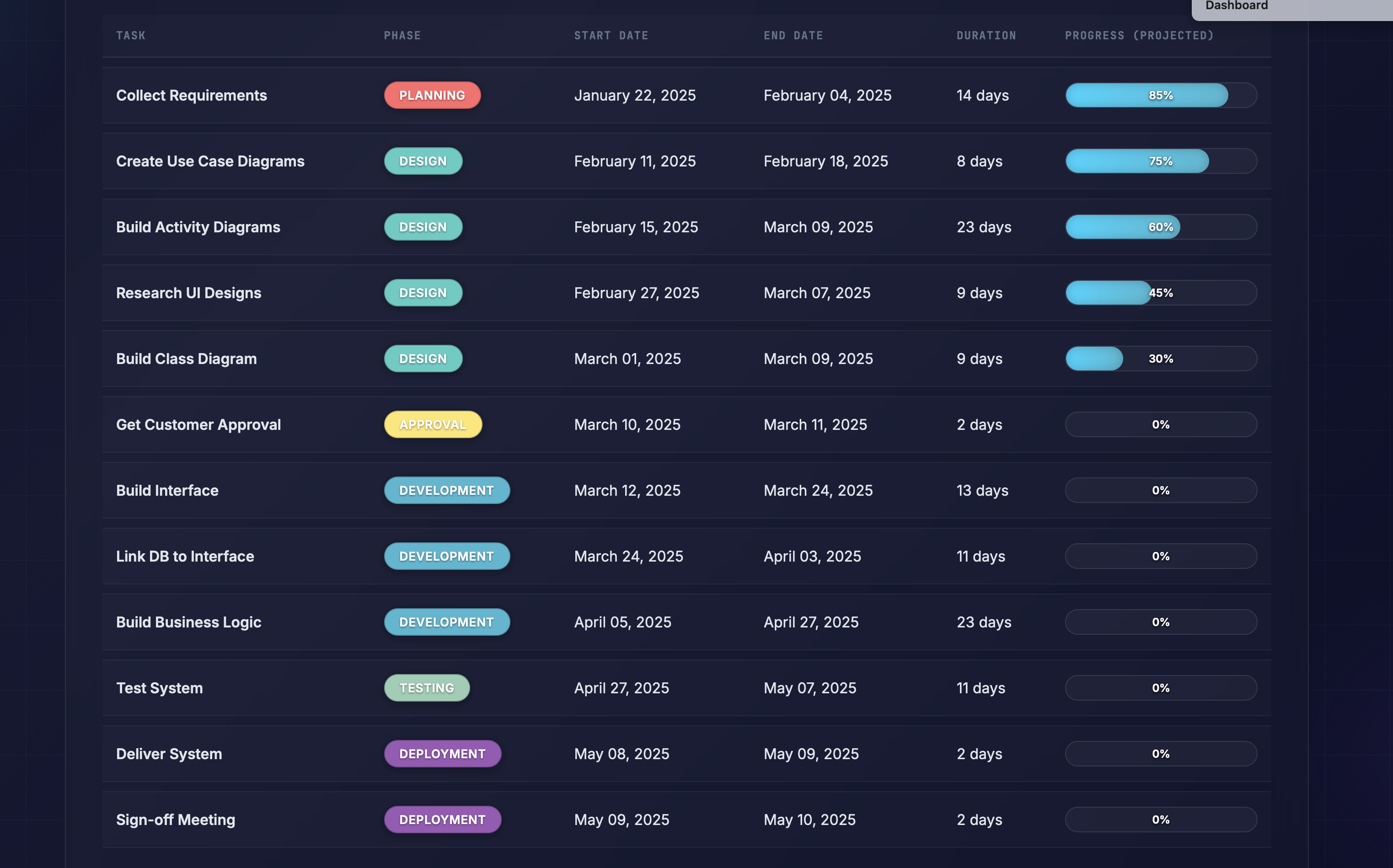
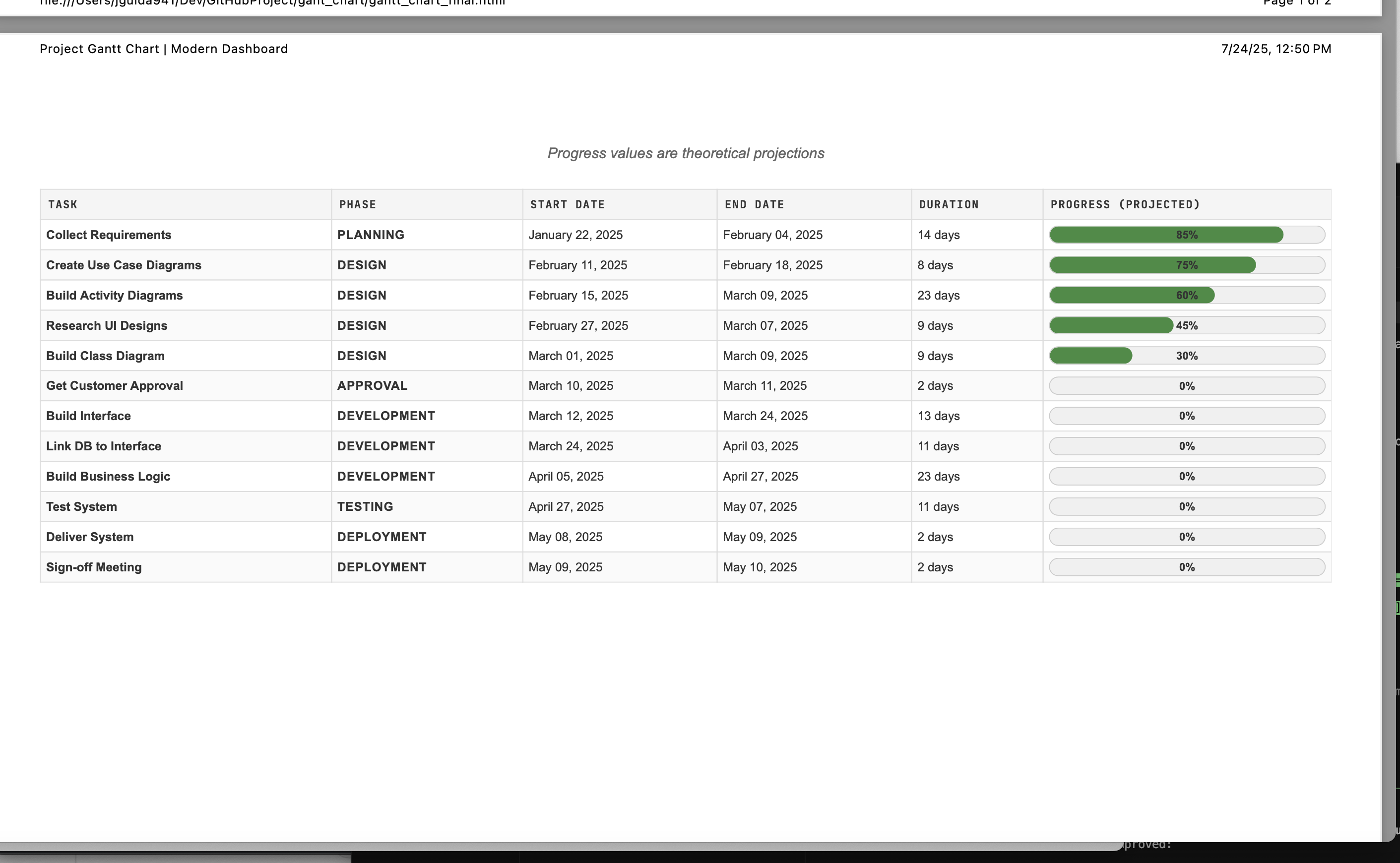
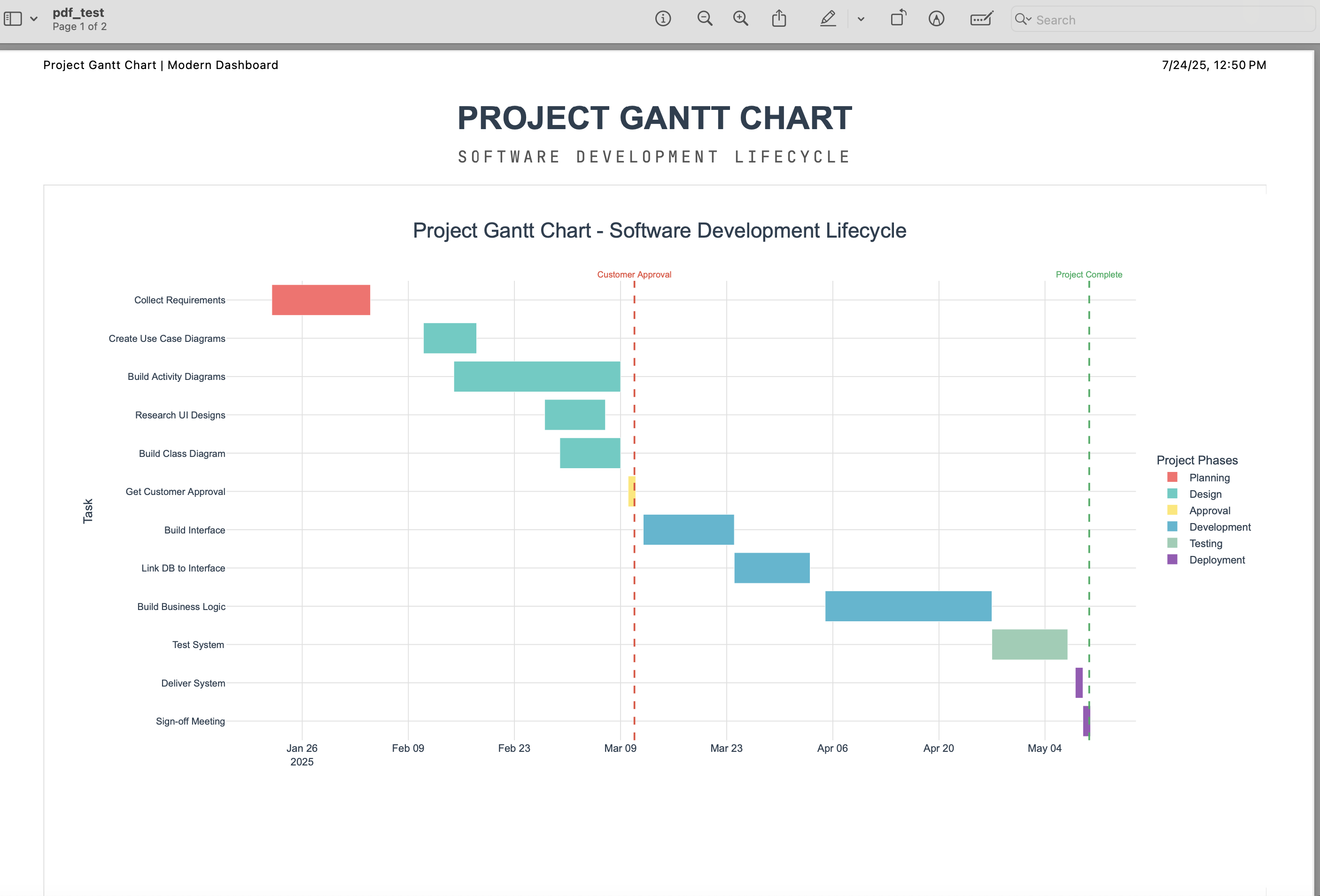
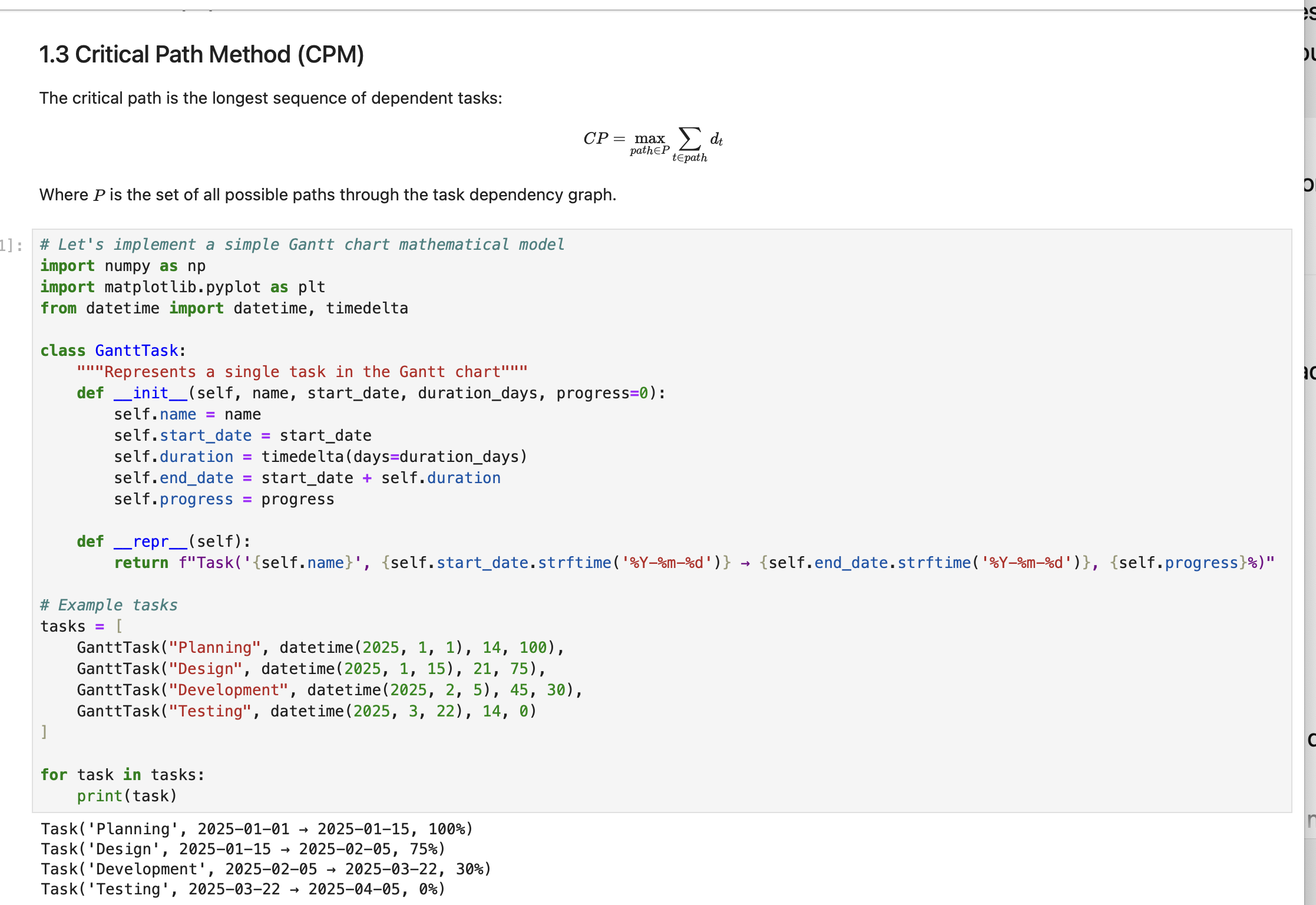
I opted to generate the Gantt chart programmatically using Python and Plotly, which allowed me to: (lucid chart repeatedly tells me i'm out of shape etc and wants me to pay money). This version is better and allows me to:

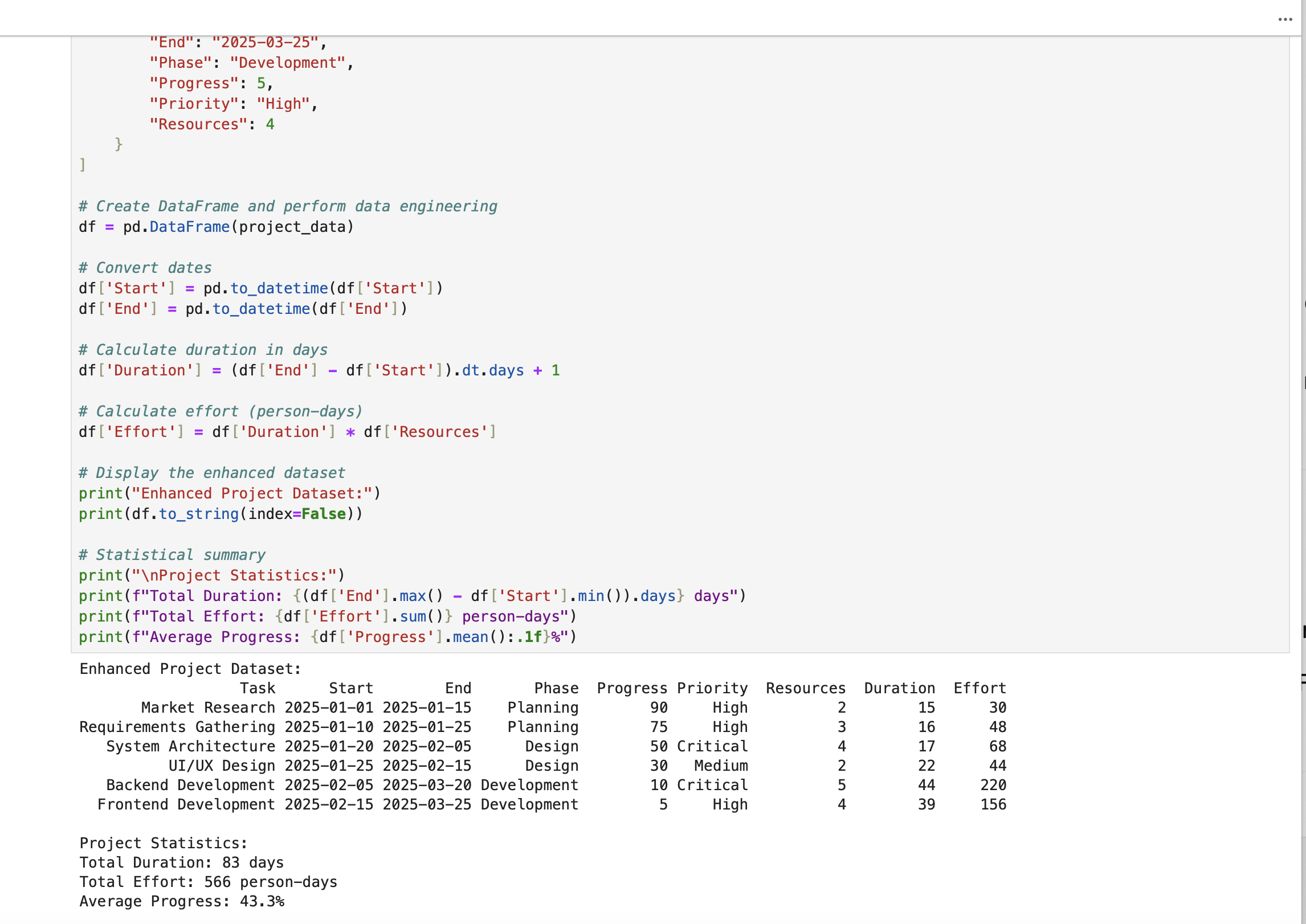
* Compute durations, effort, and progress mathematically
* Render a dynamic, interactive, and exportable chart
* Apply data visualization principles and responsive design
* Maintain full control over logic, structure, and customization

**HTML output of an interactive Gantt chart generated via Python and Plotly:**

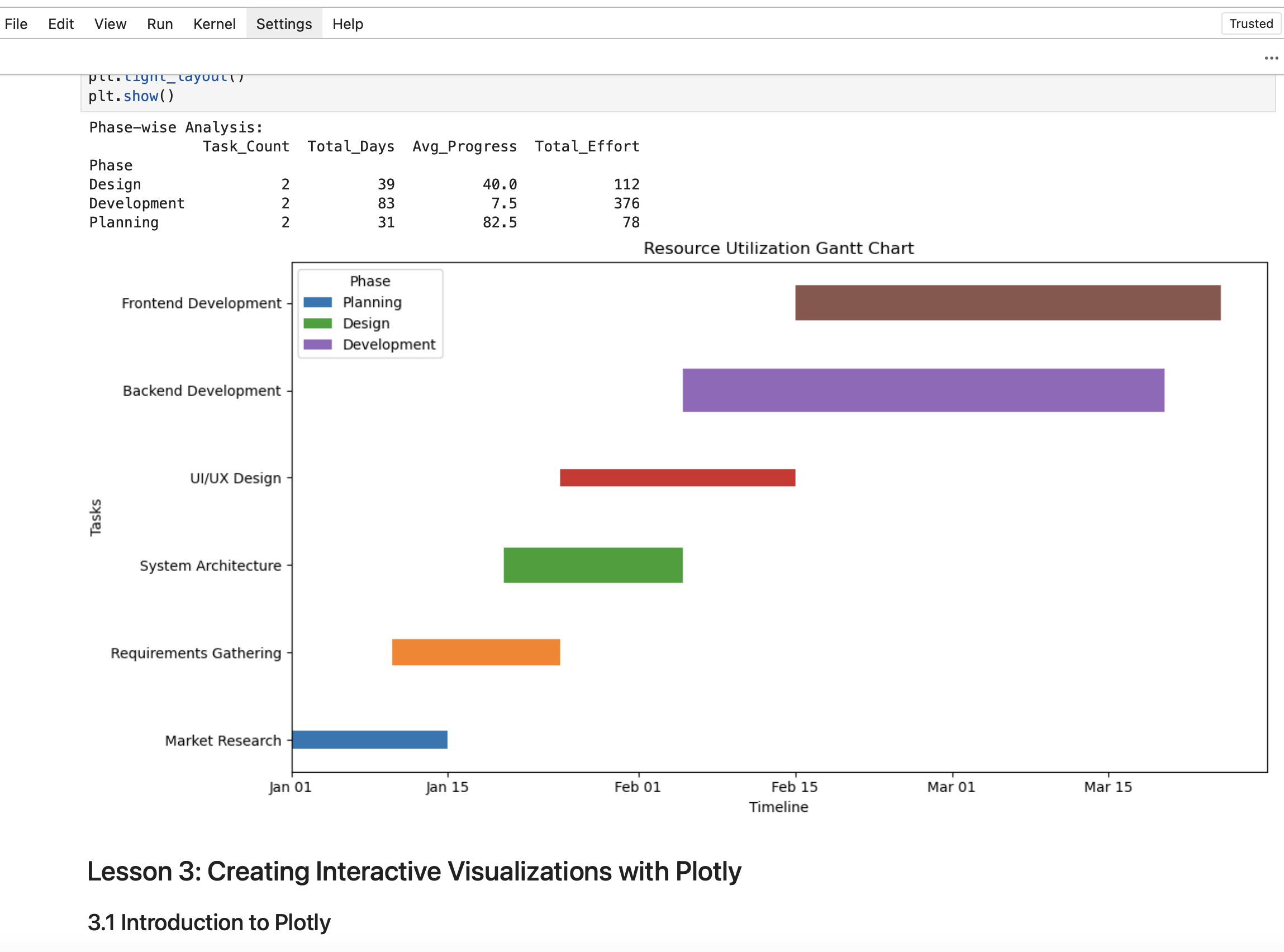


**Export to** **PDF:**

**I also built a Jupyter Notebook to** **teach anyone how to do this:**



**It’s a complete walkthrough  from math theory to data modeling to visualization to deployment.**

**If you****’re interested in how to build Gantt charts like a dev,  DM me or check the repo. Happy to share.**

**1.  Why is object modeling important for system analysis and design?:**

It maps real-world things directly into code, for me it’s about visualizing structure like spotting reusable patterns, reducing duplication, and understanding the full system flow before writing a single line of logic.

 I rely heavily on flowcharts, detailed notes, and animated UML diagrams (built in PyQt6) to plan and retain system architecture. That’s how I was able to take raw task data, generate the chart programmatically in Python, render it in HTML, export to PDF, and build a Jupyter Notebook around it. The planning and modeling up front gave me a reusable, teachable template. This is exactly how I architect my software:  systems first -> then code.

**2  How are the object-oriented concepts of encapsulation and polymorphism related to object models?**

Encapsulation: keeps each part of the system clean and focused which is classic OOP and good practice. it’s like wrapping "every chunk of logic in its own brain" so I don’t have to think about internals constantly. I use it to isolate task logic, chart rendering, export handling: all separate.

 Polymorphism: lets me plug in different behaviors (like exporting to HTML, PNG, PDF) without rewriting anything. I design my objects to behave differently based on context, which keeps the whole system flexible, especially when I’m automating something like Gantt chart generation.

**3.  What are the differences between an object model and a process model?**

An object model shows what the system is: relating to its classes, data, relationships.

A process model shows how the system worksL this comes down to sequences, flows, operations.

In my project, the object model handled everything: Task, Phase, GanttChartGenerator etc.

The process model came in when I planned the pipeline: data in -> cleaned -> visualized -> exported. I use both side-by-side. If I don’t have both, I end up making unnecessary mistakes.

**4. What benefits do you see in using a Gantt chart for scheduling purposes?**

It makes time become visual. When you see overlaps, critical paths, delays everything the entire picture of the product comes into view.. For dev projects, it shows when things "start slipping or stack on top of each other" because some phases can become harder then others, especially if you just came out of a crucial or tougher phase. More importantly, it’s a communication tool, in the sense it's way easier to show someone a chart than explain dates etc they can just look at the chart and get it.. My system also lets me track phase progress and export clean PDFs for reports or meetings etc.

**5. What challenges did you experience when creating your Gantt chart?:**

Honestly, generating the Gantt chart logic was the easy partL  I’m a Python developer, so setting up the data model, computing durations, and visualizing it in Plotly was easy.

The real challenge was getting the final export (PDF/print view) to look clean.

Problem I ran into and fix if anyone ever ends up building something like this:

 Problem: Plotly’s HTML -> PDF export is a little messy by default I found out.

I found out that Plotly includes gray rect backgrounds like #EBF0F8, #C8D4E3, etc. These override your theme when printed. Even the Margins, fonts, and even Plotly UI elements (range sliders, legends) break in print mode, at least it did for me when trying to export.  My

**Fix: Wrote a full @media print override system**

I patched this with raw CSS like:

@media print {

  .plotly .bg, .js-plotly-plot .plotbg {

    fill: white !important;

  }

  rect[fill="#EBF0F8"], rect[fill="#C8D4E3"] {

    fill: white !important;

  }

  body::before, .glow, .download-section {

    display: none !important;

  }

  .progress-bar {

    background: #f0f0f0 !important;

  }

  .task-details-page {

    page-break-before: always;

  }

}

**Why this matters to me:**

Most developers stop at “plot.show()”  but I wanted to export a professionally styled, print-optimized, interactive dashboard that works offline and looks consistent in PDF, and I'm simply tired of lucid chart lol.

**Summary:**

 I reverse-engineered Plotly’s internal rendering quirks, cleaned up SVG layer bugs, and injected a CSS system that supports clean PDF export with precise layout control.

**Now, this cool system:**

* Renders perfectly to HTML.
* Looks professional in the browser,
* Exports perfectly to PDF with no grey box issues, and supports reuse across future Gantt charts or dashboards.

Pretty cool right?