PythonTeX LaTeX Integration

A powerful project that combines the computational capabilities of Python with the typesetting excellence of LaTeX using the PythonTeX module. This integration allows you to embed Python code directly within LaTeX documents, enabling dynamic data calculation, chart generation, and seamless data export to Google Sheets and Excel files.

Features

- **Dynamic Data Processing**: Execute Python code directly within LaTeX documents
- **Chart Generation**: Create and embed charts dynamically using Python plotting libraries
- **Data Export**: Save processed data to Google Sheets and Excel files
- **Real-time Calculations**: Perform calculations on-the-fly during document compilation
- **Module Integration**: Import and use any Python module within your LaTeX documents
- **Code Visibility Control**: Decide whether Python code itself should appear in the PDF or remain hidden

Prerequisites

- Linux-based system (Ubuntu/Debian recommended)
- LaTeX distribution (TeX Live)
- Python 3.x
- Virtual environment support

Installation

Follow these steps to set up your environment:

1. System Updates and Dependencies

```
sudo apt update
sudo apt install texlive
```

2. Python Virtual Environment Setup

```
virtualenv .venv
source .venv/bin/activate
pip install -r requirements.txt
```

Usage

Compiling Documents

To execute the sample projects, navigate to any sample folder and run the following commands in sequence:

```
# Initial LaTeX compilation
pdflatex -interaction=nonstopmode doc.tex

# Execute embedded Python code
pythontex doc.tex

# Final LaTeX compilation with Python results
pdflatex -interaction=nonstopmode doc.tex
```

Python Code Blocks in LaTeX

There are many types of PythonTeX code environments but for this project using `pylabblock` and `pyblock` will be sufficient.
Dor more information about PythonTex see: `https://github.com/gpoore/pythontex`

1. `pylabblock` (Visible in PDF)

- Code written inside a `pylabblock` is printed in the final PDF.
- Both the **Python code** and the **output** appear in the document.
- Useful for tutorials, reproducible research, or teaching material.

Example:

```
\begin{pylabblock}
import numpy as np
x = np.arange(5)
x.sum()
\end{pylabblock}
```

Result in PDF → Shows the Python code and its computed sum.

---

2. `pyblock` (Hidden Code, Only Output Shown)

- Code inside a `pyblock` runs **in the background**.
- Only the **results** are displayed in the final PDF.
- Useful when you want clean reports without showing the underlying code.

Example:

```
\begin{pyblock}
import numpy as np
x = np.arange(5)
x.sum()
\end{pyblock}
```

Result in PDF → Only shows the number `10` (not the code).

Google Sheets Integration

For Google Sheets functionality (available in import_formula_to_google_sheet_and_excel):

Option 1: Use Your Own Service Account

1. Create a Google Cloud Project
2. Enable Google Sheets API
3. Create a service account and download the JSON key
4. Rename credential to `service_account.json` and place it in import_formula_to_google_sheet_and_excel folder
5. Replace `SPREADSHEET_ID` with your sheet id
6. In your sheet give access to your email. Note that making sheet public is not enough and direct access to sheet have to be given to associated email.
7. execute pdf generation commands.

Option 2: Share with Existing Service Account

1. Ask me for credential.
2. place it in import_formula_to_google_sheet_and_excel folder.
3. Share your Google Sheet with the following email address:

```
latex-python@latex-python.iam.gserviceaccount.com
```

1. Change `SPREADSHEET_ID` with your sheet id
2. execute pdf generation commands.

Option 3: Use Existing Sheet

1. Ask me for credential.
2. place it in import_formula_to_google_sheet_and_excel folder.
3. See the results in: `https://docs.google.com/spreadsheets/d/1kSz21B5faifZ6JYvH_2wdvosoQVhP1f1XnYEoo9olzk/`
4. execute pdf generation commands.

Formula Analysis

This sample performs lexical analysis on Excel spreadsheets to identify and group formulas with similar patterns. It reads formulas from an Excel file, identifies their structure, and consolidates them into generalized pattern representations.

How It Works

The analyzer processes formulas in three main steps: - Extraction: Reads all formulas from formula.xlsx - Pattern Recognition: Identifies structural similarities between formulas - Grouping: Consolidates formulas into pattern notation

Example

- Input Formulas: `A1 * B4 , A2 * B5 , A3 * B6`
- Output Pattern: `A<1-3> * B<4-6>`

Pattern Notation

- Continuous Range: indicates sequential values. Example: A<1-3> from A1 to A3
- Discrete Values: <val1,val2,...> indicates non-sequential values Example: B<2,5> represents B2 and B5

Project Structure

```
├── sample1/                      # Basic PythonTeX example
│   └── doc.tex                   # Sample LaTeX document
├── import_formula_to_google_sheet_and_excel/
# Google Sheets integration
│   ├── doc.tex                   # Sample with Google Sheets
│   └── service_account.json      # Google API credentials
├── extract_formula_from_excel
│   ├── formula.xlsx
│   └── lexical_analysis_excel_formula.json
├── requirements.txt              # Python dependencies
├── proj.ipynb                    # A demonstration of openpyxl
abilities
└── README.md                     # This file
```

Workflow

1. **Write**: Create LaTeX documents with embedded Python code blocks
   - Use pylabblock for visible code + output
   - Use pyblock for hidden code, output only

2. **Process**: Run the compilation sequence to execute Python code
3. **Generate**: Python calculations and charts are automatically integrated
4. **Export**: Data is saved to Google Sheets/Excel as configured
5. **Output**: Final PDF with dynamic content and updated data