – The text to show in the info dialog.

`submit_action`() [source]

Handles submission of configured model parameters. Builds DataObject and sends it to backend.

toggle_classification_options() [source]

Updates the Classification options dynamically based on selection.

toggle_regression_options() [source]

Updates the Regression options dynamically based on selection.

---

class **pages.help_page.HelpPage**(*parent*) [source]

Bases: `CTkFrame`

This module implements a simple Help Page using CustomTkinter. A class representing the Help Page in the GUI.

This page provides user assistance and guidance.

None

__init__(*parent*) [source]

Initializes the Help Page.

Back End: Django API

The Django backend serves as the processing engine and is structured around modular views and serializers. Key responsibilities include:

- **Process Handling**:

  Performs data transformations, model training, predictions, and visualizations.

- **Modular APIs**:

  Four dedicated API routes handle distinct functionality:

  1. **Data Filtering & Preprocessing**
  2. **Regression & Classification**
  3. **AI Models**
  4. **Image Processing**

---

class **backend.api.image_processing_engine.ImageProcessingAPIView**(**args*, ***kwargs*) [source]

Bases: `APIView`

API endpoint to process zipped image datasets for training a CNN-based image classifier.

The pipeline includes: 1. Loading and preprocessing zipped image data 2. Splitting dataset 3. Creating a CNN model 4. Training the model 5. Evaluating test accuracy/loss 6. Generating a confusion matrix

post(*request*) [source]

Handle POST request containing DataObject with zipped image dataset and model parameters.

Expects structure: - dataobject[image_processing][fileio, train_test_split, model_params, training_params]

Parameters: **request** – HTTP request from frontend.
Returns: Response containing test accuracy, loss, and confusion matrix.

---

class **backend.api.data_preprocessing_engine.DataFilteringFileAPIView**(**args*, ***kwargs*) [source]

Bases: `APIView`

post(*request*) [source]

run_outlier_detection(*dataset*, *data_object*) [source]

Runs the outlier detection based on the method defined in DataObject.

---

class **backend.api.data_preprocessing_engine.InterpolationAPIView**(**args*, ***kwargs*) [source]

Bases: `APIView`

post(*request*) [source]

run_interpolation(*cleaned_outlier_data*) [source]

Runs the cubic spline interpolation on the dataset after outlier detection.

---

class **backend.api.data_preprocessing_engine.SmoothingAPIView**(**args*, ***kwargs*) [source]

Bases: `APIView`

post(*request*) [source]

run_smoothing(*interpolated_data*, *data_object*) [source]

Runs the smoothing process based on the method defined in DataObject.

class **backend.api.ai_models_engine.AIModelAPIView**(**args*, ***kwargs*) [source]

Bases: `APIView`

API endpoint to handle training and evaluation of AI models.

Supports classification and regression models: - RandomForest - CatBoost - Artificial Neural Network (ANN) - XGBoost

Uses user-selected hyperparameters and preprocessed data to fit and evaluate the chosen model.

extract_hyperparameters(*data_object*, *model_name*) [source]

Extract user-defined hyperparameters from the data_object or fallback to defaults.

Parameters:
- **data_object** – Structured DataObject containing model info.
- **model_name** – Name of the selected model.

Returns: Dictionary of validated hyperparameters.

post(*request*) [source]

Handle POST request from the frontend containing a *dataobject*.

Parameters: **request** – HTTP request object with model input and configuration.
Returns: Response with model evaluation results.

run_selected_model(*data_object*) [source]

Orchestrates the training and evaluation pipeline for the selected model.

Converts JSON lists back into NumPy arrays, instantiates the model class, trains it, evaluates performance, and returns metrics.

Parameters: **data_object** – DataObject containing training data and parameters.
Returns: Dictionary of results such as accuracy, confusion matrix, or regression scores.

class **backend.api.scaling_encoding_engine.ScalingEncodingAPIView**(**args*, ***kwargs*) [source]

Bases: `APIView`

API endpoint for scaling, encoding, and train-test splitting of preprocessed data.

This endpoint accepts smoothed data and configuration from a DataObject, applies encoding and scaling transformations, splits the dataset, and returns the processed splits ready for training.

post(*request*) [source]

Handle POST request with a DataObject and smoothed dataset.

Performs encoding, scaling, and train-test split using provided parameters.

Parameters: **request** – HTTP request with JSON-encoded data and smoothed data.
Returns: Response containing split, encoded, and scaled data.

`run_encoding_scaling_train_test_split`(*data*, *params*) [source]

Perform encoding, feature scaling, and split into training/testing sets.

Parameters:
- **data** – The input pandas DataFrame.
- **params** – Parameters for preprocessing such as scalers, encoders, test size.

Returns: Dictionary containing split DataFrames for X_train, X_test, y_train, y_test.

---

class `backend.api.regression_engine.RegressionAPIView`(**args*, ***kwargs*) [source]

Bases: `APIView`

API endpoint for training and evaluating regression models on preprocessed data.

Supports: - Linear Regression - Polynomial Regression - Ridge Regression - Lasso Regression

Takes preprocessed training/test sets and regression configuration from a frontend DataObject. Returns appropriate scores, predictions, and hyperparameters.

`post`(*request*) [source]

Handle POST request with dataobject that includes: - Train/test split (X_train, y_train, etc.) - Selected model - Model-specific parameters

Automatically trains and evaluates the chosen model, and returns regression metrics.

Parameters: **request** – HTTP request containing a JSON-serialized DataObject.
Returns: JSON response with evaluation results including R2 score, predictions, and parameters.

---

class `backend.api.classification_engine.ClassificationAPIView`(**args*, ***kwargs*) [source]

Bases: `APIView`

API endpoint to handle training and evaluation of classification models.

Supported Models:

- Random Forest
- Support Vector Classifier (SVC)
- K-Nearest Neighbors (KNN)

Accepts data and model parameters via POST request and returns evaluation metrics.

post(*request*) [source]

Handle POST request to train and evaluate a classification model.

Expects a JSON request with model configuration and split training/testing data inside a *dataobject* structure.

Example

{“dataobject”: {“classification”: {“Model_Selection”: “RandomForest”,”RandomForest”: {“n_estimators”: 100,”max_depth”: 5}},”data_filtering”:”Train-Test Split”: {“split_data”: {“X_train”: […],”X_test”: […],”y_train”: […],”y_test”: […]}}}}}

Parameters: **request** (*Request*) – Django REST framework request containing the dataobject.

Returns: **A JSON response with model evaluation results including:**

- accuracy (float)
- confusion matrix (list)
- mean squared error (float)

Return type: Response

Raises: **ValueError** – If an invalid model name is provided.