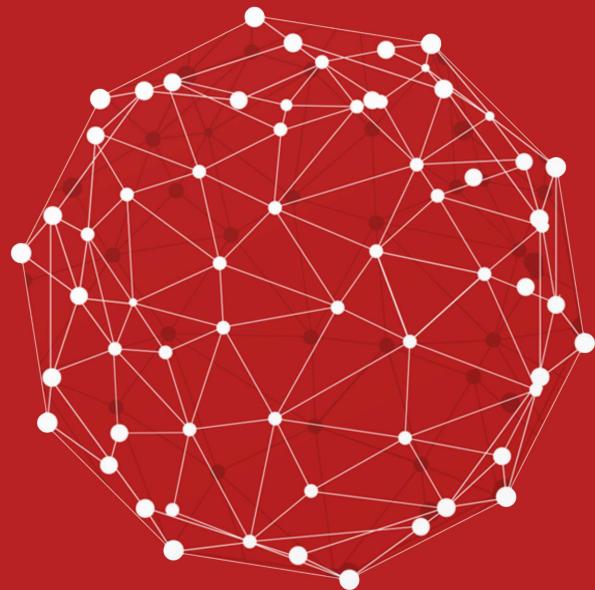


— ACE THE — DATA SCIENCE INTERVIEW



201 Real Interview Questions Asked By
Top Tech Companies & Wall Street Firms

KEVIN HUO NICK SINGH

Ex-Facebook, Now Wall Street

Ex-Facebook, Ex-Data Startup

Praise for Ace the Data Science Interview

“The advice in this book directly helped me land my dream job.”

— Advitya Gemawat, ML Engineer, Microsoft

“Super helpful career advice on breaking into data, and landing your first job in the field.”

— Prithika Hariharan, President of Waterloo Data Science Club
Data Science Intern, Wish

“Solving the 201 interview questions is helpful for people in ALL industries, not just tech!”

— Lars Hulstaert, Senior Data Scientist, Johnson & Johnson

“FINALLY! A book like Cracking the Coding Interview but for Data Science & ML!”

— Jack Morris, AI Resident, Google

“The authors explain exactly what hiring managers look for — a must read for any data job seeker.”

— Michelle Scarbrough
Former Data Analytics Manager, F500 Co.

“Ace the Data Science Interview provides a comprehensive overview of the information an academic needs to transition into industry. I highly recommend this book to any graduate student looking to navigate data science job interviews and prepare for the corporate sector.”

— Lindsay Warrenburg, PhD; Data Scientist, Sonde Health
Head of Data Science Projects at The Erdős Institute

“What I found most compelling was the love story that unfolds through the book. From the first date to the data science interview, Ace reveals his true character and what follows is incredible. I’m thrilled by their avant garde style that uses career advice as a vehicle for fictional narrative.

Once you pick up on it, you feel as though you’re in on a secret even the authors weren’t aware of!”

— Naveen Iyer, Former Machine Learning Engineer, Instagram

“Covers pretty much every topic I’ve been tested on during data science & analytics interviews.”

— Jeffrey Ugochukwu, Data Analyst Intern, Federal Reserve; UC Davis Statistics’ 23

“An invaluable resource for the Data Science & ML community.”

— Aishwarya Srinivasan, AI & ML Innovation Leader, IBM

“Highly recommend this for aspiring or current quantitative finance professionals.”

— Alex Wang, Portfolio Analytics Analyst, BlackRock

“I strongly recommend this book to both data science aspirants and professionals in the field.”

— Chirag Subramanian, Former Data Scientist, Amwins Group
Georgia Tech, MS in Analytics’ 23

“Perfectly covers the many real-world considerations which ML interview questions test for.”

— Neha Pusarla, Data Science Intern, Regeneron
Columbia, MS in Data Science’ 21

“Amazing tips for creating portfolio projects, and then leveraging them to ace behavioral interviews.”

— Jordan Pierre, ML Engineer, Nationwide Insurance

“Nick, Kevin, and this book have been extremely helpful resources as I navigate my way into the world of data science.”

— Tania Dawood, USC MS in Communications Data Science ‘23

“Excellent practice to keep yourself sharp for Wall Street quant and data science interviews!”

— Mayank Mahajan, Data Scientist, Blackstone

“The authors did an amazing job presenting the frameworks for solving practical case study interview questions in simple, digestible terms.”

— Rayan Roy, University of Waterloo Statistics’ 23

“Navigates the intricacies of data science interviews without getting lost in them.”

— Sourabh Shekhar, Former Senior Data Scientist,
Neustar & American Express

—ACE THE— DATA SCIENCE INTERVIEW

201 Real Interview Questions Asked By
FAANG, Tech Startups, & Wall Street

KEVIN HUO
Ex-Facebook, Now Hedge Fund

NICK SINGH
Ex-Facebook, Now Career Coach

About Kevin Huo

Kevin Huo is currently a Data Scientist at a Hedge Fund, and previously was a Data Scientist at Facebook working on Facebook Groups. He holds a degree in Computer Science from the University of Pennsylvania and a degree in Business from Wharton. In college, he interned at Facebook, Bloomberg, and on Wall Street. On the side, he's helped hundreds of people land data jobs at companies including Apple, Lyft, and Citadel.

About Nick Singh

Nick Singh started his career as a Software Engineer on Facebook's Growth Team, and most recently, worked at SafeGraph, a geospatial analytics startup. He holds a degree in Systems Engineering with a minor in Computer Science from the University of Virginia. In college, he interned at Microsoft and on the Data Infrastructure team at Google's Nest Labs. His career advice has been read by over 10 million people on LinkedIn.

All rights reserved. No part of this book may be used or reproduced in any manner without written permission except in the case of brief quotations in critical articles or reviews.

The author, and/or copyright holder, assume no responsibility for the loss or damage caused or allegedly caused, directly or indirectly, by the use of information contained in this book. The authors specifically disclaim any liability incurred from the user or application of the contents of this book.

Throughout this book, trademarked names are referenced. Rather than using a trademark symbol with every occurrence of a trademarked name, we state that we are using the names in an editorial fashion only and to the benefit of the trademark owner, with no intention of infringement of the trademark.

Copyright 2021 Ace the Data Science Interview.

All rights reserved.

ISBN 978-0-578-97383-8

For my family: Bin, Jing, Matt, and Allen
~ Kevin

For Mom and Dad, Priya and Dev, and my brother, Naman—
My family, who supports me in every endeavor

~ Nick

Table of Contents

Introduction.....	vii-x
-------------------	-------

Career Advice to Ace the Data Science Job Hunt

Chapter 1

4 Resume Principles to Live by for Data Scientists	1-10
--	------

Chapter 2

How to Make Kick-Ass Portfolio Projects.....	11-16
--	-------

Chapter 3

Cold Email Your Way to Your Dream Job in Data	17-26
---	-------

Chapter 4

Ace the Behavioral Interview	27-34
------------------------------------	-------

Ace the Technical Data Science Interview

Chapter 5

Probability	35-52
-------------------	-------

Chapter 6

Statistics	53-76
------------------	-------

Chapter 7

Machine Learning.....	77-140
-----------------------	--------

Chapter 8

SQL & DB Design.....	141-180
----------------------	---------

Chapter 9

Coding.....	181-232
-------------	---------

Chapter 10

Product Sense	233-270
---------------------	---------

Chapter 11

Case Studies.....	271-290
-------------------	---------

Introduction

Data scientists are not only privileged to be solving some of the most intellectually stimulating and impactful problems in the world — they’re getting paid very well for it too. At Facebook, the median total compensation reported by Levels.fyi for a Senior Data Scientist is a whopping \$253,000 per year. According to data on Glassdoor, the median base salary for a Quantitative Researcher at hedge fund Two Sigma is \$173,000 per year, with opportunities to DOUBLE take-home pay thanks to generous performance bonuses.

Given how intellectually stimulating and damn lucrative data science is, it shouldn’t be a surprise that competition for these top data jobs is fierce. Between “entry-level” positions in data science weirdly expecting multiple years of experience, and these entry-level jobs themselves being relatively rare in this field saturated with Ph.D. holders, early-career data scientists face hurdles in even landing interviews at many firms.

Worse, job seekers at all experience levels face obstacles with online applications, likely never hearing back from most jobs they apply to. Sometimes, this is due to undiagnosed weaknesses in a data scientist’s resume, causing recruiters to pass on talented candidates. But often, it’s simply because candidates aren’t able to stand out from the sea of candidates an online job application attracts. Forget about acing the data science interview — given the amount of job hunting challenges a data scientist faces, just getting an interview at a top firm can be considered an achievement in itself.

Then there’s the question of actually passing the rigorous technical interviews. In an effort to minimize false positives (aka “bad hires”), top companies run everyone from interns to industry veterans through tough technical challenges to filter out weak candidates. These interviews cover a lot of topics because the data science role is itself so nebulous and varied — what one company calls a data scientist, another company might call a data analyst, data engineer, or machine learning engineer. Only after passing these onerous technical interviews — often three or four on the same day — can you land your dream job in data science.

We know this all must sound daunting. Spoiler alert: it is!

The good news is that in this book we teach you exactly how to navigate the data science job search so that you can land more interviews in the first place. We’ve put together a shortlist of the most essential topics to brush up on as you prepare for your interviews so that you can ace these tough technical questions. Most importantly, to put your technical skills to the test, we included 201 interview questions from real data scientist interviews. By solving actual problems from FANG companies, Silicon Valley darlings like Airbnb and Robinhood, and Wall Street firms like Two Sigma and Citadel, we’re confident our book will prepare you to Ace the Data Science Interview and help you land your dream job in data.

Who Are We?

Who are we, and how’d we find ourselves writing this book?

I (Nick) have worked in various data-related roles. My first internship was at a defense contractor, CCRi, where I did data science work for the U.S. Intelligence Community. Later in college, I interned as a software engineer at Microsoft and at Google’s Nest Labs, doing data infrastructure engineering. After graduating from the University of Virginia with a degree in systems engineering, I started my

full-time career as a new grad software engineer on Facebook's Growth team. There, I implemented features and ran A/B tests to boost new user retention.

After Facebook, I found myself hungry to learn the business side of data, so I joined geospatial analytics startup SafeGraph as their first marketing hire. There, I helped data science and machine learning teams at Fortune 500 retailers, hedge funds, and ad-tech startups learn about SafeGraph's location analytics datasets.

On the side, I started to write about my career journey, and all the lessons I learned from being both a job applicant and an interviewer. Ten million views on LinkedIn later, it's obvious the advice struck a nerve. From posting on LinkedIn, and sending emails to my tech careers newsletter with 45,000 subscribers, I've been privileged to meet and help thousands of technical folks along their career journey. But, there was one BIG problem.

As a mentor, I was able to point software engineers and product managers to many resources for interview prep and career guidance, like *Cracking the Coding Interview*, *Cracking the PM Interview*, and LeetCode. But, from helping data scientists, I realized just how confusing the data science interview process was, and how little quality material was out there to help people land jobs in data. I reached out to Kevin to see if he felt the same way.

You might be wondering...

Why'd I turn to Kevin?

For several reasons! We're longtime friends, having attended high school together in Northern Virginia at Thomas Jefferson High School for Science and Technology. Though we went our separate ways for college, we became close friends once again after becoming roommates in Palo Alto, California, when we both worked for Facebook as new grads, and bonded over our shared love of Drake.

By living with Kevin, I learned firsthand three things about him:

1. Kevin is an expert data scientist.
2. Kevin loves helping people.
3. Kevin is a fantastic freestyle rapper.

Because my rapping skills paled in comparison to Kevin's, and I can't sing worth a damn (even though my last name is Singh), it made sense to delay the mixtape and instead focus on our other shared passion: helping people navigate the data science job hunt.

Kevin has successfully landed multiple offers in the data world. It started in college, when he interned on the Ad Fraud team at Facebook. After graduating from the University of Pennsylvania with a major in computer science, and a degree in business from Wharton, Kevin started his career as a data scientist at Facebook, where he worked on reducing bad actors and harmful content on the Facebook Groups' platform. After a year, Wall Street came calling. Kevin currently works as a data scientist at a hedge fund in New York.

On the side, Kevin combined his passion for data science and helping people, which led him to found DataSciencePrep.com, become a course creator on *DataCamp*, and coach dozens of people in their data science careers.

Ace the Data Science Interview results from Kevin's and my experience working in Silicon Valley and Wall Street, the insights we've garnered from networking with recruiters and data science managers, our personal experience coaching hundreds of data scientists to land their dream role, and our shared frustration with the career and interview guidance that's been available to Data Scientists — that is, until now!

What Exactly Do We Cover?

We start with a “behind the scenes” look at how recruiters and hiring managers at top companies evaluate your resumes and portfolios so you can see what it takes to stand out from the rest. After reviewing hundreds of resumes, we've seen technical folks make the same mistakes over and over again. But not you, after you follow the advice in Chapter 1.

In Chapter 2, we show you how to make kick-ass portfolio projects. These projects will leap off the resume, and will make any person reading your application want to interview you. A well-crafted portfolio project will also help you ace the behavioral interview.

But how do we get folks to read your application in the first place?

In Chapter 3, we teach you how to cold-email your way to your dream job in data. We give you a new way of finding a job so that you don't have to keep applying online and getting ghosted. By getting to the top of the recruiter's and hiring manager's email inbox, you'll get noticed, start the networking process early, and often get an inside scoop into the role.

Next comes Chapter 4: Ace the Behavioral Interview. While there's no one right answer to “tell me about yourself” or “do you have any questions for us,” there are plenty of wrong ways to approach these conversations. Learn how to avoid these mistakes, craft a better personal story, and tailor your answers, so that the interviewer is left thinking you're born for the role.

Finally, we're ready for the trickiest part of the data science job hunt, and the meat of our book: acing the technical data science interview. Chapters 5–10 give you an overview of the common technical subjects asked during data science interviews. We detail what to brush up on and what to skip, for topics within Probability, Statistics, Machine Learning, SQL & Database Design, Coding, and Product Sense. In Chapter 11 — the boss chapter — we cover how to approach open-ended case questions, which blend multiple topics into one big problem.

Each of these technical chapters also tests your knowledge with real interview questions asked by tech behemoths like Facebook, Google, Amazon, and Microsoft, mid-sized tech companies like Stripe, Robinhood and Palantir, and Wall Street's biggest banks and funds like Goldman Sachs, Two Sigma, and Citadel. With problems organized into easy, medium, and hard difficulty levels, there is something to learn for everyone from the data science neophyte all the way to a Kaggle champion. If you get stuck, there's nothing to fear, as each problem has a fully worked-out solution too.

Additional Resources to Accompany the Book

Alongside reading this book, you are 94.6% encouraged to join our Instagram community at instagram.com/acedatascienceinterviews for additional career tips, interview problems, memes, and the chance to see our glowing faces from time to time when we flex on the gram.

Also, make sure you've subscribed to Nick's monthly career advice email newsletter: nicksingh.com/signup
It's just one email a month with the latest tips, resources, and guides to help you excel in your technical career.

And speaking of email, if you have suggestions, find any mistakes, have success stories to share, or just want to say hello, send us an email: hello@acethedatascienceinterview.com or feel free to connect with us on social media.

Nick Singh

nicksingh.com — where I blog my long-form essays and career guides

[linkedin.com/in/Nipun-Singh](https://www.linkedin.com/in/Nipun-Singh) --- where I share career advice daily (please send a connection request with a message that you've got the book. I'm close to the 30k connection limit so don't want to miss your connection request!)

[instagram.com/DJLilSingh](https://www.instagram.com/DJLilSingh) — for a glimpse into my hobbies (DJing and being Drake's #1 stan)

twitter.com/NipunFSingh — for tweets on careers and tech startups

Kevin Huo

[linkedin.com/in/Kevin-Huo](https://www.linkedin.com/in/Kevin-Huo)

[instagram.com/Kwhuo](https://www.instagram.com/Kwhuo)

4 Resume Principles to Live by for Data Scientists

CHAPTER 1

Before you kick off the job hunt, get your resume in order. Time and effort spent here can pay rich dividends later on in the process. No one is going to grant you an interview if your resume doesn't scream success and competence. So here are four principles your resume should live by, along with miscellaneous hacks to level up. We even included our actual resumes from our senior year of college to show you how, in real life, we practiced what we preach to land our dream jobs at Facebook.

Principle #1: The Sole Purpose of Your Resume Is to Land an Interview

No resume results in an immediate job offer; that isn't its role. What your resume must do is convince its recipient to take a closer look at you. During the interview process, your data science chops and people skills will carry you toward an offer. Your resume merely opens the door to the interview process. In practice, that means keeping your resume short! One page if you have under a decade of experience, and two pages if more. Save whatever else you want to say for your in-person interview when you'll be given ample time to get into the weeds and impress the interviewer with your breadth of knowledge and experience.

Since the main aim of your resume is to land an interview, the best way to do that is by highlighting a few of your best achievements. It's crucial that these highlights are as easy and as obvious as possible to find so that the person reading your resume decides to grant you an interview. Recruiters are busy

people who often only have 10 seconds or less to review your resume and decide whether to give you an interview. Keeping the resume short and removing fluff is key to making sure your highlights shine through in the short timespan when a recruiter is evaluating your resume.

One way to shorten the resume and keep it focused on the highlights is by omitting non-relevant jobs. Google doesn't care that you were a lifeguard three summers ago. The exception to this rule is if you have zero job experience. If that's the case, do include the job on your resume to prove that you've held a position that required a modicum of "adulting."

Another way to make sure your resume lands you an interview is by tailoring the resume to the specific job and company. You want the recruiter reading your resume to think you are a perfect fit for the position you're gunning for. For example, say you worked as a social media assistant part-time three years ago promoting your aunt's restaurant. This experience isn't relevant for most data science roles and can be left off your resume. But if you're applying to Facebook and interested in ads, it's worth including. If you're applying to be a data analyst on a marketing analytics team, it can help to leave in the social media marketing job.

Another resume customization example: when I (Nick) applied to government contractors with my foreign-sounding legal name (Nipun Singh), I put "U.S. Citizen" at the top of my resume. This way, the recruiter knew I had the proper background to work on sensitive projects, and could be eligible later for a top secret security clearance.

Principle #2: Build Your Resume to Impress the Recruiter

The person you most need to impress with your resume is a nontechnical recruiter. The senior data scientist or hiring manager at the company you want to work for is NOT your target audience — they will have a chance to review your experience in-depth during your on-site interview. As such, spell out technical acronyms if they aren't obvious. Give a little background; don't just assume the recruiter will understand what you did and why it's impressive. For example, a research project called "Continuous Deep Q-Learning with Model-Based Acceleration" doesn't make sense to most people. But a "Flappy Bird Bot Using Machine Learning" is more memorable and intriguing to the average nontechnical recruiter.

Do you know what else recruiters love (along with hiring managers and execs)?

Numbers. BIG numbers!

Don't be afraid to report usage or view counts for your projects. Talking about user metrics shows that you drove a project to completion, and got it in front of real people who were impacted by your work. Even if the project is technically straightforward, real user or view counts go a long way in helping a recruiter understand you made something of value.

For example, in college, I (Nick) made RapStock.io. The website didn't have a sleek UI, and the underlying code wasn't complex, but at its peak, it had 2,000 Monthly Active Users. This experience opened many doors and gave me a great story to tell because recruiters realized I'd actually shipped something of value before.

An even better way to impress the recruiter is if you can quantify your impact in business terms: write out the specific dollars earned, or dollars saved, due to your work. Most recruiters would prefer to read about the analysis you did that led to \$20,000 in savings for a business rather than that small little side project where you solved P vs. NP.

Why? Because it's hard to explain the P vs. NP problem and why it matters on a resume, unless you describe it with secondary results like, "Won a Nobel Prize." But \$20,000 cash is \$20,000 cash; no explanation needed. And because most data science job seekers are applying to businesses, talking about real value you've generated in the past gives businesses confidence that you'll do the same at their company. The truth is that, ultimately, what the recruiter is looking for isn't necessarily an expert data scientist per se, but someone who will move the business and product forward — who just so happens to do it using data science!

Principle #3: Only Include Things That Make You Look Good

Your resume should make you shine, so don't be humble or play it cool. If you deserve credit for something, put it on your resume. But never lie or overstate the truth. It's easy for a recruiter or other company employee to chat with you about projects and quickly determine if you did it all or if it was a three-person group project you're passing off as your own. And technical interviewers love to ask probing questions about projects, so don't overstate your contributions.

Another way to look good is to not volunteer negative information. Sounds obvious enough, but I've seen people make this mistake often. For example, you don't have to list your GPA! Only write down your GPA if it helps. A 3.2 at MIT might be okay to list, but a 3.2 at a local lesser-known college might not be worth listing if you are applying to Google, especially if Google doesn't usually recruit at your college. Why? Because Google being Google might be expecting you to be at the top of your class with a 4.0 coming from a non-target university. As a result, a 3.2 might look bad.

Avoid Neutral Information

A mistake more common than volunteering negative information is adding neutral details to your resume. Drowning out your accomplishments with irrelevant information detracts from your resume in much the same way as volunteering negative information. Remember: the average recruiter will only spend around 10 seconds skimming your resume, so you can't afford for them to be distracted and miss your strongest points.

One big source of neutral information is the summary or objective section at the top. Your aim must be to look exceptional, not typical or neutral. How boring and undifferentiated is: "hard-working, results-oriented analytics professional looking for a Data Science job starting fall 2022." Worse, this section is usually right at the top of a resume, taking up valuable real estate. Get rid of it completely!

You Probably Don't Need a Skills or Technologies Section

Another section of the resume packed with neutral details is the skills and technologies section. Traditional resume advice says to jampack keywords in here. We disagree. You can eliminate this section entirely or shorten it to two lines max (if you do choose to include it). There are several reasons why we advocate shortening or removing the skills and technologies section.

First off, we need to address why traditional resume advice advocates for including this section. The reasoning is that to please the application tracking system (ATS), which has an algorithm that flags a recruiter that your application is relevant to the job description you applied for, you need to stuff your resume with keywords. We don't agree with this advice. As we detail in Chapter 3, applying online is an ineffective strategy, so pleasing the ATS isn't paramount. Also, anything you list on your resume is fair game for the interviewer to ask you about. Filling this section with tools you aren't familiar with to please the ATS algorithm can easily backfire come interview time.

Another reason to get rid of the skills section: remember “Show and Tell” in grade school? Well, it’s still way better to show rather than just to tell! Include the technologies inline with your past work experience or portfolio projects. Listing the tech stack this way contextualizes the work you did and shows an interviewer what you’re able to achieve with different tools. Plus, in explaining your projects and past work experiences, you’ll have enough keywords covered to appease the ATS which traditional career advice is overly focused on.

The last reason to ditch a big skills and technologies section is that you are expected to learn new languages and frameworks quickly at most companies. The specific tools you already know are helpful but not crucial to landing a job. You are expected to be an expert at data science in general, not in specific tools. Plus, at large tech companies like Facebook and Google, the tools and systems are often proprietary. Thus, it doesn’t matter much about the specific tools you know — it’s about what you’ve actually accomplished with those tools in the past.

Mistakes College Students and New Grads Often Make

Listing too many volunteer experiences and club involvements from high school and college is a frequent mistake we see college students and new grads make on their resume. They still think it’s like college applications, where listing your involvement in varsity soccer, piano lessons, and the National Honor Society means something. It’s great that you are a civically engaged, respectable human involved with your community, but competitive tech companies and Wall Street firms are selecting you for your ability to turn data into insight, and not much other than that. Attending ACM meetings or going to the data science club is practically worthless as far as resume material goes. Unless your involvement was significant, don’t list it.

Another source of potentially irrelevant details: things from a long time ago, like your SAT score, or which high school you attended. One caveat that traditional career services folks don’t tell you to do: leave in details from high school if they are exceptional.

Got more than a 2350 (or 1450) on your SAT? Leave it in!

Did well in USAMO, ISEF, or Putnam? Leave it in!

Are you a NCAA athlete or attended college on a full-ride merit scholarship? Leave it in!

Same goes if you attended a prestigious high school like Phillips Exeter, Harker, or Thomas Jefferson High School for Science & Technology (go Colonials!).

We’ve found that at elite tech companies and Wall Street firms, the interviewers went to these same schools and won these same competitions. It may be okay to keep on your resume even if it’s from a long while ago, provided it doesn’t take too much space.

And if none of this applies to you, don’t worry — there’s so much more to you and your resume than the brand names you’ve listed.

Principle #4: Break Formatting and Convention to Your Favor

We believe traditional resume advice is too rigid, with a one-size-fits-all approach. If breaking conventional writing rules makes your resume more readable, then do it!

Typically, bold font is reserved for section headings. But feel free to bold any words or phrases that make your resume easier to understand so the interviewer can decide — in 10 seconds or less! — to interview you.

For example, back in college, I (Nick) made the size of the companies I worked at bigger than other text. This way, while scanning my resume, a recruiter could quickly see and say, “Microsoft Intern. Check. Google Intern. Check. Okay, let’s interview this kid.” I also bolded the user metrics for RapStock.io on my resume. I wanted to quickly call out this information because it was sandwiched between additional details about my projects.

Another resume convention you can break to your favor is section order. There is no hard and fast rule on ordering your education, work experience, projects, and skills sections. Just remember: in English, we scan from top to bottom. So list your best things at the top of the resume.

Keeping what’s most important up top is an important piece of advice to remember, since it may conflict with advice from many college career advisors who suggest listing your university at the top. For example, if you currently attend a small unknown university, but interned at Uber this summer, don’t be afraid to list your work experience with Uber at the top, and move education to the bottom. Went to MIT but have no relevant industry internships yet? Then it’s fine to lead with your education, and not the work experience section.

Another resume rule on ordering you can break: items within a section don’t have to be in chronological order! For example, I had a friend who interned at Google one summer, then interned part-time at a small local startup later that fall. It’s okay to keep Google at the top of the resume, ahead of the small local startup, even though the startup experience was more recent work experience. List first what makes you look the best.

Another convention you can safely break is keeping standard margins and spacing. Change margins to your favor, to give yourself more space and breathing room. You can also use different font sizes and spacing to emphasize other parts of your resume. Just don’t use margin changes to double the content on your one-page resume. If you do, you’re likely drowning out your best content, which is a big no-no (as mentioned in Principle #3).

Oh, and speaking of breaking the resume rules: you can ignore any of the tips I’ve listed earlier in this chapter if it helps you tell a better story on your resume. Earlier, I mentioned that listing irrelevant jobs — like being a waiter — won’t help you land a data science gig. But go ahead and list it if, for example, you were a waiter who then built a project that analyzed the data behind restaurant food waste. Same way, there’s nothing wrong with listing the waiter position if you’re applying to DoorDash or Uber Eats. If listing something helps tell the story of you, leave it in.

So don’t listen to folks who tell you that linking to your SoundCloud from your resume is unprofessional. Suppose you’ve made a hackathon project around music, or are applying to a company like Spotify. In that case, it’s perfectly fine to list the SoundCloud link since it shows a recruiter that you followed your passions and created projects to further your interests. And by the way, if your mixtape is fire, please email us the link to hello@acethedatascienceinterview.com and we’ll give it a listen.

Miscellaneous Resume Hacks

Make Your Name and Email Prominent

Self-explanatory. Near your name, add a professional email address too. Also, do not use Yahoo mail or Hotmail addresses. Silicon Valley tech folks are especially judgy about this.

Never Include Your Mailing Address

Companies are biased toward hiring local candidates because they don’t need to pay relocation fees. And recruiters are compensated based on the number of candidates they can close. So, put yourself

in a recruiter's chair. Let's say you're a Silicon Valley-based company recruiter with two identically-skilled candidates, but one lives in the Bay Area and the other lives in NYC. Which of the two are you more likely to close? The NYC candidate who needs to decide to move to SF and uproot her family before accepting your offer or the local person who can take your offer and start next week?

Don't List Your Phone Number Unless It's Local

Because of the spam robocall epidemic, anyone who calls you will email you first to ask for your phone number and set up a time. So, there's no need to list it. Remember: you have 10 seconds to rivet someone reading your resume. Don't waste a second of their time by presenting nonessential information. Plus, if your phone number is international, it'll hurt even more, as often there's a bias to hire local candidates. And yes, hiring managers and recruiters do notice when your number is from a far-flung area code.

Include Your GitHub

Your GitHub link doesn't need to be super prominent. It's okay if your GitHub is a bit messy. Merely having a GitHub listed is a sign that you have done work out in the open and that you're aware of version control. And remember: since the first interview gatekeeper is a nontechnical recruiter, he or she most likely won't be able to tell a messy GitHub from an okay one anyway.

Hyperlink to Outside Work Where Possible

Include links to your data analysis blogs or hackathon projects whenever possible. Linking to the outside world validates to the recruiter that you made something and published it to the real world. Even if the recruiter never clicks on a link, just seeing the blue hyperlink will give them a sense that there's something more there.

Save Your Resumes as PDFs with Good Names

PDFs maintain formatting better and lead to better viewing experiences on mobile.

And be sure to save your resumes as "First Name Last Name Company Name Resume."

When people name it "Resume.pdf," it implies you're applying to many jobs willy-nilly. You should be customizing your resumes for specific companies and roles anyway, so saving them with the company name included will help you be more organized.

Resume Hacks for IRL

The first hack for sharing a resume in real life is to print your resume on heavier paper. Doing this will make the resume stand out, and it will feel more professional. Next, bring your resume to every place where you'll be meeting hiring managers and recruiters: job fairs, coffee chats, conferences, meetups, and, of course, onsite interviews.

You might be skeptical of the value of doing this, especially since your LinkedIn and personal website might have all the same information, but here's the main reason: you don't want to have to pull these up on your phone if someone is curious. In many contexts, it just is way easier to hand them a piece of paper.

And there's a subtle reason for doing this too. Even if the person you are trying to network with doesn't read your resume in the moment, they'll likely hang onto it and read it later while waiting for their Uber or when they're stuck at the airport. It's a physical memento you gave them, and they'll be more likely to remember you and respond to your email follow-ups later (covered in Chapter 3).

And lastly, since you're carrying your resume everywhere, get a folio to carry it. Crumpled resumes look unprofessional. If you are in college or are a new grad, and went to a top school, get a leather padfolio with the school logo front and center for the subtle flex.

Nick's Facebook Resume (Senior Year, Fall Semester)

Nick (Nipun) Singh

ns2se@virginia.edu, nipunsingh.com, github.com/NipunSingh

Experience:

Google/Nest Labs, Software Engineering Intern

May-Aug 2016

- On the Date Infrastructure team, built a monitoring & deployment tool for GCP Dataflow jobs in Python (Django)
- Wrote Spark jobs in Scala to take Avro-formatted data from HDFS and published it to Google's Pub-Sub services

Microsoft, Software Engineering Intern

May-Aug 2015

- Reduced latency from 45 seconds to 80 milliseconds for a new monitoring dashboard for payments team
- Did the above by developing an efficient ASP.NET Web API in C# which leveraged caching and pre-processing of payment data queried from Cosmos via Scope (Microsoft internal versions of Hadoop File System and Hive)

CCRI, Data Science Intern

Jun-Aug 2014

- Worked on a NLP algorithm for a contract with the Office of Naval Research
- Improved F1 measure of algorithm 70% compared to the original geo-location algorithm used by Northrup Grumman, by designing new algorithm in Scala which used Stanford NLP package to geo-locate events in news

Projects

Founder, RapStock.io

Jan-May 2015

- Grew site to 2,000 Monthly Active Users, and received 150,000 page views
- Developed using Python (Django), d3.js, JQuery, Bootstrap, PostgreSQL, and deployed to Heroku
- Game similar to fantasy football but players bet on the real world popularity and commercial success of rappers --- the rappers' performance is based on metrics scraped from Spotify and Billboards
- "Great to see that folks stick around" - Alexis Ohanian, Founder of Reddit, commenting on our retention metrics

PennApps Hackathon, AutoCaption.co

Sep 2015

- Won ‘Best Use Of Amazon Web Services’ award at the 2,000 person Hack-a-thon
- Created a website which takes in a photo and then automatically captions it.
- Used IBM Watson API and Image recognition for tagging. Scrapped 7,000 jokes and quotes with Python. Used and tuned AWS Cloud Search to search the database of captions. Backend built with Django.

Education

University of Virginia

2013-2017

- B.S. Systems & Information Engineering, Minor in Computer Science and Applied Math
- Rodman Scholar (Top 5% of Engineering Class)

Thomas Jefferson High School for Science and Technology

2009-2013

- SAT: 2360 (800 Math, 800 Writing, 760 Reading)

Technologies

- Languages: Python (Django), Java, Scala, R
- Other: PostgreSQL, Spark, Google Cloud Platform, AWS, Heroku

Entrepreneurial Activities

Entrepreneurship Group at UVA

Jan 2014-Aug 2015

- Relations Officer for a club with 130 active members: planned speaker events and pitch competitions

Mobile DJ Business (DJ Lil Singh)

Jun 2012-Aug 2013

- Performed at 25 events as a Hip-Hop & Bollywood DJ, providing music, lighting, & MC services

Kevin’s Facebook Resume (Senior Year, Fall Semester)

Kevin Huo

EDUCATION

University of Pennsylvania - Philadelphia, PA

Graduating: May 2017

The Wharton School: BS in Economics with concentrations in *Statistics & Finance*

School of Engineering and Applied Sciences: BSE in *Computer Science*

GPA: 3.65/4.00

Honors: Dean’s List (2013-2014), PennApps Health Hack Award (2014)

Statistics Coursework: Modern Data Mining, Statistical Inference, Stochastic Process, Probability, Applied Probability Modeling

Computer Science Coursework: Algorithms, Data Structures, Automata and Complexity, Databases, Intro to Data Science, Software Engineering, Machine Learning

Finance Coursework: Investment Management, Derivatives, Monetary Economics

Thomas Jefferson High School for Science and Technology - Alexandria, VA Graduated: June 2013

GPA: 4.44/4.00 (Weighted), SAT: 2350 (Math-800, Writing-800, Reading-750)

Honors: National Merit Finalist, National AP Scholar, American Invitational Math Exam (AIME) Qualifier

ACTIVITIES

Wharton Undergraduate Data Analytics Club (Team Leader) January 2016-January 2017

- Participated in speaker series, tech talks, and data hackathons as a general member
- Project leader for consulting group performing analyses

WORK EXPERIENCE

Facebook (Data Science Intern) June 2016-August 2016

- Analyzed fraud within Atlas by looking at edge cases among existing systems
- Built cost view of fraud for advertisers and presented recommendations to relevant teams
- Technologies used: SQL, R, Python

Bloomberg LP (Software Engineering Intern) May 2015-August 2015

- Developed a contributor analysis tool for the Interest Rate Volatility Team
- Constructed various statistical metrics to gauge contributors
- Built UI component using JavaScript and in-house technologies
- Wrote various Python scripts to monitor metrics
- Technologies used: Python, JavaScript

Zetta Mobile (Software Engineering Intern) June-August 2014

- Wrote Python scripts to compile recorded data from logs of mobile advertisements
- Used R to look for useful trends and patterns in the compiled data
- Built scripts to automate the data analysis of click-through rate
- Scripts are being used in beta-testing for future automated data analyses for the company to use
- Technologies used: Python

Computer Science Teaching Assistant January 2014-December 2016

- Held weekly recitation and office hours and responsible for grading of homework, tests, and quizzes
- Discrete Math (Spring & Fall 2014), Data Structures & Algorithms (Spring/Fall 2015 & 2016)

LANGUAGES/FRAMEWORKS

- Proficient: Python, R, SQL, Java, Familiar: JavaScript, HTML/CSS, Basic: OCaml, Hadoop, Linux

How to Make Kick-Ass Portfolio Projects

CHAPTER 2

Unanimously, data science hiring managers have told us that not having portfolio projects was a big red flag on a candidate's application. This holds true especially for college students or people new to the industry, who have more to prove. From mentoring many data scientists, we've found that having kick-ass portfolio projects was one of the best ways to stand out in the job hunt. And from our own experience, we know that creating portfolio projects is a great way to apply classroom knowledge to real-world problems in order to get some practical experience under your belt. Whichever way you slice it, creating portfolio projects is a smart move. In this chapter, you'll learn 5 tips to level-up your data science and machine learning projects so that recruiters and hiring managers are jumping at the chance to interview you. We teach you how to create, position, and market your data science project. When done right, these projects will give you something engaging to discuss during your behavioral interviews. Plus, they'll help make sure your cold emails get answered (Chapter 3).

The High-Level Philosophy

As we discussed to death in Chapter 1, the recruiter is the person we need to impress because they are the interview gatekeeper. A recruiter won't dive deep into your Jupyter Notebook, look at line #94, see the clever model you chose, and then offer you an interview. That's not how recruiting (or people!) work.

The majority of recruiters just read the project description for 10 seconds in the cold email you send them or when reviewing your resume. Maybe — if you’re lucky — they click a link to look at a graphic or demo of the project. At this point, usually in under 30 seconds, they think to themselves, “This is neat and relevant to the job description at hand,” and decide to give you an interview.

Thus, we’re optimizing our data science portfolio projects to impress the decision-maker in this process — the busy recruiter. We’re optimizing for projects that are easily explainable via email. We’re optimizing for ideas that are “tweetable”: ones whose essence can be conveyed in 140 characters or less. By having this focus from day one when you kick off the portfolio project, you will skyrocket your chances of ending up with “kick-ass” portfolio project that gets recruiters hooked.

Don’t worry if you think that focusing on the recruiter will cheapen your portfolio project’s technical merits. Believe us: the technical hiring manager and senior data scientists interviewing you will also appreciate how neatly packaged and easily understandable your project is. And following our tips won’t stop you from making the project technically impressive; an interesting and understandable project does not need to come at the expense of demonstrating strong technical data science skills.

Tip #1: Pick a Project Idea That Makes for an Interesting Story

Recruiters and hiring managers are human. Human beings love to hear and think in terms of stories. You can read the book that’s quickly become a Silicon Valley favorite, *Sapiens: A Brief History of Humankind*, by Yuval Harari, to understand how fundamental storytelling is to our success as a species. In the book, Harari argues that it’s through the shared stories we tell each other that Homo sapiens are able to cooperate on a global scale. We are evolutionarily hardwired to listen, remember, and tell stories. So do yourself a favor and pick ideas to work on which help you tell a powerful story.

A powerful story comes from making sure there is a buildup, then some conflict, and a nice, satisfying resolution to said conflict. To apply the elements of a story to a portfolio project, make sure your work has some introductory exploratory data analysis that builds up context around what you are making and why. Then pose a hypothesis, which is akin to a conflict. Finally, share the verdict of your posed hypothesis to resolve the conflict you posed earlier. By structuring your portfolio like a story, it’ll be easier to talk more eloquently about your project in an interview. Plus, the interviewer is hardwired to be more interested — and therefore more likely to remember you and your project — when you tell it in a format that we’re hardwired to love.

So, how do you discover projects that will translate into captivating stories?

Looking at trending stories in the news is a great starting point because they are popular topics that are easy to storytell around. For example, in the fall of 2020, the biggest news stories were the COVID-19 pandemic and the 2020 U.S. presidential election. Interesting projects on these topics could be to look at vaccination rates by zip code for other diseases, and see how they correlate to demographic factors in order to understand healthcare inequities and complications with vaccine rollout plans. For the 2020 U.S. presidential election, an interesting project would be to see what demographic factors correlate highest for a county flipping from Donald Trump in 2016 to Joe Biden in 2020, and then predicting which counties are the most flippable for future elections.

If you ever get stuck on these newsworthy topics, data journalism teams at major outlets like the *New York Times* and *FiveThirtyEight* have already made a whole host of visualizations related to these issues. These can serve as inspiration or as a jumping-off point for more granular analysis.

Another easy source of ideas with good story potential is to think about problems you personally face. You’ll have a great story to tell where you’re positioned as the hero problem-solver if you can convey how annoying a problem was to you and that you needed to solve it for yourself and other

sufferers. I've seen friends at hackathons tackle projects on mental health (something they personally struggled with), resulting in a very powerful and moving narrative to accompany the technical demo.

Tip #2: Pick a Project Idea That Visualizes Well

A picture is worth a thousand words. And a GIF is worth a thousand pictures. So go with a portfolio project idea that visualizes well to stand out to recruiters. Ideally, make a cool GIF, image, or interactive tool that summarizes your results.

I (Nick) saw the power of a catchy visualization firsthand at the last company I worked at, SafeGraph, when we launched a new geospatial dataset. When we just wrote a blog post and put it on SafeGraph's Twitter, we wouldn't get much engagement. But when we included a GIF of the new dataset visualized, we'd get way more attention.

This phenomenon wasn't just isolated to social media — the power of catchy photos and GIFs even extended to cold email. When we'd send sales emails with a GIF embedded at the top, we got much higher engagement than when we'd send boring emails that only contained text to announce a product. These marketing lessons apply to your data science portfolio projects as well, as you should be emailing your work to hiring managers and recruiters (covered in detail in Chapter 3). You might be thinking, "Why are we wasting time on this and not focusing on the complicated technical skills that a portfolio project should demo?"

We want to remind you: your ability to convey results succinctly and accurately is a very real skill. Explaining your work and presenting it well is a great signal to companies, because real-world data science means convincing stakeholders and decision makers to go down a certain path. Fancy models are great, but not unless you can easily explain to higher ups their results and business impact. A compelling visual is one of the easiest ways to accomplish that goal in the business world. Demonstrating this ability through a portfolio project gives any interviewer confidence you'll be able to excel at this aspect of data science when actually on the job.

Tip #3: Make Your Project About Your Passion

Making your portfolio project about your passion is a cheat code for a whole slew of reasons. Passion is contagious. If you're having fun talking about your passion project, chances are those same good vibes will catch on with the interviewer. Plus, when you work on something you're naturally passionate about, it becomes much easier and more comfortable for you to talk about the work during an otherwise nerve-wracking interview. This effortless communication will help you come across as a more articulate communicator — a highly desirable attribute for any hire. Making your project about your passion, and then communicating this passion, also leads to a halo effect, where you come across as passionate for related things, like the field of data science, the job at hand, and the company.

This passion and enthusiasm halo effect is especially crucial to create for more junior data science candidates. Early-career data scientists require more hand holding and resources invested by a company compared to experienced hires. From talking to hiring managers, we found that they chose to invest in more junior candidates when the candidate displayed high amounts of enthusiasm and passion. This enthusiasm and passion is a great signal that the junior candidate will be motivated to learn quickly and close the skill gap fast. Thus, by signaling passion by working on passion projects, you help make companies want to invest in you over a more senior candidate who might have more technical skills but lacks the same interest in the field.

What does this advice mean in practice? If you love basketball, then use datasets from the NBA in your portfolio projects. Passionate about music? Classify songs into genres based on their lyrics.

Binge Netflix shows? Take the IMDB movies dataset and make your own movie recommender algorithm.

For example, I (Nick) — a passionate hip hop music fan and DJ that's always on the hunt for upcoming artists and new music — made RapStock.io, a platform to bet on upcoming rappers. When talking about the project to recruiters, it was effortless for me to come across as passionate about data science and pricing algorithms because the underlying passion for hip hop music was shining through.

Another benefit of working on a project related to your passion: it's less of a chore to get the damn project over the finish line when work becomes play. And getting the project done is paramount to your success, as we later detail in tip #5.

Tip #4: Work with Interesting Datasets

Don't work with datasets that people have worked with in their school work, such as the classic Iris Plant dataset or the Kaggle passenger survival classification project using the Titanic dataset — they're overdone. Worse, working on these datasets likely means you are not working on something you are passionate about, which goes against the advice in the previous tip. I stand corrected, though, if you're a weirdo whose true passion is classifying flowers into their respective species based on petal and sepal lengths.

Another reason to not work with these standard datasets is because the recruiter and hiring manager will have seen this project done multiple times already. This situation occurs frequently if you are a college student or in a bootcamp, and are trying to pitch a required class project as your portfolio project. You can imagine how lame that comes across to recruiters during university recruiting events to see the same project over and over again from everyone who took the same class.

Kevin heard a recruiter complain about exactly this after she saw one too many Convolutional Neural Nets trained with Tensorflow for handwriting recognition based on the MNIST Digit Recognition Dataset. So stay away from this classic dataset, along with avoiding stock ticker data (unless you are gunning for Wall Street jobs) and the Twitter firehose data (unless you truly have a fresh take on analyzing tweets).

To drive home how you can seek out interesting datasets that tell a good story and relate to your passion, let's work with a concrete example. Suppose you love space and dream of working as a data scientist at NASA. How would you find exciting datasets to help you break into your dream job?

Our first step would be to go on Kaggle and find NASA's Asteroid Classification challenge. Or we can analyze the Kepler Space Observatory Exoplanet Search dataset. If we wanted to start more simply and only knew Excel, we could look at the CSV of all 357 astronauts and their backgrounds and make a few cool graphics about what their most common undergrad majors in college were. So much data is out there just one Google Search away — you have no excuse to be working on something boring!

The upside of using a website like Kaggle to get datasets is that it's well-formatted, clean, and often there are starter notebooks exploring the data. The downside is that others may also be looking at it, hurting your project's uniqueness. There are also some lost learning opportunities, since collecting and cleaning data is a big part of a data scientist's work. However, if you find something you really love on Kaggle, it's not a big problem. Go for it, and maybe later find a different, complementary dataset to add another dimension to your project.

One way to tackle interesting datasets that are unique is to scrape the data yourself. Packages like BeautifulSoup and Scrapy in Python can help, or Rvest for R users. Plus it's an excellent way to practice your programming skills and also show how scrappy you are. And since collecting and cleaning data is such a large part of a real data scientist's workflow, scraping your own dataset and cleaning it up shows a hiring manager you're familiar with the whole life cycle of a data science project.

Tip #5: Done > Perfect. Prove You Are Done.

As long as your work is approximately correct, the actual technical details don't matter as much for getting an interview. Again, as mentioned above, a recruiter will not dig into your project and notice that you didn't remove some outliers from the data. However, a recruiter can quickly determine how complete a project is! So make sure you go the extra mile in "wrapping up a project." See if you can "productionize" the project.

Turn the data science analysis into a product. For example, if your project was training a classifier to predict age from a picture of a face, go the extra step and stand up a web app that allows anyone to upload a photo and predict their own age. As part two to the project, use a neural net to transform the person's face to a different age, similar to FaceApp. Putting in this extra work, and then cold-emailing the project to hiring managers, could be your ticket into companies like Snapchat, Instagram, and TikTok.

If your project was less productizable and more exploratory, see if you can make an interactive visualization that helps you tell a story and share your results. For example, let's say you did an exploratory data analysis on the relationship between median neighborhood income and quality of school district. To wrap this project up, try to make and host an interactive map visualization so that folks can explore and visualize the data for themselves. I like D3.js for interactive charts and Leaflet.js for interactive maps. rCharts is also pretty cool for R users. By creating a visualization, and then sending this completed interactive map to hiring managers at Zillow or Opendoor, you'll be able to stand out from other candidates.

Lastly, your portfolio project isn't done until it's public, so make sure you publicly share your code on GitHub. You can also use Google Collab to host and share your interactive data analysis notebook. Even if no one sees the code or runs the notebook (which is likely!), just having a link to it sends a signal that you are proud enough of your work to publish it openly. It also shows that you actually did what you said you did and didn't just fabricate something to pad the resume.

Tip #6: Demonstrate Business Value

The best portfolio projects are able to demonstrate business value. Try to make it concrete, not theoretical. This advice is crucial for PhDs breaking into industry, especially if the academic is trying to break into smaller companies or startups. When you are applying to businesses, talk in business terms. Show how your technical skills can drive business value. Try to make sure your project has a crisp business recommendation or fascinating takeaway.

If you can't point to exact metrics like dollars earned or time saved by creating the project, you can instead put down usage numbers as a proxy for the amount of value you created for people. Plus, mentioning view counts or downloads or active users helps demonstrate to a business that you drove a project to completion. It's okay to skip out on demonstrating business value IF you work with interesting enough data and can tell a good story. An example project we find interesting, creative and fun, but technically simple and not obviously a driver of real business value: A Highly Scientific Analysis of Chinese Restaurant Names. Send that project to recruiters at Yelp or DoorDash and watch the interviews come pouring in.

Double Down on One Portfolio Project to Breeze Through Your Behavioral Interview

If the tips above seem like a lot of work, that's because they are. But the good news is you don't need to do this for every project. Spending all your energy on having a single killer project is worth it, as long as you end up with a lot to show for it, like beautiful graphics, a working demo, and some usage statistics or metrics on business value created.

The other reason it's okay to focus your time and energy on creating a single kick-ass project is because this can help carry you through a behavioral interview. Typical behavioral interview questions start with "Tell me about a time..." or "Tell me about a project where...." By coming back to the same project, you don't have to waste valuable time setting context or background. Instead you can dive straight into answering the behavioral interview question. Plus, focusing your time and energy into one project means you'll be able to tackle more challenging problems and apply more advanced techniques. This is good because common behavioral interview questions include, "What was the hardest data-related problem you tackled?"

An additional benefit to going deep into one project is that you're much more likely to create real business value or gather users and views for your project. Trying to market and promote multiple side projects is a recipe for disaster, because showing traction for a single project is hard enough for most people. Sticking to one project makes it much more likely you'll discover some method or angle to generate value and publicity.

Another reason why a well-crafted portfolio project allows you to breeze through your behavioral interview is because one common question is, "Why this company?" or "Why this industry?" Hopefully, you worked on something you are passionate about that is related to the company or industry you are interviewing with. Now your project is able to "show, not tell" your interest.

One example of using a project to "show, not tell" was when I (Nick) applied to Facebook's Growth team. They asked a common question: "Why Facebook's Growth team?"

I was able to tell the story of creating consumer products and being a DJ, which led me to create a music-tech startup called RapStock.io. From RapStock.io I found my love of growing consumer demand through engineering. This project sparked my interest in combining Software Engineering, Data & Experimentation Design, and creating consumer products. This trajectory mapped exactly to what Facebook's Growth team did all day, so I'd like to think I gave the perfect answer backed up by an authentic story.

You might be thinking, "Nick, you got lucky working on a consumer tech startup with a focus on growth that lined up beautifully with what Facebook's Growth team does." But dear reader, here's a little secret:

- *When I talked to fintech companies, I told them about the stock market and commodity pricing aspect of my game.*
- *When I talked to data companies, I went deeper into the algorithm that assigned prices to rappers from Spotify data.*
- *When I talked to marketing and ad-tech companies, I talked about how this tech project piqued my interest in the world of advertising after a failed Google ads campaign.*
- *When I interviewed with startups, I talked about how I, too, was a startup founder in the past, and wanted to move fast and ship things quickly rather than suffer through big company bureaucracy.*

With just one project, I was able to "show, not tell" my direct interest in a variety of companies and types of work, while showcasing my technical expertise and personality at the same time. A kick-ass portfolio project, along with the more detailed behavioral interview tips we present in Chapter 4, will allow you to do the same.

Cold Email Your Way to Your Dream Job in Data

CHAPTER 3

You've crafted the perfect resume and made a kick-ass portfolio project, which means it's time to apply to an open data science position. Eagerly, you go to an online job portal and submit your resume, and maybe even a cover letter. And then it's crickets. Not even an automated rejection letter.

If you've applied online and then been effectively ghosted, you're not alone. Kevin and I have been in the same situation plenty of times. We are all too familiar with the black hole effect of online job applications, where it almost feels like you're tossing your resume into the void.

So how do you reliably land interviews, especially if you have no connections or referrals? Two words: Cold. Emails.

While in college, Snapchat and Cloudflare interviewed me (Nick) when I had no connections or referrals at those companies. I got these interviews by writing an email, out of the blue, to the company's recruiters. This process is known as cold emailing (in contrast to getting a warm introduction to a recruiter). Even my previous job at data startup SafeGraph is the result of a cold email that I sent to the CEO. We firmly believe this tactic can be a game changer on the data science job hunt.

We don't want to over-promise, though. The best written cold email won't help if you're pursuing jobs you aren't a good fit for, like a new grad applying to be a VP of Data Science. Plus, you need to

have a strong resume (Chapter 1) and strong portfolio projects (Chapter 2). But if you've got your ducks in a row, yet struggle to land the first interview, this chapter will be a game changer.

Who are we even cold emailing?

Before we talk about the content of the cold email, let's cover who we're reaching out to in the first place.

At smaller companies with less than 50 employees, emailing the CEO or CTO works very well. At mid-range companies (from between 50 and 250 people), see if there is a technical recruiter to email; otherwise just a normal recruiter should do. Another option is emailing the hiring manager for the team you want to join.

For larger companies, finding the right person can be trickier. If you are looking for internships or are a new grad, many of the larger companies (1,000+ employees) have a person titled "University Recruiter" or "Campus Recruiter." Reaching out to these recruiters is how I (Nick) had the most luck when cold emailing in college.

At very large companies like FANG, there should also be dedicated recruiters only working with data scientists. To find these recruiters, go to the company's LinkedIn page and hit "employees." Then, filter the search results by title and search for "Data Recruiter." When doing this at Google, I found six relevant data science recruiters to reach out to.

Another option is to just filter the search by "recruiter." You'll get hundreds of results that you can sift through manually. Doing so at Google uncovered an "ML recruiter," "PhD (Data Science) Recruiter," and a "Lead Recruiter, Databases & Data Analytics (GCP)," all in just a few minutes.

Another good source of people to email at a company is alumni from your school who work there. Even if they work in a non-data science role, they may be able to refer you or know the right person to connect you with. To find these people, search your university on LinkedIn and click "alumni." From there, you can filter the alumni profiles based on what companies they work at or what titles they hold. I resort to this tactic if my first few cold emails to hiring managers and recruiters go unanswered.

How do we find their email address?

Now that we know who we want to email, how do we find their email address?

You can use a free tool like Clearbit Connect or Hunter.io to look this up. If you can't find the person you want with an email-lookup tool, you can always guess. At smaller companies, or when dealing with the founders, a good guess is `firstname@companydomain.com`. For example, Jeff Bezos's real email is `jeff@amazon.com`.

At mid-sized companies, `firstname.lastname@companydomain.com` may work well, along with using their first initial and last name with no spaces. When you're in doubt, you can use Hunter.io or Clearbit Connect to see the format that others at the company use, and then you can make an educated guess. Then put your best guess email address in the "to" section of the email, and cc a few more email guesses in the hopes that one of the emails will be on target.

To shortcut all this work, you can also use MassApply (massapply.com), which has hundreds of technical jobs with the recruiter contact information already available inside the platform. It allows you to send customized cold emails in a single click to recruiters, as well as to track your job applications. We're huge fans, but are a bit biased, because Nick's brother founded MassApply!

Can't Find an Email? Every address you guessed bounced?

Every address you guessed bounced? Send them a LinkedIn InMail or Twitter DM instead!

While email should still be your first choice (busy people like recruiters and CEOs are in their email inbox all day — not on LinkedIn or Twitter), you have nothing to lose when reaching out on other platforms! The tips we soon cover on writing effective cold emails apply to other types of cold messages as well.

8 Tips for Writing Effective Cold Emails

Now that we know who to email and how to get their email, what do we actually say in the email?

Here are 8 quick tips for writing effective emails.

Tip #1: Keep the Email Short

Recruiters are busy people, as we covered in depth in Chapter 1 on resumes. Just like with your resume, they don't have a lot of time to read your email. You've got 10 seconds to impress them with your email so that they respond to it rather than ignore it. So keep your email short.

The data backs email brevity. HubSpot analyzed 40 million emails and found the ideal length of a cold sales email is between 50 and 125 words to maximize response rates. I've personally had the best luck at around 100 words. It's all about maintaining a high signal-to-noise ratio. You don't need to include phrases like "I hope you are doing well today!" or "I hope this email finds you well." At best, it's extraneous and at worst, insincere.

Tip #2: Mention an Accomplishment or Two

Our cold email is just like a sales email. In one paragraph, you are trying to sell yourself to the recruiter as someone worthy of an interview. This is sales — don't be shy.

Highlight a relevant accomplishment or internship experience that makes you worthy of a response. Name-drop that hackathon you won. Hyperlink to your favorite project or an app on the app store with a few thousand downloads and mention that usage number. If you went to an impressive engineering school, lean into that.

However, you don't need to link to too many things or copy-paste the entire resume. That would end up breaking Principle #1: Keep It Short. Instead, attach your resume to the initial email so the recruiter can get more background if needed.

Tip #3: Add Urgency and Establish a Timeline

My favorite tip: if you already have a return internship offer with a different company or a competing job offer extended to you, mention that. It puts pressure on a recruiter to respond promptly and might even fast-track you to an onsite interview. This tactic works especially well if it's an offer with a well-known company.

Even if the deadline is very far from now, so there is no true urgency, name-dropping the other company is helpful as social proof. If other companies desire you, then a recruiter is more likely to feel you are valuable and have #fomo. This leads to them responding to you.

You don't even need the offer in hand to make this tactic work! Just having an onsite interview scheduled with a top company helps other companies realize you've got something worthwhile and that there is a specific timeline to adhere to.

Former Microsoft Intern With Upcoming Deadline Interested in Uber ATG  

Inbox x

Nick Singh <hello@nicksingh.com> 9:18 AM (0 minutes ago)   

to recruiter ▾

Hello Recruiter Name,

I have an upcoming onsite-interview with Microsoft's Azure ML team next month, but wanted to also interview with Uber because self-driving cars is where I believe Computer Vision will help improve the world the most in the next decade.

Helping the world through CV became my passion after seeing the impact of the last project I made, which used CV to find and categorize skin diseases.

From reading the ATG engineering blogs, I know Uber is the best place for a passionate computer vision engineer to make an impact, and am eager to start the interview process before I go too far down the process with Microsoft.

I've attached my resume.
Thanks,
Nick Singh

One warning: be careful not to make it seem like the company you are talking to is the backup option. To do this, make sure you convey enthusiasm for the company and mission. Below is an example of that.

Tip #4: Relate Personally to the Recruiter or Company

Yes, this is a cold email, but you don't have to be so cold! The person at the other end of the email is still a human, and you can make a real-life connection even if you haven't met before. It's well worth it to do your research. For example, see if you have a mutual connection with the recruiter. Use LinkedIn to see if you have any commonalities like education or cities you've both lived in. Even two minutes of sleuthing on the internet looking for a commonality can pay huge dividends when it comes to response rates.

Tip #5: Have a Specific Ask

Be up front with what you want. A vague email hoping to "set up a time to chat" or "learn more about the interview process" is too meek and indirect. The recruiter knows that between your friends, Google Search, Quora, and Glassdoor, you can find any information you need about a company and the interview process. They undoubtedly know you are angling for a job or internship but are too shy to ask directly.

So why not be bold --- after all, fortune favors the bold -- and always include a specific ask:

"I'd like to interview for a Data Science Internship for Summer 2021."

"I'd like to start the interview process for the Senior Data Scientist position at your company."

Tip #6: Have a Strong Email Subject Line

An email is only read if it's opened. Without a strong subject line to lure the recipient into actually opening the email, the email is wasted. Thus, it's worth spending time crafting a strong email subject line. To make the subject line click-worthy, it's key to include your most noteworthy and relevant

details. It's okay if the subject line is keyword driven and a "big flex," as the teens say these days. Borrow from BuzzFeed clickbait titles — they actually work! I wouldn't go so far as to say "21 Weird Facts That'll Leave You DYING to Hire This Data Scientist," but you get the gist.

What I (Nick) used, for example: "*Former Google & Microsoft Intern Interested in FT @ X*"

This subject line works because I lead immediately with my background, which is click-worthy since Google and Microsoft are well-known companies, and I have my specific ask (for full-time software jobs) included in the subject line. Some other subject line examples that are short and to the point if you can't rely on internship experience at name-brand companies:

"Computer Vision Ph.D. Interested In Waymo"

"Princeton Math Major Interested in Quant @ Goldman Sachs"

"Kaggle Champion Interested in Airbnb DS"

"UMich Junior & Past GE Intern Seeking Ford Data Science FT"

If I (Nick) found a recruiter from my alma mater (UVA), I'd be sure to include that in the subject line to show that it's personalized. For reaching out to UVA alumni, I'd thrown in a "Wahoowa" (similarly, a "Go Bears" or "Roll Tide" if you went to Berkeley or Alabama, respectively). Including the name of the recruiter should also increase the click-through rate.

Example: "*Dan | FinTech Hackathon Winner And Wahoo Interested in Robinhood*"

Another hack: including "Re:" in the subject line to make it look like they've already engaged in conversation with you.

Tip #7: Follow Up 3 Times

A perfectly written email sent only once may not work. You should follow up at least three times. You can reply directly to the thread, so that the context from the first email still remains there.

Don't worry about feeling too pushy — it's standard in sales to reach out 3+ times. I know first hand not to give up too early: some of the cold emails that turned into interviews only got responses after the third email. Send the first follow-up after 3–4 days, and send the second follow-up 4–5 days later. Don't think putting in a 2-week delay will make you come across as more polite.

A free Gmail plugin like Boomerang, which will flag when an email hasn't been responded to in some time, can help to keep yourself accountable. There is also automatic email scheduling within MassApply so that you can follow up three times.

If after a few emails you don't get a response, reach out to another recruiter at the same company. It's okay to reach out to multiple people at a company that you want to work for. Trust me, it's not a weird thing to do. In enterprise sales lingo, reaching out to multiple people at your target company is called being "multi-threaded into an account," and it's a time-tested tactic.

Tip #8: Send the Email at the Right Time

We've all been guilty of getting an email, reading it, and waiting till later to respond to it. And then "later" never comes. That's why Principle #7 — following up three times — works so damn well.

But sending an email out at the right time can save you from having to bump up emails. To maximize your reply rate, send the email when you think the reader is most likely to be free and in the mood to respond. That means no weekend emails. No emails on holidays or days people typically might take a long weekend. Figure out the time zone for the recruiter, and be sure to not send it after business hours.

I had the best luck emailing Silicon Valley recruiters at ~11:00 A.M. or 2 P.M. P.S.T. The psychology behind this is we've all felt ourselves counting down the minutes to lunch, aimlessly refreshing your email and Slack to pass the time. That's a great time to catch someone. Same with the after lunch lull.

The best days I found to send emails were Tuesday through Thursday. I avoided Mondays since that's the day many people have 1:1s or team meetings or have work they are catching up on from the weekend. On Fridays, many people might be on PTO, or even if they are in office, have some other kind of event like happy hour in the afternoon (or they've already mentally checked out before the weekend).

3 Successful Cold Email Examples

Here are some real cold emails I've sent in the job hunt. The text is exactly the same as what I sent, but I just re-created it to protect the recipient's name and email address.

These emails aren't perfect by any means, but generally follow the eight tips I've laid out above. Just remember: even sending a cold email that's bad puts you in the top decile of job seekers. Most people will never make an effort to personally write an email to someone they don't know, and then have the tenacity to follow up a few times. It's precisely why cold email works so well!

The 4 Sequence Email Drip to Periscope Data

Intro Email:

Ex-Google & Microsoft Intern Interested in Working FT at Periscope Data

Nick Singh <hello@nicksingh.com> to recruiter ▾

Tue, Sep 8, 2020, 2:43 PM

☆ ↶ :

Hi X,

Found your email on Hacker news. I'm a former Software Engineering Intern @ Google's Nest Labs and Microsoft who will be graduating from college May'17.

I'm interested in working full time at Periscope Data because of my interest in data engineering (spent the summer on the Data Infrastructure team @ Nest) and my interest in turning data into insights (built dashboards to do just that the past two summers).

How can I start the interview process?

Best,

N

My Two Follow-Ups

Nick Singh 9:32 AM (1 minute ago)

to recruiter ▾

Just wanted to follow up with you about full-time roles at Periscope data. I believe my interest in data engineering, along with past experience building dashboards and visualization tools, makes me a good fit.

...

Nick Singh 9:33 AM (1 minute ago)

to recruiter ▾

Hi X, Wanted to circle back on this. What do next steps look like?

The Hail Mary:

I send this email when I have on-site interviews near the target company planned, or when I have offer deadlines approaching. This email is often sent weeks after the initial outreach. It works because it adds an element of urgency to the recruiter, and it gives social proof that other companies have vetted me enough to bring me on-site.

Nick Singh <hello@nicksingh.com> Tue, Sep 8, 2020, 2:47 PM

to recruiter ▾

Hi X,

Just wanted to follow up with you regarding opportunities with Periscope Data. I will be in the bay area doing interviews with Facebook and Uber next week. Would love a chance to do a phone interview with Periscope Data this week to assess technical fit. If we are a good match, I'd be happy to swing by the office the following week for technical interviews while I am already in town.

Thanks,
Nick Singh

More Examples of Real Cold Emails I've Sent

Cold Email to Airbnb in 2016

Former Google Intern from UVA Interested in Airbnb  

Inbox x

Nick Singh <hello@nicksingh.com> 9:42 AM (0 minutes ago)   
to recruiter ▾

Hello X,

We met briefly at the UVA in SF mixer this past summer.

I just wanted to reach out to you about new grad Software Engineering positions @ Airbnb. My friend, Y, interned at Airbnb on the Infrastructure team and really loved their experience.

This past summer, I was on the Data Infrastructure team at Google's Nest Labs. From talking to Y, I think I can be a good fit for similar teams at Airbnb. Let me know what the next steps are.

Thanks,
Nick Singh

Cold Email to Reddit in 2015

Former Microsoft Intern @ Avid Redditor Interested in SWE Internship  

Inbox x

Nick Singh <hello@nicksingh.com> 9:47 AM (0 minutes ago)   
to recruiter ▾

Hello X,

I saw your post on Hacker News and wanted to reach out regarding why I'm a good fit to be a Software Engineering intern at Reddit for Summer 2016.

I interned at Microsoft this past summer on the Payments Team where I helped the team turn data into insight to diagnose payment issues faster.

In my free time (when I'm not on Reddit) I built RapStock.io which I grew to 2000 users. 1400 out of the 2000 users came from Reddit when we went viral so I have a soft spot for the community and product.

Let me know what next steps I should take.

Cold Email to SafeGraph in 2018 (how I got my last job!)

Below is the screenshot of the exact email I sent in summer of 2018 to SafeGraph CEO, Auren Hoffman. I decided to email the CEO directly, in addition to my AngelList application, due to my poor track record of hearing back from online job portals. Within 24 hours of sending this email, I had an interview booked with Auren and ended up working at SafeGraph for close to two years. That's the power of cold email!

By utilizing these cold email tips and taking inspiration from these cold-email examples, in conjunction with a strong resume and kick-ass portfolio projects, you're well on your way to landing more data science interviews. Now comes the next challenge of the data science job hunt: acing the behavioral interview.

The screenshot shows an email interface with the following details:

From: Nipun Singh <ns2se@virginia.edu>
to auren, auren.hoffman ▾

Date: Sun, Jul 15, 2018, 2:29 AM **Star** **Reply** **⋮**

Message Content:

Auren,

I'm super interested in the CoS role at Safe Graph. I applied on Angel list but figured I'd also shoot you an email.

I'm a good fit for the role because

- I'm currently a Software Engineer on Facebook's Growth team. I'm data-driven to the max - all day either coding or cutting data to understand the impact of what I coded and the A/B tests I ran.
- I'm a hustler. I ran a startup in college, which I grew to 2,000 MAU. I ran a DJ business in highschool, which taught me how to be a people person and also sell. I helped run the Venture Capital club (Virginia Venture Fund) and Entrepreneurship Group (HackCville) at my college.
- I studied Systems Engineering in college, which is super similar to Industrial Engineering / OR (your major!). I also studied Computer Science. I've taking classes on ML, Computer Vision, Stochastic Processes, Databases. I'd be able to understand the technical details of SafeGraph and the space we operate in very quickly.

I've attached my resume. I'd love to call or meet up in person to talk more about why I'm excited about SafeGraph.

Thanks,

Nipun Singh
www.nipunsingh.com

Ace the Behavioral Interview

CHAPTER 4

*Now that you've **finally** built a kick-ass resume, compiled an impressive project portfolio and intrigued the HR department at your dream company enough to call you in for an interview based on your strategically written emails, you're ready to ace the technical data science interview questions and land the job. But there's one more piece to the puzzle whose importance is usually underestimated: the behavioral interview. While it's true that 90% of the reason candidates pass interviews for the most coveted big tech and finance jobs is because of their technical skills — their ability to code on the spot, write SQL queries, and answer conceptual statistics questions — neglecting the other 10%, which stems from the behavioral interview, can be a huge mistake.*

Behavioral Interviews Aren't Fluffy B.S.

You may not agree that the behavioral questions are important. You might think this behavioral interview stuff is fluffy bullshit, and that you can simply wing it. Sure, in some companies, as long as you aren't an asshole in the interview and have the technical skills, you'll be hired. But some companies take this very seriously. Amazon, for example, interviews every single candidate on the company leadership principles like "customer obsession" and "invent and simplify" in their bar raiser interviews. Uber also includes a bar-raiser round which focuses on behavioral questions about culture fit, previous work experience, and past projects. If you want to work there, you've got to take the behavioral interviews seriously.

Even small companies have their unique twist on this. Consider this: my (Nick's) former employer, SafeGraph, displayed the company values on a poster hung in every room in the building. Even the

bathroom. While you’re pissing, the company values are in your face. No joke. *It's that paramount.* So imagine if you came to an interview with *me* and didn’t share any stories that exhibited SafeGraph’s company values. You’d have a pretty piss-poor chance of passing the interview!

When Do Behavioral Interviews Happen?

You might be wondering, “*I had four interviews for a position, and not one of them was called a behavioral interview.*”

The behavioral interview is an integral part of any interview and consists of questions that help the interviewer assess candidates based on actual experiences in prior jobs or situations. Even the “friendly chat” time at the start of the interview can essentially be a behavioral interview.

So while you might not have an explicit calendar invite for “Behavioral Interview,” don’t be fooled: behavioral interviews occur *all the damn time*. That casual, icebreaker of a question, “So, tell me about yourself...” — that’s *an interview question!* For every job, and at practically every round, there will be a behavioral component, whether it’s explicit or not. Behavioral interview questions can happen:

- With a recruiter before getting to technical rounds. In which case you might not even get to the technical interview rounds...
- During your technical interviews, where the first 5-10 minutes are usually carved out for a casual chat about your past projects and your interest in the company.
- During lunch, to understand how you behave outside of the interview setting.
- At the end of the on-site interview; they know you can do the work, but are you someone they want to personally work with? You’ll meet with your future boss, and maybe even their boss, where they’ll both try to sell you on the company, but also see if you’d be a good culture fit.

The reality is, *you are constantly being assessed!* That’s why, on the basis of frequency alone, preparing for and practicing answers to these questions is well worth the effort.

Ace the Behavioral Interview to Beat the Odds

Acing the behavioral interview can be the X-factor — the thing that separates you out from the horde of other applicants. You don’t want to be lying in bed at night, wide awake, thinking, “Damn it. I forgot to tell them about that time I caught that data analysis mistake and saved the company \$50,000!” A little prep work for your interview can mean the difference between a strikeout and a home run.

Focusing on behavioral interviews is especially important if you’re new to the data science game. When a company makes an investment in junior talent, they are looking at 3-6 months of training before that junior data person becomes truly productive. You probably won’t give the best answers on technical questions, and there will always be more senior candidates in the pipeline, but the coachability, enthusiasm, and eagerness to learn that you show in the behavioral interview could be what convinces a company to take a chance and invest in you.

3 Things Behavioral Interviews Test For

You know that behavioral interviews are important, that they happen all the time, and that the stakes are high especially for junior talent. Now you might be wondering, “What are interviewers even

looking for?" Behavioral questions have to do with...well...behavior. There are three basic kinds of things an employer tests for:

- **Soft skills:** How well do you communicate? So much of effective data science is dealing with stakeholders — would you be able to articulate your proposals to them, or sell your ideas convincingly enough to get buy-in from them? How well do you work with others? Data science is a team sport, after all! How do you deal with setbacks and failures? Do you get defensive, or exhibit a growth mindset?
- **Position fit:** How interested are you in the job and team you're gunning for? What motivates you about the position — only the paycheck or passion as well?
- **Culture fit:** How well do you fit the team and company's culture? Can you get behind the company's mission and values? Basically the corporate version of a "vibe check"!

Essentially, while technical interviews are about whether you can do the job, behavioral interviews are about whether you want to do the job, and if you are someone others will want to work with. Fortunately, you can have the charisma of Sheldon Cooper from *Big Bang Theory* and still pass behavioral interviews — if you prepare for the most common behavioral questions asked in data science interviews.

Tell Me About Yourself: The #1 Behavioral Interview Question

"Tell me about yourself" may seem like a simple icebreaker to ease tension and get the interview rolling, but it's actually the #1 most asked behavioral interview question! If you are not properly prepared with your answer, you can stumble through it blithely, telling your life history and all sorts of irrelevant details that are *not* what they want to know about you. First impressions matter, and a well-thought-out answer can impress the hell out of your interviewer and put you in the running from the get-go.

So, how do you prepare an awesome answer to this seemingly innocuous question?

- Limit your answer to a minute or two; don't ramble! As such, start your story at a strategically relevant point (which is often college for most early-career folks).
- Relate your story to the position and company at hand. See if you can weave your pitch with key terms from the job description and company values. Speak their language!
- Mention a big accomplishment or two; even though they've seen your resume, don't let them forget about your biggest selling point!
- Rehearse. You know this question will be asked at the start of every interview.

Your answer should include these three key points:

- 1) Who you are
- 2) How you came to be where you are today (sprinkle in your achievements here)
- 3) What you're into/looking for now (*hint hint: it's basically this role + this company*)

To make this more concrete, here's the "about me" pitch we authors used on the job hunt.

Kevin's Wall Street "About Me" Pitch

Hi, I'm Kevin, currently a data scientist at Facebook. I graduated from Penn in 2017, studying computer science, statistics, and finance. At Facebook I focused on analytics within the groups team, making sure Facebook Groups is free of spam and hate speech. Before Facebook, I briefly interned at a hedge fund, working on looking at alternative data sets, like clickstream data and satellite imagery, to analyze stocks. Having worked in both big tech and Wall Street, I've come to realize I'm more passionate about applying data science in financial markets because of the fast-paced nature and high stakes environment. I was drawn to your fund in particular due to the small team, high autonomy, and chance to be part of a more greenfield data science effort.

Nick's Google Nest Data Infrastructure Internship "About Me" Pitch

Hi! I'm Nick, and I'm currently a 3rd year student at the University of Virginia! I love the intersection between software and data, which is why I'm studying Systems Engineering and Computer Science at UVA. It's also why two summers ago, I interned as a Data Scientist at a defense contractor, and last summer I interned on the Payments team at Microsoft doing back-end work.

I'm super excited to potentially work on the Data Infrastructure team at Nest Labs since it's the perfect blend of my past data and SWE experience. Plus, Nest's intelligent home automation products rely on great data and machine learning, and I want to work at a company where data is at the forefront. Lastly, I love how you all are a smaller, faster-paced division within Google. Having made a startup in the past, which I growth hacked to 2,000 users in just a few months, I love the "move fast" attitude of smaller companies. I think that Nest being an autonomous company within Google strikes the perfect blend between startup and big tech company, and it's why I'm so excited by this team and company.

Why did you choose Data Science?

Here's another question you might be asked, which is closely related to your personal pitch: "Why did you choose Data Science?" Likely your answer to "tell me about yourself" contains some element of how you got into the field, but you may be asked to harp on this point more, especially if you're an industry switcher, or come from an untraditional background.

If your path isn't the most straightforward, don't be nervous — capitalize on this opportunity to show you are a go-getter who decided to make a career change, a fast learner who has accomplished so much in a short time, and how your passion for the field is genuine! This is also a great opportunity to talk about how your skills from prior jobs and industry experience naturally led you into data science. Remember, data science is much more than modeling — even if you weren't throwing XGBoost at random datasets in your last job, there must have been some relevant data science-adjacent skills you acquired. And deep down, internalize that your newness to the field isn't a weakness, but a strength — you've probably got extra subject matter expertise and a fresh perspective!

Tell Me About a Time: The #1 Most Common Pattern for Questions

Once you have your opening pitch prepared, along with the story of how you got into the field, it's time to focus on the other questions most likely to be asked.

The #1 question after "tell me about yourself" is: "Tell me about a situation where" something happened. Note that this question can be phrased in various ways: "Give me an example of when you

X” and “Describe a situation when you Y.” This is your time to share war stories from past jobs. If you lack work experience, this is the time for your independent portfolio projects to shine.

Most Common “Tell Me About a Time” Questions

Some of the most commonly asked “tell me about a time” questions are:

Tell me about a time...

- you dealt with a setback — how did you handle it?
- you had to deal with a particularly difficult co-worker --- how did you manage it?
- you made a decision that wasn’t popular — how did you go about implementing it?
- you accomplished something in your career that made you very proud — why was that moment meaningful to you?
- you missed a big deadline — how did you handle it?

“Tell Me About a Time” for Data Scientists

While the above popular questions are fair game, you might also be asked a twist on these questions so that they’re better geared towards data scientists, analysts, and machine learning engineers. Below are some more data-driven behavioral interview questions.

Tell me about a time...

- *when data helped drive a business decision.*
- *where the results of your analysis were much different than what you would have expected. Why was that? What did you do?*
- *when you had to make a decision BUT the data you needed wasn’t available.*
- *you had an interesting hypothesis — how did you validate it?*
- *when you disagreed with a PM or engineer.*

Now that we’ve got the laundry list of situational questions out of the way, how do you answer these questions well, on the spot?

A superSTAR Answer

The trick to answering the behavioral questions we listed earlier on the spot is...well...to NOT answer them on the spot! A lot of preparation needs to go into this so you can give effortless off-the-cuff answers come interview time. Your first step in preparing flawless answers is to prepare stories that address the questions we mentioned earlier. But don’t prepare *factual* answers.

Prepare stories.

“But I’m no storyteller, I’m a data scientist! How am I supposed to “weave a fascinating tale” about something as mundane as my work history?”

Luckily, there is a simple formula you can use as a framework to structure your story. It’s easy to remember, too. Just remember that a great story will make you a STAR, so you have to use the STAR formula:

- **Situation** --- Describe a specific challenge (problem or opportunity) you or your team, your company, or your customers encountered.
- **Task** — Describe the goal you needed to accomplish (the *project* or *task*).

- **Action** — Describe your role in the project or task using first person (not what your *team* did, but what *you* did).
- **Result** — Describe what happened as a result of your actions. What did you learn or accomplish? Keep in mind that not all outcomes need be positive; if things didn't go your way, explain what lesson you learned (for example, "I learned about the importance of transparency and clear communication"). Showing that you can handle failure and learn from it is a great trait!

Write your stories out using the STAR formula. Where possible, weave into your narrative key phrases from the job description and the company culture or values page, so that you hit the position fit and culture fit elements of the interview.

Amazon Data Scientist Interview Example

For a concrete example of STAR, assume I (Nick) am interviewing to be a data scientist on the AWS Product Analytics team. According to the job opening, the role entails influencing the long-term roadmap of the AWS EC2 Product Team. The job description also mentions looking for someone with a startup mentality, since "AWS is a high-growth, fast-moving division." Finally, their preferred qualifications include "demonstrated ability to balance technical and business needs" and "independently drive issues to resolution while communicating insights to nontechnical audiences."

Now that we've set up the role I'm interviewing for, imagine the Amazon bar-raiser hits me with the question: "Tell me about a time you were not satisfied with the status quo."

My answer:

- **Situation:** I challenged the status quo back when I worked on Facebook's Growth Team, specifically on the New User Experience division. Our main goal was to improve new user retention rates. In 2018, there was a company-wide push for Facebook Stories based on the success of Instagram stories and the fear of Snapchat gaining even more market share. The status quo was to prioritize features that would promote the adoption of Facebook Stories, but I had a strong hunch this wasn't good for new users.
- **Task:** My goal was to understand how new users used Facebook Stories, and whether the feature helped or hurt new user retention rates.
- **Action:** For 3 weeks, I sliced and diced data to better understand whether Facebook Stories helped or hurt new users. In the process, I found multiple bugs and user experience gaps related to Stories for new users, which led to decreased retention rates for new users. I fixed the smaller bugs, and presented the bigger data-driven insights into the user experience problems with the wider Facebook Stories team as well as the New Person Experience team.
- **Result:** Fixing the bugs resulted in new user retention rates increasing by X%, and Y% more usage in Facebook Stories. More importantly, by questioning the status quo that Facebook Stories was good for everyone, I made the Facebook Stories team more conscious of gaps in the product as it related to new users. This affected the Facebook Stories product roadmap, and led them to prioritize user onboarding features for their next quarter.

This is an effective answer because it emphasizes how my data-driven work impacted the product roadmap — essentially what this Amazon product analytics job is all about. It also demonstrates my passion for new users, which jives with Amazon's company value of customer obsession.

Remember, though, a winning answer to a behavioral interview question is about more than just words. Project the confidence of a college sophomore who thinks majoring in business means they'll

be a CEO one day. Embody BDE — big data energy. To dial in your delivery, practice telling your stories out loud. Do this in front of a mirror — it'll force you to pay attention to your nonverbal skills, which are also very important in an interview. Use a timer, and without rushing, ensure your answers are under two minutes long.

How to Ace Project Walk-Through Questions

Rather than asking you about a situation, project walk-through questions let you talk about a project in detail. These questions often have follow-ups where they ask for more details — and they may even be a jumping-off point to ask more general technical questions.

In addition to checking your communication skills, like the more traditional behavioral questions, these questions are also testing to see if you've *actually* done what you say you did. The bullet points on your resume don't always tell the whole story — maybe the work is less (or more!) impressive than you made it sound. In fact, with the length limitations on a resume's job description, there's probably a LOT more to the story than the resume reveals.

In project walk-throughs, you might specifically be asked questions such as:

- *How did you collect and clean the data? Did you run into any issues when interpreting the data?*
- *How did you decide what models and techniques to use? What did you eventually try?*
- *How did you evaluate the success of your projects? Was there a baseline to compare against? What metrics did you use to quantify the project's impact?*
- *Did you deploy the final solution? What challenges did you face launching your work?*
- *What tough technical problems did you face — and how did you overcome them?*
- *How did you work with stakeholders and teammates to ensure the project was successful? If there were any conflicts, how did you resolve them?*
- *If you did the project again, what would you do differently?*

If the above questions look familiar, that's because you can consider the project walk-through questions as the inverse of "tell me about a time" questions. Said another way, given a project, the interviewer asks a lot of the same "tell me about a time" questions, except the "time" is all about a certain project. The same concepts for applying STAR still apply!

"Do you have any questions for us?"

Pretty much every interview includes a segment where you get to ask questions. "I don't really have any" is NOT the right answer. So, what *should* you ask when the interviewer says, "Do you have any questions for us?"

There is a right way to answer this! Don't waste this time asking random questions like how much time you get off or how much the job pays! Traditional advice says, "This is the time to interview the company." We disagree! Be strategic about what questions you ask. Have the mindset "until I have the offer in hand, I need to keep showing why I'm a good fit" — you aren't interviewing *them*, you're selling *yourself*! As such, prepare at least three smart, interesting questions per interviewer. *Don't pass on this!* You can ask about salary once you've got the job in the bag. At that point, you are in a far better position to discuss compensation.

As we mention later in Chapter 10: Product Sense, *this* is the time to leverage the company and product research you did. You'll gain much more by asking questions that convey your interest in

the company and what they do. From your readings and product analysis, surely you must be curious about some internal detail, design decision, or what's coming next for some product. This point in the interview is your opportunity not only to have your intellectual curiosity fulfilled, but to impress them with your research and genuine interest in the company and its business.

Another idea is to check out details about your interviewer on LinkedIn. It's not uncommon to know who you'll be interviewing with. Asking a personal question is a sure way to get the interviewer talking about themselves. And people love to do that! If you can tailor questions to their background or projects they've worked on, great! If not, you can ask these sure-fire conversation starters:

- *How did you come into this role or company?*
- *What's the most interesting project you've worked on?*
- *What do you think is the most exciting opportunity for the company or product?*
- *In your opinion, what are the top three challenges facing the business?*
- *What do you think is the hardest part of this role?*
- *How do you see the company values in action during your day-to-day work?*

Going against the grain from traditional career advice, we think asking questions about the role isn't the most beneficial use of this opportunity. Sure, you're not going to get into trouble for asking about the growth trajectory for the role at hand, or what success looks like for the position. It's just that you'll have ample time, and it's a better use of your time to ask these questions *after* you have the job. While you're in the interview mode, again, it's important to either reinforce your interest in the company, their mission and values, or at least have the interviewer talk about themselves. We believe discussing nuances about the role isn't the most productive step to take without an offer at hand.

Post-Interview Etiquette

Whew! Your interview is finally over!

No, it's not!

Send a follow-up thank you note via email a few hours after your interview to keep your name and abilities fresh in their mind. Plus it shows them your interest in the position is deep and sincere. Ideally, you'll mention a few of the specific things you connected with them over during the interview in your email/note. This will help jog their memory as to which interviewee you were and hopefully bring that connection to mind when they see your name again.

The Best Is Yet to Come!

Now that we've walked you through our process for landing more data science interviews and passing the behavioral interview screen, you're *almost* ready for the meat of the book: acing the technical data science interview.

Before we get to the 201 interview problems, we have a quick favor to ask from you. Yes, *you*. If you're enjoying this book, share a photo of your copy of *Ace the Data Science Interview* on LinkedIn and tag us (Nick Singh and Kevin Huo). Feel free to add a quick sentence or two on what's resonated with you so far. We'll both connect with you as well as like and comment on the post. You'll get more LinkedIn profile views, followers, and brownie points from us this way!

Probability

CHAPTER 5

One of the most crucial skills a data scientist needs to have is the ability to think probabilistically. Although probability is a broad field and ranges from theoretical concepts such as measure theory to more practical applications involving various probability distributions, a strong foundation in the core concepts of probability is essential.

In interviews, probability's foundational concepts are heavily tested, particularly conditional probability and basic applications involving PDFs of various probability distributions. In the finance industry, interview questions on probability, including expected values and betting decisions, are especially common. More in-depth problems that build off of these foundational probability topics are common in statistics interview problems, which we cover in the next chapter. For now, we'll start with the basics of probability.

Basics

Conditional Probability

We are often interested in knowing the probability of an event A given that an event B has occurred. For example, what is the probability of a patient having a particular disease, given that the patient tested positive for the disease? This is known as the conditional probability of A given B and is often found in the following form based on **Bayes' rule**:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Under Bayes' rule, $P(A)$ is known as the prior, $P(B|A)$ as the likelihood, and $P(A|B)$ as the posterior.

If this conditional probability is presented simply as $P(A)$ —that is, if $P(A|B) = P(A)$ —then A and B are independent, since knowing about B tells us nothing about the probability of A having also occurred. Similarly, it is possible for A and B to be conditionally independent given the occurrence of another event C : $P(A \cap B|C) = P(A|C)P(B|C)$.

The statement above says that, given that C has occurred, knowing that B has also occurred tells us nothing about the probability of A having occurred.

If other information is available and you are asked to calculate a probability, you should always consider using Bayes' rule. It is an incredibly common interview topic, so understanding its underlying concepts and real-life applications involving it will be extremely helpful. For example, in medical testing for rare diseases, Bayes' rule is especially important, since it is may be misleading to simply diagnose someone as having a disease—even if the test for the disease is considered “very accurate”—without knowing the test’s base rate for accuracy.

Bayes' rule also plays a crucial part in machine learning, where, frequently, the goal is to identify the best conditional distribution for a variable given the data that is available. In an interview, hints will often be given that you need to consider Bayes' rule. One such strong hint is an interviewer's wording in directions to find the probability of some event having occurred “given that” another event has already occurred.

Law of Total Probability

Assume we have several disjoint events within B having occurred; we can then break down the probability of an event A having also occurred thanks to the law of total probability, which is stated as follows: $P(A) = P(A|B_1)P(B_1) + \dots + P(A|B_n)P(B_n)$.

The equation above provides a handy way to think about partitioning events. If we want to model the probability of an event A happening, it can be decomposed into the weighted sum of conditional probabilities based on each possible scenario having occurred. When asked to assess a probability involving a “tree of outcomes” upon which the probability depends, be sure to remember this concept. One common example is the probability that a customer makes a purchase, conditional on which customer segment that customer falls within.

Counting

The concept of counting typically shows up in one form or another in most interviews. Some questions may directly ask about counting (e.g., “How many ways can five people sit around a lunch table?”), while others may ask a similar question, but as a probability (e.g., “What is the likelihood that I draw four cards of the same suit?”).

Two forms of counting elements are generally relevant. If the order of selection of the n items being counted k at a time matters, then the method for counting possible permutations is employed:

$$n * (n - 1) * \dots * (n - k + 1) = \frac{n!}{(n - k)!}$$

In contrast, if order of selection does not matter, then the technique to count possible number of combinations is relevant:

$$\binom{n}{k} = \frac{n!}{k!(n - k)!}$$

Knowing these concepts is necessary in order to assess various probabilities that involve counting procedures. Therefore, remember to determine when selection does versus does not matter.

For some real-life applications of both, consider making up passwords (where order of characters matters) versus choosing restaurants nearby on a map (where order does not matter, only the options). Lastly, both permutations and combinations are frequently encountered in combinatorial and graph theory-related questions.

Random Variables

Random variables are a core topic within probability, and interviewers generally verify that you understand the principles underlying them and have a basic ability to manipulate them. While it is not necessary to memorize all mechanics associated with them or specific use cases, knowing the concepts and their applications is highly recommended.

A random variable is a quantity with an associated probability distribution. It can be either discrete (i.e., have a countable range) or continuous (have an uncountable range). The probability distribution associated with a discrete random variable is a probability mass function (PMF), and that associated with a continuous random variable is a probability density function (PDF). Both can be represented by the following function of x : $f_X(x)$

In the discrete case, X can take on particular values with a particular probability, whereas, in the continuous case, the probability of a particular value of x is not measurable; instead, a “probability mass” per unit per length around x can be measured (imagine the small interval of x and $x + \delta$).

Probabilities of both discrete and continuous random variables must be non-negative and must sum (in the discrete case) or integrate (in the continuous case) to 1:

$$\text{Discrete: } \sum_{x \in X} f_X(x) = 1, \text{ Continuous: } \int_{-\infty}^{\infty} f_X(x) dx = 1$$

The cumulative distribution function (CDF) is often used in practice rather than a variable’s PMF or PDF and is defined as follows in both cases: $F_X(x) = p(X \leq x)$

For a discrete random variable, the CDF is given by a sum: $F_X(x) = \sum_{k \leq x} p(k)$; whereas, for a continuous random variable, the CDF is given by an integral:

$$F_X(x) = \int_{-\infty}^x p(y) dy$$

Thus, the CDF, which is non-negative and monotonically increasing, can be obtained by taking the sums of PMFs for discrete random variables, and the integral of PDFs for continuous random variables.

Knowing the basics of PDFs and CDFs is very useful for deriving properties of random variables, so understanding them is important. Whenever asked about evaluating a random variable, it is essential to identify both the appropriate PDF and CDF at hand.

Joint, Marginal, and Conditional Probability Distributions

Random variables are often analyzed with respect to other random variables, giving rise to *joint PMFs* for discrete random variables and *joint PDFs* for continuous random variables. In the continuous case, for the random variables X and Y varying over a two-dimensional space, the integration of the joint PDF yields the following:

$$\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f_{X,Y}(x,y) dx dy = 1$$

This is useful, since it allows for the calculation of probabilities of events involving X and Y .

From a joint PDF, a marginal PDF can be derived. Here, we derive the marginal PDF for X by integrating out the Y term:

$$f_X(x) = \int_{-\infty}^{\infty} f_{X,Y}(x,y) dy$$

Similarly, we can find a joint CDF where $F_{X,Y}(x,y) = P(X \leq x, Y \leq y)$ is equivalent to the following:

$$F_{X,Y}(x,y) = \int_{-\infty}^x \int_{-\infty}^y f_{X,Y}(u,v) dv du$$

It is also possible to condition PDFs and CDFs on other variables. For example, for random variables X and Y , which are assumed to be jointly distributed, we have the following conditional probability:

$$f_{X|Y}(x|y) = \int_{-\infty}^{\infty} f_Y(y) f_{X|Y}(x|y) dy$$

where X is conditioned on Y . This is an extension of Bayes' rule and works in both the discrete and continuous case, although in the former, summation replaces integration.

Generally, these topics are asked only in very technical rounds, although a basic understanding helps with respect to general derivations of properties. When asked about more than one random variable, make it a point to think in terms of joint distributions.

Probability Distributions

There are many probability distributions, and interviewers generally do not test whether you have memorized specific properties on each (although it is helpful to know the basics), but, rather, to see if you can properly apply them to specific situations. For example, a basic use case would be to assess the probability that a certain event occurs when using a particular distribution, in which case you would directly utilize the distribution's PDF. Below are some overviews of the distributions most commonly included in interviews.

Discrete Probability Distributions

The *binomial distribution* gives the probability of k number of successes in n independent trials, where each trial has probability p of success. Its PMF is

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

and its mean and variance are: $\mu = np$, $\sigma^2 = np(1-p)$.

The most common applications for a binomial distribution are coin flips (the number of heads in n flips), user signups, and any situation involving counting some number of successful events where the outcome of each event is binary.

The *Poisson distribution* gives the probability of the number of events occurring within a particular fixed interval where the known, constant rate of each event's occurrence is λ . The Poisson distribution's PMF is

$$P(X = k) = \frac{e^{-\lambda} \lambda^k}{k!}$$

and its mean and variance are: $\mu = \lambda$, $\sigma^2 = \lambda$.

The most common applications for a Poisson distribution are in assessing counts over a continuous interval, such as the number of visits to a website in a certain period of time or the number of defects in a square foot of fabric. Thus, instead of coin flips with probability p of a head as a use case of the binomial distribution, applications on the Poisson will involve a process X occurring at a rate λ .

Continuous Probability Distributions

The *uniform distribution* assumes a constant probability of an X falling between values on the interval a to b . Its PDF is

$$f(x) = \frac{1}{b-a}$$

and its mean and variance are:

$$\mu = \frac{a+b}{2}, \sigma^2 = \frac{(b-a)^2}{12}$$

The most common applications for a uniform distribution are in sampling (random number generation, for example) and hypothesis testing cases.

The *exponential distribution* gives the probability of the interval length between events of a Poisson process having a set rate parameter of λ . Its PDF is $f(x) = \lambda e^{-\lambda x}$ and its mean and variance are:

$$\mu = \frac{1}{\lambda}, \sigma^2 = \frac{1}{\lambda^2}$$

The most common applications for an exponential distribution are in wait times, such as the time until a customer makes a purchase or the time until a default in credit occurs. One of the distribution's most useful properties, and one that makes for natural questions, is the property of memorylessness of the distribution.

The *normal distribution* distributes probability according to the well-known bell curve over a range of X 's. Given a particular mean and variance, its PDF is

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

and its mean and variance are given by: $\mu = \mu$, $\sigma^2 = \sigma^2$

Many applications involve the normal distribution, largely due to (a) its natural fit to many real-life occurrences, and (b) the Central Limit Theorem (CLT). Therefore, it is very important to remember the normal distribution's PDF.

Markov Chains

A Markov chain is a process in which there is a finite set of states, and the probability of being in a particular state is only dependent on the previous state. Stated another way, the Markov property is such that, given the current state, the past and future states it will occupy are conditionally independent.

The probability of transitioning from state i to state j at any given time is given by a transition matrix, denoted by P :

$$\begin{pmatrix} p_{11} & p_{1n} \\ \dots & \dots \\ p_{m1} & p_{mn} \end{pmatrix}$$

Various characterizations are used to describe states. A *recurrent state* is one whereby, if entering that state, one will always transition back into that state eventually. In contrast, a transient state is one in which, if entered, there is a positive probability that upon leaving, one will never enter that state again.

A stationary distribution for a Markov chain satisfies the following characteristic: $\pi = \pi P$, where P is a transition matrix, and remains fixed following any transitions using P . Thus, P contains the long-run proportions of the time that a process will spend in any particular state over time.

Usual questions asked on this topic involve setting up various problems as Markov chains and answering basic properties concerning Markov chain behavior. For example, you might be asked to model the states of users (new, active, or churned) for a product using a transition matrix and then be asked questions about the chain's long-term behavior. It is generally a good idea to think of Markov chains when multiple states are to be modeled (with transitions between them) or when questioned concerning the long-term behavior of some system.

Probability Interview Questions

Easy

- 5.1. Google: Two teams play a series of games (best of 7 — whoever wins 4 games first) in which each team has a 50% chance of winning any given round (no draws allowed). What is the probability that the series goes to 7 games?
- 5.2. JP Morgan: Say you roll a die three times. What is the probability of getting two sixes in a row?
- 5.3. Uber: You roll three dice, one after another. What is the probability that you obtain three numbers in a strictly increasing order?
- 5.4. Zenefits: Assume you have a deck of 100 cards with values ranging from 1 to 100, and that you draw two cards at random without replacement. What is the probability that the number of one card is precisely double that of the other?
- 5.5. JP Morgan: Imagine you are in a 3D space. From (0,0,0) to (3,3,3), how many paths are there if you can move only up, right, and forward?
- 5.6. Amazon: One in a thousand people have a particular disease, and the test for the disease is 98% correct in testing for the disease. On the other hand, the test has a 1% error rate if the person being tested does not have the disease. If someone tests positive, what are the odds they have the disease?
- 5.7. Facebook: Assume two coins, one fair (having one side heads and one side tails) and the other unfair (having both sides tails). You pick one at random, flip it five times, and observe that it comes up as tails all five times. What is the probability that you are flipping the unfair coin?
- 5.8. Goldman Sachs: Players A and B are playing a game where they take turns flipping a biased coin, with p probability of landing on heads (and winning). Player A starts the game, and then the players pass the coin back and forth until one person flips heads and wins. What is the probability that A wins?
- 5.9. Microsoft: Three friends in Seattle each told you it is rainy, and each person has a 1/3 probability of lying. What is the probability that Seattle is rainy, assuming that the likelihood of rain on any given day is 0.25?
- 5.10. Bloomberg: You draw a circle and choose two chords at random. What is the probability that those chords will intersect?

- 5.11. Morgan Stanley: You and your friend are playing a game. The two of you will continue to toss a coin until the sequence HH or TH shows up. If HH shows up first, you win. If TH shows up first, your friend wins. What is the probability of you winning?
- 5.12. JP Morgan: Say you are playing a game where you roll a 6-sided die up to two times and can choose to stop following the first roll if you wish. You will receive a dollar amount equal to the final amount rolled. How much are you willing to pay to play this game?
- 5.13. Facebook: Facebook has a content team that labels pieces of content on the platform as either spam or not spam. 90% of them are diligent raters and will mark 20% of the content as spam and 80% as non-spam. The remaining 10% are not diligent raters and will mark 0% of the content as spam and 100% as non-spam. Assume the pieces of content are labeled independently of one another, for every rater. Given that a rater has labeled four pieces of content as good, what is the probability that this rater is a diligent rater?
- 5.14. D.E. Shaw: A couple has two children. You discover that one of their children is a boy. What is the probability that the second child is also a boy?
- 5.15. JP Morgan: A desk has eight drawers. There is a probability of $1/2$ that someone placed a letter in one of the desk's eight drawers and a probability of $1/2$ that this person did not place a letter in any of the desk's eight drawers. You open the first 7 drawers and find that they are all empty. What is the probability that the 8th drawer has a letter in it?
- 5.16. Optiver: Two players are playing in a tennis match, and are at deuce (that is, they will play back and forth until one person has scored two more points than the other). The first player has a 60% chance of winning every point, and the second player has a 40% chance of winning every point. What is the probability that the first player wins the match?
- 5.17. Facebook: Say you have a deck of 50 cards made up of cards in 5 different colors, with 10 cards of each color, numbered 1 through 10. What is the probability that two cards you pick at random do not have the same color and are also not the same number?
- 5.18. SIG: Suppose you have ten fair dice. If you randomly throw these dice simultaneously, what is the probability that the sum of all the top faces is divisible by 6?

Medium

- 5.19. Morgan Stanley: A and B play the following game: a number k from 1-6 is chosen, and A and B will toss a die until the first person throws a die showing side k , after which that person is awarded \$100 and the game is over. How much is A willing to pay to play first in this game?
- 5.20. Airbnb: You are given an unfair coin having an unknown bias towards heads or tails. How can you generate fair odds using this coin?
- 5.21. SIG: Suppose you are given a white cube that is broken into $3 \times 3 \times 3 = 27$ pieces. However, before the cube was broken, all 6 of its faces were painted green. You randomly pick a small cube and see that 5 faces are white. What is the probability that the bottom face is also white?
- 5.22. Goldman Sachs: Assume you take a stick of length 1 and you break it uniformly at random into three parts. What is the probability that the three pieces can be used to form a triangle?
- 5.23. Lyft: What is the probability that, in a random sequence of H's and T's, HHT shows up before HTT?

- 5.24. Uber: A fair coin is tossed twice, and you are asked to decide whether it is more likely that two heads showed up given that either (a) at least one toss was heads, or (b) the second toss was a head. Does your answer change if you are told that the coin is unfair?
- 5.25. Facebook: Three ants are sitting at the corners of an equilateral triangle. Each ant randomly picks a direction and begins moving along an edge of the triangle. What is the probability that none of the ants meet? What would your answer be if there are, instead, k ants sitting on all k corners of an equilateral polygon?
- 5.26. Robinhood: A biased coin, with probability p of landing on heads, is tossed n times. Write a recurrence relation for the probability that the total number of heads after n tosses is even.
- 5.27. Citadel: Alice and Bob are playing a game together. They play a series of rounds until one of them wins two more rounds than the other. Alice wins a round with probability p . What is the probability that Bob wins the overall series?
- 5.28. Google: Say you have three draws of a uniformly distributed random variable between $(0, 2)$. What is the probability that the median of the three is greater than 1.5?

Hard

- 5.29. D.E. Shaw: Say you have 150 friends, and 3 of them have phone numbers that have the last four digits with some permutation of the digits 0, 1, 4, and 9. What's the probability of this occurring?
- 5.30. Spotify: A fair die is rolled n times. What is the probability that the largest number rolled is r , for each r in $1, \dots, 6$?
- 5.31. Goldman Sachs: Say you have a jar initially containing a single amoeba in it. Once every minute, the amoeba has a 1 in 4 chance of doing one of four things: (1) dying out, (2) doing nothing, (3) splitting into two amoebas, or (4) splitting into three amoebas. What is the probability that the jar will eventually contain no living amoeba?
- 5.32. Lyft: A fair coin is tossed n times. Given that there were k heads in the n tosses, what is the probability that the first toss was heads?
- 5.33. Quora: You have N i.i.d. draws of numbers following a normal distribution with parameters μ and σ . What is the probability that k of those draws are larger than some value Y ?
- 5.34. Akuna Capital: You pick three random points on a unit circle and form a triangle from them. What is the probability that the triangle includes the center of the unit circle?
- 5.35. Citadel: You have r red balls and w white balls in a bag. You continue to draw balls from the bag until the bag only contains balls of one color. What is the probability that you run out of white balls first?

Probability Interview Solutions

Solution #5.1

For the series to go to 7 games, each team must have won exactly three times for the first 6 games, an occurrence having probability

$$\frac{\binom{6}{3}}{2^6} = \frac{20}{64} = \frac{5}{16}$$

where the numerator is the number of ways of splitting up 3 games won by either side, and the denominator is the total number of possible outcomes of 6 games.

Solution #5.2

Note that there are only two ways for 6s to be consecutive: either the pair happens on rolls 1 and 2 or 2 and 3, or else all three are 6s. In the first case, the probability is given by

$$2 * \left(\frac{5}{6}\right) \left(\frac{1}{6}\right)^2 = \frac{10}{216}$$

and, for all three, the probability is

$$\left(\frac{1}{6}\right)^3 = \frac{1}{216}$$

The desired probability is given by: $\frac{10}{216} + \frac{1}{216} = \frac{11}{216}$

Solution #5.3

First, note that the three rolls must all yield different numbers; otherwise, no strictly increasing order is possible. The probability that the three numbers will be different is given by the following reasoning. The first number can be any value from 1 through 6, the second number has a 5/6 chance of not being the same number as the first, and the third number has a 4/6 chance of not being the prior two numbers. Thus,

$$1 * \frac{5}{6} * \frac{4}{6} = \frac{5}{9}$$

Conditioned on there being three different numbers, there is exactly one particular sequence that will be in a strictly increasing order, and this sequence occurs with probability $1/3! = 1/6$.

Therefore, the desired probability is given by: $\frac{5}{9} * \frac{1}{6} = \frac{5}{54}$

Solution #5.4

Note that there are a total of $\binom{100}{2} = 4950$

ways to choose two cards at random from the 100. There are exactly 50 pairs that satisfy the condition: (1, 2), ..., (50, 100). Therefore, the desired probability is:

$$\frac{50}{4950} \approx 0.01$$

Solution #5.5

Note that getting to (3, 3, 3) requires 9 moves. Using these 9 moves, it must be the case that there are exactly three moves in each of the three directions (up, right, and forward). There are therefore $9!$ ways to order the 9 moves in any given direction. We must divide by $3!$ for each direction to avoid overcounting, since each up move is indistinguishable. Therefore, the number of paths is:

$$\frac{9!}{3!3!3!} = 1680$$

Solution #5.6

Let A denote the event that someone has the disease, and B denote the event that this person tests positive for the disease. Then we want: $P(A|B)$

By applying Bayes' theorem, we obtain: $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$

From the problem description, we know that $P(B|A) = 0.98$, $P(A) = 0.001$

Let A' denote the event that someone does not have the disease. Then, we know that $P(B|A') = 0.01$. For the denominator, we have:

$$P(B) = P(B|A)P(A) + P(B|A')P(A') = 0.98(0.001) + 0.01(0.999)$$

Therefore, after combining terms, we have the following:

$$P(A|B) = \frac{0.98 * 0.001}{0.98(0.001) + 0.01(0.999)} = 8.93\%$$

Solution #5.7

We can use Bayes' theorem here. Let U denote the case where we are flipping the unfair coin and F denote the case where we are flipping a fair coin. Since the coin is chosen randomly, we know that $P(U) = P(F) = 0.5$. Let $5T$ denote the event of flipping 5 tails in a row. Then, we are interested in solving for $P(U|5T)$, i.e., the probability that we are flipping the unfair coin, given that we obtained 5 tails in a row.

We know $P(5T|U) = 1$, since, by definition, the unfair coin always results in tails. Additionally, we know that $P(5T|F) = 1/2^5 = 1/32$ by definition of a fair coin. By Bayes' theorem, we have:

$$P(U|5T) = \frac{P(5T|U)*P(U)}{P(5T|U)*P(U) + P(5T|F)*P(F)} = \frac{0.5}{0.5 + 0.5 * 1/32} = 0.97$$

Therefore, the probability we picked the unfair coin is about 97%.

Solution #5.8

Let $P(A)$ be the probability that A wins. Then, we know the following to be true:

1. If A flips heads initially, A wins with probability 1.
2. If A flips tails initially, and then B flips a tail, then it is as if neither flip had occurred, and so A wins with probability $P(A)$.

Combining the two outcomes, we have: $P(A) = p + (1 - p)^2P(A)$, and simplifying this yields $P(A) = p + P(A) - 2pP(A) + p^2P(A)$ so that $p^2P(A) - 2pP(A) + p = 0$

$$\text{and hence: } P(A) = \frac{1}{2-p}$$

Solution #5.9

Let R denote the event that it is raining, and Y be a “yes” response when you ask a friend if it is raining. Then, from Bayes' theorem, we have the following:

$$P(R|YYY) = \frac{P(YYY|R)P(R)}{P(YYY)}$$

where the numerator is given by:

$$P(YYY|R)P(R) = \left(\frac{2}{3}\right)^3 \left(\frac{1}{4}\right) = \frac{2}{27}$$

Let R' denote the event of no rain; then the denominator is given by the following:

$$P(YYY) = P(YYY|R)P(R) + P(YYY|R')P(R') = \left(\frac{2}{3}\right)^3 \left(\frac{1}{4}\right) + \left(\frac{1}{3}\right)^3 \left(\frac{3}{4}\right)$$

which, when simplified, yields: $P(YYY) = \frac{11}{108}$

Combining terms, we obtain the desired probability: $P(R|YYY) = \frac{\frac{2}{27}}{\frac{11}{108}} = \frac{8}{11}$

Solution #5.10

By definition, a chord is a line segment where the two endpoints lie on the circle. Therefore, two arbitrary chords can always be represented by any four points chosen on the circle. If you choose to represent the first chord by two of the four points, then you have:

$$\binom{4}{2} = 6$$

choices of choosing the two points to represent chord 1 (and, hence the other two will represent chord 2). However, note that in this counting, we are duplicating the count of each chord twice, since a chord with endpoints p1 and p2 is the same as a chord with endpoints p2 and p1. That is, chord AB is the same as BA, (likewise with CD and DC). Therefore, the proper number of valid chords is:

$$\frac{1}{2} \binom{4}{2} = 3$$

Among these three configurations, only one of the chords will intersect; hence, the desired probability is:

$$p = \frac{1}{3}$$

Solution #5.11

Although there is a formal way to apply Markov chains to this problem, there is a simple trick that simplifies the problem greatly. Note that, if T is ever flipped, you cannot then reach HH before your friend reaches TH, since the first heads thereafter will result in them winning. Therefore, the probability of you winning is limited to just flipping an HH initially, which we know is given by the following probability:

$$P(HH) = \frac{1}{2} * \frac{1}{2} = \frac{1}{4}$$

Therefore, you have a 1/4 chance of winning, whereas your friend has a 3/4 chance.

Solution #5.12

The price you would be willing to pay is equal to the expectation of the final amount. Note that, for the first roll, the expectation is

$$\sum_{i=1}^6 \frac{i}{6} = \frac{21}{6} = 3.5$$

Therefore, there are two events on which you need to condition. The first is on getting a 1, 2, or 3 on the first roll, in which case you would roll again (since a new roll would have an expectation of 3.5, and so, overall, you have an expectation of 3.5. The second is on if you roll a 4, 5, or 6 on the first roll, in which case you would keep that roll and end the game, and the overall expectation would be 5, the average of 4, 5, and 6. Therefore, the expected payoff of the overall game is

$$\frac{1}{2} * 3.5 + \frac{1}{2} * 5 = 4.25$$

Therefore, you would be willing to pay up to \$4.25 to play.

Solution #5.13

Let D denote the case where a rater is diligent, and E the case where a rater is non-diligent. Further, let $4N$ denote the case where four pieces of content are labeled as non-spam. We want to solve for $P(D|4N)$, and can use Bayes' theorem as follows to do so:

$$P(D|4N) = \frac{P(4N|D)*P(D)}{P(4N|D)*P(D)+P(4N|E)*P(E)}$$

We are given that $P(D) = 0.9$, $P(E) = 0.1$. Also, we know that $P(4N|D) = 0.8 * 0.8 * 0.8 * 0.8$ due to the independence of each of the 4 labels assigned by a diligent rater. Similarly, we know that $P(4N|E) = 1$, since a non-diligent rater always labels content as non-spam. Substituting into the equation above yields the following:

$$\frac{P(4N|D)*P(D)}{P(4N|D)*P(D)+P(4N|E)*P(E)} = \frac{0.8^4 * 0.9}{0.8^4 * 0.9 + 1^4 * 0.1} = 0.79$$

Therefore, the probability that the rater is diligent is 79%.

Solution #5.14

This is a tricky problem, because your mind probably jumps to the answer of 1/2 because knowing the gender of one child shouldn't affect the gender of the other. However, the phrase "the second child is also a boy" implies that we want to know the probability that both children are boys given that one is a boy. Let B represent a boy and G represent a girl. We then have the following total sample space representing the possible genders of 2 children: BB , BG , GB , GG .

However, since one child was said to be a boy, then valid sample space is reduced to the following: BB , BG , GB .

Since all of these options are equally likely, the answer is simply 1/3.

Solution #5.15

Let A denote the event that there is a letter in the 8th drawer, and B denote the event that the first 7 drawers are all empty.

The probability of B occurring can be found by conditioning on whether a letter was put in the drawers or not; if so, then each drawer is equally likely to contain a letter, and if not, then none contain the letter. Therefore, we have the following:

$$P(B) = \left(\frac{1}{2}\right)\left(\frac{1}{8}\right) + \left(\frac{1}{2}\right)(1) = \frac{9}{16}$$

For A and B to both occur, we also know that: $P(A \cap B) = \left(\frac{1}{2}\right)\left(\frac{1}{8}\right) = \frac{1}{16}$

Therefore, we have: $P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{1}{9}$

Solution #5.16

We can use a recursive formulation. Let p be the probability that the first player wins. Assume the score is 0-0 (on a relative basis).

If the first player wins a game (with probability 0.6), then two outcomes are possible: with probability 0.6 the first player wins, and with probability 0.4 the score is back to 0-0, with p being the probability of the first player winning overall.

Similarly, if the first player loses a game (with probability 0.4), then with probability 0.6 the score is back to 0-0 (with p being the probability of the first player winning), or, with probability 0.4, the first player loses. Therefore, we have: $p = 0.6^2 + 2(0.6)(0.4)p$

Solving this yields the following for p : $p \approx 0.692$

The key idea to solving this and similar problems is that, after two points, either the game is over, or we're back where we started. We don't need to ever consider the third, fourth, etc., points in an independent way.

Solution #5.17

The first card will always be a unique color and number, so let's consider the second card. Let A be the event that the color of card 2 does not match that of card 1, and let B be the event that the number of card 2 does not match that of card 1. Then, we want to find the following:

$$P(A \cap B)$$

Note that the two events are mutually exclusive: two cards with the same colors cannot have the same numbers, and vice versa. Hence, $P(A \cap B) = P(A)P(B|A)$

For A to occur, there are 40 remaining cards of a color different from that of the first card drawn (and 49 remaining cards altogether). Therefore,

$$P(A) = \frac{40}{49}$$

For B , we know that, of the 40 remaining cards, 36 of them (9 in each color) do not have the same number as that of card 1.

$$\text{Therefore, } P(B|A) = \frac{36}{40}$$

$$\text{Thus, the desired probability is: } P(A \cap B) = \frac{40}{49} * \frac{36}{40} = \frac{36}{49}$$

Solution #5.18

Consider the first nine dice. The sum of those nine dice will be either 0, 1, 2, 3, 4, or 5 modulo 6. Regardless of that sum, exactly one value for the tenth die will make the sum of all 10 divisible by 6. For instance, if the sum of the first nine dice is 1 modulo 6, the sum of the first 10 will be divisible by 6 *only* when the tenth die shows a 5. Thus, the probability is 1/6 for any number of dice, and, therefore, the answer is simply 1/6.

Solution #5.19

To assess the amount A is willing to pay, we need to calculate the expected probabilities of winning for each player, assuming A goes first. Let the probability of A winning (if A goes first) be given by $P(A)$, and the probability of B winning (if A goes first but doesn't win on the first roll) be $P(B')$.

Then we can use the following recursive formulation: $P(A) = \frac{1}{6} + \frac{5}{6}(1 - P(B'))$

Since A wins immediately with a $1/6$ chance (the first roll is k), or with a $5/6$ chance (assuming the first roll is not a k), A wins if B does not win, with B now going first.

However, notice that, if A doesn't roll side k immediately, then $P(B') = P(A)$, since now the game is exactly symmetric with player B going first.

Therefore, the above can be modeled as follows: $P(A) = \frac{1}{6} + \frac{5}{6} - \frac{5}{6}P(A)$

Solving yields $P(A) = 6/11$, and $P(B) = 1 - P(A) = 5/11$. Since the payout is \$100, then A should be willing to pay an amount up to the difference in expected values in going first, which is $100 * (6/11 - 5/11) = 100/11$, or about \$9.09.

Solution #5.20

Let $P(H)$ be the probability of landing on heads, and $P(T)$ be the probability of landing tails for any given flip, where $P(H) + P(T) = 1$. Note that it is impossible to generate fair odds using only one flip. If we use two flips, however, we have four outcomes: HH , HT , TH , and TT . Of these four outcomes, note that two (HT , TH) have equal probabilities since $P(H) * P(T) = P(T) * P(H)$. We can disregard HH and TT and need to complete only two sets of flips, e.g., HHT wouldn't be equivalent to HT .

Therefore, it is possible to generate fair odds by flipping the unfair coin twice and assigning heads to the HT outcome on the unfair coin, and tails to the TH outcome on the unfair coin.

Solution #5.21

The only possible candidates for the cube you selected are the following: either it is the inside center piece (in which case all faces are white) or a middle face (where 5 faces are white, and one face is green).

The former can be placed in six different ways, and the latter can only be placed in one particular way. Since all cubes are chosen equally randomly, let A be the event that the bottom face of the cube picked is white, and B be the event that the other five faces are white.

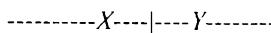
Note that there is a $1/27$ chance that the piece is the center piece and a $6/27$ chance that the piece is the middle piece. Therefore, the probability of B happening is given by the following:

$$P(B) = \frac{1}{27}(1) + \frac{6}{27}\left(\frac{1}{6}\right)$$

Then, using Bayes' rule: $P(A | B) = \frac{P(A \cap B)}{P(B)} = \frac{\frac{1}{27}}{\frac{1}{27} + \frac{6}{27} * \frac{1}{6}} = \frac{1}{2}$

Solution #5.22

Assume that the stick looks like the following, with cut points at X and Y



Let M (shown as | above) denote the stick's midpoint at 0.5 of the stick's 1-unit length. Note that, if X and Y fall on the same side of the midpoint, either on its left or its right, then no triangle is possible,

because, in that case, the length of one of the pieces would be greater than 1/2 (and thus we would have two sides having a total length strictly less than that of the longest side, making forming a triangle impossible). The probability that X and Y are on the same side (since the breaks are assumed to be chosen randomly) is simply 1/2.

Now, assume that X and Y fall on different sides of the midpoint. If X is further to the left in its half than Y is in its half, then no triangle is possible in that case, since then the part lying between X and Y would have a length strictly greater than 0.5 (for example, X at 0.2 and Y at 0.75). This has a 1/2 chance of occurring by a simple symmetry argument, but it is conditional on X and Y being on different sides of the midpoint, an outcome which itself has a 1/2 chance of occurring. Therefore, this case occurs with probability 1/4. The two cases represent all cases in which no valid triangle can be formed; thus, it follows that probability of a valid triangle being formed equals $1 - 1/2 - 1/4 = 1/4$.

Solution #5.23

Note that both sequences require a heads first, and any sequence of just tails prior to that is irrelevant to either showing up. Once the first H appears, there are three possibilities. If the next flip is an H , HHT will inevitably appear first, since the next T will complete that sequence. This has probability 1/2.

If the next flip is a T , there are two possibilities. If TT appears, then HTT appeared first. This has probability 1/4. Alternatively, if TH appears, we are back in the initial configuration of having gotten the first H . Thus, we have:

$$p = \frac{1}{2} + \frac{1}{4} p$$

Solving yields $p = \frac{2}{3}$

Solution #5.24

Let A be the event that the first toss is a heads and B be the event that the second toss is a heads. Then, for the first case, we are assessing: $P(A \cap B | A \cup B)$, whereas for the second case we are assessing: $P(A \cap B | B)$

For the first case, we have: $P(A \cap B | A \cup B) = \frac{P(A \cap B) \cap P(A \cup B)}{P(A \cup B)} = \frac{P(A \cap B)}{P(A \cup B)} = \frac{\frac{1}{4}}{\frac{3}{4}} = \frac{1}{3}$

And, for the second case, we have: $P(A \cap B | B) = \frac{P(A \cap B) \cap P(B)}{P(B)} = \frac{P(A \cap B)}{P(B)} = \frac{\frac{1}{4}}{\frac{1}{2}} = \frac{1}{2}$

Therefore, the second case is more likely. For an unfair coin, the outcomes are unchanged, because it will always be true that $P(A \cup B) > P(B)$, so the first case will always be less probable than the second case.

Solution #5.25

Note that the ants are guaranteed to collide unless they each move in the exact same direction. This only happens when all the ants move clockwise or all move counter-clockwise (picture the triangle in 2D). Let $P(N)$ denote the probability of no collision, $P(C)$ denote the case where all ants go clockwise, and $P(D)$ denote the case where all ants go counterclockwise. Since every ant can choose either direction with equal probability, then we have:

$$P(N) = P(C) + P(D) = \left(\frac{1}{2}\right)^3 + \left(\frac{1}{2}\right)^3 = \frac{1}{4}$$

If we extend this reasoning to k ants, the logic is still the same, so we obtain the following:

$$P(N) = P(C) + P(D) = \left(\frac{1}{2}\right)^k + \left(\frac{1}{2}\right)^k = \frac{1}{2^{k-1}}$$

Solution #5.26

Let A be the event that the total number of heads after n tosses is even, B be the event that the first toss was tails, and B' be the event that the first toss was heads. By the law of total probability, we have the following: $P(A) = P(A|B)P(B) + P(A|B')P(B')$

Then, we can write the recurrence relation as follows: $P_n = (1-p)P_{n-1} + p(1-P_{n-1})$

Solution #5.27

Note that since Alice can win with probability p , Bob, by definition, can win with probability $1-p$. Denote $1-p$ as q for convenience. Let B_i represent the event that Bob wins i matches for $i = 0, 1, 2$. Let B^* denote the event that Bob wins the entire series. We can use the law of conditional probability as follows:

$$P(B^*) = P(B^* | B_2) * P(B_2) + P(B^* | B_1) * P(B_1) + P(B^* | B_0) * P(B_0)$$

Since Bob wins each round with probability $1-p$, we have: $P(B_2) = q^2$, $P(B_1) = 2pq$, $P(B_0) = p^2$

Substituting these values into the above expression yields: $P(B^*) = 1 * q^2 + P(B^*) * 2pq + 0 * p^2$

Hence, the desired probability is the following: $P(B^*) = \frac{q^2}{1 - 2qp}$

Solution #5.28

Because the median of three numbers is the middle number, the median is at least 1.5 if at most one of the 3 is strictly less than 1.5 (since the other 2 must be strictly greater than 1.5). Since each is uniformly randomly distributed, then the probability of any one of them being strictly less than 1.5 is given by the following:

$$\frac{1.5}{2} = \frac{3}{4}$$

Therefore, the chance that at most one is strictly less than 1.5 is given by the sum of probabilities for exactly one being strictly less than 1.5 and none being strictly less than 1.5

$$p = \binom{3}{1} \left(\frac{3}{4}\right) \left(\frac{1}{4}\right)^2 + \left(\frac{1}{4}\right)^3 = \frac{10}{64}$$

Solution #5.29

Let p be the probability that a phone number has the last 4 digits involving only the above given digits (0, 1, 4 and 9).

We know that the total number of possible last 4 digit combinations is $10^4 = 10000$, since there are 10 digits (0-9). There are $4!$ ways to pick a 4-digit permutation of 0, 1, 4 and 9.

Therefore, we have: $p = \frac{4!}{10000} = \frac{3}{1250}$

Now, since you have 150 friends, the probability of there being exactly 3 with this combination is given by:

$$\binom{150}{3} p^3 (1-p)^{147} \approx 0.00535$$

Solution #5.30

Let B be the event that all n rolls have a value less than or equal to r . Then we have:

$$P(B_r) = \frac{r^n}{6^n}$$

since all n rolls must have a value less than or equal to r . Let A be the event that the largest number is r . We have: $B_r = B_{r-1} \cup A_r$, and, since the two events on the right-hand side are disjoint, we have the following: $P(B_r) = P(B_{r-1}) + P(A_r)$

Therefore, the probability of A is given by: $P(A_r) = P(B_r) - P(B_{r-1}) = \frac{r^n}{6^n} - \frac{(r-1)^n}{6^n}$

Solution #5.31

Let p be the probability that the amoeba(s) die out. At any given time step, the probability of dying out eventually must still be p .

For case (1), we know the probability of survival is 0 (for one amoeba).

For case (2), we know the probability of dying out is p .

For case (3), there are now two amoebas, and both have a probability p of dying.

For case (4), each of the three amoebas has a probability p of dying.

Putting all four together, we note that the probability of the population dying out at $t = 0$ minutes must be the same as the probability of the population dying out at $t = 1$ minutes. Therefore, we have:

$$p = \frac{1}{4}(1 + p + p^2 + p^3)$$

and solving this yields: $p = \sqrt{2} - 1$

Solution #5.32

Note that there are $\binom{n-1}{k}$ ways to choose k heads with the first coin being a T , and a total of $\binom{n}{k}$

ways to obtain k heads. So, the probability of having a tails first is given by: $\frac{\binom{n-1}{k}}{\binom{n}{k}} = \frac{n-k}{n}$

and, therefore, the probability of obtaining a heads first is given by the following:

$$1 - \frac{n-k}{n} = \frac{k}{n}$$

Solution #5.33

Let the n draws be denoted as X_1, X_2, \dots, X_n

We know that, for any given draw i , we have the following:

$$P(X_i > Y) = 1 - P(X_i \leq Y) = 1 - P\left(\frac{X_i - \mu}{\sigma} \leq \frac{Y - \mu}{\sigma}\right) = 1 - \Phi\left(\frac{Y - \mu}{\sigma}\right)$$

Additionally, the probability that k of those draws are greater than Y follows a binomial distribution with the value above being the p parameter:

$$p = 1 - \Phi\left(\frac{Y - \mu}{\sigma}\right)$$

Then, the desired probability is given by: $\binom{n}{k} p^k (1-p)^{n-k}$

Solution #5.34

Note that, without loss of generality, the first point can be located at $(1, 0)$. Using the polar coordinate system, we have the two other points at angles: θ, ϕ , respectively.

Note that the second point can be placed on either half (top or bottom) without loss of generality. Therefore, assume that it is on the top half. Then, $0 < \theta < \pi$

If the third point is also in the top half, then the resulting triangle will not contain the center of the unit circle. It will also not contain the center if the following is the case (try drawing this out): $\phi \geq \theta + \pi$

Therefore, for any given second point, the probability of the third point making the resulting triangle contain the center of the unit circle is the following:

$$p = \frac{\theta}{2\pi}$$

Therefore, the overall probability is given by the integrating over possible values of θ , where the constant in front is to take the average:

$$\frac{1}{\pi} \int_0^\pi p d\theta = \frac{1}{4}$$

Solution #5.35

In order to run out of white balls first, all the white balls must be drawn before the r -th red ball is drawn. We can consider the draws until $w+r-1$ (we know the last ball must be red), and count how many include w white balls.

The first white ball has $w+r-1$ options, the second white ball has $w+r-2$ options, etc., until the drawing of the w -th white ball: $(w+r-1)(w+r-2)\dots(r)$, which can be written as a factorial:

$$\frac{(w+r-1)!}{(r-1)!}$$

Similarly, there are $r!$ ways to arrange the drawing of the remaining r red balls. We know the total number of balls is $r+w$, so there are $(r+w)!$ total arrangements. Therefore, the probability is:

$$\frac{\frac{(w+r-1)!}{(r-1)!} r!}{(r+w)!} = \frac{r}{w+r}$$

A more intuitive way to approach the problem is to consider just the last ball drawn. The probability that the ball is red is simply the chance of it being red when picking randomly, which is the following:

$$\frac{r}{w+r}$$

Statistics

CHAPTER 6

Statistics is a core component of any data scientist's toolkit. Since many commercial layers of a data science pipeline are built from statistical foundations (for example, A/B testing), knowing foundational topics of statistics is essential.

Interviewers love to test a candidate's knowledge about the basics of statistics, starting with topics like the Central Limit Theorem and the Law of Large Numbers, and then progressing on to the concepts underlying hypothesis-testing, particularly p-values and confidence intervals, as well as Type I and Type II errors and their interpretations. All of those topics play an important role in the statistical underpinning of A/B testing. Additionally, derivations and manipulations involving random variables of various probability distributions are also common, particularly in finance interviews. Lastly, a common topic in more technical interviews will involve utilizing MLE and/or MAP.

Topics to Review Before Your Interview

Properties of Random Variables

For any given random variable X , the following properties hold true (below we assume X is continuous, but it also holds true for discrete random variables).

The expectation (average value, or mean) of a random variable is given by the integral of the value of X with its probability density function (PDF) $f_X(x)$:

$$\mu := E[X] := \int_{-\infty}^{\infty} xf_X(x)dx$$

and the variance is given by:

$$Var(X) = E[(X - E[X])^2] = E[X^2] - (E[X])^2$$

The variance is always non-negative, and its square root is called the standard deviation, which is heavily used in statistics.

$$\sigma = \sqrt{Var(X)} = \sqrt{E[(X - E[X])^2]} = \sqrt{E[X^2] - (E[X])^2}$$

The conditional values of both the expectation and variance are as follows. For example, consider the case for the conditional expectation of X , given that $Y = y$:

$$E[X | Y = y] = \int_{-\infty}^{\infty} xf_{X|Y}(x | y)dx$$

For any given random variables X and Y , the covariance, a linear measure of relationship between the two variables, is defined by the following:

$$Cov(X, Y) = E[(X - E[X])(Y - E[Y])] = E[XY] - E[X]E[Y]$$

and the normalization of covariance, represented by the Greek letter ρ , is the correlation between X and Y :

$$\rho(X, Y) = \frac{Cov(X, Y)}{\sqrt{Var(X)Var(Y)}}$$

All of these properties are commonly tested in interviews, so it helps to be able to understand the mathematical details behind each and walk through an example for each.

For example, if we assume X follows a Uniform distribution on the interval $[a, b]$, then we have the following:

$$f_X(x) = \frac{1}{b - a}$$

Therefore the expectation of X is:

$$E[X] = \int_a^b xf_X(x)dx = \int_a^b \frac{x}{b - a} dx = \frac{x^2}{2(b - a)} \Big|_a^b = \frac{a + b}{2}$$

Although it is not necessary to memorize the derivations for all the different probability distributions, you should be comfortable deriving them as needed, as it is a common request in more technical interviews. To this end, you should make sure to understand the formulas given above and be able to apply them to some of the common probability distributions like the exponential or uniform distribution.

Law of Large Numbers

The Law of Large Numbers (LLN) states that if you sample a random variable independently a large number of times, the measured average value should converge to the random variable's true expectation. Stated more formally,

$$\bar{X}_n = \frac{X_1 + \dots + X_n}{n} \rightarrow \mu, \text{ as } n \rightarrow \infty$$

This is important in studying the longer-term behavior of random variables over time. As an example, a coin might land on heads 5 times in a row, but over a much larger n we would expect the proportion

of heads to be approximately half of the total flips. Similarly, a casino might experience a loss on any individual game, but over the long run should see a predictable profit over time.

Central Limit Theorem

The Central Limit Theorem (CLT) states that if you repeatedly sample a random variable a large number of times, the distribution of the sample mean will approach a normal distribution regardless of the initial distribution of the random variable.

Recall from the probability chapter that the normal distribution takes on the form:

$$f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

with the mean and standard deviation given by μ and σ respectively.

The CLT states that: $\bar{X}_n = \frac{X_1 + \dots + X_n}{n} \rightarrow N\left(\mu, \frac{\sigma^2}{n}\right)$; hence $\frac{\bar{X}_n - \mu}{\sigma/\sqrt{n}} \sim N(0,1)$

The CLT provides the basis for much of hypothesis testing, which is discussed shortly. At a very basic level, you can consider the implications of this theorem on coin flipping: the probability of getting some number of heads flipped over a large n should be approximately that of a normal distribution. Whenever you're asked to reason about any particular distribution over a large sample size, you should remember to think of the CLT, regardless of whether it is Binomial, Poisson, or any other distribution.

Hypothesis Testing

General Setup

The process of testing whether or not a sample of data supports a particular hypothesis is called hypothesis testing. Generally, hypotheses concern particular properties of interest for a given population, such as its parameters, like μ (for example, the mean conversion rate among a set of users).

The steps in testing a hypothesis are as follows:

1. State a null hypothesis and an alternative hypothesis. Either the null hypothesis will be rejected (in favor of the alternative hypothesis), or it will fail to be rejected (although failing to reject the null hypothesis does not necessarily mean it is true, but rather that there is not sufficient evidence to reject it).
2. Use a particular test statistic of the null hypothesis to calculate the corresponding p -value.
3. Compare the p -value to a certain significance level α .

Since the null hypothesis typically represents a baseline (e.g., the marketing campaign did not increase conversion rates, etc.), the goal is to reject the null hypothesis with statistical significance and hope that there is a significant outcome.

Hypothesis tests are either one- or two-tailed tests. A one-tailed test has the following types of null and alternative hypotheses:

$$H_0 : \mu = \mu_0 \text{ versus } H_1 : \mu < \mu_0 \text{ or } H_1 : \mu > \mu_0$$

whereas a two-tailed test has these types: $H_0 : \mu = \mu_0$ versus $H_1 : \mu \neq \mu_0$

where H_0 is the null hypothesis and H_1 is the alternative hypothesis, and μ is the parameter of interest.

Understanding hypothesis testing is the basis of A/B testing, a topic commonly covered in tech companies' interviews. In A/B testing, various versions of a feature are shown to a sample of different users, and each variant is tested to determine if there was an uplift in the core engagement metrics.

Say, for example, that you are working for Uber Eats, which wants to determine whether email campaigns will increase its product's conversion rates. To conduct an appropriate hypothesis test, you would need two roughly equal groups (equal with respect to dimensions like age, gender, location, etc.). One group would receive the email campaigns and the other group would not be exposed. The null hypothesis in this case would be that the two groups exhibit equal conversion rates, and the hope is that the null hypothesis would be rejected.

Test Statistics

A test statistic is a numerical summary designed for the purpose of determining whether the null hypothesis or the alternative hypothesis should be accepted as correct. More specifically, it assumes that the parameter of interest follows a particular sampling distribution under the null hypothesis.

For example, the number of heads in a series of coin flips may be distributed as a binomial distribution, but with a large enough sample size, the sampling distribution should be approximately normally distributed. Hence, the sampling distribution for the total number of heads in a large series of coin flips would be considered normally distributed.

Several variations in test statistics and their distributions include:

1. *Z*-test: assumes the test statistic follows a normal distribution under the null hypothesis
2. *t*-test: uses a student's *t*-distribution rather than a normal distribution
3. Chi-squared: used to assess goodness of fit, and to check whether two categorical variables are independent

Z-Test

Generally the *Z*-test is used when the sample size is large (to invoke the CLT) or when the population variance is known, and a *t*-test is used when the sample size is small and when the population variance is unknown. The *Z*-test for a population mean is formulated as:

$$z = \frac{\bar{x} - \mu_0}{\sigma/\sqrt{n}} \sim N(0,1)$$

in the case where the population variance σ^2 is known.

t-Test

The *t*-test is structured similarly to the *Z*-test, but uses the sample variance s^2 in place of population variance. The *t*-test is parametrized by the degrees of freedom, which refers to the number of independent observations in a dataset, denoted below by $n - 1$:

$$t = \frac{\bar{x} - \mu_0}{s/\sqrt{n}} \sim t_{n-1}$$

where $s^2 = \frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n-1}$

As stated earlier, the t-distribution is similar to the normal distribution in appearance but has larger tails (i.e., extreme events happen with greater frequency than the modeled distribution would predict), a common phenomenon, particularly in economics and Earth sciences.

Chi-Squared Test

The Chi-squared test statistic is used to assess goodness of fit, and is calculated as follows:

$$\chi^2 = \sum_i \frac{(O_i - E_i)^2}{E_i}$$

where O_i is the observed value of interest and E_i is its expected value. A Chi-squared test statistic takes on a particular number of degrees of freedom, which is based on the number of categories in the distribution.

To use the squared test to check whether two categorical variables are independent, create a table of counts (called a contingency table), with the values of one variable forming the rows of the table and the values of the other variable forming its columns, and check for intersections. It uses the same style of Chi-squared test statistic as given above.

Hypothesis Testing for Population Proportions

Note that, due to the CLT, the Z-test can be applied to random variables of any distribution. For example, when estimating the sample proportion of a population having a characteristic of interest, we can view the members of the population as Bernoulli random variables, with those having the characteristic represented by “1s” and those lacking it represented by “0s”. Viewing the sample proportion of interest as the sum of these Bernoulli random variables divided by the total population size, we can then compute the sample mean and variance of the overall proportion, about which we can form the following set of hypotheses:

$$H_0 : \hat{p} = p_0 \text{ versus } H_1 : \hat{p} \neq p_0$$

and the corresponding test statistic to conduct a Z-test would be: $z = \frac{\hat{p} - p_0}{\sqrt{p_0(1 - p_0)/n}}$

In practice, these test statistics form the core of A/B testing. For instance, consider the previously discussed case, in which we seek to measure conversion rates within groups A and B, where A is the control group and B has the special treatment (in this case, a marketing campaign). Adopting the same null hypothesis as before, we can proceed to use a Z-test to assess the difference in empirical population means (in this case, conversion rates) and test its statistical significance at a predetermined level.

When asked about A/B testing or related topics, you should always cite the relevant test statistic and the cause of its validity (usually the CLT).

p-values and Confidence Intervals

Both p-values and confidence intervals are commonly covered topics during interviews. Put simply, a p-value is the probability of observing the value of the calculated test statistic under the null hypothesis assumptions. Usually, the p-value is assessed relative to some predetermined level of significance (0.05 is often chosen).

In conducting a hypothesis test, an α , or measure of the acceptable probability of rejecting a true null hypothesis, is typically chosen prior to conducting the test. Then, a confidence interval can also be calculated to assess the test statistic. This is a range of values that, if a large sample were taken, would contain the parameter value of interest $(1-\alpha)\%$ of the time. For instance, a 95% confidence interval would contain the true value 95% of the time. If 0 is included in the confidence intervals, then we cannot reject the null hypothesis (and vice versa).

The general form for a confidence interval around the population mean looks like the following, where the term is the critical value (for the standard normal distribution):

$$\mu \pm z_{\alpha/2} \frac{\sigma}{\sqrt{n}}$$

In the prior example with the A/B testing on conversion rates, we see that the confidence interval for a population proportion would be

$$\hat{p} \pm z_{\alpha/2} \sqrt{\frac{\hat{p}(1 - \hat{p})}{n}}$$

since our estimate of the true proportion will have the following parameters when estimated as approximately Gaussian:

$$\mu = \frac{np}{n} = p, \sigma^2 = \frac{np(1-p)}{n^2} = \frac{p(1-p)}{n}$$

As long as the sampling distribution of a random variable is known, the appropriate p-values and confidence intervals can be assessed.

Knowing how to explain p-values and confidence intervals, in technical and nontechnical terms, is very useful during interviews, so be sure to practice these. If asked about the technical details, always remember to make sure you correctly identify the mean and variance at hand.

Type I and II Errors

There are two errors that are frequently assessed: type I error, which is also known as a “false positive,” and type II error, which is also known as a “false negative.” Specifically, a type I error is when one rejects the null hypothesis when it is correct, and a type II error is when the null hypothesis is not rejected when it is incorrect.

Usually $1-\alpha$ is referred to as the confidence level, whereas $1-\beta$ is referred to as the power. If you plot sample size versus power, generally you should see a larger sample size corresponding to a larger power. It can be useful to look at power in order to gauge the sample size needed for detecting a significant effect. Generally, tests are set up in such a way as to have both $1-\alpha$ and $1-\beta$ relatively high (say at 0.95 and 0.8, respectively).

In testing multiple hypotheses, it is possible that if you ran many experiments — even if a particular outcome for one experiment is very unlikely — you would see a statistically significant outcome at least once. So, for example, if you set $\alpha = 0.05$ and run 100 hypothesis tests, then by pure chance you would expect 5 of the tests to be statistically significant. However, a more desirable outcome is to have the overall α of the 100 tests be 0.05, and this can be done by setting the new α to α/n , where n is the number of hypothesis tests (in this case, $\alpha/n = 0.05/100 = 0.0005$). This is known as Bonferroni correction, and using it helps make sure that the overall rate of false positives is controlled within a multiple testing framework.

Generally, most interview questions concerning Type I and II errors are qualitative in nature — for instance, requesting explanations of terms or of how you would go about assessing errors/power in an experimental setup.

MLE and MAP

Any probability distribution has parameters, so fitting parameters is an extremely crucial part of data analysis. There are two general methods for doing so. In maximum likelihood estimation (MLE), the goal is to estimate the most likely parameters given a likelihood function: $\theta_{MLE} = \arg \max L(\theta)$, where $L(\theta) = f_n(x_1 \dots x_n | \theta)$.

Since the values of X are assumed to be i.i.d., then the likelihood function becomes the following:

$$L(\theta) = \prod_{i=1}^n f(x_i | \theta)$$

The natural log of $L(\theta)$ is then taken prior to calculating the maximum; since log is a monotonically increasing function, maximizing the log-likelihood $\log L(\theta)$ is equivalent to maximizing the likelihood:

$$\log L(\theta) = \sum_{i=1}^n \log f(x_i | \theta)$$

Another way of fitting parameters is through maximum a posteriori estimation (MAP), which assumes a “prior distribution.”

$$\theta_{MAP} = \arg \max g(\theta) f(x_1 \dots x_n | \theta)$$

where the similar log-likelihood is again employed, and $g(\theta)$ is a density function of θ .

Both MLE and MAP are especially relevant in statistics and machine learning, and knowing these is recommended, especially for more technical interviews. For instance, a common question in such interviews is to derive the MLE for a particular probability distribution. Thus, understanding the above steps, along with the details of the relevant probability distributions, is crucial.

40 Real Statistics Interview Questions

Easy

- 6.1. Uber: Explain the Central Limit Theorem. Why it is useful?
- 6.2. Facebook: How would you explain a confidence interval to a non-technical audience?
- 6.3. Twitter: What are some common pitfalls encountered in A/B testing?
- 6.4. Lyft: Explain both covariance and correlation formulaically, and compare and contrast them.
- 6.5. Facebook: Say you flip a coin 10 times and observe only one heads. What would be your null hypothesis and p-value for testing whether the coin is fair or not?
- 6.6. Uber: Describe hypothesis testing and p-values in layman's terms?
- 6.7. Groupon: Describe what Type I and Type II errors are, and the trade-offs between them.
- 6.8. Microsoft: Explain the statistical background behind power.
- 6.9. Facebook: What is a Z-test and when would you use it versus a t-test?

- 6.10. Amazon: Say you are testing hundreds of hypotheses, each with t-test. What considerations would you take into account when doing this?

Medium

- 6.11. Google: How would you derive a confidence interval for the probability of flipping heads from a series of coin tosses?
- 6.12. Two Sigma: What is the expected number of coin flips needed to get two consecutive heads?
- 6.13. Citadel: What is the expected number of rolls needed to see all six sides of a fair die?
- 6.14. Akuna Capital: Say you're rolling a fair six-sided die. What is the expected number of rolls until you roll two consecutive 5s?
- 6.15. D.E. Shaw: A coin was flipped 1,000 times, and 550 times it showed heads. Do you think the coin is biased? Why or why not?
- 6.16. Quora: You are drawing from a normally distributed random variable $X \sim N(0, 1)$ once a day. What is the approximate expected number of days until you get a value greater than 2?
- 6.17. Akuna Capital: Say you have two random variables X and Y , each with a standard deviation. What is the variance of $aX + bY$ for constants a and b ?
- 6.18. Google: Say we have $X \sim \text{Uniform}(0, 1)$ and $Y \sim \text{Uniform}(0, 1)$ and the two are independent. What is the expected value of the minimum of X and Y ?
- 6.19. Morgan Stanley: Say you have an unfair coin which lands on heads 60% of the time. How many coin flips are needed to detect that the coin is unfair?
- 6.20. Uber: Say you have n numbers $1 \dots n$, and you uniformly sample from this distribution with replacement n times. What is the expected number of distinct values you would draw?
- 6.21. Goldman Sachs: There are 100 noodles in a bowl. At each step, you randomly select two noodle ends from the bowl and tie them together. What is the expectation on the number of loops formed?
- 6.22. Morgan Stanley: What is the expected value of the max of two dice rolls?
- 6.23. Lyft: Derive the mean and variance of the uniform distribution $U(a, b)$.
- 6.24. Citadel: How many cards would you expect to draw from a standard deck before seeing the first ace?
- 6.25. Spotify: Say you draw n samples from a uniform distribution $U(a, b)$. What are the MLE estimates of a and b ?

Hard

- 6.26. Google: Assume you are drawing from an infinite set of i.i.d random variables that are uniformly distributed from $(0, 1)$. You keep drawing as long as the sequence you are getting is monotonically increasing. What is the expected length of the sequence you draw?
- 6.27. Facebook: There are two games involving dice that you can play. In the first game, you roll two dice at once and receive a dollar amount equivalent to the product of the rolls. In the second game, you roll one die and get the dollar amount equivalent to the square of that value. Which has the higher expected value and why?

- 6.28. Google: What does it mean for an estimator to be unbiased? What about consistent? Give examples of an unbiased but not consistent estimator, and a biased but consistent estimator.
- 6.29. Netflix: What are MLE and MAP? What is the difference between the two?
- 6.30. Uber: Say you are given a random Bernoulli trial generator. How would you generate values from a standard normal distribution?
- 6.31. Facebook: Derive the expectation for a geometric random variable.
- 6.32. Goldman Sachs: Say we have a random variable $X \sim D$, where D is an arbitrary distribution. What is the distribution $F(X)$ where F is the CDF of X ?
- 6.33. Morgan Stanley: Describe what a moment generating function (MGF) is. Derive the MGF for a normally distributed random variable X .
- 6.34. Tesla: Say you have N independent and identically distributed draws of an exponential random variable. What is the best estimator for the parameter λ ?
- 6.35. Citadel: Assume that $\log X \sim N(0, 1)$. What is the expectation of X ?
- 6.36. Google: Say you have two distinct subsets of a dataset for which you know their means and standard deviations. How do you calculate the blended mean and standard deviation of the total dataset? Can you extend it to K subsets?
- 6.37. Two Sigma: Say we have two random variables X and Y . What does it mean for X and Y to be independent? What about uncorrelated? Give an example where X and Y are uncorrelated but not independent.
- 6.38. Citadel: Say we have $X \sim \text{Uniform}(-1, 1)$ and $Y = X^2$. What is the covariance of X and Y ?
- 6.39. Lyft: How do you uniformly sample points at random from a circle with radius R ?
- 6.40. Two Sigma: Say you continually sample from some i.i.d. uniformly distributed $(0, 1)$ random variables until the sum of the variables exceeds 1. How many samples do you expect to make?

40 Real Statistics Interview Solutions

Solution #6.1

The Central Limit Theorem (CLT) states that if any random variable, regardless of distribution, is sampled a large enough number of times, the sample mean will be approximately normally distributed. This allows for studying of the properties for any statistical distribution as long as there is a large enough sample size.

The mathematical definition of the CLT is as follows: for any given random variable X , as n approaches infinity,

$$\bar{X}_n = \frac{X_1 + \dots + X_n}{n} \xrightarrow{\sim} N\left(\mu, \frac{\sigma^2}{n}\right)$$

At any company with a lot of data, like Uber, this concept is core to the various experimentation platforms used in the product. For a real-world example, consider testing whether adding a new feature increases rides booked in the Uber platform, where each X is an individual ride and is a Bernoulli random variable (i.e., the rider books or does not book a ride). Then, if the sample size is sufficiently large, we can assess the statistical properties of the total number of bookings, as well as the booking rate (rides booked / rides opened on app). These statistical properties play a key role in hypothesis testing, allowing companies like Uber to decide whether or not to add new features in a data-driven manner.

Solution #6.2

Suppose we want to estimate some parameters of a population. For example, we might want to estimate the average height of males in the U.S. Given some data from a sample, we can compute a sample mean for what we think the value is, as well as a range of values around that mean. Following the previous example, we could obtain the heights of 1,000 random males in the U.S. and compute the average height, or the sample mean. This sample mean is a type of point estimate and, while useful, will vary from sample to sample. Thus, we can't tell anything about the variation in the data around this estimate, which is why we need a range of values through a confidence interval.

Confidence intervals are a range of values with a lower and an upper bound such that if you were to sample the parameter of interest a large number of times, the 95% confidence interval would contain the true value of this parameter 95% of the time. We can construct a confidence interval using the sample standard deviation and sample mean. The level of confidence is determined by a margin of error that is set beforehand. The narrower the confidence interval, the more precise the estimate, since there is less uncertainty associated with the point estimate of the mean.

Solution #6.3

A/B testing has many possible pitfalls that depend on the particular experiment and setup employed. One common drawback is that groups may not be balanced, possibly resulting in highly skewed results. Note that balance is needed for all dimensions of the groups — like user demographics or device used — because, otherwise, the potentially statistically significant results from the test may simply be due to specific factors that were not controlled for. Two types of errors are frequently assessed: Type I error, which is also known as a “false positive,” and Type II error, also known as a “false negative.” Specifically, Type I error is rejecting a null hypothesis when that hypothesis is correct, whereas Type II error is failing to reject a null hypothesis when its alternative hypothesis is correct.

Another common pitfall is not running an experiment for long enough. Generally speaking, experiments are run with a particular power threshold and significance threshold; however, they often do not stop immediately upon detecting an effect. For an extreme example, assume you're at either Uber or Lyft and running a test for two days, when the metric of interest (e.g., rides booked) is subject to weekly seasonality.

Lastly, dealing with multiple tests is important because there may be interactions between results of tests you are running and so attributing results may be difficult. In addition, as the number of variations you run increases, so does the sample size needed. In practice, while it seems technically feasible to test 1,000 variations of a button when optimizing for click-through rate, variations in tests are usually based on some intuitive hypothesis concerning core behavior.

Solution #6.4

For any given random variables X and Y , the covariance, a linear measure of relationship, is defined by the following: $\text{Cov}(X, Y) = E[(X - E[X])(Y - E[Y])] = E[XY] - E[X]E[Y]$

Specifically, covariance indicates the direction of the linear relationship between X and Y and can take on any potential value from negative infinity to infinity. The units of covariance are based on the units of X and Y , which may differ.

The correlation between X and Y is the normalized version of covariance that takes into account the variances of X and Y :

$$\rho(X, Y) = \frac{\text{Cov}(X, Y)}{\sqrt{\text{Var}(X)\text{Var}(Y)}}$$

Since correlation results from scaling covariance, it is dimensionless (unlike covariance) and is always between -1 and 1 (also unlike covariance).

Solution #6.5

The null hypothesis is that the coin is fair, and the alternative hypothesis is that the coin is biased towards tails (note this is a one-sided test):

$$H_0 : p_0 = 0.5, H_1 : p_1 < 0.5$$

Note that, since the sample size here is 10, you cannot apply the Central Limit Theorem (and so you cannot approximate a binomial using a normal distribution).

The p-value here is the probability of observing the results obtained given that the null hypothesis is true, i.e., under the assumption that the coin is fair. In total for 10 flips of a coin, there are $2^{10} = 1024$ possible outcomes, and in only 10 of them are there 9 tails and one heads. Hence, the exact probability of the given result is the p-value, which is $\frac{10}{1024} = 0.0098$. Therefore, with a significance level set, for example, at 0.05, we can reject the null hypothesis.

Solution #6.6

The process of testing whether data supports particular hypotheses is called hypothesis testing and involves measuring parameters of a population's probability distribution. This process typically employs at least two groups — one a control that receives no treatment, and the other group(s), which do receive the treatment(s) of interest. Examples could be the height of two groups of people, the conversion rates for particular user flows in a product, etc. Testing also involves two hypotheses — the null hypothesis, which assumes no significant difference between the groups, and the alternative hypothesis, which assumes a significant difference in the measured parameter(s) as a consequence of the treatment.

A p-value is the probability of observing the given test results under the null hypothesis assumptions. The lower this probability, the higher the chance that the null hypothesis should be rejected. If the p-value is lower than the predetermined significance level α , generally set at 0.05, then it indicates that the null hypothesis should be rejected in favor of the alternative hypothesis. Otherwise, the null hypothesis cannot be rejected, and it cannot be concluded that the treatment has any significant effect.

Solution #6.7

Both errors are relevant in the context of hypothesis testing. Type I error is when one rejects the null hypothesis when it is correct, and is known as a false positive. Type II error is when the null hypothesis is not rejected when the alternative hypothesis is correct; this is known as a false negative. In layman's terms, a type I error is when we detect a difference, when in reality there is no significant difference in an experiment. Similarly, a type II error occurs when we fail to detect a difference, when in reality there is a significant difference in an experiment.

Type I error is given by the level of significance α , whereas the type II error is given by β . Usually, $1-\alpha$ is referred to as the confidence level, whereas $1-\beta$ is referred to as the statistical power of the test being conducted. Note that, in any well-conducted statistical procedure, we want to have both α and β be small. However, based on the definition of the two, it is impossible to make both errors small

simultaneously: the larger α is, the smaller β is. Based on the experiment and the relative importance of false positives and false negatives, a data scientist must decide what thresholds to adopt for any given experiment. Note that experiments are set up so as to have both $1-\alpha$ and $1-\beta$ relatively high (say at .95, and .8, respectively).

Solution #6.8

Power is the probability of rejecting the null hypothesis when, in fact, it is false. It is also the probability of avoiding a Type II error. A Type II error occurs when the null hypothesis is not rejected when the alternative hypothesis is correct. This is important because we want to detect significant effects during experiments. That is, the higher the statistical power of the test, the higher the probability of detecting a genuine effect (i.e., accepting the alternative hypothesis and rejecting the null hypothesis). A minimum sample size can be calculated for any given level of power — for example, say a power level of 0.8.

An analysis of the statistical power of a test is usually performed with respect to the test's level of significance (α) and effect size (i.e., the magnitude of the results).

Solution #6.9

In a Z-test, your test statistic follows a normal distribution under the null hypothesis. Alternatively, in a t-test, you employ a student's t-distribution rather than a normal distribution as your sampling distribution.

Considering the population mean, we can use either Z-test or t-test only if the mean is normally distributed, which is possible in two cases: the initial population is normally distributed, or the sample size is large enough ($n \geq 30$) that we can apply the Central Limit Theorem.

If the condition above is satisfied, then we need to decide which type of test is more appropriate to use. In general, we use Z-tests if the population variation is *known*, and vice versa: we use t-test if the population variation is *unknown*.

Additionally, if the sample size is very large ($n > 200$), we can use the Z-test in any case, since for such large degrees of freedom, t-distribution coincides with z-distribution up to thousands.

Considering the population proportion, we can use a Z-test (but not t-test) where $np_0 \geq 10$ and $n(1-p_0) \geq 10$, i.e., when each of the number of successes and the number of failures is at least 10.

Solution #6.10

The primary consideration is that, as the number of tests increases, the chance that a stand-alone p-value for any of the t-tests is statistically significant becomes very high due to chance alone. As an example, with 100 tests performed and a significance threshold of $\alpha = 0.05$, you would expect five of the experiments to be statistically significant due only to chance. That is, you have a very high probability of observing at least one significant outcome. Therefore, the chance of incorrectly rejecting a null hypothesis (i.e., committing Type I error) increases.

To correct for this effect, we can use a method called the Bonferroni correction, wherein we set the significance threshold to α/m , where m is the number of tests being performed. In the above scenario with 100 tests, we can set the significance threshold to instead be $0.05/100 = 0.0005$. While this correction helps to protect from Type I error, it is still prone to Type II error (i.e., failing to reject the null hypothesis when it should be rejected). In general, the Bonferroni correction is mostly useful when there is a smaller number of multiple comparisons of which a few are significant. If the number of tests becomes sufficiently high such that many tests yield statistically significant results, the number of Type II errors may also increase significantly.

Solution #6.11

The confidence interval (CI) for a population proportion is an interval that includes a true population proportion with a certain degree of confidence $1 - \alpha$.

For the case of flipping heads from a series of coin tosses, the proportion follows the binomial distribution. If the series size is large enough (each of the number of successes and the number of failures is at least 10), we can utilize the Central Limit Theorem and use the normal approximation for the binomial distribution as follows:

$$N\left(\hat{p}, \frac{\hat{p}(1 - \hat{p})}{n}\right)$$

where \hat{p} is the proportion of heads tossed in series, and n is the series size. The CI is centered at the series proportion, and plus or minus a margin of error:

$$\hat{p} \pm z_{\alpha/2} \sqrt{\frac{\hat{p}(1 - \hat{p})}{n}}$$

where $z_{\alpha/2}$ is the appropriate value from the standard normal distribution for the desired confidence level.

For example, for the most commonly used level of confidence, 95%, $z_{\alpha/2} = 1.96$.

Solution #6.12

Let X be the number of coin flips needed to obtain two consecutive heads. We then want to solve for $E[X]$. Let H denote a flip that results in heads, and T denote a flip that results in tails. Note that $E[X]$ can be written in terms of $E[X|H]$ and $E[X|T]$, i.e., the expected number of flips needed, conditioned on a flip being either heads or tails, respectively.

Conditioning on the first flip, we have: $E[X] = \frac{1}{2}(1 + E[X|H]) + \frac{1}{2}(1 + E[X|T])$

Note that $E[X|T] = E[X]$ since if a tail is flipped, we need to start over in getting two heads in a row.

To solve for $E[X|H]$, we can condition it further on the next outcome: either heads (HH) or tails (HT).

Therefore, we have: $E[X|H] = \frac{1}{2}(1 + E[X|HH]) + \frac{1}{2}(1 + E[X|HT])$

Note that if the result is HH, then $E[X|HH] = 0$, since the outcome has been achieved. If a tail was flipped, then $E[X|HT] = E[X]$, and we need to start over in attempting to get two heads in a row. Thus:

$$E[X|H] = \frac{1}{2}(1 + 0) + \frac{1}{2}(1 + E[X]) = 1 + \frac{1}{2}E[X]$$

Plugging this into the original equation yields:

$$E[X] = \frac{1}{2}\left(1 + 1 + \frac{1}{2}E[X]\right) + \frac{1}{2}(1 + E[X])$$

and after solving we get: $E[X] = 6$. Therefore, we would expect 6 flips.

Solution #6.13

Let k denote the number of distinct sides seen from rolls. The first roll will always result in a new side being seen. If you have seen k sides, where $k < 6$, then the probability of rolling an unseen value will be $(6 - k)/6$, since there are $6 - k$ values you have not seen, and 6 possible outcomes of each roll.

Note that each roll is independent of previous rolls. Therefore, for the second roll ($k = 1$), the time until a side not seen appears has a geometric distribution with $p = 5/6$, since there are five of the six sides left to be seen. Likewise, after two sides ($k = 2$), the time taken is a geometric distribution, with $p = 4/6$. This continues until all sides have been seen.

Recall that the mean for a geometric distribution is given by $1/p$, and let X be the number of rolls needed to show all six sides. Then, we have the following:

$$E[X] = 1 + \frac{6}{5} + \frac{6}{4} + \frac{6}{3} + \frac{6}{2} + \frac{6}{1} = 6 \sum_{p=1}^6 \frac{1}{p} = 14.7 \text{ rolls}$$

Solution #6.14

Similar in methodology to question 13, let X be the number of rolls until two consecutive fives. Let Y denote the event that a five was just rolled.

Conditioning on Y , we know that either we just rolled a five, so we only have one more five to roll, or we rolled some other number and now need to start over after having rolled once:

$$E[X] = \frac{1}{6}(1 + E[X|Y]) + \frac{5}{6}(1 + E[X])$$

Note that we have the following: $E[X|Y] = \frac{1}{6}(1) + \frac{5}{6}(1 + E[X])$

Plugging the results in yields an expected value of 42 rolls: $E[X] = 42$

Solution #6.15

Because the sample size of flips is large (1,000), we can apply the Central Limit Theorem. Since each individual flip is a Bernoulli random variable, we can assume that p is the probability of getting heads. We want to test whether p is .5 (i.e., whether it is a fair coin or not). The Central Limit Theorem allows us to approximate the total number of heads seen as being normally distributed.

More specifically, the number of heads seen out of n total rolls follows a binomial distribution since it is a sum of Bernoulli random variables. If the coin is not biased ($p = .5$), then the expected number of heads is as follows: $\mu = np = 1000 * 0.5 = 500$, and the variance of the number of heads is given by:

$$\sigma^2 = np(1-p) = 1000 * 0.5 * 0.5 = 250, \sigma = \sqrt{250} \approx 16$$

Since this mean and standard deviation specify the normal distribution, we can calculate the corresponding z -score for 550 heads as follows:

$$z = \frac{550 - 500}{16} = 3.16$$

This means that, if the coin were fair, the event of seeing 550 heads should occur with a < 0.1% chance under normality assumptions. Therefore, the coin is likely biased.

Solution #6.16

Since X is normally distributed, we can employ the cumulative distribution function (CDF) of the normal distribution: $\Phi(2) = P(X \leq 2) = P(X \leq \mu + 2\sigma) = 0.9772$

Therefore, $P(X > 2) = 1 - 0.9772 = 0.023$ for any given day. Since each day's draws are independent, the expected time until drawing an $X > 2$ follows a geometric distribution, with $p = 0.023$. Letting T be a random variable denoting the number of days, we have the following:

$$E[T] = \frac{1}{p} = \frac{1}{.02272} \approx 44 \text{ days}$$

Solution #6.17

Let the variances for X and Y be denoted by $Var(X)$ and $Var(Y)$.

Then, recalling that the variance of a sum of variables is expressed as follows:

$$Var(X + Y) = Var(X) + Var(Y) + 2Cov(X, Y)$$

and that a constant coefficient of a random variable is assessed as follows: $Var(aX) = a^2Var(X)$

We have $Var(aX + bY) = a^2Var(X) + b^2Var(Y) + 2abCov(X, Y)$, which would provide the bounds on the designated variance; the range will depend on the covariance between X and Y .

Solution #6.18

Let $Z = \min(X, Y)$. Then we know the following: $P(Z \leq z) = P(\min(X, Y) \leq z) = 1 - P(X > z, Y > z)$

For a uniform distribution, the following is true for a value of z between 0 and 1:

$$P(X > z) = 1 - z \text{ and } P(Y > z) = 1 - z$$

Since X and Y are i.i.d., this yields: $P(Z \leq z) = 1 - P(X > z, Y > z) = 1 - (1 - z)^2$

Now we have the cumulative distribution function for z . We can get the probability density function by taking the derivative of the CDF to obtain the following: $f_Z(z) = 2(1 - z)$. Then, solving for the expected value by taking the integral yields the following:

$$E[Z] = \int_0^1 z f_Z(z) dz = 2 \int_0^1 z(1 - z) dz = 2 \left(\frac{1}{2} - \frac{1}{3} \right) = \frac{1}{3}$$

Therefore, the expected value for the minimum of X and Y is $1/3$.

Solution #6.19

Say we flip the unfair coin n times. Each flip is a Bernoulli trial with a success probability of p :

$$x_1, x_2, \dots, x_n, x_i \sim Ber(p)$$

We can construct a confidence interval for p as follows, using the Central Limit Theorem. First, we decide on our level of confidence. If we select a 95% confidence level, the necessary z -score is $z = 1.96$. We then construct a 95% confidence interval for p . If it does not include 0.5 as its lower bound, then we can reject the null hypothesis that the coin is fair.

Since the trials are i.i.d., we can compute the sample mean for p from a large number of trials:

$$\hat{p} = \frac{1}{n} \sum_{i=1}^n x_i$$

We know the following properties hold: $E[\hat{p}] = \frac{np}{n} = p$ and $Var(\hat{p}) = \frac{np(1-p)}{n^2} = \frac{p(1-p)}{n}$

Therefore, our 95% confidence interval is given by the following: $p \pm z \sqrt{\frac{p(1-p)}{n}}$

Since the true $p = 0.6$, plugging that in and setting the lower bound of the interval equal to 0.5 yields:

$$0.6 - 1.96 \sqrt{\frac{0.6(1-0.6)}{n}} = 0.5$$

Solving for n yields 93 flips.

Solution #6.20

Let the following be an indicator random variable: $X_i = 1$ if i is drawn in n turns

We would then want to find the following: $\sum_{i=1}^n E[X_i]$

We know that $p(X_i = 1) = 1 - p(X_i = 0)$, so the probability of a number not being drawn (where each draw is independent) is the following:

$$p(X_i = 0) = \left(\frac{n-1}{n}\right)^n$$

Therefore, we have: $p(X_i = 1) = 1 - \left(\frac{n-1}{n}\right)^n$ and by linearity of expectation, we then have:

$$\sum_{i=1}^n E[X_i] = nE[X_i] = n\left(1 - \left(\frac{n-1}{n}\right)^n\right)$$

Solution #6.21

Say that we have n noodles. At any given step, we will have one of two outcomes: (1) we pick two ends from the same noodle (which makes a loop), or (2) we pick two ends from different noodles. Let X_n denote a random variable representing the number of loops with n noodles remaining.

The probability of case (1) happening is: $\frac{\binom{n}{2}}{\binom{2n}{2}} = \frac{1}{2n-1}$

where the denominator represents the number of ends we can choose from the noodles, and the numerator represents the number of cases where we choose the same noodle.

Therefore, the probability of case (2) happening is: $1 - \frac{1}{2n-1} = \frac{2n-2}{2n-1}$

Then, taking case (1) and (2), we have the following recursive formulation for the expectation of the number of loops formed:

$$E[X_n] = \frac{1}{2n-1} + \frac{2n-2}{2n-1} E[X_{n-1}]$$

Plugging in $E[X_1] = 1$ and calculating the first few terms, we can notice the following pattern, for which we can plug in $n = 100$ to obtain the answer:

$$E[X_{100}] = 1 + \frac{1}{3} + \dots + \frac{1}{2(100)-1} \approx 3.3$$

Solution #6.22

Since we only have two dice, let the maximum value between the two be m . Let

$$X_1, X_2, Y = \max(X_1, X_2)$$

denote the first roll, second roll, and the max of the two. Then we want to find the following:

$$E[Y] = \sum_{i=1}^6 i * P(Y = i)$$

We can condition $Y = m$ on three cases: (1) die one is the max roll; (2) die two is the max roll; or (3) they are both the same.

For cases (1) and (2) we have: $P(X_1 = i, X_2 < i) = P(X_2 = i, X_1 < i) = \frac{1}{6} * \frac{i-1}{6}$

"For case (3), where both dice are the maximum:"

$$P(X_1 = X_2 = i) = \frac{1}{6} * \frac{1}{6}$$

Putting everything together yields the following: $E[Y] = \sum_{i=1}^6 i * \left(\frac{1}{6} * \frac{i-1}{6} * 2 + \frac{1}{6} * \frac{1}{6} \right) = \frac{161}{36}$

A simpler way to visualize this is to use a contingency table, such as the one below:

1	2	3	4	5	6
(1, 1)	(1, 2)	(1, 3)	(1, 4)	(1, 5)	(1, 6)
(2, 1)	(2, 2)	(2, 3)	(2, 4)	(2, 5)	(2, 6)
(3, 1)	(3, 2)	(3, 3)	(3, 4)	(3, 5)	(3, 6)
(4, 1)	(4, 2)	(4, 3)	(4, 4)	(4, 5)	(4, 6)
(5, 1)	(5, 2)	(5, 3)	(5, 4)	(5, 5)	(5, 6)
(6, 1)	(6, 2)	(6, 3)	(6, 4)	(6, 5)	(6, 6)

Then the expectation is simply given by:

$$E[Y] = 1 * \frac{1}{6} + 2 * \frac{3}{6} + 3 * \frac{5}{6} + 4 * \frac{7}{6} + 5 * \frac{9}{6} + 6 * \frac{11}{6} = \frac{161}{36} \approx 4.5$$

Solution #6.23

For $X \sim U(a, b)$, we have the following: $f_X(x) = \frac{1}{b-a}$

Therefore, we can calculate the mean as:

$$E[X] = \int_a^b x f_X(x) dx = \int_a^b \frac{x}{b-a} dx = \frac{x^2}{2(b-a)} \Big|_a^b = \frac{a+b}{2}$$

Similarly, the variance can be as expressed as follows: $Var(X) = E[X^2] - E[X]^2$

Giving us:

$$E[X^2] = \int_a^b x^2 f_X(x) dx = \int_a^b \frac{x^2}{b-a} dx = \frac{x^3}{3(b-a)} \Big|_a^b = \frac{a^2 + ab + b^2}{3}$$

$$\text{Therefore: } Var(X) = \frac{a^2 + ab + b^2}{3} - \left(\frac{a+b}{2} \right)^2 = \frac{(b-a)^2}{12}$$

Solution #6.24

Although one can enumerate all the probabilities, this can get a bit messy from an algebraic standpoint, so obtaining the following intuitive answer is more preferable. Imagine we have aces A1, A2, A3, A4. We can then draw a line in between them to represent an arbitrary number (including 0) of cards between each ace, with a line before the first ace and after the last.

$$|A1|A2|A3|A4|$$

There are $52 - 4 = 48$ non-ace cards in a deck. Each of these cards is equally likely to be in any of the five lines. Therefore, there should be $48/5 = 9.6$ cards drawn prior to the first ace being drawn. Hence, the expected number of cards drawn until the first ace is seen is $9.6 + 1 = 10.6$ cards — we can't forget to add 1, because we need to include drawing the ace card itself.

Solution #6.25

Note that for a uniform distribution, the probability density is $\frac{1}{b-a}$ for any value on the interval $[a, b]$. The likelihood function is therefore as follows:

$$f(x_1, \dots, x_n | a, b) = \left(\frac{1}{b-a}\right)^n$$

To obtain the MLE, we maximize this likelihood function, which is clearly maximized if b is the largest of the samples and a is the smallest of the samples. Therefore, we have the following:

$$\hat{a} = \min(x_1, \dots, x_n), \hat{b} = \max(x_1, \dots, x_n)$$

Solution #6.26

Assume that we have an indicator random variable: $X_i = 1$ if the sequence is increasing up to i th element, and otherwise $X_i = 0$.

Then, we calculate the expectation: $E[X_1 + X_2 + \dots]$. Consider some arbitrary i . In order to draw up to element i , the entire sequence up to i must be monotonically increasing, which means that the following is true: $X_1 < X_2 < \dots < X_i$. Given that there are n possible sequences of the elements, there is a

$$\frac{1}{i!}$$

chance of X_i being 1. Since each X is i.i.d., we then have: $E[X_1 + X_2 + \dots] = 1 + \frac{1}{2!} + \dots = e - 1$

Solution #6.27

One method of solving this problem is the brute force method, which consists of computing the expected values by listing all of the outcomes and associated probabilities and payoffs. However, there exists an easier way of solving the problem.

Assume that the outcome of the roll of a die is given by a random variable X (meaning that it takes on the values 1...6 with equal probability). Then, the question is equivalent to asking, “What is $E[X] * E[X] = E[X]^2$ (i.e., the expected value of the product of two separate rolls), versus $E[X^2]$ (the expected value of the square of a single roll)?”

Recall that the variance of a given random variable X is as follows:

$$Var(X) = E[(X - E[X])^2] = E[X^2] - 2E[X]E[X] + E[X]^2 = E[X^2] - E[X]^2$$

Typically, this variance term is exactly the difference between the two sets of die rolls — the two “games” — (the payoff of the second game minus the payoff of the first game). Since the left-hand side is positive, as expected for the value of a squared number, then the right-hand side is also positive. Therefore, it must be the case that the second game has a higher expected value than the first.

Solution #6.28

In both cases, we are dealing with an estimator of the true parameter value. An estimator is unbiased if the expectation of the estimator is the true underlying parameter value. An estimator is consistent if, as the sample size increases, the estimator’s sampling distribution converges towards the true parameter value.

Consider the following random variable X , which is normally distributed, and n i.i.d. samples used to calculate a sample mean:

$$X \sim N(\mu, \sigma^2) \text{ and } \bar{x} = \frac{x_1 + x_2 + \dots + x_n}{n}$$

The first sample is an example of an unbiased but not consistent estimator. It is unbiased since $E[x_1] = \mu$. However, it is not consistent since, as the sample size increases, the sampling distribution of the first sample does not become more concentrated with respect to the true mean.

An example of a biased but consistent estimator is the sample variance: $S_n^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2$

It can be shown that $E[S_n^2] = \frac{n-1}{n} \sigma^2$

The formal proof of the above is called Bessel's correction, but there is an intuitive way to grasp the presence of the term preceding the variance. If we uniformly sample two numbers randomly from the series of numbers 1 to n , we have an $n/n^2 = 1/n$ chance that the two equal the same number, meaning the sampled squared difference of the numbers will be zero. The sample variance will therefore slightly underestimate the true variance. However, this bias goes to 0 as n approaches infinity, since the term in front of the variance, $(n-1/n)$, approaches 1. Therefore, the estimator is consistent.

Solution #6.29

MLE stands for maximum likelihood estimation, and MAP for maximum a posteriori. Both are ways of estimating variables in a probability distribution by producing a single estimate of that variable.

Assume that we have a likelihood function $P(X|\theta)$. Given n i.i.d. samples, the MLE is as follows:

$$MLE(\theta) = \max_{\theta} P(X|\theta) = \max_{\theta} \prod_i^n P(x_i|\theta)$$

Since the product of multiple numbers all valued between 0 and 1 might be very small, maximizing the log function of the product above is more convenient. This is an equivalent problem, since the log function is monotonically increasing. Since the log of a product is equivalent to the sum of logs, the MLE becomes the following:

$$MLE_{\log}(\theta) = \max_{\theta} \sum_{i=1}^n \log P(x_i|\theta)$$

Relying on Bayes rule, MAP uses the posterior $P(\theta|X)$ being proportional to the likelihood multiplied by a prior $P(\theta)$, i.e., $P(X|\theta)P(\theta)$. The MAP for θ is thus the following:

$$MAP(\theta) = \max_{\theta} P(X|\theta) = \max_{\theta} \prod_i^n P(x_i|\theta) P(\theta)$$

Employing the same math as used in calculating the MLE, the MAP becomes:

$$MAP_{\log}(\theta) = \max_{\theta} \sum_{i=1}^n \log P(x_i|\theta) + \log P(\theta)$$

Therefore, the only difference between the MLE and MAP is the inclusion of the prior in MAP; otherwise, the two are identical. Moreover, MLE can be seen as a special case of the MAP with a uniform prior.

Solution #6.30

Assume we have n Bernoulli trials, each with a p probability of success. Altogether, they form a binomial distribution: $x_1, x_2, \dots, x_n, X \sim B(n, p)$ where $x_i = 1$ means success and $x_i = 0$ means failure. Assuming i.i.d. trials, we can compute the sample proportion for \hat{p} as follows:

$$\hat{p} = \frac{1}{n} \sum_{i=1}^n x_i$$

We know that if n is large enough, then the binomial distribution approximates the following normal distribution:

$$\hat{p} \sim N\left(p, \frac{p(1-p)}{n}\right)$$

where n must be $np \geq 10$, $n(1-p) \geq 10$

Therefore, the value \hat{p} can be used as simulation for a normal distribution. The sample size n must only be large enough to satisfy the conditions above (at least $n = 20$ for $p = .5$), but it is recommended to use a significantly larger n to get the better normal approximation.

Finally, to simulate the standard normal distribution, we normalize \hat{p} : $\hat{p}_0 = \frac{\hat{p} - p}{\sqrt{\frac{p(1-p)}{n}}}$

At this point, we can derive the final formula for our normal random generator: $\bar{x} = \frac{\frac{1}{n} \sum_{i=1}^n x_i - p}{\sqrt{\frac{p(1-p)}{n}}}$

The previous expression can be simplified to the following: $\bar{x} = \frac{\sum_{i=1}^n x_i - np}{\sqrt{np(1-p)}}$

where x_1, \dots, x_n is the Bernoulli series we get from the given random generator, with probability of success p .

Solution #6.31

We are seeking the expected value of geometric random variable X as follows: $E[X] = \sum_{k=1}^{\infty} kf_X(k)$

The expression above contains a summation instead of an integral since k is a discrete rather than continuous random variable, and we know the probability mass function of the geometric probability distribution is given by the following: $f_X(k) = (1-p)^{k-1} p$

Therefore, we obtain the expected value of X as follows: $E[X] = \sum_{k=1}^{\infty} k(1-p)^{k-1} p$

Since p is constant with respect to k , we separate out p as follows: $E[X] = p \sum_{k=1}^{\infty} k(1-p)^{k-1}$

Note that the term inside the summation is really the following:

$$\sum_{k=1}^{\infty} k(1-p)^{k-1} = \sum_{k=1}^{\infty} k(1-p)^{k-1} + \sum_{k=2}^{\infty} k(1-p)^{k-1} + \dots$$

This simplifies to the following:

$$\begin{aligned} \sum_{k=1}^{\infty} k(1-p)^{k-1} &= \left(\frac{1}{p}\right) + (1-p)/p + (1-p)^2/p + \dots = \frac{1}{p}(1 + (1-p) + (1-p)^2 + \dots) \\ &= \frac{1}{p} \cdot \frac{1}{1-(1-p)} = \frac{1}{p^2} \end{aligned}$$

Plugging this back into the equation for the expected value of X yields the following:

$$E[X] = p * \frac{1}{p^2} = \frac{1}{p}$$

Solution #6.32

We can define a new variable $Y = F(X)$, and, hence, we want to find the CDF of y (where y is between 0 and 1 by definition of a CDF): $F_Y(y) = P(Y \leq y)$

Substituting in for Y yields the following: $F_Y(y) = P(F(X) \leq y)$

Applying the inverse CDF on both sides yields the following:

$$F_Y(y) = P(F^{-1}(F(X)) \leq F^{-1}(y)) = P(X \leq F^{-1}(y))$$

Note that the last expression is simply the CDF for: $P(X \leq F^{-1}(y)) = F(F^{-1}(y)) = y$

Therefore, we have: $F_Y(y) = y$

Since y falls between 0 and 1, Y 's distribution is simply a uniform one from 0 to 1, i.e., $U(0, 1)$.

Solution #6.33

A moment generating function is the following function for a given random variable:

$$M_X(s) = E[e^{sX}]$$

If X is continuous (as in the case of normal distributions), then the function becomes the following:

$$M_X(s) = \int_{-\infty}^{\infty} e^{sx} f_X(x) dx$$

Hence, the moment generating function is a function for a given value of s . It is useful for calculating moments, since taking derivatives of the moment generating function and evaluating at $s = 0$ yields the desired moment.

For a normal distribution, recall that: $f_X(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$

First, taking the special case of the standard normal random variable, we have the following:

$$f_X(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$$

Plugging this into the above MGF yields: $M_X(s) = \int_{-\infty}^{\infty} e^{sx} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2} dx = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-\frac{1}{2}x^2 + sx} dx$

Completing the square yields: $M_X(s) = e^{\frac{s^2}{2}} \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-\frac{1}{2}x^2 + sx - \frac{s^2}{2}} dx = e^{\frac{s^2}{2}} \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-\frac{(x-s)^2}{2}} dx = e^{\frac{s^2}{2}}$

Note that the last step uses the fact that the expression within the integral is a PDF for a normally distributed random variable with mean s and variance 1, and hence the integral evaluates to 1.

To solve for a general random variable, you can plug in $X = \sigma Y + \mu$, where Y is standard normal variable, to yield: $M_Y(s) = e^{s\mu}$, $M_X(s\sigma) = e^{(s^2\sigma^2/2) + s\mu}$

Solution #6.34

Denote the n i.i.d. draws as: x_1, x_2, \dots, x_n where, for any individual draw, we have the pdf: $f_X(x_i) = \lambda e^{-\lambda x_i}$

Therefore the likelihood of the data is given by the following:

$$L(\lambda; x_1, \dots, x_n) = \prod_{i=1}^n f_X(x_i) = \lambda^n \exp\left(-\lambda \sum_{i=1}^n x_i\right)$$

Taking the log of the equation above to obtain the log-likelihood results in the following:

$$\log L(\lambda; x_1 \dots x_n) = n \log(\lambda) - \lambda \sum_{i=1}^n x_i$$

Taking the derivative with respect to λ and setting the results to 0 yields: $\frac{n}{\lambda} - \sum_{i=1}^n x_i = 0$

Therefore, the best estimate of λ is given by: $\hat{\lambda} = \frac{n}{\sum_{i=1}^n x_i}$

Solution #6.35

Define $Y = \log X$. We then want to solve for: $E[e^Y] = E[X]$

Recall that a moment generating function has the following form: $M_Y(s) = E[e^{sY}]$

Therefore, we want the moment generating function for $Y \sim N(0, 1)$, which was derived in problem 33 and has the form: $M_Y(s) = e^{s^2/2}$

Therefore, evaluating at $s = 1$ (since we want the mean) gives: $M_Y(1) = e^{1/2}$ which is the desired answer.

Solution #6.36

Say that the two have two distinct group sizes: n_1 = size of group 1, and n_2 = size of group 2.

Given the means of two groups, μ_1 and μ_2 , the blended mean can be found simply by taking a weighted average:

$$\bar{\mu} = \frac{n_1 \mu_1 + n_2 \mu_2}{n_1 + n_2}$$

We know that the blended standard deviation for the total data set has the form:

$$\bar{s} = \sqrt{\frac{\sum_{i=1}^{n_1+n_2} (z_i - \bar{\mu})^2}{n_1 + n_2}}$$

where z_i is the union of the points from both groups.

However, since we are not given the initial data points from the two groups, we have to rearrange this formula by using instead the given variations of these groups, s_1^2 and s_2^2 , as follows:

$$\bar{s} = \sqrt{\frac{n_1 s_1^2 + n_2 s_2^2 + n_1 (\bar{y}_1 - \bar{y})^2 + n_2 (\bar{y}_2 - \bar{y})^2}{n_1 + n_2}}$$

Applying the Bessel correction, the blended standard deviation for the two groups is as follows:

$$\bar{s} = \sqrt{\frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2 + n_1 (\bar{\mu}_1 - \bar{\mu})^2 + n_2 (\bar{\mu}_2 - \bar{\mu})^2}{n_1 + n_2 - 2}}$$

To extend the definition above to subsets, the mean is as follows: $\bar{\mu} = \frac{\sum_{i=1}^K n_i \mu_i}{\sum_{i=1}^K n_i}$

And the standard deviation is: $\bar{s}_K = \sqrt{\frac{\sum_{i=1}^K (n_i - 1)s_i^2 + n_i (\bar{\mu}_i - \bar{\mu})^2}{\sum_{i=1}^K n_i - 1}}$

where n_i are the sizes of initial groups, μ_i and s_i are their respective means and standard deviations.

Solution #6.37

Independence is defined as follows: $P(X = x, Y = y) = P(X = x)P(Y = y)$ for all x, y . Equivalently, we can use the following definitions: $P(X = x | Y = y) = P(X = x)$, $P(Y = y | X = x) = P(Y = y)$

When two random variables X and Y are uncorrelated, their covariance, which is calculated as follows, is 0: $\text{Cov}(X, Y) = E[XY] - E[X]E[Y]$

For an example of uncorrelated but not independent variables, let X take on values $-1, 0$, or 1 with equal probability, and let $Y = 1$ if $X = 0$ and $Y = 0$ otherwise. Then we can verify that X and Y are uncorrelated:

$$E(XY) = \frac{1}{3}(-1)(0) + \frac{1}{3}(0)(1) + \frac{1}{3}(1)(0) = 0$$

And $E[X] = 0$, so the covariance between the two random variables is zero. However, it is clear that the two are not independent, since we defined Y in such a way that it obviously depends on X .

$$P(Y = y | X = x) \neq P(Y = y)$$

For example, $P(Y = 1 | X = 0) = 1$

Solution #6.38

By definition of the covariance, we have: $\text{Cov}(X, Y) = \text{Cov}(X, X^2) = E[(X - E[X])(X^2 - E[X^2])]$

Expanding terms of the equation above yields: $\text{Cov}(X, Y) = E[(X^3 - XE[X^2] - X^2E[X] + E[X]E[X^2])]$

Using linearity of expectation, we obtain: $\text{Cov}(X, Y) = E[X^3] - E[X]E[X^2] - E[X^2]E[X] + E[X]E[X^2]$

Since the second and last terms cancel one another, we end up with the following:

$$\text{Cov}(X, Y) = E[X^3] - E[X^2]E[X]$$

Here, we conclude that $E[X] = 0$ (based on the definition of X) and that $E[X^3] = 0$ by evaluating the probability density function of X as follows:

$$f_X(x) = \frac{1}{b-a} = \frac{1}{1-(-1)} = \frac{1}{2}$$

Since we are evaluating X from -1 to 1 , we then have: $E[X^3] = \int_{-1}^1 x^3 f(x) dx = \int_{-1}^1 \frac{1}{2} x^3 dx = 0$

Thus, the covariance between X and Y is 0.

Solution #6.39

This can be proved using the inverse-transform method, whereby we sample from a uniform distribution and then simulate the points on the circle employing the inverse cumulative distribution functions (i.e., inverse CDFs).

We can define a random point within the circle using a given radius value and an angle (and obtain the corresponding x, y values from polar coordinates). To sample a random radius, consider the following. If we sample points from a radius r , we know that there are $2\pi r$ points to consider (i.e., the circumference of the circle). Likewise, if we sample a radius $2r$, there are $4\pi r$ points to consider. Therefore, we have the following probability density function given by the following:

$$f_R(r) = \frac{2r}{R^2}$$

This follows from the CDF, which is given by the ratio of the areas of the two circles: $F_R(r) = \frac{r^2}{R^2}$

Therefore, for the inverse sampling, we want the following: $y = \frac{r^2}{R^2}$

This simplifies to the following: $\sqrt{R^2 y} = r$

Therefore, we can sample $Y \sim U(0, 1)$ and the corresponding radius will be the following:

$$r = R\sqrt{y}$$

For the corresponding angles, we can sample theta uniformly from the range 0 to 2π : $\theta \in [0, 2\pi]$ and then set the following: $x = r \cos(\theta)$, $y = r \sin(\theta)$

Solution #6.40

Let us define: N_t = smallest n such that: $\sum_{i=1}^n U_i > t$ for any value t between 0 and 1. Then we want to find: $m(t) = E[N_t]$

Consider the first draw. Assuming that result is some value x , we then have two cases as follows. The first is that $x > t$, in which $N_t = 1$

The second is that $x < t$, necessitating that we sample again, yielding: $N_t = 1 + N_{t-x}$

Putting these two together, we have: $m(t) = 1 + \int_0^1 m(t-x) dx$

Employing the following change of variables: $u = t - x$, $du = -dx$

We then substitute and simplify to obtain: $m(t) = 1 + \int_0^t m(u) du$

Differentiating both sides, we then obtain: $m'(t) = m(t)$

Since $m(0) = 1$, we then have: $m(t) = e^t$

Since we actually need to find $m(N_{t+1})$, we can plug in $t = 1$ into the equation, which yields the desired result $m(1) = e$.

Machine Learning

CHAPTER 7

*How much machine learning do you **actually** need to know to land a top job in Silicon Valley or Wall Street? Probably less than you think! From coaching hundreds of data folks on the job hunt, one of the most common misconceptions we saw was candidates thinking their lack of deep learning expertise would tank their performance in data science interviews. However, the truth is that most data scientists are hired to solve business problems — not blindly throw complicated neural networks on top of dirty data. As such, a data scientist with strong business intuition can create more business value by applying linear regression in an Excel sheet than a script kiddie whose knowledge doesn't extend beyond the Keras API.*

So, unless you're interviewing for ML Engineering or research scientist roles, a solid understanding of the classical machine learning techniques covered in this chapter is all you need to ace the data science interview. However, if you are aiming for ML-heavy roles that do require advanced knowledge, this chapter will still be handy! Throughout this chapter, we frequently call attention to which topics and types of questions show up in tougher ML interviews. Plus, the 35 questions at the end of the chapter — especially the hard ones — will challenge even the most seasoned ML practitioner.

What to Expect for ML Interview Questions

When machine learning is brought up in an interview context, the problems fall into three major buckets:

- ***Conceptual questions:*** Do you have a strong theoretical ML background?
- ***Resume-driven questions:*** Have you actively applied ML before?
- ***End-to-end modeling questions:*** Can you apply ML to a hypothetical business problem?

Conceptual Questions

Conceptual questions usually center around what different machine learning terms mean and how popular machine learning techniques operate. For example, two frequently asked questions are “What is the bias-variance tradeoff?” and “How does PCA work?” To test your ability to communicate with nontechnical stakeholders, a common twist on these conceptual questions is to ask you to explain the answer as if they (the interviewer) were five years old (similar to Reddit’s popular r/ELI5 subreddit).

Because many data science roles don’t require hardcore machine learning knowledge, easier, straightforward questions such as these represent the vast majority of questions you’d expect during a typical interview. Being asked easier ML questions is especially the case when interviewing for a data science role that’s more product and business analytics oriented, as having to build models just simply isn’t part of the day-to-day work.

For ML-intensive positions like ML Engineer or Research Scientist, interviews also start with high-level easier conceptual questions but then push you to dive deeper into the details via follow-up questions. Companies do this to make sure you aren’t a walking, talking ML buzzword generator. For example, as a follow-up to defining the bias-variance trade-off, you might be asked to whiteboard the math behind the concept. Instead of simply asking you how PCA (principal components analysis) works, you might also be asked about the most common pitfalls of using PCA.

Since ML interviews are so expansive in scope, if asked about a particular technique you may not be overly familiar with, it’s perfectly okay to say, “I’ve read about it in the past. I don’t have any experience with these types of techniques, but I am interested to learn more about them!”

This signals honesty and an eagerness to learn (and don’t be ashamed to admit not knowing something nobody knows all the techniques in detail! Trust us, it’s better than pretending you know the techniques and then falling apart when questions are asked).

If nothing on your resume seems interesting to an interviewer, but they still want to go deep into one ML topic, they have you pick the topic. They do this by either asking “What’s your favorite ML algorithm?” or “What’s a model you use often and why?” Consequently, it pays to have a deep understanding of at least a single technique — something you’ve actually used before and that is listed on your resume.

Word of caution: don’t choose something about a state-of-the-art transformer model to discuss as your favorite technique. Your details on it may be hazy, and your interviewer might not know enough about it to carry on a good conversation. You are better off picking something fundamental yet interesting (to you) so that you and your interviewer can have a meaningful discussion. For example, our answer happens to be that we both like random forests because they can handle classification or regression tasks with all kinds of input features with minimal preprocessing needed. Additionally, we both have projects on our resume to back up our interest in random forests.

Resume-Driven Questions

The next most common type of interview question for ML interviews is the resume-driven question. Resume-driven questions are often about showcasing that you have practical experience (as opposed

to conceptual knowledge). As such, if you have job experience that is directly relevant, interviewers will often ask about that. If not, they'll often fall back to asking about your projects.

While anything listed on your resume is fair game to be picked apart, this is especially true for more ML-heavy roles. Because the field is so vast and continually evolving, an interviewer isn't able to assess your fit for the job by asking about some niche topic unrelated to the position at hand. For example, say you are going for a general data science role — it's not fair to ask a candidate about CNNs and their use in computer vision if they have no experience with this topic and it's not relevant to the job. But, suppose you hacked together a self-driving toy car last summer, and listed it on your resume. In that case — even though the role at hand may not require computer vision — it's totally fair game to be asked more about the neural network architecture you used, model training issues you faced, and trade-offs you made versus other techniques. Plus, in an effort to see if you know the details not just of your project, but of the greater landscape, you'd also be expected to answer questions tangentially related to the project.

End-to-End Modeling Questions

Finally, the last type of ML-related problems can expect during interviews are end-to-end modeling questions. Interviewers are testing your ability to go beyond the ML theory covered in books like *An Introduction to Statistical Learning* and actually apply what you learned to solve real-world problems. Examples of questions include "How would you match Uber drivers to riders?" and "How would you build a search autocomplete feature for Pinterest?" While these open-ended problems are an interview staple for any machine-learning-heavy role, they do also pop up during generalist data science interviews.

At the end of this chapter, we cover the end-to-end machine learning workflow, which can serve as a framework for answering these broad ML questions. We cover steps like problem definition, feature engineering, and performance metric selection — things you'd do before jumping into the various ML techniques we soon cover. To better solve these ML case study problems, we also recommend reading Chapter 11: Case Study to understand the non-ML-specific advice we offer for tackling open-ended problems.

The Math Behind Machine Learning

While the probability and statistics concepts upon which machine learning's foundation is built are fair game for interviews, you're less likely to be asked about the linear algebra and multivariable calculus concepts that underlie machine learning. There are, however, two notable exceptions: if you're interviewing for a research scientist position or for quant finance. In these cases, you may be expected to whiteboard proofs and derive formulas. For example, you could be asked to derive the least squares estimator in linear regression or explain how to calculate the principal components in PCA. Sometimes, to see how strong your first principles are, you'll be given a math problem more indirectly. For instance, you could be asked to analyze the statistical factors driving portfolio returns (which essentially boils down to explaining the math behind PCA). Regardless of the role and company, we still recommend you review the basics, since understanding them will help you grok the theoretical underpinnings of the techniques covered later in this chapter.

Linear Algebra

The main linear algebra subtopic worth touching on for interviews is eigenvalues and eigenvectors. Mechanically, for some $n \times n$ matrix A , x is an eigenvector of A if: $Ax = \lambda x$, where λ is a scalar. A

matrix can represent a linear transformation and, when applied to a vector x , results in another vector called an *eigenvector*, which has the same direction as x and is in fact x multiplied by a scaling factor λ known as an *eigenvalue*.

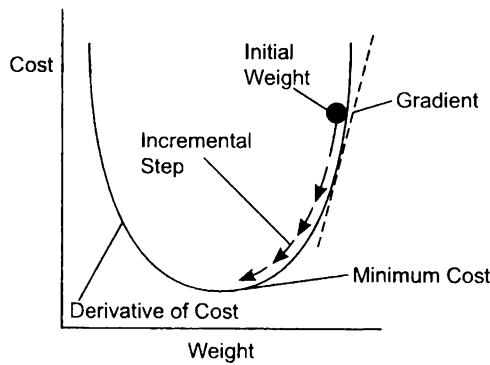
The decomposition of a square matrix into its eigenvectors is called an *eigendecomposition*. However, not all matrices are square. Non-square matrices are decomposed using a method called singular value decomposition (SVD). A matrix to which SVD is applied has a decomposition of the form: $A = U\Sigma V^T$, where U is an $m \times m$ matrix, Σ is an $m \times n$ matrix, and V is an $n \times n$ matrix.

There are many applications of linear algebra in ML, ranging from the matrix multiplications during backpropagation in neural networks, to using eigendecomposition of a covariance matrix in PCA. As such, during technical interviews for ML engineering and quantitative finance roles, you should be able to whiteboard any follow-up questions on the linear algebra concepts underlying techniques like PCA and linear regression. Other linear algebra topics you're expected to know are core building blocks like vector spaces, projections, inverses, matrix transformations, determinants, orthonormality, and diagonalization.

Gradient Descent

Machine learning is concerned with minimizing some particular objective function (most commonly known as a loss or cost function). A loss function measures how well a particular model fits a given dataset, and the lower the cost, the more desirable. Techniques to optimize the loss function are known as optimization methods.

One popular optimization method is gradient descent, which takes small steps in the direction of steepest descent for a particular objective function. It's akin to racing down a hill. To win, you always take a "next step" in the steepest direction downhill.



For convex functions, the gradient descent algorithm eventually finds the optimal point by updating the below equation until the value at the next iteration is very close to the current iteration (convergence):

$$x_{t+1} = x_t - \alpha_t \nabla f(x_t)$$

that is, it calculates the negative of the gradient of the cost function and scales that by some constant α_t , which is known as the learning rate, and then moves in that direction at each iteration of the algorithm.

Since many cost functions in machine learning can be broken down into the sum of individual functions, the gradient step can be broken down into adding separate gradients. However, this process can be computationally expensive, and the algorithm may get stuck at a local minimum or saddle

point. Therefore, we can use a version of gradient descent called *stochastic gradient descent* (SGD), which adds an element of randomness so that the gradient does not get stuck. SGD uses one data point at a time for a single step and uses a much smaller subset of data points at any given step, but is nonetheless able to obtain an unbiased estimate of the true gradient. Alternatively, we can use *batch gradient descent* (BGD), which uses a fixed, small number (a mini-batch) of data points per step.

Gradient descent and SGD are popular topics for ML interviews since they are used to optimize the training of almost all machine learning methods. Besides the usual questions on the high-level concepts and mathematical details, you may be asked when you would want to use one or the other. You might even be asked to implement a basic version of SGD in a coding interview (which we cover in Chapter 9, problem #30).

Model Evaluation and Selection

With the math underlying machine learning techniques out of the way, how do we actually choose the best model for our problem, or compare two models against each other? *Model evaluation* is the process of evaluating how well a model performs on the *test set* after it's been trained on the *train set*. Separating out your training data — usually 80% for the train set — from the 20% of the test set is critical because the usefulness of a model boils down to how good predictions are on data that has not been seen before.

Model selection, as the name implies, is the process of selecting which model to implement after each model has been evaluated. Both steps (evaluation and selection) are critical to get right, because even tiny changes in model performance can lead to massive gains at big tech companies. For example, at Facebook, a model that can cause even a 0.1% lift in ad click-through rates can lead to \$10+ million in extra revenue.

That's why in interviews, especially during case-study questions where you solve an open-ended problem, discussions often head toward comparing and contrasting models, and selecting the most suitable one after factoring in business and product constraints. Thus, internalizing the concepts covered in this section is key to succeeding in ML interviews.

Bias-Variance Trade-off

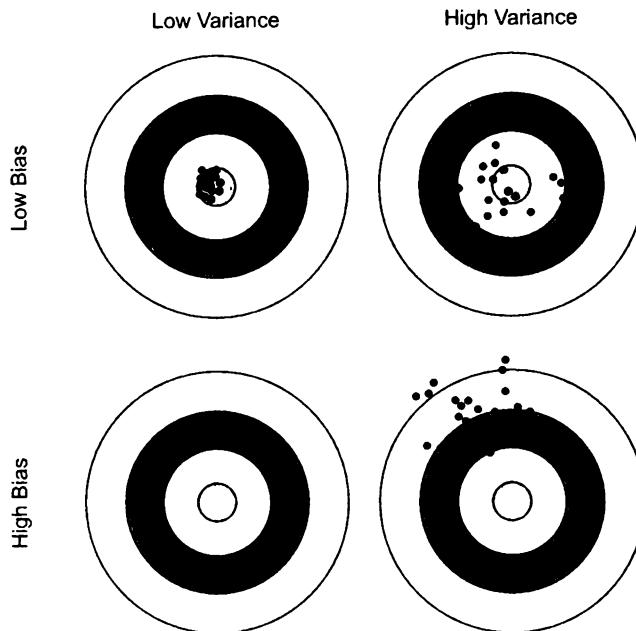
The bias-variance trade-off is an interview classic, and is a key framework for understanding different kinds of models. With any model, we are usually trying to estimate a function $f(x)$, which predicts our target variable y based on our input x . This relationship can be described as follows:

$$y = f(x) + w$$

where w is noise, not captured by $f(x)$, and is assumed to be distributed as a zero-mean Gaussian random variable for certain regression problems. To assess how well the model fits, we can decompose the error of y into the following:

- 1 **Bias:** how close the model's predicted values come to the true underlying $f(x)$ values, with smaller being better
- 2 **Variance:** the extent to which model prediction error changes based on training inputs, with smaller being better
- 3 **Irreducible error:** variation due to inherently noisy observation processes

The trade-off between bias and variance provides a lens through which you can analyze different models. Say we want to predict housing prices given a large set of potential predictors (square footage of a house, the number of bathrooms, and so on). A model with high bias but low variance, such as linear regression, is easy to implement but may oversimplify the situation at hand. This high bias but low variance situation would mean that predicted house prices are frequently off from the market value, but the variance in these predicted prices is low. On the flip side, a model with low bias and high variance, such as neural networks, would lead to predicted house prices closer to market value, but with predictions varying wildly based on the input features.

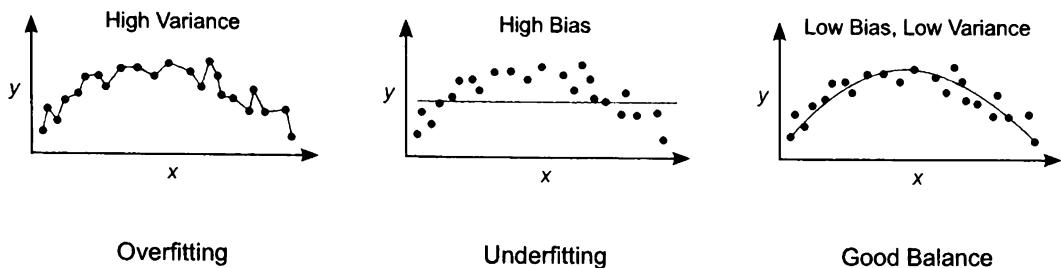


While the bias-variance trade-off equation occasionally shows up in data science interviews, more frequently, you'll be asked to reason about the bias-variance trade-off given a specific situation. For example, presented with a model that has high variance, you could mention how you'd source additional data to fix the issue. Posed with a situation where the model has high bias, you could discuss how increasing the complexity of the model could help. By understanding the business and product requirements, you'll know how to make the bias-variance trade-off for the interview problem posed.

Model Complexity and Overfitting

“All models are wrong, but some are useful” is a well-known adage, coined by statistician George Box. Ultimately, our goal is to discover a model that can generalize to learn some relationship within datasets. Occam’s razor, applied to machine learning, suggests that simpler models are generally more useful and correct than more complicated models. That’s because simpler, more parsimonious models tend to generalize better.

Said another way, simpler, smaller models are less likely to *overfit* (fit too closely to the training data). Overfit models tend not to generalize well out of sample. That’s because during overfitting, the models pick up too much noise or random fluctuations using the training data, which hinders performance on data the model has never seen before.



Underfitting refers to the opposite case — the scenario where the model is not learning enough of the true relationship underlying the data. Because overfitting is so common in real-world machine learning, interviewers commonly ask you how you can detect it, and what you can do to avoid it, which brings us to our next topic: regularization.

Regularization

Regularization aims to reduce the complexity of models. In relation to the bias-variance trade-off, regularization aims to decrease complexity in a way that significantly reduces variance while only slightly increasing bias. The most widely used forms of regularization are L1 and L2. Both methods add a simple penalty term to the objective function. The penalty helps shrink coefficients of features, which reduces overfitting. This is why, not surprisingly, they are also known as shrinkage methods.

Specifically, L1, also known as *lasso*, uses the absolute value of a coefficient to the objective function as a penalty. On the other hand, L2, also known as *ridge*, uses the squared magnitude of a coefficient to the objective function. The L1 and L2 penalties can also be linearly combined, resulting in the popular form of regularization called *elastic net*. Since having models overfit is a prevalent problem in machine learning, it's important to understand when to use each type of regularization. For example, L1 serves as a feature selection method, since many coefficients shrink to 0 (are zeroed out), and hence, are removed from the model. L2 is less likely to shrink any coefficients to 0. Therefore, L1 regularization leads to sparser models, and is thus considered a more strict shrinkage operation.

Interpretability & Explainability

In Kaggle competitions and classwork, you might be expected to maximize a model performance metric like accuracy. However, in the real world, rather than just maximizing a particular metric, you might also be responsible for explaining how your model came up with that output. For example, if your model predicts that someone shouldn't get a loan, doesn't that person deserve to know why? More broadly, interpretable models can help you identify biases in the model, which leads to more ethical AI. Plus, in some domains like healthcare, there can be deep auditing on decisions, and explainable models can help you stay compliant. However, there's usually a trade-off between performance and model interpretability. Often, using a more complex model might increase performance, but make results harder to interpret.

Various models have their own way of interpreting feature importance. For example, linear models have weights which can be visualized and analyzed to interpret the decision making. Similarly, random forests have feature importance readily available to identify what the model is using and learning. There are also some general frameworks that can help with more "black-box" models. One is *SHAP* (SHapley Additive exPlanation), which uses "Shapley" values to denote the average marginal contribution of a feature over all possible combinations of inputs. Another technique is *LIME* (Local Interpretable Model-agnostic Explanations), which uses sparse linear models built around various predictions to understand how any model performs in that local vicinity.

While it's rare to be asked about the details of SHAP and LIME during interviews, having a basic understanding of why model interpretability matters, and bringing up this consideration in more open-ended problems is key.

Model Training

We've covered frameworks to evaluate models, and selected the best-performing ones, but how do we actually train the model in the first place? If you don't master the art of model training (aka teaching machines to learn), even the best machine learning techniques will fail. Recall the basics: we first train models on a training dataset and then test the models on a testing dataset. Normally, 80% of the data will go towards training data, and 20% serves as the test set. But as we soon cover, there's much more to model training than the 80/20 train vs. test split.

Cross-Validation

Cross-validation assesses the performance of an algorithm in several subsamples of training data. It consists of running the algorithm on subsamples of the training data, such as the original data without some of the original observations, and evaluating model performance on the portion of the data that was excluded from the subsample. This process is repeated many times for the different subsamples, and the results are combined at the end.

Cross-validation helps you avoid training and testing on the same subsets of data points, which would lead to overfitting. As mentioned earlier, in cases where there isn't enough data or getting more data is costly, cross-validation enables you to have more faith in the quality and consistency of a model's test performance. Because of this, questions about how cross-validation works and when to use it are routinely asked in data science interviews.

One popular way to do cross-validation is called *k-fold cross-validation*. The process is as follows:

1. Randomly shuffle data into equally-sized blocks (folds).
2. For each fold k , train the model on all the data except for fold i , and evaluate the validation error using block i .
3. Average the k validation errors from step 2 to get an estimate of the true error.

Dataset

Estimation 1	Test 1	Train	Train	Train	Train
Estimation 2	Train	Test 2	Train	Train	Train
Estimation 3	Train	Train	Test 3	Train	Train
Estimation 4	Train	Train	Train	Test 4	Train
Estimation 5	Train	Train	Train	Train	Test 5

Example of 5-Fold Cross Validation

Another form of cross-validation you're expected to know for the interview is *leave-one-out cross-validation*. LOOCV is a special case of k -fold cross-validation where k is equal to the size of the dataset (n). That is, it is where the model is testing on every single data point during the cross-validation.

In the case of larger datasets, cross-validation can become computationally expensive, because every fold is used for evaluation. In this case, it can be better to use *train validation split*, where you split

the data into three parts: a training set, a dedicated validation set (also known as a “dev” set), and a test set. The validation set usually ranges from 10%-20% of the entire dataset.

An interview question that comes up from time to time is how to apply cross-validation for time-series data. Standard k -fold CV can’t be applied, since the time-series data is not randomly distributed but instead is already in chronological order. Therefore, you should not be using data “in the future” for predicting data “from the past.” Instead, you should use historical data up until a given point in time, and vary that point in time from the beginning till the end.

Bootstrapping and Bagging

The process of bootstrapping is simply drawing observations from a large data sample repeatedly (sampling with replacement) and estimating some quantity of a population by averaging estimates from multiple smaller samples. Besides being useful in cases where the dataset is small, bootstrapping is also useful for helping deal with class imbalance: for the classes that are rare, we can generate new samples via bootstrapping.

Another common application of bootstrapping is in ensemble learning: the process of averaging estimates from many smaller models into a main model. Each individual model is produced using a particular sample from the process. This process of bootstrap aggregation is also known as *bagging*. Later in this chapter, we’ll show concrete examples of how random forests utilize bootstrapping and bagging.

Ensemble methods like random forests, AdaBoost, and XGBoost are industry favorites, and as such, interviewers tend to ask questions about bootstrapping and ensemble learning. For example, one of the most common interview questions is: “What is the difference between XGBoost and a random forest?”

Hyperparameter Tuning

Hyperparameters are important because they impact a model’s training time, compute resources needed (and hence cost), and, ultimately, performance. One popular method for tuning hyperparameters is *grid search*, which involves forming a grid that is the Cartesian product of those parameters and then sequentially trying all such combinations and seeing which yields the best results. While comprehensive, this method can take a long time to run since the cost increases exponentially with the number of hyperparameters. Another popular hyperparameter tuning method is *random search*, where we define a distribution for each parameter and randomly sample from the joint distribution over all parameters. This solves the problem of exploring an exponentially increasing search space, but is not necessarily guaranteed to achieve an optimal result.

While not generally asked about in data science interviews for research scientist or machine learning engineering roles, hyperparameter tuning techniques such as the methods mentioned earlier, along with Bayesian hyperparameter optimization, might be brought up. This discussion mostly happens in the context of neural networks, random forests, or XGBoost. For interviews, you should be able to list a couple of the hyperparameters for your favorite modeling technique, along with what impacts they have on generalization.

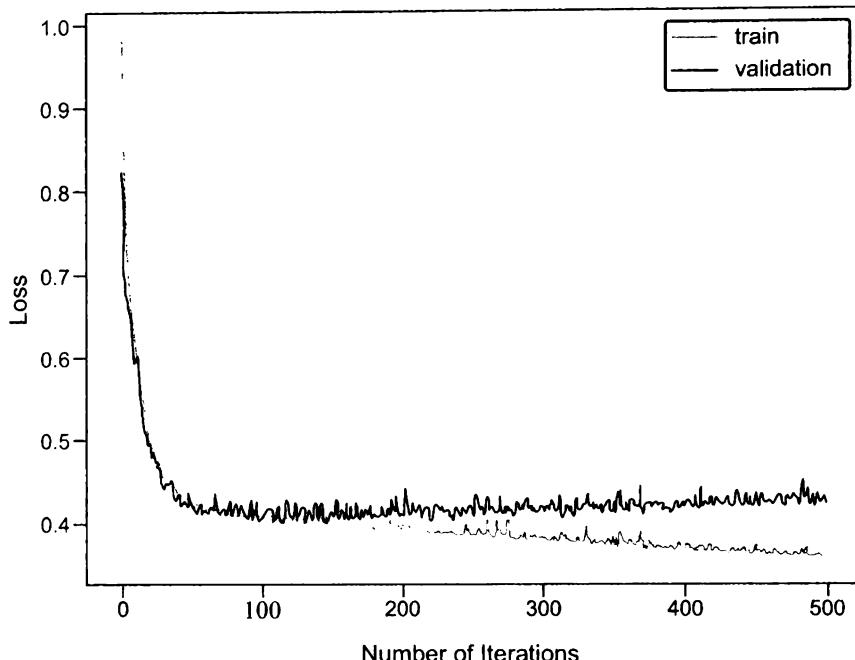
Training Times and Learning Curves

Training time is another factor to consider when it comes to model selection, especially for exceedingly large datasets. As we explain later in the coding chapter, it’s possible to use big-O notation to clarify the theoretical bounds on training time for each algorithm. These training time estimates are based on the number of data points and the dimensionality of the data.

For real-life training ML models, you should also factor in training time considerations and resource constraints during model selection. While you can always train more complex models that might

achieve marginally higher model performance metrics, the trade-off versus increased resource usage and training time might make such a decision suboptimal.

Learning curves are plots of model learning performance over time. The y -axis is some metric of learning (for example, classification accuracy), and the x -axis is experience (time).



A popular data science interview question involving learning curves is “How would you identify if your model was overfitting?” By analyzing the learning curves, you should be able to spot whether the model is underfitting or overfitting. For example, above, you can see that as the number of iterations is increasing, the training error is getting better. However, the validation error is not improving — in fact, it is increasing at the end — a clear sign that the model is overfitting and training can be stopped. Additionally, learning curves should help you discover whether a dataset is representative or not. If the data was not representative, the plot would show a large gap between the training curve and validation curve, which doesn’t get smaller even as training time increases.

Linear Regression

Linear regression is a form of *supervised learning*, where a model is trained on labeled input data. Linear regression is one of the most popular methods employed in machine learning and has many real-life applications due to its quick runtime and interpretability. That’s why there’s the joke about regression to regression: where you try to solve a problem with more advanced methods but end up falling back to tried and true linear regression.

As such, linear regression questions are asked in all types of data science and machine learning interviews. Essentially, interviewers are trying to make sure your knowledge goes beyond importing linear regression from scikit-learn and then blindly calling `linear_regression.fit(X, Y)`. That’s why deep knowledge of linear regression — understanding its assumptions, addressing edge cases that come up in real-life scenarios, and knowing the different evaluation metrics — will set you apart from other candidates.

In linear regression, the goal is to estimate a function $f(x)$, such that each feature has a linear relationship to the target variable y , or:

$$y = X\beta$$

where X is a matrix of predictor variables and β is a vector of parameters that determines the weight of each variable in predicting the target variable. So, how do you compare the performance of two linear regression models?

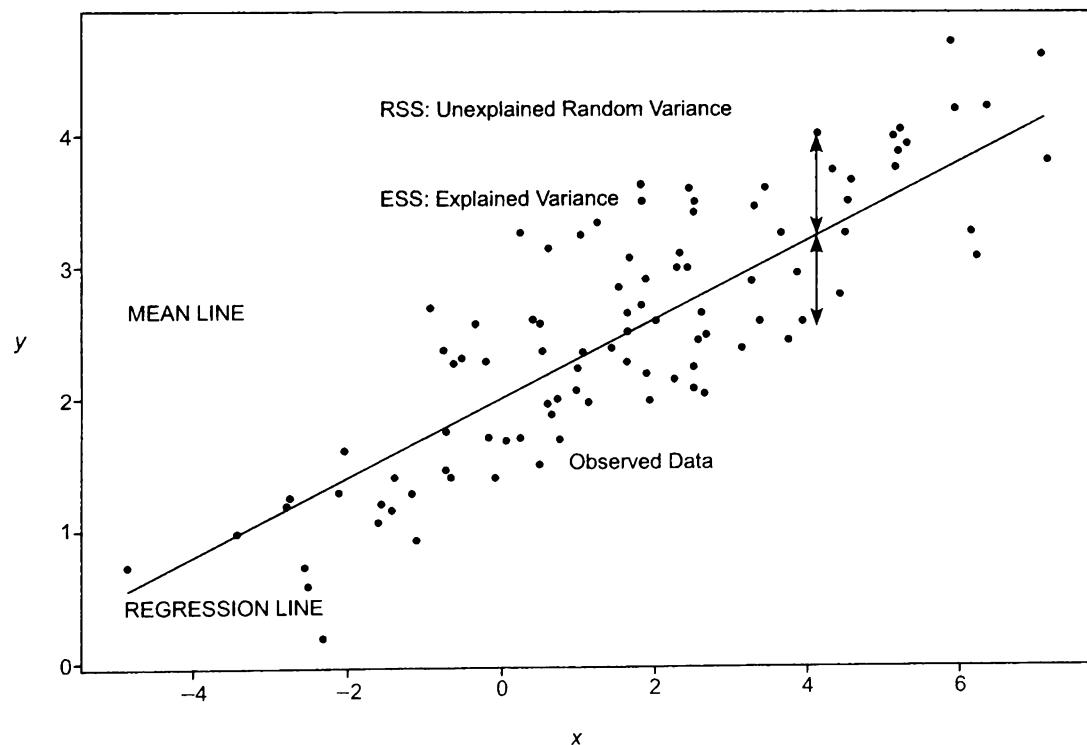
Evaluating Linear Regression

Evaluation of a regression model is built on the concept of a *residual*: the distance between what the model predicted versus the actual value. Linear regression estimates β by minimizing the residual sum of squares (RSS), which is given by the following:

$$RSS(\beta) = (y - X\beta)^T(y - X\beta)$$

Two other sum of squares concepts to know besides the RSS are the total sum of squares (TSS) and explained sum of squares (ESS). The total sum of squares is the combined variation in the data ($ESS + RSS$). The explained sum of squares is the difference between TSS and RSS. R^2 , a popular metric for assessing goodness-of-fit, is given by $R^2 = 1 - \frac{RSS}{TSS}$. It ranges between zero and one, and represents the proportion of variability in the data explained by the model. Other prominent error metrics to measure the goodness-of-fit of linear regression are MSE (mean squared error) and MAE (mean absolute error). MSE measures the *variance* of the residuals, whereas MAE measures the *average* of the residuals; hence, MSE penalizes larger errors more than MAE, making it more sensitive to outliers.

Actual vs Predicted Values from the Dummy Dataset



A common interview question is “What’s the expected impact on R^2 when adding more features to a model?” While adding more features to a model always increases the R^2 , that doesn’t necessarily make for a better model. Since any machine learning model can overfit by having more parameters, a goodness-of-fit measure like R^2 should likely also be assessed with model complexity in mind. Metrics that take into account the number of features of linear regression models include AIC, BIC, Mallow’s CP, and adjusted R^2 .

Subset Selection

So, how do you reduce model complexity of a regression model? *Subset selection*. By default, we use all the predictors in a linear model. However, in practice, it’s important to narrow down the number of features, and only include the most important features. One way is *best subset selection*, which tries each model with k predictors, out of p possible ones, where $k < p$. Then, you choose the best subset model using a regression metric like R^2 . While this guarantees the best result, it can be computationally infeasible as p increases (due to the exponential number of combinations to try). Additionally, by trying every option in a large search space, you’re likely to get a model that overfits with a high variance in coefficient estimates.

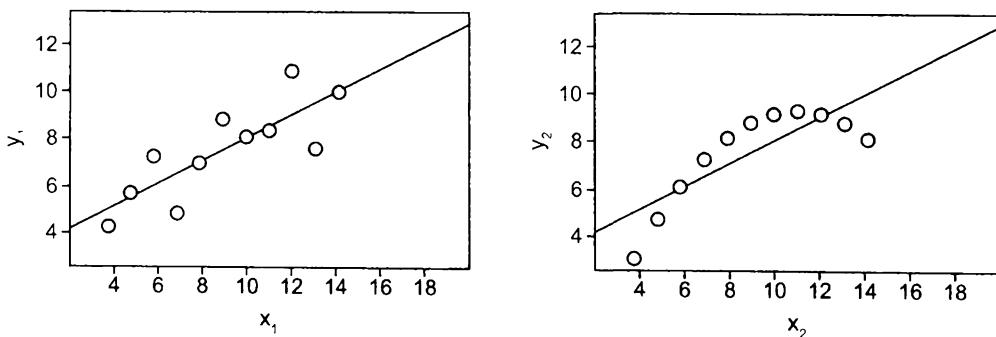
Therefore, an alternative is to use *stepwise selection*. In forward stepwise selection, we start with an empty model and iteratively add the most useful predictor. In backward stepwise selection, we start with the full model and iteratively remove the least useful predictor. While doing stepwise selection, we aim to find a model with high R^2 and low RSS, while considering the number of predictors using metrics like AIC or adjusted R^2 .

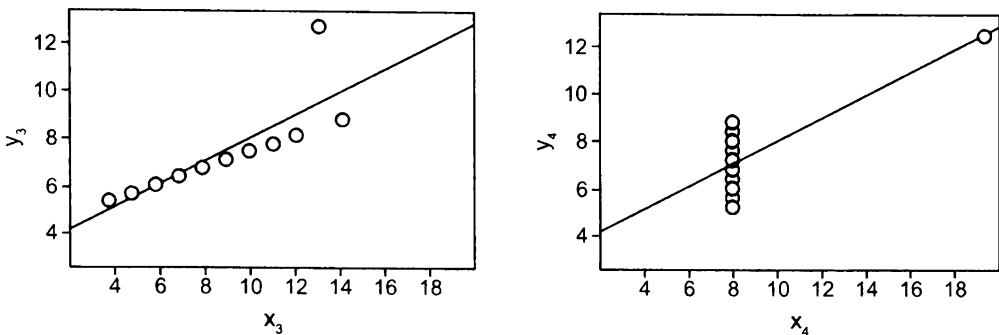
Linear Regression Assumptions

Because linear regression is one of the most commonly applied models, it has the honor of also being one of the most misapplied models. Before you can use this technique, you must validate its four main assumptions to prevent erroneous results:

- **Linearity:** The relationship between the feature set and the target variable is linear.
- **Homoscedasticity:** The variance of the residuals is constant.
- **Independence:** All observations are independent of one another.
- **Normality:** The distribution of Y is assumed to be normal.

These assumptions are crucial to know. For example, in the figures that follows, there are four lines of best fit that are the same. However, only in the top left dataset are these four assumptions met.



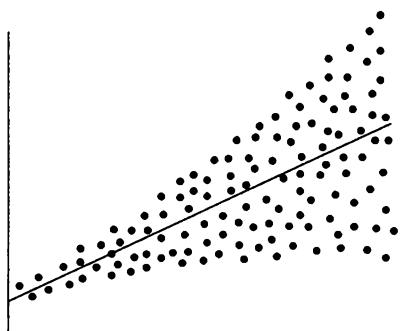


Note: for the independence and normality assumption, use of the term “i.i.d.” (independent and identically distributed) is also common. If any of these assumptions are violated, any forecasts or confidence intervals based on the model will most likely be misleading or biased. As a result, the linear regression model will likely perform poorly out of sample.

Avoiding Linear Regression Pitfalls

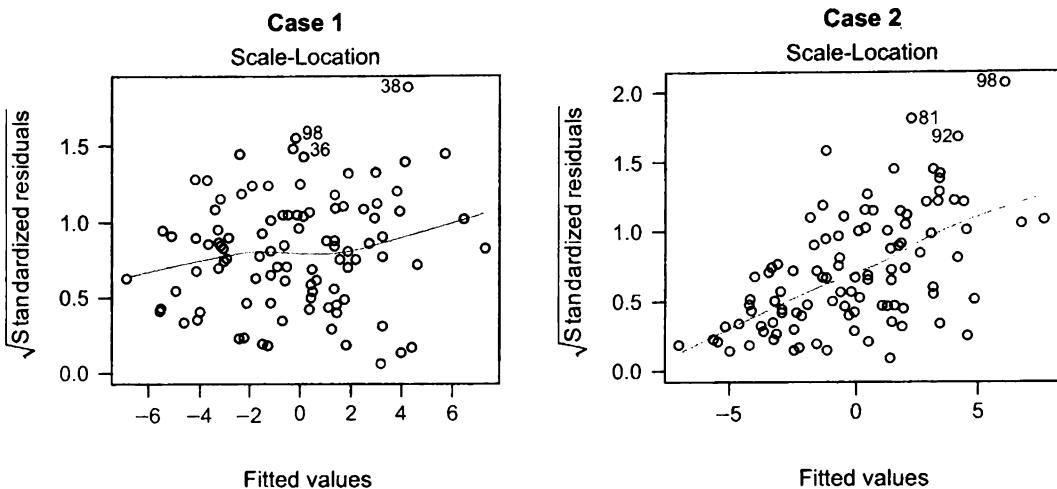
Heteroscedasticity

Linear regression assumes that the residuals (the distance between what the model predicted versus the actual value) are identically distributed. If the variance of the residuals is not constant, then *heteroscedasticity* is most likely present, meaning that the residuals are not identically distributed. To find heteroscedasticity, you can plot the residuals versus the fitted values. If the relationship between residuals and fitted values has a nonlinear pattern, this indicates that you should try to transform the dependent variable or include nonlinear terms in the model.



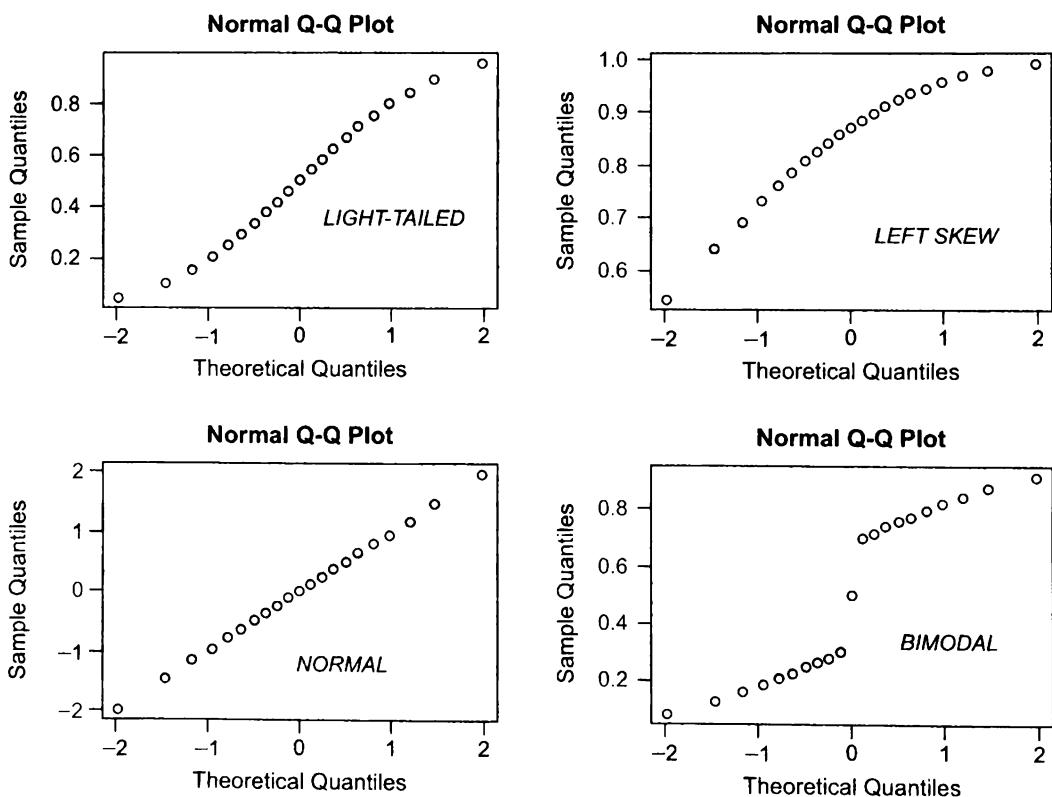
Example of heteroscedasticity. As the x-value increases, the residuals increase too

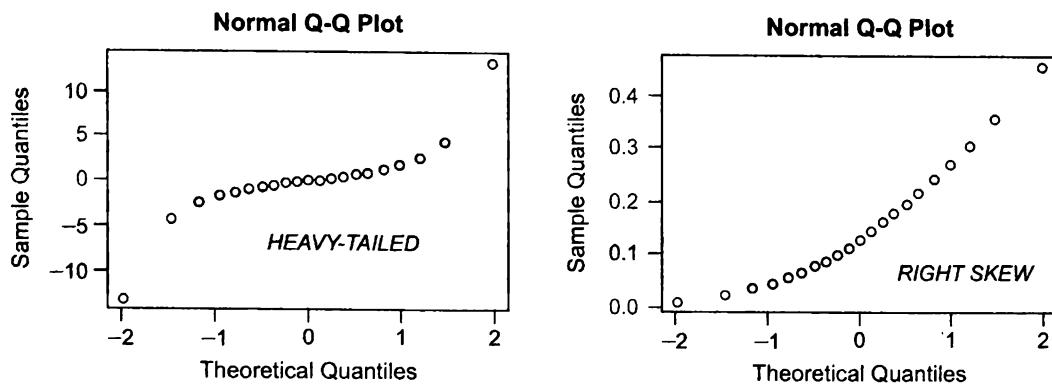
Another useful diagnostic plot is the scale-location plot, which plots standardized residuals versus the fitted values. If the data shows heteroscedasticity, then you will not see a horizontal line with equally spread points.



Normality

Linear regression assumes the residuals are normally distributed. We can test this through a QQ plot. Also known as a quantile plot, a QQ plot graphs the standardized residuals versus theoretical quantiles and shows whether the residuals appear to be normally distributed (i.e., the plot resembles a straight line). If the QQ plot is not a reasonably straight line, this is a sign that the residuals are not normally distributed, and hence, the model should be reexamined. In that case, transforming the dependent variable (with a log or square-root transformation, for example) can help reduce skew.





Outliers

Outliers can have an outsized impact on regression results. There are several ways to identify outliers. One of the more popular methods is examining *Cook's distance*, which is the estimate of the influence of any given data point. Cook's distance takes into account the residual and leverage (how far away the X value differs from that of other observations) of every point. In practice, it can be useful to remove points with a Cook's distance value above a certain threshold.

Multicollinearity

Another pitfall is if the predictors are correlated. This phenomenon, known as *multicollinearity*, affects the resulting coefficient estimates by making it problematic to distinguish the true underlying individual weights of variables. Multicollinearity is most commonly observed by weights that flip magnitude. It is one of the reasons why model weights cannot be directly interpreted as the importance of a feature in linear regression. Features that initially would appear to be independent variables can often be highly correlated: for example, the number of Instagram posts made and the number of notifications received are most likely highly correlated, since both are related to user activity on the platform, and one generally causes another.

One way to assess multicollinearity is by examining the variance inflation factor (VIF), which quantifies how much the estimated coefficients are inflated when multicollinearity exists. Methods to address multicollinearity include removing the correlated variables, linearly combining the variables, or using PCA/PLS (partial least squares).

Confounding Variables

Multicollinearity is an extreme case of *confounding*, which occurs when a variable (but not the main independent or dependent variables) affects the relationship between the independent and dependent variables. This can cause invalid correlations. For example, say you were studying the effects of ice cream consumption on sunburns and find that higher ice cream consumption leads to a higher likelihood of sunburn. That would be an incorrect conclusion because temperature is the confounding variable — higher summer temperatures lead to people eating more ice cream and also spending more time outdoors (which leads to more sunburn).

Confounding can occur in many other ways, too. For example, one way is *selection bias*, where the data are biased due to the way they were collected (for example, group imbalance). Another problem, known as *omitted variable bias*, occurs when important variables are omitted, resulting in a linear regression model that is biased and inconsistent. Omitted variables can stem from dataset generation issues or choices made during modeling. A common way to handle confounding is stratification, a

process where you create multiple categories or subgroups in which the confounding variables do not vary much, and then test significance and strength of associations using chi square.

Knowing about these regression edge cases, how to identify them, and how to guard against them is crucial. This knowledge separates the seasoned data scientists from the data neophyte — precisely why it's such a popular topic for data science interviews.

Generalized Linear Models

In linear regression, the residuals are assumed to be normally distributed. The generalized linear model (GLM) is a generalization of linear regression that allows for the residuals to not just be normally distributed. For example, if Tinder wanted to predict the number of matches somebody would get in a month, they would likely want to use a GLM like the one below with a Poisson response (called Poisson regression) instead of a standard linear regression. The three common components to any GLM are:

Link Function	Systematic Component	Random Component
$\ln \lambda_i$	$= b_0 + b_1 x_i$	$+ \varepsilon$
y_i	\sim	Poisson (λ_i)

- **Random Component:** is the distribution of the error term, i.e., normal distribution for linear regression.
- **Systematic Component:** consists of the explanatory variables, i.e., the predictors combined in a linear combination.
- **Link function:** is the link between the random and system components, i.e., a linear regression, logit regression, etc.

Note that in GLMs, the response variable is still a linear combination of weights and predictors.

Regression can also use the weights and predictors nonlinearly; the most common examples of this are polynomial regressions, splines, and general additive models. While interesting, these techniques are rarely asked about in interviews and thus are beyond the scope of this book.

Classification

General Framework

Interview questions related to classification algorithms are commonly asked during interviews due to the abundance of real-life applications for assigning categories to things. For example, classifying users as likely to churn or not, predicting whether a person will click on an ad or not, and distinguishing fraudulent transactions from legitimate ones are all applications of the classification techniques we mention in this section.

The goal of classification is to assign a given data point to one of K possible classes instead of calculating a continuous value (as in regression). The two types of classification models are *generative models* and *discriminative models*. Generative models deal with the joint distribution of X and Y , which is defined as follows:

$$p(X, Y) = p(Y|X)p(X)$$

Maximizing a posterior probability distribution produces decision boundaries between classes where the resulting posterior probability is equivalent. The second type of model is discriminative. It directly determines a decision boundary by choosing the class that maximizes the probability:

$$\hat{y} = \arg \max_k p(Y = k|x)$$

Thus, both methods choose a predicted class that maximizes the posterior probability distribution; the difference is simply the approach. While traditional classification deals with just two classes (0 or 1), multi-class classification is common, and many of the below methods can be adapted to handle multiple labels.

Evaluating Classifiers

Before we detail the various classification algorithms like logistic regression and Naive Bayes, it's essential to understand how to evaluate the predictive power of a classification model.

Say you are trying to predict whether an individual has a rare cancer that only happens to 1 in 10,000 people. By default, you could simply predict that every person doesn't have cancer and be accurate 99.99% of the time. But clearly, this isn't a helpful model — Pfizer won't be acquiring our diagnostic test anytime soon! Given imbalanced classes, assessing accuracy alone is not enough — this is known as the "accuracy paradox" and is the reason why it's critical to look at other measures for misclassified observations.

Building and Interpreting a Confusion Matrix

When building a classifier, we want to minimize the number of misclassified observations, which in binary cases can be termed false positives and false negatives. In a false positive, the model incorrectly predicts that an instance belongs to the positive class. For the cancer detection example, a false positive would be classifying an individual as having cancer, when in reality, the person does not have it. On the other hand, a false negative occurs when the model incorrectly produces a negative class. In the cancer diagnostic case, this would mean saying a person doesn't have cancer, when in fact they do.

A *confusion matrix* helps organize and visualize this information. Each row represents the actual number of observations in a class, and each column represents the number of observations predicted as belonging to a class.

		Predicted		
		Positive	Negative	
Actual Class	Positive	True Positive (TP) Type 2 Error	False Negative (FN) Type 2 Error	Sensitivity $\frac{TP}{(TP + FN)}$
	Negative	False Positive (FP) Type 1 Error	True Negative (TN)	Specificity $\frac{TN}{(TN + FP)}$
	Precision $\frac{TP}{(TP + FP)}$	Negative Predictive Value $\frac{TN}{(TN + FN)}$	Accuracy $\frac{TP + TN}{(TP + TN + FP + FN)}$	

Precision and Recall

Two metrics that go beyond accuracy are *precision* and *recall*. In classification, precision is the actual positive proportion of observations that were predicted positive by the classifier. In the cancer

diagnostic example, it's the percentage of people you said would have cancer who actually ended up having the disease. Recall, also known as *sensitivity*, is the percentage of total positive cases captured, out of all positive cases. It's essentially how well you do in finding people with cancer.

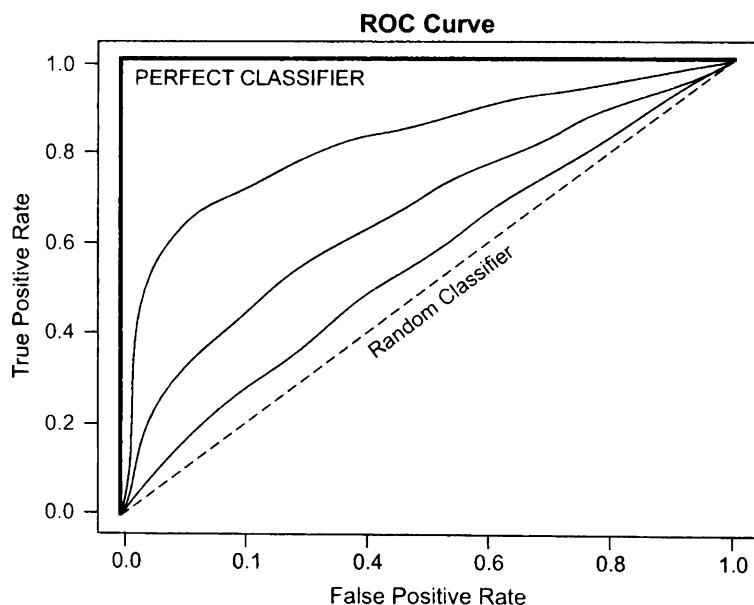
In real-world modeling, there's a natural trade-off between optimizing for precision or recall. For example, having high recall — catching most people who have cancer — ends up saving the lives of some people with the disease. However, this often leads to misdiagnosing others who didn't truly have cancer, which subjects healthy people to costly and dangerous treatments like chemotherapy for a cancer they never had. On the flip side, having high precision means being confident that when the diagnostic comes back positive, the person really has cancer. However, this often means missing some people who truly have the disease. These patients with missed diagnoses may gain a false sense of security, and their cancer, left unchecked, could lead to fatal outcomes.

During interviews, be prepared to talk about the precision versus recall trade-off. For open-ended case questions and take-home challenges, be sure to contextualize the business and product impact of a false positive or a false negative. In cases where both precision and recall are equally important, you can optimize the *F1 score*: the harmonic mean of precision and recall.

$$F_1 = 2 * \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}}$$

Visualizing Classifier Performance

Besides precision, recall, and the F1 score, another popular way to evaluate classifiers is the receiver operating characteristic (ROC) curve. The ROC curve plots the true positive rate versus the false positive rate for various thresholds. The area under the curve (AUC) measures how well the classifier separates classes. The AUC of the ROC curve is between zero and one, and a higher number means the model performs better in separating the classes. The most optimal is a curve that "hugs" the top left of the plot, as shown below. This indicates that a model has a high true-positive rate and relatively low false-positive rate.



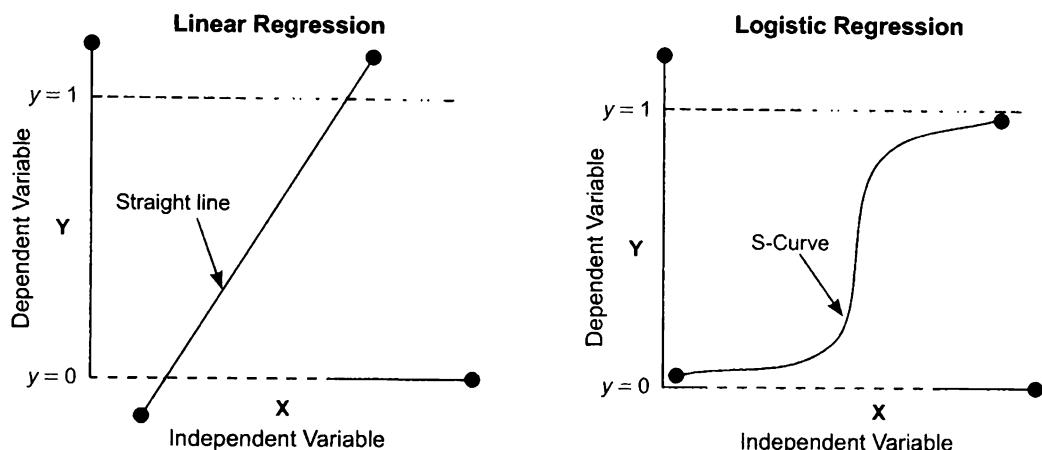
Logistic Regression

One of the most popular classification algorithms is logistic regression, and it is asked about almost as frequently as linear regression during interviews. In logistic regression, a linear output is converted into a probability between 0 and 1 using the sigmoid function:

$$S(x) = \frac{1}{1 + e^{-x\beta}}$$

In the equation above, X is the set of predictor features and β is the corresponding vector of weights. Computing $S(x)$ above produces a probability that indicates if an observation should be classified as a “1” (if the calculated probability is at least 0.5), and a “0” otherwise.

$$P(\hat{Y} = 1 | X) = S(X\beta)$$



The loss function for logistic regression, also known as log-loss, is formulated as follows:

$$L(w) = \sum_{i=1}^n y_i \log\left(\frac{1}{S(X\beta)}\right) + (1 - y_i) \log\left(\frac{1}{1 - S(X\beta)}\right)$$

Note that in cases where more than two outcome classes exist, softmax regression is a commonly used technique that generalizes logistic regression.

In practice, logistic regression, much like its cousin linear regression, is often used because it is highly interpretable: its output, a predicted probability, is easy to explain to decision makers. Additionally, its quickness to compute and ease of use often make it the first model employed for classification problems in a business context.

Note, however, that logistic regression does not work well under certain circumstances. Its relative simplicity makes it a high-bias and low-variance model, so it may not perform well when the decision boundary is not linear. Additionally, when features are highly correlated, the coefficients β won't be as accurate. To address these cases, you can use techniques similar to those used in linear regression (regularization, removal of features, etc.) for dealing with this issue. For interviews, it is critical to understand both the mechanics and pitfalls of logistic regression.

Naive Bayes

Naive Bayes classifiers require only a small amount of training data to estimate the necessary parameters. They can be extremely fast compared to more sophisticated methods (such as support

vector machines). These advantages lead to Naive Bayes being a popularly used first technique in modeling, and is why this type of classifier shows up in interviews.

Naive Bayes uses Bayes' rule (covered in Chapter 6: Statistics) and a set of conditional independence assumptions in order to learn $P(Y|X)$. There are two assumptions to know about Naive Bayes:

1. It assumes each X_i is independent of any other X_j given Y for any pair of features X_i and X_j .
2. It assumes each feature is given the same weight.

The decoupling of the class conditional feature distributions means that each distribution can be independently estimated as a one-dimensional distribution. That is, we have the following:

$$P(X_1 \dots X_n | Y) = \prod_{i=1}^n P(X_i | Y)$$

Using the conditional independence assumption, and then applying Bayes' theorem, the classification rule becomes:

$$\hat{y} = \arg \max_{y_i} P(Y = y_i) \prod_j P(X_j | Y = y_i)$$

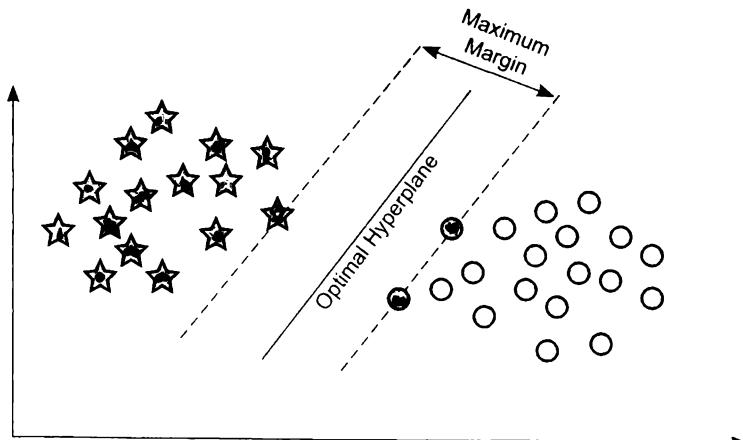
To understand the beauty of Naive Bayes, recall that for any ML model having k features, there are 2^k possible feature interactions (the correlations between them all). Due to the large number of feature interactions, typically you'd need 2^k data points for a high-performing model. However, due to the conditional independence assumption in Naive Bayes, there only need to be k data points, which removes this problem.

For text classification (e.g., classifying spam, sentiment analysis), this assumption is convenient since there are many predictors (words) that are generally independent of one another.

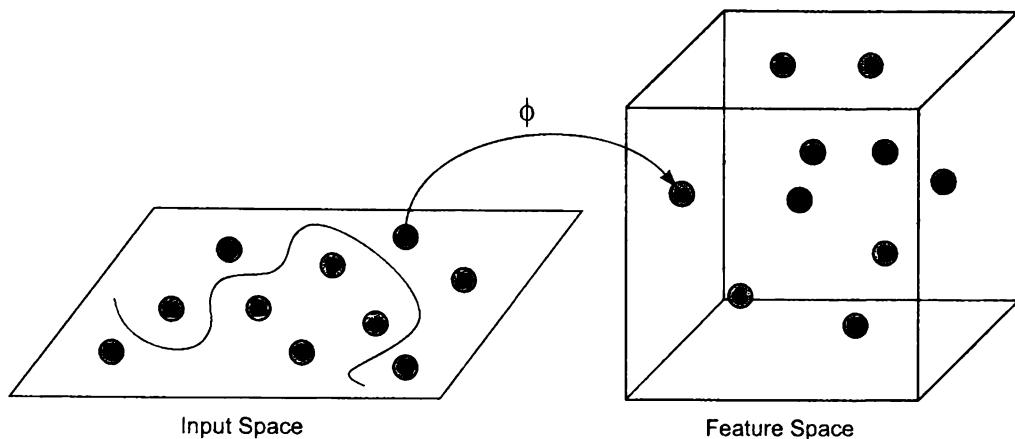
While the assumptions simplify calculations and make Naive Bayes highly scalable to run, they are often not valid. In fact, the first conditional independence assumption generally never holds true, since features do tend to be correlated. Nevertheless, this technique performs well in practice since most data is linearly separable.

SVMs

The goal of SVM is to form a hyperplane that linearly separates the training data. Specifically, it aims to maximize the margin, which is the minimum distance from the decision boundary to any training point. The points closest to the hyperplane are called the support vectors. Note that the decision boundaries for SVMs can be nonlinear, which is unlike that of logistic regression, for example.



In the image above, it's easy to visualize how a line can be found that separates the points correctly into their two classes. In practice, splitting the points isn't that straightforward. Thus, SVMs rely on a kernel to transform data into a higher-dimensional space, where it then finds the hyperplane that best separates the points. The image below visualizes this kernel transformation:



Mathematically, the kernel generalizes the dot product to a higher dimension:

$$k(x, y) = \phi(x)^T \phi(y)$$

The RBF (radial basis function) and Gaussian kernels are the two most popular kernels used in practice. The general rule of thumb is this: for linear problems, use a linear kernel, and for nonlinear problems, use a nonlinear kernel like RBF. SVMs can be viewed as a kernelized form of ridge regression because they modify the loss function employed in ridge regression.

SVMs work well in high-dimensional spaces (a larger number of dimensions versus the number of data points) or when a clear hyperplane divides the points. Conversely, SVMs don't work well on enormous data sets, since computational complexity is high, or when the target classes overlap and there is no clean separation. Compared to simpler methods with linear decision boundaries such as logistic regression and Naive Bayes, you may want to use SVMs if you have nonlinear decision boundaries and/or a much smaller amount of data. However, if interpretability is important, SVMs are not preferred because they do not have simple-to-understand outputs (like logistic regression does with a probability).

For ML-heavy roles, know when to use which kernel, the kernel trick, and the underlying optimization problem that SVMs solve.

Decision Trees

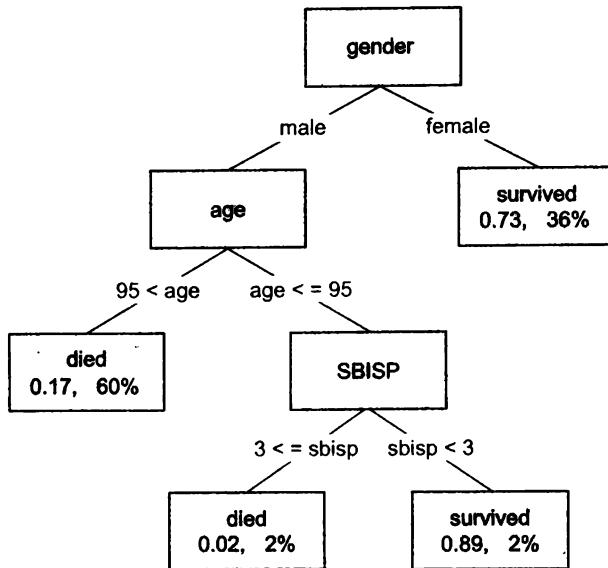
Decision trees and random forests are commonly discussed during interviews since they are flexible and often perform well in practice for both classification and regression use cases. Since both use cases are possible, decision trees are also known as CART (classification and regression trees). For this section, we'll focus on the classification use case for decision trees. While reading this section, keep in mind that for interviews, it helps to understand how both decision trees and random forests are trained. Related topics of entropy and information gain are also crucial to review before a data science interview.

Training

A decision tree is a model that can be represented in a treelike form determined by binary splits made in the feature space and resulting in various leaf nodes, each with a different prediction. Trees are

trained in a greedy and recursive fashion, starting at a root node and subsequently proceeding through a series of binary splits in features (i.e., variables) that lead to minimal error in the classification of observations.

Survival of Passengers on the Titanic



Entropy

The entropy of a random variable Y quantifies the uncertainty in Y . For a discrete variable Y (assuming k states) it is stated as follows:

$$H(Y) = -\sum_{i=1}^k P(y=i)\log P(Y=i)$$

For example, for a simple Bernoulli random variable, this quantity is highest when $p = 0.5$ and lowest when $p = 0$ or $p = 1$, a behavior that aligns intuitively with its definition since if $p = 0$ or 1 , then there is no uncertainty with respect to the result. Generally, if a random variable has high entropy, its distribution is closer to uniform than a skewed one. There are many measures of entropy — in practice, the Gini index is commonly used for decision trees.

In the context of decision trees, consider an arbitrary split. We have $H(Y)$ from the initial training labels and assume that we have some feature X on which we want to split. We can characterize the reduction in uncertainty given by the feature X , known as information gain, which can be formulated as follows:

$$IG(Y, X) = H(Y) - H(Y|X)$$

The larger $IG(Y, X)$ is, the higher the reduction in uncertainty in Y by splitting on X . Therefore, the general process assesses all features in consideration and chooses the feature that maximizes this information gain, then recursively repeats the process on the two resulting branches.

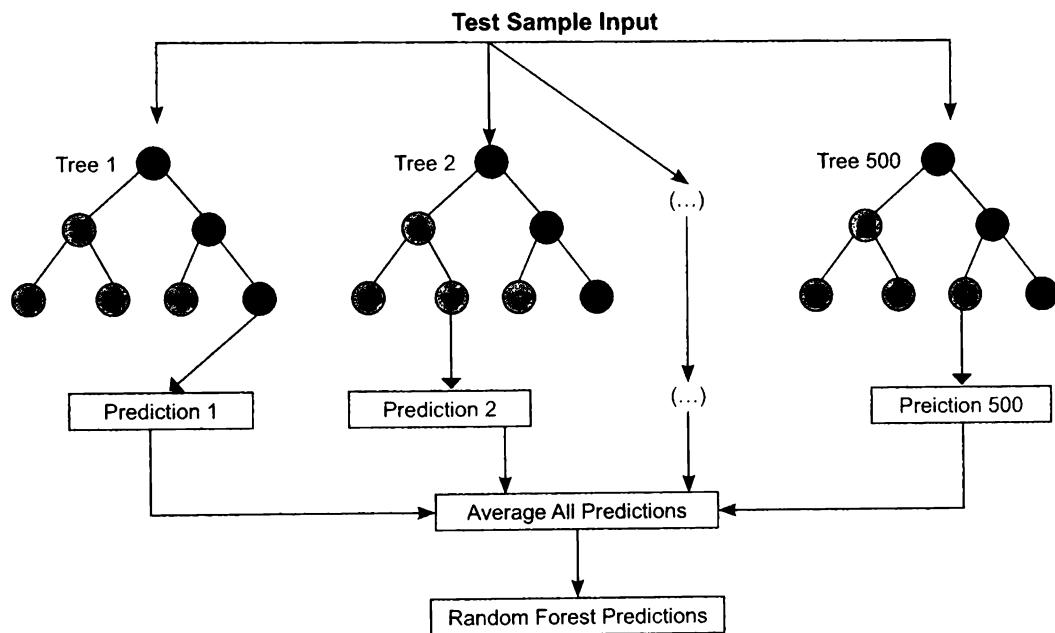
Random Forests

Typically, an individual decision tree may be prone to overfitting because a leaf node can be created for each observation. In practice, random forests yield better out-of-sample predictions than decision

trees. A random forest is an ensemble method that can utilize many decision trees, whose decisions it averages.

Two characteristics of random forests allow a reduction in overfitting and the correlation between the trees. The first is bagging, where individual decision trees are fitted following each bootstrap sample and then averaged afterwards. Bagging significantly reduces the variance of the random forest versus the variance of any individual decision trees. The second way random forests reduce overfitting is that a random subset of features is considered at each split, preventing the important features from always being present at the tops of individual trees.

Random forests are often used due to their versatility, interpretability (you can quickly see feature importance), quick training times (they can be trained in parallel), and prediction performance. In interviews, you'll be asked about how they work versus a decision tree, and when you would use a random forest over other techniques.



Boosting

Boosting is a type of ensemble model that trains a sequence of “weak” models (such as small decision trees), where each one sequentially compensates for the weaknesses of the preceding models. Such weaknesses can be measured by the current model’s error rate, and the relative error rates can be used to weigh which observations the next models should focus on. Each training point within a dataset is assigned a particular weight and is continually re-weighted in an iterative fashion such that points that are mispredicted take on higher weights in each iteration. In this way, more emphasis is placed on points that are harder to predict. This can lead to overfitting if the data is especially noisy.

One example is *AdaBoost* (adaptive boosting), which is a popular technique used to train a model based on tuning a variety of weak learners. That is, it sequentially combines decision trees with a single split, and then weights are uniformly set for all data points. At each iteration, data points are re-weighted according to whether each was classified correctly or incorrectly by a classifier. At the end, weighted predictions of each classifier are combined to obtain a final prediction.

The generalized form of AdaBoost is called gradient boosting. A well-known form of gradient boosting used in practice is called XGBoost (extreme gradient boosting). Gradient boosting is similar to AdaBoost, except that shortcomings of previous models are identified by the gradient rather than high weight points, and all classifiers have equal weights instead of having different weights. In industry, XGBoost is used heavily due to its execution speed and model performance.

Since random forests and boosting are both ensemble methods, interviewers tend to ask questions comparing and contrasting the two. For example, one of the most common interview questions is “What is the difference between XGBoost and a random forest?”

Dimensionality Reduction

Imagine you have a dataset with one million rows but two million features, most of which are null across the data points. You can intuitively guess that it would be hard to tease out which features are predictive for the task at hand. In geometric terms, this situation demonstrates sparse data spread over multiple dimensions, meaning that each data point is relatively far away from other data points. This lack of distance is problematic, because when extracting patterns using machine learning, the idea of similarity or closeness of data often matters a great deal. If a particular data point has nothing close to it, how can an algorithm make sense of it?

This phenomenon is known as the *curse of dimensionality*. One way to address this problem is to increase the dataset size, but often, in practice, it’s costly or infeasible to get more training data. Another way is to conduct feature selection, such as removing multicollinearity, but this can be challenging with a very large number of features.

Instead, we can use *dimensionality reduction*, which reduces the complexity of the problem with minimal loss of important information. Dimensionality reduction enables you to extract useful information from such data, but can sometimes be difficult or even too expensive, since the algorithm we would use would incorporate so many features. Decomposing the data into a smaller set of variables is also useful for summarizing and visualizing datasets. For example, dimensionality reduction methods can be used to project a large dataset into 2D or 3D space for easier visualization.

Principal Components Analysis

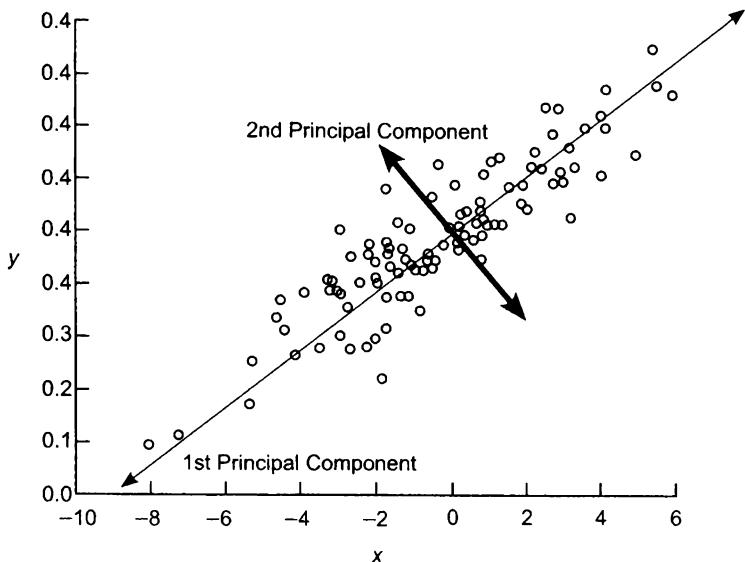
The most commonly used method to reduce the dimensionality of a dataset is principal components analysis (PCA). PCA combines highly correlated variables into a new, smaller set of constructs called *principal components*, which capture most of the variance present in the data. The algorithm looks for a small number of linear combinations for each row vector to explain the variance within X . For example, in the image below, the variation in the data is largely summarized by two principal components.

More specifically, PCA finds the vector w of weights such that we can define the following linear combination:

$$y_i = w^T x = \sum_{j=1}^p w_j x_j$$

subject to the following:

y_i is uncorrelated with y_j , $\text{var}(y_i)$ is maximized



Hence, the algorithm proceeds by first finding the component having maximal variance. Then, the second component found is uncorrelated with the first and has the second-highest variance, and so on for the other components. The algorithm ends with some number k dimensions such that

$$y_1, \dots, y_k \text{ explain the majority of } k \text{ variance, } k \ll p$$

The final result is an eigendecomposition of the covariance matrix of X , where the first principal component is the eigenvector corresponding to the largest eigenvalue and the second principal component corresponds to the eigenvector with the second largest eigenvalue, and so on. Generally, the number of components you choose is based on your threshold for the percent of variance your principal components can explain. Note that while PCA is a linear dimensionality reduction method, t-distributed stochastic neighbor embedding (t-SNE) is a non-linear, non-deterministic method used for data visualization.

In interviews, PCA questions often test your knowledge of the assumptions (like that the variables need to have a linear relationship). Commonly asked about as well are pitfalls of PCA, like how it struggles with outliers, or how it is sensitive to the units of measurement for the input features (data should be standardized). For more ML-heavy roles, you may be asked to whiteboard the eigendecomposition.

Clustering

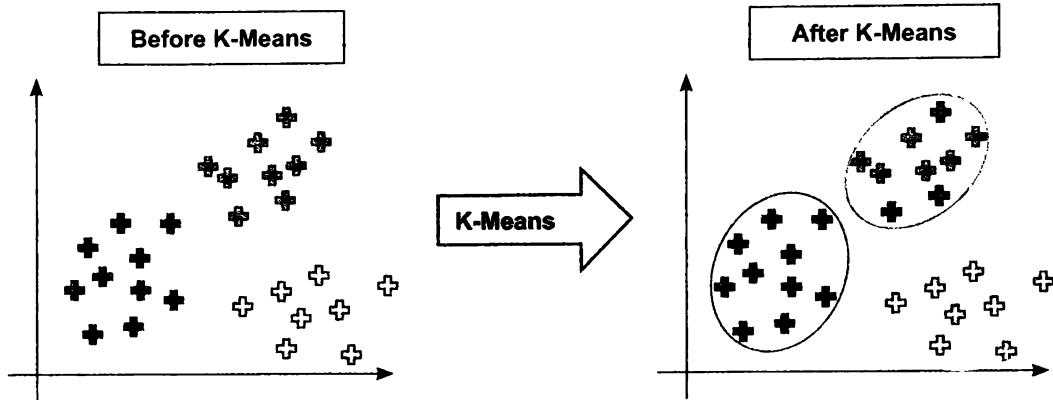
Clustering is a popular interview topic since it is the most commonly employed unsupervised machine learning technique. Recall that unsupervised learning means that there is no labeled training data, i.e., the algorithm is trying to infer structural patterns within the data, without a prediction task in mind. Clustering is often done to find “hidden” groupings in data, like segmenting customers into different groups, where the customers in a group have similar characteristics. Clustering can also be used for data visualization and outlier identification, as in fraud detection, for instance. The goal of clustering is to partition a dataset into various clusters or groups by looking only at the data’s input features.

Ideally, the clustered groups have two properties:

- Points within a given cluster are similar (i.e., high intra-cluster similarity).
- Points in different clusters are not similar (i.e., low inter-cluster similarity).

K-Means clustering

A well-known clustering algorithm, k -means clustering is often used because it is easy to interpret and implement. It proceeds, first, by partitioning a set of data into k distinct clusters and then arbitrarily selects centroids of each of these clusters. It iteratively updates partitions by first assigning points to the closest cluster, then updating centroids, and then repeating this process until convergence. This process essentially minimizes the total inter-cluster variation across all clusters.



Mathematically, k -means clustering reaches a solution by minimizing a loss function (also known as distortion function). In this example, we minimize Euclidean distance (given x_i points and centroid value μ_j):

$$L = \sum_{j=1}^k \sum_{x \in S_j} \|x - \mu_j\|^2$$

where S_j represents the particular cluster.

The iterative process continues until the cluster assignment updates fail to improve the objective function. Note that k -means clustering uses Euclidean distance when assessing how close points are to one another and that k , the number of clusters to be estimated, is set by the user and can be optimized if necessary.

K-means Alternatives

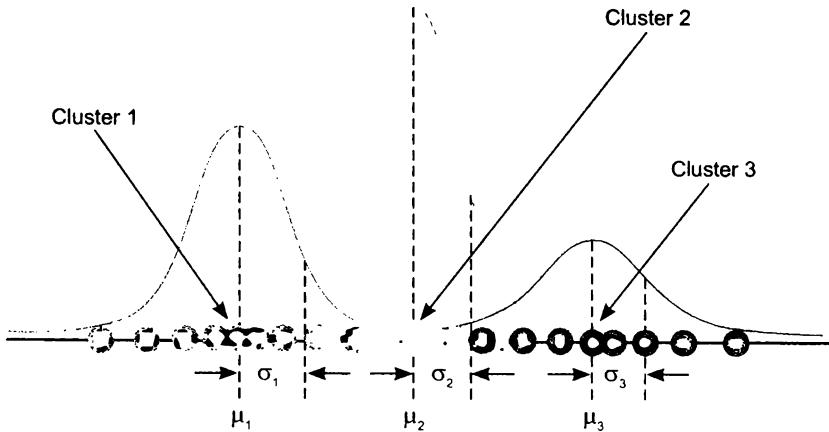
One alternative to k -means is *hierarchical clustering*. Hierarchical clustering assigns data points to their own cluster and merges clusters that are the nearest (based on any variety of distance metrics) until there is only one cluster left, generally visualized using a dendrogram. In cases where there is not a specific number of clusters, or you want a more interpretable and informative output, hierarchical clustering is more useful than k -means.

While quite similar to k -means, *density clustering* is another distinct technique. The most well-known implementation of this technique is DBSCAN. Density clustering does not require a number of clusters as a parameter. Instead, it infers that number, and learns to identify clusters of arbitrary shapes. Generally, density clustering is more helpful for outlier detection than k -means.

Gaussian Mixture Model (GMM)

A GMM assumes that the data being analyzed come from a “mixture” of k Gaussian/normal distributions, each having a different mean and variance, where the mixture components are basically the proportion of observations in each group. Compared to k -means, which is a deterministic algorithm where k is set in advance, GMMs essentially try to learn the true value of k .

For example, TikTok may be on the lookout for anomalous profiles, and can use GMMs to cluster various accounts based on features (number of likes sent, messages sent, and comments made) and identify any accounts whose activity metrics don't seem to fall within the typical user activity distributions.



Compared to k -means, GMMs are more flexible because k -means only takes into account the mean of a cluster, while GMMs take into account the mean and variance. Therefore, GMMs are particularly useful in cases with low-dimensional data or where cluster shapes may be arbitrary. While practically never asked about for data science interviews (compared to k -means), we brought up GMMs for those seeking more technical ML research and ML engineering positions.

Neural Networks

While the concepts behind neural networks have been around since the 1950s, it's only in the last 15 years that they've grown in popularity, thanks to an explosion of data being created, along with the rise of cheap cloud computing resources needed to store and process the massive amounts of newly created data. As mentioned earlier in the chapter, if your résumé has any machine learning projects involving deep learning experience, then the technical details behind neural networks will be considered fair game by most interviewers. But for a product data science position or a finance role (where data can be very noisy, so most models are not purely neural networks), don't expect to be bombarded with tough neural network questions. Knowing the basics of classical ML techniques should suffice.

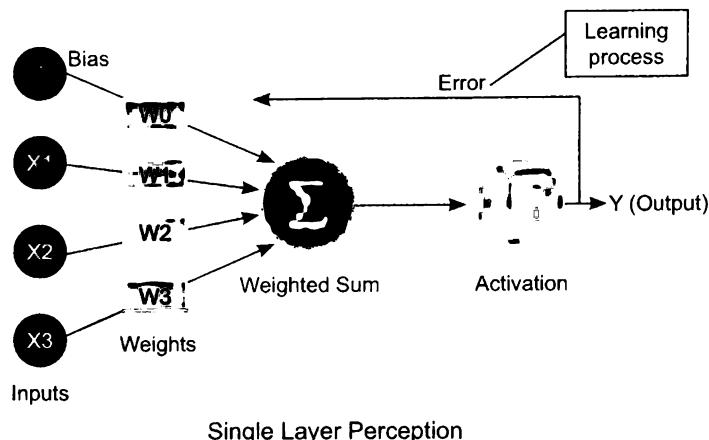
When neural nets are brought up during interviews, questions can range anywhere from qualitative assessments on how deep learning compares to more traditional machine learning models to mathematical details on gradient descent and backpropagation. On the qualitative side, it helps to understand all of the components that go into training neural networks, as well as how neural networks compare to simpler methods.

Perceptron

Neural networks function in a way similar to biological neurons. They take in various inputs (at input layers), weight these inputs, and then combine the weighted inputs through a linear combination (much like linear regression). If the combined weighted output is past some threshold set by an *activation function*, the output is then sent out to other layers. This base unit is generally referred to as

CHAPTER 7 : MACHINE LEARNING

a perceptron. Perceptrons are combined to form neural networks, which is why they are also known as *multi-layer perceptrons* (MLPs).

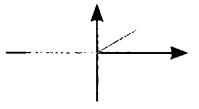


Single Layer Perception

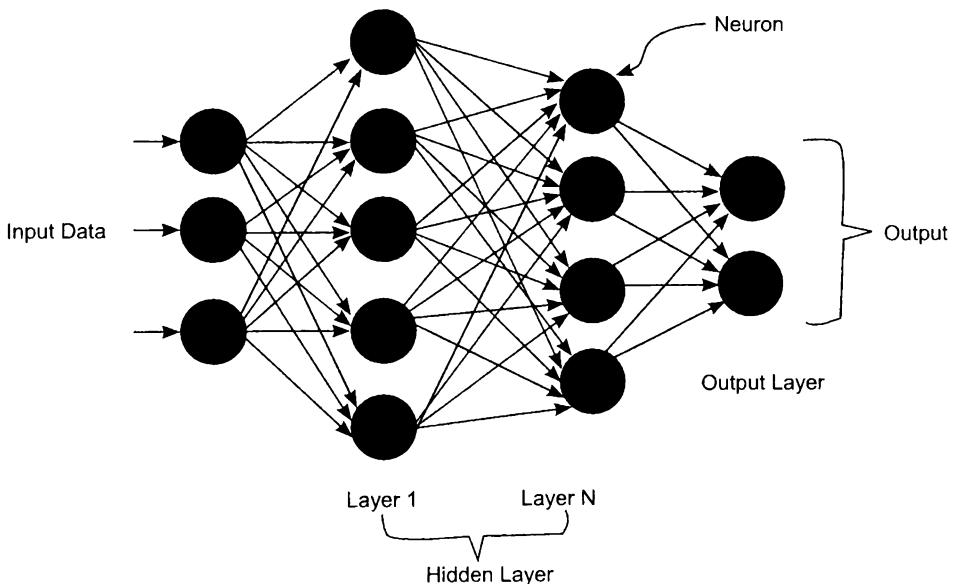
While the inputs for a neural network are combined via a linear combination, often, the activation function is nonlinear. Thus, the relationship between the target variable and the predictor features (variables) frequently ends up also being nonlinear. Therefore, neural networks are most useful when representing and learning nonlinear functions.

For reference, we include a list of common activation functions below. The scope of when to use which activation function is outside of this text, but any person interviewing for an ML-intensive role should know these use cases along with the activation function's formula.

Activation Function	Equation	Example	1D Graph
Unit Step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perception variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perception variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq 0, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1+e^{-z}}$	Logistic regression Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer Neural Network	

Rectifier, ReLU (Rectifier Linear Unit)	$\phi(z) = \max(0, z)$	Multi-layer Neural Network	
Rectifier, softplus	$\phi(z) = \ln(1 + e^z)$	Multi-layer Neural Network	

In neural networks, the process of receiving inputs and generating an output continues until an output layer is reached. This is generally done in a forward manner, meaning that layers process incoming data in a sequential forward way (which is why most neural networks are known as “feed-forward”). The layers of neurons that are not the input or output layers are called the *hidden layers* (hence the name “deep learning” for neural networks having many of these). Hidden layers allow for specific transformations of the data within each layer. Each hidden layer can be specialized to produce a particular output — for example, in a neural network used for navigating roads, one hidden layer may identify stop signs, and another hidden layer may identify traffic lights. While those hidden layers are not enough to independently navigate roads, they can function together within a larger neural network to drive better than Nick at age 16.



Backpropagation

The learning process for neural networks is called *backpropagation*. This technique modifies the weights of the neural network iteratively through calculation of deltas between predicted and expected outputs. After this calculation, the weights are updated backward through earlier layers via stochastic gradient descent. This process continues until the weights that minimize the loss function are found.

For regression tasks, the commonly used loss function to be optimized is mean squared error, whereas for classification tasks the common loss function used is cross-entropy. Given a loss function L , we can update the weights w through the chain rule of the following form, where z is the model’s output (and the best guess of our target variable y):

$$\frac{\partial L(z, y)}{\partial w} = \frac{\partial L(z, y)}{\partial x} * \frac{\partial x}{\partial z} * \frac{\partial z}{\partial w}$$

and the weights are updated via:

$$w = w - \alpha \frac{\partial L(z, y)}{\partial w}$$

For ML-heavy roles, we've seen interviewers expect candidates to explain the technical details behind basic backpropagation on a whiteboard, for basic methods such as linear regression or logistic regression.

Interviewers also like to ask about the hyperparameters involved in neural networks. For example, the amount that the weights are updated during each training step, α , is called the *learning rate*. If the learning rate is too small, the optimization process may freeze. Conversely, if the learning rate is too large, the optimization might converge prematurely at a suboptimal solution. Besides the learning rate, other hyperparameters in neural networks include the number of hidden layers, the activation functions used, batch size, and so on. For an interview, it's helpful to know how each hyperparameter affects a neural network's training time and model performance.

Training Neural Networks

Because neural networks require a gargantuan amount of weights to train, along with a considerable amount of hyperparameters to be searched for, the training process for a neural network can run into many problems.

General Framework

One issue that can come up in training neural nets is the problem of *vanishing gradients*. Vanishing gradients refers to the fact that sometimes the gradient of the loss function will be tiny, and may completely stop the neural network from training because the weights aren't updated properly. Since backpropagation uses the chain rule, multiplying n small numbers to compute gradients for early layers in a network means that the gradient gets exponentially smaller with more layers. This can happen particularly with traditional activation functions like hyperbolic tangent, whose gradients range between zero and one. The opposite problem, where activation functions p create large derivatives, is known as the *exploding gradient problem*.

One common technique to address extremes in gradient values is to allow gradients from later layers to directly pass into earlier layers without being multiplied many times — something which residual neural networks (ResNets) and LSTMs both utilize. Another approach to prevent extremes in the gradient values is to alter the magnitude of the gradient changes by changing the activation function used (for example, ReLU). The details behind these methods are beyond this book's scope but are worth looking into for ML-heavy interviews.

Training Optimization Techniques

Additionally, there are quite a few challenges in using vanilla gradient descent to train deep learning models. A few examples include getting trapped in suboptimal local minima or saddle points, not using a good learning rate, or dealing with sparse data with features of different frequencies where we may not want all features to update to the same extent. To address these concerns, there are a variety of optimization algorithms used.

Momentum is one such optimization method used to accelerate learning while using SGD. While using SGD, we can sometimes see small and noisy gradients. To solve this, we can introduce a new

parameter, velocity, which is the direction and speed at which the learning dynamics change. The velocity changes based on previous gradients (in an exponentially decaying manner) and increases the step size for learning in any iteration, which helps the gradient maintain a consistent direction and pace throughout the training process.

Transfer Learning

Lastly, for training neural networks, practitioners use or repurpose a pre-trained layer (components of a model that have already been trained and published). This approach is called *transfer learning* and is especially common in cases where models require a large amount of data (for example, BERT for language models and ImageNet for image classification). Transfer learning is beneficial when you have insufficient data for a new domain, and there is a large pool of existing data that can be transferred to the problem of interest. For example, say you wanted to help Jian Yang from the TV show *Silicon Valley* build an app to detect whether something was a hot dog or not a hotdog. Rather than just using your 100 images of hot dogs, you can use ImageNet (which was trained on many millions of images) to get a great model right off the bat, and then layer on any extra specific training data you might have to further improve accuracy.

Addressing Overfitting

Deep neural networks are prone to overfitting because of the model complexity (there are many parameters involved). As such, interviewers frequently ask about the variety of techniques which are used to reduce the likelihood of a neural network overfitting. Adding more training data is the simplest way to address variance if you have access to significantly more data and computational power to process that data. Another way is to standardize features (so each feature has 0 mean and unit variance), since this speeds up the learning algorithm. Without normalized inputs, each feature takes on a wide range of values, and the corresponding weights for those features can vary dramatically, resulting in larger updates in backpropagation. These large updates may cause oscillation in the weights during the learning stage, which causes overfitting and high variance.

Batch normalization is another technique to address overfitting. In this process, activation values are normalized within a given batch so that the representations at the hidden layers do not vary drastically, thereby allowing each layer of a network to learn more independently of one another. This is done for each hidden neuron, and also improves training speed. Here, applying a standardization process similar to how inputs are standardized is recommended.

Lastly, *dropout* is a regularization technique that deactivates several neurons randomly at each training step to avoid overfitting. Dropout enables simulation of different architectures, because instead of a full original neural network, there will be random nodes dropped at each layer. Both batch normalization and dropout help with regularization since the effects they have are similar to adding noise to various parts of the training process.

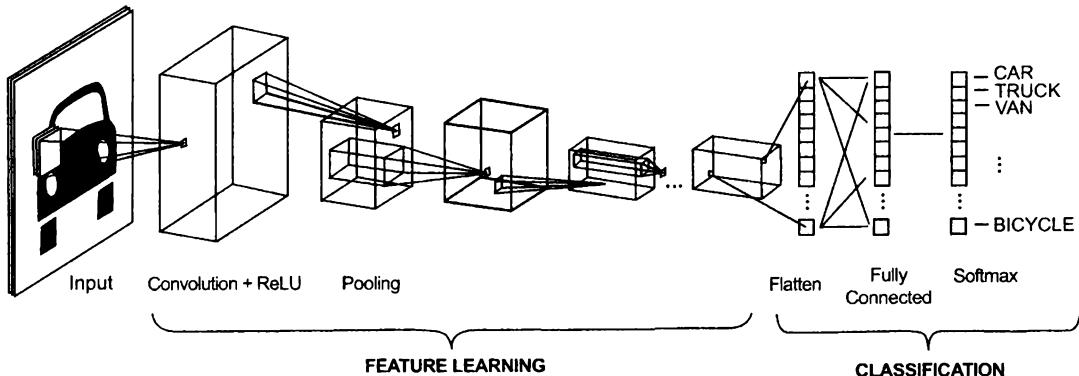
Types of Neural Networks

While a deep dive into the various neural network architectures is beyond the scope of this book, below, we jog your memory with a few of the most popular ones along with their applications. For ML-heavy roles, interviewers tend to ask about the different layer types used within each architecture, and to compare and contrast each architecture against one another.

CNNs

Convolutional neural networks (CNNs) are heavily used in computer vision because they can capture the spatial dependencies of an image through a series of filters. Imagine you were looking at a picture

of some traffic lights. Intuitively, you need to figure out the components of the lights (i.e., red, green, yellow) when processing the image. CNNs can determine elements within that picture by looking at various neighborhoods of the pixels in the image. Specifically, *convolution layers* can extract features such as edges, color, and gradient orientation. Then, *pooling layers* apply a version of dimensionality reduction in order to extract the most prominent features that are invariant to rotation and position. Lastly, the results are mapped into the final output by a fully connected layer.



RNNs

Recurrent neural networks (RNNs) are another common type of neural network. In an RNN, the nodes form a directed graph along a temporal sequence and use their internal state (called memory). RNNs are often used in learning sequential data such as audio or video — cases where the current context depends on past history. For example, say you are looking through the frames of a video. What will happen in the next frame is likely to be highly related to the current frame, but not as related to the first frame of the video. Therefore, when dealing with sequential data, having a notion of memory is crucial for accurate predictions. In contrast to CNNs, RNNs can handle arbitrarily input and output lengths and are not feed-forward neural networks, instead using this internal memory to process arbitrary sequences of data.

LSTMs

Long Short-Term Memory (LSTMs) are a fancier version of RNNs. In LSTMs, a common unit is composed of a cell, an input gate (writing to a cell or not), an output gate (how much to write to a cell), and a forget gate (how much to erase from a cell). This architecture allows for regulating the flow of information into and out of any cell. Compared to vanilla RNNs, which only learn short-term dependencies, LSTMs have additional properties that allow them to learn long-term dependencies. Therefore, in most real-world scenarios, LSTMs are used instead of RNNs.

Reinforcement Learning

Reinforcement learning (RL) is an area of machine learning outside of supervised and unsupervised learning. RL is about teaching an agent to learn which decisions to make in an environment to maximize some reward function. The agent takes a series of actions throughout a variety of states and is rewarded accordingly. During the learning process, the agent receives feedback based on the actions taken and aims to maximize the overall value acquired.

The main components of an RL algorithm include:

- **Reward function:** defines the goal of the entire RL problem and quantifies what a “good” or “bad” action is for the agent in any given state.
- **Policy:** defines how the agent picks its actions by mapping states to actions.
- **Model:** defines how the agent predicts what to do next as the agent understands the environment — given a state and action, the model will predict the reward and the next state.
- **Value function:** predicts the overall expected future reward discounted over time; it is consistently re-estimated over time to optimize long-term value.

RL is most widely known for use cases in gaming (AlphaGo, chess, Starcraft, etc.) and robotics. It is best used when the problem at hand is one of actions rather than purely predictions (that is, you do not know what constitutes “good” actions — supervised learning assumes you already know what the output should be). Unless you have particular projects or experience with reinforcement learning, it won’t be brought up in data science interviews.

The End-to-End ML Workflow

End-to-end machine learning questions are asked in interviews to see how well you apply machine learning theory to solve business problems. It isn’t just about confronting a real-world problem like “How would you design Uber’s surge pricing algorithm?” and then jumping to a technique like linear regression or random forests. Instead, it’s about asking the right questions about the business goals and constraints that inform the machine learning system design. It’s about walking through how you’d explore and clean the data and the features you would try to engineer. It’s about picking the right model evaluation metrics and modeling techniques, contextualizing model performance in terms of business impact, mentioning model deployment strategies, and much, much more.

To help you solve these all-encompassing problems — something most ML-theory textbooks don’t cover — we walk you through the entire machine learning workflow below. However, to really ace these open-ended ML problems come interview time, also read Chapter 11: Case Studies. In addition to tips on dealing with open-ended problems, the case study chapter includes ML case interview questions that force you to apply the concepts detailed below.

Step 1: Clarify the Problem and Constraints

If you frame the business and product problem correctly, you’ve done half the work. That’s because it’s easy to throw random ML techniques at data — but it’s harder to understand the business motivations, technical requirements, and stakeholder concerns that ultimately affect the success of a deployed machine learning solution. As such, make sure to start your answer by discussing what the problem at hand is, the business process and context surrounding the problem, any assumptions you might have, and what prospective stakeholder concerns are likely to be.

Some questions you can use to clarify the problem and the constraints include:

- What is the dependent variable we are trying to model? For example, if we are building a user churn model, what criteria are we using to define churn in the first place?
- How has the problem been approached in the past by the business? Is there a baseline performance we can compare against? How much do we need to beat this baseline for the project to be considered a success?

- Is ML even needed? Maybe a simple heuristics or a rules-based approach works well enough? Or perhaps a hybrid approach with humans in the loop would work best?
- Is it even legal or ethical to apply ML to this problem? Are there regulatory issues at play dictating what kinds of data or models you can use? For example, lending institutions cannot legally use some demographic variables like race.
- How do end users benefit from the solution, and how would they use the solution (as a standalone, or an integration with existing systems)?
- Is there a clear value add to the business from a successful solution? Are there any other stakeholders who would be affected?
- If an incorrect prediction is made, how will it impact the business? For example, a spam email making its way into your inbox isn't as problematic as a high-risk mortgage application accidentally being approved.
- Does ML need to solve the entire problem, end-to-end, or can smaller decoupled systems be made to solve sub-problems, whose output is then combined? For example, do you need to make a full self-driving algorithm, or separate smaller algorithms for environment perception, path planning, and vehicle control?

Once you've understood the business problem that your ML solution is trying to solve, you can clarify some of the technical requirements. Aligning on the technical requirements is especially important when confronted with a ML systems design problem. Some questions to ask to anchor the conversation:

- What's the latency needed? For example, search autocomplete is useless if it takes predictions longer to load than it takes users to type out their full query. Does every part of the system need to be real time—while inference may need to be fast, can training be slow?
- Are there any throughput requirements? How many predictions do you need to serve every minute?
- Where is this model being deployed? Does the model need to fit on-device? If so, how big is too big to deploy? And how costly is deployment? For example, adding a high-end GPU to a car is feasible cost-wise, but adding one to a drone might not be.

While spending so much time on problem definition may seem tedious, the reality is that defining the right solution for the right problem can save you many weeks of technical work and painful iterations later down the road. That's why interviewers, when posing open-ended ML problems, expect you to ask the right questions — ones that scope down your solution. By clarifying these constraints and objectives up front, you make better decisions on downstream steps of the end-to-end workflow. To further your business and product clarification skills, read the sections on product sense and company research in Chapter 10: Product Sense.

And one last piece of advice: don't go overboard with the questions! Remember, this is a time-bound interview, so make sure your questions and assumptions are reasonable and relevant (and concise). You don't want to be like a toddler and ask 57 questions without getting anywhere.

Step 2: Establish Metrics

Once you've understood the stakeholder objectives and constraints imposed, it's best to pick simple, observable, and attributable metrics that encapsulate solving the problem. Note that sometimes the business is only interested in optimizing their existing business KPIs (for example, the time to resolve a customer request). In that case, you need to be able to align your model performance metrics with

solving the business problem. For example, for a customer support request classification model, a 90% model accuracy means that 50% of the customer tickets that previously needed to be rerouted now end up in the right place, resulting in a 10% decrease in time to resolution.

In real-world scenarios, it's best to opt for a single metric rather than picking multiple metrics to capture different sub-goals. That's because a single metric makes it easier to rank model performance. Plus, it's easier to align the team around optimizing a single number. However, in interview contexts, it may be beneficial to mention multiple metrics to show you've thought about the various goals and trade-offs your ML solution needs to satisfy. As such, in an interview, we recommend you start your answer with a single metric, but then hedge your answer by mentioning other potential metrics to track.

For example, if posed a question about evaluation metrics for a spam classifier, you could start off by talking about accuracy, and then move on to precision and recall as the conversation becomes more nuanced. In an effort to optimize a single metric, you could recommend using the F-1 score. A nuanced answer could also incorporate an element of satisficing — where a secondary metric is just good enough. For example, you could optimize precision @ recall 0.95 — i.e., constraining the recall to be at least 0.95 while optimizing for precision. Or you could suggest blending multiple metrics into one by weighting different sub-metrics, such as false positives versus false negatives, to create a final metric to track. This is often known as an OEC (overall evaluation criterion), and gives you a balance between different metrics.

Once you've picked a metric, you need to establish what success looks like. While for a classifier, you might desire 100% accuracy, is this a realistic bar for measuring success? Is there a threshold that's good enough? This is why inquiring about baseline performance in Step 1 becomes crucial. If possible, you should use the performance of the existing setup for comparison (for example, if the average time to resolution for customer support tickets is 2 hours, you could aim for 1 hour — not a 97% ticket classification accuracy). Note: in real-world scenarios, the bar for model performance isn't as high as you'd think to still have a positive business impact.

Be sure to voice all these metric considerations to your interviewer so that you can show you've thought critically about the problem. For more guidance, read Chapter 10: Product Sense, which covers the nuances and pitfalls of metric selection.

Step 3: Understand Your Data Sources

Your machine learning model is only as good as the data it sees; hence, the phrase “garbage in, garbage out.” For classroom projects or Kaggle competitions, there isn't much you can do since you usually have a fixed dataset, and it's all about fitting a model that maximizes some metric. However, in the real world, you have leeway in what data you use to solve the business problem. As such, clearly articulate what data sources you would prefer to solve the interview problem. While it's intuitive to use the internal company data relevant to the problem at hand, that's not the be-all end-all of data sourcing.

For open-ended ML questions, especially at startups that might be testing your scrappiness, you should think outside the box regarding what data to use. Data sources to consider:

- Can you acquire more data by crowdsourcing it via Amazon Mechanical Turk?
- Can you ask users for data as part of the user onboarding process?
- Can you buy second- and third-party datasets?
- Can you ethically scrape the data from online sources?
- Can you send your unlabeled internal data off to a labeling and annotation service?

To boost model performance, it might not be about collecting more data generally. Instead, you can intentionally source more examples of edge cases via data augmentation or artificial data synthesis. For example, suppose your traffic light detector struggles in low-contrast situations. You could make a version of your training images that has less contrast in order to give your neural network more practice on these trickier photos. Taken to the extreme, you can even simulate the entire environment, as is common in the self-driving car industry. Simulation is used in the autonomous vehicle space because encountering the volume of rare and risky situations needed to adequately train a model based on only real-world driving is infeasible.

Finally, do you understand the data? Questions to consider:

- How fresh is the data? How often will the data be updated?
- Is there a data dictionary available? Have you talked to subject matter experts about it?
- How was the data collected? Was there any sampling, selection, or response bias?

Step 4: Explore Your Data

A good first step in exploratory data analysis is to profile the columns at first glance: which ones might be useful? Which ones have practically no variance and thus wouldn't offer up any real predictive value? Which columns look noisy? Which ones have a lot of missing or odd values? Besides skimming through your data, also look at summary statistics like the mean, median, and quantiles.

"The greatest value of a picture is when it forces us to notice what we never expected to see."

—John Tukey

Because a picture is worth a thousand words, visualizing your data is also a crucial step in exploratory data analysis. For columns of interest, you want to visualize their distributions to understand their statistical properties like skewness and kurtosis. Certain features (e.g., age, weight) may be better visualized with a histogram through binning. It also helps to visualize the range of continuous variables and plot categories of categorical variables. Finally, you can visually inspect the basic relationships between variables using a correlation matrix. This can help you quickly spot which variables are correlated with one another, as well as what might be correlated with the target variable at hand.

Step 5: Clean Your Data

Did you get suckered into data science and machine learning because you thought most of your time would be spent doing sexy data analysis and modeling work? Don't worry, we got fooled too! The joke (and grievance) that most data scientists spend 80% of their time cleaning data stems from a frustrating truth: if you feed garbage data into a model, you get garbage out. Between the logging issues, missing values, data entry errors, data merges gone wrong, changing schemas due to product changes, and columns whose meaning changes over time, there are many reasons why your data might be a hot mess. That's precisely why a critical step before modeling is data munging.

One aspect of data munging is dropping irrelevant data or erroneously duplicated rows and columns. You should also handle incorrect values that don't match up with the supposed data schema. For example, for fields containing human-inputted data, there is often a pattern of errors or typos that you could find and then use to clean up.

To handle missing data, you should first understand the root cause. Based on the results, several techniques to deal with missing values include:

- Imputing the missing values via basic methods such as column mean/median

- Using a model or distribution to impute the missing data
- Dropping the rows with missing values (as a last resort)

Another critical data cleaning step is dealing with outliers. Outliers may be due to issues in data collection, like manual data entry issues or logging hiccups. Or maybe they accurately reflect the actual data. Outliers can be removed outright, truncated, or left as is, depending on their source and the business implications of including them or not.

Note that outliers may be univariate, while others are multivariate and require looking over many dimensions. For an example of a multivariate outlier, consider a dataset of human ages and heights. A 4-year-old human wouldn't be strange, a 5-foot-tall person isn't odd, but a 4-year-old that's 5-feet tall would either be an outlier, data entry error, or a Guinness world record holder.

Step 6: Feature Engineering

Feature engineering is the art of presenting data to machine learning models in the best way possible. One part of feature engineering is feature selection: the process of selecting a relevant subset of features for model construction, based on domain knowledge. The other is feature preprocessing: transforming your data in a way that allows an algorithm to learn the underlying structure of the data without having to sift through noisy inputs. Careful feature selection and feature processing can boost model performance and let you get away with using simpler models. The specific workflows for feature engineering depend on the type of data involved.

For quantitative data, several common operations are performed:

- **Transformations:** applying a function (like log, capping, or flooring) can help when the data is skewed or when you want to make the data conform to more standard statistical distributions (a requirement for certain models).
- **Binning:** also known as discretization or bucketing, this process breaks down a continuous variable into discrete bins, enabling a reduction of noise associated with the data.
- **Dimensionality Reduction:** to generate a reduced set of uncorrelated features, you can use a technique like PCA.

Also, standardize and scale data as needed: this is especially important for machine learning algorithms that may be sensitive to variance in the feature values. For example, K-means uses Euclidean distance to measure distances between points and clusters, so all features need comparable variances. To normalize data, you can use min/max scaling, so that all data lies between zero and one. To standardize features, you can use z-scores, so that data has a mean of zero and a variance of one.

For categorical data, two common approaches are:

- **One-hot encoding:** turns each category into a vector of all zeroes, with the exception of a “one” for the category at hand
- **Hashing:** turns the data into a fixed dimensional vector using a hashing function; great for when features have a very high cardinality (large range of values) and the output vector of one-hot encoding would be too big

While you might not be asked about NLP during most interviews, if it's listed on your resume, it's good to know a few text preprocessing techniques as well:

- **Stemming:** reduces a word down to a root word by deleting characters (for example, turning the words “liked” and “likes” into “like”).

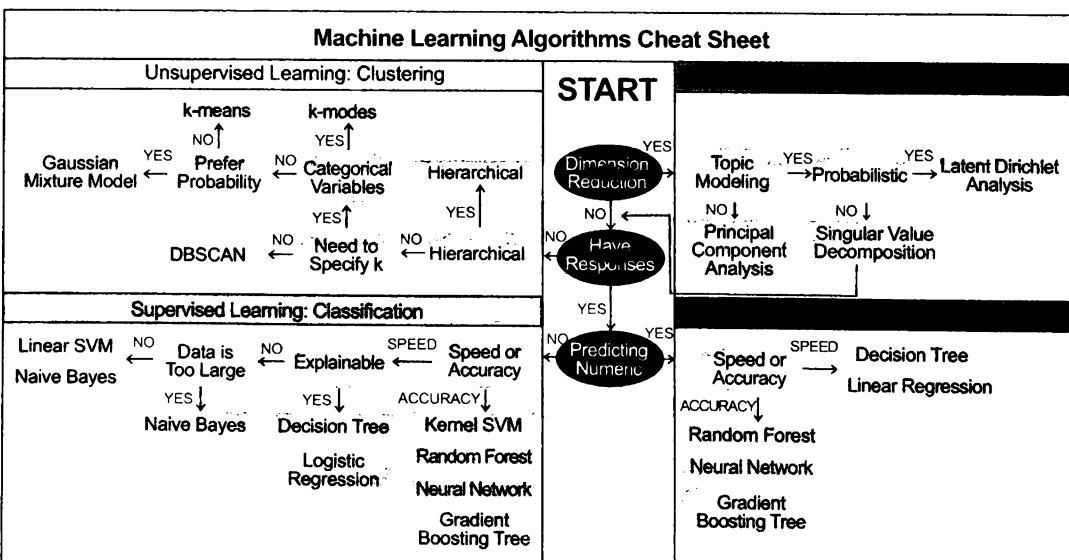
- **Lemmatization:** somewhat similar to stemming, but instead of just reducing words into roots, it takes into account the context and meaning of the word (for example, it would turn “caring” to “care,” whereas stemming would turn “caring” to “car”).
- **Filtering:** removes “stop words” that don’t add value to a sentence — like “the” and “a”, along with removing punctuation.
- **Bag-of-words:** represents text as a collection of words by associating each word and its frequency.
- **N-grams:** an extension of bag-of-words where we use N words in a sequence.
- **Word embeddings:** a representation that converts words to vectors that encode the meaning of the word, where words that are closer in meaning are closer in vector space (popular methods include word2vec and GloVe).

Step 7: Model Selection

Given the business constraints, evaluation metrics, and data sources available, what types of models make the most sense to try? Factors to consider when selecting a model include:

- **Training & Prediction Speed:** for example, linear regression is much quicker than neural networks for the same amount of data
- **Budget:** neural networks, for instance, can be computationally intensive models to train
- **Volume & Dimensionality of Data:** for example, neural networks can handle large amounts of data and higher-dimensional data versus $k\text{-NN}$
- **Categorical vs. Numerical Features:** for example, linear regression cannot handle categorical variables directly, as they need to be one-hot encoded, versus trees (which can generally handle them directly)
- **Explainability:** choosing interpretable models like linear regression may be favorable to “black box” neural networks due to regulatory concerns or their ease of debugging

Below is a quick cheat sheet to also consider:



Step 8: Model Training & Evaluation

So you picked a type of model, or at least narrowed it down to a few candidates. At this point, interviewers will expect you to talk about how to train the model. This is a good time to mention the train-validation-test split, cross-validation, and hyper-parameter tuning.

You might also be asked how to compare your model to other models, how to learn its parameters, and how you'd know if its performance is good enough to ship. They might ask about edge cases, like handling biased training data, how you'd assess feature importance, or dealing with data imbalances.

The primary ways to deal with these issues, like picking the right model evaluation metric, using a validation set, regularizing your models to avoid overfitting, and using learning curves to find performance plateaus, are explained in-depth earlier in the chapter.

On the flip side, if you are dealing with so much data that your models are taking forever to train, then it is worth looking into various sampling techniques. The most common ones in practice are random sampling (sampling with or without replacement) and stratified sampling (sampling from specific groups among the entire population). In addition, for imbalanced datasets, undersampling and oversampling techniques are frequently used (SMOTE, for example). It is important to make sure you understand the sampling method at hand because a wrong sampling strategy may lead to incorrect results.

Step 9: Deployment

So, now that you've picked out a model, how do you deploy it? The process of operationalizing the entire deployment process is referred to as "MLOps" — when DevOps meets ML. Two popular tools in this space are Airflow and MLFlow. Because frameworks come and go, and many large tech companies use their own internal versions of these tools, it's rare to be asked about these specific technologies in interviews. However, knowledge of high-level deployment concepts is still helpful.

Generally, systems are deployed online, in batch, or as a hybrid of the two approaches. Online means latency is critical, and thus model predictions are made in real time. Since model predictions generally need to be served in real time, there will typically be a caching layer of cached features. Downsides for online deployment are that it can be computationally intensive to meet latency requirements and requires robust infrastructure monitoring and redundancy.

Batch means predictions are generated periodically and is helpful for cases where you don't need immediate results (most recommendation systems, for example) or require high throughput. But the downside is that batch predictions may not be available for new data (for example, a recommendation list cannot be updated until the next batch is computed). Ideally, you can work with stakeholders to find the sweet spot, where a batch predictor is updated frequently enough to be "good enough" to solve the problem at hand.

One deployment issue worth bringing up, that's common to both batch and online ML systems, is model degradation. Models degrade because the underlying distributions of data for your model change. For a concrete example, suppose you were working for a clothing e-commerce site and were training a product recommendation model in the winter time. Come summer, you might accidentally be recommending Canada Goose jackets in July --- not a very relevant product suggestion to anyone besides Drake.

This feature drift leads to the phenomenon known as the *training-serving skew*, where there's a performance hit between the model in training and evaluation time versus when the model is served in production. To show awareness for the training-serving skew issue, be sure to mention to your

interviewer how often you'd retrain a model, what events might trigger a model refresh, and how much new data to use versus how much to rely on historical data. Also, mention how you'd add logging to monitor and catch model degradation.

Step 10: Iterate

Deploying a model isn't the end of the machine learning workflow. Business objectives change. Evaluation metrics change. Data features change. As such, you should plan to iterate on deployed models.

A big part of knowing how to iterate on your system can be learned via error analysis — the act of analyzing the wrong predictions manually. By looking at bad predictions and bucketing them into the types of reasons they occurred, you can better prioritize what projects to work on next. For example, say our traffic light detector tended to mislabel photos taken during rain — this tells you that you should collect more images in inclement weather. Or maybe this tells you to source whether it's raining or not as a feature for your model.

By focusing on iterating your design, and continuously seeking ways for your system to improve, you can generate increasing amounts of business value with your deployed models.

Machine Learning Interview Questions

As mentioned in the introduction, most machine learning interview questions take the simplistic form of "What does term X mean?" or "How does technique Y work?" To make the 35 interview questions more useful to you, dear reader, we intentionally shied away from including too many boring Google-able definitional questions. Instead, we chose to include the more interesting ML problems that showed up in real interviews.

In addition to the problems below, look to the multi-part case study problems in Chapter 11. We intentionally put these end-to-end machine learning modeling questions in the last chapter, because they incorporate system design and product-sense skills — concepts which we cover later.

Easy

- 7.1. Robinhood: Say you are building a binary classifier for an unbalanced dataset (where one class is much rarer than the other, say 1% and 99%, respectively). How do you handle this situation?
- 7.2. Square: What are some differences you would expect in a model that minimizes *squared* error versus a model that minimizes *absolute* error? In which cases would each error metric be appropriate?
- 7.3. Facebook: When performing K -means clustering, how do you choose k ?
- 7.4. Salesforce: How can you make your models more robust to outliers?
- 7.5. AQR: Say that you are running a multiple linear regression and that you have reason to believe that several of the predictors are correlated. How will the results of the regression be affected if several are indeed correlated? How would you deal with this problem?
- 7.6. Point72: Describe the motivation behind random forests. What are two ways in which they improve upon individual decision trees?
- 7.7. PayPal: Given a large dataset of payment transactions, say we want to predict the likelihood of a given transaction being fraudulent. However, there are many rows with missing values for various columns. How would you deal with this?

- 7.8. Airbnb: Say you are running a simple logistic regression to solve a problem but find the results to be unsatisfactory. What are some ways you might improve your model, or what other models might you look into using instead?
- 7.9. Two Sigma: Say you were running a linear regression for a dataset but you accidentally duplicated every data point. What happens to your beta coefficient?
- 7.10. PWC: Compare and contrast gradient boosting and random forests.
- 7.11. DoorDash: Say that DoorDash is launching in Singapore. For this new market, you want to predict the estimated time of arrival (ETA) for a delivery to reach a customer after an order has been placed on the app. From an earlier beta test in Singapore, there were 10,000 deliveries made. Do you have enough training data to create an accurate ETA model?

Medium

- 7.12. Affirm: Say we are running a binary classification loan model, and rejected applicants must be supplied with a reason why they were rejected. Without digging into the weights of features, how would you supply these reasons?
- 7.13. Google: Say you are given a very large corpus of words. How would you identify synonyms?
- 7.14. Facebook: What is the bias-variance trade-off? How is it expressed using an equation?
- 7.15. Uber: Define the cross-validation process. What is the motivation behind using it?
- 7.16. Salesforce: How would you build a lead scoring algorithm to predict whether a prospective company is likely to convert into being an enterprise customer?
- 7.17. Spotify: How would you approach creating a music recommendation algorithm?
- 7.18. Amazon: Define what it means for a function to be convex. What is an example of a machine learning algorithm that is not convex and describe why that is so?
- 7.19. Microsoft: Explain what information gain and entropy are in the context of a decision tree and walk through a numerical example.
- 7.20. Uber: What is L1 and L2 regularization? What are the differences between the two?
- 7.21. Amazon: Describe gradient descent and the motivations behind stochastic gradient descent.
- 7.22. Affirm: Assume we have a classifier that produces a score between 0 and 1 for the probability of a particular loan application being fraudulent. Say that for each application's score, we take the square root of that score. How would the ROC curve change? If it doesn't change, what kinds of functions would change the curve?
- 7.23. IBM: Say X is a univariate Gaussian random variable. What is the entropy of X ?
- 7.24. Stitch Fix: How would you build a model to calculate a customer's propensity to buy a particular item? What are some pros and cons of your approach?
- 7.25. Citadel: Compare and contrast Gaussian Naive Bayes (GNB) and logistic regression. When would you use one over the other?

Hard

- 7.26. Walmart: What loss function is used in k -means clustering given k clusters and n sample points? Compute the update formula using (1) batch gradient descent and (2) stochastic gradient descent for the cluster mean for cluster k using a learning rate ϵ .

- 7.27. Two Sigma: Describe the kernel trick in SVMs and give a simple example. How do you decide what kernel to choose?
- 7.28. Morgan Stanley: Say we have N observations for some variable which we model as being drawn from a Gaussian distribution. What are your best guesses for the parameters of the distribution?
- 7.29. Stripe: Say we are using a Gaussian mixture model (GMM) for anomaly detection of fraudulent transactions to classify incoming transactions into K classes. Describe the model setup formulaically and how to evaluate the posterior probabilities and log likelihood. How can we determine if a new transaction should be deemed fraudulent?
- 7.30. Robinhood: Walk me through how you'd build a model to predict whether a particular Robinhood user will churn?
- 7.31. Two Sigma: Suppose you are running a linear regression and model the error terms as being normally distributed. Show that in this setup, maximizing the likelihood of the data is equivalent to minimizing the sum of the squared residuals.
- 7.32. Uber: Describe the idea behind Principle Components Analysis (PCA) and describe its formulation and derivation in matrix form. Next, go through the procedural description and solve the constrained maximization.
- 7.33. Citadel: Describe the model formulation behind logistic regression. How do you maximize the log-likelihood of a given model (using the two-class case)?
- 7.34. Spotify: How would you approach creating a music recommendation algorithm for Discover Weekly (a 30-song weekly playlist personalized to an individual user)?
- 7.35. Google: Derive the variance-covariance matrix of the least squares parameter estimates in matrix form.

Machine Learning Solutions

Solution #7.1

Unbalanced classes can be dealt with in several ways.

First, you want to check whether you can get more data or not. While in many scenarios, data may be expensive or difficult to acquire, it's important to not overlook this approach, and at least mention it to your interviewer.

Next, make sure you're looking at appropriate performance metrics. For example, accuracy is not a correct metric to use when classes are imbalanced --- instead, you want to look at precision, recall, F1 score, and the ROC curve.

Then, you can resample the training set by either oversampling the rare samples or undersampling the abundant samples; both can be accomplished via bootstrapping. These approaches are easy and quick to run, so they should be good starting points. Note, if the event is inherently rare, then oversampling may not be necessary, and you should focus more on the evaluation function.

Additionally, you could try generating synthetic examples. There are several algorithms for doing so --- the most popular is called SMOTE (synthetic minority oversampling technique), which creates synthetic samples of the rare class rather than pure copies by selecting various instances. It does this by modifying the attributes slightly by a random amount proportional to the difference in neighboring instances.

Another way is to resample classes by running ensemble models with different ratios of the classes, or by running an ensemble model using all samples of the rare class and a differing amount of the abundant class. Note that some models, such as logistic regression, are able to handle unbalanced classes relatively well in a standalone manner. You can also adjust the probability threshold to something besides 0.5 for classifying the unbalanced outcome.

Lastly, you can design your own cost function that penalizes wrong classification of the rare class more than wrong classifications of the abundant class. This is useful if you have to use a particular kind of model and you're unable to resample. However, it can be complex to set up the penalty matrix, especially with many classes.

Solution #7.2

We can denote squared error as MSE and absolute error as MAE. Both are measures of distances between vectors and express average model prediction in units of the target variable. Both can range from 0 to infinity; the lower the score, the better the model.

The main difference is that errors are squared before being averaged in MSE, meaning there is a relatively high weight given to large errors. Therefore, MSE is useful when large errors in the model are trying to be avoided. This means that outliers disproportionately affect MSE more than MAE — meaning that MAE is more robust to outliers. Computation-wise, MSE is easier to use, since the gradient calculation is more straightforward than that of MAE, which requires some linear programming to compute the gradient.

Therefore, if the model needs to be computationally easier to train or doesn't need to be robust to outliers, then MSE should be used. Otherwise, MAE is the better option. Lastly, MSE corresponds to maximizing the likelihood of Gaussian random variables, and MAE does not. MSE is minimized by the conditional mean, whereas MAE is minimized by the conditional *median*.

Solution #7.3

The elbow method is the most well-known method for choosing k in k -means clustering. The intuition behind this technique is that the first few clusters will explain a lot of the variation in the data, but past a certain point, the amount of information added is diminishing. Looking at a graph of explained variation (on the y-axis) versus the number of clusters (k), there should be a sharp change in the y-axis at some level of k . For example, in the graph that follows, we see a dropoff at approximately $k = 6$.

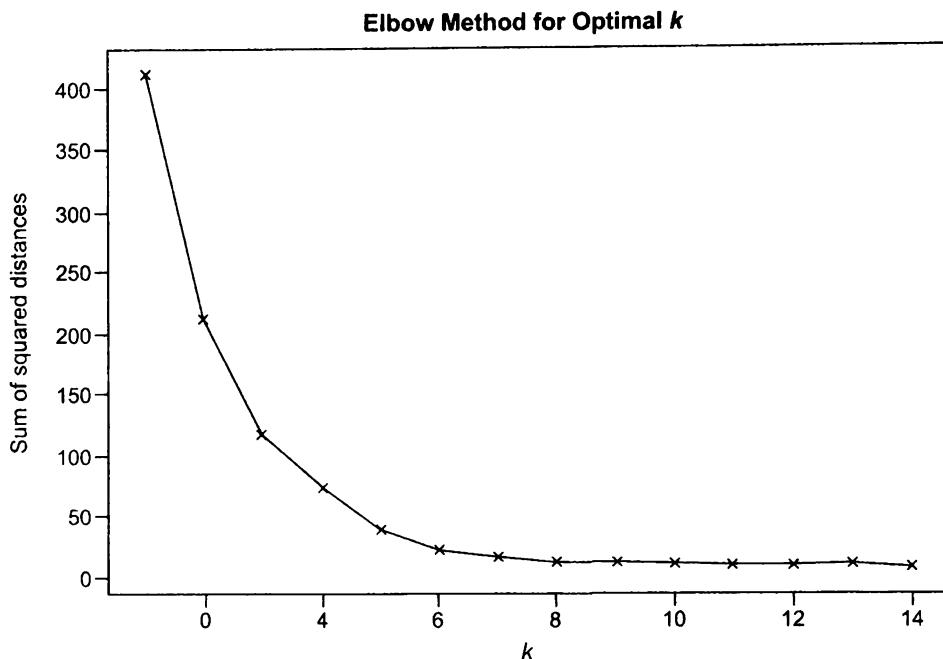
Note that the explained variation is quantified by the within-cluster sum of squared errors. To calculate this error metric, we look at, for each cluster, the total sum of squared errors (using Euclidean distance). A caveat to keep in mind: the assumption of a drop in variation may not necessarily be true — the y-axis may be continuously decreasing slowly (i.e., there is no significant drop).

Another popular alternative to determining k in k -means clustering is to apply the silhouette method, which aims to measure how similar points are in its cluster compared to other clusters. Concretely, it looks at:

$$\frac{(x - y)}{\max(x, y)}$$

where x is the mean distance to the examples of the nearest cluster, and y is the mean distance to other examples in the same cluster. The coefficient varies between -1 and 1 for any given point. A value of 1 implies that the point is in the "right" cluster and vice versa for a score of -1. By plotting the score

on the y-axis versus k , we can get an idea for the optimal number of clusters based on this metric. Note that the metric used in the silhouette method is more computationally intensive to calculate for all points versus the elbow method.



Taking a step back, while both the elbow and silhouette methods serve their purpose, sometimes it helps to lean on your business intuition when choosing the number of clusters. For example, if you are clustering patients or customer groups, stakeholders and subject matter experts should have a hunch concerning how many groups they expect to see in the data. Additionally, you can visualize the features for the different groups and assess whether they are indeed behaving similarly. There is no perfect method for picking k , because if there were, it would be a supervised problem and not an unsupervised one.

Solution #7.4

Investigating outliers is often the first step in understanding how to treat them. Once you understand the nature of why the outliers occurred, there are several possible methods we can use:

- Add regularization: reduces variance, for example L1 or L2 regularization.
- Try different models: can use a model that is more robust to outliers. For example, tree-based models (random forests, gradient boosting) are generally less affected by outliers than linear models.
- Winsorize data: cap the data at various arbitrary thresholds. For example, at a 90% winsorization, we can take the top and bottom 5% of values and set them to the 95th and 5th percentile of values, respectively.
- Transform data: for example, do a log transformation when the response variable follows an exponential distribution or is right skewed.
- Change the error metric to be more robust: for example, for MSE, change it to MAE or Huber loss.

- Remove outliers: only do this if you're certain that the outliers are true anomalies not worth incorporating into the model. This should be the last consideration, since dropping data means losing information on the variability in the data.

Solution #7.5

There will be two primary problems when running a regression if several of the predictor variables are correlated. The first is that the coefficient estimates and signs will vary dramatically, depending on what particular variables you included in the model. Certain coefficients may even have confidence intervals that include 0 (meaning it is difficult to tell whether an increase in that X value is associated with an increase or decrease in Y or not), and hence results will not be statistically significant. The second is that the resulting p-values will be misleading. For instance, an important variable might have a high p-value and so be deemed as statistically insignificant even though it is actually important. It is as if the effect of the correlated features were “split” between them, leading to uncertainty about which features are actually relevant to the model.

You can deal with this problem by either removing or combining the correlated predictors. To effectively remove one of the predictors, it is best to understand the causes of the correlation (i.e., did you include extraneous predictors such as X and $2X$ or are there some latent variables underlying one or more of the ones you have included that affect both? To combine predictors, it is possible to include interaction terms (the product of the two that are correlated). Additionally, you could also (1) center the data and (2) try to obtain a larger size of sample, thereby giving you narrower confidence intervals. Lastly, you can apply regularization methods (such as in ridge regression).

Solution #7.6

Random forests are used since individual decision trees are usually prone to overfitting. Not only can these utilize multiple decision trees and then average their decisions, but they can be used for either classification or regression. There are a few main ways in which they allow for stronger out-of-sample prediction than do individual decision trees.

- As in other ensemble models, using a large set of trees created in a resample of the data (bootstrap aggregation) will lead to a model yielding more consistent results. More specifically, and in contrast to decision trees, it leads to diversity in training data for each tree and so contributes to better results in terms of bias-variance trade-off (particularly with respect to variance).
- Using only $m < p$ features at each split helps to de-correlate the decision trees, thereby avoiding having very important features always appearing at the first splits of the trees (which would happen on standalone trees due to the nature of information gain).
- They’re fairly easy to implement and fast to run.
- They can produce very interpretable feature-importance values, thereby improving model understandability and feature selection.

The first two bullet points are the main ways random forests improve upon single decision trees.

Solution #7.7

Step 1: Clarify the Missing Data

Since these types of problems are generally context dependent, it’s best to start your answer with clarifying questions. For example,

- Is the amount of missing values uniform by feature?

- Are these missing values numerical or categorical?
- How many features with missing data are there?
- Is there a pattern in the types of transactions that have a lot of missing data?

It would also be useful to think about why the data is missing, because this affects how you'd impute the data. Missing data is commonly classified as:

- ***Missing completely at random (MCAR)***: the probability of being missing is the same for all classes
- ***Missing at random (MAR)***: the probability of being missing is the same within groups defined by the observed data
- ***Not missing at random (NMAR)***: if the data is not MCAR and not MAR

Step 2: Establish a Baseline

One reason to ask these questions is because a good answer would consider that the missing data may not actually be a problem. What if the missing data was in transactions that were almost never fraud? What if the missing data is mostly in columns whose data features don't have good predictive value? For example, if you were missing the IP-address derived geolocation of the person making the payment, that would likely be bad for model performance. On the other hand, if you were missing the user's middle name, it likely wouldn't have any bearing on whether the transaction was fraud or not. Even simpler yet, can a baseline model be built that meets the business goals, without having to deal with any missing data?

Step 3: Impute Missing Data

If the baseline model indicates that dealing with the missing data is worth it, one technique we could use is imputation. For continuous features, we can start by using the mean or median for missing values within any feature. However, the downside to this approach is that it does not factor in any of the other features and correlations between them -- it is unlikely that two transactions in differing locations for different category codes would have the same transaction price. An alternative could be to use a nearest neighbors method to estimate any given feature based on the other features available.

Step 4: Check Performance with Imputed Data

With these modeled features, we can then run a set of classification algorithms to predict fraud or not fraud. With this imputation technique, we can also cross-validate to check whether performance increases by including the imputed data, relative to just the original data. Note that a performance increase is only expected if the feature contains valuable information (for those rows/entries that have it). If this isn't the case, and you won't see a significant impact as a result, it may be easiest to drop the existing missing data before training the model.

Step 5: Other Approaches for Missing Data

Finally, thinking outside the box, is it possible to use a third-party dataset to fill in some of the missing information? Suppose a common missing piece of information was the type of business that a person paid. But, let's say we have the business's address on file. Could we use the address against a business listings dataset to infer the type of business the transaction was made at? Lastly, note that some models are capable of dealing with missing data, without requiring imputation.

Solution #7.8

There are several possible ways to improve the performance of a logistic regression:

- **Normalizing features:** The features should be normalized such that particular weights do not dominate within the model.
- **Adding additional features:** Depending on the problem, it may simply be the case that there aren't enough useful features. In general, logistic regression is high bias, so adding more features should be helpful.
- **Addressing outliers:** Identify and decide whether to retain or remove them.
- **Selecting variables:** See if any features have introduced too much noise into the process.
- **Cross validation and hyperparameter tuning:** Using k -fold cross validation along with hyperparameter tuning (for example, introducing a penalty term for regularization purposes) should help improve the model.
- The classes may not be linearly separable (logistic regression produces linear decision boundaries), and, therefore, it would be worth looking into SVMs, tree-based approaches, or neural networks instead.

Solution #7.9

For regular regression, recall we have the following for our least squares estimator:

$$\beta = (X^T X)^{-1} X^T y$$

So if we double the data, then we are using: $\begin{pmatrix} X \\ X \end{pmatrix}, \begin{pmatrix} Y \\ Y \end{pmatrix}$

instead of and respectively. Then plugging this into our estimator from above, we get:

$$\beta = \left(\begin{pmatrix} X \\ X \end{pmatrix}^T \begin{pmatrix} X \\ X \end{pmatrix} \right)^{-1} \begin{pmatrix} X \\ X \end{pmatrix}^T \begin{pmatrix} Y \\ Y \end{pmatrix}$$

Simplifying yields:

$$\beta = (2X^T X)^{-1} 2X^T y$$

Therefore, we see that the coefficient remains unchanged.

Solution #7.10

In both gradient boosting and random forests, an ensemble of decision trees are used. Additionally, both are flexible models and don't need much data preprocessing.

However, there are two main differences. The first main difference is that, in gradient boosting, trees are built one at a time, such that successive weak learners learn from the mistakes of preceding weak learners. In random forests, the trees are built independently at the same time.

The second difference is in the output: gradient boosting combines the results of the weak learners with each successive iteration, whereas, in random forests, the trees are combined at the end (through either averaging or majority).

Because of these structural differences, gradient boosting is often more prone to overfitting than are random forests due to their focus on mistakes over training iterations and the lack of independence in tree building. Additionally, gradient boosting hyperparameters are harder to tune than those of

random forests. Lastly, gradient boosting may take longer to train than random forests because the trees of the latter are built sequentially. In real-life applications, gradient boosting generally excels when used on unbalanced datasets (fraud detection, for example), whereas random forests generally excel at multi-class object detection with noisy data (computer vision, for example).

Solution #7.1

Because “accurate enough” is subjective, it’s best to ask the interviewer clarifying questions before addressing the lack of training data. To stand out, you can also proactively mention ways to source more training data at the end of your answer.

Step 1: Clarify What “Good” ETA Means

To determine how accurate the ETA model needs to be, first ask what the ETA prediction will be used for. The level of accuracy needed in ETA predictions might be higher for the order-driver matching algorithm than what DoorDash needs to display to the customer in the app. Also, consider if your ETA estimate under-promises and over-delivers. Maybe that’s okay — customers would likely be happy that the delivery arrived faster than expected. At the same time, high ETA estimates across the board may lead people to say, “Screw it, I’ll just go to the store and pick it up myself.” By considering the context around how the ETA predictions will be used, you’ll be one step closer to understanding what a good-enough ETA model looks like.

One data-driven approach to establishing how accurate your ETA model needs to be involves looking at ETA models in similar markets. From data in other locations, we could better understand the economic impact of both over and underestimated ETAs. Tying the model output to its business impact can help DoorDash decide if investing money into solving this problem is even warranted in the first place.

Step 2: Assess Baseline ETA Performance

After you understand what “good-enough” ETA means, it’s best practice to next see how a baseline model, trained on the beta 10,000 deliveries made, performs. A baseline model can be something as simple as the estimated driving time plus the average preparation time (conditional on the restaurant and time of day). Since predicting the ETAs is a regression problem, potential metrics we can use to assess this baseline model include root mean square error (RMSE), MAE, R^2 , etc.

Step 3: Determine How More Data Improves Accuracy

Say that we use R^2 as the main metric. One way to gauge the benefit from having more training data would be to build learning curves. A learning curve depicts how the accuracy changes when we train a model on a progressively larger percentage of the data. For example, say that with 25% of the data, we get an R^2 of 0.5, with 50% of the data we get an R^2 of 0.65, and with 75% of the data we get an R^2 of 0.67. Note that the improvement drops off significantly between the use of 50% and 75% of the training data. The point at which the drop-off starts to become a problem is the signal that we should look into reevaluating the features rather than simply adding more training data.

This process is analogous to looking at the learning curves discussed earlier in this chapter, which look at how the training error and validation error change over the number of iterations — here, instead, we are looking at how the model performance changes over the amount of training data used.

Step 4: In Case Performance Isn’t “Good Enough”

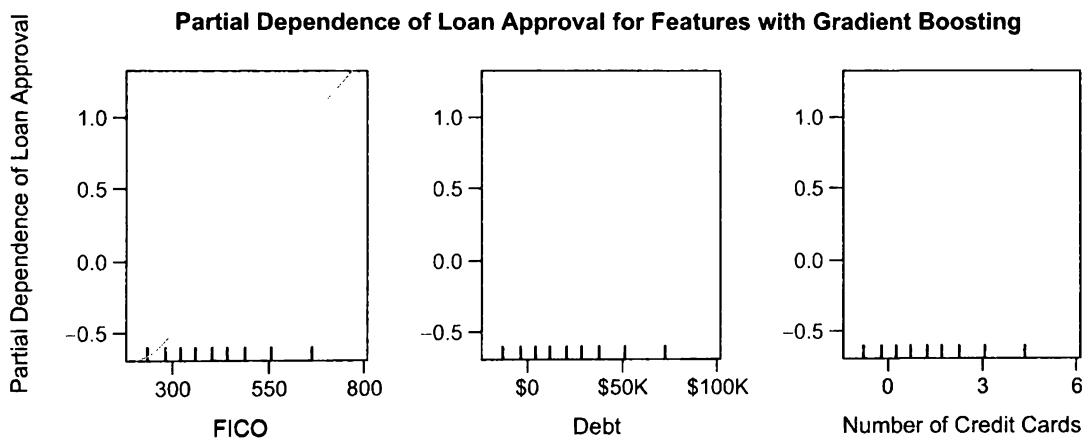
If after learning curves you realize that you don’t have sufficient data to build an accurate enough model, the interviewer would likely shift the discussion to dealing with this lack of data. Or, if

you are feeling like an overachiever with a can-do attitude, you could proactively bring up these discussion points:

- Are there too few features? If so, you want to look into adding features like marketplace supply and demand indicators, traffic patterns on the road at the time of the delivery, etc.
- Are there too many features? If there are almost as many or more features than data points, then our model will be prone to overfitting and we should look into either dimensionality reduction or feature selection techniques.
- Can different models be used that handle smaller training datasets better?
- Is it possible to acquire more data in a cost-effective way?
- Is the less accurate ETA model a true launch blocker? If we launched in the new market, which generates more training data, can the ETA model be retrained?

Solution #7.12

Without looking at features, we could look at partial dependence plots (also called response curves) to assess how any one feature affects the model's decision. A partial dependence plot shows the marginal effect of a feature on the predicted target of a machine learning model. So, after the model is fit, we can take all the features and start plotting them individually against the loan approval/rejection, while keeping all the other features constant.



For example, if we believe that FICO score has a strong relationship to the predicted probability of loan rejection, then we can plot the loan approvals and rejections as we adjust the FICO score from low to high. Thus, we can get an idea of how features impact the model without explicitly looking at feature weights, and supply reasons for rejection accordingly.

As a concrete example, consider having four applicants: 1, 2, 3, and 4. Assume that the features include annual income, current debt, number of credit cards, and FICO score. Suppose we have the following situation:

1. \$100,000 income, \$10,000 debt, 2 credit cards, and FICO score of 700.
2. \$100,000 income, \$10,000 debt, 2 credit cards, and FICO score of 720.
3. \$100,000 income, \$10,000 debt, 2 credit cards, and FICO score of 600.
4. \$100,000 income, \$10,000 debt, 2 credit cards, and FICO score of 650.

If 3 and 4 were rejected but 1 and 2 were accepted, then we can intuitively reason that a lower FICO score was the reason the model made the rejections. This is because the remaining features are equal, so the model chose to reject 3 and 4 “all-else-equal” versus 1 and 2.

Solution #7.13

To find synonyms, we can first find word embeddings through a corpus of words. Word2vec is a popular algorithm for doing so. It produces vectors for words based on the words’ contexts. Vectors that are closer in Euclidean distance are meant to represent words that are also closer in context and meaning. The word embeddings that are thus generated are weights on the resulting vectors. The distance between these vectors can be used to measure similarity, for example, via cosine similarity or some other similar measure.

Once we have these word embeddings, we can then run an algorithm such as K -means clustering to identify clusters within word embeddings or run a K -nearest neighbor algorithm to find a particular word for which we want to find synonyms. However, some edge cases exist, since word2vec can produce similar vectors even in the case of antonyms; consider the words “hot” and “cold,” for example, which have opposite meanings but appear in many similar contexts (related to temperature or in a Katy Perry song).

Solution #7.14

The bias-variance trade-off is expressed as the following: Total model error = Bias + Variance + Irreducible error. Flexible models tend to have low bias and high variance, whereas more rigid models have high bias and low variance. The bias term comes from the error that occurs when a model underfits data. Having a high bias means that the machine learning model is too simple and may not adequately capture the relationship between the features and the target. An example would be using linear regression when the underlying relationship is nonlinear.

The variance term represents the error that occurs when a model overfits data. Having a high variance means that a model is susceptible to changes in training data, meaning that it is capturing and so reacting to too much noise. An example would be using a very complex neural network when the true underlying relationship between the features and the target is simply a linear one.

The irreducible term is the error that cannot be addressed directly by the model, such as from noise in data measurement.

When creating a machine learning model, we want both bias and variance to be low, because we want to be able to have a model that predicts well but that also doesn’t change much when it is fed new data. The key point here is to prevent overfitting and, at the same time, to attempt to retain sufficient accuracy.

Solution #7.15

Cross validation is a technique used to assess the performance of an algorithm in several resamples/subsamples of training data. It consists of running the algorithm on subsamples of the training data, such as the original data less some of the observations comprising the training data, and evaluating model performance on the portion of the data that was not present in the subsample. This process is repeated many times for the different subsamples, and the results are combined at the end. This step is very important in ML because it reveals the quality and consistency of the model’s true performance.

The process is as follows:

1. Randomly shuffle data into k equally-sized blocks (folds).
2. For each i in fold $1 \dots k$, train the model on all the data except for fold i , and evaluate the validation error using block i .
3. Average the k validation errors from step 2 to get an estimate of the true error.

This process aids in accomplishing the following: (1) avoiding training and testing on the same subsets of data points, which would lead to overfitting, and (2) avoiding using a dedicated validation set, with which no training can be done. The second of these points is particularly important in cases where very little training data is available or the data collection process is expensive. One drawback of this process, however, is that roughly k times more computation is needed than using a dedicated holdout validation set. In practice, cross validation works very well for smaller datasets.

Solution #7.16

Step 1: Clarify Lead Scoring Requirements

Lead scoring is the process of assigning numerical scores for any leads (potential customers) in a business. Lead scores can be based on a variety of attributes, and help sales and marketing teams prioritize leads to try and convert them to customers.

As always, it's smart to ask the interviewer clarifying questions. In this case, learning more about the requirements for the lead scoring algorithm is critical. Questions to ask include:

- Are we building this for our own company's sales leads? Or, are we building an extensible version as part of the Salesforce product?
- Are there any business requirements behind the lead scoring (i.e., does it need to be easy to explain internally and/or externally)?
- Are we running this algorithm only on companies in our sales database (CRM), or looking at a larger landscape of all companies?

For our solution, we'll assume the interviewer means we want to develop a leading scoring model to be used internally — that means using the company's internal sales data to predict whether a prospective company will purchase a Salesforce product.

Step 2: Explain the Features You'd Use

Some elements which should influence whether a prospective company turns into a customer:

- **Firmographic Data:** What type of company is this? Industry? Amount of revenue? Employee count?
- **Marketing Activity:** Have they interacted with marketing materials, like clicking on links within email marketing campaigns? Have employees from that company downloaded whitepapers, read case studies, or clicked on ads? If so, how much activity has there been recently?
- **Sales Activity:** Has the prospective company interacted with sales? How many sales meetings took place, and how recently did the last one take place?
- **Deal Details:** What products are being bought? Some might be harder to close than others. How many seats (licenses) are being bought? What's the size of the deal? What's the contract length?

After selecting features, it is good to conduct the standard set of feature engineering best practices. Note that the model will only be as good as the data and judgement in feature engineering applied — in practice, many companies that predict lead scoring can face issues with missing data or lack of relevant data.

Step 3: Explain Models You'd Use

Lead scoring can be done through building a binary classifier that predicts the probability of a lead converting. In terms of model selection, logistic regression offers a straightforward solution with an easily interpretable result: the resulting log-odds is a probability score for, in this case, purchasing a particular item. However, it cannot capture complex interaction effects between different variables and could also be numerically unstable under certain conditions (i.e., correlated covariates and a relatively small user base).

An alternative to logistic regression would be to use a more complex model, such as a neural network or an SVM. Both are great for dealing with high-dimensional data and with capturing the complex interactions that logistic regression cannot. However, unlike logistic regression, neither is easy to explain, and both generally require a large amount of data to perform well.

A suitable compromise is tree-based models, such as random forests or XGBoost, which typically perform well. With tree-based models, the features that have the highest influence on predictions are readily perceived, a characteristic that could be very useful in this particular case.

Step 4: Model Deployment Nuances

Lastly, it is important to monitor for feature shifts and/or model degradations. As the product line and customer base changes over time, models trained on old data may not be as relevant. For a mature company like Salesforce, for example, it's very likely that companies signing up now aren't exactly like the companies that signed up with Salesforce 5 or 10 years ago. That's why it's important to monitor feature drift and continuously update the model.

Solution #7.17

Collaborative filtering would be a commonly used method for creating a music recommendation algorithm. Such algorithms use data on what feedback users have provided on certain items (songs in this case) in order to decide recommendations. For example, a well-known use case is for movie recommendation on Netflix. However, there are several differences compared to the Netflix case:

- Feedback for music does not have a 1-to-5 rating scale as Netflix does for its movies.
- Music may be subject to repeated consumption; that is, people may watch a movie once or twice but will listen to a song many times over.
- Music has a wider variety (i.e., niche music).
- The scale of music catalog items is much larger than movies (i.e., there are many more songs than movies).

Therefore, a user-song matrix (or a user-artist matrix) would constitute the data for this issue, with the rows of the dataset being users and the columns various songs. However, in considering the first point above, since explicit ratings are lacking, we can employ a binary system to count the number of times a song is streamed and store this count.

We can then proceed with matrix factorization. Say there are M songs and N users in the matrix, which we will label R . Then, we want to solve: $R = PQ^T$

where user preferences are captured by the vectors: $r_{ui} = q_i^T p_u$

Various methods can be used for this matrix factorization; a common one is alternating least squares (ALS), and, since the scale of the data is large, this would likely be done through distributed computing. Once the latent user and song vectors are discovered, then the above dot product will be able to predict the relevance of a particular song to a user. This process can be used directly for recommendation at the user level, where we sort by relevance prediction on songs that the user has not yet streamed. In addition, the vectors given above can be employed in such tasks as assessing similarity between different users and different songs using a method such as kNN (K -nearest neighbors).

Solution #7.18

Mathematically, a convex function f satisfies the following for any two points x and y in the domain of f : $f((1 - \alpha)x + \alpha y) \leq (1 - \alpha)f(x) + \alpha f(y)$, $0 \leq \alpha \leq 1$

That is, the line segment from x to y lies above the function graph of f for any points x and y . Convexity matters because it has implications about the nature of minima in f . Stated more specifically, any local minimum of f is also a global minimum.

Neural networks provide a significant example of non-convex problems in machine learning. At a high level, this is because neural networks are universal function approximators, meaning that they can (with a sufficient number of neurons) approximate any function arbitrarily well. Because not all functions are convex (convex functions cannot approximate non-convex ones), by definition, they must be non-convex. In particular, the cost function for a neural network has a number of local minima; you could interchange parameters of different nodes in various layers and still obtain exactly the same cost function output (all inputs/outputs the same, but with nodes swapped). Therefore, there is no particular global minima, so neural networks cannot be convex.

Solution #7.19

Because information gain is based on entropy, we'll discuss entropy first.

$$\text{The formula for entropy is } Entropy = \sum_{i=1} -P(Y = k) \log P(Y = k)$$

The equation above yields the amount of entropy present and shows exactly how homogeneous a sample is (based on the attribute being split). Consider a case where $k = 2$. Let a and b be two outputs/labels that we are trying to classify. Given these values, the formula considers the proportion of values in the sample that are a and the proportion that are b , with the sample being split on a different attribute.

A completely homogeneous sample will have an entropy of 0. For instance, if a given attribute has values a and b , then the entropy of splitting on that given attribute would be

$$Entropy = -1 * \log_2(1) - 0 * \log_2(0) = 0$$

whereas a completely split (50%-50%) would result in an entropy of 1. A lower entropy means a more homogeneous sample.

Information gain is based on the decrease in entropy after splitting on an attribute.

$$IG(X, Y) = H(Y) - H(Y|X)$$

This concept is better explained with a simple numerical example. Consider the above case again with $k = 2$. Let's say there are 5 instances of value a and 5 instances of value b . Then, we decide to split on some attribute X . When $X = 1$, there are 5 a 's and 1 b , whereas when $X = 0$, there are 4 b 's and 0 a 's.

Now, by splitting on X , we have two classes: $X = 1$ and $X = 0$. However, by splitting on this attribute, we now have $X = 1$, which has 5 a 's and 1 b , while $X = 0$ has 4 b 's and 0 a 's.

$$\text{Entropy(After)} = \left(\frac{4}{10}\right) * \text{Entropy}(X=0) - \left(\frac{6}{10}\right) * \text{Entropy}(X=1)$$

The entropy value for $X=0$ is 0, since the sample is homogeneous (all b 's, no a 's).

$$\text{Entropy}(X=0) = 0$$

$$\text{Entropy}(X=1) = -\left(\frac{1}{6}\right) * \log_2\left(\frac{1}{6}\right) - \left(\frac{5}{6}\right) * \log_2\left(\frac{5}{6}\right) = .65$$

Plug these into the entropy(after) formula to obtain the following:

$$\text{Entropy(After)} = \left(\frac{4}{10}\right) * 0 - \left(\frac{6}{10}\right) * .65 = .39$$

Finally, we can go back to our original formula and obtain information gain: $IG = 1 - .39 = .61$

These results make intuitive sense, since, ideally, we want to split on an attribute that splits the output perfectly. Therefore, we ideally want to split on something that is homogeneous with regards to the output, and this something would thus have an entropy equal to 0.

Solution #7.20

In machine learning, L_1 and L_2 penalization are both regularization methods that prevent overfitting by coercing the coefficients of a regression model towards zero. The difference between the two methods is the form of the penalization applied to the loss function. For a regular regression model, assume the loss function is given by L . Using L_1 regularization, the least absolute shrinkage and selection operator, or Lasso, adds the absolute value of the coefficients as a penalty term, whereas ridge regression uses L_2 regularization, that is, adding the squared magnitude of the coefficients as the penalty term.

The loss function for the two are thus the following:

$$\text{Loss}(L_1) = L + \lambda |w_i| \quad \text{Loss}(L_2) = L + \lambda |w_i^2|$$

where the loss function L is the sum of errors squared, given by the following, where $f(x)$ is the model of interest — for example, a linear regression with p predictors:

$$L = \sum_{i=1}^n (y_i - f(x_i))^2 = \sum_{i=1}^n \left(y_i - \sum_{j=1}^p (x_{ij} w_j) \right)^2 \text{ for linear regression}$$

If we run gradient descent on the weights w , L_1 regularization forces any weight closer to 0, irrespective of its magnitude, whereas with L_2 regularization, the rate at which the weight approaches 0 becomes slower as the weight approaches 0. Because of this, L_1 is more likely to “zero” out particular weights and hence completely remove certain features from the model, leading to models of increased sparseness.

Solution #7.21

The gradient descent algorithm takes small steps in the direction of steepest descent to optimize a particular objective function. The size of the “steps” the algorithm takes are proportional to the negative gradient of the function at the current value of the parameter being sought. The stochastic version of the algorithm, SGD, uses an approximation of the nonstochastic gradient descent algorithm instead of the function’s actual gradient. This estimate is done by using only one randomly selected sample at each step to evaluate the derivative of the function, making this version of the algorithm much faster and more attractive for situations involving lots of data. SGD is also useful when redundancy in the data is present (i.e., observations that are very similar).

Assume function f at some point x and at time t . Then, the gradient descent algorithm will update x as follows until it reaches convergence:

$$x^{t+1} = x^t - \alpha_t \nabla f(x^t)$$

That is, we calculate the negative of the gradient of f and scale that by some constant and move in that direction at the end of each iteration.

Since many loss functions are decomposable into the sum of individual functions, then the gradient step can be broken down into addition of discrete, separate gradients. However, for very large datasets, this process can be computationally intensive, and the algorithm might become stuck at local minima or at saddle points.

Therefore, we use the stochastic gradient descent algorithm to obtain an unbiased estimate of the true gradient without going through all data points by uniformly selecting a point at random and performing a gradient update then and there.

The estimate is therefore unbiased since we have the following:

$$\nabla f(x) = \frac{1}{n} \sum_{i=1}^n \nabla f_i(x)$$

Since the data are assumed to be i.i.d., for the SGD, the expectation of $g(x)$ is: $E[g(x)] = \nabla f(x)$ where $g(x)$ is the stochastic gradient descent.

Solution #7.22

Recall that the ROC curve plots the true positive rate versus the false positive rate. If all scores change simultaneously, then none of the actual classifications change (since thresholds are adjusted), leading to the same true positive and false positive rates, since only the relative ordering of the scores matters. Therefore, taking a square root would not cause any change to the ROC curve because the relative ordering has been maintained. If one application had a score of X and another a score of Y , and if $Y > X$, then $\sqrt{Y} > \sqrt{X}$ still. Only the model thresholds would change.

In contrast, any function that is not monotonically increasing would change the ROC curve, since the relative ordering would not be maintained. Some simple examples are the following:

$$f(x) = -x, f(x) = -x^2, \text{ or a stepwise function.}$$

Solution #7.23

We have: $X \sim N(\mu, \sigma^2)$, and entropy for a continuous random variable is given by the following:

$$H(x) = - \int_{-\infty}^{\infty} p(x) \log p(x) dx$$

For a Gaussian, we have the following: $p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$

Substituting into the above equation yields

$$H(x) = - \int_{-\infty}^{\infty} p(x) \log \sigma\sqrt{2\pi} dx - \int_{-\infty}^{\infty} p(x) \left(-\frac{(x-\mu)^2}{2\sigma^2} \right) \log(e) dx$$

where the first term equals $-\log \sigma\sqrt{2\pi} \int_{-\infty}^{\infty} p(x) dx = -\log \sigma\sqrt{2\pi}$

since the integral evaluates to 1 (by the definition of a probability density function). The second term is given by:

$$\frac{1}{2\sigma^2} \int_{-\infty}^{\infty} p(x)(x - \mu)^2 dx = \frac{\sigma^2}{2\sigma^2} = \frac{1}{2}$$

since the inner term is the expression for the variance. The entropy is therefore as follows:

$$H(x) = \frac{1}{2} + \log \sigma \sqrt{2\pi}$$

Solution #7.24

The standard approach is (1) to construct a large dataset with the variable of interest (purchase or not) and relevant covariates (age, gender, income, etc.) for a sample of platform users and (2) to build a model to calculate the probability of purchase of each item. Propensity models are a form of binary classifier, so any model that can accomplish this could be used to estimate a customer's propensity to buy the product.

In selecting a model, logistic regression offers a straightforward solution with an easily interpretable result: the resulting log-odds is a probability score for, in this case, purchasing a particular item. However, it cannot capture complex interaction effects between different variables and could also be numerically unstable under certain conditions (i.e., correlated covariates and a relatively small user base).

An alternative to logistic regression would be to use a more complex model, such as a neural network or an SVM. Both are great with dealing with high-dimensional data and with capturing the complex interactions that logistic regression cannot. However, unlike logistic regression, neither is easy to explain, and both generally require a large amount of data to perform well.

A good compromise is tree-based models, such as random forests, which are typically highly accurate and are easily understandable. With tree-based models, the features which have the highest influence on predictions are readily perceived, a characteristic that could be very useful in this particular case.

Solution #7.25

Both Gaussian naive Bayes (GNB) and logistic regression can be used for classification. The two models each have advantages and disadvantages, which provide the answer as to which to choose under what circumstances. These are discussed below, along with their similarities and differences:

Advantages:

1. GNB requires only a small number of observations to be adequately trained; it is also easy to use and reasonably fast to implement; interpretation of the results produced by GNB can also be highly useful.
2. Logistic regression has a simple interpretation in terms of class probabilities, and it allows inferences to be made about features (i.e., variables) and identification of the most relevant of these with respect to prediction.

Disadvantages:

1. By assuming features (i.e., variables) to be independent, GNB can be wrongly employed in problems where that does not hold true, a very common occurrence.
2. Not highly flexible, logistic regression may fail to capture interactions between features and so may lose prediction power. This lack of flexibility can also lead to overfitting if very little data are available for training.

Differences:

1. Since logistic regression directly learns $P(Y|X)$, it is a discriminative classifier, whereas GNB directly estimates $P(Y)$ and $P(X|Y)$ and so is a generative classifier.

2. Logistic regression requires an optimization setup (where weights cannot be learned directly through counts), whereas GNB requires no such setup.

Similarities:

1. Both methods are linear decision functions generated from training data.
2. GNB's implied $P(Y|X)$ is the same as that of logistic regression (but with particular parameters).

Given these advantages and disadvantages, logistic regression would be preferable assuming training provided data size is not an issue, since the assumption of conditional independence breaks down if features are correlated. However, in cases where training data are limited or the data-generating process includes strong priors, using GNB may be preferable.

Solution #7.26

Assume we have k clusters and n sample points: $x_1 \dots x_n, \mu_1 \dots \mu_k$

The loss function then consists of minimizing total error using a squared L_2 norm (since it is a good way to measure distance) for all points within a given cluster:

$$L = \sum_{j=1}^k \sum_{x_i \in S_j} \|x_i - \mu_j\|^2$$

Taking the derivatives yields the following: $\frac{\partial L}{\partial \mu_k} = \frac{\partial}{\partial \mu_k} \sum_{x_i \in S_k} (x_i - \mu_k)^T (x_i - \mu_k) = \sum_{x_i \in S_k} 2(x_i - \mu_k)$

For batch gradient descent, the update step is then given by the following:

$$\mu_k = \mu_k + \epsilon \sum_{x_i \in S_k} 2(x_i - \mu_k)$$

However, for stochastic gradient descent, the update step is given by the following:

$$\mu_k = \mu_k + \epsilon (x_i - \mu_k)$$

Solution #7.27

The idea behind the kernel trick is that data that cannot be separated by a hyperplane in its current dimensionality can actually be linearly separable by projecting it onto a higher dimensional space.

$$k(x, y) = \phi(x)^T \phi(y)$$

and we can take any data and map that data to a higher dimension through a variety of functions ϕ . However, if ϕ is difficult to compute, then we have a problem — instead, it is desirable if we can compute the value of k without blowing up the computation.

For instance, say we have two examples and want to map them to a quadratic space. We have the following:

$$\phi(x_1, x_2) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1^2 \\ x_2^2 \\ x_1 x_2 \end{bmatrix}$$

and we can use the following: $k(x, y) = (1 + x^T y)^2 = \phi(x)^T \phi(y)$

If we now change $n = 2$ (quadratic) to arbitrary n , we can have arbitrarily complex ϕ . As long as we perform computations in the original feature space (without a feature transformation), then we avoid the long compute time while still mapping our data to a higher dimension!

In terms of which kernel to choose, we can choose between linear and nonlinear kernels, and these will be for linear and nonlinear problems, respectively. For linear problems, we can try a linear or logistic kernel. For nonlinear problems, we can try either radial basis function (RBF) or Gaussian kernels.

In real-life problems, domain knowledge can be handy — in the absence of such knowledge, the above defaults are probably good starting points. We could also try many kernels, and set up a hyperparameter search (a grid search, for example) and compare different kernels to one another. Based on the loss function at hand, or certain performance metrics (accuracy, F1, AUC of the ROC curve, etc.), we can determine which kernel is appropriate.

Solution #7.28

Assume we have some dataset X consisting of n i.i.d observations: x_1, \dots, x_n

Our likelihood function is then $p(X|\mu, \sigma^2) = \prod_{i=1}^n N(x_i|\mu, \sigma^2)$ where $N(x_i|\mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}}$

and therefore the log-likelihood is given by:

$$\log p(X|\mu, \sigma^2) = \sum_{i=1}^n \log N(x_i|\mu, \sigma^2) = -\frac{1}{2\sigma^2} \sum_{i=1}^n (x_i - \mu)^2 - \frac{n}{2} \log \sigma^2 - \frac{n}{2} \log \pi$$

Taking the derivative of the log-likelihood with respect to μ and setting the result to 0 yields the following:

$$\frac{d \log p(X|\mu, \sigma^2)}{d\mu} = \frac{1}{\sigma^2} \sum_{i=1}^n (x_i - \mu) = 0$$

Simplifying the result yields: $\sum_{i=1}^n x_i = \sum_{i=1}^n \hat{\mu} = n\hat{\mu}$, and therefore the maximum likelihood estimate for μ is given by:

$$\hat{\mu} = \frac{1}{n} \sum_{i=1}^n x_i$$

To obtain the variance, we take the derivative of the log likelihood with respect to σ^2 and set the result equal to 0

$$\frac{d \log p(X|\mu, \sigma^2)}{d\sigma^2} = \frac{1}{2\sigma^4} \sum_{i=1}^n (x_i - \mu)^2 - \frac{n}{2\sigma^2} = 0$$

Simplifying yields the following: $\frac{1}{2\sigma^4} \sum_{i=1}^n (x_i - \mu)^2 = \frac{n}{2\sigma^2}$

$$\sum_{i=1}^n (x_i - \mu)^2 = n\sigma^2$$

The maximum likelihood estimate for the variance is thus given by the following:

$$\hat{\sigma}^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \hat{\mu})^2$$

Solution #7.29

The GMM model assumes a Gaussian probability distribution function, across K classes:

$$p(x) = \sum_{k=1}^K \pi_k N(x|\mu_k, \Sigma_k)$$

where the π coefficients are the mixing coefficients on the clusters and are normalized so that they sum to 1.

The posterior probabilities for each cluster are given by Bayes' rule and can be interpreted as “what is the probability of being in class k given the data x ”:

$$z_k = p(k|x) = \frac{p(k)p(x|k)}{p(x)}$$

and hence: $z_k = p(k|x) = \frac{\pi_k N(x|\mu_k, \Sigma_k)}{\sum_{k=1}^K \pi_k N(x|\mu_k, \Sigma_k)}$

The unknown set of parameters θ consists of the mean and variance parameters for each of the K classes, along with the K coefficients. The likelihood is therefore given by:

$$p(\theta|X) = \prod_{i=1}^n p(x) = \prod_{i=1}^n \sum_{k=1}^K \pi_k N(x|\mu_k, \Sigma_k)$$

and therefore the log-likelihood is $\log p(\theta|X) = \sum_{i=1}^n \log \sum_{k=1}^K \pi_k N(x|\mu_k, \Sigma_k)$

The parameters can be calculated iteratively using expectation-maximization and the information above. After the model has been trained, for any new transaction we can then calculate the posterior probabilities of any new transactions over the K classes as above. If the posterior probabilities calculated are low, then the transaction most likely does not belong to any of the K classes, so we can deem it to be fraudulent.

Solution #7.30

Step 1: Clarify What Churn Is & Why It's Important

First, it is important to clarify with your interviewer what churn means. Generally, the word “churn” defines the process of a platform’s loss of users over time.

To determine what qualifies as a churned user at Robinhood, it’s helpful to first follow the money and understand how Robinhood monetizes. One primary way is by trading activity — whether it is through their Robinhood Gold offering or order flow sold to market makers like Citadel. Thus, a cancellation of their Robinhood Gold membership or a long period of no trading activity could constitute churn. The other way Robinhood monetizes is through a user’s account balance. By collecting interest on uninvested cash and making stock loans to counterparties, Robinhood is incentivized to have user’s manage a large portfolio on the platform. As such, a negligible account balance or portfolio maintained over a period of time — say a quarter — could constitute a churned user.

Churn is a big deal, because even a small monthly churn can compound quickly over time: consider that a 2% monthly churn translates to almost a 27% yearly churn. Since it is much more expensive to acquire new customers than to retain existing ones, businesses with high churn rates will need to continually dedicate more financial resources to support new customer acquisition, which is costly,

and therefore to be avoided if possible. So, if Robinhood is to stay ahead of WeBull, Coinbase, and TD Ameritrade, predicting who will churn, and then helping these at-risk users, is beneficial.

After you've worked with your interviewer to clarify what churn is in this context, and why it's important to mitigate, be sure to ask the obvious question: how is my model output going to be used? If it's not clear how the model will be used for the business, then even if the model has great predictive power, it is not useful in practice.

Step 2: Modeling Considerations

Any classification algorithm could be used to model whether a particular customer would be in the churned or active state. However, models that produce probabilities (e.g., logistic regression) would be preferable if the interest is in the probability of the customer's loss rather than simply a final prediction about whether the customer will be lost or not.

Another key consideration when picking a model in this instance would be model explainability. This is because company representatives likely want to understand the main reasons for churn to support marketing campaigns or customer support programs. In this case, interpretable models such as logistic regression, decision trees, or random forests should be used. However, if by talking with the interviewer you learn that it's okay to simply detect churn, and that explainability isn't required, then less interpretable models like neural networks and SVMs can work.

Step 3: Features We'd Use to Model Churn

Some feature ideas include:

- ***Raw Account Balance:*** Is the portfolio value close to the threshold where it doesn't make sense to check the app anymore (say under \$10)?
- ***Account Balance Trend:*** Is there a negative trend – they used to have \$20k in their account, but have steadily been withdrawing money out of the account?
- ***Experienced Heavy Losses:*** Maybe they recently lost a ton of money trading Dogecoin, making them want to quit investing and rethink their trading strategies (and their life).
- ***Recent Usage Patterns:*** Maybe they used to be a Daily Active User, but recently have started logging in less and less — a sign that the app isn't as important any more.
- ***User Demographics:*** Standard user profile variables like age, gender, and location can also be used to model churn.

It's also wise to collaborate with business users to see their perspectives and to look for basic heuristics they might use that can be factored into the model. For example, maybe the customer support team has some insights into signals that indicate a user will churn out.

After running the model, it is good to double-check the results to see if the feature importance roughly matches what we would intuitively expect; for example, it is unlikely that a higher balance would result in a higher likelihood of churn.

Step 4: Deploying the Churn Model

We want to make sure the various metrics of interest (confusion matrix, ROC curve, F1 score, etc.) are satisfactory during offline training before deploying the model in production. As with any prediction task, it is important to monitor model performance and adjust features as necessary whenever there is new data or feedback from customer-facing teams. This helps prevent model degradation, which is a common problem in real-world ML systems. We'd also continuously conduct error analysis by

looking at where the model is wrong, in order to keep refining the model. Finally, we'd also make sure to A/B test our model to validate its impact.

Solution #7.31

In matrix form, we assume Y is distributed as multivariate Gaussian: $Y \sim N(X\beta, \sigma^2 I)$

$$\text{The likelihood of } Y \text{ given above is } L(\beta, \sigma^2) = (2\sigma^2\pi)^{-\frac{N}{2}} \exp\left(-\frac{1}{2\sigma^2}(X\beta - Y)^T(X\beta - Y)\right)$$

Of which we can take the log in order to optimize:

$$\log L(\beta, \sigma^2) = -\frac{N}{2} \log(2\sigma^2\pi) - \frac{1}{2\sigma^2}(X\beta - Y)^T(X\beta - Y)$$

Note that, when taking a derivative with respect to β , the first term is a constant, so we can ignore it, making our optimization problem as follows:

$$\arg \max_{\beta} -\frac{1}{2\sigma^2}(X\beta - Y)^T(X\beta - Y)$$

We can ignore the constant and flip the sign to rewrite as the following: $\arg \min_{\beta} (X\beta - Y)^T(X\beta - Y)$

which is exactly equivalent to minimizing the sum of the squared residuals.

Solution #7.32

PCA aims to reconstruct data into a lower dimensional setting, and so it creates a small number of linear combinations of a vector x (assume it to be p dimensional) to explain the variance within x . More specifically, we want to find the vector w of weights such that we can define the following linear combination:

$$y_i = w_i^T x = \sum_{j=1}^p w_{ij} x_j$$

subject to the constraint that w is orthonormal and that the following is true:

y_i is uncorrelated with y_j , $\text{var}(y_i)$ is maximized

Hence, we perform the following procedure, in which we first find $y_1 = w_1^T x$ with maximal variance, meaning that the scores are obtained by orthogonally projecting the data onto the first principal direction, w_1 . We then find $y_2 = w_2^T x$, is uncorrelated with y_1 and has maximal variance, and we continue this procedure iteratively until ending with the k th dimension such that

y_1, \dots, y_k explain the majority of variance, $k \ll p$

To solve, note that we have the following for the variance of each y , utilizing the covariance matrix of x : $\text{var}(y_i) = w_i^T \Sigma w_i = w_i^T \Sigma w_i$

Without any constraints, we could choose arbitrary weights to maximize this variance, and hence we will normalize by assuming orthonormality of w , which guarantees the following: $w_i^T w_i = 1$

We now have a constrained maximization problem where we can use Lagrange multipliers. Specifically, we have the function $w_i^T \Sigma w_i - \lambda_i(w_i^T w_i - 1) = 0$, which we differentiate with respect to w to solve the optimization problem:

$$\frac{d}{dw_i} w_i^T \Sigma w_i - \lambda_i(w_i^T w_i - 1) = \Sigma w_i - \lambda_i(w_i) = 0$$

Simplifying, we see that: $\Sigma w_i = \lambda_i(w_i)$

This is the result of an eigen-decomposition, whereby w is the eigenvector of the covariance matrix and λ is this vector's associated eigenvalue. Noting that we want to maximize the variance for each y , we pick: $w_i^T \Sigma w_i = w_i^T \lambda_i w_i = \lambda_i w_i^T w_i = \lambda_i$ to be as large as possible. Hence, we choose the first eigenvalue to be the first principal component, the second largest eigenvalue to be the second principal component, and so on.

Solution #7.33

Logistic regression aims to classify X into one of k classes by calculating the following:

$$\log \frac{P(C = i | X = x)}{P(C = K | X = x)} = \beta_{i0} + \beta_i^T x$$

Therefore, the model is equivalent to the following, where the denominator normalizes the numerator over the k classes:

$$P(C = k | X = x) = \frac{e^{\beta_{k0} + \beta_k^T x}}{\sum_{i=1}^K e^{\beta_{i0} + \beta_i^T x}}$$

The log-likelihood over N observations in general is the following:

$$L(\beta | X, C) = \sum_{i=1}^N \log P(C = k | X = x_i, \beta)$$

Use the following notation to denote classes 1 and 2 for the two-class case:

$$y_i = 1 \text{ if the class is 1, otherwise 0}$$

Then we have the following: $P(C = 2 | X = x_i, \theta) = 1 - P(C = 1 | X = x_i, \theta)$

Using the following notation, $P(C = 1 | X = x_i, \theta) = p(x_i)$ such that the log-likelihood can be written as follows:

$$L(\beta) = \sum_{i=1}^N (y_i \log p(x_i) + (1 - y_i) \log (1 - p(x_i)))$$

Simplifying yields the following:

$$L(\beta) = \sum_{i=1}^N \log(1 - p(x_i)) + \sum_{i=1}^N y_i \log \frac{p(x_i)}{1 - p(x_i)}$$

Substituting for the probabilities yields the following:

$$L(\beta) = \sum_{i=1}^N \log(1 - e^{\beta_0 + \beta_1 x_i}) + \sum_{i=1}^N y_i (\beta_0 + \beta_1 x_i)$$

To maximize this log-likelihood, take the derivative and set it equal to 0

$$\frac{\partial L(\beta)}{\partial \beta} = \sum_{i=1}^N (x_i (y_i - p(x_i))) = 0$$

$$\text{We note that: } \frac{\partial}{\partial \beta} = \sum_{i=1}^N \log(1 - e^{\beta_0 + \beta_1 x_i}) = \frac{\partial}{\partial \beta} \sum_{i=1}^N -\log(1 + e^{\beta_0 + \beta_1 x_i}) = \sum_{i=1}^N -\frac{e^{\beta_0 + \beta_1 x_i}}{1 + e^{\beta_0 + \beta_1 x_i}}$$

$$\text{which is equivalent to the latter half of the above expression: } -\sum_{i=1}^N p(x_i)$$

The solutions to these equations are not closed form, however, and, hence, the above should be iterated until convergence.

Solution #7.34

Step 1: Clarify Details of Discover Weekly

First we can ask some clarifying questions:

- What is the goal of the algorithm?
- Do we recommend just songs, or do we also include podcasts?
- Is our goal to recommend new music to a user, and push their musical boundaries? Or is it to just give them the music they'll want to listen to the most, so they spend more time on Spotify? Said more generally, how do we think about the trade-off of exploration versus exploitation?
- What are the various service-level agreements to consider (e.g., does this playlist need to change week to week if the user doesn't listen to it?)
- Do new users get a Discover Weekly playlist?

Step 2: Describe What Data Features You'd Use

The core features will be user-song interactions. This is because users' behaviors and reactions to various songs should be the strongest signal for whether or not they enjoy a song. This approach is similar to the well-known use case for movie recommendations on Netflix, with several notable differences:

- Feedback for music does not have a 1-to-5 rating scale as Netflix does for its movies.
- Music may be subject to repeated consumption (i.e., people may watch a movie once or twice but will listen to a song many times).
- Music has a wider variety (i.e., niche music).
- The scale of music catalog items is much larger than movies (i.e., there are many more songs than movies).

There is also a variety of other features outside of user-song interactions that could be interesting to consider. For example, we have plenty of metadata about the song (the artist, the album, the playlists that include that song) that could be factored in. Additionally, potential audio features in the songs themselves (tempo, speechiness, instruments used) could be used. And finally, demographic information (age, gender, location, etc.) can also impact music listening preferences — people living in the same region are more likely to have similar tastes than someone living on the other side of the globe.

Step 3: Explain Collaborative Filtering Model Setup

There are two types of recommendation systems in general: collaborative filtering (recommending songs that similar users prefer) and content-based recommendation (recommending similar types of songs). Our answer will use collaborative filtering.

Collaborative filtering uses data from feedback users have provided on certain items (in this case, songs) in order to decide recommendations. Therefore, a user-song matrix (or a user-artist matrix) would constitute the dataset at hand, with the rows of the dataset being users and the columns various songs. However, as discussed in the prior section, since explicit song ratings are lacking, we can proxy liking a song by using the number of times a user streamed it. This song play count is stored for every entry in the user-song matrix.

The output of collaborative filtering is a latent user matrix and a song matrix. Using vectors from these matrices, a dot product denotes the relevance of a particular song to a particular user. This process can be used directly for recommendation at the user level, where we sort by relevance scores on songs that the user has not yet streamed. You can also use these vectors to assess similarity between different users and different songs using a method such as kNN (K -nearest neighbors).

Step 4: Additional Considerations

Also relevant to this discussion are the potential pros and cons of collaborative filtering. For example, one pro is that you can run it in a scalable manner to find correlations behind user-song interactions. On the flip side, one con is the “cold start” problem, where an existing base of data is needed for any given user.

Another important consideration is scale. Since Spotify has hundreds of millions of users, the Discover Weekly algorithm could be updated in batch for various users at different times to help speed up data processing and model training.

Another consideration is the dynamic nature of the problem; the influx of new users and songs, along with fast-changing music trends, would necessitate constant retraining.

Lastly, it is important to consider how you can measure and track the impact of this system over time — collaborative filtering doesn’t come with clear metrics of performance. Ideally, you’d use an A/B test to find that users with the improved recommendations had increased engagement on the platform (as measured by time spent listening, for example).

Solution #7.35

We are attempting to solve for $\text{Var}(\hat{\beta})$

Recall that the parameter estimates have the following closed-form solution in matrix form:

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

To derive the variance of the estimates, recall that for any given random variable X :

$$\text{Var}(X) = E[X^2] - E[X]^2$$

Therefore, we have the following: $\text{Var}(\hat{\beta}) = E(\hat{\beta}^2) - E[\hat{\beta}]^2$

We can evaluate the second term since we can assume the parameter estimates are unbiased.

Therefore, $E[\hat{\beta}] = \beta$

$$\text{Var}(\hat{\beta}) = E(\hat{\beta}^2) - \beta^2$$

Substituting into the closed-form solution yields the following: $\text{Var}(\hat{\beta}) = E[((X^T X)^{-1} X^T y)^2] - \beta^2$

Since least squares assumes that: $y = X\beta + \epsilon$ where $\epsilon \sim N(0, \sigma^2)$, we have the following:

$$\text{Var}(\hat{\beta}) = E[((X^T X)^{-1} X^T (X\beta + \epsilon))^2] - \beta^2$$

$$\text{Var}(\hat{\beta}) = E[(X^T X)^{-1} X^T (X\beta) + (X^T X)^{-1} X^T \epsilon]^2 - \beta^2$$

Note that: $(X^T X)^{-1} X^T X = 1$

So simplifying yields: $\text{Var}(\hat{\beta}) = E[(\beta + (X^T X)^{-1} X^T \epsilon)^2] - \beta^2$

$$\text{Var}(\hat{\beta}) = \beta^2 + E[((X^T X)^{-1} X^T \epsilon)^2] - \beta^2$$

where the middle terms were canceled since the expectation of the error term is 0. Canceling out the first and last squared terms and simplifying the middle part yields the following:

$$\text{Var}(\hat{\beta}) = E[(X^T X)^{-1} X^T X (X^T X)^{-1} \epsilon^2] = (X^T X)^{-1} E[\epsilon^2] = (X^T X)^{-1} \sigma^2$$

SQL & DB Design

CHAPTER 8

Upon hearing the term “data scientist,” buzzwords such as predictive analytics, big data, and deep learning may leap to mind. So, let’s not beat around the bush: data wrangling isn’t the most fun or sexy part of being a data scientist. However, as a data scientist, you will likely spend a great deal of your time working writing SQL queries to retrieve and analyze data. As such, almost every company you interview with will test your ability to write SQL queries. These questions are practically guaranteed if you are interviewing for a data scientist role on a product or analytics team, or if you’re after a data science-adjacent role like data analyst or business intelligence analyst. Sometimes, data science interviews may go beyond just writing SQL queries, and cover the basic principles of database design and other big data systems. This focus on data architecture is particularly true at early-stage startups, where data scientists often take an active role in data engineering and data infrastructure development.

SQL

How SQL Interview Questions Are Asked

Because most analytics workflows require quick slicing and dicing of data in SQL, interviewers will often present you with hypothetical database tables and a business problem, and then ask you to write SQL on the spot to get to an answer. This is an especially common early interview question.

conducted via a shared coding environment or through an automated remote assessment tool. Because of the many different flavors of SQL used by industry, these questions aren't usually testing your knowledge of database-specific syntax or obscure commands. Instead, interviews are designed to test your ability to translate reporting requirements into SQL.

For example, at a company like Facebook, you might be given a table on user analytics and asked to calculate the month-to-month retention. Here, it's relatively straightforward what the query should be, and you're expected to write it. Some companies might make their SQL interview problems more open-ended. For example, Amazon might give you tables about products and purchases and then ask you to list the most popular products in each category. Robinhood may give you a table and ask why users are churning. Here, the tricky part might not be just writing the SQL query, but also figuring out collaboratively with the interviewer what "popular products" or "user churn" means in the first place.

Finally, some companies might ask you about the performance of your SQL query. While these interview questions are rare, and they don't expect you to be a query optimization expert, knowing how to structure a database for performance, and avoid slow running queries, can be helpful. This knowledge can come in handy as well when you are asked more conceptual questions about database design and SQL.

Tips for Solving SQL Interview Questions

First off, don't jump into SQL questions without fully understanding the problem. Before you start whiteboarding or typing out a solution, it's crucial to repeat back the problem so you can be sure you've understood it correctly. Next, try to work backwards, especially if the answer needs multiple joins, subqueries, and common table expressions (CTEs). Don't overwhelm yourself trying to figure out the multiple parts of the final query at the same time. Instead, imagine you had all the information you needed in a single table, so that your query was just a single SELECT statement. Working backwards slowly from this ideal table, one SQL statement at a time, try to end up with the tables you originally started with.

For more general problem-solving tips, be sure to also read the programming interview tips in the coding chapter. Most of what applies to solving coding questions — like showing your work and asking for help if stuck — applies to solving SQL interview questions too.

Basic SQL Commands

Before we cover the must-know SQL commands, a quick note — please don't be alarmed by minor variations in syntax between your favorite query language and our PostgreSQL snippets:

- **CREATE TABLE:** Creates a table in a relational database and, depending on what database you use (e.g., MySQL), can also be used to define the table's schema.
- **INSERT:** Inserts a row (or a set of rows) into a given table.
- **UPDATE:** Modifies already-existing data.
- **DELETE:** Removes a row (or a group of rows) from a database.
- **SELECT:** Selects certain columns from a table. A common part of most queries.
- **GROUP BY:** Groups/aggregates rows having the contents of a specific column or set of columns.
- **WHERE:** Provides a condition on which to filter before any grouping is applied.
- **HAVING:** Provides a condition on which to filter after any grouping is applied.

- ORDER BY: Sorts results in ascending or descending order according to the contents of a specific column or set of columns.
- DISTINCT: Returns only distinct values.
- UNION: Combines results from multiple SELECT statements.

Joins

Imagine you worked at Reddit, and had two separate tables: users and posts.

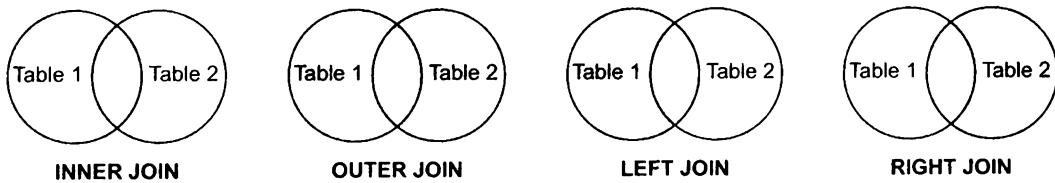
Reddit users

Aa column_name	≡ type
<u>user_id</u>	integer
<u>country</u>	string
active_status	boolean
join_time	datetime

Reddit Posts

Aa column_name	≡ type
<u>post_id</u>	integer
<u>user_id</u>	integer
subreddit_id	integer
title	string
body	string
active_status	boolean
post_time	datetime

Joins are used to combine rows from multiple tables based on a common column. As you can see, the user_id column is the common column between the two tables and links them; hence it is known as a join key. There are four main types of joins:



INNER JOIN

Inner joins combine multiple tables and will preserve the rows where column values match in the tables being combined. The word INNER is optional and is rarely used because it's the default type of join. As an example, we use an inner join to find the number of Reddit users who have made a post:

```

SELECT
    COUNT(DISTINCT u.user_id)
FROM
    users u
JOIN posts p ON u.user_id = p.user_id
  
```

A self join is a special case of an inner join where a table joins itself. The most common use case for a self join is to look at pairs of rows within the same table.

OUTER JOIN

Outer joins combine multiple tables by matching on the columns provided, while preserving all rows. As an example of an outer join, we list all inactive users with posts and all inactive posts from any user:

```
SELECT
  *
FROM
  users u
  OUTER JOIN posts p ON u.user_id = p.user_id
WHERE
  u.active_status = False
  OR p.active_status = False
```

LEFT JOIN

Left joins combine multiple tables by matching on the column names provided, while preserving all the rows from the first table of the join. As an example, we use a left join to find the percentage of users that made a post:

```
SELECT
  COUNT(
    DISTINCT CASE
      WHEN p.post_id IS NOT NULL THEN u.user_id
    END
  ) / COUNT(*) AS pct_users
FROM
  users u
  LEFT JOIN posts p ON u.user_id = p.user_id
```

RIGHT JOIN

Right joins combine multiple tables by matching on the column names provided, while preserving all the rows from the second table of the join. For example, we use the right join to find the percentage of posts made where the user is located in the U.S.:

```
SELECT
  COUNT(
    DISTINCT CASE
      WHEN u.country = 'US' THEN p.post_id
    END
  ) / COUNT(*) AS pct_posts
FROM
  users u
  RIGHT JOIN posts p ON u.user_id = p.user_id
```

Join Performance

Joins are an expensive operation to process, and are often bottlenecks in query runtimes. As such, to write efficient SQL, you want to be working with the fewest rows and columns before joining two tables together. Some general tips to improve join performance include the following:

- Select specific fields instead of using SELECT *
- Use LIMIT in your queries
- Filter and aggregate data before joining
- Avoid multiple joins in a single query

Advanced SQL Commands

Aggregation

For interviews, you need to know how to use the most common aggregation functions like COUNT, SUM, AVG, or MAX:

```
SELECT COUNT(*) FROM users ...
```

Filtering

SQL contains various ways to compare rows, the most common of which use = and <> (not equal), >, and <, along with regex and other types of logical and filtering clauses such as OR and AND. For example, below we filter to active Reddit users from outside the U.S.:

```
SELECT
  *
FROM
  users
WHERE
  active_status = True
  AND country <> 'US'
```

Common Table Expressions and Subqueries

Common Table Expressions (CTEs) define a query and then allow it to be referenced later using an alias. They provide a handy way of breaking up large queries into more manageable subsets of data. For example, below is a CTE which gets the number of posts made by each user, which is then used to get the distribution of posts made by users (i.e., 100 users posted 5 times, 80 users posted 6 times, and so on):

```
WITH user_post_count AS (
  SELECT
    users.user_id,
    COUNT(post_id) AS num_posts
  FROM
```

```

    users
    LEFT JOIN posts on users.user_id = posts.user_id
GROUP BY
    1
)

SELECT
    num_posts,
    COUNT(*) as num_users
FROM
    user_post_count
GROUP BY
    1

```

Subqueries serve a similar function to CTEs, but are inline in the query itself and must have a unique alias for the given scope.

```

SELECT
    num_posts,
    COUNT(*) AS num_users
FROM
(
    SELECT
        users.user_id,
        COUNT(post_id) AS num_posts
    FROM
        users
    LEFT JOIN posts on users.user_id = posts.user_id
    GROUP BY
        1
) u

```

CTEs and subqueries are mostly similar, with the exception that CTEs can be used recursively. Both concepts are incredibly important to know and practice, since most of the harder SQL interview questions essentially boil down to breaking the problem into smaller chunks of CTEs and subqueries.

Window Functions

Window functions perform calculations across a set of rows, much like aggregation functions, but do not group those rows as aggregation functions do. Therefore, rows retain their separate identities even with aggregated columns. Thus, window functions are particularly convenient when we want to use both aggregated and non-aggregated values at once. Additionally, the code is often easier to manage than the alternative: using group by statements and then performing joins on the original input table.

Syntax-wise, window functions require the OVER clause to specify a particular window. This window has three components:

- **Partition Specification:** separates rows into different partitions, analogous to how GROUP BY operates. This specification is denoted by the clause PARTITION BY
- **Ordering Specification:** determines the order in which rows are processed, given by the clause ORDER BY
- **Window Frame Size Specification:** determines which sliding window of rows should be processed for any given row. The window frame defaults to all rows within a partition but can be specified by the clause ROWS BETWEEN (start, end)

For instance, below we use a window function to sum up the total Reddit posts per user, and then add each post_count to each row of the users table:

```
SELECT
  *,
  SUM(posts) OVER (PARTITION BY user_id) AS post_count
FROM
  users u
  LEFT JOIN posts p ON u.user_id = p.user_id
```

Note that a comparable version without using window functions looks like the following:

```
SELECT
  a.*,
  b.post_count
FROM
  users a
  JOIN (
    SELECT
      user_id,
      SUM(posts) AS post_count
    FROM
      users u
      LEFT JOIN posts p ON u.user_id = p.user_id
    GROUP BY
      1
  ) b ON a.user_id = b.user_id
```

As you can see, window functions tend to lead to simpler and more expressive SQL.

LAG and LEAD

Two popular window functions are (LAG) and (LEAD). These are both positional window functions, meaning they allow you to refer to rows after the current row (LAG), or rows before the current row (LEAD). The below example uses LAG so that for every post, it finds the time difference between the post at hand, and the post made right before it in the same subreddit:

```

SELECT
    p.*,
    LAG(post_time, 1) OVER (
        PARTITION BY user_id,
        subreddit_id

        ORDER BY
            post_time ASC
    ) AS prev_subreddit_post_time
FROM
    posts p

```

RANK

Say that for each user, we wanted to rank posts by their length. We can use the window function RANK() to rank the posts by length for each user:

```

SELECT
    *,
    RANK() OVER (
        PARTITION BY user_id
        ORDER BY
            LENGTH(body) DESC
    ) AS rank
FROM
    users u
LEFT JOIN posts p ON u.user_id = p.user_id

```

Databases and Systems

Although knowing all of the database's inner workings isn't strictly necessary, having a high-level understanding of basic database and system design concepts is very helpful. Database interview questions typically do not involve minutiae about specific databases but, instead, focus on how databases generally operate and what trade-offs are made during schema design. For example, you might be asked how you'd set up tables to represent a real-world situation, like storing data if you worked at Reddit. You'd need to define the core tables (users, posts, subreddits) and then define the relationships between. For the Reddit example, posts would have a user_id column for the corresponding user that made the post.

You also may be asked to choose which columns should be indexed, which allows for more rapid lookup of data. For the Reddit example, you would want to index the user_id column, since it's likely heavily used as a join key across many important queries.

While data science interviews don't go into system design concepts as deeply or as often as software engineering and data engineering interviews, it can still show up from time to time. This is particularly the case if you are joining a smaller company, where your data science job might involve creating and

managing data pipelines. Besides generic questions about scaling up data infrastructure, you might be asked conceptual questions about popular large-scale processing frameworks (Hadoop, Spark) or orchestration frameworks (Airflow, Luigi) — especially if you happen to list these technologies on your resume.

Keys & Normalization

Primary keys ensure that each entity has its own unique identifier, i.e., no rows in a table are duplicated with respect their primary key. *Foreign keys*, on the other hand, establish mappings between entities. By using a foreign key to link two related tables, we ensure that data is only stored once in the database. For the Reddit example, in the posts table, the post_id column is the primary key, and each post has a user_id which is a foreign key to the users table.

Item	Primary Key	Foreign Key
Consists of One or More Columns	Yes	Yes
Duplicate Values Allowed	No	Yes
NULLs Allowed	No	Yes
Uniquely Identify Rows in a Table	Yes	Maybe
Number Allowed Per Table	One	Zero or More
Indexed	Automatically Indexed	No Index Automatically created

Keys allow us to split data efficiently into separate tables, but still enforce a logical relationship between two tables. rather than having everything duplicated into one table. This process of generally separating out data to prevent redundancy is called normalization. Along with reducing redundancy, normalization helps you enforce database constraints and dependencies, which improves data integrity.

The disadvantage to normalization is that now we need an expensive join operation between the two related tables. As such, in high-performance systems. denormalization is an optimization technique where we keep redundant data to prevent expensive join operations. This speeds up read times, but at the cost of having to duplicate data. At scale. this can be acceptable since storage is cheap, but compute is expensive.

When normalization comes up in interviews. it often concerns the conceptual setup of database tables: why a certain entity should have a foreign key to another entity, what the mapping relationship is between two types of records (one-to-one, one-to-many, or many-to-many), and when it might be advantageous to denormalize a database.

Properties of Distributed Databases

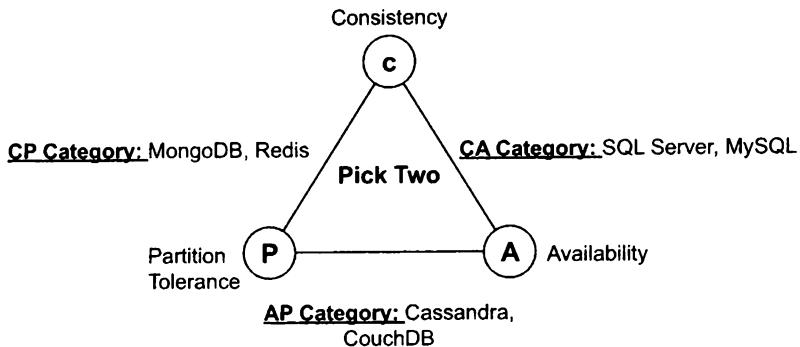
Two concepts, the CAP theorem and the ACID framework, are commonly used to assess theoretical guarantees of databases and are discussed in detail below.

The CAP theorem provides a framework for assessing properties of a distributed database, although only two of the theorem's three specifications can be met simultaneously. The name CAP is an acronym based on the following desirable characteristics of distributed databases:

Consistency: All clients using the database see the same data.

Availability: The system is always available, and each request receives a non-error response, but there's no guarantee that the response contains the latest data.

Partition tolerance: The system functions even if communication between nodes is lost or delayed.



Although the CAP theorem is a theoretical framework, one should consider the real-life trade-offs that need to be made based on the needs of the business and those of the database's users. For example, the Instagram feed focuses on availability and less so on consistency, since what matters is that you get a result instantly when visiting the feed. The penalty for inconsistent results isn't high. It's not going to crush users to see @ChampagnePapi's last post has 57,486 likes (instead of the correct 57,598 likes). In contrast, when designing the service to handle payments on WhatsApp, you'd favor consistency over availability, because you'd want all servers to have a consistent view of how much money a user has to prevent people from sending money they didn't have. The downside is that sometimes sending money takes a minute or a payment fails and you are asked to re-try. Both are reasonable trade-offs in order to prevent double-spend issues.

The second principle for measuring the correctness and completeness of a database transaction is called the *ACID framework*. ACID is an acronym derived from the following desirable characteristics:

- **Atomicity**: an entire transaction occurs as a whole or it does not occur at all (i.e., no partial transactions are allowed). If a transaction aborts before completing, the database does a “rollback” on all such incomplete transactions. This prevents partial updates to a database, which cause data integrity issues.
- **Consistency**: integrity constraints ensure that the database is consistent before and after a given transaction is completed. Appropriate checks handle any referential integrity for primary and foreign keys.
- **Isolation**: transactions occur in isolation so that multiple transactions can occur independently without interference. This characteristic properly maintains concurrency.
- **Durability**: once a transaction is completed, the database is properly updated with the data associated with that transaction, so that even a system failure could not remove that data from it.

The ACID properties are particularly important for online transactional processing (OLTP) systems, where databases handle large volumes of transactions conducted by many users in real time.

Scaling Databases

Traditionally, database scaling was done by using full-copy clusters where multiple database servers (each referred to as a node within the cluster) contained a full copy of the data, and a load balancer would round robin incoming requests. Since each database server had a full copy of the data, each node experienced the issues mentioned in the CAP theorem discussed above (especially during high-load periods). With the advent of the cloud, the approach towards scaling databases has evolved rapidly.

Nowadays, the cloud makes two main strategies to scaling feasible: vertical and horizontal scaling.

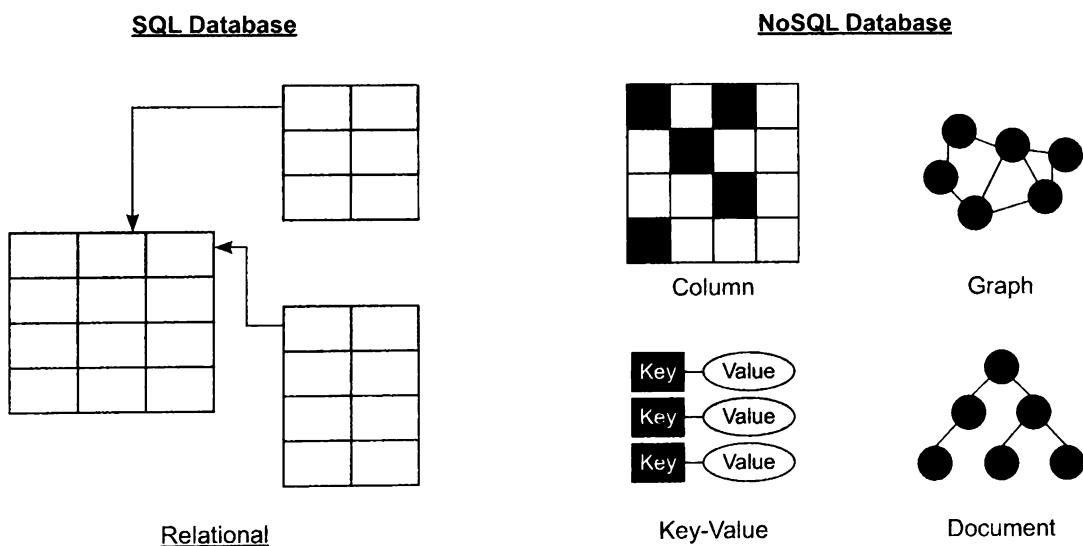
Vertical scaling, also known as scaling up, involves adding CPU and RAM to existing machines. This approach is easy to administer and does not require changing the way a system is architected. However, vertical scaling can quickly become prohibitively expensive, eventually limiting the scope for upgrades. This is because certain machines may be close to their physical limits, making it practically impossible to replace them with more performant servers.

In horizontal scaling, also known as scaling out, more commodity machines (nodes) are added to the resource pool. In comparison to vertical scaling, horizontal scaling has a much cheaper cost structure and has better fault tolerance than vertical scaling. However, as expected, there are trade-offs with this approach. With many more nodes, you have to deal with issues that arise in any distributed system, like handling data consistency between nodes. Therefore, horizontal scaling offers a greater set of challenges in infrastructure management compared to vertical scaling.

Sharding, in which database rows themselves are split across nodes in a cluster, is a common example of horizontal scaling. For all tables, each node has the same schema and columns as the original table, but the data are stored independently of other shards. To split the rows of data, a sharding mechanism determines which node (*shard*) that data for a given key should exist on. This sharding mechanism can be a hash function, a range, or a lookup table. The same operations apply for reading data as well, and so, in this way, each row of data is uniquely mapped to one particular shard.

Relational Databases vs. NoSQL Databases

Relational databases, like MySQL and Postgres, have a table-based structure with a fixed, pre-defined schema. In contrast, NoSQL databases (named because they are “non-SQL” and “non-relational”) store data in a variety of forms rather than in a strict table-based structure.



One type of NoSQL database is the document database. MongoDB, the most popular document database, associates each record with a document. The document allows for arbitrarily complex, nested, and varied schemas inside it. This flexibility allows for new fields to be trivially added compared to a relational database, which has to adhere to a pre-defined schema.

Another type of NoSQL database is the graph database. Neo4J is a well-known graph database, which stores each data record along with direct pointers to all the other data records it is connected to.

By making the relationships between the data as important as storing the data itself, graph databases allow for a more natural representation of nodes and edges, when compared to relational databases.

BASE Consistency Model

Analogous to the ACID consistency model for relational databases, the BASE model applies to NoSQL databases:

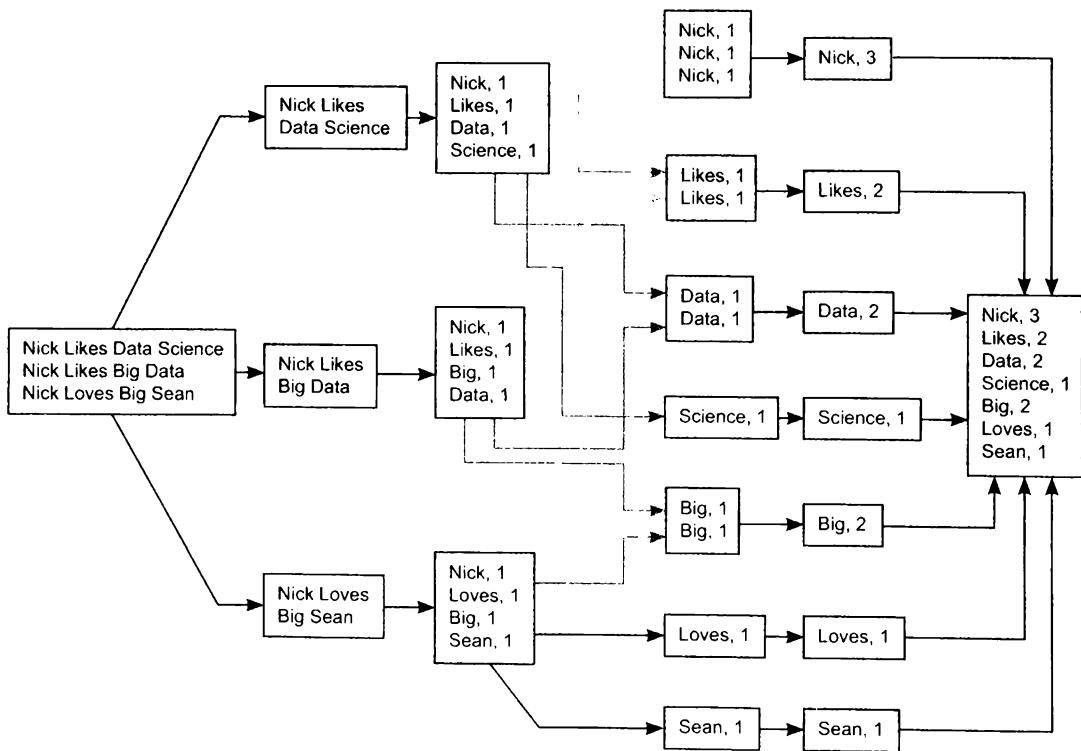
- **Basically Available:** data is guaranteed to be available; there will be a response to any request. This occurs due to the highly distributed approach of NoSQL databases. However, the requested data may be inconsistent and inaccurate.
- **Soft State:** system's state may change over time, even without input. These passive changes occur due to the eventual consistency property.
- **Eventual Consistency:** data will eventually converge to a consistent state, although no guarantees are made on when that will occur.

If you compare and contrast ACID and BASE, you will see that the BASE model puts a stronger focus on availability and scalability but less of an emphasis on data correctness.

MapReduce

MapReduce is a popular data processing framework that allows for the concurrent processing of large volumes of data. MapReduce involves four main steps:

The overall MapReduce word count process



- 1) **Split step:** splits up the input data and distributes it across different nodes
- 2) **Map step:** takes the input data and outputs <key, value> pairs
- 3) **Shuffle step:** moves all the <key, value> pairs with the same key to the same node
- 4) **Reduce step:** processes the <key, value> pairs and aggregates them into a final output

The secret sauce behind MapReduce's efficiency is the shuffle step; by grouping related data onto the same node, we can take advantage of the locality of data. Said another way, by shuffling the related <key, value> pairs needed by the reduce step to the same node rather than sending them to a different node for reducing, we minimize node-to-node communication, which is often the bottleneck for distributed computing.

For a concrete example of how MapReduce works, assume you want to count the frequency of words in a multi-petabyte corpus of text data. The MapReduce steps are visualized on left:

Here's how each MapReduce step operates in more detail:

1. Split step: We split the large corpus of text into smaller chunks and distribute the pieces to different machines.
2. Map step: Each worker node applies a specific mapping function to the input data and writes the output <key, value> pairs to a memory buffer. In this case, our mapping function simply converts each word into a tuple of the word and its frequency (which is always 1). For example, say we had the phrase "hello world" on a single machine. The map step would convert that input into two key value pairs: <"hello", 1> and <"world", 1>. We do this for the entire corpus, so that if our corpus is words big, we end up with key-value pairs in the map step.
3. Shuffle step: Data is redistributed based on the output keys from the prior step's map function, such that tuples with the same key are located on the same worker node. In this case, it means that all tuples of <"hello", 1> will be located on the same worker node, as will all tuples of <"world", 1>, and so on.
4. Reduce step: Each worker node processes each key in parallel using a specified reducer operation to obtain the required output result. In this case, we just sum up the tuple counts for each key, so if there are 5 tuples for <"hello", 1> then the final output will be <"hello", 5>, meaning that the word "hello" occurred 5 times.

Because the shuffle step moved all the "hello" key-value pairs to the same node, the reducer can operate locally and, hence, efficiently. The reducer doesn't need to communicate with other nodes to ask for their "hello" key-value pairs, which minimizes the amount of precious node-to-node bandwidth consumed.

In practice, since MapReduce is just the processing technique, people rely on Hadoop to manage the steps of the MapReduce algorithm. Hadoop involves:

- 1) **Hadoop File System (HDFS):** manages data storage, backup, and replication
- 2) **MapReduce:** as discussed above
- 3) **YARN:** a resource manager which manages job scheduling and worker node orchestration

Spark is another popular open-source tool that provides batch processing similar to Hadoop, with a focus speed and reduced disk operations. Unlike Hadoop, it uses RAM for computation, enabling faster inmemory performance but higher running costs. Additionally, unlike Hadoop, Spark has built-in resource scheduling and monitoring, whereas MapReduce relies on external resource managers like YARN.

SQL & Database Design Questions

Easy Problems

- 8.1. Facebook: Assume you have the below events table on app analytics. Write a query to get the clickthrough rate per app in 2019.

events

Aa column_name	≡ type
<u>app_id</u>	integer
<u>event_id</u>	string ("impression", "click")
timestamp	datetime

- 8.2. Robinhood: Assume you are given the tables below containing information on trades and users. Write a query to list the top three cities that had the most number of completed orders.

trades

Aa column_name	≡ type
<u>order_id</u>	integer
<u>user_id</u>	integer
price	float
quantity	integer
status	string ("complete", "cancelled")
timestamp	datetime

users

Aa column_name	≡ type
<u>user_id</u>	integer
<u>city</u>	string
email	string
signup_date	datetime

- 8.3. New York Times: Assume that you are given the table below containing information on viewership by device type (where the three types are laptop, tablet, and phone). Define “mobile” as the sum of tablet and phone viewership numbers. Write a query to compare the viewership on laptops versus mobile devices.

viewership

Aa column_name	≡ type
<u>user_id</u>	integer
<u>device_type</u>	string
view_time	datetime

- 8.4. Amazon: Assume you are given the table below for spending activity by product type. Write a query to calculate the cumulative spend so far by date for each product over time in chronological order.

total_trans

Aa column_name	≡ type
order_id	integer
user_id	integer
product_id	string
spend	float
trans_date	datetime

- 8.5. eBay: Assume that you are given the table below containing information on various orders made by customers. Write a query to obtain the names of the ten customers who have ordered the highest number of products among those customers who have spent at least \$1000 total.

user_transactions

Aa column_name	≡ type
transaction_id	integer
product_id	integer
user_id	integer
spend	float
trans_date	datetime

- 8.6. Twitter: Assume you are given the table below containing information on tweets. Write a query to obtain a histogram of tweets posted per user in 2020.

tweets

Aa column_name	≡ type
tweet_id	integer
user_id	integer
msg	string
tweet_date	datetime

- 8.7. Stitch Fix: Assume you are given the table below containing information on user purchases. Write a query to obtain the number of people who purchased at least one or more of the same product on multiple days.

purchases

Aa column_name	≡ type
purchase_id	integer
user_id	integer
product_id	integer
quantity	integer
price	float
purchase_time	datetime

- 8.8. LinkedIn: Assume you are given the table below that shows the job postings for all companies on the platform. Write a query to get the total number of companies that have posted duplicate job listings (two jobs at the same company with the same title and description).

job_listings

Aa column_name	≡ type
<u>job_id</u>	integer
<u>company_id</u>	integer
<u>title</u>	string
<u>description</u>	string
<u>post_date</u>	datetime

- 8.9. Etsy: Assume you are given the table below on user transactions. Write a query to obtain the list of customers whose first transaction was valued at \$50 or more.

user_transactions

Aa column_name	≡ type
<u>transaction_id</u>	integer
<u>product_id</u>	integer
<u>user_id</u>	integer
<u>spend</u>	float
<u>transaction_date</u>	datetime

- 8.10. Twitter: Assume you are given the table below containing information on each user's tweets over a period of time. Calculate the 7-day rolling average of tweets by each user for every date.

tweets

Aa column_name	≡ type
<u>tweet_id</u>	integer
<u>msg</u>	string
<u>user_id</u>	integer
<u>tweet_date</u>	datetime

- 8.11. Uber: Assume you are given the table below on transactions made by users. Write a query to obtain the third transaction of every user.

transactions

Aa column_name	≡ type
<u>user_id</u>	integer
<u>spend</u>	float
<u>transaction_date</u>	datetime

- 8.12. Amazon: Assume you are given the table below containing information on customer spend on products belonging to various categories. Identify the top three highest-grossing items within each category in 2020.

product_spend

Aa column_name	≡ type
<u>transaction_id</u>	integer
<u>category_id</u>	integer
<u>product_id</u>	integer
<u>user_id</u>	integer
spend	float
<u>transaction_date</u>	datetime

- 8.13. Walmart: Assume you are given the below table on transactions from users. Bucketing users based on their latest transaction date, write a query to obtain the number of users who made a purchase and the total number of products bought for each transaction date.

user_transactions

Aa column_name	≡ type
<u>transaction_id</u>	integer
<u>product_id</u>	integer
<u>user_id</u>	integer
spend	float
<u>transaction_date</u>	datetime

- 8.14. Facebook: What is a database view? What are some advantages views have over tables?
- 8.15. Expedia: Say you have a database system where most of the queries made were UPDATEs/INSERTs/DELETEs. How would this affect your decision to create indices? What if the queries made were mostly SELECTs and JOINs instead?
- 8.16. Microsoft: What is a primary key? What characteristics does a good primary key have?
- 8.17. Amazon: Describe some advantages and disadvantages of relational databases vs. NoSQL databases.
- 8.18. Capital One: Say you want to set up a MapReduce job to implement a shuffle operator, whose input is a dataset and whose output is a randomly ordered version of that same dataset. At a high level, describe the steps in the shuffle operator's algorithm.
- 8.19. Amazon: Name one major similarity and difference between a WHERE clause and a HAVING clause in SQL.
- 8.20. KPMG: Describe what a foreign key is and how it relates to a primary key.
- 8.21. Microsoft: Describe what a clustered index and a non-clustered index are. Compare and contrast the two.

Medium Problems

- 8.22. Twitter: Assume you are given the two tables below containing information on the topics that each Twitter user follows and the ranks of each of these topics. Write a query to obtain all existing users on 2021-01-01 that did not follow any topic in the 100 most popular topics for that day.

user_topics

<u>Aa</u> column_name	≡ type
<u>user_id</u>	integer
<u>topic_id</u>	integer
<u>follow_date</u>	datetime

topic_rankings

<u>Aa</u> column_name	≡ type
<u>topic_id</u>	integer
<u>ranking</u>	integer
<u>ranking_date</u>	datetime

- 8.23. Facebook: Assume you have the tables below containing information on user actions. Write a query to obtain active user retention by month. An active user is defined as someone who took an action (sign-in, like, or comment) in the current month.

user_actions

<u>Aa</u> column_name	≡ type
<u>user_id</u>	integer
<u>event_id</u>	string ("sign-in", "like", "comment")
<u>timestamp</u>	datetime

- 8.24. Twitter: Assume you are given the tables below containing information on user session activity. Write a query that ranks users according to their total session durations for each session type between the start date (2021-01-01) and the end date (2021-02-01).

sessions

<u>Aa</u> column_name	≡ type
<u>session_id</u>	integer
<u>user_id</u>	integer
<u>session_type</u>	string
<u>duration</u>	integer
<u>start_time</u>	datetime

- 8.25. Snapchat: Assume you are given the tables below containing information on Snapchat users and their time spent sending and opening snaps. Write a query to obtain a breakdown of the time spent sending vs. opening snaps (as a percentage of total time spent) for each of the different age groups.

activities

<u>Aa</u> column_name	≡ type
<u>activity_id</u>	integer

age_breakdown

<u>Aa</u> column_name	≡ type
<u>user_id</u>	integer

<u>user_id</u>	integer	<u>age_bucket</u>	string
type	string ('send', 'open')		
time_spent	float		
activity_date	datetime		

- 8.26. Pinterest: Assume you are given the table below containing information on user sessions, including their start and end times. A session is considered to be concurrent with another user's session if they overlap. Write a query to obtain the user session that is concurrent with the largest number of other user sessions.

sessions

<u>Aa</u> column_name	≡ type
<u>session_id</u>	integer
<u>start_time</u>	datetime
<u>end_time</u>	datetime

- 8.27. Yelp: Assume you are given the table below containing information on user reviews. Define a top-rated business as one whose reviews contain only 4 or 5 stars. Write a query to obtain the number and percentage of businesses that are top rated.

reviews

<u>Aa</u> column_name	≡ type
<u>business_id</u>	integer
<u>user_id</u>	integer
<u>review_text</u>	string
<u>review_stars</u>	integer
<u>review_date</u>	datetime

- 8.28. Google: Assume you are given the table below containing measurement values obtained from a sensor over several days. Measurements are taken several times within a given day. Write a query to obtain the sum of the odd-numbered measurements and the sum of the even-numbered measurements by date.

measurements

<u>Aa</u> column_name	≡ type
<u>measurement_id</u>	integer
<u>measurement_value</u>	float
<u>measurement_time</u>	datetime

- 8.29. Etsy: Assume you are given the two tables below containing information on user signups and user purchases. Of the users who joined within the past week, write a query to obtain the percentage of users that also purchased at least one item.

signups

Aa column_name	≡ type
user_id	integer
signup_date	datetime

user_purchases

Aa column_name	≡ type
user_id	integer
product_id	integer
purchase_amount	float
purchase_date	datetime

- 8.30. Walmart: Assume you are given the following tables on customer transactions and products. Find the top 10 products that are most frequently bought together (purchased in the same transaction).

transactions

Aa column_name	≡ type
transaction_id	integer
product_id	integer
user_id	integer
quantity	integer
transaction_time	datetime

products

Aa column_name	≡ type
product_id	integer
product_name	string
price	float

- 8.31. Facebook: Assume you have the table given below containing information on user logins. Write a query to obtain the number of reactivated users (i.e., those who didn't log in the previous month, who then logged in during the current month).

user_logins

Aa column_name	≡ type
user_id	integer
login_date	datetime

- 8.32. Wayfair: Assume you are given the table below containing information on user transactions for particular products. Write a query to obtain the year-on-year growth rate for the total spend of each product, for each week (assume there is data each week).

user_transactions

Aa column_name	≡ type
transaction_id	integer
product_id	integer
user_id	integer
spend	float
transaction_date	datetime

- 8.33. Stripe: Assume you are given the table below containing information on user transactions for a particular business using Stripe. Write a query to obtain the account's rolling 7-day earnings.

user_transactions

Aa	column_name	≡ type
	transaction_id	integer
	user_id	integer
	amount	float
	transaction_date	datetime

- 8.34. Facebook: Say you had the entire Facebook social graph (users and their friendships). How would you use MapReduce to find the number of mutual friends for every pair of Facebook users?

- 8.35. Google: Assume you are tasked with designing a large-scale system that tracks a variety of search query strings and their frequencies. How would you design this, and what trade-offs would you need to consider?

SQL & Database Design Solutions

Note: Due to the variety of SQL flavors, don't be alarmed by minor variations in syntax. We've written the SQL snippets in this book in PostgreSQL.

Solution #8.1

To get the click-through rate, we use the following query, which includes a SUM along with a IF to obtain the total number of clicks and impressions, respectively. Lastly, we filter the timestamp to obtain the click-through rate for just the year 2019.

```

SELECT
    app_id,
    SUM(IF(event_id = 'click', 1, 0)) / SUM(IF(event_id = 'impression', 1, 0))
    AS ctr
FROM
    events
WHERE
    timestamp >= '2019-01-01'
    AND timestamp <= '2020-01-01'
GROUP BY
    1

```

Solution #8.2

To find the cities with the top three highest number of completed orders, we first write an inner query to join the trades and user table based on the user_id column and then filter for complete orders. Using COUNT DISTINCT, we obtain the number of orders per city. With that result, we then perform a simple GROUP BY on city and order by the resulting number of orders, as shown below:

```

SELECT
    u.city,
    COUNT(DISTINCT t.order_id) AS num_orders
FROM
    trades t
    JOIN users u ON t.user_id = u.user_id
WHERE
    t.status = 'complete'
GROUP BY
    city
ORDER BY
    num_orders DESC
LIMIT
    3

```

Solution #8.3

To compare the viewership on laptops versus mobile devices, we first can use a IF statement to define the device type according to the specifications. Since the tablet and phone categories form the “mobile” device type, we can set laptop to be its own device type (i.e., “laptop”). We can then simply SUM the counts for each device type:

```

SELECT
    SUM(IF(device_type = 'laptop', 1, 0)) AS laptop_views,
    SUM(IF(device_type IN ('phone', 'tablet'), 1, 0)) AS mobile_views
FROM
    viewership

```

Solution #8.4

Since we don’t care about the particular order_id or user_id, we can use a window function to partition by product and order by transaction date. Spending is then summed over every date and product as follows:

```

SELECT
    trans_date,
    product_id,
    SUM(spend) OVER (
        PARTITION BY product_id
        ORDER BY
            trans_date
    ) AS cum_spend
FROM
    total_trans
ORDER BY

```

```
product_id,
trans_date ASC
```

Solution #8.5

In order to obtain a count of products by user, we employ COUNT *product_id* for each user; hence, the GROUP BY is performed over *user_id*. To filter on having spent at least \$1000, we use a HAVING SUM(*spend*) > 1000 clause. Lastly, we order *user_ids* by *product_id* count and take the top 10.

```
SELECT
    user_id,
    COUNT(product_id) AS num_products
FROM
    user_transactions
GROUP BY
    user_id
HAVING
    SUM(spend) > 1000
ORDER BY
    num_products DESC
LIMIT
    10
```

Solution #8.6

First, we obtain the number of tweets per user in 2020 by using a simple COUNT within an initial subquery. Then, we use that tweet column as the bucket within a new GROUP BY and COUNT as shown below:

```
SELECT
    num_tweets AS tweet_bucket,
    COUNT(*) AS num_users
FROM
(
    SELECT
        user_id,
        COUNT(*) AS num_tweets
    FROM
        tweets
    WHERE
        tweet_date BETWEEN '2020-01-01'
        AND '2020-12-31'
    GROUP BY
        user_id
) total_tweets
```

```

GROUP BY
    num_tweets
ORDER BY
    num_tweets ASC

```

Solution #8.7

We can't simply perform a count since, by definition, the purchases must have been made on different days (and for the same products). To address this issue, we use the window function RANK while partitioning by *user_id* and *product_id* and then order the result by purchase time in order to determine the purchase number. From this inner subquery, we then obtain the count of *user_ids* for which purchase number was 2 (note that we don't need above 2 since any purchase number above 2 denotes multiple products).

```

SELECT
    COUNT(DISTINCT user_id)
FROM
(
    SELECT
        user_id,
        RANK() OVER (
            PARTITION BY user_id,
            product_id
            ORDER BY
                CAST(purchase_time AS DATE)
        ) AS purchase_no
    FROM
        purchases
) t
WHERE
    purchase_no = 2

```

Solution #8.8

To find all companies with duplicate listings based on title and description, we can use a RANK() window function partitioning on *company_id*, *job_title*, and *job_description*. Then, we can filter for companies where the largest row number based on those partition fields is greater than 1, which indicates duplicated jobs, and then take a count of the number of companies:

```

WITH job_listing_ranks AS (
    SELECT
        company_id,
        title,
        description,
        ROW_NUMBER() OVER (

```

```

        PARTITION BY company_id,
        title,
        description
        ORDER BY
            post_date
        ) AS rank
    FROM
        job_listings
)
SELECT
    COUNT(DISTINCT company_id)
FROM
(
    SELECT
        company_id
    FROM
        job_listing_ranks
    WHERE
        MAX(rank) > 1
)

```

Solution #8.9

Although we could use a self join on $transaction_date = \text{MIN}(transaction_date)$ for each user, we could also use the `ROW_NUMBER` window function to get the ordering of customer purchases. We could then use that subquery to filter on customers whose first purchase (shown in row one) was valued at 50 dollars or more. Note that this would require the subquery to include spend also:

```

WITH purchase_num AS (
    SELECT
        user_id,
        spend,
        ROW_NUMBER() OVER (
            PARTITION BY user_id
            ORDER BY
                transaction_date ASC
        ) as rounum
    FROM
        user_transactions u
)
SELECT
    user_id
FROM
    purchase_num
WHERE
    rounum = 1
    AND spend >= 50.00

```

Solution #8.10

First, we need to obtain the total number of tweets made by each user on each day, which can be gotten in a CTE using GROUP BY with *user_id* and *tweet_date*, while also applying a COUNT DISTINCT to *tweet_id*. Then, we use a window function on the resulting subquery to take an AVG number of tweets over the six prior rows and the current row (thus giving us the 7-day rolling average), while ordering by *user_id* and *tweet_date*:

```
WITH tweet_counts AS (
    SELECT
        user_id,
        CAST(tweet_date AS date) AS tweet_date,
        COUNT(*) AS num_tweets
    FROM
        tweets
    GROUP BY
        user_id,
        CAST(tweet_date AS date)
)
SELECT
    user_id,
    tweet_date,
    AVG(num_tweets) OVER(
        PARTITION BY user_id
        ORDER BY
            user_id,
            tweet_date ROWS BETWEEN 6 preceding
            AND CURRENT ROW
    ) AS rolling_avg_7d
FROM
    tweet_counts
```

Solution #8.11

First, we obtain the transaction numbers for each user. We can do this by using the ROW_NUMBER window function, where we PARTITION by the *user_id* and ORDER by the *transaction_date* fields, calling the resulting field a transaction number. From there, we can simply take all transactions having a transaction number equal to 3.

```
WITH nums AS (
    SELECT
        * ,
        ROW_NUMBER() OVER (
            PARTITION BY user_id
            ORDER BY
                transaction_date
```

```

    ) AS trans_num
FROM
  transactions
)
SELECT
  user_id,
  spend,
  transaction_date
FROM
  nums
WHERE
  trans_num = 3

```

Solution #8.12

First, we calculate a subquery with total spend by product and category using SUM and GROUP BY. Note that we must filter by a 2020 transaction date. Then, using this subquery, we utilize a window function to calculate the rankings (by spend) for each product category using the RANK window function over the existing sums in the previous subquery. For the window function, we PARTITION by category and ORDER by product spend. Finally, we use this result and then filter for a rank less than or equal to 3 as shown below.

```

WITH product_category_spend AS (
  SELECT
    product_id,
    category_id,
    SUM(spend) AS total_product_spend
  FROM
    product_spend
  WHERE
    transaction_date BETWEEN '2020-01-01'
    AND '2020-12-31'
  GROUP BY
    product_id,
    category_id
),
top_spend AS (
  SELECT
    p.*,
    RANK() OVER (
      PARTITION BY category_id
      ORDER BY
        total_product_spend DESC
    ) AS rnk
  FROM

```

```

        product_category_spend p
)
SELECT
    *
FROM
    top_spend
WHERE
    rnk <= 3
ORDER BY
    category_id,
    rnk DESC

```

Solution #8.13

First, we obtain the latest transaction date for each user. This can be done in a CTE using the RANK window function to get rankings of products purchased per user based on the purchase transaction date. Then, using this CTE, we simply COUNT both the user ids and product ids where the latest rank is 1 while grouping by each transaction date.

```

WITH latest_date AS (
    SELECT
        transaction_date,
        user_id,
        product_id,
        RANK() OVER (
            PARTITION BY user_id
            ORDER BY
                CAST(transaction_date AS DATE) DESC
        ) AS days_rank
    FROM
        user_transactions
)
SELECT
    transaction_date,
    COUNT(DISTINCT user_id) AS num_users,
    COUNT(product_id) AS total_products
FROM
    latest_date
WHERE
    days_rank = 1
GROUP BY
    transaction_date
ORDER BY
    transaction_date desc

```

Solution #8.14

A database view is the result of a particular query within a set of tables. Unlike a normal table, a view does not have a physical schema. Instead, a view is computed dynamically whenever it is requested. If the underlying tables that the views reference are changed, then the views will change accordingly.

Views have several advantages over tables:

1. Views can simplify workflows by aggregating multiple tables, thus abstracting the complexity of underlying data or operations.
2. Since views can represent only a subset of the data, they provide limited exposure of the table's underlying data and hence increase data security.
3. Since views do not store actual data, there is significantly less memory overhead.

Solution #8.15

SQL statements that modify the database, like UPDATE, INSERT, and DELETE, need to change not only the rows of the table but also the underlying indexes. Therefore, the performance of those statements depends on the number of indexes that need to be updated. The larger the number of indexes, the longer it takes those statements to execute. On the flip side, indexing can dramatically speed up row retrieval since no underlying indexes need to be modified. This is important for statements performing full table scans, like SELECTs and JOINs.

Therefore, for databases used in online transaction processing (OLTP) workloads, where database updates and inserts are common, indexes generally lead to slower performance. In situations where databases are used for online analytical processing (OLAP), where database modifications are infrequent but searching and joining the data is common, indexes generally lead to faster performance.

Solution #8.16

A primary key uniquely identifies an entity. It can consist of multiple columns (known as a *composite key*) and cannot be NULL.

Characteristics of a good primary key are:

- **Stability:** a primary key should not change over time.
- **Uniqueness:** having duplicate (non-unique) values for the primary key defeats the purpose of the primary key.
- **Irreducibility:** no subset of columns in a primary key is itself a primary key. Said another way, removing any column from a good primary key means that the key's uniqueness property would be violated.

Solution #8.17

Advantages of Relational Databases: Ensure data integrity through a defined schema and the ACID properties. Easy to get started with and use for small-scale applications. Lends itself well to vertical scaling. Uses an almost standard query language, making learning or switching between different types of relational databases easy.

Advantages of NoSQL Databases: Offers more flexibility in data format and representations, which makes working with unstructured or semistructured data easier. Hence, useful when still iterating on the data schema or adding new features/functionality rapidly like in a startup environment. Convenient to scale with horizontal scaling. Lends itself better to applications that need to be highly available.

Disadvantages of Relational Databases: Data schema needs to be known in advance. Altering schemas is possible, but frequent changes to the schema for large tables can cause performance issues. Horizontal scaling is relatively difficult, leading to eventual performance bottlenecks.

Disadvantages of NoSQL Databases: As outlined by the BASE framework, weaker guarantees on data correctness are made due to the soft-state and eventual consistency property. Managing data consistency can also be difficult due to the lack of a predefined schema that's strictly adhered to. Depending on the type of NoSQL database, it can be challenging for the database to handle some types of complex queries or access patterns.

Solution #8.18

At a high level, to shuffle the data randomly, we need to map each row of the input data to a random key. This ensures that the row of input data is randomly sent to a reducer, where it's simply outputted. More concretely, the steps of the MapReduce algorithm are:

1. Map step: Each row is assigned a random value from $1, \dots, k$, where k is the number of reducer nodes available. Therefore, for every key, the output is the tuple (key, row).
2. Shuffle step: Rows with the same input key go to the same reducer.
3. Reduce step: For each record, the row is simply outputted.

Since the reducer only has rows that were filtered randomly for a given value of i , where i is from $1, \dots, k$, the resulting output will be ordered randomly.

Solution #8.19

A couple of answers are possible, but here are some examples:

Similarities:

1. Both clauses are used to limit/filter a given query's results.
2. Both clauses are optional within a query.
3. Usually, queries utilizing one of the two can be transformed to use the other.

Differences:

1. A HAVING clause can follow a GROUP BY statement, but WHERE cannot.
2. A WHERE clause evaluates per row, whereas a HAVING clause evaluates per group.
3. Aggregate functions can be referred to in a logical expression if a HAVING clause is used.

Solution #8.20

Foreign keys are a set of attributes that aid in joining tables by referencing primary keys (although joins can occur without them). Primarily, they exist to ensure data integrity. The table with the primary key is called the parent table, whereas the table with the foreign key is called the child table. Since foreign keys create a link between the two tables, having foreign keys ensures that these links are valid and prevents data from being inserted that would otherwise violate these conditions. Foreign keys can be created during CREATE commands, and it is possible to DROP or ALTER foreign keys.

When designating foreign keys, it is important to think about the *cardinality* -- the relationship between parent and child tables. Cardinality can take on four forms: one-to-one (one row in the parent table maps to one row in the child table), one-to-many (one row in the parent table maps to many rows in the child table), many-to-one (many rows in the parent table map to one row in the

child table), and many-to-many (many rows in the parent table map to many rows in the child table). The particular type of relationship between the parent and child table determines the specific syntax used when setting up foreign keys.

Solution #8.21

Both clustered indexes and non-clustered indexes help speed up queries in a database. With a clustered index, database rows are stored physically on the disk in the same exact order as the index. This arrangement allows you to rapidly retrieve all rows that fall into a range of clustered index values. However, there can only be one clustered index per table since data can only be sorted physically on the disk in one particular way at a time.

In contrast, a non-clustered index does not match the physical layout of the rows on the disk on which the data are stored. Instead, it duplicates data from the indexed column(s) and contains a pointer to the rest of data. A non-clustered index is stored separately from the table data, and hence, unlike a clustered index, multiple non-clustered indexes can exist per table. Therefore, insert and update operations on a non-clustered index are faster since data on the disk doesn't need to match the physical layout as in the case of a clustered index. However, this makes the storage requirement for a non-clustered index higher than for a clustered index. Additionally, lookup operations for a non-clustered index may be slower than that of a clustered index since all queries must go through an additional layer of indirection.

Solution #8.22

First, we need to obtain the top 100 most popular topics for the given date by employing a simple subquery. Then, we need to identify all users who followed no topic included within these top 100 for the date specified. Equivalently, we could identify those that did follow one of these topics and then filter them out of this list of users that existed on 2021-01-01.

Two approaches are as follows:

1. use the MINUS (or EXCEPT) operator and subtract those following a top 100 topic (via an inner join) from the entire user universe
2. use a WHERE NOT EXISTS clause in a similar fashion.

For simplicity, the solution below uses the MINUS operator. Note that we need to filter for date in the *user_topics* table so that we capture only existing users as of 2020-01-01:

```
WITH top_topics AS (
    SELECT
        *
    FROM
        topic_rankings
    WHERE
        ranking_date = '2021-01-01'
        AND rank <= 100
)
SELECT
    DISTINCT user_id
FROM
```

```

    user_topics
WHERE
    follow_date <= '2021-01-01'
MINUS
SELECT
    u.user_id
FROM
    user_topics u
JOIN top_topics t ON u.topic_id = t.topic_id

```

Solution #8.23

In order to calculate user retention, we need to check for each user whether they were active this month versus last month. To bucket days into each month, we need to obtain the first day of the month for the specified date by using DATE_TRUNC. We use a COUNT DISTINCT over *user_id* to obtain the monthly active user (MAU) count for the month. This can be put into a subquery called *curr_month*, and then EXISTS can be used to check it against another subquery for the previous month, *last_month*. In that subquery, ADD_MONTHS can be used with an argument of 1 to get the previous month, thereby allowing us to check for user actions from the previous month (since that would mean they were logged in), as shown below:

```

SELECT
    DATE_TRUNC('month', curr_month.timestamp) AS month,
    COUNT(DISTINCT curr_month.user_id) AS mau
FROM
    user_actions curr_month
WHERE
    EXISTS (
        SELECT
            *
        FROM
            user_actions last_month
        WHERE
            add_months(DATE_TRUNC('month', last_month.timestamp), -1) =
            DATE_TRUNC('month', curr_month.timestamp)
    )
GROUP BY
    DATE_TRUNC('month', curr_month.timestamp)
ORDER BY
    month ASC;

```

Solution #8.24

First, we can perform a CTE to obtain the total session duration by user and session type between the start and end dates. Then, we can use RANK to obtain the rank, making sure to partition by session type and then order by duration as in the query below:

```

WITH user_duration AS (
    SELECT
        user_id,
        session_type,
        SUM(duration) AS duration
    FROM
        sessions
    WHERE
        start_time BETWEEN '2021-01-01'
        AND '2021-02-01'
    GROUP BY
        user_id,
        session_type
)
SELECT
    user_id,
    session_type,
    RANK() OVER (
        partition by user_id
        session_type
        ORDER BY
        user_id
        duration DESC
    ) AS rank
FROM
    user_duration
ORDER BY
    session_type,
    rank DESC

```

Solution #8.25

We can obtain the total time spent on sending and opening using conditional IF statements for each activity type while getting the amount of time spent in a CTE. We can also obtain the total time spent in the same CTE. Next, we take that result and JOIN by the corresponding *user_id* with activities. We filter for just send and open activity types and group by age bucket. Then, using this CTE, we can calculate the percentages of send and open time spent versus overall time spent as follows:

```

WITH time_stats AS (
    SELECT
        age_breakdown.age_bucket,
        SUM(IF(type = 'send', time_spent, 0)) AS send_timespent,
        SUM(IF(type = 'open', time_spent, 0)) AS open_timespent,
        SUM(time_spent) AS total_timespent
    FROM

```

```

    age_breakdown
    JOIN activities ON age_breakdown.user_id = activities.user_id
  WHERE
    activities.type IN ('send', 'open')
  GROUP BY
    age_breakdown.age_bucket
)
SELECT
  age_bucket,
  send_time / total_time AS pct_send,
  open_time / total_time AS pct_open
FROM
  time_stats

```

Solution #8.26

The first step is to determine the query logic for when two sessions are concurrent. Say we have two sessions, session 1 and session 2. Note that there are two cases in which they overlap:

1. If session 1 starts first, then the start time for session 2 is less than or equal to session 1's end time
2. If session 2 starts first, then session 1's end time for session 1 is greater than or equal to session 2's start time

In total, this simplifies to session 2's start time falling between session 1's start time and session 1's end time.

With this in mind, we can calculate the number of sessions that started during the time another session was running by using an inner join and using BETWEEN to check the concurrency case as follows:

```

SELECT
  s1.session_id,
  COUNT(s2.session_id) AS concurrents
FROM
  sessions s1
  JOIN sessions s2 ON s1.session_id != s2.session_id
  AND s2.start_time BETWEEN s1.start_time
  AND s1.end_time
GROUP BY
  s1.session_id
ORDER BY
  concurrents DESC
LIMIT
  1

```

Solution #8.27

First, we need to identify businesses having reviews consisting of only 4 or 5 stars. We can do so by using a CTE to find the lowest number of stars given to a business across all its reviews. Then, we

can use a SUM and IF statement to filter across businesses with a minimum review of 4 or 5 stars to get the total number of top-rated businesses, and then divide this by the total number of businesses to find the percentage of top-rated businesses.

```
WITH min_review AS (
    SELECT
        business_id,
        min(review_stars) AS min_stars
    FROM
        reviews
    GROUP BY
        business_id
)
SELECT
    (1.0 * SUM(IF(min_stars >= 4, 1, 0)) / COUNT(*)) * 100.0 AS top_places_pct
FROM
    min_review
```

Solution #8.28

First, we need to establish which measurements are odd numbered and which are even numbered. We can do so by using the ROW_NUMBER window function over the *measurement_time* to obtain the measurement number during a day. Then, we filter for odd numbers by checking if a measurement's mod 2 is 1 for odds or is 0 for evens. Finally, we sum by date using a conditional IF statement, summing over the corresponding *measurement_value*:

```
WITH measurements_by_count AS (
    SELECT
        CAST(measurement_time AS date) measurement_day,
        measurement_value,
        ROW_NUMBER() OVER (
            PARTITION BY CAST(measurement_time AS date)
            ORDER BY
                measurement_time ASC
        ) AS measurement_count
    FROM
        measurements
)
SELECT
    measurement_day,
    SUM(
        IF(measurement_count % 2 != 0, measurement_value, 0)
    ) AS odd_sum,
    SUM(
        IF(measurement_count % 2 = 0, measurement_value, 0)
    ) AS even_sum
```

```

FROM
    measurements_by_count
GROUP BY
    measurement_day
ORDER BY
    measurement_day ASC

```

Solution #8.29

First, we obtain the latest week's users. To do this, we use NOW for the current time and subtract an INTERVAL of 7 days, thus providing the relevant user IDs to look at. By using LEFT JOIN, we have all signed-in users, and whether they made a purchase or not. Now we take the COUNT of DISTINCT users from the purchase table, divide it by the COUNT of DISTINCT users from the signup table, and then multiply the results by 100 to obtain a percentage:

```

SELECT
    COUNT(DISTINCT p.user_id) / COUNT(DISTINCT s.user_id) * 100 AS
    last_week_pct
FROM
    signups s
    LEFT JOIN user_purchases p ON p.user_id = s.user_id
WHERE
    s.signup_date > NOW() - INTERVAL 7 DAY

```

Solution #8.30

First, we can join the transactions and product tables together based on *product_id* to get the *user_id*, *product_name*, and *transaction_time* for the transactions. With the CTE at hand, we can do a self join to fetch products that were purchased together by a single user by joining on *transaction_id*. Note that we want all pairs of products, but we don't want to overcount, i.e., if user A purchased products X and Y in the same transaction, then we only want to count the (X, Y) transaction once, and not also (Y, X). To handle this, we can use a condition within the inner join that the *product_id* of A is less than that of B (where A and B are the CTE results from before). Lastly, we use a GROUP BY clause for each pair of products and sort by the resulting count, taking the top 10:

```

WITH (
    SELECT
        t.user_id,
        p.product_name,
        t.transaction_id
    FROM
        transactions t
        JOIN product p ON t.product_id = p.product_id
) AS purchase_info

```

```

SELECT
    p1.product_name AS product1,
    p2.product_name AS product2,
    COUNT(*) AS count
FROM
    purchase_info p1
JOIN purchase_info p2 ON p1.transaction_id = p2.transaction_id
    AND p1.product_id < p2.product_id
GROUP BY
    1,
    2
ORDER BY
    3 DESC
LIMIT
    10

```

Solution #8.31

First, we look at all users who did not log in during the previous month. To obtain the last month's data, we subtract an INTERVAL of 1 month from the current month's login date. Then, we use a WHERE EXISTS against the previous month's interval to check whether there was a login in the previous month. Finally, we COUNT the number of users satisfying this condition.

```

SELECT
    DATE_TRUNC('month', current_month.login_date) AS current_month,
    COUNT(*) AS num_reactivated_users
FROM
    user_logins current_month
WHERE
    NOT EXISTS (
        SELECT
            *
        FROM
            user_logins last_month
        WHERE
            DATE_TRUNC('month', last_month.login_date) BETWEEN DATE_TRUNC('month', current_month.login_date) AND DATE_TRUNC('month', current_month.login_date) - INTERVAL '1 month'
    )

```

Solution #8.32

First, we need to obtain the total weekly spend by product using SUM and GROUP BY operations and use DATE_TRUNC on the transaction date to specify a particular week. Using this information, we then calculate the prior year's weekly spend for each product. In particular, we want to take a LAG for 52 weeks, and PARTITION BY product, to calculate that week's prior year spend for the

given product. Lastly, we divide the current total spend by the corresponding previous 52-week lag value:

```

WITH weekly_spend AS (
    SELECT
        DATE_TRUNC('week', transaction_date :: DATE) AS week,
        product_id,
        SUM(spend) AS total_spend
    FROM
        user_transactions
    GROUP BY
        week,
        product_id
),
total_weekly_spend AS (
    SELECT
        w.*,
        LAG(total_spend, 52) OVER (
            PARTITION BY product_id
            ORDER BY
                week ASC
        ) as prev_total_spend
    FROM
        weekly_spend w
)
SELECT
    product_id,
    total_spend,
    prev_total_spend,
    total_spend / prev_total_spend AS spend_yoy
FROM
    total_weekly_spend

```

Solution #8.33

First, we need to obtain the total daily transactions using a simple SUM and GROUP BY operation. Having the daily transactions, we then perform a self join on the table using the condition that the transaction date for one transaction occurs within 7 days of the other, which we can check by using the DATE_ADD function along with the condition that the earlier date doesn't precede the later date:

```

WITH daily_transactions AS (
    SELECT
        CAST(transaction_date AS DATE),
        SUM(amount) AS total_amount
    FROM
        user_transactions

```

```

GROUP BY
    transaction_date
)
SELECT
    t2.transaction_date,
    SUM(t1.amount) AS weekly_rolling_total
FROM
    daily_transactions t1
    INNER JOIN daily_transactions t2 ON t1.transaction_date > DATE_ADD('DAY',
        -7, t2.transaction_date) AND t1.transaction_date <= t2.transaction_date
GROUP BY
    t2.transaction_date
ORDER BY
    t2.transaction_date ASC

```

Solution #8.34

To use MapReduce to find the number of mutual friends for all pairs of Facebook users, we can think about what the end output needs to be and then work backward. Concretely, for all given pairs of users X and Y, we want to identify which friends they have in common, from which we'll derive the mutual friend count. The core of this algorithm is finding the intersection between the friends list for X and the friends list for Y. This operation can be delegated to the reducer. Therefore, it is sensible that the key for our reduce step should be the tuple (X, Y) and that the value to be reduced is the combination of the friends list of X and the friends list of Y. Thus, in the map step, we want to output the tuple (X, Z) for each friend Z that X has.

As an example, assume that X is friends with [W, Y, Z] and Y is friends with [X, Z].

1. Map step: For X, we want to output the following tuples: 1) ((X, W), [W, Y, Z]), 2) ((X, Y), [W, Y, Z]), and 3) ((X, Z), [W, Y, Z]). For Y we want to output the following tuples: 1) ((Y, X), [X, Z]), and 2) ((Y, Z), [X, Z]). Note that the key is sorted, so that (Y, X) → (X, Y).
2. Shuffle step: Each machine is delegated data based on the keys from the map step, i.e., each tuple (X, Y). So, in the previous example, note that the map step outputs the key (X, Y) for both X and Y, and therefore both of the keys are on the same machine. That machine will therefore have the tuple (X, Y) as the key, and will store [W, Y, Z] and [X, Z] to be used in the reduce step.
3. Reduce step: We group by keys and take the intersection of the resulting lists. For the example of (X, Y) → [W, Y, Z], [X, Z], we take the intersection of [W, Y, Z] and [X, Z], which is [Z]. Thus, we return the length of the set (1) for the input (X, Y).

Therefore, we are able to identify Z as the common friend of X and Y, and can return 1 as the number of mutual friends. The process outlined above is repeated in parallel for every pair of Facebook users in order to find the final mutual friend counts between each pair of users.

Solution #8.35

To design a system that tracks search query strings and their frequencies, we can start with a basic keyvalue store. For each search query string, we store the corresponding frequency in a database table containing only those two fields. To build the system at scale, we have two options: vertical scaling or horizontal scaling. For vertical scaling, we would add more CPU and RAM to existing

machines, an action that is not likely to work well at Google's scale. Instead, we should consider horizontal scaling, in which more machines (nodes) are added to a cluster. We would then store search query strings across a large set of nodes and be able to quickly find which node contains a given search query string.

For the actual sharding logic, consisting of mapping query strings to particular shards, several approaches are possible. One way is to use a range of values; for example, we could have 26 shards and map query strings beginning with A to shard 1, B to shard 2, and so on. While this approach is simple to implement, its primary drawback is that the data obtained could be unevenly distributed, meaning that certain shards would need to deal with much more data than others. For example, the shard containing strings starting with the letter 'x' will have much less load than the shard containing strings starting with the letter 'a.'

An alternative sharding scheme could be to use a hash function that maps the query string to a particular shard number. This is another simple solution and would help reduce the problem of all data being mapped to the same shard. However, adding new nodes is troublesome since the hash function must be re-run across all nodes and the data rebalanced. However, those problems can be addressed through a method called "consistent hashing," which aids in data rebalancing when new servers are added.

Coding

CHAPTER 9

Every Superman has his kryptonite, but as a Data Scientist, coding can't be yours. Between data munging, pulling in data from APIs, and setting up data processing pipelines, writing code is a near-universal part of a Data Scientist's job. This is especially true at smaller companies, where data scientists tend to wear multiple hats and are responsible for productionizing their analyses and models. Even if you are the rare Data Scientist that never has to write production code, consider the collaborative nature of the field — having strong computer science fundamentals will give you a leg up when working with software and data engineers.

*To test your programming foundation, Data Science interviews often take you on a stroll down memory lane back to your Data Structures and Algorithms class (**you did take one, right?**). These coding questions test your ability to manipulate data structures like lists, trees, and graphs, along with your ability to implement algorithmic concepts such as recursion and dynamic programming. You're also expected to assess your solution's runtime and space efficiency using Big O notation.*

Approaching Coding Questions

Coding interviews typically last 30 to 45 minutes and come in a variety of formats. Early in the interview process, coding interviews are often conducted via remote coding assessment tools like HackerRank, Codility, or CoderPad. During final-round onsite interviews, it's typical to write code on a whiteboard. Regardless of the format, the approach outlined below to solve coding interview problems applies.

After receiving the problem: Don't jump right into coding. It's crucial first to make sure you are solving the correct problem. Due to language barriers, misplaced assumptions, and subtle nuances that are easy to miss, misunderstanding the problem is a frequent occurrence. To prevent this, make sure to repeat the question back to the interviewer so that the two of you are on the same page. Clarify any assumptions made, like the input format and range, and be sure to ask if the input can be assumed to be non-null or well formed. As a final test to see if you've understood the problem, work through an example input and see if you get the expected output. Only after you've done these steps are you ready to begin solving the problem.

When brainstorming a solution: First, explain at a high level how you could tackle the question. This usually means discussing the brute-force solution. Then, try to gain an intuition for why this brute-force solution might be inefficient, and how you could improve upon it. If you're able to land on a more optimal approach, articulate how and why this new solution is better than the first brute-force solution provided. Only after you've settled on a solution is it time to begin coding.

When coding the solution: Explain what you are coding. Don't just sit there typing away, leaving your interviewer in the dark. Because coding interviews often let you pick the language you write code in, you're expected to be proficient in the programming language you chose. As such, avoid pseudocode in favor of proper compilable code. While there is time pressure, don't take many shortcuts when coding. Use clear variable names and follow good code organization principles. Write well-styled code --- for example, following PEP 8 guidelines when coding in Python. While you are allowed to cut some corners, like assuming a helper method exists, be explicit about it and offer to fix this later on.

After you're done coding: Make sure there are no mistakes or edge cases you didn't handle. Then write and execute test cases to prove you solved the problem.

At this point, the interviewer should dictate which direction the interview heads. They may ask about the time and space complexity of your code. Sometimes they may ask you to refactor and clean the code, especially if you cut some corners while coding the solution. They may also extend the problem, often with a new constraint. For example, they may ask you not to use recursion and instead tell you to solve the problem recursively. Or, they might ask you to not use surplus memory and instead solve the problem in place. Sometimes, they may pose a tougher variant of the problem as a follow-up, which might require starting the problem-solving process all over again.

Space & Time Complexity

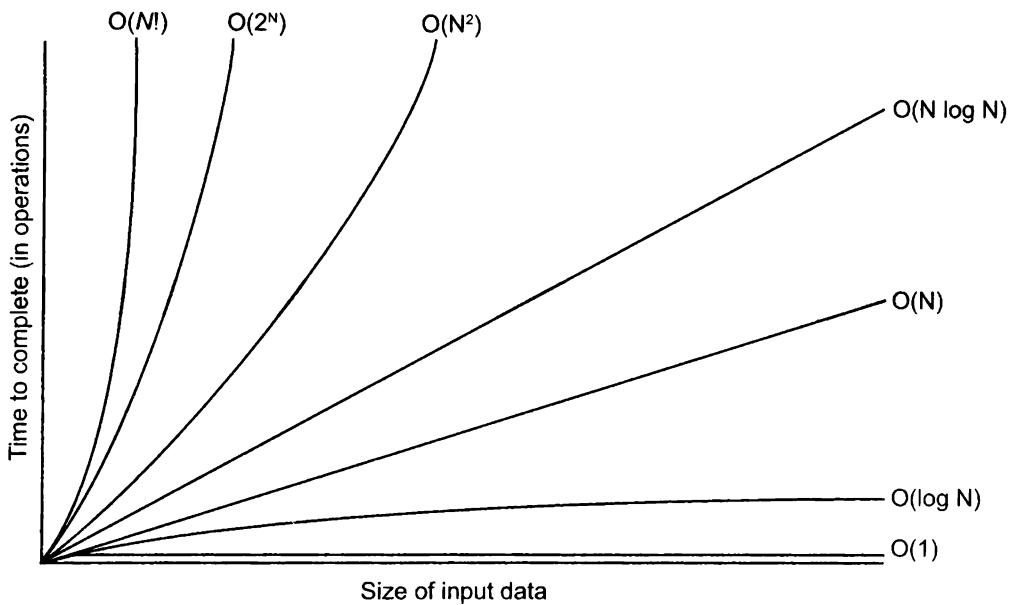
Determining the runtime and space usage (how much memory is utilized) of an algorithm is essential for coding interviews and real-world data science. Because compute and storage resources can be bottlenecks to machine learning model training and deployment, analyzing an algorithm's performance can affect what techniques you choose to implement. Consider OpenAI's GPT-3 language model, which contains over 175 billion parameters and took \$12 million in compute resources to train. Much of the work bringing GPT-3 to the world involved optimizing resource usage to efficiently train such a large model.

Computer scientists analyze and classify the behavior of an algorithm's time and space usage via asymptotic complexity analysis. This technique considers how an algorithm performs when the input size goes toward infinity and characterizes the behavior of the runtime and space used as a function of n . In academic settings, we establish tight bounds on performance in terms of n using Big O (big theta) notation. However, in industry, the technical definitions have been muddled, and we tend to denote these tight bounds on performance using Big O notation.

In the context of companies asking interview questions, we care about not just establishing tight bounds on performance but thinking about the worst-case scenario for this performance. As such, Big O notation often describes the “worst-case upper bound,” or the longest an algorithm would run or the maximal amount of space it would need in the worst case.

For instance, consider an array of size N . Here are the following classes of runtime complexities, from fastest to slowest, using Big O notation:

- $O(1)$: Constant time. Example: getting a value at a particular index from an array
- $O(\log N)$: Logarithmic time. Example: binary search on a sorted array
- $O(N)$: Linear time. Example: using a for-loop to traverse through an array
- $O(N \log N)$: Log-linear time. Example: running mergesort on an array
- $O(N^2)$: Quadratic time. Example: iterating over every pair of elements in an array using a double for-loop
- $O(2^N)$: Exponential time. Example: recursively generating all binary numbers that are N digits long
- $O(N!)$: Factorial time. Example: generating all permutations of an array



The same Big-O runtime analysis concepts apply analogously to space complexity. For example, if we need to store a copy of an input array with N elements, that would be an additional $O(N)$ space. If we wanted to store an adjacency matrix among N nodes, we would need $O(N^2)$ space to keep the N -by- N sized matrix.

For a basic example for both runtime and space complexity analysis, we can look at binary search, where we are searching for a particular value within a sorted array. The code that implements this algorithm is below (with an extra set of conditions that returns the closest if the exact value is not found):

```

def binary_search(a, k):
    lo, hi = 0, len(a) - 1
    best = lo
    while lo <= hi:
        mid = lo + (hi - lo) // 2
        if a[mid] < k:
            lo = mid + 1
        elif a[mid] > k:
            hi = mid - 1
        else:
            best = mid
            break
        if abs(a[mid] - k) < abs(a[best] - k):
            best = mid
    return best

```

If we start the binary search with an input of N elements, then at the next iteration, we would only need to search through $N/2$ elements, and so on. The runtime complexity for binary search is $O(\log N)$ since at each iteration we cut the remaining search space in half. The space complexity would simply be $O(1)$ since the array is size N , and we do not need auxiliary space.

Complexity Analysis Applied to ML Algorithms

For an example of complexity analysis for a machine learning technique, consider the Naive Bayes classifier. Recall that the algorithm aims to calculate

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

for each of the n training data points (of dimension d) for each of the classes, and that $P(B|A)$ is the likelihood probability, and $P(A)$ is the prior probability. In simpler terms, Naive Bayes is counting how many times each of the d features co-occurs within each class.

Now, consider the training runtime. For all n training points, Naive Bayes will look at the posterior and prior probabilities over all d features, for all k classes. This will take $O(nkd)$ total runtime since the operations boil down to a series of counts. The space complexity is just $O(kd)$ to store the probabilities needed to compute results for new data points.

As another example, consider logistic regression. Recall that we need to calculate the following:

$$S(x) = \frac{1}{1 + e^{-x\beta}}$$

for any given x . There are n training data points (each with dimension d); hence, β is a d -by-1 vector of weights. Recall that the goal of logistic regression is to find the optimal decision boundary to split the data into two classes. This involves multiplying each of the n training points with β , which is d -by-1 vector, so the training runtime complexity is $O(d)$. The space complexity is just $O(d)$ to store the weights (β) to classify new data points.

Data Structures

Below is a brief overview of the most common data structures used for coding interviews. The best way to become familiar with each data structure is by implementing a basic version of it in your favorite language. Knowing the Big-O for common operations, like inserting an element or finding an element within the structure, is also essential. The table below can be used for reference:

Data Structure	Time Complexity								Space Complexity	
	Average				Worst					
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion		
Array	O(1)	O(n)	O(n)	O(n)	O(1)	O(n)	O(n)	O(n)	O(n)	
Stack	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)	
Queue	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)	
Linked List	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)	
Hash Map	N/A	O(1)	O(1)	O(1)	N/A	O(n)	O(n)	O(n)	O(n)	
Binary Search Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)	

Arrays

An array is a series of consecutive elements stored sequentially in memory. Arrays are optimal for accessing elements at particular indices, with an O(1) access and index time. However, they are slower for searching and deleting a specific value, with an O(N) runtime, unless sorted. An array's simplicity makes it one of the most commonly used data structures during coding interviews.

Common array interview questions include:

- Moving all the negative elements to one side of an array
- Merging two sorted arrays
- Finding specific sub-sequences of integers within the array, such as the longest consecutive subsequence or the consecutive subsequence with the largest sum

A frequent pattern for array interview questions is the existence of a straightforward brute-force solution that uses O(n) space, and a more clever solution that uses the array itself to lower the space complexity down to O(1). Another pattern we've seen when dealing with arrays is the prevalence of off-by-1 errors — it's easy to crash the program by accidentally reading past the last element of an array.

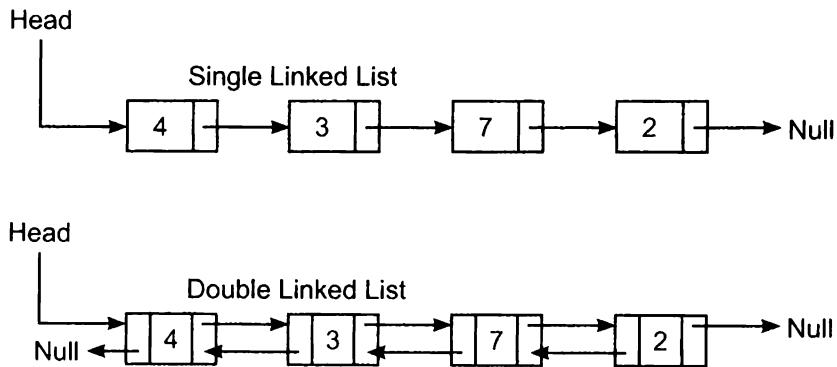
For jobs where Python knowledge is important, interviews may cover list comprehensions, due to their expressiveness and ubiquity in codebases. As an example, below, we use a list comprehension to create a list of the first 10 positive even numbers. Then, we use another list comprehension to find the cumulative sum of the first list:

```
a = [x*2 for x in range(1, 11)] # list creation
c = [sum(a[:x]) for x in range(len(a)+1)] # cumulative sum
```

Arrays are also at the core of linear algebra since vectors are represented as 1-D arrays, and matrices are represented by 2-D arrays. For example, in machine learning, the feature matrix X can be represented by a 2-D array, with one dimension as the number of data points (n) and the other as the number of features (d).

Linked Lists

A linked list is composed of nodes with data that have pointers to other nodes. The first node is called the head, and the last node is called the tail. Linked lists can be circular, where the tail points to the head. They can also be doubly linked, where each node has a reference to both the previous and next nodes. Linked lists are optimal for insertion and deletion, with $O(1)$ insertion time at the head or tail, but are worse for indexing and searching, with a runtime complexity of $O(N)$ for indexing and $O(N)$ for search.



Common linked list questions include:

- Reversing a linked list
- Detecting a cycle in a linked list
- Removing duplicates from a sorted linked list
- Checking if a linked list represents a palindrome

As an example, below we reverse a linked list. Said another way, given the input linked list $4 \rightarrow 1 \rightarrow 3 \rightarrow 2$, we want to write a function which returns $2 \rightarrow 3 \rightarrow 1 \rightarrow 4$. To implement this, we first start with a basic node class:

```
class Node:
    def __init__(self, val):
        self.val = val
        self.next = None
```

Then we create the `LinkedList` class, along with the method to reverse its elements. The `reverse` function iterates through each node of the linked list. At each step, it does a series of swaps between the pointers of the current node and its neighbors.

```
class LinkedList:
    def __init__(self):
        self.head = None

    def reverse(self):
        prev = None
```

```

curr = self.head
while curr:
    next = curr.next
    curr.next = prev
    prev = curr
    curr = next
self.head = prev

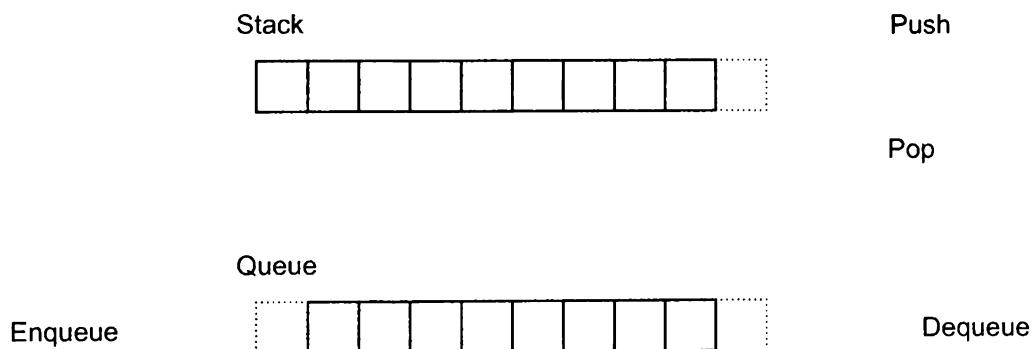
```

Like array interview questions, linked list problems often have an obvious brute-force solution that uses $O(n)$ space, but then also a more clever solution that utilizes the existing list nodes to reduce the memory usage to $O(1)$. Another commonality between array and linked list interview solutions is the prevalence of off-by-one errors. In the linked list case, it's easy to mishandle pointers for the head or tail nodes.

Stacks & Queues

A stack is a data structure that allows adding and removing elements in a last-in, first-out (LIFO) order. This means the element that is added last is the first element to be removed. Another name for adding and removing elements from a stack is pushing and popping. Stacks are often implemented using an array or linked list.

A queue is a data structure that allows adding and removing elements in a first-in, first-out (FIFO) order. Queues are also typically implemented using an array or linked list.



The main difference between a stack and a queue is the removal order: in the stack, there is a LIFO order, whereas in a queue it's a FIFO order. Stacks are generally used in recursive operations, whereas queues are used in more iterative processes.

Common stacks and queues interview questions include:

- Writing a parser to evaluate regular expressions (regex)
- Evaluating a math formula using order of operations rules
- Running a breadth-first or depth-first search through a graph

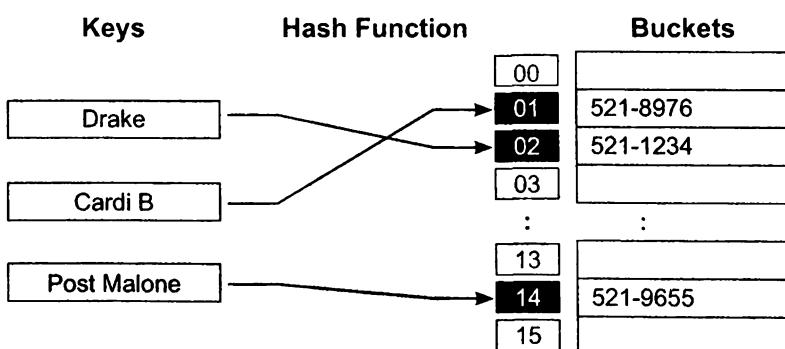
An example interview problem that uses a stack is determining whether a string has balanced parentheses. Balanced, in this case, means every type of left-side parentheses is accompanied by valid right-side parentheses. For instance, the string “({}()){}” is correctly balanced, whereas the string “{{()}}” is not balanced, due to the last character, ‘)’. The algorithm steps are as follows:

- 1) Starting parentheses (left-sided ones) are pushed onto the stack.
- 2) Ending parentheses (right-sided ones) are verified to see if they are of the same type as the most recently seen left-side parentheses on the stack.
- 3) If the parentheses are of the same type, pop from the stack. If they don't match, return false since the parentheses are mismatched.
- 4) Continue parsing the input until it's completely processed, and the stack is empty (every pair of parentheses was correctly accounted for), in which case, return true. Or, if the stack is not empty, in which case, return false.

```
def check_balance(s):
    left_side = ["(", "{", "["] # left parentheses
    right_side = [")", "}", "]"] # right parentheses
    stack = [] # stack
    for i in s:
        if i in left_side:
            stack.append(i) # push onto stack
        elif i in right_side:
            pos = right_side.index(i) # get right char
            # check match
            if len(stack) == 0 or (left_side[pos] != stack[len(stack)-1]):
                return False
            else:
                stack.pop() # remove and continue
    if len(stack) == 0: # balanced
        return True
    else:
        return False
```

Hash Maps

A hash map stores key-value pairs. For every key, a hash map uses a hash function to compute an index, which locates the bucket where that key's corresponding value is stored. In Python, a dictionary offers support for key-value pairs and has the same functionality as a hash map.



While a hash function aims to map each key to a unique index, there will sometimes be “collisions” where different keys have the same index. In general, when you use a good hash function, expect the elements to be distributed evenly throughout the hash map. Hence, lookups, insertions, or deletions for a key take constant time.

Due to their optimal runtime properties, hash maps make a frequent appearance in coding interview questions.

Common hash map questions center around:

- Finding the unions or intersection of two lists
- Finding the frequency of each word in a piece of text
- Finding four elements a, b, c and d in a list such that $a + b = c + d$

An example interview question that uses a hash map is determining whether an array contains two elements that sum up to some value. For instance, say we have a list [3, 1, 4, 2, 6, 9] and k . In this case, we return true since 2 and 9 sum up to 11.

The brute-force method to solving this problem is to use a double for-loop and sum up every pair of numbers in the array, which provides an $O(N^2)$ solution. But, by using a hash map, we only have to iterate through the array with a single for-loop. For each element in loop, we’d check whether the complement of the number (target - that number) exists in the hash map, achieving an $O(N)$ solution:

```
def check_sum(a, target):
    d = {} # create dictionary
    for i in a:
        if (target - i) in d: # check hashmap
            return True
        else:
            d[i] = i # add to hashmap
    return False
```

Due to a hash function’s ability to efficiently index and map data, hashing functions are used in many real-world applications (in particular, with regards to information retrieval and storage). For example, say we need to spread data across many databases to allow for data to be stored and queried efficiently while distributed. Sharding, covered in depth in the databases chapter, is one way to split the data. Sharding is commonly implemented by taking the given input data, and then applying a hash function to determine which specific database shard the data should reside on.

Trees

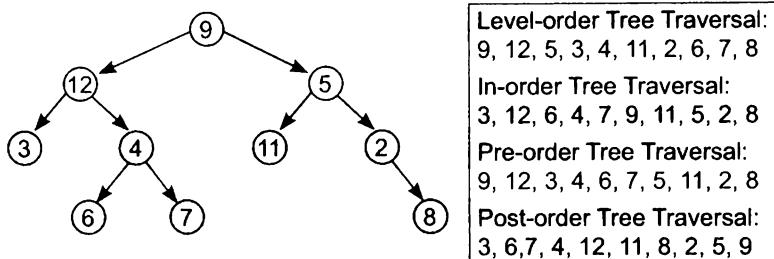
A tree is a basic data structure with a root node and subtrees of children nodes. The most basic type of tree is a binary tree, where each node has at most two children nodes. Binary trees can be implemented with a left and right child node, like below:

```
class TreeNode:
    def __init__(self, val):
        self.val = val
        self.left = None
        self.right = None
```

There are various types of traversals and basic operations that can be performed on trees. For example, in an in-order traversal, we first process the left subtree of a node, then process the current node, and, finally, process the right subtree:

```
def inorder(node):
    if node is None:
        return []
    else:
        return inorder(node.left) + [node.val] + inorder(node.right)
```

The two other closely related traversals are post-order traversal and pre-order traversal. A simple way to remember how these three algorithms work is by remembering that the “post/pre/in” refers to the placement of the processing of the root value. Hence, a post-order traversal processes the left child node first, then the right child node and, in the end, the root node. A pre-order traversal processes the root node first, then the left child node, and then, the right child node.

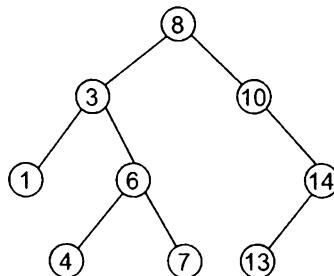


For searching, insertion, and deletion, the worst-case runtime for a binary tree is $O(N)$, where N is the number of nodes in the tree.

Common tree questions involve writing functions to get various properties of a tree, like the depth of a tree or the number of leaves in a tree. Oftentimes, tree questions boil down to traversing through the tree and recursively passing some data in a top-down or a bottom-up manner. Coding interview problems also often focus on two specific types of trees: Binary Search Trees and Heaps.

Binary Search Trees

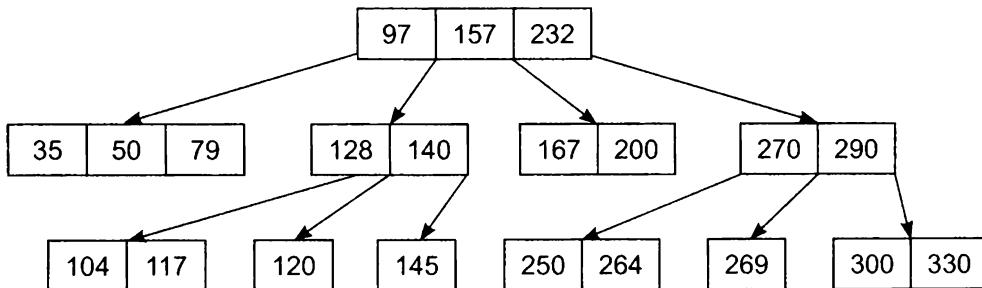
A binary search tree (BST) is composed of a series of nodes, where any node in a left subtree is smaller than or equal to the root, and any node in the right subtree is larger than or equal to the root. When BSTs are height balanced so no one leaf is much deeper than another leaf from the root, searching for elements becomes efficient. To demonstrate, consider searching for the value 9 in the balanced BST below:



Example of a Binary Search Tree

To find 9, we first examine the root value, 8. Since 9 is greater than 8, the node containing 9, if it exists, would have to be on the right side of the tree. Thus, we've cut the search space in half. Next, we compare against the node 10. Since 9 is less than 10, the node, should it exist, has to be on the left of 10. Again, we've cut the search space in half. In conclusion, since 10 doesn't have a left child, we know 9 doesn't occur in the tree. By cutting the search space in half at each iteration, BSTs support search, insertion, and deletion in $O(\log N)$ runtime.

Because of their lookup efficiency, BSTs show up frequently not just in coding interviews but in real-life applications. For instance, B-trees, which are used universally in database indexing, are a generalized version of BSTs. That is, they allow for more than 2 nodes (up to M children), but offer a searching and insertion process similar to that of BST. These properties allow B-trees to have $O(\log N)$ lookup and insertion runtimes similar to that of BSTs, where N is the total number of nodes in the B-tree. Because of the logarithmic growth of the tree depth, database indexes with millions of records often only have a B-tree depth of four or five layers.



Example of a B-Tree

Common BST questions cover:

- Testing if a binary tree has the BST property
- Finding the k -th largest element in a BST
- Finding the lowest common ancestor between two nodes (the closest common node to two input nodes such that both input nodes are descendants of that node)

An example implementation of a BST using the `TreeNode` class, with an `insert` function, is as follows:

```

class TreeNode:
    def __init__(self, val):
        self.val = val
        self.left = None
        self.right = None

class BST:
    def __init__(self, val):
        self.root = TreeNode(val)

    def insert(self, node, val):
        if node is not None:
            if val < node.val:
                if node.left is None:

```

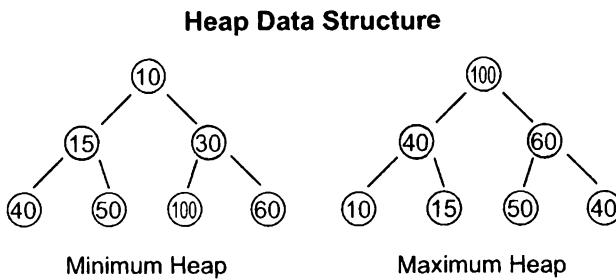
```

        node.left = TreeNode(val)
    else:
        self.insert(node.left, val)
    else:
        if node.right is None:
            node.right = TreeNode(val)
        else:
            self.insert(node.right, val)
    else:
        self.root = TreeNode(val)
return

```

Heaps

Another common tree data structure is a heap. A max-heap is a type of heap where each parent node is greater than or equal to any child node. As such, the largest value in a max-heap is the root value of the tree, which can be looked up in $O(1)$ time. Similarly, for a min-heap, each parent node is smaller than or equal to any child node, and the smallest value lies at the root of the tree and can be accessed in constant time.



To maintain the heap property, there is a sequence of operations known as “heapify,” whereby values are “bubbled up/down” within the tree based on what value is being inserted or deleted. For example, say we are inserting a new value into a min-heap. This value starts at the bottom of the heap and then is swapped with its parent node (“bubbled up”) until it is no longer smaller than its parent (in the case of a min-heap). The runtime of this heapify operation is the height of the tree, $O(\log N)$.

In terms of runtime, inserting or deleting is $O(\log N)$, because the heapify operation runs to maintain the heap property. The search runtime is $O(N)$ since every node may need to be checked in the worst-case scenario. As mentioned earlier, heaps are optimal for accessing the min or max value because they are at the root, i.e., $O(1)$ lookup time. Thus, consider using heaps when you care mostly about finding the min or max value and don’t need fast lookups or deletes of arbitrary elements.

Commonly asked heap interview questions include:

- finding the K largest or smallest elements within an array
- finding the current median value in a stream of numbers
- sorting an almost-sorted array (where elements are just a few places off from their correct spot)

To demonstrate the use of heaps, below we find the k -largest elements in a list, using the `heapq` package in Python:

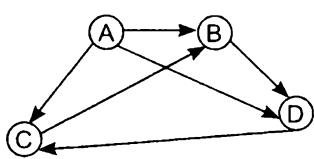
```

import heapq
k = 5
a = [13, 5, 2, 6, 10, 9, 7, 4, 3]
heapq.heapify(a) # creates heap
heapq.nlargest(k, a) # finds k-largest

```

Graphs

Graphs are composed of nodes (vertices) and a set of edges that connect the nodes. Edges are often represented in two ways: an adjacency matrix or an adjacency list. In the case of an adjacency matrix, there is a n -by- n matrix for n -nodes where the entries are either Boolean values (denoting an edge is present or not between two nodes) or a weight. In contrast, an adjacency list stores a list of neighbors for each node.



To Graph

	A	B	C	D
A	0	1	1	1
B	0	0	0	1
C	0	1	0	0
D	0	0	1	0

To Adjacency Matrix

Node	Adjacent Node(s)
A	B C D
B	D
C	B
D	C

To Adjacency List

The lookup time to check whether two nodes are neighbors for an adjacency matrix is $O(1)$. For an adjacency list, it could be $O(N)$ in the worst case (if a node has edges to every other node). What an adjacency list lacks in time efficiency compared to an adjacency matrix, it makes up for in space efficiency. For a large graph with n nodes that are sparse (the N nodes don't have many connections to each other), an adjacency list offers a compact way to represent the edges, versus an adjacency matrix which requires you to store an N^2 sized matrix in memory. For instance, consider the Facebook friendship graph, which has 2 billion users (nodes), but a maximum of only 5,000 connections per node (the 5,000 friend limit). A full adjacency matrix would need to be 2 billion rows by 2 billion columns, whereas an adjacency list would require considerably less memory.

An example implementation of a graph is below. The `Vertex` class uses an adjacency list to keep track of its neighbors with weights:

```

class Vertex:
    def __init__(self, val):
        self.val = val
        self.neighbors = {}

    def add_to_neighbors(self, neighbor, w): # add to neighbor dict with weight
        self.neighbors[neighbor] = w

    def get_neighbors(self):
        return self.neighbors.keys()

```

```

class Graph:
    def __init__(self):
        self.vertices = {}

    def add_vertex(self, val):
        new_vertex = Vertex(val)
        self.vertices[val] = new_vertex
        return new_vertex

    def add_edge(self, u, v, weight): # add edge from u to v
        if u not in self.vertices:
            self.add_vertex(Vertex(u))
        if v not in self.vertices:
            self.add_vertex(Vertex(v))
        self.vertices[u].add_to_neighbors(self.vertices[v], weight)

    def get_vertex(self, val):
        return self.vertices[val]

    def get_vertices(self):
        return self.vertices.keys()

    def iter_(self):
        return iter(self.vertices)

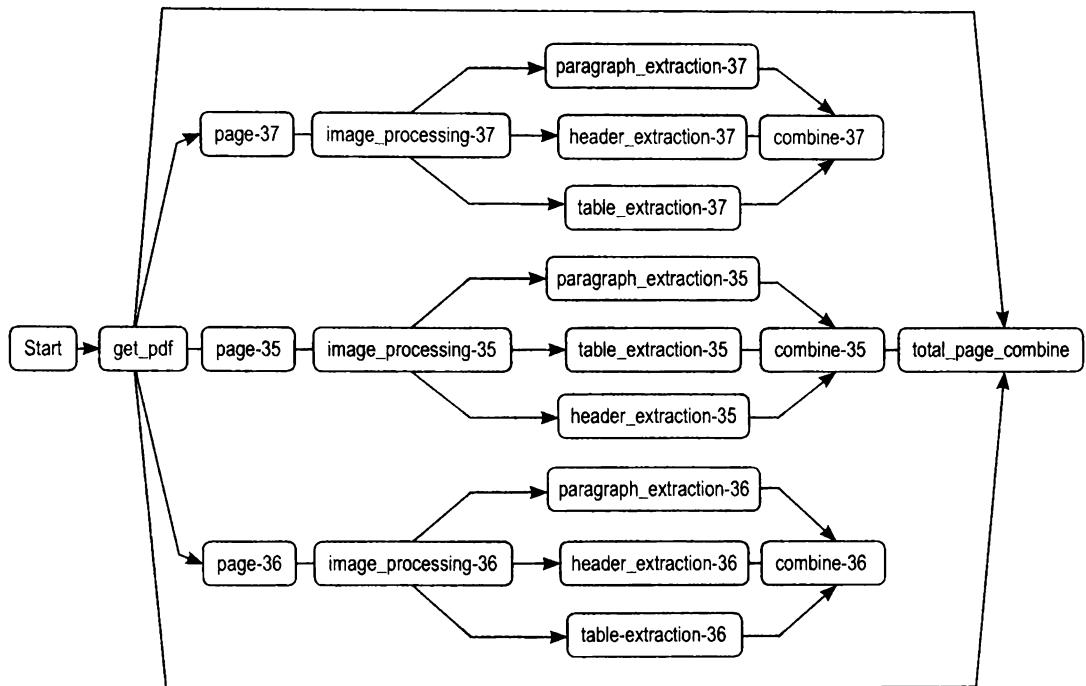
```

Real-World Applications of Graphs

Graphs serve as a prevalent data structure for representing many real-world problems. For example, PageRank, developed by Google co-founders Larry Page and Sergey Brin, is an algorithm that measures how important various web pages are. PageRank represents each web page as a node, and each hyperlink from one page to another represents an edge. The underlying assumption is that important web pages are more likely to have been linked to by other influential pages on the web. In the context of graphs, this means that nodes with a high number of edges from other high-quality nodes are likely to have a better PageRank score.

Another real-world use of graphs is for computational graphs used within scheduling tools like Airflow or large-scale data processing frameworks like Spark or Tensorflow. These frameworks represent the series of computations and data flows that need to be performed as a directed acyclic graph (DAG), which is a directed graph (meaning edges are unidirectional) that has no cycles. Nodes represent operations that create or manipulate data, and edges represent how data (typically multidimensional arrays also known as tensors) must flow from one operation (node) to another.

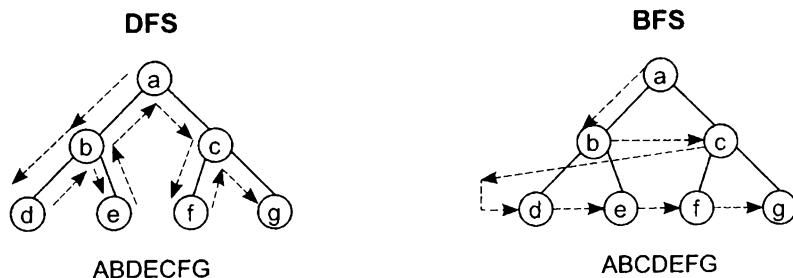
These computational graphs enable parallelism, since it's easy to order the operations in such a way that certain operations that do not depend on one another can be run concurrently. Another advantage is that the computational graph offers a language-agnostic representation of the computations we want, which can easily be ported between underlying frameworks.



An example Airflow execution graph to extract a data table from a large PDF file using image processing

Breadth-First and Depth-First Search

Interview problems related to graphs often involve traversing the graph, either in a breadth-first-search (BFS) or in a depth-first-search (DFS). In BFS, start with one node and add it to a queue. For each node in the queue, process the node and add its neighbors to the queue. In DFS, you also start with one node, but instead of processing all the neighbors next, you recursively process each neighbor one by one.



Below is an example of BFS using a queue for iterative processing:

```

def bfs(graph, v):
    n = len(graph.vertices)
    visited = [False] * (n+1)
    queue = []
  
```

```

queue.append(v)
visited[v] = True

while queue:
    curr = queue.pop(0)
    for i in graph.get_vertex(curr):
        if visited[i.val] == False:
            queue.append(i.val)
            visited[i.val] = True
return visited

```

Below is a primitive example of DFS, where *visited* tracks the set of processed nodes, and *v* is the node currently being processed:

```

def dfs_helper(graph, v, visited):
    visited.add(v)
    for neighbor in graph.get_vertex(v).get_neighbors():
        if neighbor.val not in visited:
            dfs_helper(graph, neighbor.val, visited)
    return visited

def dfs(graph, v):
    visited = set()
    return dfs_helper(graph, v, visited)

```

The runtime for both is $O(E + V)$, where E is the number of edges in the graph and V is the number of vertices. For most basic use cases, DFS and BFS are interchangeable. However, there are some differences in their use cases and advantages.

When interview problems concern optimization, such as finding the shortest path between nodes, consider using BFS. It can find solutions definitively and never gets trapped in recursive sub-calls. Some drawbacks to BFS are that it uses more memory when storing nodes and can take a lot of time if solutions are far away from the root.

When interview problems concern analyzing a graph's structure, such as when finding cycles or orderings in directed graphs (for example, topological sorting), consider using DFS. Because it exhausts paths before trying others, it may be trapped recursively and cannot guarantee a solution. However, DFS has fewer memory requirements and can find long-distance elements in a shorter amount of time compared to BFS.

Algorithms

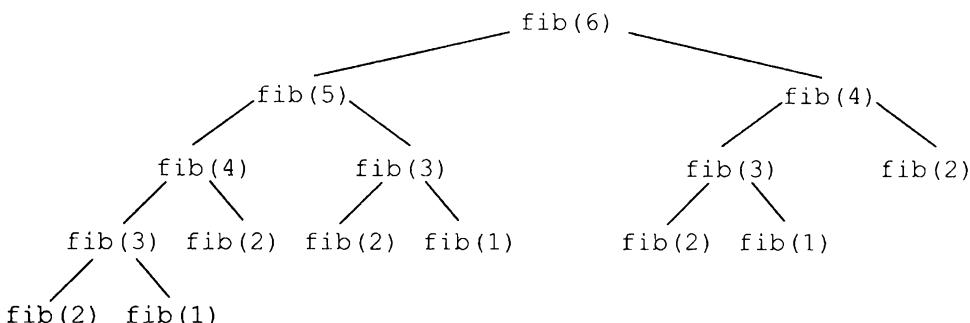
Recursion

A recursive function is one that calls itself directly or indirectly. It can be broken down into two parts: the recursive case (the part that calls itself) and the base case (which terminates the recursion).

For an example of a recursive algorithm, consider the classic problem of producing Fibonacci numbers (0, 1, 1, 2, 3, 5, 8, etc.):

```
def fib(n):
    if n == 0:
        return 0
    if n == 1 or n == 2:
        return 1
    else:
        return fib(n-1)+fib(n-2)
```

If you draw the tree of recursive calls, you'll see that many times $\text{fib}(n)$ for a particular n is called repeatedly. This repetition leads to an undesirable exponential runtime and memory usage. Since recursive calls are stored on a stack, and often, there is limited allocated memory for a program, recursive solutions can also lead to stack overflows which crash the program.



In contrast, iterative algorithms do not overwhelm the call stack. Such algorithms rely on using a for-loop or while-loop, rather than calling themselves repeatedly like in recursion. An iterative example of the Fibonacci sequence is as follows: we use a for-loop to iterate up to n , set the current result to be the sum of the previous Fibonacci number at indexes $n-2$ and $n-1$, and continue this until we reach the desired n :

```
def fib(n):
    if n == 0:
        return 0
    if n == 1 or n == 2:
        return 1
    else:
        prev2 = 0 # fib(n-2)
        prev = 1 # fib(n-1)
        res = 0 # fib(n)
        for i in range(1, n): # process iteratively
            res = prev2 + prev
            prev2 = prev
            prev = res
    return res
```

Compared to the recursive solution, the iterative approach is less expressive code-wise but more memory efficient since there aren't $O(N)$ recursive calls, but instead just two variables to keep track of (the most recent two Fibonacci numbers).

Greedy Algorithms

Greedy algorithms choose at each step what seems to be the best option. In other words, greedy problems reduce problems into smaller ones by making the locally optimal choice at each step.

A classic example where a greedy algorithm works well is when making change with the minimum number of coins. Say, for instance, you wanted to make change for 67 cents, and all you had were pennies, nickels, dimes, and quarters. The greedy approach is to use as many coins of the highest denomination as possible (two quarters), before continuing to the next denomination (one dime). The code is as follows:

```
def minCoins(k):
    coins = [1, 5, 10, 25]
    n = len(coins)
    res = []
    i = n-1
    while(i >= 0 and k >= 0):
        if k >= coins[i]:
            k -= coins[i]
            res.append(coins[i])
        else:
            i -= 1
    return res
```

In the real world, greedy algorithms show up in various areas of machine learning. For example, decision trees are split in a greedy manner to maximize information gain (reduction in entropy based on the feature chosen) at each split. For every feature, we evaluate all possible features, then, in a greedy fashion, choose the feature with the best information gain to split on next. This process takes place recursively until we end up with leaf nodes. The pseudocode for getting the best feature is as follows:

```
info_gains = [getInfoGain(feature) for feature in features]
best_feature_index = np.argmax(info_gains)
best_feature = features[best_feature_index]
```

For interview questions, keep greedy algorithms in mind for optimization problems, where there's an obvious set of choices to select from, and it's easy to know what the appropriate choice is. Keep in mind that it's often easier to reason about a greedy algorithm recursively, but then implement it later iteratively for better memory performance.

Dynamic Programming

Earlier in the recursion section, we saw how an iterative solution to generating the n -th Fibonacci number had its advantage over a recursive version since it used less memory and didn't recompute

work. That's where dynamic programming, the art of storing results of subproblems to speed up recursion, comes in. Although, technically, the iterative solution serves as a form of dynamic programming, to be more explicit, for pedagogical purposes, below we show how an array can be used to cache existing results. Now, instead of making the previous recursive calls all the way down, we get the stored sub-result from the cache to prevent values from being recomputed.

```
dp = [0] * 1000

def fib(n):
    if n == 0 or n == 1:
        return n
    else:
        dp[n] = fib(n-1) + fib(n-2)
        return dp[n]
```

Two properties are needed for dynamic programming (DP) to be applicable:

- 1) optimal substructure
- 2) overlapping subproblems

An optimal substructure means the problem can be solved optimally by breaking it down into subproblems and solving those subproblems. Overlapping subproblems indicates the problem can be broken down into subproblems, that are then reusable in other subproblems. If both are present, then calculating and storing the solutions to subproblems in a recursive manner will solve the overall problem. Note that the Fibonacci satisfies the two requirements because:

- 1) Fib(n) is found by solving the subproblems fib($n-1$) and fib($n-2$)
- 2) Subproblems are overlapping: both fib(n) and fib($n-1$) require having solved fib($n-2$)

Dynamic programming can be implemented with a top-down or bottom-up approach. In a top-down approach, we start with the top and break the problem into smaller chunks (as in the Fibonacci example). In practice, the top-down approaches often are implemented with recursion, and the intermediate results are cached in a hash table.

In contrast, in a bottom-up manner, we start with the smaller problems and continue to the top. These solutions are often implemented iteratively, where an n -dimensional array is used to cache previous results. A bottom-up Fibonacci example would be:

```
def fib(n):
    dp = [0 for _ in range(n+1)]
    dp[1] = 1
    for i in range(2, n+1):
        dp[i] = dp[i-1] + dp[i-2]
    return dp[n]
```

Dynamic programming isn't just an academic exercise or coding interview favorite — it often shows up in the real world, too. For instance, in reinforcement learning, the goal is to understand what

actions to take given a particular state of the universe to maximize an expected eventual payoff. The famous Bellman equations are a core part of the reinforcement learning process and utilize the dynamic programming approach. The Bellman equations break down the total eventual payoff into a series of smaller subproblems that can each be optimized, where the best eventual payoff comes from combining different subproblem payoffs.

Greedy Algorithms Vs. Dynamic Programming

As explained earlier, a greedy algorithm does whatever is locally optimal and hence always chooses the option that leads to the best result for the next step. This contrasts with dynamic programming, which will exhaust the search space and is guaranteed to find the globally optimal solution, not just the locally optimal solution. Therefore, you can think of greedy algorithms as getting the “local maximum/minimum,” but not necessarily the “global maximum/minimum” that dynamic programming achieves.

For a concrete example of where greedy algorithms can fall short, consider the classic 0/1 knapsack problem with the values below. In this problem, we are choosing between N items, each with some positive value and positive weight. We want to maximize the amount of value obtained by selecting items with a total weight of no more than W .

Example Knapsack

Aa Item	Value	Weight	Value/Weight
A	3	4	3/4
B	1.2	2	1.2/2
C	2	3	2/3

The greedy implementation is to just sort the elements that maximize value per unit weight and choose the most efficient items.

```
def knapsackGreedy(values, weights, max_weight):
    curr_weight = 0
    value_list, weight_list = ([], []) # values and weights
    total_sorted = sorted(zip(values, weights),
                          key=lambda x: x[0]/x[1], reverse=True)
    values_sorted, weights_sorted = zip(*total_sorted)
    # iterate in sorted fashion
    for value, weight in zip(values_sorted, weights_sorted):
        if weight + curr_weight <= max_weight: # meets weight requirement
            value_list.append(value)
            weight_list.append(value)
            curr_weight += weight
    return sum(value_list)
```

However, this does not maximize the amount of value obtained overall. Take, for example, the table of weights and values provided earlier, along with a total weight limit of 5.

Greedily, we would select item A (the highest value/weight ratio) and end up with a total value of 3, since the weight constraint does not allow for extra items. However, the best possible method would be to take B and C for a total value of 3.2, which is larger than 3.

In contrast, a dynamic programming approach achieves the global maximum. An example of a bottom-up approach is below, where the core logic is in deciding whether to take an item or not. If an item meets the weight requirement, then we either

- Take the item, and get a new optimal maximum value, which now gives us a new weight constraint (current weight minus item's weight), or
- Do not take the item, and continue with the current optimal maximum value and the current weight constraint.

```
def knapsackDP(values, weights, max_weight):
    n = len(values)
    dp = [[0 for x in range(max_weight+1)] for x in range(n+1)] # 2d-array

    for i in range(n+1):
        for w in range(max_weight+1):
            if weights[i-1] <= w: # meets weight requirement
                dp[i][w] = max(values[i-1]+dp[i-1][w-weights[i-1]],
                                dp[i-1][w]) # either take value or not
            else:
                dp[i][w] = dp[i-1][w] # not taking value
    return dp[n][max_weight]
```

The dynamic programming approach coded above leads to the optimal solution of taking items B and C for a total value of 3.2. When solving coding interview problems, it's crucial to not fall for a more apparent greedy approach when the correct answer is found through dynamic programming.

Sorting

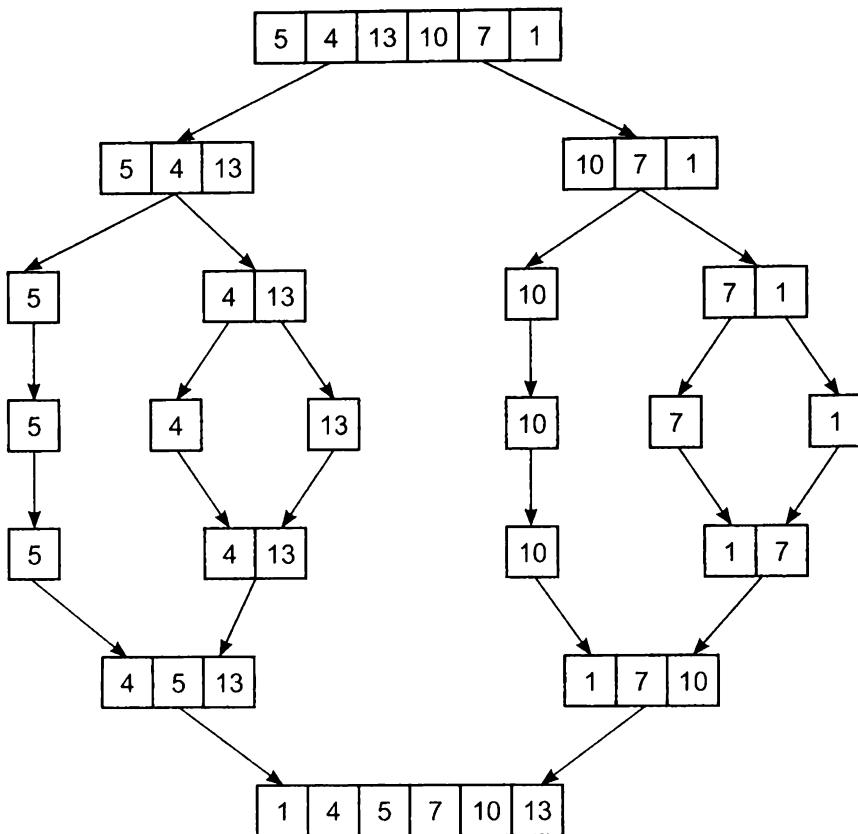
Two unique sorting algorithms — mergesort and quicksort — arise in coding interviews from time to time. While it's rare to be asked to code up these algorithms, both sorts are often modified to solve a problem or used as an intermediate step within a larger coding interview solution. This is because sorting the input can expose some structure, which makes the problem simpler.

Mergesort

Mergesort uses a “divide-and-conquer” approach as follows:

1. Repeatedly divide the input into smaller subarrays, until a base case of a single element is reached (this single element subarray is considered sorted).
2. Repeatedly merge the smaller sorted subarrays into bigger sorted subarrays, until the entire input is merged back together.

Below is an example of mergesort:



Overall, mergesort has a runtime complexity of $O(N \log N)$ and also requires $O(N \log N)$ space to support the auxiliary merging steps. An example implementation of mergesort that utilizes a helper function for merging subarrays is as follows:

```

def merge_helper(a, low, high, mid):
    if len(a) == 1:
        return a
    i = j = k = 0 # k is for merged array
    left = a[:mid] # copy of left side
    right = a[mid:] # copy of right side
    while i < len(left) and j < len(right): # compare left and right sides, add
        if left[i] < right[j]:
            a[k] = left[i]
            i += 1
        else:
            a[k] = right[j]
            j += 1
        k += 1
    while i < len(left): # remaining on left
        a[k] = left[i]
        i += 1
        k += 1
    while j < len(right): # remaining on right
        a[k] = right[j]
        j += 1
        k += 1
  
```

```

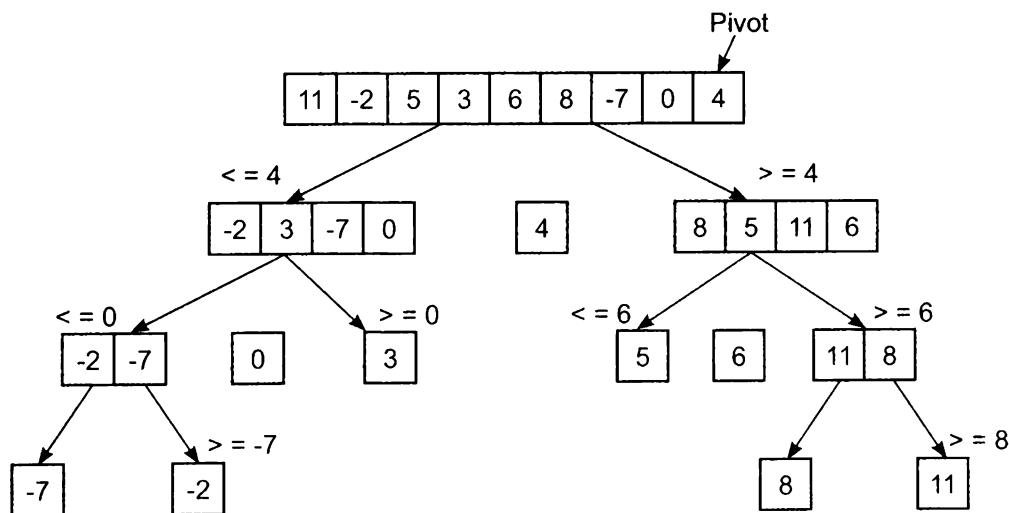
i += 1
k += 1
while j < len(right): # remaining on right
    a[k] = right[j]
    j += 1
    k += 1
return a

def mergesort(a, low, high):
    if low >= high:
        return a
    mid = (low + high-1) // 2
    mergesort(a, low, mid)
    mergesort(a, mid+1, high)
    merge_helper(a, low, high, len(a) // 2)
    return a

```

Quicksort

Quicksort selects an arbitrary pivot element and puts all elements smaller than the pivot to the left of the pivot, and larger elements to the right of the pivot. The exchanging of elements occurs through swaps. This process of selecting a pivot and swapping the left and right elements ensues recursively, until the base case is hit when just two elements are swapped into their correct relative order.



The worst-case runtime for quicksort is $O(N^2)$, although, in practice the expected runtime is closer to $O(N \log N)$ through picking a “smart” pivot whereby the elements are roughly divided into equal halves upon each iteration.

Below is an example implementation of quicksort that utilizes a helper function to partition the array:

```
def helper(a, low, high):
    i = low-1 # smaller element index
    pivot = a[high] # pivot
    for j in range(low, high):
        if a[j] <= pivot:
            i += 1 # increment
            a[i], a[j] = a[j], a[i] # swap
    a[i+1], a[high] = a[high], a[i+1]
    return i+1

def quicksort(a, low, high):
    if len(a) == 1:
        return a
    if low < high:
        pivot = helper(a, low, high) # place pivot
        quicksort(a, low, pivot-1) # recurse left side
        quicksort(a, pivot+1, high) # recurse right side
    return a
```

Matrix Multiplication

Since all machine learning algorithms eventually reduce to a series of matrix multiplications, speeding up these operations is crucial for large-scale machine learning applications. Using the direct mathematical definition of matrix multiplication, an implementation of matrix multiplication gives an $O(N^3)$ runtime, as seen by the three nested for-loops below:

```
def matrix_multiply(A, B):
    m = [[0 for row in range(len(B[0]))] for col in range(len(A))]
    for i in range(len(A)):
        for j in range(len(B[0])):
            for k in range(len(B)):
                m[i][j] = m[i][j] + A[i][k] * B[k][j] # add
    return m
```

Nevertheless, improving upon this runtime is possible. For example, we can break down the multiplication into a series of sub-matrix multiplications. These sub-matrix multiplications can be calculated in parallel, which enables the task to be accomplished in a distributed manner.

Finance companies like to ask candidates about coding up matrix multiplication or implementing other common linear algebra topics such as singular-value decomposition because it allows firms to test your mathematical understanding and programming ability at once. These questions are also picked because optimizing numerical calculations done in linear algebra is a relevant concept when trying to reduce the latency of trading systems.

Interview Questions

Easy Problems

- 9.1. Amazon: Given two arrays, write a function to get the intersection of the two. For example, if $A = [1, 2, 3, 4, 5]$, and $B = [0, 1, 3, 7]$ then you should return $[1, 3]$.
- 9.2. D.E. Shaw: Given an integer array, return the maximum product of any three numbers in the array. For example, for $A = [1, 3, 4, 5]$, you should return 60, while for $B = [-2, -4, 5, 3]$ you should return 40.
- 9.3. Facebook: Given a list of coordinates, write a function to find the k closest points (measured by Euclidean distance) to the origin. For example, if $k = 3$, and the points are: $[[2, -1], [3, 2], [4, 1], [-1, -1], [-2, 2]]$, then return $[[{-1, -1}], [2, -1], [-2, 2]]$.
- 9.4. Google: Say you have an n -by- n matrix of elements that are sorted in ascending order both in the columns and rows of the matrix. Return the k -th smallest element of the matrix. For example, consider the matrix below:

$$\begin{bmatrix} 1 & 4 & 7 \\ 3 & 5 & 9 \\ 6 & 8 & 11 \end{bmatrix}$$

If $k = 4$, then return 5.

- 9.5. Akuna Capital: Given an integer array, find the sum of the largest contiguous subarray within the array. For example, if the input is $[-1, -3, 5, -4, 3, -6, 9, 2]$, then return 11 (because of $[9, 2]$). Note that if all the elements are negative, you should return 0.
- 9.6. Facebook: Given a binary tree, write a function to determine whether the tree is a mirror image of itself. Two trees are a mirror image of each other if their root values are the same and the left subtree is a mirror image of the right subtree.

Medium Problems

- 9.7. Google: Given an array of positive integers, a peak element is greater than its neighbors. Write a function to find the index of any peak elements. For example, for $[3, 5, 2, 4, 1]$, you should return either 1 or 3 because the values at those indexes, 5 and 4, are both peak elements.
- 9.8. AQR: Given two lists X and Y, return their correlation.
- 9.9. Amazon: Given a binary tree, write a function to determine the diameter of the tree, which is the longest path between any two nodes.
- 9.10. D.E. Shaw: Given a target number, generate a random sample of n integers that sum to that target that also are within σ standard deviations of the mean.
- 9.11. Facebook: You have the entire social graph of Facebook users, with nodes representing users and edges representing friendships between users. Given a social graph and two users as input, write a function to return the smallest number of friendships between the two users. For example, take the graph that consists of 5 users A, B, C, D, E, and the friendship edges are: (A, B), (A, C), (B, D), (D, E). If the two input users are A and E, then the function should return 3 since A is friends with B, B is friends with D, and D is friends with E.
- 9.12. LinkedIn: Given two strings A and B, write a function to return a list of all the start indices within A where the substring of A is an anagram of B. For example, if A = "abcdcbac" and

$B = \text{"abc,"}$ then you want to return $[0, 4, 5]$ since those are the starting indices of substrings of A that are anagrams of B .

- 9.13. Yelp: You are given an array of intervals, where each interval is represented by a start time and an end time, such as $[1, 3]$. Determine the smallest number of intervals to remove from the list, such that the rest of the intervals do not overlap. Intervals can “touch,” such as $[1, 3]$ and $[3, 5]$, but are not allowed to overlap, such as $[1, 3]$ and $[2, 5]$). For example, if the input interval list given is: $\{[1, 3], [3, 5], [2, 4], [6, 8]\}$, then return 1, since the interval $[2, 4]$ should be removed.
- 9.14. Goldman Sachs: Given an array of strings, return a list of lists where each list contains the strings that are anagrams of one another. For example, if the input is $\{\text{"abc"}, \text{"abd"}, \text{"cab"}, \text{"bad"}, \text{"bca"}, \text{"acd"}\}$ then return: $\{[\text{"abc"}, \text{"cab"}, \text{"bca"}], [\text{"abd"}, \text{"bad"}], [\text{"acd"}]\}$.
- 9.15. Two Sigma: Say that there are n people. If person A is friends with person B, and person B is friends with person C, then person A is considered an indirect friend of person C.
Define a friend group to be any group that is either directly or indirectly friends. Given an n -by- n adjacency matrix N , where $N[i][j]$ is one if person i and person j are friends, and zero otherwise, write a function to determine how many friend groups exist.
- 9.16. Workday: Given a linked list, return the head of the same linked list but with k -th node from the end of a linked list removed. For example, given the linked list $3 \rightarrow 2 \rightarrow 5 \rightarrow 1 \rightarrow 4$ and $k = 3$, remove the 5 node and, thus, return the linked list $3 \rightarrow 2 \rightarrow 1 \rightarrow 4$.
- 9.17. Goldman Sachs: Estimate π using a Monte Carlo method. Hint: think about throwing darts on a square and seeing where they land within a circle.
- 9.18. Palantir: Given a string with lowercase letters and left and right parentheses, remove the minimum number of parentheses so the string is valid (every left parenthesis is correctly matched by a corresponding right parenthesis). For example, if the string is $\text{"})a(b((cd)e(f)g)"}$ then return "ab((cd)e(f)g)" .
- 9.19. Citadel: Given a list of one or more distinct integers, write a function to generate all permutations of those integers. For example, given the input $[2, 3, 4]$, return the following 6 permutations: $[2, 3, 4], [2, 4, 3], [3, 2, 4], [3, 4, 2], [4, 2, 3], [4, 3, 2]$.
- 9.20. Two Sigma: Given a list of several categories (for example, the strings A, B, C, and D), sample from the list of categories according to a particular relative weighting scheme. For example, say we give A a relative weight of 5, B a weight of 10, C a weight of 15, and D a weight of 20. How do we construct this sampling? How do you extend the solution to an arbitrarily large number of k categories?
- 9.21. Amazon: Given two arrays with integers, return the maximum length of a common subarray within both arrays. For example, if the two arrays are $[1, 3, 5, 6, 7]$ and $[2, 4, 3, 5, 6]$ then return 3, since the length of the maximum common subarray, $[3, 5, 6]$, is 3.
- 9.22. Uber: Given a list of positive integers, return the maximum increasing subsequence sum. In other words, return the sum of the largest increasing subsequence within the input array. For example, if the input is $[3, 2, 5, 7, 6]$, return 15 because it's the sum of 3, 5, 7. If the input is $[5, 4, 3, 2, 1]$, return 5 (since no subsequence is increasing).
- 9.23. Palantir: Given a positive integer n , find the smallest number of perfect squares that sum up to n . For example, for $n = 7$, you should return 4, since $7 = 4 + 1 + 1 + 1$. For $n = 13$, you should return 2, since $13 = 9 + 4$.

- 9.24. Facebook: Given an integer n and an integer k , output a list of all of the combinations of k numbers chosen from 1 to n . For example, if $n = 3$ and $k = 2$, return [1, 2], [1, 3], [2, 3].

Hard Problems

- 9.25. Citadel: Given a string with left and right parentheses, write a function to determine the length of the longest well-formed substring. For example, if the input string is “)())(,” then return 4, since the longest well-formed substring is “()”).
- 9.26. Bloomberg: Given an m -by- n matrix with positive integers, determine the length of the longest path of increasing integers within the matrix. For example, consider the input matrix:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

In this case, return 5, since one of the longest paths would be 1-2-5-6-9.

- 9.27. Google: Given a number n , return the number of lists of consecutive positive integers that sum up to n . For example, for $n = 9$, return 3, since the consecutive positive integer lists are: [2, 3, 4], [4, 5], and [9]. Can you solve this in linear time?
- 9.28. Citadel: Given a continuous stream of integers, write a class with functions to add new integers to the stream, and a function to calculate the median at any time.
- 9.29. Two Sigma: Given an input string and a regex, write a function that checks whether the regex matches the input string. The input string is composed of the lowercase letters a-z. The regular expression contains lowercase a-z, ‘?’ , or ‘*’ , where the ‘?’ matches any one character, and the ‘*’ matches an arbitrary number of characters (empty as well). For example, if the input string is “abcdba” and the regex is “a*c?*”, return true. However, if the regex was instead “b*c?*” return false.
- 9.30. Citadel: A fire department wants to build a new fire station in a location where the total distance from the station to all houses in the town (in Euclidean terms) is minimized. Given a list of coordinates for the n houses, return the coordinates of the optimal location for the new fire station.

Solutions

Solution #9.1

The simplest way to check for intersecting elements of two lists is to use a doubly-nested for-loop to iterate over one array and check against every element in the other array. However, this leads to a time complexity of $O(N*M)$ where N is the length of A , and M is the length of B .

A better approach is to use sets (which utilizes a hash map implementation underneath) since the runtime time is $O(1)$ for each lookup operation. Then we can do the series of lookups over the larger set (resulting in a $O(\min(N, M))$ total runtime):

```
def intersection(a, b):
    set_a = set(a)
    set_b = set(b)
```

```

if len(set_a) < len(set_b):
    return [x for x in set_a if x in set_b]
else:
    return [x for x in set_b if x in set_a]

```

The time complexity is $O(N + M)$ due to the creation of the sets, and the space complexity is also $O(N + M)$ since the arrays might contain all distinct elements.

Solution #9.2

If all of the numbers were positive, then the maximum product of three numbers is a matter of finding the largest three numbers in the array and multiplying them. Be sure to clarify with the interviewer what's in the integer array — don't just surmise they are all positive integers. However, with negative integers, the largest product could arise if we take the two smallest numbers (both could be negative) and multiply that result by the largest positive number. We'd need to compare this potential product to the number involving just the largest three numbers.

By first sorting the array, you can get the largest three numbers and the smallest two numbers. Alternatively, we can use a heap (finding the largest three numbers using a max-heap, and the smallest two numbers using a min-heap), rather than sorting. An implementation involving heaps is below:

```

import heapq

def max_three(arr):
    a = heapq.nlargest(3, arr) # largest 3 numbers
    b = heapq.nsmallest(2, arr) # smallest 2 (for negatives case)
    return max(a[2]*a[1]*a[0], b[1]*b[0]*a[0]) # compare

```

The time complexity is where $O(N)$ is the size of the input array and the space complexity is $O(1)$ for the heap, since the number of elements is fixed.

Solution #9.3

The brute-force solution to finding the k -closest coordinates to the origin would be to iterate over all coordinates, compute the distance from each coordinate to the origin, and then sort by distance to find the k -closest ones. This would yield a runtime of $O(N \log N)$ because of sorting N coordinates. A more optimal way is to instead utilize a min-heap whereby we add coordinates based on the Euclidean distance. Then, at the end, we can just take the first k elements, which will already be the smallest due to the min-heap:

```

from heapq import heappush, heappop

def closest(points, k):
    def get_dist(x, y): # distance helper function
        return x**2 + y**2
    min_heap = [] # heap

```

```

n = len(points)
for i in range(n):
    x = points[i][0]
    y = points[i][1]
    heappush(min_heap, (get_dist(x, y), points[i])) # add to heap
res = []
for i in range(k):
    res.append(heappop(min_heap)[1]) # get top k
return res

```

The time complexity is $O(N \log K)$, since we have $O(N)$ additions to the heap, each of which takes $O(\log K)$ time. The space complexity is $O(K)$ to accommodate the min-heap.

Solution #9.4

The simple approach to finding the k -th smallest element is to iterate over all elements in the matrix, add each element to a min-heap, and then pop from the heap k times. However, because the matrix is sorted, we know that the k -th smallest element must be within row k and column k . Therefore, we can use a double for-loop with k as the max value for each of the two indices. Once all elements are added, then we can pop the top- k smallest elements from the stack and return the last element.

```

from heapq import heappush, heappop

def smallest(m, k):
    n = len(m)
    heap = [] # heap
    res = -1
    for i in range(min(k, n)):
        for j in range(min(k, n)):
            heappush(heap, m[i][j]) # add to heap
    for _ in range(k):
        res = heappop(heap) # pop k times
    return res

```

We will presume, that, in general $K \ll N$ (although the code handles any event in which this is not the case). Then the time complexity is $O(K^2 \log K)$ because each heap push operation takes $O(\log K)$ time, and this operation is within a doubly-nested for-loop. The space complexity is $O(K)$ to support the heap.

Solution #9.5

A brute-force way solution to finding the sum of the largest contiguous subarray is to compute the sum over all possible contiguous subarrays, and then return the biggest sum found. This would have a runtime complexity of $O(N^2)$ due to the doubly-nested for-loops. However, there is no need to do multiple passes: in a single iteration of the array, if we keep track of the current sum and it ever goes below 0, there is no need to include elements in that previous subarray, and we can set the current sum to 0. Note that if the array is all negatives, then we get a final result of 0 (by taking no elements).

While doing the single iteration through the array, we keep track of the maximum seen at every point and return that at the end:

```
def max_subarray(arr):
    n = len(arr)
    max_sum = arr[0] # max
    curr_sum = 0 # current sum
    for i in range(n):
        curr_sum += arr[i]
        max_sum = max(max_sum, curr_sum) # get max
        if curr_sum < 0: # reset
            curr_sum = 0
    return max_sum
```

The time complexity is $O(N)$, where N is the size of the input array, and the space complexity is $O(1)$.

Solution #9.6

Finding out if a tree is a mirror image of itself can be solved recursively in a straightforward manner. We will use the `TreeNode` class implementation below. The helper function checks that for two subtrees:

- a) The root values match
- b) The first subtree's left side is a mirror of the second subtree's right side
- c) The first subtree's right side is a mirror of the second subtree's left side

This helper function is then called on the root's left and right subtrees:

```
class TreeNode:
    def __init__(self, val):
        self.val = val
        self.left = None
        self.right = None

    def is_mirror(self):
        if self is None:
            return True
        return helper(self.left, self.right)

    def helper(x, y):
        if (x is None and y is None):
            return True
        elif (x is None or y is None):
            return False
        return x.val == y.val and
               helper(x.left, y.right) and
               helper(x.right, y.left)
```

The time complexity and space complexity are both $O(N)$, where N is the size of the tree since the entire tree is traversed once.

Solution #9.7

The brute-force way to check for peak elements is to iterate through the array and check, for each element, whether the left and right neighbor is less than the element's value. That would yield an $O(N)$ runtime for one linear scan. However, we can improve upon this by doing a modified binary search.

Take the middle value and check the number to the left of it (say it is smaller). If the middle value's right element is also smaller, then the middle value is a local peak. Alternatively, if the right value is greater than the middle value, then there must be a local peak on the right-hand side (numbers ascending and then a descent, or it keeps ascending till it reaches the end of the array, which is also a local peak). We do this in an iterative manner until we reach the condition that the element's value is larger than both its left and right neighbor values.

```
def get_peak(arr):
    start = 0
    end = len(arr) - 1
    while True:
        mid = (start + end) // 2 # get middle element and check bounds
        left = arr[mid-1] if mid - 1 >= 0 else float('-inf')
        right = arr[mid+1] if mid + 1 < len(arr) else float('inf')
        if left < arr[mid] and right < arr[mid]: # left < mid > right
            return mid
        elif arr[mid] < right: # change bounds
            start = mid + 1
        else:
            end = mid - 1
```

The time complexity is $O(\log(N))$ since it is a variant of binary search and the space complexity of $O(1)$.

Solution #9.8

We know that correlation is given by: $\rho_{x,y} = \frac{Cov(X,Y)}{\sigma_x * \sigma_y}$

where the numerator is the covariance of X and Y , and the denominator is the product of the standard deviation of X and the standard deviation of Y . Recall the definition of covariance:

$$Cov(X, Y) = E[(X - \mu_X)(Y - \mu_Y)]$$

Therefore, a simple implementation is to have helper functions for calculating both the mean and standard deviation. Note that calculating both the mean and standard deviation is $O(N)$ runtime (since both take a single sum across all N elements). Therefore, the correlation runtime is $O(N)$, since we calculate a few means and standard deviations, as well as iterate over all N elements once through. The space complexity is $O(N)$ to keep track of N elements in the correlation to be processed.

```

import math

def mean(x):
    return sum(x)/len(x)

def sd(x):
    m = mean(x)
    ss = sum((i-m) ** 2 for i in x)
    return math.sqrt(ss / len(x))

def corr(x, y):
    x_m = mean(x)
    y_m = mean(y)
    xy_d = [] # product of de-meaned X and Y, for covariance calc
    for i in range(len(x)):
        x_d = x[i] - x_m
        y_d = y[i] - y_m
        xy_d.append(x_d * y_d) # add product of X_i and Y_i
    return mean(xy_d) / (sd(x) * sd(y)) # from formula above

```

Solution #9.9

Finding the diameter of a tree can be solved recursively, since at any given root node, the diameter is just one more than the maximum of the depths from its left and right subtrees. Depth, in this context, is defined as the number of nodes from the root to the farthest leaf node. We can utilize a helper function called depth() to calculate tree depth, and have its base case to return 0 if the root is None. Inside the diameter function, we want to recursively call depth(root.left) and depth(root.right) and return $1 + \max(\text{left result}, \text{right result})$.

```

class TreeNode:
    def __init__(self, val):
        self.val = val
        self.left = None
        self.right = None

    def calc_diameter(self):
        def depth(root, diameter): # helper
            if root is None:
                return 0, diameter # base case
            left, diameter = depth(root.left, diameter)
            right, diameter = depth(root.right, diameter)
            diameter = max(diameter, left + right) # update diameter
            return max(left, right) + 1, diameter # return root's longest path
        depth, diameter = depth(self, 0) # call helper
        return diameter

```

The time complexity is $O(N)$, where N is the size of the tree, since the entire tree is traversed once. The space complexity is also $O(N)$, since there will be N recursive calls on the stack in the worst case.

Solution #9.10

To generate the numbers at random, we can start by calculating the mean and defining upper and lower bounds on any given integer based on the standard deviation. Let the target be represented by T , the number of elements by N , and the sigma by σ . The mean μ will be T/N , and the allowable standard deviation is given by $\sigma * \mu$. That means any given element must be within $\sigma * \mu$ of μ , which establishes our lower and upper bounds.

Once we have the upper and lower bounds, we can take all n integers, starting each at the lower bound, and increment them at random in an iterative manner until the sum of the numbers is the given target value. For random incrementing, we can randomly sample one of the N elements and increment it by 1 if it is not beyond the upper bound. If it is, we skip it for that iteration. An example implementation is as follows:

```
import random

def generate_nums(target, n, sigma):
    mean = target / n
    sd = int(sigma * mean)
    max_val, min_val = mean + sd, mean - sd # bounds
    results = [min_val] * n
    remaining = target - n * min_val

    while remaining > 0:
        a = random.randint(0, n-1) # choose random index
        if results[a] >= max_val: # skip if above bound
            continue
        results[a] += min(remaining, 1)
        remaining -= 1
    return results
```

After creating the initial results array of n elements that are each min_val , let R be the remaining amount results need to be incremented to hit the target sum.

The while loop has a time complexity of $O(R)$, which is a function of $O(T*N)$ for the following reasons:

$$R = T - n\left(\mu - \sigma * \frac{T}{n}\right)$$

based on the above formulation. Simplifying yields: $R = T - n(\mu) + \sigma * T$

Recall that by definition $T \approx n * \mu$ therefore, we can simplify the above to be:

$$R = T - n\left(\frac{T}{n}\right) + \sigma * T = \sigma * T$$

Thus, the runtime is a function of $O(\max(\sigma * T), N)$ since the bottleneck is either the while loop, or in generating the results of size N initially. The space complexity will be $O(N)$ to store the results.

Solution #9.11

To find the shortest distance in a graph between two nodes (in this case, the number of friends between two individuals), we can use a breadth-first-search (BFS) and maintain a list of distances from one of the input users. Each distance in this list starts at 0. In typical BFS fashion, we utilize a queue and add the initial node to the queue, processing the queue in an iterative fashion as long as it is not empty. At every step, for any friend node we see, we update the distance value for that friend node. Since the distance array will track all of the distances from the starting user x , we return the distance of the other user y :

```
import queue

def friendship_distance(n, edges, x, y):
    distance = [0] * n # distances from x
    processed = [False] * n # visited or not already
    q = queue.Queue()
    q.put(x)
    processed[x] = True
    while (not q.empty()):
        curr = q.get() # current node
        if curr not in edges:
            continue
        for neighbor in range(len(edges[curr])):
            if not processed[edges[curr][neighbor]]: # process if not visited
                distance[edges[curr][neighbor]] = distance[curr] + 1
                q.put(edges[curr][neighbor]) # add neighbor
                processed[edges[curr][neighbor]] = True
    return distance[y]
```

The time complexity is $O(N \cdot M)$, where N is the number of users, and M is the number of friendships. The space complexity is $O(N)$ to maintain the queue.

Solution #9.12

The brute-force way to find all the anagrams of B within A is to find all substrings of A and then check if each is an anagram of B . This is inefficient because it leads to an $O(N^2 \cdot K)$ runtime since there are N^2 substrings of A , and the anagram check takes $O(K)$ time.

A better way is to have a rolling window approach: say the length of B is k . We can iterate over A and check a window of size k and compare that window to B to check for an anagram match.

A basic way to confirm if two strings are anagrams is to check if the sorted versions of the two strings are equal. However, that is $O(N \log N)$ due to sorting. To speed this up to an $O(N)$ check, we can utilize two dictionaries. For each window in A , keep a dictionary of character counts, and compare it to B 's dictionary (for an $O(N)$ comparison rather than an $O(N \log N)$ via sorting). After setting up each initial dictionary for A and B (via utilizing helper function below for adding), we can then run through all valid windows and check at each step whether the two dictionaries have valid anagrams (and, if so, add to result). We can check for anagrams, and then keep sliding the window within A .

```

def total_anagrams(A, B):
    n, k = len(A), len(B)
    if n < k:
        return []
    def is_anagram(d_a, d_b):
        for x in d_b:
            if x not in d_a: # key doesn't exist
                return False
            if d_b[x] != d_a[x]: # count doesn't match
                return False
        return True
    def add(char, d): # tracking char freq
        if char in d:
            d[char] += 1
        else:
            d[char] = 1
        return d
    d_a, d_b = {}, {}
    start, res = 0, []
    for i in range(k):
        d_a = add(A[i], d_a) # set up dictionary for A
        d_b = add(B[i], d_b) # set up dictionary for B
    for i in range(k, n+1):
        if is_anagram(d_a, d_b):
            res.append(start)
        if i < n:
            d_a = add(A[i], d_a) # add next char of window to dict
            d_a[A[start]] -= 1 # remove leftmost char of window from dict
            start += 1
    return res

```

The time complexity is $O(NK)$, since the anagram comparison is $O(K)$ and there are $O(N)$ windows (assuming $N \gg K$). The space complexity is $O(N)$ to support the resulting list and dictionaries.

Solution #9.13

How do we know when two intervals overlap? First, sort all intervals by start time, and then by end time. Two intervals overlap when the ending time of an interval is larger than the start time of the next interval. Therefore, we want to iterate over the intervals and increment a count whenever two intervals overlap.

To keep track of the current and the previous interval, we create two pointers called low and high. Let low point to the interval with the smaller start time. At each iteration, we can detect when the

intervals overlap by checking the low pointer of the next interval versus the higher pointer of the current one.

In case intervals need to be merged (when one interval is completely within another), we set the previous low pointer value to be the current high pointer value. In this way, no intervals are deleted, but in the next iteration we need to compare against the current high interval's begin and end time:

```
def interval_removal(interval_list):
    if len(interval_list) == 0:
        return 0
    intervals = sorted(interval_list, key=lambda k: (k[0], k[1])) # sort
    res, low, count = 0, 0, 0
    for high in range(1, len(intervals)):
        if intervals[low][1] > intervals[high][0]: # overlap
            count += 1
        if not intervals[high][0] < intervals[low][1] < intervals[high][1]:
            low = high # merge
    return count
```

The time complexity is $O(N)$ since there is just one for-loop. The space complexity is $O(1)$ since there is no extra space being used.

Solution #9.14

We can use a helper function to identify anagrams, and then identify the strings that have the same composition. For the helper function that identifies anagrams, we can create a character-frequency map. As an example, for “abc” we can have the following dictionary: {“a”: 1, “b”: 1, “c”: 1}. To compare anagram compositions among different strings, we will need to uniquely identify the maps: for example, if for “abc” we have the map: {“a”: 1, “b”: 1, “c”: 1} and for “cab” we have: {“c”: 1, “a”: 1, “b”: 1} then we want to make sure both correspond to the same anagram grouping. To do this, we can sort each dictionary in \langle key, value \rangle format and use the string representation. Therefore, we simply keep a final dictionary whereby that anagram string is the key, and the value is the list of strings that fall under that anagram grouping:

```
def anagram_group(str_list):
    cfd_str_list = {} # {anagram_composition_string : {strings of common anagram}}
    for s in str_list:
        cfd = {} # char freq dict of s {character : frequency}
        for c in s:
            if c in cfd:
                cfd[c] += 1
            else:
                cfd[c] = 1
        # got sorted char freq dict of s
        cfd = dict(sorted(cfd.items()))
    # flatten cfd to cfd_str as single string
```

```

cfd_str = ''.join('{}{}'.format(k, v)
                  for k, v in cfd.items()) # format

# anagrams will produce identical cfd_str
if cfd_str not in cfd_str_list:
    cfd_str_list[cfd_str] = [s] # add anagram
else:
    cfd_str_list[cfd_str].append(s) # not existing anagram yet
res = []
for cfd_str in cfd_str_list:
    res.append(cfd_str_list[cfd_str])
return res

```

The time complexity is $O(NK(\log K))$, where N is the length of the string list input and K is the maximum length of any given string within the list, since the outer for-loop will take $O(N)$ time for each string, the helper function takes $O(K)$ time, and sorting the dictionary takes $O(K \log K)$. The space complexity is $O(NK)$ since there are at most N results of size K .

Solution #9.15

Counting the friend groups is just a variation of the classic graph theory problem of finding all disconnected subgraphs within a graph. At a high level, we can use depth-first-search (DFS) to find all the nodes in a friend group.

In particular, we can start with any person and run DFS recursively until there are no more direct friends to any of the covered individuals in the friend group. By the end of this process, any indirect friend to the initial person is also included in the group. Note that each person will only be in one friend group (if A were in one, and directly or indirectly related to B, then there is no way B is in a different friend group). Therefore, we do this until all people have been accounted for, and simply increment the number of friend groups found whenever we need to run a new DFS for a friend group. The algorithm implementation is below:

```

def dfs(friends, i, N):
    friends.add(i)
    for j in range(len(N[i])): # checks all neighbors of existing person
        if N[i][j] == 1 and j not in friends:
            dfs(friends, j, N)

def count_groups(N):
    num = len(N[0])
    groups = 0
    friends = set() # tracks friend groups
    for i in range(num):
        if i not in friends: # get all related groups that i is in
            dfs(friends, i, N)
            groups += 1
    return groups

```

The space complexity is $O(N)$ since there are N elements in the set, and the time complexity is $O(N^2)$ since there are $O(N^2)$ edges to traverse in the adjacency matrix.

Solution #9.16

To remove the k -th element from the back, we can create a count helper function to get the ordering of the nodes, and then take the k -th last node and make sure the previous node's next value is the node $k + 1$. Along the way, we need to address some corner cases (for example, if k is the last element in the linked list).

We want to avoid recalculating node counts, so we cache the ordering of the nodes in a dictionary. Thus, our count() helper function below will store the ordering of the nodes, with the order number as the key. Then, when looking for nodes $k + 1$ and $k - 1$ for editing the linked list, we can query the dictionary for those two nodes. Finally, we return the head:

```
class Node:
    def __init__(self, val):
        self.val = val
        self.next = None

def remove(head, k):
    def count(node, d): # helper function
        if node.next is None:
            d[1] = node # tail
            return 1
        res = count(node.next, d) # recursive call
        d[1+res] = node # store res in dictionary
        return 1 + res
    d = {}
    n = count(head, d) # get counts
    if k == len(d): # tail case
        if len(d) == 1:
            return None
        head.next = None
        return d[k-1]

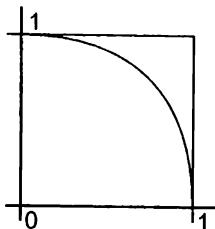
    prev_node = d[k+1] # get prev node
    if k == 1:
        next_node = None
    else:
        next_node = d[k-1] # get next node

    prev_node.next = next_node # set next node
    return head
```

The time complexity is $O(N)$ since the count() function will visit every linked list node once, and the remaining operations are constant time. The space complexity is $O(N)$ to accommodate the dictionary that contains all the input nodes.

Solution #9.17

To estimate π , start by imagining a unit square and choosing a random pair of coordinates (x, y) inside it. Within the unit square, we can also form a quarter of a circle.



For reference, the equation for a circle is given by: $x^2 + y^2 = r^2$

Therefore, we can sample random (x, y) pairs and verify if it falls within the unit circle (if $x^2 + y^2 \leq 1$). Then, we can count the resulting number of points inside the circle; when divided by the total number of points, this yields the proportion of points within the quarter circle:

$$\frac{1}{4}\pi r^2 = \frac{\pi}{4}$$

Therefore, π should be 4 times the resulting proportion of points. The time complexity is $O(N)$, where N is the number of iterations, due to the for-loop. The space complexity is $O(1)$ since only a fixed number of values is looked at per iteration.

```
import random
import math

count = 0 # num points inside quarter circle
n = 10000000 # iterations

for i in range(0, n):
    x_sq = random.random()**2
    y_sq = random.random()**2
    if math.sqrt(x_sq + y_sq) < 1.0: # check if inside circle
        count += 1

pi = (float(count) / n) * 4 # accounts for quarter circle
return pi
```

Solution #9.18

Note that an invalid string has more left or right parentheses than its respective counterpart. To keep track of parentheses, we can use a stack and push to it when encountering a left parenthesis, and pop from it when encountering a right parenthesis. Note that for a valid string, every right parentheses must be matched by the latest (according to the stack) left parentheses. Therefore, when we encounter a right parentheses, we must set the valid left and right parentheses accordingly. If we iterate over a letter, then we use it in the result. A temporary array can be used to track the valid letters and parentheses:

```

def splitParen(s):
    stk = [] # stack
    res = [''] * len(s) # chars to return

    for index, val in enumerate(s): # index and value of each char
        if val == '(':
            stk.append(index) # push index of left paren
        elif val == ')':
            if stk:
                latest = stk.pop() # pop the latest corresponding left paren
                res[latest] = s[latest] # or '('
                res[index] = val # or ')'
            else:
                res[index] = val
    return ''.join(res)

```

The space complexity is $O(N)$ due to the temporary array, and the time complexity is $O(N)$ since there is one scan through all of the characters in the string.

Solution #9.19

We can generate permutations in a recursive manner as follows: start with an empty array of results. For the base case, note that if the length of the input is 1, we just return an array of that number. Then, for every element in the input array, we make a recursive call to the subarray with all elements except the current element. Remember that each recursive call returns a list of permutations. Therefore, while iterating over all elements, in each recursive call take the results and add the element to each list that is returned. At the end we return that array of results:

```

def permute(nums):
    n = len(nums)
    res = [] # store results
    if n <= 1:
        return [nums]
    else:
        for i in range(n): # this is the element E
            # recurse on previous combos
            for combo in permute(nums[:i] + nums[i+1:]):
                res.append([nums[i]] + combo)
    return res

```

The time complexity is $O(N!)$ since there are $N!$ permutations being generated in the recursive call, and the space complexity is $O(N*N!)$ since each of the $N!$ calls uses $O(N)$ space.

Solution #9.20

Let N be the sum of the weights. In the example, N would be $5 + 10 + 15 + 20 = 50$. A basic way to sample from a fixed number of categories would be to generate a list of size N and create list elements proportional to their weights as follows (using the example):

```

import random
w_a, w_b, w_c, w_d = 5, 10, 15, 20
l = ['A'] * w_a + ['B'] * w_b + ['C'] * w_c + ['D'] * w_d
return(random.choice(l))

```

However, this solution is not optimal because the space usage here is $O(N)$ since we store the number of elements according to sum of the weights. If we wanted to do this more generally, keeping space usage in mind, we can do the following:

1. Calculate the cumulative sum of weights.
2. Choose a random number k between 0 and the sum of weights.
3. Assign k the corresponding category where the cumulative sum is above k .

In the example below, we use the weights [5, 10, 15, 20] to create a list of cumulative sums: [5, 15, 30, 50]. Then, we choose a random number between 1 and the total weight (50, in this case) and use a modified binary search to look for that number in the list of cumulative sums. That modified binary search will return the closest corresponding index to the random number, and we can return the category label associated with that index.

```

def binary_search(a, k):
    lo, hi = 0, len(a) - 1
    best = lo
    while lo <= hi:
        mid = lo + (hi - lo) // 2
        if a[mid] < k:
            lo = mid + 1
        elif a[mid] > k:
            hi = mid - 1
        else:
            best = mid
            break
        if a[mid] - k > 0:
            best = mid
    return best

import random
categories = ['A', 'B', 'C', 'D']
weights = [5, 10, 15, 20] # list of weights
# cumulative sum [5, 15, 30, 50]
cum_sum = [sum(weights[:i]) for i in range(1, len(weights) + 1)]
k = random.randrange(cum_sum[-1]) # choose randomly in range (0, total sum)
i = binary_search(cum_sum, k) # binary search for k
return categories[i]

```

If there are K categories and the total sum of weights is N , where $N \gg K$, then this method uses $O(K)$ space and has $O(K)$ runtime (since the binary search part is $O(\log(K)) < O(K)$), whereas the first method uses $O(N)$ space and $O(N)$ time to create the full list.

Solution #9.21

The brute-force solution to find the maximum common subarray would be to iterate over all possible subarrays for each input and compare the subarrays to verify if they match. However, this takes $O(N^5)$ time, since there are N^2 subarrays for both arrays, and there is an $O(N)$ check to compare any two subarrays. It is inefficient because it doesn't exploit the overlapping subproblems. Specifically, once you have a smaller common array, you can see if the next character between the two arrays is the same, which extends that common subarray by one.

Due to these overlapping subproblems, we can utilize dynamic programming: say that $dp[i][j]$ denotes the length of the longest common subarray for $a[:i]$ and $b[:j]$, where a and b are the two arrays. If the i -th character of a and j -th character of b match ($a[i-1] = b[j-1]$), then we can extend the common subarray by one ($dp[i][j] = 1 + dp[i-1][j-1]$). Therefore, we can iterate over all relevant (i, j) , where i ranges from 1 to the length of a , and j ranges from 1 to the length of b , and update $dp[i][j]$ within each iteration to keep track of the maximum value seen:

```
def longest_common(a, b):
    m = len(a)
    n = len(b)

    dp = [[0 for i in range(n+1)] for j in range(m+1)] # setup dp

    max_val = 0
    for i in range(1, m+1):
        for j in range(1, n+1):
            if a[i-1] == b[j-1]:
                dp[i][j] = 1 + dp[i-1][j-1] # update dp[i][j]
                max_val = max(max_val, dp[i][j]) # keep track of max
    return max_val
```

The time complexity is $O(MN)$, where M is the length of a and N is the length of b since, in the worst case, we will fill out every cell accordingly in a bottom-up manner. The space complexity is $O(MN)$ as well, to accommodate the 2d-array of cached results.

Solution #9.22

The brute-force method to find the maximum increasing subsequence sum (MISS) would be to examine all possible subsequences (checking if it's increasing) and then calculate the sums of each to compare. Since there are N^2 possible subsequences (each of which takes $O(N)$ time to sum/verify it's increasing), this brute-force method takes $O(N^3)$ to calculate the MISS. However, this is inefficient because it doesn't take advantage of the overlapping subproblems: if you have an existing best MISS so far, and if we encounter a new element larger than the current maximum of the MISS, we can extend that MISS by that element. As such, we can use dynamic programming to store previous MISS values.

To be more precise, say we have an array tracking the max increasing subsequence sum (MISS) up to index i of the array. We can solve for that index by looking over all previous indices j (each with the

MISS up to index $j < i$). We know that we have a new MISS if the current element and the previous MISS sum up to more than the current MISS.

```
def max_subseq_sum(arr):
    n = len(arr)
    res = [0 for x in range(n)] # store results
    for a in range(n):
        res[a] = arr[a]
    for i in range(1, n):
        for j in range(i):
            if arr[j] < arr[i] and res[i] < res[j] + arr[i]: # extend the MISS
                res[i] = res[j] + arr[i] # add incremental element
    return max(res)
```

The time complexity is $O(N^2)$ due to the doubly-nested for-loops, and the space complexity is $O(N)$ to store the array of MISS results.

Solution #9.23

At first, finding the smallest number of perfect squares that sum up to n may look like an application of a greedy algorithm, where you try to find the biggest square less than the value. However, consider $4^2 + 5^2 = 16 + 25 = 41$ as an example. If you used $6^2 = 36$, you'd get a non-optimal result of $36 + 4 + 1$ rather than $25 + 16$.

As such, to get the correct answer for 41, check the smallest number of perfect squares for 41-1, 41-4, 41-9, etc. for all squares up to 41. There are overlapping subproblems, so we can use dynamic programming to store sub-results. Let $\text{res}[i]$ represent the smallest number of perfect squares that sum to n . Our recursive step is:

```
res(i) = min(res(i), res(i-j^2) + 1)
```

where j is an integer less than or equal to the square root of i . This is because either the current value is minimal, or we can subtract off a square and yield a smaller number of squares to add up to the remaining value. We can solve the subproblems in a bottom-up manner:

```
def square_count(n):
    res = [x for x in range(n+1)] # store results
    for i in range(2, n+1):
        for j in range(1, int(i ** 0.5) + 1):
            res[i] = min(res[i], res[i - j ** 2] + 1)
    return res[n]
```

The time complexity is $O(N^2)$ due to the doubly-nested for-loops, and the space complexity is $O(N)$ to store the array of results.

Solution #9.24

To find all combinations, we can use a method called backtracking, which builds upon a solution set according to some constraints and abandons paths that cannot lead to a valid solution. To get all combinations with k elements, first, pick an initial element from the set of existing numbers, then concatenate that element with all other possible combinations with $k-1$ elements produced so far, which occurs in a recursive call.

For instance, assume we have a `backtrack()` helper function that takes in the following parameters: n , k , res (which is the resulting list of lists that we can return eventually), $combo$ (any list of elements that is a candidate combination), num (the number of elements in the list) and $start$ (where we start within a list in generating combinations).

Then the logic is as follows: anytime num is equal to k , append the $combo$ list to res . If $start$ is beyond n or num is beyond k , we can return early. Otherwise, going from $start$ until n , we can do the following:

1. Add the current element to $combo$.
2. Recursively call `backtrack()` on the existing version of a $combo$ with updated num and $start$ parameters.
3. Remove that element from $combo$.

Step #3 is essential, because further combinations generated do not always include that element from Step #1. In the end, we return the result we obtain from calling `backtrack()`:

```
def combos(n, k):
    def backtrack(n, k, res, combo, num, start):
        if num == k:
            res.append(list(combo)) # add to result
        if start > n or num >= k:
            return
        for i in range(start, n+1): # iterate over every element
            combo.append(i)
            backtrack(n, k, res, combo, num+1, i+1) # recurse
            combo.remove(i)
    res = []
    backtrack(n, k, res, [], 0, 1)
    return res
```

The time complexity is $O(N^k)$, and there is no way to circumvent it since that is the number of returned combinations. This also holds true for the space complexity.

Solution #9.25

The brute-force way is to check every possible substring and see if it's well-formed. A more optimal way is to incrementally grow the substring, and use a stack to make sure parentheses are balanced and the string is well formed. For every left parenthesis, push its index within the string onto the stack. For every right parenthesis, pop the topmost element and then calculate the length based on the current index versus the index of the first leftmost parentheses of the current valid substring (the

top element of the stack). Hence, the difference between the current index and that top element in the stack is potentially the longest length of a well-formed string seen so far. In the case where the stack is empty after the pop, we can add the current index to the stack. This occurs when you have right parentheses before left parentheses.

To address the case where the first character is a right parentheses, we can add an initial dummy value onto the stack. This dummy value needs to be -1 for the following reason: we would give the correct length of the substring when popping the first ' $($ ' at matching the corresponding ' $)$ ' at $\text{stack}[i]$ as $i + 1$, only if the initial dummy value is -1 .

```
def longest_parens(s):
    stack = []
    longest = 0
    n = len(s)
    stack.append(-1) # initial dummy value
    for i in range(n):
        if s[i] == '(':
            stack.append(i) # append index
        else:
            stack.pop()
            if len(stack) == 0:
                stack.append(i)
            else:
                longest = max(longest, i - stack[-1]) # get length
    return longest
```

The time complexity and space complexity are both $O(N)$, since the stack goes through every element one time.

Solution #9.26

To find the length of the longest path of increasing integers, first, consider a single element in the matrix. Presume that we have found the longest path of increasing integers for all other paths not involving that number. When evaluating the element, we know that the length of the new longest path must be 1 plus the maximum length of the longest paths of its four neighbors. To be specific, for any given indices (i, j) , there are up to 4 potential neighboring indices, of which the longest increasing path from (i, j) is 1 plus the max of the longest increasing path from its four neighbors, if the element is larger than its neighbor.

This structure lends itself well to dynamic programming, and as such, we can keep track of a max length and a table (to serve as a cache) with indices (i, j) , which tracks the length of the longest increasing path starting at index i, j of the matrix. To calculate this max, we can utilize depth-first-search (DFS), where we recurse on the 4 neighboring elements. For each DFS, we also need to keep track of a previous max element (since the path needs to be increasing — so if the current value at i, j is less than that value, we can just return 0). To start the algorithm, we can instantiate the max to be $-\infty$ and start with an empty dictionary for the table cache, and then run DFS for each i, j :

```

def longest_increasing_path(mx):
    table = {} # cache
    if len(mx) == 0:
        return 0
    m = len(mx)
    n = len(mx[0])

    def dfs(i, j, prev, table): # DFS helper
        # edge case
        if(i < 0 or i >= len(mx) or j < 0 or j >= len(mx[0]) or mx[i][j] <= prev):
            return 0
        if (i, j) in table:
            return table[(i, j)] # get cached value
        curr_len = 1 + max(dfs(i-1, j, mx[i][j], table),
                            dfs(i+1, j, mx[i][j], table),
                            dfs(i, j-1, mx[i][j], table),
                            dfs(i, j+1, mx[i][j], table))
        table[(i, j)] = curr_len # get max
        return curr_len

    for i in range(m): # call DFS from each i,j
        for j in range(n):
            dfs(i, j, -float("inf"), table) # set initial max and table

    return max(table.values())

```

The time complexity is $O(MN)$, since in the worst case, the DFS will visit every cell within the matrix. The space complexity is $O(MN)$ as well to accommodate the table cache.

Solution #9.27

The brute force method to find consecutive integers that sum to n is to start creating lists from 1 to n , and keep adding consecutive integers to each list until they equal n or go over. However, this solution has a runtime of $O(N^2)$ and is inefficient. Here is that code:

```

def consecutive_sum(n):
    num = 0
    for i in range(1, n+1):
        total = 0
        while total < n:
            total = total + i
            i += 1
        if total == n:
            num += 1
    return num

```

Due to the structure of this problem, we can look for some properties that valid solutions follow, to more intelligently create candidate lists rather than brute-force generating all of them. Let's reframe the problem in a mathematical way. Say we start at some number k , and go up to some number $k + m$, so we have a sequence of $m + 1$ terms. Note that:

$$k + (k + 1) + \dots + k + m - 1 = (2k + m)\left(\frac{m + 1}{2}\right)$$

Now, if we set this to n , we have the following: $(2k + m)\left(\frac{m + 1}{2}\right) = n$

Solving for k in terms of m and n , we get: $2k = \frac{2n}{m + 1} - m$

Since k needs to be an integer for a valid solution, we can check whether the right-hand side satisfies two conditions:

1. $2n$ is divisible by $m + 1$ (otherwise the right-hand side is not an integer).
2. The right-hand side is divisible by 2 (i.e., it is an even number, since $2k$ is even).

Note that for the right-hand side we have: $\frac{2n}{m + 1} - m$

We do not need to check all values of m since we need k to be a positive integer. Thus, to find the bounds on m , we take the above expression and find when it is greater than 0:

$$\frac{2n}{m + 1} - m > 0$$

$$\frac{2n}{m + 1} > m$$

$$2n > m(m + 1) = m^2 + m$$

Since $m^2 + m > m^2$, we have:

$$\begin{aligned} 2n &> m^2 \\ \sqrt{2n} &> m \end{aligned}$$

As such, we only need to iterate m from 1 to $\sqrt{2n}$:

```
import math

def consecutive_sum(n):
    upper_limit = int(math.sqrt(2*n))
    num = 0
    for m in range(upper_limit):
        if (2*n) % (m+1) == 0 and (2*n/(m+1)-m) % 2 == 0:
            num += 1
    return num
```

The runtime is $O(N)$, and the space complexity is $O(1)$.

Solution #9.28

The brute-force way to find the median of a continuous stream of elements is to store all seen elements in an array and then find the median there. However, this is inefficient in the find operation - - it is generally $O(N \log N)$, since a sort is needed.

A more efficient way is to keep track of two heaps: a max-heap to store the smaller half of numbers, and a min-heap to store the larger half of numbers. The intuition is that the median will always be either the largest value of the max-heap, the smallest value of the min-heap, or the average of the two (try some examples out on paper). The question, then, is how to maintain the balance, since this approach only works if both elements contain about half the elements (otherwise, the median would be deep inside one of the two heaps if the sizes were skewed). To handle this, simply check the heap size when adding elements. If the size of one heap is larger than the other, add an element from one heap to another to balance it out.

Below, we use the `heapq` library for a priority queue implementation and use the negatives of values (since by default the priority queue implementation in Python weights elements by their value from smallest to largest):

```
import heapq

class MedianFinder(object):
    def __init__(self):
        self.min_heap = []
        self.max_heap = []

    def add_num(self, num):
        heapq.heappush(self.max_heap, -num) # add negative for max heap
        heapq.heappush(self.min_heap, -heapq.heappop(self.max_heap))
        if(len(self.min_heap) > len(self.max_heap)):
            # balance heaps
            heapq.heappush(self.max_heap, -heapq.heappop(self.min_heap))

    def find_median(self):
        if len(self.max_heap) > len(self.min_heap): # return max heap value
            return 1.0 * -self.max_heap[0]
        else: # return average of two
            return 1.0 * (self.min_heap[0] - self.max_heap[0]) / 2
```

The runtime of `add_num()` is $O(\log(N))$ because each heap insertion (up to 3) takes $O(\log(N))$ time. The runtime of `find_median()` is $O(1)$. Therefore, the overall procedure takes $O(\log N)$. The space complexity is $O(N)$, since N elements are stored among the heaps.

Solution #9.29

Instead of checking whether the whole input string matches the regex, let's start by breaking down the problem and considering just one character at a time. We can start from the back of both the regex and the input strings. If both are regular 'a-z' characters and do not match, we can immediately return false. Otherwise, check whether the remaining regex string and input strings match without those two ending characters.

Now, say we see that the last character of the regex string is '?'. Because '?' matches any single character, we can check if the regex without the "?" matches the rest of the input string without the most recent character. Here we begin to see the recursive nature of the problem.

If we see a '*', then there are two cases. The first is that '*' could represent an arbitrary number of characters. In this case, its behavior is similar to '?' - for any one character, we just recursively check the rest of the input minus that one character with the same regex string. On the other hand, '*' could have the behavior of an empty string, in which case check the rest of the input with the same regex string without the '*'.

Note the overlapping subproblems: if we find that a part of the regex matches a substring of the input, we can determine whether another character added onto the regex ('?', '*', or character) matches the input substring with another character added on to it.

In particular, say we have inputs s (string) and r (regex). We can keep track of a table that stores Booleans for the function output. In this table, indices (i, j) will store the function result of checking whether or not the substring of the input up to character i matches that of the regex substring up to character j (whether $s[:i]$ and $r[:j]$ match). Thus, after this table is filled, we want to check the index $i = -1, j = -1$ (the last element of the table) to confirm whether the full regex string matches the full input string.

We can start by instantiating the dp matrix to be false for all indices. Then, for the value at $dp[0][0]$, we can set it to true, since it's trivially true that an empty regex string matches an empty string. In addition, since we know that any regex substrings with just '*' characters matches any substring, we can set $dp[0][j]$ to be 0 as long as the regex contains '*' characters up to index j .

Summarizing the above, our recurrence relations are as follows:

1. If $r[j-1]$ is '?' or matches $s[i-1]$, then we set $dp[i][j] = dp[i-1][j-1]$ (recursive call where we check everything before the current last character in both the regex and input string).
2. If $r[j-1]$ is '*', then the result is the logical OR of the following 3 sub-cases:
 - a) $dp[i-1][j-1]$ (recursive call as if the '*' functioned as a '?').
 - b) $dp[i][j-1]$ (recursive call where the '*' matches the current last character and will continue from there); or
 - c) $dp[i-1][j]$ (recursive call where the '*' matches an empty string).

The time complexity is $O(MN)$, where M is the length of the first string, and N is the length of the second string, since, in the worst case, we will fill out every cell accordingly in a bottom-up manner. The space complexity is $O(MN)$ as well, to accommodate the 2-d array.

If neither of the prior cases applies, then we ignore changing $dp[i][j]$ since the entry will be false (and we can set it to that by default). Finally, we return the last indices for the string and regex within dp, i.e. $dp[-1][-1]$:

```
def wildcard_match(string, regex):
    m = len(string)
    n = len(regex)
    dp = [[False for _ in range(n+1)] for _ in range(m+1)] # initialization
    dp[0][0] = True
    for j in range(1, n+1): # filling in Trues for any *
        if regex[j-1] != "*":
            break
        dp[0][j] = True
```

```

for i in range(1, m+1):
    for j in range(1, n+1):
        if regex[j-1] == '?' or regex[j-1] == string[i-1]: # case 1
            dp[i][j] = dp[i-1][j-1]
        elif regex[j-1] == "*": # case 2
            dp[i][j] = dp[i-1][j-1] or dp[i][j-1] or dp[i-1][j]
return dp[-1][-1]

```

Solution #9.30

To find the optimal location of the new fire station, minimize the following cost function, where (x_c, y_c) represents the coordinates of the new fire station, and there are n houses each with coordinates (x_i, y_i) :

$$L(x, y) = \sum_{i=1}^n \sqrt{(x_c - x_i)^2 + (y_c - y_i)^2}$$

One possible method to solve this optimization problem is through grid search, where you manually sweep through all possible locations. Our solution, however, will focus on gradient descent. Recall that gradient descent allows you to find a local minimum for a convex function by taking steps in the opposite direction of the gradient of the function. Note that because there is no other local minimum other than the global minimum, and the sum of squares function above is convex, gradient descent leads us to an optimal result. For each iteration of gradient descent, as we refine the current location and get closer to the optimal location, we decay the learning rate and take smaller and smaller steps until we arrive at the solution.

Before we can implement gradient descent in code, we need to calculate the partial derivatives of the sum of squares function for both x and y . We get the following for the partial with respect to x :

$$\frac{\partial L(x, y)}{\partial x} = \frac{(x_c - x_i)}{\sum_{i=1}^n \sqrt{(x_c - x_i)^2 + (y_c - y_i)^2}}$$

And an analogous partial with respect to y : $\frac{\partial L(x, y)}{\partial y} = \frac{(y_c - y_i)}{\sum_{i=1}^n \sqrt{(x_c - x_i)^2 + (y_c - y_i)^2}}$

To control the rate of gradient descent, we initialize the following hyperparameters:

1. Initial learning rate: the starting value for the learning rate. The learning rate is used to scale the size of the moves we take in the opposite direction of the gradient.
2. Rate of decay: used to decrease the learning rate. We make the learning rate progressively smaller to converge upon an eventual answer.
3. Terminal rate: the stopping condition. When the learning rate becomes smaller than the terminal rate, we can exit the gradient descent process.
4. Damping factor: constant that helps with numerical stability. During gradient descent there are always oscillations back and forth around a range of values --- by applying a damping factor to reduce the magnitude of step sizes, there is a better chance of a smooth convergence.

To implement gradient descent, we begin with the mathematical centroid of all the coordinates as the starting estimate for the optimal location (x_c, y_c) . Then, we can set the hyperparameters defined above and update the values of (x_c, y_c) iteratively until the terminal condition is hit and the estimate for the optimal location is reached:

```

import math

def get_optimal_location(coords):
    def partialX(x_c, y_c): # partial with respect to x
        return sum(((x_c-x_i)/math.sqrt((x_c-x_i)**2 +
                                         (y_c-y_i)**2)) if x_c != x_i else 0) for x_i, y_i in coords)

    def partialY(x_c, y_c): # partial with respect to y
        return sum(((y_c-y_i)/math.sqrt((x_c-x_i)**2 +
                                         (y_c-y_i)**2)) if y_c != y_i else 0) for x_i, y_i in coords)

    rate = 1 # initial learning rate
    rate_decay = 0.99 # rate of decay
    terminal_rate = 1e-8 # 1e-8 # terminal rate (stop condition)
    damping_factor = 0.7

    l = len(coords)
    x_c = sum(x for x, y in coords) / l # centroid x
    y_c = sum(y for x, y in coords) / l # centroid y

    dx = 0
    dy = 0

    while rate > terminal_rate:
        dx = partialX(x_c, y_c) + damping_factor * dx
        dy = partialY(x_c, y_c) + damping_factor * dy
        x_c = x_c - rate * dx # update coordinates
        y_c = y_c - rate * dy
        rate = rate * rate_decay # update learning rate
    return (x_c, y_c)

```

To assess the runtime complexity, assume that the minimum values of (x_i, y_i) are given by (x_{min}, y_{min}) and the maximum values are given by: (x_{max}, y_{max}) . Let $x_d = x_{max} - x_{min}$ and $y_d = y_{max} - y_{min}$. Then, we know the upper bound, in terms of points tested, is $x_d * y_d$. Since calculating the partial derivatives for x and y takes $O(N)$ each, then the time complexity should be $O(N * x_d * y_d)$. The space complexity is $O(N)$ since, during the partial derivatives calculating, we are summing over N elements in a temporary list.

The algorithm's runtime is also dependent on the hyperparameters that control the gradient descent. Here is how the runtime is affected by each:

1. Initial learning rate: this is often set at 1. The higher it is, the longer the runtime, since the initial learning rate needs to converge to the terminal rate.
2. The rate of decay: this is set between 0 and 1. The higher it is, the faster the runtime, since a larger rate of decay means reaching the terminal learning rate faster.

3. The terminal rate: the higher it is, the faster the runtime, since convergence conditions are reached more quickly.
4. The damping factor: set between 0 and 1. The smaller the value, the faster the runtime, since there is less damping; hence, larger moves due to the gradient.

Product Sense

CHAPTER 10

A magikarp, a one-legged man in an ass-kicking contest, and an ejector seat in a helicopter. These three are examples of things more useful than a data scientist with a weak product sense and business acumen. Because data scientists often work cross-functionally with product managers (PMs) and business stakeholders to help create product roadmaps and understand the root cause of various business problems, they are expected to have a strong product and business intuition. It's not just data scientists who can expect product-sense interview questions — these topics are also frequently covered during product analyst, data analyst, and business intelligence analyst interviews.

Between questions on the art of selecting product metrics, troubleshooting A/B test results, and weighing business trade-offs, the scope of product interview questions is massive. But fear not! In this chapter we cover both actionable strategies to approach the four most common types of product interview questions you'll face and long-term tips to develop your overall product and business sense. We also solve 18 real product-sense interview questions from companies like Amazon, Airbnb, and Facebook.

Four Most Common Types of Product Interview Questions

Before we dive into specific product management topics, it's important for you, the reader, to first get a glimpse at the four most common types of product-focused data science interview questions:

- **Defining a product metric:** What metrics would you define to measure the success of a new product launch? If a product manager (PM) thought it was a good idea to change an existing feature, what metrics would you analyze to validate their hypothesis?

- **Diagnosing a metric change:** How would you investigate the root cause behind a metric going up or down? What if other counter metrics changed at the same time — how would you handle the metric trade-offs?
- **Brainstorming product features:** At a high level, should a company launch a particular new product? Why or why not? For an existing product, what feature ideas do you have to improve a certain metric?
- **Designing A/B tests:** How would you set up an A/B test to measure the success of a new feature? What are some likely pitfalls you might run into while performing A/B tests, and how would you deal with them?

By keeping these frequently asked types of questions top of mind, we hope you'll better understand how the following high-level advice can be concretely applied to acing product questions.

Big-Picture Advice for Product Sense Interview Questions

Framework for Approaching Product Interview Questions

The tips below work for approaching product questions, as well as for the occasional business question:

- **Ask clarifying questions:** Make sure you understand the user flow for a product, who the end users are for the product, who the other stakeholders are that are involved with this problem, and what product and business goals we aim to achieve by solving the problem. Even if you've done your research into the company and product and know many details, frame your knowledge as a question so you don't inadvertently head down the wrong path. For example: "I know Robinhood's mission is to democratize finance for all. It seems this crypto wallet feature is meant to democratize access to crypto currencies, which can also help us better compete with Coinbase. Am I on the right track?"
- **Establish problem boundaries:** These are big problems; scope them down. Establish with your interviewer what you're purposely choosing to ignore to solve the problem within the time frame of the interview.
- **Talk Out Loud:** You've seen this tip now many times. These interviews are held to see your thought process - and until Elon Musk invents mind reading at Neuralink, you need to voice your thinking!
- **Be conversational:** Don't talk *at* the interviewer — talk *to* the interviewer. Engage them in conversation from time to time as a means of checking in. For instance, "I think a good metric for engagement on YouTube is average time spent watching videos, so I'll focus on that. How does that sound?"
- **Keep goals forefront:** It's easy to get lost in technical details. Never forget your answer stems from the company's mission and vision, which you hopefully articulated at the start of your conversation!
- **Bring in outside experience tactfully:** Because these problems are rooted in the real world, it's okay to flex your past domain experience. Just don't go overboard and come across as arrogant or cargo cult-y by saying, "This is the only way a problem should be solved, because that's how we solved it at Google."

Feeling like there's too many tips to keep track of? Ultimately, if you can remember just one thing when solving product problems, it's this: pretend you've already been hired at the company as a data scientist. You're just having a meeting about the problem with another co-worker. When you adopt the mindset that you're already working for the company, behaviors like talking out loud with your "co-worker" or keeping the company mission top of mind should come naturally.

How to Develop Your Product Sense

Because data scientists help Product Managers (PMs) quantitatively understand the business and look for opportunities for product improvement within the data, they play a crucial role during the product roadmap creation process. As such, questions asking you to brainstorm new products and features are very common during product-focused data science interviews. The best way to improve your performance on this type of problem is to improve your general product sense.

However, don't let the term "product sense" faze you; this isn't an innate gift you're born with, but, rather, a skill that can be developed over time. By following the tips in this section to enhance your overall product sensibilities, you won't freeze up like a deer in the headlights in your next interview with Google, when you're asked to brainstorm features to help students better use Google Hangouts. Instead, you'll tackle the problem with the confidence of Sundar Pichai after yet another Alphabet quarterly earnings beat.

The Daily Habit You Need to Build Your Product Sense

An easy way to develop your product sense is through analyzing the products you naturally encounter in your daily life. When using a product, think about:

- Who was this product created for?
- What's the main problem it was designed to solve?
- What are the product's end-user benefits (this is bigger than simply what problem it solves!)?
- How do the visual design and marketing copy help convey the product's purpose and benefits?
- How does the product tie in with the company's mission and vision?

A great deal of good product sense is having empathy for a product's or service's users. That's why, when answering the above questions while analyzing a product or service, you must try to put yourself in a user's shoes.

Take Snapchat, for example. Sure, you can post photos to your story or send messages to people on Snapchat. But so can iMessage, WhatsApp, Instagram, and Messenger. At a deeper level, Snapchat is about being able to stay in touch with your closest friends in a casual, authentic way. That's why opening up the Snapchat app puts you directly on the camera, in order to make it frictionless to express yourself and live in the moment — two core elements of Snap's company mission. It's also why photos and messages disappear by default — this lowers the barrier to expressing yourself and pushes you to share whatever you captured rather than spending time editing a photo you know will soon be gone.

Contrast this with Instagram, which defaults to the feed to promote consumption rather than visual communication. On Snap's more polished rival, you are made to feel that your posts need to be perfect, lest they be judged by acquaintances and extended family. There's an associated permanence to the photos you post, which takes away some of the whimsicalness that Snap's optimized for. Similarly, Snap's default ephemeral messages set it apart from other messaging platforms like iMessage and Instagram.

While we could go on and on about the two apps, which have overlapping functionality but serve two very different user needs, we want to emphasize that the point of this exercise is to go beyond simply relegating the app to just a “dumb Gen Z” thing or “basically the same as Instagram.” By thinking more critically about products in your everyday life, you can sharpen your product intuition, ace interviews, and eventually build successful products in the workplace.

Calibrate Your Intuition by Analyzing Reviews

The daily habit of analyzing products, who they’re made for, and what benefits they offer is fine and dandy, but how do you know you’re right? How do you know if your reasoning lines up with how others perceive the product and its benefits? One way to calibrate yourself and fine tune your intuition is by analyzing customer reviews.

By looking at the positive reviews and press for the product, you can see how user benefits are described and what is expected of the product. Reading these positive reviews helps you better articulate user benefits; it also helps you notice benefits you might have taken for granted. Similarly, by reading negative reviews, you can understand how the products you encounter are falling short in meeting user needs. By comparing the issues the negative reviews flagged against your own product evaluation, you can start to develop a more critical eye. Additionally, negative reviews are a great source of ideas for product brainstorming.

Reddit is a great place to see unfiltered conversations about products and services. For apps, also check out the App Store and Google Play Store. For enterprise products, check out G2 Crowd and Gartner Special Reports. For physical products, check out the reviews on Amazon. (While you’re there, help us immensely by spending two minutes to rate and review our book — we greatly appreciate this!).

How to Build Your Business Sense

While it’s exceedingly rare to be explicitly asked business questions like “How do you measure the health of the enterprise sales pipeline?” or “How do you model free cash flow from revenue?” having some general business knowledge is crucial. Why? Because cash rules everything around me (C.R.E.A.M.).

Wu-Tang reference aside, the honest truth is that, in the workplace, having the best technical skills possible won’t matter if you solve the wrong business problems. By following the money and understanding how the products you work on help the business make more money, you give yourself a better chance of working on high-leverage technical projects. Stronger business sense helps even your product-sense get sharper, since ultimately knowing what products to build and what product metrics to improve upon stems from the company’s business model and strategy.

The Daily Habit You Need to Build Your Business Sense

Much like the daily habit of understanding user incentives we recommended earlier in order to develop product sense, asking yourself the following questions when you encounter a new business can help develop your business sense:

- **Business Model:** How does the business monetize? What product levers can be pulled to improve the business’s ability to monetize?
- **Metrics:** Which key performance indicators (KPIs) would I measure if I were working on this business? What factors and variables influence those particular metrics?
- **Landscape:** How does the business fit into the broader ecosystem of the companies comprising that industry? What companies does the business compete with, and what companies does it partner with?

Another way to grow your business awareness is by reading some of the best business books out there. We'll admit, there's a lot of fluffy business books out there, most of which are just self-help manuals or written to pad the author's ego. And plenty should have been a blog post but got padded with filler to become a book. Then there's books like *How to Win Friends and Influence People* or *Think & Grow Rich*, which, while interesting, don't really help you foundationally understand business. As such, we curated a list of what business and product books helped us out the most in our career: acethedatascienceinterview.com/business-books.

How to Hack Your Domain Experience

We'll let you in on a psychological quirk interviewers have which you can hack: your interviewers likely live in a bubble. They're knee deep in the problem in workplaces designed to be hard to unplug — eating the company-provided free three meals a day, talking about the problem at dinner with their teammates, and thinking about work while commuting back home on the company shuttle. That's why it shouldn't be surprising that, come interview time with you, an outsider, they forget you don't have as much context as they do! Bubbles are real, y'all, and techies love to live in them!

While unfair, the data scientist's familiarity and time spent with the product and company can cloud their ability to accurately assess your product sense. Plus, you might be interviewing against internal candidates with better context on the problem. Or you might be competing against candidates who worked in similar businesses, so naturally get a leg up. That's why doing your homework is one of the best ways to level up.

Doing Your Homework: Uber Eats Example

As a real-world example, let's say that you had an upcoming interview with the Uber Eats team and wanted to prepare for any product or business case questions you might be asked. To prepare, you should learn how Uber as a whole makes its money and how much revenue comes from their transportation products versus their delivery business. You should dig deeper into Uber Eats in order to understand how it fits into Uber's overall strategy as a logistics and transportation company. In addition, learn about the common metrics used to measure two- and three-sided marketplaces. Finally, prior to the interview, you should attempt to uncover the key inputs for Uber's pricing and payout algorithms that determine how much it charges a customer and how much it pays the delivery driver and restaurant.

Most of the research described above can be done with information available free-of-charge on the web, by searching "company name business model." Google News reports what financial analysts say about the company. If the company is public, looking at its earnings reports provides another good way to hone your business sense, and this lets you directly see what key business metrics are being tracked. If the company you are analyzing isn't public, see if there are comparable public businesses, since they'll most likely use similar metrics to what the private company tracks internally.

Another great resource which candidates unfortunately tend to underutilize (to their great peril) is a company's engineering blog. This resource gives you an inside look into the business and the technical systems underlying it. For instance, Uber Eats (from the above example) often publishes blog posts with titles like "Optimizing Delivery Times on Uber Eats" and "Food Discovery with Uber Eats." You could discover and deduce a great deal about the business from posts like these. DoorDash's Engineering blog wouldn't be a bad place to look either!

Doing Your Homework Means Use the Damn Product

It's easy to be an armchair analyst, reading earnings reports and blogs about the product. But at some point, it's absolutely necessary to do your own due diligence by exploring the product on your own.

And even though you probably knew this, and maybe the recruiter told you to do this already, we are surprised by how many candidates we've coached that skip this crucial step.

So, let's be honest — how are you going to reason about a high-level business strategy if your fundamental understanding of the product is weak? In the Uber Eats example, you, like most candidates, have most likely ordered food before, but have you also tried to download the Uber Driver app? Have you looked up the UI that restaurants use to fulfill orders? Have you looked at Uber's marketing website targeted at signing up new eateries, which explicitly spells out the value props that Uber offers restaurants? Putting in this extra effort to understand the product from its multiple angles would easily put you in the top 1% of candidates.

This much company and product research is crazy, right?

Yes, dear reader, we realize this is a lot of work to do before each interview. But the more company and product knowledge you can sneak into your answers, the better you'll do. Additionally, a common interview question — especially at smaller companies — is "Did you have a chance to use our product?" followed up with "What did you think? Any ideas on how to improve it?" This amount of preparation will help you knock these questions out of the park.

Finally, even if there aren't product-sense questions directly concerning the information you researched, don't be dismayed. Your effort wasn't wasted! At the end of the interview, your research would enable you to ask the interviewer more intelligent questions than would have been possible otherwise. Moreover, it would demonstrate your passion and willingness to put forth extra effort. Instead of asking the interviewer run-of-the-mill questions like "What's your favorite part about working at Uber Eats?" you can instead comment, "I was reading about the food delivery time estimation algorithm on your blog and found X fascinating. I was curious why you used approach Y, and if you ever thought about trying out Z instead?" Suddenly, you're showing enthusiasm and insight into the business, offering a suggestion, and gaining the opportunity to learn something new and meaningful in the process.

Metrics for Product & Case Interviews

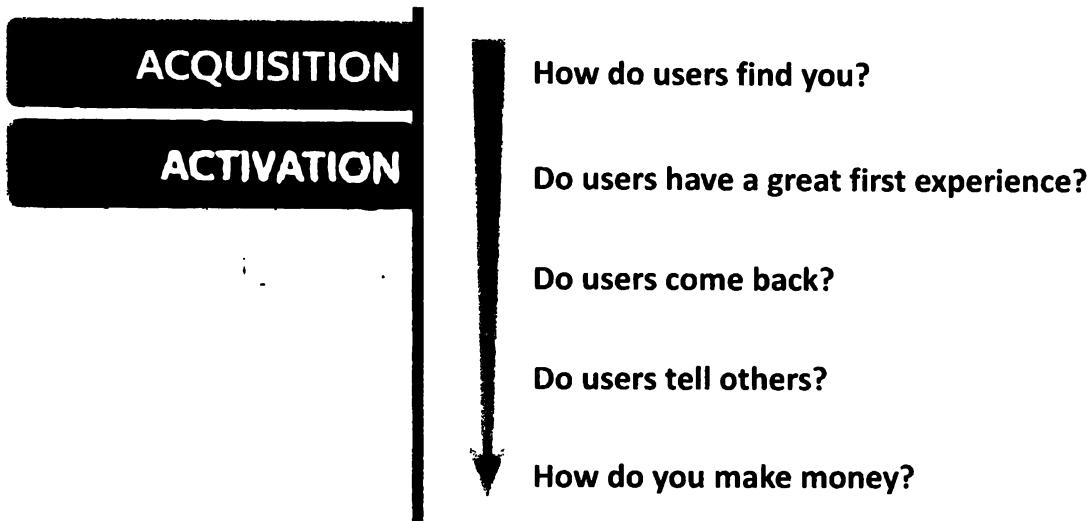
User Acquisition Funnel

Before we address the art of metric selection, it's key we cover the popular marketing concept of a customer acquisition funnel. There is no one funnel to rule them all — you'll see different frameworks depending on the product's use case (B2B vs. B2C), how the product is monetized (one-time purchase vs. recurring vs. freemium). One specific customer lifecycle we like is the pirate metrics framework created by the founder of startup accelerator 500 Startups, Dave McClure. It's called *pirate metrics* because the funnel steps form the acronym *AARRR*.

User Acquisition Metrics

User acquisition metrics try to capture how people are finding and trying out your product. Standard metrics used to track success in this area include new user counts, sign-up conversion rates, and customer acquisition costs (CAC). You'll often hear the word "top of funnel" when referring to this stage of the customer journey. For Pinterest, relevant user acquisition metrics might be the number of new users who download the app or the number of new customers who create an account per week. Related metrics Pinterest would track for this step include the sign-up conversion rate and customer acquisition costs (CAC).

AARRR Pirate Metrics



User Activation Metrics

User activation metrics refer to the point at which a user has successfully onboarded and reached the product “aha” moment — the spot where they experience the core value of the product. For DoorDash, it could be the number of people who make their first delivery order per week. On Instagram, it could be after a user views 10 unique posts in their newsfeed, or after they make their first post or story.

User Engagement

While not an explicit step in the AARRR pirate metrics framework, we thought it was important to bring up the term user engagement. After user activation, it’s important to then look at how often and how well users interact with the product. The product interaction being measured could be time spent on platforms like Facebook or rides booked for Uber. A common way to incorporate frequency into the measure of user engagement is by looking at the unique number of users who took a core action within a fixed time period. For example, Facebook measures daily active users (DAU), weekly active users (WAU), and monthly active users (MAU).

User Retention Metrics

User retention refers to whether or not users keep coming back to a product or service over a prolonged period of time. The process of users joining and then leaving permanently is called churn. Maximizing retention and minimizing churn is a primary focus of most businesses, because acquiring a new customer is typically tougher and more costly than retaining an existing user or reactivating a prior user who has churned. Metrics to measure user retention include monthly retention and monthly churn.

User Referral

Referral, where a user shares the product with others, could technically happen at any time. However, in practice, users typically invite others only after having been activated, engaged, and retained on the product for a while. To quantify virality, growth marketers use the k-factor, which is just the

average number of referrals sent per user multiplied by the conversion rate of each referral. A k-factor over 1 would indicate exponential growth.

User Revenue

The popular funnel idea starts to break down, as you may be able to make money from the user at any time — not necessarily after they have been retained or sent referrals. One way to measure a user's revenue is the *lifetime value per customer* (LTV) — how much money a customer brings into the business before they churn. Sustainable companies seek to have a high LTV-to-CAC ratio, so that their marketing efforts pay off before the user eventually churns.

Do Your Homework: Metrics Edition

As mentioned in the “do your homework” section, you need to look up the primary metrics associated with the type of company with which you are interviewing prior to your interview, so as to have a leg up on metric-related questions. For example, if you’re interviewing with an enterprise Software-as-a-Service (SaaS) company like Hubspot or Workday, knowing what terms like “ACV” (average contract value) or “MRR” (monthly recurring revenue) mean is helpful.

As we explained at the start of *Ace the Data Science Interview*, we are merely trying to refresh your memory with quick deep dives. For a better deep dive into different types of companies and their associated metrics, we recommend the book *Lean Analytics* by Alistair Croll and Benjamin Yoskovitz. The book devotes entire chapters to measuring such different business models as a freemium mobile app and a two-sided marketplace. (For a full list of books we recommend, visit acethedatascienceinterview.com/best-books-for-data-scientists)

What Makes a Metric Good or Bad?

Now that we understand the stages of the user journey, we are one step closer to addressing the most common product interview question you’ll face: defining an appropriate metric for a new product or feature. An example interview question of this nature is “How would you measure the success of Facebook Dating?”

Product-focused tech companies ask this type of interview question because data scientists often help product managers determine the best analytics to measure the success of a new product launch or feature change. Note, these conversations don’t necessarily happen just at launch time: often it helps to have the end result in mind before building a new product or feature. As such, often right after the product brainstorm phase, when ideas are being triaged to see if they should make it into the roadmap, conversations about what success would look like occur. Working to define metrics isn’t just a skill for data scientists: data analysts and business intelligence engineers also play a role in building out the dashboards to visualize and monitor the health of the product post launch.

Before we jump into the framework for answering a product definition question, it’s important we clarify what makes a metric good or bad in the first place. It’s easier to start with examples of what makes a metric bad, since a good metric essentially manages to avoid the flaws of bad metrics.

Examples of Bad Product Metrics

Here’s some common types of bad metrics, as well as an example of each for the Facebook dating question above:

- **Vanity metrics:** These metrics sound nice, but don’t capture anything meaningful. For example, dating profiles viewed could be a proxy for engagement, but if this number is too high or low,

does it really tell us if the app is working well? Does it really impact the number of meaningful relationships formed from the product?

- **Irrelevant Metrics:** These metrics aren't tied to the business goal. For example, time spent using Facebook Dating. Does it really matter? Sure, it's an indicator that there is use, but it doesn't capture the true value a dating app offers. Time spent is better for media consumption-related products like YouTube and Netflix, not activity-driven dating apps.
- **Impractical Metrics:** An example of an impractical metric is the number of 3rd dates that occurred. While a good sign of a meaningful match being made, how would you even measure this? More advanced dating apps have a "did you meet?" user prompt, or they use NLP on the conversation to determine if they think you met, but this approach likely works only for a first date, after which the conversation usually moves off-app.
- **Complicated Metrics:** Is the metric easy to explain to stakeholders? If a complex metric changed, would it be easy to understand what actually happened, or would you have to break into its sub-component parts and definition to really understand which pieces were affected?
- **Delayed Metrics:** Number of marriages that occurred. Not only is this impractical to know (just like the 3rd date metric), it would take a very long time to figure out because people tend not to get married within a few months of connecting on an app. While downstream success metrics have their place, you want to track leading indicators so that data can be collected earlier, which enables you to make decisions and notice problems faster.

What Makes a Good Metric Good?

By reversing what makes metrics bad, you end up the following four qualities good metrics possess:

- **Meaningful:** Tied to business goals; isn't easily gamed; is actionable and can drive decisions
- **Measurable:** Simple to consistently and reliably track
- **Understandable:** Easy for stakeholders to understand; intuitive to know what is being measured based on the name
- **Timely:** Can be collected within a reasonable time frame

North Star & Guardrail Metrics

When asked an interview question about defining metrics, you need to go beyond the simple framework of only suggesting good metrics and avoiding bad ones. Besides mentioning the most important and relevant metric to the problem at hand (known as a *north star* metric), you should also mention guardrail metrics (also known as counter metrics). Guardrail metrics are business metrics that should not be degraded while optimizing the metric of interest. These are important to monitor because it's easy to boost a given metric at the expense of counter metrics or other KPIs.

For example, when Kevin was working on Facebook Groups, trying to reduce harmful content like hate speech and spam, his team's topline metric was to decrease the prevalence of harmful posts. One way to achieve this goal is to remove 99% of posts from the feed. You can't have harmful content on newsfeed if there isn't any content on the newsfeed to begin with! Obviously, such an approach doesn't make sense. That's why, for every A/B test, they'd also monitor guardrail metrics (posts made, posts viewed, number of likes and comments per post).

It is good to bring up counter metrics with the interviewer after they agree with your key success metrics. Proactively having this discussion shows that you realize building a product isn't just about

optimizing a specific set of feature-related metrics, but more about holistically making sure the product is benefiting the business at-large.

3-Step Framework to Answer Product Metrics Definition Questions

Now that we understand what good metrics look like, and the stages of the user journey, we can cover the specific framework you can use to answer the question, “How would you measure the success of Facebook Dating?”

Step 1: Clarify the Product & Its Purpose

We need to start by defining the problem. This can be done by clarifying the business purpose behind the product, the product’s goal, who this feature is for, and what the user flow looks like. For the Facebook Dating example, you might clarify:

- Why is Facebook interested in a dating product? Is it to boost engagement on the app? Is it to create another surface for ads, which can help increase ad revenue?
- What’s the point of Facebook Dating — is it meant for casual dating, or people looking for marriage?
- Is the dating feature for all people, or a certain age demographic or orientation?
- How does the product work? Is this a stand-alone app from Facebook, or integrated to be within Facebook itself? Is there a Tinder-esque swipe-based UI? Are there any gimmicks involved, like women initiating the conversation, similar to Bumble?

In the situation with Facebook Dating, let’s assume that you learn that it’s an app, similar to Tinder, but with more elaborate profiles to foster more intentional swipes and matches. You learn it’s being launched in New Zealand first, because it approximates the larger English-speaking populations like the U.S. and the U.K. where Facebook eventually wants to launch the product.

Step 2: Explain the Product & Business Goals

While you might expect that you can just ask the interviewer what the product and business goal are, often, the interviewer is looking to you to synthesize what the main purpose behind the product is, and how it ties into the business. You’ll get bonus points for tying it back to the company mission. This is exactly why earlier we strongly recommended you to do company and product research — it makes a huge difference on this step.

In the case of Facebook Dating, it relates back to the product czars wanting to drive engagement to the app. If you log in to check your matches, it’s easy to see one of the super-targeted ads and drive revenue for the company. Plus, it keeps the rest of the Facebook ecosystem attractive — if you check your notification about a new match, maybe you also check your notification about a group post or a friend’s birthday at the same time. Plus, with in-built messaging, it keeps people within the Facebook, Instagram, Messenger, WhatsApp ecosystem.

Lastly, the Facebook company mission is to help you develop meaningful relationships. Keeping in touch with family, interacting with your community via groups, and messaging with your friends are all part of the mandate. So why not help you in the romantic relationship department as well?

Step 3: Define Success Metrics

It is crucial to determine and measure the main actions a user needs to take in order to drive the product and business goals. You can follow the user-acquisition funnel we talked about earlier to

serve as a framework on how to guide the analysis. As you are defining the metrics, remember to restate how they align with the product and business goals you had mentioned earlier.

Acquisition metrics: How many users sign up and how many users fill out their profile could be great metrics to measure top-of-funnel. This is important, because people don't think of Facebook as a dating company, and it's important to know if Facebook is in the news for privacy and data-use concerns. This reputation can get in the way of users trusting it for their most intimate needs.

Activation metrics: These could be onboarding metrics if the profile is filled out and a certain number of photos are uploaded. Why? Because rich detailed profiles are the point of a dating app seeking to be more long-term-relationship focused. Other activation metrics might be if they view ten profiles, or if they manage to get a single match. You want them to hit a point of user delight.

Engagement metrics: Some example engagement metrics can be: how many matches were formed per user, how many matches were made overall, swipes done per user, and real life meetings (this could be indicated by a phone number swap). This would reflect if they were able to foster actual meaningful connections. Retention metrics could be an off-shoot of these core engagement metrics, like what percentage of users swipe on the app after 28 days of signing up, or, of the users who managed to get a match last month, what percentage of them managed to get a match this month?

Revenue metrics: Revenue could be measured by the number of paid memberships or upgrades bought. If Facebook Dating is completely free but ad-supported, then revenue from ad impressions could be measured per user.

However, paid memberships or ad revenue isn't a priority on launch for a company like Facebook, which is primarily trying to nail product-market-fit first. Facebook tends to take a long-term approach to monetization. Their biggest worries are feature adoption and user retention; they know they can easily toss in ads to support the product when needed.

4-Step Framework for Diagnosing Metric Changes

The second most common product interview question that data scientists face is one of diagnosing metric changes. For example, you might be asked: "Instagram's average number of comments per post is declining — how would you troubleshoot this?"

Step 1: Scope Out the Metric Change

Before even jumping into a solution, you need to clarify and gather context.

Questions to ask:

- **Metric Definition Nuance:** What does the metric in question mean? Are we dealing with proportional metrics? If so, by isolating which part of the ratio changed — the numerator or the denominator — you're able to offer a more targeted hypothesis for what is driving the metric change.
- **Importance:** Is this metric actually consequential to the team? With thousands of metrics being tracked at a company like Facebook, there's constant fluctuation in all parts of the business, but not all changes are equal or relevant.
- **Time frame:** Is this a singular occurrence or an ongoing issue? A sudden change or an ongoing trend? At what granularity (over a day, a week, a month) has the trend occurred?
- **Magnitude:** How big is the change, in both relative and absolute terms? Are we comparing this change to last week or on a year-over-year basis?

Say that the interviewer tells you that the number of posts has been increasing on the platform. The number of comments being made on the platform has also been increasing, but at a lesser rate, causing the average number of comments per post to be decreasing. This has been a slow decline over the last 6 months, and today's average comments per post is down 10% from this time last year. This trend concerns leadership because people engaging with posts in the comments section is an important part of Instagram, and helps the product be more interactive rather than consumption oriented.

Step 2: Hypothesize Contributing Factors

Now that you know what the metric change entails, it's time to start brainstorming possible issues that could have occurred to cause the metric to change. Generally, contributing factors fall into four different buckets:

- ***Accidental Changes:*** Is the metric drop even real? Were there any data generation processes or bugs with instrumentation and logging that caused a change that isn't actually reflective of user behavior?
- ***Natural Changes:*** Is the change simply due to seasonality? Could the day of the week, or the fact that it's a holiday, or a change in weather cause the product changes you are seeing?
- ***Internal Changes:*** Issues like new feature launches, bug fixes, intentional product changes, or new marketing campaigns going live can cause metrics to change.
- ***External Changes:*** Competitors launching new products, or more macro events such as a pandemic or a recession, can cause shifts in user behavior.

One good way to brainstorm factors, especially internal ones, is to walk up the product funnel. You can do this by starting with the feature at hand (local) and working your way upstream to broader issues that could affect the metric. For example, in the Instagram case, we can look at the following:

- Were there any UI changes to how users can make comments? Maybe the comment composer UI got less emphasis? Or maybe there is stronger comment moderation in place, leading to more automatically deleted comments?
- Were there any changes to how users can give feedback on a post (like, comment, or share)? Maybe the UI changed so that liking or sharing a post to your story got more emphasis than commenting on a post, leading to cannibalization?
- Were there any changes to the types of posts in the feed? Maybe there are more ads with comments turned off? Or maybe there are more reels, which are more consumption based, rather than things people comment on. Maybe the ranking model changed, favoring posts which are more likely to be liked and shared, rather than commented on?
- Were there any changes to the feed itself? Maybe comments and posts are both being cannibalized by stories, reels, or Instagram Explore?

As you walk backwards through the product funnel, listing a few of the likely culprits behind the metric change, the interviewer may stop you and ask you how you'd validate each hypothesis. Otherwise, stop yourself – it's easy to list off 20 potential reasons for why something happened, but you don't want to overdo it. The meat of these types of problems is explaining what metrics you'd look at to validate each factor, not how many you can come up with.

Step 3: Validate Each Factor

Validating hypotheses means slicing and dicing data into many different segments, hunting for insights that validate or disprove your hypothesis. For example, you could slice along user demographics

(i.e., age, gender, location, language, device type) to see if any of these factors correlate to a trend of fewer comments per post.

You should also look at upstream metrics. This is where the product funnel approach we mentioned earlier becomes applicable, where you continuously zoom out until you find something that can explain the metric change. For the Instagram example, metrics to look at include:

- Number of views on the comment composer or comments sections of a post, to understand if there is less emphasis on commenting.
- Ratio of likes to comments per post, and ratio of shares to comments per post, to understand if this issue is specific to comments or a more general issue with post engagement.
- Amount of engagement per post relative to trends in engagement per story and engagement per reel to see if the issue is Instagram feed losing engagement to other features within the app.

Generally, an interviewer isn't expecting you to ramble on about every dimension you'd cut or every metric you'd check. By prioritizing which hypotheses are most likely based on your product sense and product research, you can narrow down the space and focus the conversation on the most likely culprits.

Step 4: Classify Each Factor

After you've explained how you'd validate each potential factor, the interviewer will often share the results of your hypothetical data analysis. Based on this information, the next step would be to bucket each of the hypotheses into the following categories:

- **Root cause:** The root cause of the metric change
- **Contributing factor:** While not the root cause, still contributing to the root cause
- **Correlated result:** Factors that are symptoms of the root cause but not a contributing factor
- **Unrelated factor:** Factors that are unrelated to the metric change

For the Instagram example, you find that overall activity on Instagram's feed looks normal; people are still viewing posts, and liking and sharing posts at the same rate. This tells us the issue is localized to comments — not a more general engagement or Instagram feed issue.

From slicing comments per post by account age, you notice that posts made from newer accounts are experiencing a sharper decline in the average number of comments than posts made from older accounts. Running cohort analysis, based on when a poster joined Instagram, you are able to confirm that about 6 months ago, the average number of comments made per post by an old user starts to diverge from a new user. This might give you intuition that something changed for new users about 6 months ago. From auditing the product and checking in with the PMs of the Instagram onboarding team, you find out that there was an interstitial added by the Trust & Safety Team that makes new users aware they can turn off comments on a post. This feature has led to some percentage of posts simply having no comments on them at all. By removing posts with comments turned off from the analysis, you realize there was no decline in the number of comments per post. Boom! You found the root cause for the metric change!

Assessing Metric Trade-Offs

Lauded economist Thomas Sowell insisted, "There are no solutions, only trade-offs." As a data scientist, you'll be asked to weigh in on tough business decisions, where there is no clearly right answer. That's why, to assess your judgement, you might get asked a question like "We tested a new feature that shows more ads on LinkedIn. It increases revenue by 1% but hurts time spent by 3% --- should we ship it?"

The steps to approach a metric trade-off problem are very similar to the steps in the frameworks for defining a metric and troubleshooting a metric change. First, to reason about a trade-off, you'd need to know more details about what the metrics in question actually mean. Thus, you follow the first step of the metric troubleshooting framework, where you clarify what the metric definitions are (except now you have two metrics to ask clarifying questions about instead of one).

The second major aspect of solving this type of problem is understanding the product and business goal. By applying your own research into the company, along with your intuition, and then asking smart clarifying questions, you can determine which metrics are more important than others. By doing so, you can reason on how to proceed with the trade-off. After collecting more information about the trade-off, next steps usually are either to

- revert the feature change since the trade-off is not acceptable,
- minimize the trade-off's impact by brainstorming new product interventions, or
- accept the metric trade-off since it's justified.

It can be hard to give a definitive recommendation on what to do, because, in reality, you'd be solving this collaboratively with other stakeholders. However, since this is an interview, the interviewer isn't looking for a specific correct answer, but more your thought process. As such, it's okay to explain in what scenario or what more information you'd need to make each of the three different recommendations.

For the LinkedIn example, you could mention that depending on how hard or easy it was to get a 1% revenue gain or a 3% engagement gain from other features, you'd know what to do. This is reasonable, since maybe 1% revenue gains are easy to achieve through more sophisticated means (via better targeted ads), whereas a 3% engagement drop is very hard to recover from and could unwind progress in other strategic areas for the business. You could also mention how maybe the decrease in time spent is mostly from users who are getting poor quality ads, and that there should be product features like "hide this ad" or "block this advertiser" that can minimize the engagement drop while still maintaining the revenue gain.

A/B Testing & Experimental Design

While the mathematical underpinnings of A/B testing were discussed in "Chapter 6: Statistics," in this chapter we dive into the nuances of real-world A/B testing, because it's brought up in many product data science interviews. Usually, at the end of a success metrics definition problem, you'll be asked, "How would you test this new feature?" Interviewers will ask you to walk through the full experimental design setup for a hypothetical test. Often, interviewers steer the conversation towards one of the many practical testing pitfalls you may face.

Experimental Design Setup Overview

When you're faced with a general A/B testing question like "How would you A/B test a 1-click job-apply feature on LinkedIn?" it can be challenging to not ramble aimlessly. To keep your answer focused, make sure to address the four main steps of setting up an experiment:

- I. Pick a Metric to Test:** While you'll be tracking a whole host of core and guardrail metrics, narrow each experiment down to a few key metrics that capture the essence of the goal of testing the feature change. Don't cheat by cherry-picking a metric post-hoc based on whatever ended up being statistically significant!

2. **Define Thresholds:** Decide on a particular statistical significance level (alpha) which is generally set to 0.05, as well as a power threshold (1 — beta), which is generally set to 0.8. The value for the power is dependent on the minimal detectable effect (MDE) and is usually set by consulting with stakeholders. That is, we can calculate the minimal detectable effect (MDE) at x% power (for example 0.8 MDE for 80% power) for any given sample size and sample variance.
3. **Decide on Sample Size & Experiment Length:** Based on the MDE and power, along with the metric variance, one can calculate the required sample size needed for the test. Using this required sample size, the length of the experiment can be determined based on the daily traffic of the feature in question. A good rule of thumb, though, is to run a test for at least two weeks, to account for day-of-week effects typical in most consumer products.
4. **Assign Groups:** When deciding the control group and treatment group, we want to randomize these groups sufficiently; otherwise, there will be confounding variables down the line. At large companies, this consideration is often abstracted away for you by the A/B testing infrastructure.

Real-World A/B Testing Considerations

Let's face it: in the messy real world, product and business constraints can get in the way of proper experimental design. As such, data science interviews often touch on the A/B testing pitfalls you're likely to encounter in practice, and how you'd guard against them. Unless you're a seasoned product data scientist with many battle scars from A/B tests gone bad, pay careful attention to the A/B testing nuances below.

When Not to A/B Test

Sure, A/B testing is essential to businesses like Facebook and Amazon. But right off the bat, we need to acknowledge that A/B testing isn't always the correct answer to every product and business problem that arises. As 16th century British playwright and data scientist William Shakespeare once said, "To A/B test, or not to A/B test, that is the question."

Some scenarios where A/B tests typically shouldn't be run:

- **Lack of infrastructure:** Having the data engineering infrastructure needed to reliably test isn't a trivial matter; keep this in mind when interviewing at smaller companies that might not have the dedicated resources to perform complex A/B tests.
- **Lack of impact:** Don't test things that don't matter. Try to size up the opportunity --- if the test pans out, how much impact on the business would you expect? Is this benefit much larger than the time and engineering resources you'd have to spend on the test?
- **Lack of traffic:** Without enough people using a feature or performing a certain action, it can be difficult to make a statistically sound conclusion within a reasonable time frame.
- **Lack of conviction:** For high-traffic and high-value features like the Amazon "Buy Now" button, it might be worth trying out 60 variants of different copy, color, size, and iconography. But, at some level, conviction and intuition behind why a variant would be the winning option is crucial --- especially if you are operating on a much smaller scale than Amazon. While a random monkey-throwing-darts approach might work for stock-market investing, it won't cut it for making great products!
- **Lack of isolation:** How would you A/B test a logo? It's not easy to have a "control" group, since these changes make the headlines and get rolled out to every user. In these cases, qualitative methods like customer interviews and focus groups can work better.

In cases where A/B testing is not useful, we can do the following:

- Conduct user experience research via focus groups and surveys to understand what options are better.
- Analyze user activity logs to get a better sense of what option is a better fit.
- Make the product change, but then run a retrospective analysis by looking at historical data to see if the metric that we are interested in responds as we expect.

Dealing with Non-Normality

As mentioned earlier, in the statistics chapter, A/B tests are just dressed up Z and t-tests. These statistical tests assume that the distribution of the random variable of interest (the metric we are testing for) is normally distributed. At larger tech companies, this assumption tends to hold, thanks to plentiful user data and the Central Limit Theorem. But what happens if this isn't the case for some reason?

Several methods exist to deal with non-normality of the test metric:

- **Bootstrapping:** The process of generating extra samples randomly for each variant, and then averaging results at the end, in order to invoke the Central Limit Theorem.
- **Running alternative tests:** The Wilcoxon rank-sum test is a popular alternative to the t-test and does not assume normality.
- **Gathering more data:** With budget and time constraints permitting, collecting more data can give you more confidence that what is being measured is more representative of the true effect on the population at hand.

Dealing with Multiple Tests Simultaneously

Recall the multiple testing problem we covered in the statistics chapter — if you run 100 A/B tests at the same time, by pure chance, you're bound to have some test succeed with a statistically significant result. In interviews, you'll be expected to have a high-level understanding of why this problem exists and how to deal with it.

One way to account for the multiple testing problem is to use Bonferroni Correction, which adjusts the significance level required based on the total number of tests running. Alternatively, you can control the false discovery rate (FDR) or the familywise error rate (FWER). Recall from the ML chapter false positives (FP) and true positives (TP). The FDR is equal to $FP / (FP + TP)$ and, hence, is the rate of type I errors during multiple testing. In a similar vein, FWER is simply the probability of making one or more type I errors when performing multiple tests.

While statistical approaches are helpful, the reality of the situation is that experiments will unfortunately always interact to a degree. Often, the best you can do when faced with surprising or weird results, is to dig into the active experiments and see if anything could have impacted the primary experiment.

Dealing with Network Effects

Generally, it is assumed that during an A/B test, each user is selected to either the control or treatment group at random, that each user is independent, and there is no interference between the control and treatment group. However, this condition doesn't always hold in practice.

Consider a social network like Facebook — the actions of users are likely impacted by that of those around them (network effect). Therefore, it will likely be the case that the behaviors of those in the control group are being influenced by behaviors of those in the treatment group; hence, the true effect is not exactly as is stated.

As a concrete example, say Facebook is testing out a new kind of Facebook Live, which ends up being super entertaining. The test group might see a large increase in engagement metrics due to the new feature, but their increased engagement on the platform overall may drive them to also interact with their friends in the control group more. This spillover can lead to increased engagement for the control group, which causes the overall positive effect of the experiment to be understated because of the control group contamination.

One possible way to control for network effects is to create clusters of similar or connected people and divide these sub-networks into control and treatment groups for better isolation. Mathematically, this separation can be done through various graph partitioning methods (for example, normalized cuts) that place users into various groups based on social interactions.

Dealing with Novelty Effects

A/B tests may have an exaggerated initial effect, due to the novelty effect, where a new feature attracts social media attention and PR hype, causing inquisitive users to rush to check out the new changes. This flock of curious users leads to metrics like time spent and engagement to jump spuriously high. For example, when Facebook first launched emoji reactions like “haha” and “angry,” post engagement massively spiked due to curious users trying the feature out. Eventually, though, reaction rates came back down and stabilized at a new, slightly higher, baseline rate.

In the opposite direction, there could be a primary effect, where users are fixed on what is familiar and have an aversion to changes in general. For example, when Facebook originally launched News Feed in 2006, it was reviled by its then 8-million student users. To get a sense of the backlash, consider this — a Facebook Group called “Students against Facebook News Feed” grew to 750k members in just two days. Ten percent of the entire user base was upset enough to join a group boycotting the news feed. We know how that experiment turned out!

To detect primary and novelty effects, you could analyze the test results for new users only — if there is a statistically significant change there, but not for the old users, novelty effects are likely at play. Similarly, to guard against the novelty effect, you can always just run the A/B test on new users. However, this opens up its own can of worms, if new users aren’t similar to existing users. For example, at Facebook, back when Nick was on the New Person Experience team, most new people came from developing countries like India, Indonesia, and Nigeria. Usage patterns didn’t approximate well to the average tenured user of Facebook, who resided in a more developed country.

Nuances of Taking Action on A/B Test Results

If you ran an A/B test, managed to avoid all the common experimental design pitfalls, got a positive result, and $p < 0.05$, you’d definitely launch it, right? Maybe.

Just because a test reaches statistical significance doesn’t mean it’s automatically shipped. At large tech companies that run experiments on hundreds of millions of users, it’s easy for small differences to become detectable. So even with a small p-value, it’s crucial to assess the effect size before shipping the change.

It’s not just the direct effect size to consider — what happened to the counter metrics and guardrail metrics you’d set up? For example, if revenue went up, but retention went down, it’s not obvious whether to ship it or not. As we mentioned earlier, it’s best to confer with product and business stakeholders when making these metric trade-offs.

Another consideration to factor in is that any time you ship a change, there is human labor cost associated with it. There’s always a cost of properly deploying and then supporting the feature change

— for example, increased customer support tickets from users who are confused by a new experience. And then there's the chance of hidden bugs in the new feature you're launching that didn't get caught in the test. You need to make sure the impact from the A/B test justifies launching the new variant.

Launch with A/B Test Holdouts

Suppose you ran a successful A/B test and stakeholders have given you the green light to roll out the new feature. Typically, even after launch, there remains a small group of users — usually just a few percent — who don't receive the new experience. This group is known as the A/B test holdout. Because A/B tests are typically run over a shorter time period, holdouts help you quantify the long-term lift from shipping features. This makes identifying potential novelty effects, where metrics tend back towards their original baseline over time, much easier.

For a practical example of how holdouts are used, when Nick was a part of Facebook's Growth Engineering division, his team would create a shared holdout group every quarter. By having the entire team's launched features not affect a tiny portion of users, the Head of Growth could easily measure the team's combined quarterly output, which was useful come performance review and promotion time.

Note that not every A/B test merits a holdout. For bug fixes or very sensitive changes, it can be better to launch the feature to the entire user base. For example, when Kevin was working on Facebook Groups, implementing a holdout for a new model that flagged child trafficking would mean that some small number of holdout users would still see such content. In these scenarios, after getting a signal that the new model was helpful without too much downside risk, we'd say, "F*ck it, ship it!" and launch the model to 100% of users without any holdouts.

Product Questions

- 10.1. Facebook: Imagine the social graphs for both Facebook and Twitter. How do they differ? What metrics would you use to measure how these social graphs differ?
- 10.2. Uber: Why does surge pricing exist? What metrics would you track to ensure that surge pricing was working effectively?
- 10.3. Airbnb: What factors might make A/B testing metrics on the Airbnb platform difficult?
- 10.4. Google: We currently pay the Mozilla foundation 9 figures per year for Google to be the default search engine on Firefox. The deal is being renegotiated, and Mozilla is now asking for twice the money. Should we take the deal? How would you estimate the upper bound on what Google should be willing to pay?
- 10.5. LinkedIn: Assume you were working on LinkedIn's Feed. What metrics would you use to track engagement? What product ideas do you have to improve these engagement metrics?
- 10.6. Lyft: Your team is trying to figure out whether a new rider app with extra UI features would increase the number of rides taken. For an A/B test, how would you split users and ensure that your tests have balanced groups?
- 10.7. Amazon: If you were to plot the average revenue per seller on the Amazon marketplace, what would the shape of the distribution look like?
- 10.8. Facebook: Besides posts Facebook is legally obligated to remove, what other types of posts should Facebook take down? What features would you use to identify these posts? What are the trade-offs that need to be considered when removing these posts?

- 10.9. Amazon: The Amazon books team finds that books with more complete author profiles sell more. A team implements a feature which scrapes Wikipedia and Goodreads to automatically fill in more information about authors, hoping to see an improvement in sales. However, sales don't change — why might this be?
- 10.10. Snapchat: Let's say Snapchat saw an overall 5% decrease in daily active users, a trend that had been consistent over the week. How would you go about determining the root cause of this?
- 10.11. Pinterest: Say you ship a new search ranking algorithm on Pinterest. What metrics would you use to measure the impact of this change?
- 10.12. Netflix: Say a given category, such as sci-fi TV shows, has less total watch time, compared to other similar categories. What metrics would you look into to determine if the problem is that people aren't interested in that category of content (demand problem), or if the category has interest but the content is bad (supply problem)?
- 10.13. Apple: Say you have data on millions of Apple customers and their purchases made at physical Apple retail stores. How could customer segmentation analysis increase a store's sales performance? What techniques would you use to segment brick & mortar customers into different groups?
- 10.14. Facebook: If 70% of Facebook users on iOS also use Instagram, but only 50% of Facebook users on Android also use Instagram, how would you go about identifying the underlying reasons for this discrepancy in usage?
- 10.15. Capital One: How would you assess the stickiness of the Capital One Quicksilver credit card?
- 10.16. Google: Say you worked on YouTube Premium, which is an ad-free version of YouTube bundled with YouTube Music — a music streaming service. You're launching the product in a few new countries — how would you determine pricing for each country?
- 10.17. Twitter: Should Twitter add Facebook-style emoji reactions (love, haha, sad, angry, etc.) to tweets?
- 10.18. Slack: What metrics would you use to measure user engagement at Slack? How would you be able to tell early whether or not user engagement was declining?

Product Solutions

Solution #10.1

A bad answer glazes past the nuances of the social graph and jumps straight into defining metrics. For clarity — not just for you, dear reader, but also for the hypothetical interviewer asking this question, we'll structure our answer into three steps.

Step 1: Explaining User Behavior on Each Platform

Before explaining how the social graphs of Facebook and Twitter differ, it's crucial to first consider how each platform's average user interacts with their respective platform. Facebook is mostly about friendships, and so two users on the same social graph are mutual friends. Twitter, on the other hand, is more focused on followership, where one user typically follows another (who is usually an influential figure) without getting a followback. Thus, Twitter likely has a small number of people with very large followings, whereas, on Facebook, that pattern appears less often.

Step 2: Describing the Social Graph and Its Differences Between Facebook and Twitter

Modeled as a graph, let's say each user is represented as a node, and the edge linking two nodes denotes a relationship (typically, friendship on Facebook and fellowship on Twitter) between the two users whose nodes the edge connects. Most nodes on Twitter would have low degrees, but a small number of nodes (those of influential people) would have very high degrees, resulting in a “hub-and-spoke” social graph for that platform.

Step 3: Defining Metrics to Measure the Social Graph

One way to quantify the difference between the two platforms' social graphs is by looking at the distributions of friendships/followerships represented by the social graphs of each platform's typical users. Because a typical node's degrees — that is, the number of connections it has to other nodes — should capture the difference in these platforms' social graphs, one concrete metric would thus be the average degree among all nodes on each platform. Alternatively, to obtain an even more detailed understanding of the differences between the two social networks, we could construct box-and-whisker plots of the platforms' degrees among all their respective nodes.

Still another way of looking at the two graphs would be to check the distribution of degrees across all nodes in each platform's network. In all likelihood, the Twitter distribution would show a greater amount of right skewness than that of Facebook. Metrics that quantify a distribution's skewness or kurtosis could thus be used to describe the difference between the two platforms' degree distributions and, hence, social graphs.

Solution #10.2:

For any metrics definition question, it's important to first explain the business goal of the product or feature and then explain the related stakeholders, before ultimately landing on good metrics to measure the success of that feature.

Step 1: Explain Uber's Motivation for Surge Pricing

You don't have to be an econ major to realize that surge pricing is about fixing imbalances between supply and demand. In the case of Uber, such an imbalance could result from either a lack of drivers or an excess number of potential riders. Therefore, surge pricing's goal would be to increase supply by enticing more drivers to use the app through increased pay, and reduce demand by raising prices for riders.

Step 2: Consider Stakeholders Related to Surge Pricing

A nuanced answer would consider the various stakeholders involved in surge pricing, beyond just the immediately obvious drivers and riders. For example, a good candidate would mention associated business functions within Uber that could be affected by the surge pricing algorithm not working effectively.

Step 3: Define Metrics & Counter Metrics for Surge Pricing

Surge-specific metrics are the duration of the surge, the surge pricing multiplier, and the number of riders and drivers in the affected area. We should also track the following metrics during surge periods: number of rides taken, number of rides cancelled, total revenue made for both Uber and Uber's drivers, total profit made by both Uber and Uber's drivers, and the average ride wait time. These are all standard metrics, but critical to monitor to ensure the business is healthy during surge periods.

In addition, topline metrics like user's lifetime value (LTV), driver retention, rides taken, daily active riders, and drivers should also be tracked, so that we can be sure surge pricing isn't having adverse impacts on the business overall.

As with any good metrics definition question, a discussion on counter metrics is important. Even if the surge pricing is bringing in extra money, one counter metric to implement would be net promoter score (NPS). Surge pricing can annoy users, for whom frequently fluctuating sky-high prices can be a source of frustration. And then there's the potential for mistakes, or users in a less-than-sober state accidentally making a purchase (we authors can neither confirm nor deny that \$158 Uber from San Francisco to Palo Alto after a fun night out). It can even be a PR risk. Like clockwork, every New Year's, there's a news story about someone getting drunk and taking an \$800 Uber by accident. Between bad PR, frustrated users, and potentially increased support tickets, some metric to make sure it's a quality program is key.

Solution #10.3:

A good answer would demonstrate not just thorough A/B testing knowledge, but some understanding of the Airbnb product. Also, to demonstrate your problem-solving attitude, it can be a wise move to not just mention the difficulties, but briefly also mention techniques to deal with these A/B testing problems. Finally, to earn bonus points on this problem, remember to relate your own A/B testing war stories if you think it would also be relevant to Airbnb. By coloring your answer with your own experience, you're demonstrating your time spent in the trenches, which can help separate you out from the more green candidates.

Issue #1: Complexity of User Flow

Testing Airbnb platform metrics would be difficult because of the complexity of the company's booking process, which starts with a user search and often requires user-host communication before a booking can be finalized. Alternatively, a user can sometimes book a rental without having to contact the property host at all. Also, adding to the complexity of conducting A/B tests on Airbnb platform metrics, booking flows frequently depend on factors outside of Airbnb's control, such as host responsiveness to messages left by prospective renters. Therefore, since a booking can be instantaneous or a long, drawn-out process, timeboxing an experiment could be difficult. Because of these issues, any data generated by tests on the booking process would most likely be quite noisy.

To mitigate these issues, we want to make sure we are looking at the correct non-intermediary metrics. For example, there could be a few steps in between searching and booking, but the searching to booking conversion rate should be the main metric. Additionally, we want to employ best practices on managing the data generation and collection process (logging and other downstream event collection).

Issue #2: User Bucketing Due to Multiple People & Devices in Booking Flow

A second source of complexity arises because planning a vacation often involves multiple people. Since even a single person could employ multiple devices during the booking process, it could involve multiple and discontinuous uses of the Airbnb platform from different IP addresses and, to further complicate things, occur over an extended time period. Ideally, there would be a clear one-to-one user-to-device mapping, and, also ideally, the booking process would occur nearly instantaneously. In that case, testing would be easy since the variables being tested for a specific user (e.g., demographics, booking details, time needed to book, and so on) could be clearly identified and then measured. However, different members of a user group (e.g., a family) could be involved

in different parts of the booking process, or a single user could employ different devices during the process and might not be logged in to any of these devices during some parts of the process. Then, the correct user profiles of each would need to be determined during each contact in order to correctly identify them and, consequently, the correct A/B test group(s) to which they belonged.

There are a number of ways to address these various issues. For example, doing extra checks with internal and potentially external datasets (which various vendors can provide) could help address the device-mapping issue. Note that data cleaning can have nuances — for example, in the cases of multiple devices, it may be the case that the user deletes their cookies, which is a useful signal in itself that may be easily imputed or predicted. In such cases, where there is missing user information, the best approach is to see if you can use other variables to try and predict the missing information.

Issue #3: Long Time Horizon for Measuring Success

Lastly, successful consumption of a use of Airbnb's services — a happy stay at an Airbnb listing — happens over a much longer time horizon than, for instance, use of social media (where consumer enjoyment of the service is instantaneous). This delay makes it difficult to accurately measure the influence of various features of Airbnb's service through calculation of various success metrics, which can be done only much later. Plus, these are low-frequency events — it's hard to know if there was a statistically significant increase in bookings if the majority of users don't even make a booking in a given month.

As an example, consider measuring longer-term metrics such as user retention or customer lifetime value. Since A/B tests cannot be run for many months, we need to find a shorter-term proxy for such longer-term metrics. A machine learning approach works well here — for example, using various features to predict retention or customer lifetime value, and choosing any of the important features correlated with the target metric that can be measured on a shorter-term basis. Then, on the A/B test, we want to simply see the expected moves on these important features.

Solution #10.4:

At first, it may seem that Google's search market share goes unchallenged. You might think, "If we don't make a deal with Firefox, users would still default to Google on their own, because what are they going to do, *Ask Jeeves?*" However, to answer this question well, it's important to first fully explain the business motivations for why Google wants to remain Firefox's default browser. Based on these business considerations, we can then specify the metrics Google should use to price the deal.

Step 1: Explaining Google's Immediate Motivation to Be Firefox's Default

Google's advertising revenue, most of which comes from search ads, makes up more than 80% of Alphabet's revenue. Clearly there's a lot of money at stake when it comes to search. However, Google's motivations for closing a deal go beyond the immediate profit that's at risk.

For example, Google likely has a business goal of beating competitor search engines and making sure they stay beat. That's because some percentage of users probably don't care about what search engine they use and simply stick with the default one. Plus, Firefox defaulting to a competitor like Bing just might be the spark that Microsoft needs to invest more heavily in challenging Google's monopoly on search. In the case that Firefox defaulted to DuckDuckGo, there would be additional brand ramifications to deal with too. Because Firefox positions itself as a more privacy-aware browser, and DuckDuckGo's brand is built on protecting a searcher's privacy, a vote of confidence by Firefox in DuckDuckGo could bring Google privacy concerns to the forefront, fueling a negative news cycle and ultimately hurting Google's brand reputation.

Google's not just worried about beating direct competitors — vertical search engines such as Amazon, Expedia, and Yelp compete heavily with Google on specialized searches. Google boosts its own ancillary services, like Google Shopping and Google Maps, in search results. That means if Firefox were to default to a competitor, Google Shopping couldn't benefit from its top-of-page position in Google, which would cause them to cede market share to Amazon.

Similarly, if Firefox switched from Google to Bing, searching for a nearby restaurant wouldn't result in a Google Maps listing, but instead, might take a user to a Yelp page. Clearly, a whole ecosystem of products and business lines would be adversely affected if Google isn't able to be the default search engine on Firefox.

Step 2: Explaining Google's Secondary Benefits from Being Firefox's Default

There's also network effects to consider as well. Google's search relevance algorithm takes into account how users interact with their search results. This means the more people who use Google Search, the better the product becomes, creating a positive feedback loop of user engagement and product improvement. Furthermore, these network effects extend to products downstream of Google Search, like Google Reviews and Google Maps. More people searching on Google means more people reviewing businesses on Google, which helps Google Maps compete better against Yelp and Apple Maps in the lucrative local search market.

In considering the whole ecosystem of products that Google Search leads to, along with the multitude of business Google competes with, it's clear Google's motivations to be the default search engine on Firefox go far beyond the immediate search ads revenue at stake.

Step 3: Metrics Used to Inform Google's Willingness to Pay

A lazy way of setting the price is by assuming what Google paid previously to Firefox (~\$450 million in 2020) was fair, and adjusting it slightly according to the change in Firefox's install base. A similar approach could be to price this deal relative to the deal Google has with Apple, where it pays ~\$10 billion a year to be Safari's default search engine, and scaling the price to Firefox's market share. Depending on the interviewer, these answers can be seen either as clever or missing the point of the problem, so tread carefully!

For an answer based on first principles, you could look at the amount of search ad revenue Google gains from all Firefox users. If this segment completely stopped searching on Google, how much revenue would be lost? This could be one way to price the deal, but it assumes the worst-case scenario: that everyone wouldn't use Google anymore if it wasn't the default. To make the estimate more realistic, you could look at other browsers where Google isn't the default and see what percentage end up using Google. This way, you can get a more reasonable estimate of how much revenue you'd stand to lose, and then determine the price based on this number.

Instead of considering direct search ad revenue, we could also do a similar analysis, except base our bid on the total revenue generated from Firefox users. This number would account for all the downstream ways Google makes revenue from a Firefox user, and thus better account for the second order effects of weakened market share amongst Firefox users.

Solution #10.5:

Before jumping into defining metrics and brainstorming product improvements (like forcing all LinkedIn users to follow linkedin.com/in/nipun-singh/ and linkedin.com/in/kevin-huo/ for career advice), it's best to start by explaining the goal of LinkedIn's Feed.

Step #1: Explain Why LinkedIn's Feed Exists

LinkedIn's mission is to connect the world's professionals to enable them to be more productive and successful in their careers. The newsfeed helps fulfill this mission by helping users keep tabs on their professional network, stay up-to-date with industry news, connect with new people through engaging content, and more.

From a business perspective, newsfeeds (not just at LinkedIn, but other social media companies as well) tend to be very engaging products. For LinkedIn, the feed ensures people keep checking the product often, which is crucial, since without the feed, the product doesn't hold much utility for users not actively job hunting or networking. By helping to keep users on the platform, LinkedIn is able to make more money through displaying ads and sponsored jobs.

Step #2: How LinkedIn Can Measure Feed's Engagement

For a product as expansive and critical as its feed, LinkedIn needs to track a whole host of metrics. A few top-level engagement metrics include daily active users, weekly active users, and monthly active users on Feed. You could also track L7 and L28 (how many days in a week or month do users check the feed). To add a notion of duration to these visits, you can also track average user session time on Feed, and average daily, weekly, and monthly time spent on the feed.

Having users log on frequently and not engage with anything doesn't help LinkedIn's product goal with feed. To measure the depth and quality of engagement, we can track important user actions taken on feed. You could track the number of posts seen, posts liked, posts commented on, and posts shared per month. To make it easier to report on, you could combine all the activity into a single score. However, not all post engagement is equal. By weighting the value of a post *view* vs. *post like* vs. *post comment* differently, you can more accurately capture post engagement activity using just one metric. You could also make a similar metric to measure content creation, which incorporates the number of posts made, comments made, and posts shared.

Because LinkedIn Feed is so closely tied to a significant source of revenue — ads — you should also separately track engagement on ads in Feed. Metrics like ad impressions and ad clicks in Feed could be bundled under the umbrella of feed engagement.

Lastly, for all of these metrics, we want to make sure we are measuring genuine engagement — not the results of automated spam or scraping bots, which inflate activity metrics.

Step #3: Ways to Improve Engagement on LinkedIn's Feed

As mentioned in the intro section on doing your homework before the interview, hopefully you've used the product a bit and had a chance to look at competitors. That can make tackling product brainstorm questions easier since you've scoped out the competition and can "borrow" ideas, much like Facebook and Instagram love to borrow from Snapchat.

To improve these metrics, LinkedIn would want to incentivize people to stay engaged. Example features that boost engagement include personalizing the News Feed to the user and encouraging people to post more to keep the news feed fresh. Specific ways to achieve these goals would include creating up-to-date ranking models that accurately rank how likely a LinkedIn user is to consume and engage with newsfeed content or adding some new, highly requested reaction type. Enabling new post types, like LinkedIn Live streams or LinkedIn Stories can also aid in keeping the content inventory engaging.

Another way to improve metrics would be to build a model using features that you believe would affect the metric. Generally, this will be a combination of user data (demographics) and event data

(browsing behaviors and session events). Decision trees or random forests can be useful here, as they tend to have high accuracy and also can easily display feature importance. After assessing model outputs, you can determine factors contributing to the target metric and decide on an action plan accordingly.

Each of the features suggested above can be A/B tested against core engagement metrics to determine if they would drive increased engagement on the newsfeed. Since other metrics would likely also be affected (for example, with a large increase in time spent and news accessed, more bad content would also be consumed), it is paramount that such A/B testing be evaluated holistically and potential trade-offs kept in mind. Note that, overall, this process implies that improving metrics improves what you intend to measure (i.e., that metrics drive behavior) rather than the other way around, which can have some consequences. As we painfully learned firsthand running A/B tests at Facebook, sometimes the metrics measured don't accurately reflect true user behavior and sentiment.

Solution #10.6:

A good answer would walk through the basics of user assignment and address basic pitfalls the Lyft rider app may face in A/B testing. A great answer would also mention advanced issues that can occur, like network effects, and how doing geo-based randomization can help keep groups balanced. For bonus points, we also explain ways to quantitatively prove that our groups are, in fact, balanced.

Step 1: A/B Testing Basics for Lyft's Rider App

Since we want to quantify whether and to what extent instituting a change in how Lyft operates (in this case, adopting new UI features) would improve a metric of interest (number of rides taken), the most feasible strategy is to conduct an A/B test, in which users are divided into two groups — one exposed to the change, and one not exposed to it.

However, we can't just arbitrarily split Lyft riders into two groups with one having the new UI and the other having the old version; splitting the data haphazardly in that way could, and most likely would, cause the demographics of one group to differ greatly from that of the other. This would introduce a source of variability in the outcome variable not related to the UI change and would most likely skew the distribution of the dependent variable being measured (i.e., number of rides taken). Therefore, we would have to choose A users and B users so as ideally to balance the groups with respect to such user characteristics as demographics, locations, etc. Employing stratified random sampling would provide the best means of ensuring homogeneity of groups.

Step 2: Accounting for Network Effects with Geo-Assignment

We also need to take into account marketplace dynamics. Consider any location...say New York City. If we give half of the riders the new features and keep half of the riders on the old features, then if the new features *do* help people book more rides, there will be more competition for drivers on the new features (and vice versa if the features are detrimental to conducting more rides). In either direction, the resulting effect is exaggerated due to these marketplace dynamics. Therefore, our best option is to test by using comparable markets (comparable meaning the metrics in aggregate should be similar across both markets).

Step 3: Account for Geo-Assignment Flaws

Every method has its drawbacks — geo-based user assignment included. Assuming that two comparable markets are independent may not be accurate in many cases. Additionally, even if user demographics are relatively balanced between the two markets, there is no guarantee that the users

will always stay comparable and that any metric changes tracked by these pools of users will be comparable forever. Lastly, external events may happen that cause the two markets to diverge in some aspect of comparable distributions (regulatory or political change, certain competitors launch marketing campaigns in particular areas, etc.).

To make sure there weren't any geo-based assignment issues, it's best practice to check on a few baseline metrics that aren't supposed to change by market, and validate that they stayed the same so you can have more confidence in the test.

Solution #10.7:

In statistics class, it's beaten over our heads how many phenomena follow a normal distribution. But in the business world, a great many distributions actually follow the Pareto principle, where 80% of outcomes come from 20% of the causes. Just how 80% of the world's income is earned by the top 20% of people, or how many tech companies have found that 80% of crashes come from 20% of bugs, we'd expect the 80/20 rule to be valid here too.

We'd expect many small sellers doing small amounts of revenue, with a long tail of a few power sellers with enormous amounts of revenue. Hence, we'd expect the distribution to be right skewed.

Solution #10.8:

This open-ended discussion on what makes for "bad" content is one that can test your product, business, and even PR savvy. There's no right answer — people debate this question everywhere, from Facebook's headquarters, courtrooms, and even the Senate floor. As long as you're able to brainstorm content removal features well and convey the many nuances of taking down posts, you'll be golden.

Step 1: Brainstorm What Posts Should Be Taken Down

Besides what Facebook is legally obligated to take down (exploitative photos of minors, copyright and IP violations, etc.) other types of content Facebook could potentially take down:

- **Explicit Content:** Nudity, sexually suggestive imagery, self-harm, excessive violence
- **Hate Speech:** Death threats, posts that incite violence, bullying, doxxing
- **Misinformation:** Conspiracy theories, fake news about vaccines or elections
- **Content from Bad Actors:** Everything from a terrorist organization or criminal organization
- **Regulated Goods:** Posts that promote selling or trading firearms, narcotics, and human organs
- **Scams:** Ponzi schemes, fake fundraisers or charities, posts from people who stole someone's identity and are trying to now solicit money

We should also mention that there are other types of posts which could possibly be taken down; in some cases, you could avoid this by tacking on a warning below the post, with a link to verified resources that fact-check or debunk the post. This way, you can reduce harm on the platform while still allowing for freedom of expression.

Step 2: Propose Features to Find Bad Content

In classifying content, features to be considered would include the type of content, the entity posting it (i.e., who posted it), and the context (when/where the post occurred). Here are examples of features demonstrating each of these aspects:

- **Content:** Contains inappropriate language (curse words), nudity (in photos), hate speech, or sensitive keywords (e.g., “vaccine,” “election fraud”).
- **Entity:** Posted by a suspected fake account or bot; an entity with a history of posts taken down in the past; an entity with an unverified phone number or email address; an entity connected to other bad actors, etc. Since it is likely that people rarely just commit one act of harm, they may masquerade as various accounts with similar behavior. Thus, it is important to keep track of all the detailed user information (IP, device ID, etc.) to try and triangulate such users.
- **Context:** How much spam the group or feed it was posted in has, the amount of “bad actors” within the group or feed posted, etc. Often rings form online where multiple people organize and engage in harmful activity.

We should also work with product operations and manual review teams to understand what types of bad posts they are seeing and the heuristics they use to find these bad posts, as this human intuition can help us generate new features.

Step 3: Explain Trade-Offs of Taking Down Content

As with any kind of classification problem under uncertainty, there's false positives and false negatives. Classifying a post as harmful and taking it down, when in fact it was benign, can confuse and anger users. They might be perplexed as to why their post was removed, which could lead them to post less of that type of content in the future. They can also feel like their voice doesn't matter in the purported community Facebook is building, and might deactivate or cancel their Facebook account in protest of censorship. Sometimes, these news stories even work their way to Capitol Hill, with congressmen and senators calling for regulation or breaking up the purported monopoly because someone they like posted something innocuous and it got taken down (which is perceived as damning evidence of Facebook bias and censorship).

In the case of false negatives, letting harmful content remain on the platform can have many ill effects. People can be confused or harmed (no, drinking hand sanitizer won't clean the COVID out of your system!). People who don't even see the original misinformation can be affected. Almost always, when something harmful goes viral, there are negative news stories, like “*5 million people saw fake news on Facebook claiming that Drake is NOT the best rapper of the 2010s*” which can cause a PR shitshow for the company. Long term, it can even make the company seem complicit with the misinformation that spreads.

As such, for different bad content types, there can be different sensitivities used, depending on the downside risk of having a false positive or a false negative. Additionally, we will want to tweak the algorithm for sensitive accounts. For example, if a political figure with a large following posted something questionable but allowed, but by accident the algorithm flags it and it gets taken down, Facebook gets tarred and feathered in the press for censorship. For sensitive accounts like news agencies, political figures, or governmental agencies, there likely should be a human in the loop to improve accuracy.

Solution #10.9:

Two words, one equation: Correlation != Causation

Just because more complete author profiles correlate with increased sales doesn't mean it's the cause. Maybe books with more complete author profiles had a more highly reputed publisher fill it out for them, which means they likely also have better designed book covers, and that's why they have better sales.

Solution #10.10:

This is your classic metrics troubleshooting problem. Please raise your right hand, and repeat after me: *I do solemnly swear to stick to the 4-step framework for diagnosing metric changes.*

Step 1: Clarify the Scope of the Metric Change

Always start by asking clarifying questions about the metric in question. What is meant by “active users”? Is this a decrease in daily log-ins, or is it a decrease in usage of a specific feature? Did the metric suddenly drop by 5% one day, or was there a gradual decrease in active users over the week?

Also, is 5% a lot? What are we comparing this drop against — the daily active users this time last week, last month, or last year? How much does this metric typically fluctuate — maybe a 5% drop for a week isn’t large enough or sustained long enough to be a cause for alarm. Or maybe there is seasonality at play — a 5% drop wouldn’t be too surprising if this was the week after Christmas and New Year’s.

For the sake of our solution, we’ll assume that the interviewer tells us this was a decrease in the number of times logged-in users have opened the app, and that there is no seasonality at play. We’ll assume the interviewer says the 5% drop is relative to last week, and that on a weekly basis, the daily active user count tends to be consistent, which is why stakeholders are so worried.

Step 2: Hypothesize Contributing Factors

Here are some potential reasons behind the 5% drop:

- Logging Issues: Data pipelines responsible for logging daily active users broke somehow, which makes it seem like a genuine drop, but it actually isn’t.
- Upstream Issues: There could be a problem upstream of daily logins, like a bug in keeping users logged in or a decrease in push notifications being sent, which is having an effect on app opens.
- Product Changes: Did we change something inside the product, like how snap streaks work, or extend the expiration date of snaps, which is causing users to check the app less frequently?
- External Events: Has some large market experienced a hardship, like a natural disaster or an internet shutdown, which is causing users to not use the app as much?

Step 3: Validate Each Factor

For each of these factors, we can validate our hypothesis by looking at various metrics and talking to teammates:

- Logging Issues: We can check in with the SRE (site reliability engineering) team, data engineering team, and metrics and logging teams, to make sure pipelines are healthy and the drop is genuine.
- Upstream Issues: How is push notification volume looking? How are login and password recovery numbers looking — anything out of the ordinary? Did uninstalls spike in the last week?
- Product Changes: How many snaps were sent, and what was the average open rate? What about the number of messages sent between users and the number of stories posted and viewed — are these down by 5% too, or much more? If it’s a more drastic drop, maybe something within the app broke that is causing users to not check the app as much. You can also directly check for product quality issues by seeing if bug reports or app crashes spiked within the last week.
- External event: Can you segment by market, language, and OS to check to see if this problem is local to any one subgroup, and then research into that area for changes that occurred in the last week?

Step 4: Classify Each Factor

Imagine after going through the above factors, your interviewer tells you that logging is working fine, and that this drop shows up across all markets, all ages, all genders, and all types of users (both new users and tenured users). However, you learn that the drop is 7% for iOS users but only 1% for Android users. Now we're getting somewhere!

You could bucket the iOS users by the app release number they are using, what carrier they are using, and what model of iPhone they are using. Say your interviewer tells you that people running the latest version of the app have a 20% lower chance of being a daily active user compared to the baseline iOS user. From this, you look at other upstream metrics for users on the latest app and compare them to the general iOS population. The interviewer then tells you that user logins and password resets both spiked the day the new app became available.

Now we've found our likely culprit — there must be a bug when upgrading to the newest app release that's accidentally logging out users, who are then forced to log back in or reset their password. Some subset of people probably can't recover their password, and thus drop off from being a DAP.

Solution #10.11:

We first clarify what the new search-ranking algorithm change is, then connect how this algorithm change relates to Pinterest's product and business goals. Once you have done this, you can suggest concrete metrics to measure the impact of the change.

Step 1: Clarify the Product Change

It's important to clarify the scope of the change. Questions to ask include:

- Did this algorithm change have any high-level goals in mind (e.g., prioritizing trending Pins, improving discoverability of niche Pins, increasing the personalization of search results)?
- Did this change involve any UX or UI changes perceptible to the end user, or was it solely a change on the backend?
- Do search results return just as fast for users as before?

Step 2: Explain Why Search Relevance Is Important for Pinterest

Pinterest is a visual discovery engine built for helping users find inspiration for their lives, from fashion to home decor to recipes and more. With billions of Pins on Pinterest, the ability to search for and find the content that sparks a user's imagination is key. At a higher level, to keep Pinterest's product competitive against similar content discovery platforms like Instagram Explore or Houzz, the search, content recommendation, and user personalization algorithms need to be top-notch.

When the discovery engine is working well, users will stay engaged with Pinterest and keep coming back for more. This user engagement and retention is key for Pinterest's ad-supported business model. Plus, with shoppable product pins — Pinterest's push to diversify away from ad revenue and into e-commerce — the ability for users to search for products and find exactly what they were seeking is even more critical to the business than before.

Step 3: Propose Metrics to Quantify a Search Algorithm Change

We could measure the direct amount of engagement the search functionality received. For example, time spent searching or the median number of searches made per user session could be used. While growth in these metrics could be a sign that users like the new search experience, it isn't definitive

proof of a successful algorithm change. For instance, if our search results weren't relevant, a user might perform multiple searches to find what they were looking for. Here, an increase in time spent and searches made actually indicates user frustration.

Thus, a more complete way to capture the impact of the change would be to also look at the downstream effects of a relevant search algorithm. For example, we could measure how often a search leads to a user pinning a search result to their board — a sign that the user found what they were looking for. To go one step further, we can quantify the direct monetary benefits of improved search. Here you could measure the revenue generated from purchases of buyable pins that came up as a search result.

For bonus points, you could also mention evaluation measures for information retrieval systems. If you take a binary approach to the results — for example, is this pin relevant or not — you can use a metric like “precision@10” to understand the percentage of relevant pins amongst the first 10 results. However, this won't take into account each pin's position within search results. If you want to account for the actual degree of relevance for each pin, a metric like normalized discounted cumulative gain (nDCG) measures how close the results are to the best possible result. The technical details of this measure are beyond the scope of this text.

Solution #10.12:

This product interview problem is a hybrid between a root-cause analysis question and a defining success metrics question. We'll first start by connecting the supply vs. demand question to Netflix's bigger business model and product goal. Then, we'll define some metrics we'd want to analyze in order to troubleshoot the root cause of the content supply or demand problem.

Step 1: Why Netflix Cares About Content Supply & Demand

Netflix's mission is to entertain the world. Netflix Studios not only has the power to produce original content, it can directly influence what entertainment hundreds of millions consume.

By deeply understanding what its users want to watch, thanks to the myriad of analytics Netflix collects, Netflix can greenlight new shows that delight niche audiences. Let's face it: you'd never see a show like *Indian Matchmaking* or *Orange is the New Black* on cable TV. As long as Netflix creates high-quality tailored content, its users will continue to engage and retain on the service rather than churn out due to stale inventory, or switch to competitors like Disney+ or Amazon Prime Video.

However, creating original shows is very expensive. You can't just *blindly* do things, like Sandra Bullock in *Bird Box*. Netflix needs to prioritize what types of shows to produce so that every dollar spent brings back many more in the form of increased customer retention and NPS. As such, it's crucial to vet how much demand there is for a show category before investing resources into producing Netflix originals or licensing more media from other studios.

Step 2: Content Supply vs. Demand Metrics to Investigate

Knowing that there is less total viewer watch time devoted to sci-fi TV shows compared to other similar categories doesn't tell you much. Sure, total watch time for sci-fi might be less compared to other categories, but what about on a per-show basis? Maybe there's just fewer available sci-fi TV shows, so even if people like the show, there just isn't enough inventory to support high total watch times compared to a category like comedy or drama, which has many more shows to watch. Knowing that even though the total watch time is low, the watch time per show is high would indicate a supply problem, not a lack-of-interest problem. This would signal that Netflix should create or buy more sci-fi TV shows — *not* drop the few sci-fi shows they do have.

Another way to get a clue on whether it's a user interest or show quality problem is by looking at metrics related to sci-fi TV show recommendations. Are people browsing for titles in this category but just not hitting play? This could indicate there is demand, but nothing catches a user's eye. One step down the funnel, what's the conversion rate between watching the first episode of a sci-fi series and finishing the first season? How does it compare to other categories? Maybe sci-fi TV shows just aren't bingeable. Or maybe the show is low quality (ok, maybe *terrible* quality), and people can't stand to finish a season, let alone a whole series (*Sense8*). To more directly measure supply quality, you can also look at user ratings on sci-fi TV shows. Do they tend to be much lower than in other categories?

When looking at all these metrics, we should segment by user attributes, like viewer country or language. Netflix is a global platform, and it's not fair to expect users to act as a monolith. Maybe sci-fi watchtime, relative to other categories, is okay in English-speaking markets, but more effort needs to be put into closed captioning and voice dubbing to serve other countries.

Bonus Points: Zooming Out

A good answer should also consider that it might not even be an actual supply or demand issue. Maybe, there's an issue at a step higher in the funnel, like sci-fi shows that, for some reason, don't tend to make it into Netflix show recommendations. Or maybe more mainstream shows get most of the advertising budget. In both cases, there is demand for sci-fi, and there is content available to meet their needs, but discovery is broken so people aren't aware of any sci-fi show besides *Stranger Things*.

We can also look outside of Netflix for an answer. Through consumer surveys or audience insights data from a company like Nielsen, we can benchmark engagement in sci-fi shows against broader interest in the category. This way, we could tell if there is generally less demand for sci-fi content, both on and off Netflix, or if Netflix in particular is underperforming in this segment relative to competitors and cable.

Solution #10.13:

Before we jump into the technical details of performing customer segmentation, it's important to flex your business muscles and explain what customer segmentation analysis is and brainstorm a few concrete ways the analysis results could boost store sales.

Step 1: Explain How Apple Benefits from Customer Segmentation

Each person that walks into an Apple store has individual needs, desires, and preferences. In an ideal world for Apple, they would be able to hyper-target their product offerings and store design to cater to a single person's needs, one at a time. Obviously, this isn't feasible. On the other hand, treating all Apple customers the same misses the variety of customer needs. As such, by grouping similar customers and creating customer segments, Apple can customize its in-store sales strategy to large groups of customers at once.

However, our approach does have a caveat: by relying on historical data for customer segmentation, we aren't able to analyze non-Apple customers since they wouldn't show up in prior store sales data. Therefore, it's important to let the interviewer know that a customer segmentation analysis should be complemented with some competitor research or market-level analysis.

Step 2: Brainstorm Ways Customer Segmentation Can Boost Sales

There's many different ways to segment users, and then use those insights to boost sales. For example, one axis you could segment users on is their tech savviness. This information could impact the different kinds of sales scripts Apple uses. For example, a store salesperson convincing a software

developer to buy a MacBook would use a very different pitch than when explaining the benefits of the product to a nontechnical person. It could also impact the store staffing — maybe each store should have a technical expert to field the toughest questions.

Another dimension on which you could segment customers is by the main type of product they bought. Say, for example, our analysis found three main types of people routinely walk into an Apple store: iPhone purchasers, MacBook purchasers, and Apple Genius Bar customers who pay for an issue to be fixed. By separating customers into three groups, we learn that iPhone purchasers are five times as likely to buy AirPods than MacBook purchasers or customers coming in to get their device fixed by the Apple Geniuses.

This insight from customer segmentation could mean that it's best to place the AirPods next to the iPhones, to increase the chance of a cross-sell. Another implication of this insight would be to train salespeople to upsell AirPods to customers who are about to buy iPhones, but not waste their time on the upsell for MacBook shoppers. Another idea is creating a new discounted iPhone and AirPods bundle to entice customers into buying both products at once.

Step 3: Explain How to Perform Customer Segmentation

To perform customer segmentation, we could use K-means clustering. We could visualize the data or do hyperparameter tuning to find the appropriate number of clusters to segment the users into. Besides running K-means on the transactions data, we could also try to connect online sales data to in-store customers. While this analysis is primarily for understanding brick and mortar shoppers, cross-referencing in-person customers with their potential online purchases on Apple.com could help give a more complete picture of the customer.

Solution #10.14.

The first step would be to gather the basic data on both the iOS and Android users for both Facebook and Instagram. You could analyze user demographics such as age, gender, race, and location. You could also analyze user activity, looking at metrics such as time spent overall, and time spent on various activities (feed, in-app messaging, etc.) for both groups of users on both apps.

We can visualize the user activity metrics by each cut of user demographics to get a top-level understanding of where any differences may lie. For example, iOS users may, on average, spend much more time on the Facebook ecosystem than Android users do, and this “top-of-funnel” reason may lead them to use Instagram more also. Alternatively, iOS and Android users may, in general, be from different age groups, and this could be affecting their respective levels of Instagram usage, as Instagram isn't as widely used by older people.

Another set of factors to consider would be the actual Instagram's device and resource requirements, relative to Facebook's requirements. Maybe iOS devices have a much easier time downloading the Instagram app, since the app size is smaller for iOS than Android. Maybe the Instagram app only works on devices that have updated their OS within the last two years, and Apple devices tend to run the latest OS much more than Android devices.

Maybe the problem lies with the actual app experience --- do Facebook and Instagram perform the same way on both platforms? What do app store ratings, number of bug reports, feed scroll latency, and percentage of sessions with app crash for both apps on both platforms look like? Maybe Facebook works equally well on both platforms, but Instagram has under-invested in its Android app optimization.

Finally, for a difference so big, across so many users, I'd make sure to talk with user experience researchers, folks from the product strategy teams, and the Android and iOS leads for both Facebook

and Instagram. While cutting the data may reveal the underlying reason, our gut intuition is that for such a large difference across so many users, there's likely a bigger structural or strategic cause for the disparity that a purely SQL-based analysis may not uncover. At the very least, by talking to other domain experts, we can add some more color to our analysis.

Solution #10.15.

Just like with every metrics question, a good answer should start out with a brief discussion of the business goal — in this case, into the goal of Capital One's credit card. It should also mention a few of the stakeholders involved with this business goal before determining the best metrics for measuring retention.

Step 1: Explaining Capital One's Motivation Behind Credit Card Retention

A credit card's "stickiness" is the frequency and duration of its use by its holders. The bank issuing a card is motivated to encourage a card holder to use the card frequently and over the long term — in other words, to increase its "stickiness," as this earns it a greater profit in interest (if a holder carries a balance) and directly on transactions. A card holder would prefer a non-cash option having low monthly repayments, a low interest rate, and, perhaps, rewards for using the card. The goal of a bank's reward system (cash back or other perks associated with card use) is to increase card usage and its customers' reliance on the card over the long term. Thus, if Capital One gave no incentives to cardholders to encourage them to use their card to pay for purchases, then users would most likely use their cards less frequently or, possibly, not use them at all — hence, a decrease in stickiness.

Step 2: Brainstorming "Stickiness" Metrics

For Capital One to assess the stickiness of its Quicksilver card, some potential metrics it might use are as follows:

Daily active users to monthly active users: Although this ratio can be over any interval that you deem appropriate, the goal of this metric is to see what percent of active users during a longer interval (in this case, a month) are active over a shorter interval (daily). A ratio of, for instance, 0.7 would suggest that 70% of cardholders who spend with the card on a monthly basis also do so on a daily basis. The higher this metric is, the stickier the product.

Month-over-month retention: If you were to create cohorts of people based on when they signed up for the card, you could determine what percentage of the cohort churns from any given time interval. In this case, a month-to-month time interval seems reasonable, as credit cards are billed monthly. Seeing what percentage of cardholders remain after X months enables you to see trends in duration of use before a customer leaves or the average duration of time a customer remains with the card before leaving. However, a card holder can also be inactive without actually closing the account (a "silent churn"). Many cardholders typically use various cards for one particular purpose or activity (e.g., one card for travel, one for dining, and so on). Therefore, attempting to track such behavior as well is advisable.

Transaction volume churn: Tracking the total amount spent by cardholders is critical, since this correlates to the amount of revenue Capital One could make. By looking at all users who spent money last month, and their total transaction volume, and comparing it to the total transaction volume of those same users this month, you can see if adoption is growing or shrinking.

Solution #10.16:

One good question to ask the interviewer before answering is “What’s the goal of pricing?” For a new service, it could be okay to run the feature without profit if you believe you can aggressively gain market share and turn on monetization later. Assuming we don’t want to run a free service, we can price YouTube Premium using cost-plus pricing, value-based pricing, or competitor-based pricing.

Cost-Plus Pricing

For a cost-plus pricing approach, we’d look at how much it costs to provide YouTube Premium in that country and then add a margin on top of that number to enable the business to earn a profit. We’d account for product localization costs, marketing costs to advertise to the new geography, and bandwidth costs to serve content. While not technically a cost, because there’s no ads in this feature, you could also account for the lost ad revenue per user that YouTube would’ve earned.

Finally, because content licensing can be a bulk part of music-streaming service costs, and oftentimes media is licensed on a per country or per region basis, it’s important to understand how expensive offering our music library would be. By adding up these costs and then tacking on a premium to this number, we’ve got one way to price the product.

Value-Based Pricing

A value-based pricing strategy would price the service relative to the amount of value a consumer perceives from using the service. The most direct way to gauge the perceived value of YouTube Premium would be to ask users themselves through consumer surveys and focus groups.

An alternative could be to see what an optimal price was in other countries where this product launched, and assume that price hits the optimal value offered. Then, adjust the price to the local market based on the market’s average per capita income.

Competitor-Based Pricing

You could price the service based on competitor video and music streaming services like Netflix, Hulu, Spotify, and Apple Music. In each new country where YouTube Premium is being launched, by seeing how much more valuable (or less valuable!) our service is, we can adjust the pricing against the competitors. All else being equal, the larger the number of alternative options for a country’s residents, the more likely that YouTube Premium’s pricing would need to be discounted, especially given the relatively low switching costs available to consumers.

Wrapping It Up

Likely, an appropriate price for YouTube Premium would be a blend of all three pricing strategies. To triangulate on an exact answer, you’d want to consult with stakeholders like the sales, marketing, and finance teams. You could also always try A/B testing the prices, or running discounts or tiered memberships to get more signal into what an optimal price point may be.

Solution #10.17:

For questions dealing with whether a company should launch a certain feature or not, it’s best to not prematurely discuss the proposed idea’s merits. Instead, clarify with the interviewer what the feature actually is, what the company’s hypothesis is behind proposing such a feature, and how it would impact key business metrics. Great answers would elaborate on the potential pitfalls of shipping the feature and, lastly, end with a final recommendation.

Step 1: Clarify Twitter's Product Hypothesis

Currently, users on Twitter can only like, comment, or retweet a tweet. Emoji-style reactions are likely being considered as a way to improve the engagement rate of Tweets. The hypothesis is that by reducing the friction for expressing more complex emotions like “this tweet was funny” or “this tweet made me mad,” users will be more likely to engage with tweets.

This feature also addresses a common user issue: hitting “like” on sad news feels weird. By providing a more nuanced alternative to liking a tweet, there could be more engagement on tweets overall.

Step 2: Explain Twitter's Business Goal

If emoji-style reactions lead to more tweet engagement, there would be many positive downstream effects to Twitter's business. Firstly, users would be more engaged with their feed and interacting with more pieces of content, leading to longer and likely more meaningful sessions.

It's also not just about the amount of engagement given — it's about the positive impact on the receiving end too. People whose tweets get more engagement will get more notifications of people interacting with their post, driving them to check Twitter more often.

When tweeters notice their posts getting more engagement, that dopamine hit will surely incentivize them to post more on Twitter. This will improve tweet creation metrics, which would boost the amount of interesting content found on the timeline for all users. This positive flywheel of more engagement leading to more content leading to further engagement would improve both time spent on Twitter and user retention rates. More time spent and more user retention means more ads seen, which is crucial for Twitter's ad-supported business model.

Step 3: Identify Data to Support Product Hypothesis

We want to find some data that can guide us on whether emoji-style reactions are desired by users. The best way to do this would be to analyze current user activity and see if there is some unmet latent demand. If we have evidence that users are taking multiple steps to express the common reactions like “love” and “haha,” then simplifying the process for expressing these feelings via emoji reactions would be a logical next step.

In terms of specific user data on Twitter that we can use as a proxy for demand for the reactions feature, we could look at current Twitter threads and analyze how often sentiments corresponding to the proposed reactions are expressed. Take the “haha” reaction, for example. If we see many short comments laughing at a particular thread or reply, using keywords like “lol” and “haha” and “lmao,” we'd have a strong indicator that users want to express that they found the tweet funny. In that case, offering them an easier way to express that emotion would be beneficial.

We could also work with user research teams to conduct surveys on groups of users to confirm the data: do these people desire an expanded set of reactions? If so, for which reactions, and why? The user research and the data should collaboratively point to the same direction (that people want the ability to express more reactions).

Step 4: Counter-Argument for Shipping Reactions

Counter-arguments for emoji-style reactions are that it increases the complexity of Twitter. Right now, having a single reaction makes things intuitive. Another issue is that more complex emotions are expressed in comments. If we made reacting easier, we'd likely have fewer people replying to a tweet with messages like “this is so funny” or “i hate this.” Plus, from an aesthetic and brand viewpoint, the Facebook-ification of the Twitter product may not be desirable.

There are also several challenges from the product and engineering side. For example, which reactions should be used? It may be redundant or confusing to have “love,” “enjoy,” and “like.” reactions. Also, do we have the engineering resources to dedicate to build and test this nontrivial feature? It’s not a given that the potential engagement boost warrants the time and money needed to launch the product. Has opportunity sizing been done to show the potential ROI on this project?

Lastly, all engagement isn’t equal. Before running this kind of experiment, you’d want to align with stakeholders how important replies are versus reactions. By anticipating the likely trade-off that will need to be made if this feature is successful (overall reactions would be up, but number of comments would be down) and discussing this issue with stakeholders, you’d mitigate launch blockers and have an easier path to shipping it after the A/B test results come back in.

Step 4: Make Final Recommendation

If we have alignment that the goal is to increase engagement, have found good reason that the feature demand is legitimate, have intuition on why implementing the feature as described would drive engagement, and think the business benefits outweigh the development cost and time required to A/B test the feature, then and only then does it make sense for Twitter to test reactions.

As mentioned in the counter-argument section, after we have A/B test results, we should align with stakeholders about the likely metric trade-off that will occur — increased reactions but decreased comments — before making the final launch decision.

Solution #10.18:

Step 1: Stakeholders for Slack Engagement

Slack’s user engagement is important because it aims to be the go-to work and productivity tool. Since Slack serves as a place where people collaborate together, user engagement is critical to monitor and measure. A user in this context is simply any person who has an account on the platform. User engagement affects the business directly since Slack operates as a subscription-based model, where users pay per month for features on the platform. With more consistent user engagement, there is likely to be longer-term retention and new customers over time.

Step 2: Defining User Engagement Metrics for Slack

To measure user engagement for Slack, we could look at DAUs (daily active users), WAUs (weekly active users), and MAUs (monthly active users). An active Slack user is defined simply as anyone who signs into Slack on a given day. Since a product like Slack is meant to be used daily, tracking DAUs is crucial, and the ratios DAUs/WAUs and DAUs/MAUs are typically used, since a higher value of one of these ratios would mean the product was more sticky. Note that we would need to be aware of weekends and holidays (seasonality).

Since the product is a collaboration software, another core metric for engagement would be the number of messages sent. Again, as with the number of active users, we would want to look at messages sent per day, per week, and per month. Other auxiliary metrics we could track include the following: creating organizations, applying for membership in organizations created by another user, etc. However, the two primary metrics would be number of active users (i.e., those who sign in) and number of messages sent.

Additionally, measuring these trends at the cohort level would be important to ensure consistency over time. For example, DAUs could be dropping slightly, but the behavior among various cohorts

could be very different. In this case, identifying the cause(s) of why specific cohorts are seeing drops of greater significance than others would be important.

Step 3: Defining Leading Indicators for Engagement Decline

To receive an early warning of declining user engagement, we could look at trends over time in numbers of both DAUs and daily messages sent, as declines in either or both of these could be leading indicators of an overall decline in users. For example, a user who eventually leaves might initially be a DAU, but then slowly become a WAU , and, finally, become a MAU only. Such a lack of engagement would show up in DAUs and in numbers of messages sent, and, hence, tracking these would be important, as would calculating and tracking the ratios DAUs/WAUs and DAUs/MAUs. Again, looking at these metrics on a cohort level is also important.

Case Studies

CHAPTER 11

*Data Scientists don't do **pure** "statistics" work or write SQL queries in isolation. They solve business problems, end-to-end, as part of a larger team. To reflect this job's need for a wide range of soft and hard skills, and assess your overall holistic problem-solving ability, some companies ask you open-ended case interview questions. These problems force you to apply your product sense, statistics knowledge, ML experience, and system design skills to a real-world business problem. While some companies ask case interview questions as part of an on-site interview, other companies adapt these problems into a take-home project, often giving you both a problem statement and a dataset to base your answers on. In this chapter, we present tips to tackle these open-ended questions, as well as solve eight interactive case-style questions from companies like Facebook, Amazon, and Citadel.*

What Case Interview Questions Look For

Before we jump into what hiring managers look for from candidates solving case interview questions, let's start with a concrete example of what a case interview entails. At LinkedIn, for example, you might be asked how you'd recommend jobs to people. You'd need to clarify what the business and product goals are for the recommendation feature, and tie this information into your solution. You'd be asked about what data you'd use to train the model, what business metrics you'd use to know if you were successful, and what technical and product-related edge cases you might run into. Finally, you could be asked how to A/B test and deploy the solution. Essentially, it would cover the End-to-End Machine Learning steps we covered in Chapter 7, along with some of the product-sense topics from the last chapter.

Because of the similarity to the product and business-sense questions, many of the tips from Chapter 10 apply to case interviews too. Being conversational, stating your assumptions, bringing in outside experience to show your depth, and keeping business goals top of mind are all table stakes.

Here's a few more tips to keep in mind for case interview questions:

- **Clarify, Clarify, Clarify:** In the product-sense chapter, we beat this dead horse, but it's worth repeating because jumping into a solution without clarifying the business and product goals is the #1 mistake we see candidates make!
- **Be Coachable:** Be open to suggestions from your interviewer --- often these are hints that they want to steer the conversation in a particular direction. Agree and amplify when they suggest something: "Oh, yeah, I didn't consider boosting; that's an interesting angle. Maybe XGBoost could be used!"
- **Remain Pragmatic:** The interviewer's team probably has been working on this problem for several months and is not expecting a revolutionary solution from you. Stick to solutions you think are reasonable that align with the product and business goals.
- **Mention Tradeoffs:** Remember to bring up trade-offs and limitations of the approach you are presenting. Briefly mention other options or techniques you could have used, and how they compare --- it helps you show depth. Plus, you can even ask the interviewer which of the options they'd like to discuss in more detail, which lets the interviewer steer the conversation towards what they find most interesting.
- **Timebox Yourself:** Time is of the essence. Avoid going down rabbit holes. The interviewer is trying to get as much signal as quickly as possible, so rambling on about something irrelevant hurts you. If you're uncertain about how much detail you should provide, give an initial answer, and then ask the interviewer if they'd like you to elaborate -- remember, this is a two-way conversation!

Approaching Take-Home Challenges

The types of questions you're asked in case interviews are akin to what you might be assigned in a take-home data challenge. The main difference is, for a question like "How would you improve customer satisfaction of our products?" you'll now have data you need to analyze to back up your recommendations.

The biggest piece of advice we have is to tell a story. As you saw in the earlier chapters on portfolio projects and behavioral interviews, storytelling permeates every aspect of selling your abilities and your work. A take-home challenge is no different — now, you just have a way to incorporate concrete numbers and data visualizations into the story you tell. From surveying hiring managers, we know this advice works, because one of the most crucial skills hiring managers assess candidates on is their ability to turn their analysis into an accessible recommendation. And the best way to convey a recommendation and ensure stakeholders make the right decision with data is to tell a narrative that they can easily comprehend. Storytelling is how you do that!

Since take-home challenges are long, widely differ from company to company, and refer to a specific dataset, they were difficult to incorporate into this book. But fear not: we filmed step-by-step video walkthroughs of us solving three real take-home challenges to make available to you, our readers. In the past, we've usually sold our take-home challenge video course, along with the actual case study problem and data, to our coaching clients for \$89. But, if you share what you liked about Ace the Data Science Interview in an Amazon review, we'll give you a copy for free!

Just email a screenshot of your “Verified Purchase” review to nick@acethedatascienceinterview.com to access our free take-home data challenge video course!

Case Study Questions

Case interview problems have multiple follow-up questions. In a real interview, you wouldn't be presented with all the follow-up questions at once. As such, to simulate a proper interview, the follow-up questions are included in the solutions.

- 11.1. Citadel: Say you wanted to estimate the physical store sales for a publicly traded retail chain in the U.S. You have access to third-party foot traffic data for each store that was derived from anonymized GPS data collected from a panel of 10 million mobile phones. At a high level, how could you use this foot traffic dataset to predict the chain's in-store revenue?
- 11.2. Amazon: Assume that you are designing a system whose purpose is to recommend what shows a user should watch on Amazon Prime Video. What data and techniques would you use?
- 11.3. Airbnb: Assume that you are modeling the yearly revenue of new property being listed. What kinds of features would you use?
- 11.4. Walmart: How would you build an algorithm to price products sold physically at Walmart stores?
- 11.5. Accenture: Assume that you want to help a major hotel chain analyze what people say about their brand on websites like Facebook, Twitter, and Reddit. Why might this be useful, and how would you go about doing it?
- 11.6. Facebook: Suppose you are to build out Facebook's friend recommendation product, also known as the “people you may know” (PYMK) feature. How would you go about doing this?
- 11.7. Stripe: Assume you are working on a loan approval model for small businesses. What metrics would you use to evaluate the model, and what trade-offs would be involved in the evaluation?
- 11.8. Instagram: How would you provide content recommendations for Instagram Explore?

Case Study Solutions

Solution #11.1

Citadel - Retailer Revenue Estimation

Say you wanted to estimate the physical store sales for a publicly traded retail chain in the U.S. You have access to third-party foot traffic data for each store that was derived from anonymized GPS data collected from a panel of 10 million mobile phones. At a high level, how could you use this foot traffic dataset to predict the chain's in-store revenue?

Sure — before starting, I just wanted to clarify a few things. First off, what's the use case for predicting the in-store revenue — why do stakeholders want this number? I'm assuming it's to make investment decisions. Second of all, we are trying to make revenue predictions for the entire chain -- not for each store, correct? Lastly, what's the granularity needed for our revenue prediction? I'm assuming it's quarterly due to quarterly earnings for public stocks?

Good questions — we are trying to use the revenue estimate to decide whether to buy (long) or sell (short) stocks of certain big box retailers before their quarterly earnings come out. We have robust models to measure the brand's e-commerce sales, but are trying to proxy the entire chain's sales from physical retail with this foot traffic dataset. Any other questions?

Can you tell me more about this data source — what do the rows/columns look like? I presume each row is for a given store, and the columns will include some kind of non-normalized foot traffic value based on devices tracked for a given time period. I'm guessing it's at a daily granularity?

Yes, it's the amount of visits detected to a store, reported at a daily granularity. Each row is a date-storeID pair, and the column is an integer that holds the number of detected store visits that day. The numbers aren't normalized. So, how would you model the store sales?

The high-level approach would be to run a model to predict revenue based on some aggregated measure of foot traffic. Since our third-party foot traffic is likely to not be rolled out --- for example, each store location per hour --- we'd want to roll up the foot traffic to the same granularity as the company's revenues (which is reported quarterly as a publicly traded company).

Note that we want to normalize this traffic to account for the fact that the data comes from a sample, so the quality of foot traffic data (volume, frequency, noisiness, etc.) would affect the quality of predictions. Then we can look at a simple regression model that correlates revenue as a function of this normalized foot traffic.

Say the foot traffic data only goes back 3 years, which means you only have 12 quarters' worth of revenue for the retailer to train a model on. How would you avoid overfitting on such little data?

To reduce overfitting in general, we can pick simpler models like linear regression (versus more complicated models) and apply regularization. We could also report a 95% confidence interval for our revenue estimate to better help decision-makers account for uncertainty in our forecast. In the case of just having so few data points (12 quarters' worth), we can look to other similar retailers.

As a concrete example, say we were modeling Target revenue --- we include data from not only Target but also similar retailers like Walmart and Costco to add more data points to produce a more robust regression. This helps our model learn how foot traffic affects retail sales in general.

How would you use the foot traffic panel data to determine the “true” number of visits to a store?

Because this foot traffic data would come from a constantly changing panel of mobile phones, to account for panel size changes, we'd incorporate the panel size into the equation. As such, the number of “true” visits to that store would be (sample foot traffic for store / sample population size)* U.S. population size. You could also localize this normalization --- it's not obvious the panel coverage would be equal across the entire country. Instead, you could divide by the amount of people in the panel in a given census block group or zip code, and then multiply by the true census population of that locality.

What is sampling bias? In what ways do you think this dataset may be biased due to the way it was sampled?

Sampling bias is when data is sampled from some segments of the population at disproportionate rates. In this case, location data may be biased because not everyone has a smartphone. Of those that do, not everyone is as likely to have apps with a location component to them. And finally, not everyone is as likely to turn their location permissions on.

How would you check for sampling bias within the panel?

We can check if the given location analytics dataset is a representative sample by comparing it to the distribution of the underlying U.S. population. Based on where a smartphone spends most of its time stationary during nighttime hours, we can assume that to be a device's home location. We can compare the home location of the devices in our dataset to the U.S. Census's data on population at the state, county, and census block group levels to account for any geographic bias in our dataset.

Similarly, from the home location and census data, we can understand what types of neighborhoods these devices live in. From this, we can approximate the average income and race of people in our panel. We can again check this against census data to see how well it lines up with the true U.S. population.

Essentially, we want to ensure that for any given dimension, the expected proportions in the sample (of any given geography, county, etc.) are similar to those of the entire population.

Suppose you try to use a linear regression model, but it doesn't run, and communicates that there is an infinite number of best estimates for the regression coefficients (through an error message of some sort). What could be wrong?

A few things could be happening here. One is that the number of features exceeds the number of data points. In this case, there will be features that can have an infinite range of values. To address this, you could look into PCA to reduce dimensionality. Another possibility is that there are some features that are perfectly correlated (either positively or negatively) with others, i.e., multicollinearity. This would cause unstable coefficients — you'd get different coefficients per feature for different runs of the regression. To address this, you could visualize the correlations between features and remove any unnecessary ones.

You run your regression on different subsets of your data and find that in each subset, the beta value for a certain variable varies wildly. What could be the issue here?

The dataset might be heterogeneous, i.e., the distributions underlying the various subsets are drastically different. This may be due to improper data collection or sampling techniques, in which case, it is recommended to cluster datasets into different subsets wisely, and then draw different models for different subsets. Or, use models like nonparametric models (trees), which can better deal with heterogeneity. Note that there's no silver bullet — trees may not entirely solve the issue if the underlying variance in the data is extremely high.

What are some limitations of using a third-party dataset of foot traffic data derived from mobile phones to model the retailer's store sales revenue?

At a high level, there's always a risk when relying on an outside third party for a core part of your system or solution. In this case, since we are using a third-party foot traffic dataset derived from mobile location data, we likely don't have granular information or transparency into what apps are sending us location data, how the apps in the underlying panel are changing, and how users in the underlying panel are churning and shifting. Plus, foot traffic isn't just a raw number that is directly measured — it's modeled from seeing if mobile location data for a person is near or inside a store geofence. The company we are buying this alternative data from could make changes in their visit attribution algorithm, which can make maintaining consistent models or backtesting difficult.

Another glaring limitation is that foot traffic to a given store does not necessarily mean the customer made a purchase, even if the traffic correlates well with transaction data.

To address these limitations, we could look to use other alternative datasets, like receipts data, credit card data, or point-of-sale datasets. If you had access to such data, you could see how this foot traffic data and the data from those transactions compare by normalizing both datasets and running correlations between them.

Solution #11.2

Amazon - Show Recommendation

Assume that you are designing a system whose purpose is to recommend what shows a user should watch on Amazon Prime Video. At a high level, what data and techniques would you use?

Although many variations are possible, underlying most recommender systems is a collaborative filtering approach. The general strategy is to recommend shows that are watched by other users who are similar to the user at hand. In plain English, the idea is that people tend to watch and like programs that “similar” people have also watched and liked.

What kind of data would you use for collaborative filtering?

Matrices are typically used to represent consumer preferences with respect to programming. Assume an $m \times n$ matrix, whose $i = 1, \dots, m$ rows represent m users, or, in this case, Prime Video customers, as of some time interval, and whose $j = 1, \dots, n$ columns represent shows and movies in the Prime Video catalog. Each i -th to j -th position of the matrix contains either no value if the user did not watch the program or a rank that measures the i -th person’s attitude toward the j -th program (i.e., 1 through 5, where 1 = dislike greatly and 5 = like greatly) if this user watched the program and evaluated it.

		shows
		4 5 5
users		2 3 3
		n/a 4 4
		4 5 n/a

How would you calculate if two users were similar?

We could use this matrix’s customer rows to measure the similarity between viewers’ tastes in programming by calculating the cosine similarities between customer programming ratings. Note, however, that customers can have biases (expressed by, for example, awarding a particular program a very high or a very low rating), and so normalization of scores (per user or across the entire dataset) is recommended.

How would you use the user similarity score to make show recommendations?

Once we have calculated similarity measures for customers, we can use them to estimate reactions of other customers to existing programs they have not yet seen, or to new programming. The basis of this method is the assumption that customers with close similarity scores will share similar tastes in programming. For a given customer who has not viewed a certain set of movies and TV shows, this method generates recommendations by first identifying customers similar to that customer in viewing choices through proximity of similarity score. Programs liked by these similar customers and unwatched by the customer can then be recommended to them. A more finely tuned procedure is to employ viewer rankings of similar Amazon customers to calculate weighted averages of programs, and these weighted averages can then be used to rank programs to be recommended to the user.

How would you address the cold-start problem — that new shows don't have any ratings, and similarly, that new users haven't yet rated any shows?

To address these weaknesses, which stem from an absence of data, we can do the following:

1. For new shows, don't include them in recommendations, but in the UI, have a separate panel for "new to Amazon."
2. For new shows, construct measures of show similarity using features of shows (genre, actors, language, content of the show, etc.).
3. For new users, use factors outside of the given matrix (user demographics, for example) in constructing the similarity measure for new users.
4. For new users, simply recommend what's popular or trending on Amazon until more data is collected on a user's preferences.

Besides collaborative filtering, what other techniques can you look at?

An alternative approach, since collaborative filtering is based solely on user preferences with respect to content, would be to use details of shows' content (metadata related to them, for example) to generate recommendations. Here, recommendations to users would be based on content to which they had responded positively in the past.

Say that you have a new and improved model that you want to test. You decide on total watch time as the top-line metric. You run an A/B test and it shows a significant increase in watch time with a p-value of 0.04. Do you ship the new model?

Before deciding, it would be good to make sure that the experiment lasted two weeks or more, as you do not want to stop the test early, and want to account for day-of-the-week effects.

Additionally, it is always good practice to consult with business stakeholders to properly make the decision. Even if the top-line metric of total watch time changed positively, what happened to the other metrics you were tracking? For example, you could measure precision: the percentage of recommended shows a viewer watched? You could also look at recall: of the shows someone watched, how many of them were recommended to them? Discussing relevant counter metrics and evaluating the nontechnical aspects of the A/B test are also critical for making a launch decision.

All in all, just because the experiment is statistically significant does not mean it is practically significant (has enough impact to do something).

What counter metrics would you also consult?

A valid counter metric would be the average ratings of the recommended movies versus ratings of user-selected movies. We don't want our recommendation algorithm to bait users into watching movies they ultimately like less than what they would have picked on their own.

What are other model deployment or product considerations you might have?

A few more things we could address as we deploy the model, and iterate on the product:

- How computationally intensive is this model to train?
- How often should we retrain the model to account for the shifting desires of the user base as well as new media continuously added to the content library?
- Should we favor Amazon originals? If so, how much should we boost this content in recommendations?

- Should we take a more editorial approach for recommending shows to new user, or recommending newly released shows to people?
- Should we update the recommendations when the user has not interacted with the initial recommendations for a certain time? If so, what events trigger a recommendations refresh?

Solution #11.3

Airbnb - Listing Revenue Model

Assume you are building a model to predict the yearly revenue of new properties being listed. What property features would you use in making such predictions?

Great question! What's the business purpose of this? For example, are we going to be showing new listers their potential revenue, or are we using this internally to decide on what markets to expand in?

Let's assume it's to show potential customers an expectation on how much yearly revenue they can bring in.

In that case, I'm wondering whether ML is the right place to start. In my mind, it feels like what I'd really care about is getting exact comps (for example, my neighbor Joe made \$8k last year, renting out his room for 45 days). What's the benefit of using ML to surface predictions vs. directly showing nearby listings?

Let's assume that for privacy reasons, we can't show specific comps. We also, for legal reasons, need to make our predictions personalized based on a user's input data. What features would you use to model the expected yearly revenue?

For a particular property, relevant features I'd include:

- Property Details: number of bedrooms, bathrooms, square footage, if it has a kitchen
- Pricing Details: owner's set nightly rate, occupancy limit, and minimum nights required
- Location: zip code, distance to nearby attractions like the airport or convention center
- Local Market: prices of nearby listings, occupancy rates of nearby listings

What data preprocessing and cleaning would be necessary to handle duplicate or inconsistent values prior to incorporating these features into your analysis?

Duplicate values are often erroneous (due to mistakes in recording data or lack of system standardization), so we can generally safely discard the duplicates (although we should first verify that they are, in fact, duplicates). An example of where there may be duplicates is when the owner lists the whole house, as well as individual rooms, as different listings.

For inconsistent values, we can perform basic checks to ensure that all the data within the columns in which the inconsistent numbers appear are uniform in type (to check whether, for instance, price hasn't been accidentally swapped with zip code or house number fields).

After ensuring data quality and that the particular models we want to use include numerical data (which should be normalized to improve the interpretability of the different features), we would then encode categorical variables to represent binary or ordinal variables, depending on the number of categories needed.

What if many features are sparse — how would you deal with it?

Note that various algorithms will be more or less fit to deal with sparsity — for example, random forests can handle sparsity better than linear regression. To handle sparsity, if you were to use traditional ML,

techniques you could look into either feature selection or PCA/SVD for dimensionality reduction. On the deep learning side, you can look into tweaking various optimizers (some are better at handling sparse features — for example, RMSprop and Adam) or using embedding layers or autoencoders to learn a smaller relevant feature space.

What model would you want to use to model the listing revenue?

Per Occam's Razor, and for baseline purposes, it's probably better to start simple, so we could use a linear regression. This makes sense because we can expect an increasing, roughly linear trend between features like number of bathrooms, number of guests allowed, square footage, and revenue.

What about using a neural network? Would you use it over linear regression?

Neural networks can capture arbitrarily complex relationships but tend to overfit on data. They also generally require a large amount of data to be useful.

So, if we have a very large amount of training data, believe that the model does not need to be very interpretable since only its predictive accuracy matters, and believe that a linear regression will underfit because the relationship between the output and the predictors is truly non-linear, then we can try using neural networks over linear regression. We can also add regularization or different cost functions to better address the high variance (chance of overfitting).

Any other models, besides a linear regression or neural network you'd want to try?

A random forest or boosting model (XGBoost) would probably be better in terms of adding complexity versus a linear regression, but without overfitting too much like a neural network.

Let's say your model endpoint needs to support a high number of queries per second (QPS). How would this affect your model choice?

Depending on the complexity of the model, the amount of compute resources to support the QPS requirement would vary. If the model complexity is high, more computational resources are needed to support the same level of QPS versus a less complex model. However, high complexity models may be more accurate. Therefore, we can assess the trade-off between the resource consumption and model accuracy, and then decide what model to deploy that meets the QPS requirement.

Say you used linear regression and found that square footage was crucial at predicting revenue made. However, a listing's square footage was missing for 10% of properties. How would you deal with these missing values?

Missing data can either be eliminated by dropping the rows with the missing data or ignoring the column with missing data entirely. Alternatively, missing data can be imputed using simple methods like replacing missing values with that feature variable's mean, median, or mode. However, for such an important predictor, a one-size-fits-all imputation of dropping the data or imputing it with a fixed value is likely not appropriate. Besides, the missing user information itself may be a signal, so averaging would not make sense. For example, the user may not disclose their square footage in cases where the listing is abnormally small, and hence, imputing the value with the average listing's square footage may lead to incorrect results.

In general, the best approach is to build a model to predict the missing features given the other features (for example, square footage can be proxied by the number of bedrooms and bathrooms).

Do you have any outside-the-box approaches to sourcing the missing square footage data?

We can use push notifications on the app to get the end users to try and provide the missing information to get an updated revenue estimate. Or tell them they're listings won't rank well until they fill out all

their information. Getting the missing data from the users would be the most direct way to get more accurate data, but there may be a low response rate or response bias in the submissions.

Another approach could be to use third-party datasets, like parcel data or county records, which have the square footage already. This may not help if only a single room is being rented out, but for cases where the whole property is being listed, likely by matching the address to third-party datasets, we can get the square footage information even if the owner doesn't upload it.

What action would you take if there were too many features for you to look at thoroughly, say, hundreds of them?

That case describes the curse of dimensionality, in which the number of variables, or features, to be included in an analysis is so great that it makes performing the analysis extremely challenging. This phenomenon not only renders visualizing data extremely hard — or, more likely, impossible — but it also greatly increases the difficulty of assessing features to calculate similarity measures.

In this case, the first step would be some feature selection, in order to filter out features with very low variance and without any relationship to the target. This can be done through looking at correlations and VIFs (variance inflation factors) to look for features that have predictive value. Alternatively, we can apply specific feature selection methods such as recursive feature elimination (RFE), which continually refits the model with features and discards those features with the lowest feature importance.

Then we can run some sort of dimensionality reduction method that could be used to combine features that explain much of the variability exhibited by the data. Principal components analysis (PCA) is one such method; by combining two or more of the original variables that are highly correlated, a new variable is created that is uncorrelated with the remaining variables and is assumed to represent a latent feature underlying the original variables.

Solution #11.4

Walmart - Optimal Product Pricing

How would you build an algorithm to price products sold physically at Walmart stores?

To start off, I'd like to make sure I understand the scope of the problem. How many products are we trying to price? Are we determining product prices for a particular Walmart store, or all Walmart stores?

Good questions. Let's say we are trying to build an algorithm which can price all the Walmart products stocked physically at all stores in North America.

Interesting. So, just doing some back-of-the-envelope math, I think I read on your website you all have about 5,000 stores in North America. And how many items does a typical store stock?

I love how you are sizing up the problem. You are correct, we have about 5,000 stores, and at each store, and to keep the math easy, say we have about 100,000 items stocked per store.

Okay...so, 5,000 stores times the 100,000 products is...500 million. So we need to come up with around 500 million price recommendations each time our algorithm runs. That's a lot! My next question is the frequency — how often do we need to determine prices?

Good question. I want to hear your take — how often would you run the pricing algorithm? What are some advantages of running this pricing algorithm more or less often?

With the retail industry so competitive, and online retailers like Amazon changing and personalizing prices billions of times per day, it makes sense for a retailer like Walmart to try to update their prices more often — maybe every single day. This could allow Walmart to respond more effectively to changes in consumer spending habits, and quickly react to seasonal and holiday trends. For example, they could mark down the price of Christmas wreaths on December 26th. Plus, for perishable items, like in the grocery department, depending on the “best before” date and available stock, updating the prices daily could help move inventory that is on the verge of expiry.

However, there's a computational cost to making 500 million predictions per day. And, of course, pricing products is in the land of atoms — not bits. There's a real human cost to changing pricing too frequently. People would physically have to go to the items with changed prices and append a new label. This has both human labor and materials costs. Then there's brand perception — since Walmart prides itself on everyday low prices, if prices fluctuate too much and customers start to notice, they may be hesitant to purchase, thinking they could get a better deal later.

A good compromise could be to change pricing of all products just twice — set an initial price, and then a lesser price if it needs to be cleared soon, and then a final clearance price.

I love that you brought up Walmart's “Every Day Low Price” strategy. Your hunch is right that we don't want our prices to fluctuate too much. Now, let's get back to the main problem.

Before I outline how I'd build the algorithm, I need to know what's the business goal of this algorithm? Are we trying to maximize revenue, profit, or some other metric? Also, I'm a bit unsure how to incorporate the everyday low price strategy into our algorithm, since it seems that could be at odds with maximizing profit.

For simplicity, assume that the singular goal of this algorithm is to maximize profit. With this goal in mind, how could you construct a simple supply and demand curve for each item to determine an optimal price point?

I'll assume that for any given product, the profit will be the difference in price minus cost, multiplied by demand, where demand is a function of a change in price (price elasticity). I'll also assume the demand curve relationship for every product is approximately linear, and so, for any given item, plotting quantity sold on the x -axis and price of the product on the y -axis should display a linear relationship with a negative slope.

In this way, we can use the underlying demand curve to identify the optimal price point for each item in terms of profit, i.e., $(\text{selling price} - \text{per-unit cost}) \times \text{number of units sold}$.

All I need from a data perspective is the history of units sold and selling prices for each item of interest. To actually make the supply and demand curves, I could use a linear regression model for each item that would predict the number of units that should sell depending on a price. In this regression model, where we are predicting demand using price as an input, the coefficient of the price variable would be a measure of the product's price elasticity.

How do you interpret that demand curve regression coefficient?

If the value of the coefficient was negative, this would mean that as the per-unit price were to be increased, the demand would drop. Most of the everyday household goods that Walmart carries should have a negative coefficient. However, some luxury goods, such as wine, can actually benefit from having a higher price, as it makes the product seem more premium, which stimulates the demand.

If the coefficient value was a large negative number, it means the demand for the product rapidly declines as the price increases. This is known as an *elastic good*. An example of an elastic good

would be beer, as it's not a necessity, and is easily substitutable with wine or hard seltzers (Whiteclaw > Truly). On the other hand, if the coefficient's absolute value were relatively small, demand for it would not be as sensitive to price changes. This product would be considered to be *inelastic*. Examples of inelastic goods would be cigarettes and prescription drugs.

So how would you double-check these elasticities?

Determining whether similar products have similar elasticities would provide possible validation of our findings. For example, items in a product category such as home essentials would be expected to have similar elasticities, as would shampoos and conditioners.

What are some limitations of using a linear regression model for building demand curves?

There are a few areas where pricing alone may not capture demand perfectly. For example, cannibalization does not get taken into account during pricing using a linear regression — if one item is chosen to have a discount, this can affect other items. Additionally, choosing varying pricing can involve mechanisms that have different effects on the optimal price — consider discounting an item using an ad versus using a physical coupon. Lastly, on a technical level, the linear model works well when selling patterns don't vary much by location and time period across products, and this may not hold true in practice.

Great, now let's assume you want to do more than use a simple supply and demand curve with linear regression. What other kinds of data could be relevant to train a “black-box” pricing algorithm?

Data we could use to train our model:

- **Item Details:** How much did it cost to procure this item? What is the item's shelf placement? What category of item is it? How much is a consumer getting (40 fl. oz. vs. 120 fl. oz. of laundry detergent)?
- **Competitor Pricing:** How much do other e-commerce and physical retailers charge?
- **Inventory Constraints:** How many do we have stocked? When is the next shipment coming in? Are we in a rush to clear out our inventory anytime soon (due to seasonality or if the good is perishable)?
- **Historical Prices:** How much, and how quickly, did we sell our inventory in the past? Both at the current store, and also at stores nearby?

How would you test your “black-box” algorithm?

We can run an A/B test as follows. First, we can take two categories of items that should be roughly comparable from a price elasticity, seasonality, and revenue perspective. You should also avoid categories that may tend to be in the same basket, as there could be interaction. For example, it would be bad for the *control* group to price video game *consoles* if the *test* group priced video *games*, as changes in one affect the purchases of the other. In our case, toothpaste could be the control group and toilet paper could be the test group. For the control group of toothpaste, we price using the status quo strategy, and for the test group of toilet paper, we use our black-box pricing algorithm. At the end, we monitor for lift in core metrics like revenue, profit, and sell-through rates. We should wait a decent amount of time — say a few months — so that the inventory is able to be turned over a few times.

Solution #11.5

Accenture - Hotel Review Analytics

Assume that you want to help a major hotel chain analyze what people say about their brand on websites like Facebook, Twitter, and Reddit. Why might this be useful, and how would you go about doing it?

Before beginning, just want to clarify the data — what kind of content are we looking at? Are we looking at reviews, comments, threads, or a combination of these? Also, do we need to go out and collect the data or is the data there and just needs to be analyzed?

Great question — for this situation, let's assume we're talking about just public online text and reviews posted on the biggest social media sites, and that it's already been processed and cleaned and stored in a database. Why might analyzing this data be helpful for the hotel chain?

Social listening is important — looking at reviews from Yelp, Tripadvisor, and Google Reviews, where there's an explicit rating, isn't enough. In the viral era, a Tweet or Reddit thread can blow up. It would be important for the hotel chain to consider both customer feedback and public opinion.

Posts on social media, whether they be positive or negative, have the potential to influence possible customers. Moreover, the sheer volume of opinions posted online about any subject magnifies and increases the scope of their influence. The client's ongoing objective, then, is to maintain and improve public perception of the brand, and thus, obtaining a better understanding of public perception is of vital importance.

And what is the strategic value of doing such an analysis?

First, it can serve as another metric to monitor the brand's perception. Also, understanding the overall themes and commonalities behind low sentiment posts can lead the hotelier to fix commonly occurring problems. This can help raise metrics like NPS (net promoter score).

What are some examples of how the hotel could make the results of the sentiment analysis actionable?

Results of the sentiment analysis would primarily help make sure customer satisfaction remains high. There would be a focus on identifying dissatisfied customers, and then making sure they feel understood and accounted for. One way to do so would be to proactively reach out to posters of negative posts on social media platforms and offer them refunds if the hotel or employee was perceived to be in the wrong. Another would be to investigate the causes that motivated negative posts and make necessary modifications to address them. Properties that were the source of the negative feedback could be alerted, in close to real time, about the problem so that changes could be made to address the issue.

How would you do sentiment analysis?

Since sentiment analysis is such a common problem, it'd be worth trying to find existing APIs or packages that can do this task, like NLTK and TextBlob. You could also use transfer learning, since there are great off-the-shelf NLP models that can be extended and tuned from places like spaCy and Hugging Face.

What preprocessing would you do with the data?

Since most text online is messy, basic text preprocessing techniques like fixing text encoding, stripping away HTML, removing stopwords, stemming or lemmatization, and finally, vectorization, would likely be needed.

Can you describe some ways of turning the review text into numerical features that can be used for machine learning models?

Sure, the general process is often called text vectorization.

There are several possible text vectorization methods:

- **Bag-of-words:** represents text by counts of the words comprising it
- **N-grams:** allows analysis of text by identifying contiguous sequences of n words or letters present within the text
- **TF-IDF:** an abbreviation for frequency-inverse document frequency, TF-IDF is a technique that indicates the importance of a word within a single document, when compared to the importance of the word in a larger corpus

How would you then run a model using those text vectors?

Following whatever processing technique is used, each piece of content is represented by a vector containing the results of the step above. At this point, a variety of classification methods can be used to classify the sentiment underlying the text, including logistic regression, random forests, kernel-based methods such as support vector machines, discriminant analysis, and neural networks. Finally, the effectiveness of our classifier can be assessed through a confusion matrix and computation of metrics such as precision and recall.

Good job explaining how to categorize reviews based on sentiment. Let's switch gears to topic classification. Why might categorizing the reviews into different topics be useful to the business?

Categorizing reviews into different topics can be useful since feedback can be more efficiently routed to the different subdepartments of the hotelier. For example, if you could automatically split reviews into topics such as check-in experience, room quality, or room service, the corresponding teams like the front desk, housekeeping, or kitchen could more efficiently act on customer complaints.

How would you go about grouping the reviews into categories?

To do the grouping, we can do some basic clustering on the content based on vector representations, using an algorithm such as k -means. Alternatively, we can model the text-related data over an underlying set of topics using Latent Dirichlet Allocation (LDA). LDA assumes each post to be a distribution of topics and that each topic is a distribution of words. Ultimately, we should be able to characterize posts and group them into appropriate buckets, i.e., as expressing various sentiments regarding the brand.

Solution #11.6

Facebook - People You May Know

Suppose you are to build out Facebook's friend recommendation product, also known as its "People You May Know" (PYMK) feature. How would you go about doing so?

Before we begin --- just wanted to clarify -- what is the goal of this feature? Is it to maximize friend count for existing users? Or is it to drive engagement on the product -- whether that be a shorter- or longer-term basis?

Great questions. Let's say that we want to increase the number of meaningful connections formed. How would you go about building the PYMK feature for this goal?

Got it! Okay...well, two approaches are possible. The first involves recommendations based on uploaded contact information. A signup flow typically asks a user to import their phone contacts or email address book (especially important for new users) and then recommends you to friend the

people within your contacts who are on Facebook. You could offer friend suggestions to the people in the contact book, to friend the person who just signed up. The second approach to recommendation involves leveraging a user's current social graph, recommending friends of current friends. Likely, we'd want to use a blend of both approaches.

Let's talk more about the second approach — how would you go about leveraging a user's social graph for the PYMK feature?

We can rank the potential friends for any given user based on the social graph. For example, you can make a candidate list of all second- and third-degree connections. Then, we can rank this candidate list based on the likelihood that the user and the candidate become friends.

What are some features you'd use to measure the potential for two people being friends?

To look at the strength of potential friendship between two people X and Y, we can start by simply looking at how many mutual connections they have. To generalize this concept, the more two users have in common, the more likely they are to have a potential friendship. Signals like:

- **Profile Similarities:** age, alma mater, employer, hometown, current city, mutual friend count
- **In-App Activity:** high engagement with similar friends, attending common Facebook events, visiting each other's profile pages, commenting on the same post, being tagged in photos together
- **Ecosystem Signals:** being connected on Instagram, or messaging on Messenger, Instagram DMs, and WhatsApp
- **Off-App Signals:** email, phone number, mobile GPS location

Co-occurrences of any online presence would serve as signals for PYMK.

Can you give examples of specific methods you would use to rank potential friends for a given user?

We could use classification algorithms, like logistic regression or naive Bayes, to predict for a given user the likelihood of a friendship with another, non-friend user. That is, the target variable is whether they will become friends or not, and we use the various features aforementioned to train a model to learn that relationship.

What about unsupervised techniques for ranking potential friends?

Unsupervised techniques such as K-means or principal components analysis (PCA) can identify similar users who are not yet friends with that user, and then those non-friends that overlap to the greatest extent with the user's existing friend base are ranked highest. By "overlap," we mean they have the most mutual friends in common. In both cases, for any given user, we end up with a set of rankings for potential friends.

Does the model setup you described pose any potential problems for new users?

One major problem with our model setup would be assessing new users, especially PYMK (People You May Know), who choose not to upload email and contact information. In the machine learning setup described above, relevant rankings of PYMK would be difficult since there is a dearth of appropriate data to feed into the algorithm in this case.

Why do you think — from Facebook's perspective — new users are important?

Facebook is a social product whose main value to a user is realized only after that user has added a sufficient number of friends; otherwise, that user's feed is practically empty. Facebook differs from other social media platforms such as Reddit or YouTube, where much content is available for

consumption by new users without them having to form friendships first. This is known as the cold-start problem — you need supply (friend content) to get demand (usage by new users), but can't get friend content if new users churn out quickly and never make friends. Thus, helping new users make friends quickly is strategically important for Facebook. A job well done here would significantly improve the new user onboarding process and reduce new user churn. This keeps the social graph the best in the business, which is one of the strategic moats Facebook has against other social products.

What are some product ideas you have to help new users make more friendships? Both PYMK and non-PYMK related ideas are okay — just want to brainstorm with you.

I love product brainstorming. Some ideas that come to mind are:

- **Boost New Users:** Boost the odds that new users appear in existing users' PYMKs, thereby increasing a new user's inbound friend requests.
- **Friend Chaining:** If a new user accepts all their inbound friend requests, instead of leaving the surface empty, show PYMK there to keep the friending going.
- **Get More PYMK Signal:** Existing users simply ignoring or removing new users suggested in their PYMK recommendations could also be a source of training data for the PYMK algorithm.
- **Increase PYMK Units:** Show more PYMK news feed units in their first two weeks unless they hit a certain number of friends made.
- **Use Gamification:** Add a progress bar to push new users to make a certain number of friends. This progress bar can also incorporate other important steps like uploading a profile picture and filling out the new user's bio details. By pushing for people to have fleshed out profiles, outbound friend requests from new users are more likely to be accepted.

Solution #11.7

Stripe - Loan Approval Modeling

Assume you are working on a loan approval model for small businesses. What metrics would you use to evaluate the model?

First, can I know more details about the loan approval model? Are we approving businesses for their requested loan amount, or are we instead recommending an upper credit limit that we could comfortably loan out to a business?

Great point — let's say that businesses apply to us with a fixed loan amount in mind.

Cool ~ another question then. I'm curious; is this a binary situation — a loan is either approved or not, and a business pays back the principal in full or not? There aren't any half-repayments or debt collection to factor in?

For simplicity, we will ignore those cases and just assume a binary response — loans are paid back in full or defaulted on. What metrics would you use to evaluate this loan model?

Our loan model can produce a probability score for whether a particular loan application will default or not. So assuming some threshold (say 0.5), then each transaction can be classified accordingly. That is, if the score is greater than or equal to 0.5, then the application is classified as likely to default; if it is less than 0.5, then it is classified as likely not to default.

To evaluate the model, we look at precision, recall, and the corresponding precision-recall curve. Note that we wouldn't look at accuracy since this is a highly unbalanced problem, as likely only a small percentage of loans will be defaulted on.

What do false positives and negatives mean in this context?

A false positive is when the model predicts the loan would default when, in fact, it did not. A false negative is when the model predicts that the application will not default when, in reality, it does. Thus, higher precision and recall indicate smaller numbers of false positives and false negatives, so this would minimize the numbers of defaulted loans and maximize the number of valid loans that Stripe has to deal with, thereby improving bottom-line profit.

From the point of view of the business, should both false positives and false negatives be weighted equally? Why or why not?

A false negative in this context means a loan was made when the user actually defaults, and so the loss incurred would be the entire principal amount. For simplicity, I'll ignore how debt collection agencies could technically recoup some part of the principal, and just call it a total loss of the principal (i.e., a "write-off").

A lesser monetary issue would be false positives, where no loan was issued when it should have been. For simplicity, assume a 10% interest rate on the loan, and all the interest is pure profit. That means for each false positive, the business loses out on the opportunity to make 10% of the loan principal. As such, for this case, Stripe incurs a 10:1 cost for false negatives to false positives. We can use this ratio to evaluate our classifier by trying various thresholds and assessing the weighted precision and recall that would result from their use by constructing a precision-recall curve (or an ROC curve if the false positive rate rather than the false positive is used).

Would relying on this type of model produce any edge cases, especially under scenarios having increased uncertainty?

The status of some loans would not be clear even if the model flagged them as likely due to default, possibly due to regulatory issues, and so would require human review. These could involve the size of the loan being requested or some aspect of it that appears questionable. Conducting manual reviews on "borderline" cases where the model generates a likelihood of default that is barely above or barely below the threshold would also be a wise precaution and could be part of an overarching tiered system, in which the model's rating comprises the first layer and determines how much additional human review of the loan is needed.

Overall, implementation of such a multilayered system could further reduce the numbers of false positives and false negatives the Stripe accepts and acts on. Note that the cost of manual interventions should be factored into the total cost calculation — human intervention is a good way to avoid costly mistakes, but in itself is also costly.

What are some features you would recommend incorporating into the model described above?

Model construction would involve two feature dimensions. The first is at the loan applicant level and includes the applicant's demographics (assuming it is legally compliant and ethical to do so), IP address, browser, and financial health: bank account balance, credit worthiness, whether there are any outstanding liens or judgements.

The second set of features is at the loan application level and includes answers to questions such as the following: how complete (and reasonable) were the answers provided on the loan application, how much money is the business asking to borrow, what is the purpose of the loan, and how much are they putting up for collateral?

What other methods would you recommend using to improve the model?

One way to improve the model would be to utilize reject inference, which is based on the idea that not accounting for rejected applications introduces bias into the generating model. Models of loan defaults are generally trained only using data from previously granted loans, thereby introducing sampling bias into the modeling process. In contrast, reject inference involves using another model that has been trained using rejected-application data. Both the original model and the reject model can then be used in tandem to obtain a final score.

Additionally, we could integrate anomaly detection along with the model. What makes people default on loans may change over time, yet models are trained on historical examples of what encompasses a loan default. By supplementing the model with anomaly detection, we can flag more odd cases in real time. However, anomaly detection isn't a silver bullet. With the large feature space and volume of data points coming in, there will almost always be at least several anomalies on any one given dimension, due to the curse of dimensionality.

How could you test that your new loan approval model was better than the baseline model?

You could run an A/B test, but it may be difficult because of the long time horizon of loan repayments and having a business default on a loan. For a simpler way to test, you can do an offline model comparison on the various metrics, and look at a paired t-test between the two models.

Say you did run an A/B test on the new loan approval model, and you find that the revenue from loans increases, but the p-value is 0.06. What would you do?

Because there can be a fair amount of noise with p-values, we shouldn't hastily jump to the conclusion that the test is not statistically significant since $p > 0.05$. It's likely that with a larger set of data, the p-value would be different. Therefore, it would be best to run the test longer to observe any drift in the p-value in order to get clues on longer-term behavior. Additionally, we should look at the effect size of the revenue and the changes in counter metrics. This would help us get better insight into the quality and impact of the experiment before any launch decision can be made.

Solution #11.8

Instagram - Ranking for Instagram Explore

How would you provide content recommendations for Instagram Explore?

Before we begin, just wanted to make sure I understand the Instagram Explore feature. This isn't your main newsfeed, where you see posts from accounts you follow, right? This is a surface where you can see a feed of customized photos and videos — often from accounts you don't follow — that is continually refreshing, correct?

Yep, you've understood Instagram Explore. Any other questions?

I'm curious about any SLAs for our system, and the scale we are dealing with. I know that Instagram has many hundreds of millions of users, so there are probably many billions of pieces of content on Instagram that are relevant inventory for Explore. On top of that, I'm wondering how real time this needs to be? I'm guessing we probably want to serve the most relevant content in real time for every user. With 1 billion Monthly Active Users, we might have to support 1 million concurrent feed refreshes per minute? Does that sound reasonable?

That sounds reasonable — let's go with your assumptions. So, what would be your high-level approach for providing recommendations for Instagram Explore?

At a high level, we can focus on identifying accounts that have content a user would find interesting, rather than at the media (content) level. Said another way, we use a collaborative-filtering-style approach by recommending content from accounts that are similar to the ones the user interacts with, rather than recommending content based on topics.

We take an account-based approach because the universe of possible media is very large, and there is a lot of new content coming out every second. By doing recommendations mostly based on the account level, and then pulling the most recent or most relevant media from the recommended account, we can do a first pass for candidate retrieval. This helps to narrow down the universe to a manageable subset that we can then rank for each user.

What features would you use for candidate retrieval?

We can come up with an “embedding” per account, which treats account ID’s that a user interacts with (e.g., a person likes media from an account) as a sequence of words in a sentence, analogous to word2vec. We can call this an “ig2vec” embedding. Under this setup, if an individual interacts with a sequence of accounts in the same session, it’s likely to be more topically similar to those accounts than compared to a random sequence of accounts on Instagram, which helps with finding topically similar accounts.

As an alternative approach, we can build a matrix that stores interactions of users and other IG accounts, and a factorized version of this matrix can be used to explore account similarity.

How does the model utilize those features in order to come up with candidates?

We can first calculate a distance metric between two accounts using either cosine distance or dot products. Then we can use KNN to find similar accounts for any account in the embedding and train a classifier to predict a set of accounts’ topics based on that embedding. By retrieving accounts similar to those that a particular person expressed interest in, we narrow down the universe of content to a smaller personalized ranking inventory.

What models would you use for the ranking step?

We can consider a variety of classification models. At Instagram scale, training a neural network seems appropriate. Rather than having a binary outcome (such as recommending the post or not) to reflect the variety of options a user has with regards to a post, we can treat this as a multi-class classification problem. For each post, we can predict the probability that it is liked, commented, shared, hidden, and reported by a user. These probabilities can be weighted to come up with a final recommendation score. The weights for the actions can be defined based on a simple statistical analysis that links each action to some top-level KPI, like user engagement. This approach gives us the flexibility for downstream stakeholders to tune what goals they want from the system.

Should this model be deployed in batch or online?

We would want to run it online since the predictions have to be real time and include users’ most recently used activities. For online, feature engineering needs to be optimized, since at inference time the features need to be plugged into the model with low latency. Additionally, we want to make sure in this setting that the features, which may come from a variety of data sources, can be precomputed and stored in a real-time storage environment and readily accessible. For example, while most features are stored in batch in HDFS, it is not possible to query directly very quickly, so the features should be stored in Cassandra or an analogous service.

What are some challenges and what rollout strategy would you use at inference time?

In terms of rollout strategy, we can choose between the following:

1. a single deployment, where all users see the changes directly
2. controlled deployment, where a smaller subset of users see the new model and the majority see the previous model
3. “silent” deployment, where both are deployed but users do not see the predictions of the new model
4. “flighting” deployment, where you can run online A/B tests.

The first is the simplest of the four, but costly if there are mistakes; the second can be complex to implement, and the third doesn’t allow for seeing how users react to the model. The fourth is the best fit for this particular use case.

How would you assess model performance over time?

The general problem is that stale models cannot capture changes in user behaviors or understand new trends. In terms of quantifying model drift, we can (1) monitor model performance over time on a frequent basis, and (2) look at KL-divergences — a measure of how similar two distributions are — in key behavioral distributions of the models as well.

Can you describe what KL divergence is?

KL divergence measures the similarity in distributions — since machine learning algorithms are parameterized by the distributions of input data, we can use KL divergence to check the distributions of the input features over time. If the feature distributions are changing significantly from when the features were launched, then we want to retrain the models. Note that in practice, simply retraining the models on a periodic basis could be an easier approach than monitoring each feature’s KL divergence over time.

Say that you ran an A/B test on your improved model versus a baseline model and did not see any correlation between the performance of the model and specific business metrics of interest (engagement). Why might this be?

There are several possible angles here, stemming from the various parts of the experiment.

There could be some over-optimization of these specific metrics of interest, which leads to saturation of metric improvements from further model improvements. For example, it could be that the A/B test metric might be ranking relevance, BUT since the content is already relevant enough for folks, it’s not strongly correlating with increased user engagement any more.

On the user side, it is possible that model performance past a certain point may have a negative effect on user experience. For example, the recommendations are too niche or specific, or people get weirded out by the model, and then don’t use the product because it’s creepy. In that case, an improved model may lead to decreased engagement.

This might not seem reasonable, but how many times have you heard the false conspiracy theory that Facebook and Instagram ads are hyper-targeted because they listen to your conversations through your phone’s microphone? It is possible for a recommendation to be too good!

AFTERWORD

Thank you for reading Ace the Data Science Interview!

If you've got any questions, feedback, or praise, please reach us at:

hello@acethedatascienceinterview.com

instagram.com/acedatascienceinterviews

You should also connect with us personally!

hello@nicksingh.com

linkedin.com/in/Nipun-Singh

instagram.com/DJLilSingh

twitter.com/NipunFSingh

kevin.w.huo@gmail.com

linkedin.com/in/Kevin-Huo

instagram.com/Kwhuo

We couldn't end this book without thanking the many amazing people who've supported us!

Thank you to the people who've gone out of their way to help us in our careers:

Greg Mand, Auren Hoffman, Evan Barry, Lauren Spiegel, Jonathan Wolf, Ross Epstein, Shaxun Chen, Sheng Zheng, Josh Buffum, Riju Kallivalappil, Kristie Moi, Parth Detroja, Aakash Shah, Ripley Carroll, David Booth

Thank you to our friends who've supported us on the book:

Adam Rosenberg, Rohan Raval, Himanshu Ohja, Neeraj Gandhi, Brandon Hohenberg, Elakian Kankaraj, Atul Nambudri Quinn Li, Anisha Marya, Naveen Iyer, Vijay Edupuganti, Kristina Hu, Mayank Mahajan, Alex Wang, Andrew Jiang, Mat Samuel, Kara Sheldon, Noah Yonack, Jaina Mehta, Rucha Bhat, Leo Agnihotri, Shivangi Mistry, Sri Vasamsetti, Danil Kolesnikov, Vishnu Kumar, Harnoor Singh, Lawrence Hook, Salomon Lupo, Mehar Virdi, Richard Liu, Philip Ruffini, Nimisha Jain, Hamza Khawaja, Hari Devanthan, Shota Ono, Samraaj Bath, Ashley Belfort, Patrick Rivera

And thank you to everyone in the Data Science community who volunteered their time to read early versions of our book, or since publishing, have found mistakes or typos we've corrected:

Advitya Gemawat, Jeffrey Ugochukwu, Lars Hulstaert, Michelle Scarbrough, Nicholas Vadivelu, Catherine Yeo, Jordan Pierre, Rayan Roy, Jack Morris, Neha Pusarla, Aishwarya Srinivasan, Siddhartha Sharan, Daliana Liu, Marco Sandoval, Timothy Wu, Prithika Hariharan, Tania Dawood, Faith Chung, Jai Raghuvanshi, Devan Shanker, Lindsay Warrenburg, Jie Cai, Alice Hau, Jon George, Harris Vijayagopal, Thiru Veeran, Sourabh Varshney, Ayan Sengupta

And thank you to our editors and book creation gurus:

Wendy Martindale, Rajiv Kumar, Mykola Shelepa, Mary Graybeal

Made in the USA
Las Vegas, NV
07 April 2022



47068759R00168

