High Performance Computing in Journalism

Leon Yin
leonyin.org

Leon Yin is an award-winning journalist at Bloomberg News. He builds datasets and develops methods to investigate technology's impact on society.

One surprising (or not so surprising) application of high performance computing is journalism. Investigative journalism is a type of journalism that focuses on digging deep into an impactful, longer-term topic involving titans of industry, government institutions, and the powerful.

Investigative reporters rely on "shoe leather" reporting techniques to speak directly with affected individuals, as well as Freedom of Information requests to review government documents. Increasingly, reporters are embracing the affordances of technology to collate disparate data sources to unearth evidence of harm at scale.

Still Loading

One of America's most-notable monopolies was AT&T Bell Labs. Between 1913 and the 1980s, Bell Labs owned practically every telephone line connecting households across the United States. In 1982, AT&T was ordered by the US Department of Justice to split up its monopoly into several independent regional companies. These "Baby Bells" were subsequently acquired, merged, and rebranded as Verizon, AT&T, and CenturyLink (now called Lumen Technologies).

In addition to being mobile carriers and offering landline services, these three companies are also major internet service providers. In The Markup's award-winning series "Still Loading", we found that AT&T, Verizon, and CenturyLink disproportionately offered slow internet speeds to low-income, non-White neighborhoods for the same price as fast fiber in other parts of town.

Our investigation was the first of its kind to show the scale of digital redlining practices in the United States. Across the 38 major cities served by AT&T, Verizon, and CenturyLink, we found disparities based on income or race in all but two cities. Data from our investigation was used by local newsrooms to report on internet disparities across the country, cited by policymakers to combat digital discrimination, and used as teaching materials for data journalists.

Slow Speeds and Quick Iteration

When you're shopping around for internet deals, you may have visited a web provider's website and typed in your home address to see what plans they might have for you. Our data collection effort attempted to replicate this process for over a million addresses.

Our methodology was inspired by an academic article by researchers from Princeton University that found discrepancies between what internet service providers self-reported to the Federal Communications Commission (FCC) and what they advertised to customers. The researchers built scrapers for several different internet service providers' websites to collect internet offers. Although their code was public, it no longer worked, as many of the sites had changed their user interface.

This is a common occurrence, as scrapers break all the time. To save time, I try to develop a quick proof of concept analysis before investing too many resources into an investigation. It's advantageous to test and eliminate potential projects quickly; this allows us to gauge the potential of many stories and decide how to proceed. During this step it's key to build a data pipeline that is reasonably functionable and select a small sample to indicate potential patterns.

As a reporter, this also means that I interview experts to get up to speed. For this project I contacted an author from the paper to discuss their findings. I asked whether a particular provider was egregious, and where. Repetition is often a good starting place, as it provides a blueprint to reference and build off of.

Researchers were quick to point to AT&T as an internet service provider with many discrepancies, especially in the city of Green Bay, Wisconsin.

This gave me direction for a scraper to start building. When I need to build a scraper I go to the website and get intimate with the user interface and the workflow necessary to manually retrieve whatever information I'm after.

The user interface for AT&T's internet service lookup website involved a multipage process of filling in an address, filling in an apartment number, and then selecting whether you're shopping for Fiber or DSL.

I noticed that an HTTP request to a server occurs for each of these steps while I was listening for network requests in my browser's developer tools. I typically opt for finding such undocumented APIs for web scrapers; however, because the process was multiple steps with multiple APIs, I was unable to successfully combine each step to look up the internet plans for a given address.

Instead of immediately figuring out these APIs, I quickly developed a Selenium scraper to fill in these steps. Selenium and other such browser automation tools (such as Puppeteer and Playwright) are great when you need to interact with rendered elements on a page as a user would.

It's also something that is easy to debug and deploy locally on a PC. For a few thousand random addresses from Green Bay, the process was painfully slow. This is partially because browser automation loads and renders all the elements and scripts on a page, and also the website deploys some sort of traffic control that will throttle requests from the same computer (called IP blocking). Also, in order to do this Selenium needs to run a full browser instance, which takes quite a lot of resources.

Even with multiprocessing to open 9 Selenium browsers to process 9 addresses in parallel, the process was slow. It took two weeks to collect 4,000 addresses. I initially viewed this as a nonstarter for anything more ambitious involving multiple companies or cities.

But I am a reporter first, and an engineer later. Why optimize code for a story that isn't spicy? However, looking at the data suggested we had an important story to report out.

After parsing and cleaning the scraped data, we plotted each internet plan on a map in Figure 13-1, and immediately we saw clusters of fast speeds above 100 Mbps (green) and slow speeds (orange) below broadband benchmarks.

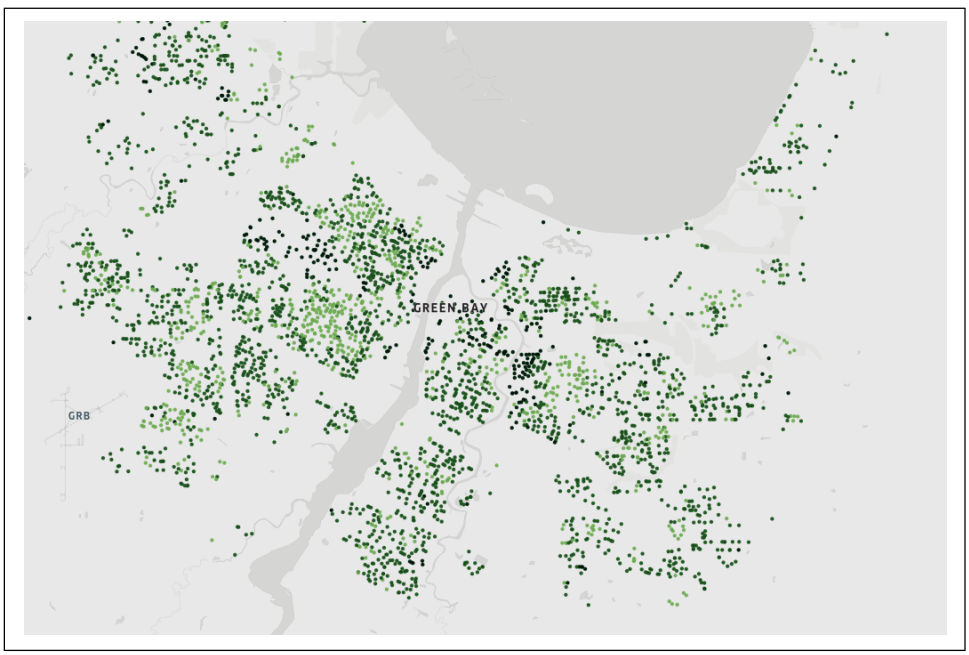

Figure 13-1. AT&T's internet speeds across a sample of Green Bay, Wisconsin, addresses, mapped using Kepler.GL

Geography is correlated with income and other socioeconomic factors. Every year, the Census Bureau conducts the American Community Survey of a select sample of the population and uses that data to estimate the median household income and racial makeup of every city in the United States.

We merged the median household income to each internet plan to create Figure 13-2, and using an extremely rough categorization system for income, we found AT&T was more likely to offer low-income residents slow speeds, and less likely to offer fast speeds compared to wealthier areas.

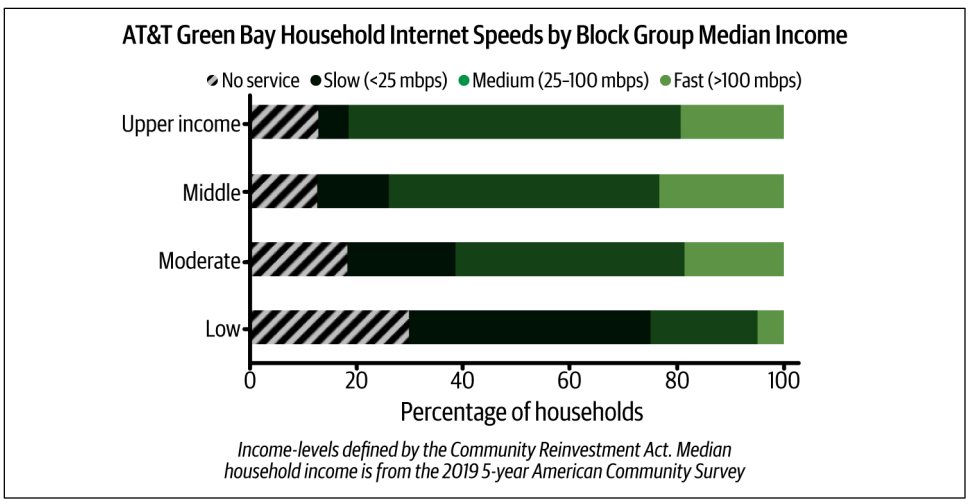

Figure 13-2. Normalized bar charts of AT&T internet plans by median household income

Despite taking two weeks to build the initial dataset, these findings immediately told us we have a story about inequity. The strangest part was that all these internet speeds were quoted at $55 a month.

We soon found a report by the internet advocacy group National Digital Inclusion Alliance (NDIA), which accused AT&T and Verizon of charging the same base price for a variety of speeds based on marketing materials, which they coined "tier flattening."

This made us wonder: How many companies practiced tier flattening, and how many states or cities did they operate in? How could we collect more data if the initial process was so slow?

We manually confirmed that AT&T, Verizon, CenturyLink, and EarthLink all practiced the same pricing system and that they served 45 states and Washington, DC.

Back-of-the-napkin math that limited our universe to a 10% sample of the most populous city in each of these states yielded 1.6 million addresses. At the current rate, that would have taken almost 15 years to finish.

Although we initially used Selenium to scrape internet plans, 15 years is not a reasonable timeline. Of course, when "interactivity" is a requirement, browser automation is a clear winner that is easy to program and debug.

In our initial appraisal of AT&T's website, we saw APIs running to serve the information in the service lookup tool as well as return internet plans via API calls. Unlocking this route would allow us the speed and scale we needed, and a quick inspection of each other provider's website revealed almost identical lookup tools for internet plans.

How would we keep track of the state between each API call? You can't list plans without reference to the input address, and no clear parameter or header keeps track of this. Luckily, my data editor Jeremy Singer-Vine introduced me to using a `session` object, which does exactly what we need: keep track of headers and cookies across requests.

I used a session to string together a series of API calls to simulate the workflow for listing internet plans for an address, which I repeated in four scrapers for each internet provider. An example of this can be seen in Example 13-1, where I collect internet plans from AT&T using multiple API requests and a single requests `session`.

Example 13-1. Getting internet plans synchronously

```
from requests import Session

def get_internet_plans(address, proxy):
    """
    Collect internet plans for one `address`.
    `session` persists cookies, `proxy` is for routing requests
    """
    with requests.Session() as session:
        session.get(url='att.com/authenticate', proxies=proxy)
        address_id = session.get(
            url='att.com/autocomplete',
            proxies=proxy,
            json={"address": address}
        )
        plans = session.post(
            url='att/com/plans',
            proxies=proxy,
            json={'addressId': address_id}
        )
        return plans.json()
```

This is already significantly faster than what we had accomplished with Selenium. But API calls are scalable in that they can be done either in parallel or asynchronously. With a few changes, the scraper function can be retrofitted to be called asynchronously while choreographing each step (API call) to occur in sequence (Example 13-2).

Example 13-2. Getting internet plans asynchronously

```
import aiohttp

async def get_internet_plans(address, proxy):
    """
    Collect internet plans for one `address`.
    `session` persists cookies, `proxy` is for routing requests
    """
    async with aiohttp.ClientSession() as session:
        await session.get(url='att.com/authenticate', proxies=proxy)
        address_id = await session.get(
            url='att.com/autocomplete',
            proxies=proxy,
            json={"address": address}
        )
        plans = await session.post(
            url='att/com/plans',
            proxies=proxy,
            json={'addressId': address_id}
        )
        return plans.json()
```

One of the issues that Princeton researchers warned us about was rate limiting and IP blocking. We used a residential IP address rotator that routed our requests through someone else's computer. We used the same provider that the Princeton researchers used. IP routing is a last resort that you should use responsibly. We listened to server status codes and scaled accordingly so as not to crash any websites.

Using undocumented APIs and asynchronous programming, we were able to collect internet plans for 300,000 addresses daily, whereas a parallelized Selenium scraper could only collect around 300 addresses a day. This process of developing these new scrapers and optimizing them took about one week.

Unlocking technical barriers allowed us to be ambitious, showing the scale of disparities in an unfair pricing system for an essential commodity. We found that neighborhoods that were offered the worst deals had lower median incomes in 9 out of 10 cities in our analysis. In two-thirds of the cities where we had enough data to compare, the providers gave the worst offers to the neighborhoods with the most non-White residents. Internet access is not considered a utility in the United States and thus is not regulated as strongly as electricity or water.

But before we had our "Big Story," we were quick, iterative, and pragmatic. We read the prior art and spoke with people to narrow down the complexity. Early tests showed great promise, while thoughtful engineering and high-performance computing provided a full solution that allowed us to reach the potential that this impactful topic deserves.

Data from our story was used to publish original reporting from nine local newsrooms and cited by Los Angeles lawmakers who passed the nation's first ruling against digital discrimination. The series was honored with several journalism awards, including the Philip Meyer Journalism Award from Investigative Reporters and Editors, which recognizes the best use of social science research methods in journalism.

Although these lessons are from investigative journalism, I hope that these same principles can be applied to the projects that are important to you.

Read more at:

- "Dollars to Megabits, You May Be Paying 400 Times as Much as Your Neighbor for Internet Service"
- "How We Uncovered Disparities in Internet Deals"
- "How We Uncovered Disparities in Internet Deals: GitHub Code and Data Repository"

Lessons from the Field of Cyber Reinsurance

James Poynter
linkedin.com/in/james-poynter-0b295375

James Poynter is employed as a Global Data Science Lead at reinsurance broker Gallagher Re. He has a track record of applying data and technology to solve commercial (re)insurance risk and underwriting problems.

As a data scientist, when I think of high performance, I like to think in terms of data flows—I dream of a fast, smooth, and efficient flow of data through a system of robust machine learning and data pipelines. Each ML system comprises a series of steps, and each step involves inputs, transformation, and outputs, with machine learning (ML) model training and inference being just another transformation step (albeit a complex one).

When it comes to unlocking business value from data science and machine learning systems, data must come in and high-quality actionable insights and information