Accessible Jupyter Notebook Output (AJNBO): A VS Code Extension for Screen-Reader-Based Review

Jan H. E. Meyerhoff-Liang[1,2], James Sargent[3], Aziz N. Zeidieh[4], and Yuqi Liang[1,5]

[1] Institute for New Economic Thinking at the Oxford Martin School (INET Oxford), University of Oxford, UK
[2] Blavatnik School of Government, University of Oxford, UK
[3] MyVision Oxfordshire, UK
[4] University of Illinois at Urbana-Champaign, USA
[5] Department of Sociology, University of Oxford, UK

Keywords: Accessibility · Jupyter Notebook · Visual Studio Code · Screen readers · Data science education

1 Motivation and Background

Jupyter Notebook (`.ipynb`) files are widely used in data science, machine learning, artificial intelligence, academic research, and teaching. However, reviewing the output of notebook cells remains difficult for blind and visually impaired (BVI) users. Although notebooks can be executed in environments such as Visual Studio Code (VS Code), common screen-reader workflows do not provide efficient navigation through many cell outputs, especially dataframes and graphs. BVI users may therefore need to export outputs separately as HTML, CSV, Markdown, or text files, or replace notebooks with Python scripts. This creates additional work for users and university accessibility staff, while limiting participation in courses and projects that use notebooks as their default format.

We present Accessible Jupyter Notebook Output (AJNBO), a companion extension for VS Code that adds a screen-reader-accessible interface for reviewing notebook-cell outputs. AJNBO reuses the existing VS Code notebook and Jupyter execution infrastructure. A TypeScript extension observes selected cells and outputs, normalizes MIME-typed results, and exposes accessible text and structured navigation interfaces. An optional Python package generates richer representations of dataframes and charts. We will evaluate AJNBO with example datasets and in a graduate course, *Introduction to Social Data Science*, using VoiceOver on macOS and NVDA on Windows.

2 Design Goals

Notebook interfaces combine executable code, explanatory text, and outputs such as tables, charts, warnings, and model summaries. A BVI user may be able

to execute cells while still being unable to review the resulting output efficiently. AJNBO addresses this gap without requiring instructors or researchers to abandon notebooks. Its core principle is progressive enhancement: plain-text outputs remain easy to read, while structured outputs receive keyboard-navigable representations that preserve their semantic organization.

3 Software Architecture

AJNBO uses a modular architecture with a TypeScript-based VS Code extension and an optional Python helper package. The TypeScript layer listens for changes in the active notebook editor, selected cell, and cell outputs. Each result is normalized internally as plain text, an error, a table, a chart, JSON, an image, or an unsupported type.

Jupyter outputs are commonly stored as MIME bundles: the same result may contain parallel representations such as `text/plain`, `text/html`, `image/png`, or structured JSON. AJNBO inspects the available MIME types and chooses the most useful accessible representation. It prioritizes a dedicated AJNBO structured MIME type, existing JSON output, chart specifications, HTML tables, plain text, and finally an image-only fallback.

The interface has three components. First, a native VS Code Tree View lists notebook cells, execution order, output count, output type, and a short summary. Second, a read-only virtual text document presents summaries, tracebacks, warnings, and text outputs in the familiar VS Code editor. Third, a semantic webview is used selectively for dataframes and chart data. It provides accessible HTML tables or ARIA grids with keyboard navigation, row and column announcements, pagination, and concise summaries.

The optional Python package enriches the output pipeline rather than controlling the editor. Through IPython display hooks such as `_repr_mimebundle_()`, Python objects can emit dedicated JSON-based MIME bundles. A dataframe wrapper can expose column names, data types, dimensions, missing-value counts, and paginated rows. Chart adapters can expose axis labels, series names, and plotted values. The TypeScript extension renders these outputs consistently across operating systems and screen readers.

4 Accessible Workflow and Evaluation

AJNBO will expose commands such as *Review Current Cell Output*, *Review Previous Output*, *Review Next Output*, and *Show Output Summary*. A dataframe summary may announce dimensions, column names, data types, and missing values before the user enters table-navigation mode. For a chart, AJNBO will provide a short description and, where available, an accessible table of the underlying data. Image-only charts will be identified honestly rather than presented as if a reliable semantic description were available.

AJNBO complements VS Code's existing Accessible View but focuses on notebook outputs that require structured exploration. It avoids forking the Microsoft Jupyter extension, reducing maintenance costs and allowing installation alongside the standard notebook environment. We will test representative text, traceback, dataframe, statistical, and chart outputs with VoiceOver and NVDA.