EconCS in Industry: Skills to Succeed as an Applied Scientist

NIKHIL R. DEVANUR
Meta
and
RENATO PAES LEME
Google
and
OKKE SCHRIJVERS
Central Applied Science, Meta

In our years as applied scientists and managers at Google, Amazon, and Meta, we have seen both the strengths that EconCS researchers can leverage in industry, as well as common challenges that these researchers face. Most EconCS PhD programs do not emphasize exploratory data analysis, applied machine learning and statistics, or a coding mindset, even though these are valuable skills to have in industry. In this article we share how these skills are leveraged, and how you can invest in building these skills now. In doing so, we hope to make it easier for people from the EconCS community to be successful in industry, be it during an internship, a sabbatical, as a part-time consultant, or as a full time applied scientist!

1. INTRODUCTION

EconCS is Multidisciplinary

While the EconCS field has its roots in Theoretical Computer Science, since its very inception, the field has spanned academic disciplines. When Nisan and Ronen published "Algorithmic Mechanism Design" in 1999, they connected the fields of economics and computer science together by focusing on possible strategic input to algorithms. Over the last 25 years the field has also seen significant cross-pollination between academia and industry: ad auctions, autobidders, blockchains and now foundation models have spurred significant theoretic work, and conversely these new theoretical results have shaped the product offerings that tech companies have developed. Applied scientists at tech companies operate at the intersection of this and facilitate a two-way street between academia and industry. They have a deep understanding of the product problems that exist and help formalize and popularize such problems in academia, and they leverage the latest developments in academia to drive impact on the companies' products.

Industry is rewarding

There is a large overlap between the types of problems academics and applied scientists work on, but there are important differences too. Applied scientists spend a good amount of time working directly with product teams to understand the in-

Authors' addresses: ndevanur@meta.com, renatoppl@google.com, okke@meta.com

tricacies of the problem space, which helps in formulating theoretical models that capture the most important factors while abstracting away from peripheral concerns. When subsequently developing algorithms or methods for these models, it's key to test this out in the actual production system to confirm that the assumptions and abstractions were indeed justified, and to make sure that the new change actually has a meaningful impact on users. Eventually, the algorithms they work on may impact millions or even billions of end users!

EconCS skills do well in industry

EconCS training provides applied scientists a unique perspective of seeing production systems from the perspective of incentives that act on participants. One classic example is ad auctions, where every modification to the auction changes the behavior of both advertisers bidding in such a system as well as the users seeing those ads. Advertisers may lower bids when presented with higher prices and users may click less when presented with lower quality ads. The same applies in many other settings: a change in the content ranking algorithm of a user's feed, leads to creators producing different content. And given an algorithmic change that decides who gets priority for scheduling jobs in a shared computing platform, users will find ways to improve their own chances of getting allocated by gaming the system.

It is often the case that a new algorithm performs very well when we simulate it under current user behavior but when we deploy, the user behavior changes and we end up in a worse place than we were originally. The EconCS perspective of reasoning about equilibrium and incentives can help identify ways in which real world systems can be gamed and mechanism design, information design and social choice can provide methods for making such systems more robust to manipulations.

What's the problem?

While applied science in industry is exciting and EconCS researchers have advantages in utilizing their skillset in industry, the three authors have found that there are also challenges to making the transition, and in our years have seen those challenges come up for others as well. Since most EconCS researchers have a background in theoretical computer science, many are less familiar working with real data, which makes it significantly more difficult to develop the product understanding that's necessary to develop good theoretical models and algorithms for them. Additionally, almost all roles in tech companies require good coding and ML engineering skills. Even when working with other software engineers (SWEs) and machine learning engineers (MLEs), there's a real benefit in being able to develop prototypes and initial ML models yourself to prove out the methods that you're proposing. It's possible to be a highly successful EconCS researcher in academia without picking up these skills, but learning these skills are key to making a smoother transition to applied science in a tech company.

How will this article help solve the problem?

In this article we discuss the areas that we have found to be most beneficial that a typical EconCS PhD program may not include in their curriculum: exploratory data analysis, machine learning and statistics, and a coding mindset. For each of these areas, we share how these areas show up in the work, what recommended resources are to build the skills, and actionable advice on how you can build the skills. In writing this article, we hope to give more insight into what working in industry is like, and give people all the resources to prepare for an industry research career. Industry research can be a really rewarding career; hopefully this article will help smooth the transition for those who are excited to pursue it!

2. EXPLORATORY DATA ANALYSIS

The problem with real-world problems is that they tend to be messy. It's all well and good to design an auction algorithm that works well for Myerson-regular distributions (the authors of this article have all done this), but how do you know if this is a reasonable assumption when you're dealing with real-world input? A key tool in an applied scientists toolbox is the ability to grab real data and work with it. One of the most useful basic skills in this domain is Exploratory Data Analysis, or EDA.

Exploratory Data Analysis is the iterative process of learning properties of the data that you're working with. Typically you start with a question or hypothesis (such as: the bids in an auction come from a Myerson-regular distribution), you then summarize, visualize or model your data, and finally you use what you learn to ask new questions about the data. For example, you may find that the aggregate distribution of bids is bimodal (which isn't Myerson-regular), prompting the question if there are two different populations in the dataset, each corresponding to one of the peaks. In this process you'll likely find that your data needs cleaning or transformations. For example, maybe there was a production error that caused bids to be logged as NULL.

While it may be tempting to defer to data scientists to conduct EDA, they may lack the domain knowledge to know the right questions to ask, and what are the most important take-aways from a visualization. Additionally, every additional dependency means that you are waiting for someone else's work queue to clear, leading to slower execution. By versing yourself in EDA, you'll be able to move fast, but what should you learn, and how do you get started?

Current Tools

First, it's useful to be aware of the tools that are used in industry. While these tools change over time (that's a caveat that applies to most of the things we share here in this article), they represent a good place to build the fundamental understanding and skills.

Data in industry is commonly stored in databases, and most companies will use some variant of SQL to access the data. While there are some that can work magic in SQL, generally speaking it's sufficient to know basic commands, as most of the visualization, analysis and transformations are easier done after pulling the data.

To analyze the data, there are two languages that are generally used: Python and R. While Python is generally more common, people with a background in statistics may be more familiar with R. If you're only learning one, Python is the way to go.

The Python data science toolkit is spread out over different packages: Pandas [Wes McKinney 2010] is used to represent the data and perform operations on it. It's built on NumPy [Harris et al. 2020] and in some cases, familiarity with NumPy can be helpful in data transformations. To visualize data, seaborn [Waskom 2021] is the easiest way to get quick results with minimal boilerplate code, but matplotlib [Hunter 2007] can be used for more freedom. It's typically useful to be in an interactive environment when doing EDA, for example by using Jupyter notebooks (previously called iPython). To learn more about these tools, and data analysis in Python more generally, check out the free books Python for Data Analysis [McKinney 2022] and the Python Data Science Handbook [VanderPlas 2016].

R is typically preferred by statisticians, and if you are only learning one language for data science, it should probably be Python (since it is easier to combine with general-purpose or ML code and it's more common in industry). However, because it is a more domain-specific language, R has some benefits (such as a streamlined syntax for common data processing steps). The best way to use R is to use the RStudio environment, along with the tidyverse [Wickham et al. 2019] packages. The tidyverse packages are based on the philosophy of having Tidy Data [Wickham 2014], and make it particularly easy to get the data in that format. Even if you don't plan on using R, reading the Tidy Data paper will be useful! If you want to learn R, one of the best resources is R for Data Science [Wickham et al. 2017].

Putting it to Practice

The books we referenced above have many exercises and sample datasets, and it's useful to go through them as you're making your way through the book. However, in our experience, the best way to fully develop these skills is by 1) using them to solve problems that you care about, and 2) using them consistently as part of your larger work. Different strategies may work for different people, but some suggestions that may help in achieving this are: doing an industry internship, participating in data science competitions, and including empirical sections in (some of) your papers.

A common recommendation to build these skills (be it EDA, ML, statistics, or coding) is to do an industry internship. The benefit here is that you work on a real problem, so you're building skills as a means to an end. This makes the importance of building the skills more salient and helps differentiate the parts that you'll use all the time vs the parts that are less common. We won't repeat the same recommendation in the following sections, but note that the recommendation applies there too.

For data science competitions, Kaggle is the most common one. In the majority of competitions, EDA is only the first step of the process, with a more heavy emphasis on building ML models on the data afterwards, but this can still be a great way to get hands-on experience (plus: experience with ML is also quite useful)! In addition to competitions, the site also hosts datasets (along with community code and discussions) and models.

Many theory papers don't have empirical sections or just a rudimentary one that doesn't add much insight beyond the theoretic results (we are definitely guilty of this). But that can be a missed opportunity! An empirical section can demonstrate that an algorithm can perform much better than worst-case bounds, or provide strong evidence that the conditions under which theorems are proved are reasonable. While it can be difficult to get access to data outside of interning/working in industry (and even then it can be hard to publish those datasets), there are a number of publicly available datasets that can be useful for this: Criteo has a number of datasets for online advertising, there's also a NeurIPS competition dataset [Su et al. 2024] for autobidding settings, and the Movielens dataset [Harper and Konstan 2015] can be used for general valuations. This list is not exhaustive (and biased towards our own experience); see if you can find datasets that are appropriate for your papers!

3. MACHINE LEARNING AND STATISTICS

While exploratory data analysis is all about interactively learning from data with a human in the loop, machine learning and statistics are used extensively across tech to develop algorithms automatically from past data and make decisions about which new features to launch.

Machine Learning

Machine learning is so crucial to all the tech companies that you will definitely need to work with machine learning models as part of the overall system. For example, in ad auctions a very important input into the auction is the probability of a click. We typically assume that we know the true probability, but in practice, this is the output of a machine learning model. If the model is not perfect—which it never is—that may affect the outcomes of your design. You need to be aware of the limitations of machine learning, and be aware of concepts such as overfitting and calibration.

Another common paradigm in EconCS is dealing with uncertainty, such as in the study of prophet inequalities. For such problems we either assume that we already know the distributions exactly, or that we have i.i.d. samples from an unknown distribution. In the latter case, we assume that each instance is independent, and we learn only from samples for that instance. In practice, often there are many parallel instances of the same problem and the data for all the instances are correlated. For example, in ad auctions, you can consider the auction for each keyword as an independent instance, but advertiser values for similar keywords are correlated. For instance, advertiser values for a keyword "green sweater" could go up because it is getting colder and all "sweater" related keywords are trending up, or because it is nearing St. Patrick's day and keywords for all green colored apparel are trending up. By using samples from all the keywords together, an ML model can learn such patterns.

You may need to prototype some simple ML models. For this, you need to know how to train a model using standard libraries. You need to know what features to collect, what model architecture to use, what is the loss function, and what are typical sanity checks to run, such as normalizing the inputs. To build these skills, there are several good online courses and Python packages that may be helpful.

For many years, Andrew Ng's Coursera course has been recommended as a great starting point to learn machine learning, and for good reason! The course gives hands-on experience building ML models in Python using Numpy and scikit-learn [Pedregosa et al. 2011]. The latter (also known as sklearn) is a common ML package used for smaller scale ML training. While you won't be building production-scale models with sklearn, it's an excellent way to build sufficient familiarity with building ML models. For people that prefer learning through reading, Andrew Ng's CS229 lecture notes are also excellent. For those seeking the thrill of a competition to motivate themselves to build better models, Kaggle has a whole range of ML competitions that you can participate in.

A word on LLMs

Large Language Model technology is very exciting and rapidly evolving. For those wondering if they need to know all the ins and outs of LLM training if they're in industry, rest assured that this is not the case. However, it is useful to know how to use GenAI products, particularly for coding, which is one of the biggest use cases of these models within tech companies. In addition, we recommend thinking of creative applications of large language models in new and different areas. These tools are, for now at least, not a replacement for building coding, EDA, or ML skills, as you'll need to not only write code, but also vouch for it's correctness.

Statistics

Many important insights in practice come from understanding the behavior of the participants in your design, such as the users or the advertisers. These are typically observed and validated using large scale experiments, a.k.a. A/B tests. Often the experiments involve two sides of a marketplace, and experiment design for such marketplaces has been one of the areas where folks from EconCS have made some very interesting contributions. Most EconCS researchers are well versed in probability theory, but there are certain statistical concepts that are crucial to understand in order to analyze these experiments. These are easy to pick up, such as p-value, t-test, power analysis, winsorization, or minimum detectable effect.

There are several excellent resources to learn about how A/B tests are run in tech companies. "Trustworthy Online Controlled Experiments" [Kohavi et al. 2020] provides a recent overview of the main considerations that go into A/B testing at tech companies, written by authors who have developed these systems at Google, LinkedIn and Microsoft. "The Econometrics of Randomized Experiments" [Athey and Imbens 2017] is a more technical survey of the analysis of randomized experiments. There are situations where it is infeasible to conduct perfectly randomized experiments. "Mostly Harmless Econometrics" [Angrist and Pischke 2009] is a great guide for the most-used tools for such settings. There are no surveys specific to experimentation in markets, though interested readers can look up "budget-split experiments", "bipartite experiments" or "experimentation under interference".

4. CODING MINDSET

Coding is an essential part of being an applied scientist and can be seen as the ability to translate the algorithms we design to practice. When implementing an algorithm, one can't ignore the messiness of the real world – corner cases, data that is not properly cleaned-up or is missing, small probability events that may cause a server to crash, et cetera. Coding also forces us to confront performance issues in a very serious way: constants in the running time of an algorithm may not make a difference in a research paper but can be a deal-breaker in practice.

Knowledge of certain fundamental languages like C++ (for high performance code) or Python (for a broad swath of applications) is certainly important but specific technologies/libraries are constantly changing. So instead of focusing on specific technologies we suggest developing three things: a coding mindset, strong software engineering skills, and understanding distributed systems.

Coding Mindset

A coding mindset refers to going one step beyond algorithmic thinking, in actually seeing the algorithm through in action on real data and reasoning about which repetitive processes can be automated. Software engineering skills have to do with writing code that is tested, easy to read, maintainable by others and follows a consistent and predictable style. It is the difference between writing programs for yourself and writing code that will be later read and modified by hundreds of other engineers over a long period of time.

Software Engineering

While there are books and courses devoted to software engineering, we think there is no better way to learn than doing it in practice, by either doing an internship where you will write actual production code or contributing to open source projects where you will develop software alongside others. It is a great idea to sharpen one's coding skills using interview-preparation websites (e.g. LeetCode or HackerRank) or coding competitions (e.g. ICPC). Those help build familiarity with languages and speed in thinking about coding solutions – which can be very helpful when building rapid prototypes.

It is important to note that coding does not refer only to writing code. A very important skill is to be able to read code effectively. Documentation in industry is frequently missing, incomplete, or outdated, so the only source of truth about the behavior of a system is the code itself. An effective applied scientist should be able to interrogate the codebase to understand the behavior of a system. When we inspect the code more closely we soon find out that the common wisdom about how the system behaves may not be entirely accurate.

A second reason to learn how to read code effectively is to be able to do code reviews, i.e. verifying and vouching for code written by others. Applied researchers typically design systems that are then implemented by other engineers in the organization. It is important to be able to verify that there are no gaps between the system designed and the system actually implemented.

Finally, when we practice reading code, we become better at writing code that is readable by others. Clear and readable code has useful comments, properly-named variables and follows a well-defined style–usually according to each company's guidelines. Well written code also contains plenty of unit-tests which verify its correctness and provide code-readers with examples on how each part of the code is used.

Distributed Systems

Most tech companies operate at a tremendous scale, serving millions and billions of users at subsecond latencies. This is enabled by complex distributed systems with many interlocking pieces. Any change that you would want to incorporate has to be a part of this overall system, so it is important to understand how these systems work, and what are the strengths and limitations of such systems. We recommend becoming familiar with the design of systems such as how a news feed is designed, or how an information retrieval system is designed. The other aspect of this is that the data that you will be working with is also large scale and typically cannot fit in the memory of a single computer. Data analysis requires working with distributed big data systems, so once again, it's good to be familiar with how to work with such systems. There are several textbooks on this, such as "Designing Data-Intensive Applications" [Kleppmann 2019] or even "System Design Interview" [Xu 2020] that can be used to learn more.

5. CONCLUSION

In this article we've aimed to give actionable recommendations for EconCS researchers that are interested in building skills for applied science in industry. A reader may go over this article and get the impression that they're woefully unprepared for a role in industry unless they do all of this! That's not the case. Many applied scientists, to a certain extent ourselves included, pick up these skills when first preparing for interviews, during internships, or on the job. However, part of the reason that so many only learn these skills at that stage, is because we simply didn't know which skills were important or how to practice them beforehand! In writing this article, hopefully we've shed some light on the skills that are valuable for applied scientists to have and how you can start building those skills right now, as part of your broader research agenda.

6. AUTHOR BIOS

Nikhil currently works in the Core Ads Growth organization at Meta. Prior to this, he was part of the Sponsored Products organization at Amazon, and even earlier, he co-founded and managed the Algorithms group in Microsoft Research, Redmond. Nikhil obtained his PhD from Georgia Tech and spent a year at Toyota Technological Institute at Chicago. He is interested in what he calls Automated Economics, which studies the question of how technology can be used to improve the efficiency of economic systems.

Renato is a Principal Research Scientist at Google Research New York, where he co-manages the Market Algorithms group in NYC. His group partners with teams in various Google products (ads, search, cloud, . . .) to apply ideas from market design to various products. Before Google, Renato obtained a PhD in Computer Science from Cornell University and did a post-doc at Microsoft Research Silicon Valley.

Okke is a Research Scientist Manager for the Experimentation and Market Algorithms team on Central Applied Science at Meta. He obtained his PhD in Computer Science from Stanford in 2017 under supervision of Tim Roughgarden and has worked at Facebook/Meta as an individual contributor and manager since then.

REFERENCES

Angrist, J. D. and Pischke, J.-S. 2009. *Mostly harmless econometrics: An empiricist's companion.* Princeton university press.

Athey, S. and Imbens, G. W. 2017. The econometrics of randomized experiments. In *Handbook of economic field experiments.* Vol. 1. Elsevier, 73–140.

Harper, F. M. and Konstan, J. A. 2015. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst. 5,* 4 (Dec.).

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. 2020. Array programming with NumPy. *Nature 585,* 7825 (Sept.), 357–362.

Hunter, J. D. 2007. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering 9,* 3, 90–95.

Kleppmann, M. 2019. *Designing data-intensive applications.* English.

Kohavi, R., Tang, D., and Xu, Y. 2020. *Trustworthy online controlled experiments: A practical guide to a/b testing.* Cambridge University Press.

McKinney, W. 2022. *Python for data analysis: Data wrangling with pandas, numpy, and jupyter.* O'Reilly Media, Inc.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12*, 2825–2830.

Su, K., Huo, Y., Zhang, Z., Dou, S., Yu, C., Xu, J., Lu, Z., and Zheng, B. 2024. Auctionnet: A novel benchmark for decision-making in large-scale games. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track.*

VanderPlas, J. 2016. *Python data science handbook: Essential tools for working with data.* O'Reilly Media, Inc.

Waskom, M. L. 2021. seaborn: statistical data visualization. *Journal of Open Source Software 6,* 60, 3021.

Wes McKinney. 2010. Data Structures for Statistical Computing in Python. In *Proceedings of the 9th Python in Science Conference*, Stéfan van der Walt and Jarrod Millman, Eds. 56 – 61.

Wickham, H. 2014. Tidy data. *Journal of statistical software 59*, 1–23.

Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L. D., François, R., Grolemund, G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T. L., Miller, E., Bache, S. M., Müller, K., Ooms, J., Robinson, D., Seidel, D. P., Spinu, V., Takahashi, K., Vaughan, D., Wilke, C., Woo, K., and Yutani, H. 2019. Welcome to the tidyverse. *Journal of Open Source Software 4,* 43, 1686.

Wickham, H., Grolemund, G., et al. 2017. *R for data science.* Vol. 2. O'Reilly Media, Inc.

Xu, A. 2020. *System design interview: An insider's guide.* Independently published.