Dataframe

`gradio.Dataframe(···)`

Description

Accepts or displays 2D input through a spreadsheet-like component for dataframes.

Behavior

As input: passes the uploaded spreadsheet data as a `pandas.DataFrame`, `numpy.array`, `polars.DataFrame`, or `List[List]` depending on `type`

As output: expects a `pandas.DataFrame`, `pandas.Styler`, `numpy.array`, `polars.DataFrame`, `List[List]`, `List`, a `Dict` with keys `data` (and optionally `headers`), or `str` path to a csv, which is rendered in the spreadsheet.

Initialization

Parameter	Description
`value` *pd.DataFrame | Styler | np.ndarray | pl.DataFrame | list | list[list] | dict | str | Callable | None* **default: None**	Default value to display in the DataFrame. If a Styler is provided, it will be used to set the displayed value in the DataFrame (e.g. to set precision of numbers) if the `interactive` is False. If a Callable function is provided, the function will be called whenever the app loads to set the initial value of the component.
`headers` *list[str] | None* **default: None**	List of str header names. If None, no headers are shown.

meter	Description
`row_count` *int | tuple[int, str]* **default: (1, 'dynamic')**	Limit number of rows for input and decide whether user can create new rows. The first element of the tuple is an `int`, the row count; the second should be 'fixed' or 'dynamic', the new row behaviour. If an `int` is passed the rows default to 'dynamic'
`col_count` *int | tuple[int, str] | None* **default: None**	Limit number of columns for input and decide whether user can create new columns. The first element of the tuple is an `int`, the number of columns; the second should be 'fixed' or 'dynamic', the new column behaviour. If an `int` is passed the columns default to 'dynamic'
`datatype` *str | list[str]* **default: "str"**	Datatype of values in sheet. Can be provided per column as a list of strings, or for the entire sheet as a single string. Valid datatypes are "str", "number", "bool", "date", and "markdown".
`type` *Literal[('pandas', 'numpy', 'array', 'polars')]* **default: "pandas"**	Type of value to be returned by component. "pandas" for pandas dataframe, "numpy" for numpy array, "polars" for polars dataframe, or "array" for a Python list of lists.
`latex_delimiters` *list[dict[str, str | bool]] | None* **default: None**	A list of dicts of the form `"left": open delimiter (str), "right": close delimiter (str), "display": whether to display in newline (bool)` that will be used to render LaTeX expressions. If not provided, `latex_delimiters` is set to `[ "left": "$$", "right": "$$", "display": True ]`, so only expressions enclosed in $$ delimiters will be rendered as LaTeX, and in a new line. Pass in an empty list to disable LaTeX rendering. For more information, see the [KaTeX documentation](https://katex.org/docs/autorender.html). Only applies to columns whose datatype is "markdown".
`label` *str | None* **default: None**	The label for this component. Appears above the component and is also used as the header if there are a table of examples for this component. If None and used in a `gr.Interface`, the label will be the name of the parameter this component is assigned to.

meter	Description
`show_label` *bool | None* **default: None**	if True, will display label.
`every` *float | None* **default: None**	If `value` is a callable, run the function 'every' number of seconds while the client connection is open. Has no effect otherwise. Queue must be enabled. The event can be accessed (e.g. to cancel it) via this component's .load_event attribute.
`height` *int* **default: 500**	The maximum height of the dataframe, specified in pixels if a number is passed, or in CSS units if a string is passed. If more rows are created than can fit in the height, a scrollbar will appear.
`scale` *int | None* **default: None**	relative width compared to adjacent Components in a Row. For example, if Component A has scale=2, and Component B has scale=1, A will be twice as wide as B. Should be an integer.
`min_width` *int* **default: 160**	minimum pixel width, will wrap if not sufficient screen space to satisfy this value. If a certain scale value results in this Component being narrower than min_width, the min_width parameter will be respected first.
`interactive` *bool | None* **default: None**	if True, will allow users to edit the dataframe; if False, can only be used to display data. If not provided, this is inferred based on whether the component is used as an input or output.
`visible` *bool* **default: True**	If False, component will be hidden.
`elem_id` *str | None* **default: None**	An optional string that is assigned as the id of this component in the HTML DOM. Can be used for targeting CSS styles.

ımeter	Description
`elem_classes` *list[str] | str | None* **default: None**	An optional list of strings that are assigned as the classes of this component in the HTML DOM. Can be used for targeting CSS styles.
`render` *bool* **default: True**	If False, component will not render be rendered in the Blocks context. Should be used if the intention is to assign event listeners now but render the component later.
`wrap` *bool* **default: False**	If True, the text in table cells will wrap when appropriate. If False and the `column_width` parameter is not set, the column widths will expand based on the cell contents and the table may need to be horizontally scrolled. If `column_width` is set, then any overflow text will be hidden.
`line_breaks` *bool* **default: True**	If True (default), will enable Github-flavored Markdown line breaks in chatbot messages. If False, single new lines will be ignored. Only applies for columns of type "markdown."
`column_widths` *list[str | int] | None* **default: None**	An optional list representing the width of each column. The elements of the list should be in the format "100px" (ints are also accepted and converted to pixel values) or "10%". If not provided, the column widths will be automatically determined based on the content of the cells. Setting this parameter will cause the browser to try to fit the table within the page width.

Shortcuts

Class	Interface String Shortcut	Initialization
`gradio.Dataframe`	"dataframe"	Uses default values
`gradio.Numpy`	"numpy"	Uses type="numpy"
`gradio.Matrix`	"matrix"	Uses type="array"

s	Interface String Shortcut	Initialization
`gradio.List`	"list"	Uses type="array", col_count=1

Demos

filter_records matrix_transpose tax_calculator sort_records

```
import gradio as gr


def filter_records(records, gender):
    return records[records["gender"] == gender]


demo = gr.Interface(
    filter_records,
    [
        gr.Dataframe(
            headers=["name", "age", "gender"]
```

Event Listeners

Description

Event listeners allow you to capture and respond to user interactions with the UI components you've defined in a Gradio Blocks app. When a user interacts with an element, such as changing a slider value or uploading an image, a function is called.

Supported Event Listeners

The `Dataframe` component supports the following event listeners. Each event listener takes the same parameters, which are listed in the Event Arguments table below.

Listener	Description
`gradio.Dataframe.change(fn, ···)`	Triggered when the value of the Dataframe changes either because of user input (e.g. a user types in a textbox) OR because of a function update (e.g. an image receives a value from the output of an event trigger). See `.input()` for a listener that is only triggered by user input.

Listener	Description
`gradio.Dataframe.input(fn, ···)`	This listener is triggered when the user changes the value of the Dataframe.
`gradio.Dataframe.select(fn, ···)`	Event listener for when the user selects or deselects the Dataframe. Uses event data gradio.SelectData to carry `value` referring to the label of the Dataframe, and `selected` to refer to state of the Dataframe. See EventData documentation on how to use this event data

Event Arguments

Parameter	Description
`fn` *Callable | None | Literal['decorator']* **default: "decorator"**	the function to call when this event is triggered. Often a machine learning model's prediction function. Each parameter of the function corresponds to one input component, and the function should return a single value or a tuple of values, with each element in the tuple corresponding to one output component.
`inputs` *Component | list[Component] | set[Component] | None* **default: None**	List of gradio.components to use as inputs. If the function takes no inputs, this should be an empty list.
`outputs` *Component | list[Component] | None* **default: None**	List of gradio.components to use as outputs. If the function returns no outputs, this should be an empty list.

Parameter	Description
`api_name` *str | None | Literal[False]* **default: None**	defines how the endpoint appears in the API docs. Can be a string, None, or False. If set to a string, the endpoint will be exposed in the API docs with the given name. If None (default), the name of the function will be used as the API endpoint. If False, the endpoint will not be exposed in the API docs and downstream apps (including those that `gr.load` this app) will not be able to use this event.
`scroll_to_output` *bool* **default: False**	If True, will scroll to output component on completion
`show_progress` *Literal[('full', 'minimal', 'hidden')]* **default: "full"**	If True, will show progress animation while pending
`queue` *bool | None* **default: None**	If True, will place the request on the queue, if the queue has been enabled. If False, will not put this event on the queue, even if the queue has been enabled. If None, will use the queue setting of the gradio app.
`batch` *bool* **default: False**	If True, then the function should process a batch of inputs, meaning that it should accept a list of input values for each parameter. The lists should be of equal length (and be up to length `max_batch_size`). The function is then ***required*** to return a tuple of lists (even if there is only 1 output component), with each list in the tuple corresponding to one output component.
`max_batch_size` *int* **default: 4**	Maximum number of inputs to batch together if this is called from the queue (only relevant if batch=True)

Parameter	Description
`preprocess` *bool* **default: True**	If False, will not run preprocessing of component data before running 'fn' (e.g. leaving it as a base64 string if this method is called with the `Image` component).
`postprocess` *bool* **default: True**	If False, will not run postprocessing of component data before returning 'fn' output to the browser.
`cancels` *dict[str, Any] | list[dict[str, Any]] | None* **default: None**	A list of other events to cancel when this listener is triggered. For example, setting cancels=[click_event] will cancel the click_event, where click_event is the return value of another components .click method. Functions that have not yet run (or generators that are iterating) will be cancelled, but functions that are currently running will be allowed to finish.
`every` *float | None* **default: None**	Run this event 'every' number of seconds while the client connection is open. Interpreted in seconds. Queue must be enabled.
`trigger_mode` *Literal[('once', 'multiple', 'always_last')] | None* **default: None**	If "once" (default for all events except `.change()`) would not allow any submissions while an event is pending. If set to "multiple", unlimited submissions are allowed while pending, and "always_last" (default for `.change()` event) would allow a second submission after the pending event is complete.
`js` *str | None* **default: None**	Optional frontend js method to run before running 'fn'. Input arguments for js method are values of 'inputs' and 'outputs', return should be a list of values for output components.

Parameter	Description
`concurrency_limit` *int | None | Literal['default']* **default: "default"**	If set, this is the maximum number of this event that can be running simultaneously. Can be set to None to mean no concurrency_limit (any number of this event can be running simultaneously). Set to "default" to use the default concurrency limit (defined by the `default_concurrency_limit` parameter in `Blocks.queue()`, which itself is 1 by default).
`concurrency_id` *str | None* **default: None**	If set, this is the id of the concurrency group. Events with the same concurrency_id will be limited by the lowest set concurrency_limit.
`show_api` *bool* **default: True**	whether to show this event in the "view API" page of the Gradio app, or in the ".view_api()" method of the Gradio clients. Unlike setting api_name to False, setting show_api to False will still allow downstream apps to use this event. If fn is None, show_api will automatically be set to False.

← ColorPicker

Dataset →