Toward Full Stack DS, ML, and AI

> Thom Ives, PhD

Why?

1. To Improve The Chances Of Your AI Tools Entering Into Production
2. To Better MARKET Your Work - To Improve The Sales Of Your Work
3. To Make Yourself More Marketable
4. To Grow To A Point Of:
    A. BEcoming A VERY Capable DS, ML, and / or AI Consultant
    B. Creating Your Own AI Products

What You Will Need

1. A Desire To Constantly Learn
    A. Must Maintain Your Ongoing DS, ML, and AI Learning
    B. Must Add New Knowledge Of:
        a. How To Create Web App UIs
        b. How To Create Web App Backends
        c. Advance Your Full Stack Knowledge Of Tools And Approaches
2. A Willingness To Continually Grow (Put New Knowledge Into Practice)
    A. Must Pick Projects To Build AND Showcase
    B. Must Constantly Question:
        a. How To Do Better
        b. How To Remove Technical Debt
        c. If You Need To Unlearn / Relearn
3. Must Recognize And Work On Weaknesses
4. Must Exploit Strengths AND Learn How To Grow Them

Example Stories

Story One: A Strength Athlete - An Important Analogy

A young friend of mine has been strength training now for about 4 years. He started out as a skinny kid that has just recovered from food poisoning. He decided he wanted to get stronger and healthier. He started researching the best way to strength train and eat. He's consistently trained for 5 years now under very good coaching. His diet is great, and he takes time to rest and recover between workouts. He is now a professional coach himself helping others on the same journey.

Why does this matter to people in our field?

When he is working out, that is like us working on our personal and work projects. When he is eating (taking in food), that is like us learning new things, and reviewing old learning, regulary. When he is resting, that is like us taking breaks from our learning and growing so that we don't burn out and can continue to grow. When he is reviewing his techniques and workout plans with his coach, that is like us finding our weaknesses and working on them, and like us learning to exploit our strengths.

I encourage you find other powerful analogies for your learning and growing in life and career.

Story Two: The Poor Man's Super Computer

In my opinion, our close cousins are engineers and physicists that do predictive modeling of system prototype designs. A friend of mine started working on very advanced product design about 25 years ago. This engineer was creating all sorts of predictive models for a proposed system design to help guide the prototyping and testing of the system. The program leads considered using a charged particle optics system for one of the subcomponents. Our friend in this system recognized right away that simulation code for this sub-system would take a LONG TIME to run each test case while seeking an optimal solution. Therefore, our friend created a system that controlled 5 different VERY powerful (circa 2000) computers within the company across different sites in different cities. Each solution from each machine provided a piece of information to create a multi-variate derivative prediction for how to change each of the sub-system's parameters to seek the best performance.

After this "poor man's super computer" was running successfully, our friend found that this "virtual" system was unable to achieve the necessary performance. Frustrated by this, our friend went to discuss these findings with an optics expert. After explaining the system, the analyses, and the analytical results, the expert knew right away what the problem was. It was a very basic issue with the charged particle emission source. Absolutely NO charged particle optics design could fix this issue.

Our engineer realized the following:

- Many other highly trained engineers and phycisists were spending an amazing amount of man-hours on a problem that could not be fixed.

- Our engineer had believed that the team and the leaders knew there was a viable method for reaching the needed sub-system performance.
- Had this expert been consulted before all the money and time were spent on this project, a better method for that subsystem could have been researched sooner.
- Having learned this hard lesson, our engineer started a feasibility analysis right away for the next proposed subsystem.
- The next subsystem proposal would cause a major limit to an important performance parameter for the whole system.
- Our friend presented these findings early using both sophisticated analytics AND new and sophisticated presentation and animation tools.
- While only a few people took heed, testing soon proved these findings of performance limitations to be correct.
- As a result, the program was cancelled.

How Our Engineer Advanced Toward A Full Stack Data Scientist:

1. **Learned** how to be extremely resourceful in predictive analytics and **Grew** major new skills in that area
2. **Growth** in methods to present findings in more compelling ways and even created an *company internal website* highlighting designs and analyses
3. **Weakness** in too quickly trusting given directions and learned the hard way to QUESTION and TEST assumptions about the abilities of new advanced system proposals no matter who proposed them
4. **Strength** was gained on this program through growth of our friend's skill set in many ways

In summary, growing both analytical, and presentation and delivery skills, helped our friend grow a tremendous amount in abilities.

Story Three: Insist On A Culture Of Automation

The team that this friend joined had created massive technical debt. However, our data scientist friend, kept being creative and kept finding ways to make advancements in his work in spite of this massive tech debt. He had also decided to start learning how to create software as a service type web applications and had the good fortune to sit right next to good coders who developed such systems for their company.

There was an existing manual process that was taking the team many MANY man hours per month to do. People had no confidence that this important process could be automated. Our data scientist friend did. However, to avoid criticism, he worked on it in the evenings.

He started with small proof of concept type experiments. As they proved promising, he moved them to progressively more extensive experiments. He finally wrapped them into the main code based and began testing there. Then he shared with his manager what he'd been doing.

A final test came where he allowed the process to run over a weekend. The system worked!

How This Analytical Modeler Advanced Toward A Full Stack Data Scientist:

1. **Learned** that just because no one else has confidence something can be done, doesn't mean that it cannot - even the attempts to solve something that can't be done, help us grow
2. **Growth** through taking progressive steps of small experiments to determine if the concepts we was investigating would work or not so as to not waste too much of his personal time if it could not work
3. **Weakness** identified in the way he released news of his findings to the organization even though he did so through his manager
4. **Strength** found in testing and following strong scientific and analytical suspicions

In summary, our analytics modeler friend gained new courage to seek out ever more challenging tasks that would benefit his teams.

Story Four: Tracer Bullets And Usable Parts

This woman was hired to build a very complicated AI system right before the era of ChapGPT and other large language models (LLMs). This friend of mine's project was to automate a very detailed entity identification to automate data entry of tecnical materials. Everytime she needed to implement something, she could get no help from the existing software group, or she was told that their code base could not support that approach. It seemed that existing technical debt was preventing one data science improvement at a time, and there seemed to be no willingness to change. She kept finding ways to keep going, but she felt that the extent to which she could help the team was being reduced one step at a time. She was supposed to have a team, but COVID and cutbacks hit, and she had to do the best that she could on her own. To make matters worse, her hiring manager quit within two months of her start date.

Along the way, as her large application began to take shape, she found someone to help create some front end tools for the application. But then they were told they could not use the approach they were trying to use. Our friend kept going and creating what tools that she could create. AND, due to her ever growing knowledge of full-stack data science, she was always able to find workarounds when she could not get support.

One interim manager was very helpful and sympathetic. It was a software manager, and she even took the time to study how to manage data scientists. This manager liked our friend's commitment to delivering tracer bullets, but her app was so large, the manager could see that it was taking a long time just to even get the tracer bullet working. So, this kind manager gave her a new philophy that really helped. She encouraged our friend to make short term sub-tool deliveries. She had done some of this along the way, but she realized that she could do more.

Somehow, this really helped her motivation and delivery speed. Soon she was delivering more tools and even helping other teams with some of the sub-tools that she had created from her work. Soon she was a popular presenter at the company wide **Lunch and Learn** series.

How This AI System Developer Advanced Toward A Full Stack Data Scientist:

1. **Learned** that it's important to deliver as many benefits along the way when building a large application
2. **Growth** through helping others with the tools that she'd made and with sharing her knowledge with all at company wide presentations
3. **Weakness** identified that she always needed a plan for how to be delivering benefits as often as possible
4. **Strength** gained by continuing to push forward with the large application in spite of not having the support that she needed

In summary, our AI system developer friend learned she could deliver a lot of benefit along the journey of working on a very large application that was taking a long time to build, and she was able to keep the large project going due to growing her skills in full stack data science.

Story Five: Fill The Gaps With Learning And Growing

This AI systems developer landed in a very supportive culture. However, while he really liked the people in IT, the IT policies were, BY FAR, the most restrictive policies he'd ever encountered. Over time, he kept finding better and better ways to go about his development by simply stating his needs. The IT organization started to find more and more ways to help him achieve the goals that they could see that we was making good progress on with his AI system development. Because he was himself committed to becoming a full-stack data scientist he never hit any real show stoppers. This was good, because he also never received any software developer support for the web UI parts of his AI system. However, he also didn't mind, since it gave him the ability to learn more of what he wanted to learn to grow as a full stack data scientist.

He was also able to help others on his DS team, and even more in his greater analytics team at the company, learn some of the many approaches that had helped him advance his project. He was able to impress an executive above him early on by simply showing his Web UI work and explaining how he felt it would help people in their company to work faster and with less error. Then, he'd regularly give updates with demos of both the advances on the AI side of the project and on the Web UI (the full-stack data science) side of the project. These frequent demos and updates also helped him to receive frequent helpful inputs on how to advance the application. When the time was right, due to all the good training he'd taken in over the decades on how to code better, he was able to rapidly plug an advanced large language model (LLM) into his application.

Finally, an internal beta release of his tool happened. People that it was made for seemed to love it at first, but then it was found that they were not using it. This was due to the way it was architected, and it was a known possibility. When feedback was sought from the user base, suspicions were confirmed, and a new direction was taken for the Web UI. Our AI system architect went immediately into the new front end method, and, due to the nature of the existing code base, the change in architecture went fairly smoothly. I am eager to hear how this story progresses for him.

He is also actively working on his own full-stack data scientist application for a domain that he loves very much.

How This AI System Architect Advanced His Current Standing As A Full Stack Data Scientist:

1. **Learned** that learning new things is the most important skill, and that ability to learn new things over time gets better
2. **Growth** is strongest when you have a way to practice what you are learning
3. **Weakness** was found in that more input was not sought early on about certain aspects of the user interface - lesson noted
4. **Strength** found in not saying that developing Web UIs is not our jobs - the Web UI and the backend parts are important integral components to our overall AI systems

In summary, our AI system architect friend found that his growing full-stack data science skills kept his project going forward.

Suggested Things To Learn

> This is NOT a complete list. It's my *suggested* list based on my *suspected* guess that you LOVE Python, and that would therefore love these tools too.

Streamlit

Streamlit is a free open-source Python library that allows users to create and share interactive web applications for machine learning and data science.

- Data scripts can be converted into shareable web apps very quickly.
- It offers a wide range of tools for creating data visualizations.
- It is known for its simplicity and user-friendly nature.

Streamlit

Streamlit apps are highly popular for being easy to build. It is also easy to do continuous development with them. They are compatible with a wide range of tools and technologies. Users say that that they can build clickable prototypes for complex web applications faster and with more flexibility. Streamlit is highly recommended for Python, machine learning, and data science projects.

The library allows users to create engaging, informative, and visually appealing apps with very little code. It also offers a Community Cloud platform for deploying, managing, and sharing apps. Streamlit's simplicity and high-level APIs make it easy for experienced software engineers, and even beginners in data science, to get started.

In summary:

- Powerful and user-friendly open-source Python library
- Simplifies the process of building interactive web applications for ML and DS
- Makes it easier for users to get started and create impactful applications

FastAPI

FastAPI is a modern web framework for building **REST**ful APIs in Python. It was first released in 2018 and has quickly gained popularity among developers due to its ease of use, speed, and robustness. FastAPI is based on Pydantic and uses type hints to validate, serialize, and deserialize data. It also automatically generates OpenAPI documentation for APIs built with it, which also replaces the need for using an API test system like POSTman. FastAPI fully supports asynchronous programming and can run on Gunicorn and ASGI servers such as Uvicorn and Hypercorn, making it a good choice for production environments.

FastAPI is standards-based and fully compatible with the open standards for APIs: OpenAPI (previously known as Swagger) and JSON Schema. It is mainly used for data science and e-commerce applications, enabling developers to use the REST API and a wide range of functions to implement them in applications.

FastAPI supports asynchronous calls, making it lightweight, faster, and easy to learn. It provides high performance, making it suitable for high-traffic APIs where speed is a priority. FastAPI is fully production-ready, with excellent documentation, support, and an easy-to-use interface.

Overall, FastAPI is a powerful and modern framework for building APIs in Python, offering a range of features that make it an excellent choice for developers looking to create high-performance and robust web APIs.

> ***REST*** *stands for Representational State Transfer and is an architectural style for distributed hypermedia systems. It is an application programming interface (API) that conforms to the constraints of REST architectural style and allows for interaction with RESTful web services. When a client request is made via a RESTful API, it transfers a representation of the state of the resource to the requester or endpoint. This information, or representation, is delivered in one of several formats via HTTP, such as JSON, HTML, XML, Python, PHP, or plain text.*

The Basic Web Stack: HTML, CSS, JavaScript

HTML (Hypertext Markup Language): HTML provides the basic structure of websites and web applications. It defines the content and layout of the web page, including headings, paragraphs, images, and links. HTML is the foundation upon which web content is built and is essential for creating the structure of a webpage.

CSS (Cascading Style Sheets): CSS is used to control the presentation, formatting, and layout of the content defined in HTML. It enhances the appearance of the web page by allowing developers to apply styles, such as colors, fonts, and spacing, to HTML elements. CSS ensures that the content is visually appealing and well-organized for the users.

JavaScript: JavaScript is a programming language that adds interactivity and dynamic behavior to web pages. It allows developers to create interactive forms, animations, and real-time updates, making the web page responsive to user actions. JavaScript complements HTML and CSS by providing the means to control the behavior of different elements on the web page, resulting in a seamless and engaging user experience. JavaScript feels much like Python as you learn it and is very flexible and powerful. I've often wondered why the DS, ML, and AI world didn't use JavaScript as its foundation instead of Python and R.

In summary, **HTML** provides the structure, **CSS** enhances the appearance, and **JavaScript** adds interactivity to websites and web applications. These three technologies work together to create a cohesive and user-friendly interface, fulfilling users' needs and expectations for an enjoyable web browsing experience.

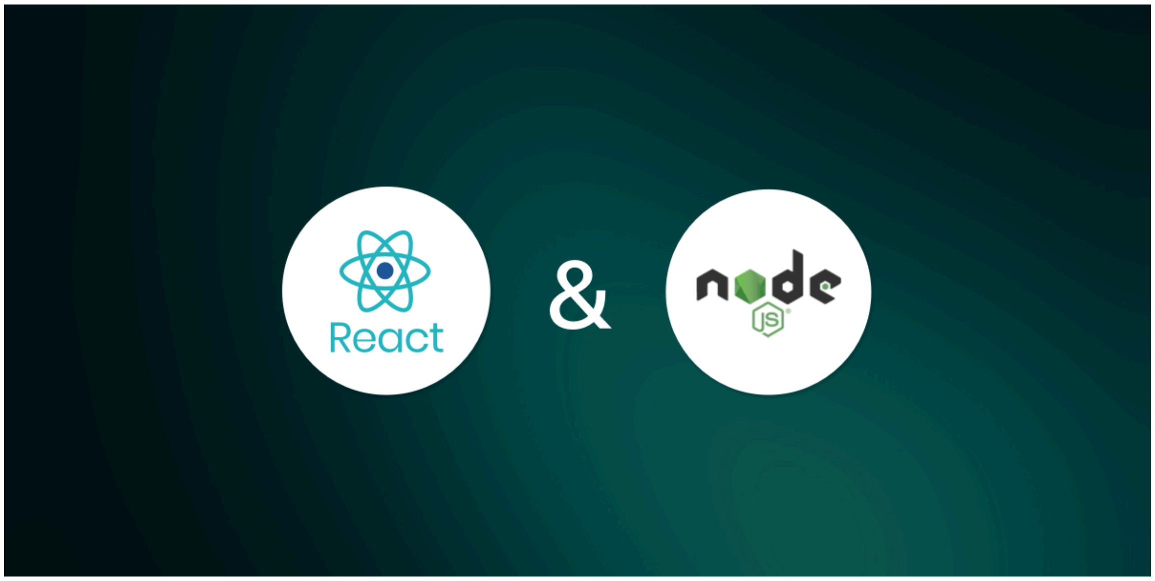
Integrations

At this point, One could:

- Integrate FastAPI into Streamlit apps
- Integrate FastAPI APIs as the backend calls from JavaScript to load or store data and display new or updated data on web pages built with HTML, CSS, and JavaScript

Node.JS and React.JS

Node.js and React.js are both significant technologies in the web development landscape for different reasons.



Node.js:

- Node.js is a server-side JavaScript runtime environment that allows developers to execute JavaScript code outside a web browser.
- It is particularly significant for building back-end services, such as APIs for web and mobile applications.
- Node.js has an asynchronous or non-blocking nature, making it efficient for handling large amounts of traffic and data.
- It provides an easy start for developers and has a large ecosystem of open-source libraries.

React.js:

- React.js is a front-end JavaScript library used for building high-performance user interfaces with a virtual DOM and reusable components.
- It offers features such as code component reusability, virtual DOM for better performance, and support for both Android and iOS platforms.
- React.js is known for its ease of use in creating UI test cases and its ability to connect with Node.js for full-stack web application development.

Combining Node.js and React.js

When it comes to web development, Node.js and React.js are not mutually exclusive and can be used together in a full-stack web application. For example, Node.js can be used to build the back-end system and APIs, while React.js can be used to build the front-end user interface.

Scalability and Framework Integration

- Node.js can be easily scaled to handle large amounts of traffic and data, making it suitable for server-side web applications like online streaming platforms.
- React.js, on the other hand, is ideal for projects with dynamic inputs and buttons, making it a better option for creating front-end user interfaces.

In summary, Node.js and React.js are both powerful and popular technologies that offer different benefits and features for building web applications. They can be used together to create robust and efficient full-stack web applications, leveraging the strengths of both technologies for back-end and front-end development.

Docker: It's Importance In Full-Stack Data Science

Docker plays a crucial role in full stack data science for several reasons:

1. Isolation and Reproducibility:

- Docker isolates the software from all other things on the same system, allowing for the creation of a reproducible and consistent environment for each data science project.
- Applications that run in containers are indifferent to the features of an individual system, promoting smooth reproducibility.

2. Portability and Flexibility:

- Docker allows for the packaging of an application and its dependencies into a single reusable artifact, which can be instantiated reliably in different environments.
- It provides an isolated, reproducible, portable, and flexible workspace for machine learning and data science projects.

3. Development and Production Workflow:

- It enables the development of offline and parallel docker images for running Python scripts for production code, maintaining a lightweight workflow that can be easily managed in a central Git repository.
- Docker facilitates the creation of a containerized stack, including the high-level application, dependencies, and the isolation layer, which is beneficial for data science projects.

4. Collaboration and Sharing:

- Docker allows for the sharing of data science projects by pushing the image to a container registry such as Docker Hub, enabling others to pull the image and run the project in their environment.

In summary, Docker is important in full stack data science as it provides isolation, reproducibility, portability, flexibility, and facilitates collaboration and sharing of data science projects.

Kubernetes

Kubernetes plays a crucial role in full stack data science projects due to its ability to manage and orchestrate complex data science workloads efficiently. Here are some key reasons why Kubernetes is important in full stack data science projects:

1. Scalability and Autoscaling: Kubernetes provides mechanisms for autoscaling, allowing data science projects to scale up and down based on demand, which is essential for handling large-scale data processing and machine learning workloads.

2. Stateful Workloads: In production, data science projects often require mechanisms to persist data and state across dynamically changing instances. Kubernetes provides ways to store and manage sensitive information, such as passwords, OAuth tokens, and SSH keys, without exposing them in the stack configuration.

3. Management of Workloads: Kubernetes efficiently manages the burstiness of data science workloads, ensuring that systems can work with any cloud vendor and across on-premises systems, providing unparalleled management power and portability.

4. Integration with Machine Learning Frameworks: Kubernetes integrates with machine learning frameworks like Kubeflow, enabling the development of machine learning pipelines to a production level. It allows machine learning engineers to run frameworks like JupyterHub, Tensorflow, PyTorch, or Seldon under Kubernetes, facilitating the creation of dedicated machine learning pipelines.

5. Streamlining Development Workflow: Kubernetes provides an unbeatable combination of features for working data scientists, streamlining the software development workflow and supporting a data science workflow.

6. Cloud-Native Applications: Kubernetes is crucial for developing cloud-native applications, enabling developers and data scientists to conquer the challenges of today's data-centric world, from distributed systems to real-time analytics to machine learning pipelines.

In summary, Kubernetes is essential in full stack data science projects for its scalability, stateful workload management, integration with machine learning frameworks, and support for cloud-native applications, making it a valuable tool for data scientists and machine learning engineers.

PyRobot For Continuous Integration And Continuous Development (CI/CD)

PyRobot is an important tool for continuous integration and continuous development in full stack data science projects. Here's why:

1. Integration with Continuous Integration Servers: PyRobot can be integrated with continuous integration servers, such as Jenkins, Go, Buildbot, and Travis CI, to centralize all continuous integration operations and provide a reliable, stable platform for building projects.

2. Seamless Integration into Business Technology Stack: PyRobot, when used in conjunction with Docker, helps in transitioning data science projects from proof-of-

concept to production by seamlessly integrating into the rest of the business technology stack.

3. Importance of Continuous Integration in Data Science: Continuous integration is crucial for data science projects, as it allows for early detection of errors without having to wait until the end of the project. It provides increased visibility, a single repository, and automated build, which are essential for coordinating and working on the project effectively.

4. Core Practice of Continuous Integration: PyRobot supports the core practice of continuous integration, which involves frequently building and testing each change done to the code automatically and as early as possible. This helps in detecting integration errors quickly and ensuring the stability and quality of the codebase.

5. Integration with Machine Learning Operation (MLOps) Principles: Continuous integration and continuous testing are at the core of Machine Learning Operation (MLOps) principles, making PyRobot essential for data scientists and machine learning engineers who are applying DevOps principles to ML workflows.

In summary, PyRobot plays a crucial role in enabling continuous integration and continuous development in full stack data science projects by integrating with continuous integration servers, supporting core CI practices, and seamlessly integrating into the business technology stack.

Summary

We explored five stories about learning, growing, working on weaknesses and exploiting stengths. Each person was looking to advance their abilities. They were able to progress, because they found better ways to advance AND deliver the benefits of their work to others. This is related to what it means to "Work Toward Full-Stack Data Science".

Then we explored some suggested tools that I believe will truly help you to advance your career.

Conclusion

It is OK if you never become a full-stack DS, ML, AI person. However, working TOWARD BEcoming a full-stack DS, ML, AI person will either open more doors for you OR keep those doors opened longer. AND, if you BEcome a true full-stack DS, ML, AI person, you could do well in consulting or as a company founder for DS, ML, and/or AI type products.