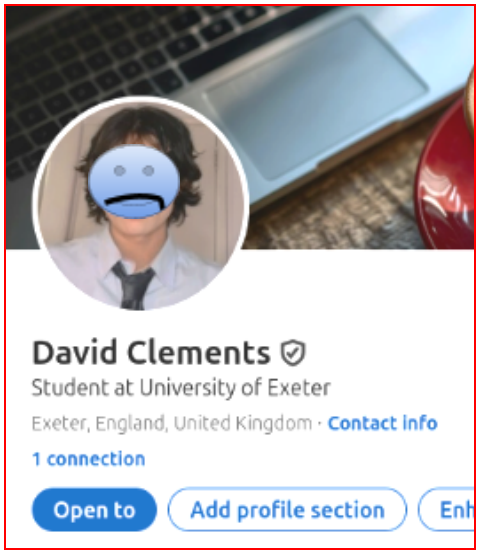
Autoconnect | How I automated my online presence

Hey this is me

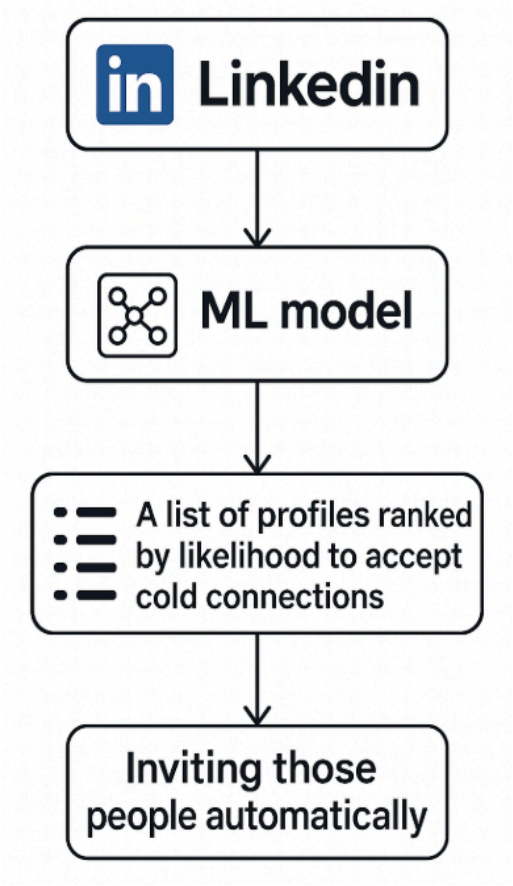
That was my LinkedIn profile ***1 lonely connection :(***



I don't really like using LinkedIn, but I've heard it may potentially help me land a job/internship if I have a significant enough presence on it. I've heard whispers that recruiters actually use it.

So I was facing a dilemma: use this social media platform I dislike, or be hindered in the race to secure a job on graduation… In the 2025 hellscape graduate job market not using it isn't an option, HOWEVER I realised, I don't have to be the one using it – Enter Automation!

Outsourcing my LinkedIn



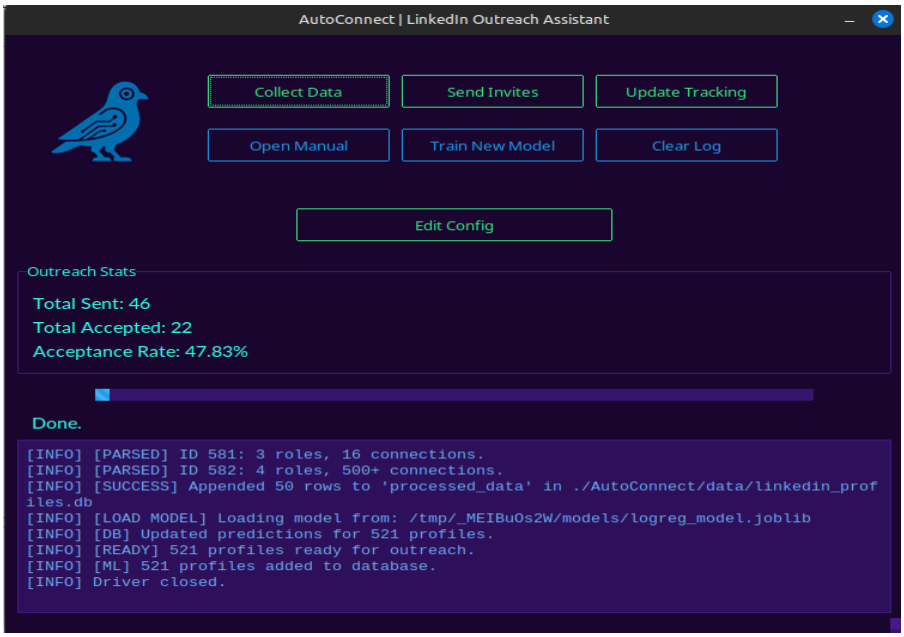
I built a bot that scrapes LinkedIn profiles, predicts who's most likely to accept a cold connection using a simple ML model, and automatically sends invites — all wrapped in a one-click GUI. It's packaged as a Linux AppImage, so it's as simple as downloading, and clicking on it if you'd like to try it for yourself, granted you have to use Linux. Windows is coming later, if I figure out how Cross-compiling with Wine in Docker works.

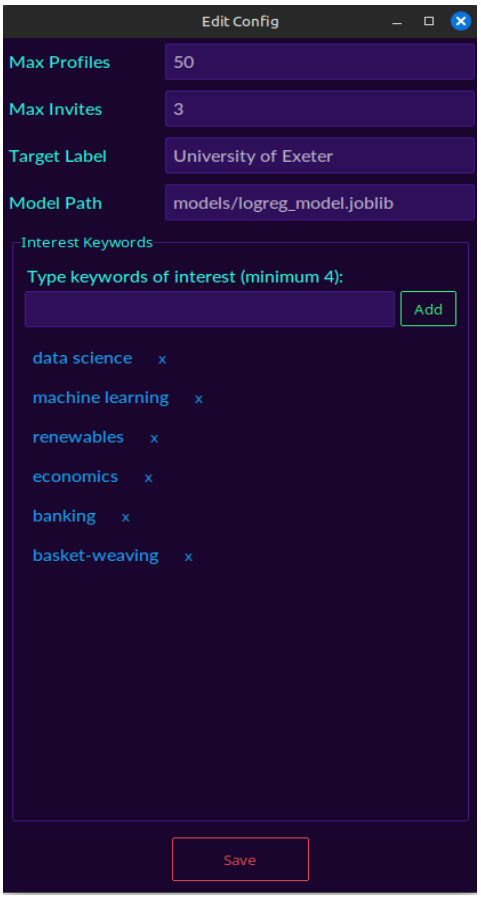
(this isn't an encouragement to violate LinkedIn Tos this is purely for educational purposes)

DOWNLOAD FROM HERE (SEE RELEASES TAB)

The visual on the left distills that process: from scraping, to ranking profiles by predicted acceptance, to sending out invites — all done automatically, operated with a GUI. Throughout this post, I'll walk you through how I built it, the insights I discovered, and the hurdles I hit along the way. For the more technical details (code, data, and setup), please checkout the GitHub repo | github repo!

Preview | GUI : main dashboard + config editor





Primary Functionality:

“Collect data”:
Scrape data
Process it and apply ML predictions
“send invites”:
Send n invites to the top n probability predicted profiles
“Update Tracking”:
Recalculate acceptance rate and Total Accepted stats
“Train New Model”:
Using data from previous scrapes and invite runs, if you have enough data, this button trains a new model and houses it in a dynamically created folder models/ which you can then use by editing configs
“Open Manual” : Opens user manual bundled into the linux binary
“Clear Logs”: Just clears the log at the bottom of the main GUI
“Edit Config”: Edit outreach and scraping instructions to the bot

Now these all work, granted in not the most efficient way especially update tracking.. But they all work as of testing them now

Gathering Data | Where do I even get the data?

The first step was figuring out where on LinkedIn I can get access to large swathes of profile data, my first thought was search bar, however I quickly found out that this section only shows 1st order and 2nd order connections (someone you've added, people who the people you know have added), my one connection was my housemate Alex, who wasn't doing much better than me at 6 total connections. So my broad network from the search bar consisted of 7 people, nowhere near enough for a baseline ML model. Because of this I pivoted my strategy towards the "People you might know from _____ " section. This is LinkedIn's profile recommender system effectively, allowing you to reach people outside of your immediate connections, although they appear on the whims of some LinkedIn algorithm.

Here you can open a tab for a particular organisation or even industry and scroll and scroll until you hit about 1000 recommended profiles… Perfect. But this brings me onto one of the main challenges with scraping LinkedIn, well there's two.

Obstacles in data collection

Lazy Loading

LinkedIn relies on a system called lazy loading, simply this means that parts of the page don't load unless there's user interaction, in order to get those 1000 profiles to appear you have to be constantly scrolling and interacting with the page, triggering more profiles to load. This methodology is used across the whole site, which means nothing is as simple as loading up the page and extracting all the data, everything has to be done as a chain of actions, that all have to fit together perfectly!

Account Flagged for suspicious behaviour"

LinkedIn also has sophisticated anti-botting systems, something I quickly learnt about when my first dummy account for data collection was nailed with the ban hammer, if you move from page to page too quickly, interact too little, scrape too aggressively, are too spammy with your number of requests and messages you're account is banished.

How to avoid detection | pretending to be a person

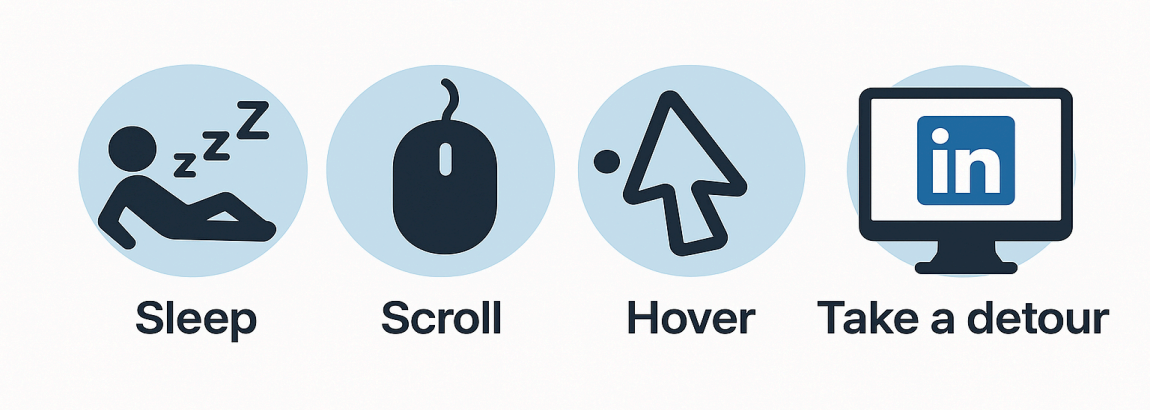
I built this tool with long-term use in mind. The goal was to make something you could run daily, safely, and without getting flagged *(HYPOTHETICALLY IF YOU WERE SUCH A HORRIBLE PERSON THAT YOU WOULD VIOLATE LINKEDIN TOS WHICH IS SOMETHING I DAVID CLEMENTS DO NOT ENCOURAGE OR SUPPORT IN ANYWAY)*; as long as you don't push it too hard. That does mean it's a bit slower when scraping or sending invites, but that's just the tradeoff when dealing with bot detection. The default settings are cautious by design, this may or may not be part of my master plan to get a job after uni, if I was to hypothetically benefit from repeated violation of LinkedIn TOS I would prefer to not be banned for it.

Here's how I've kept it under the radar so far:

Cookies & Chrome Profile Management
LinkedIn doesn't like it when you log in too frequently, too fast, or from too many "devices." A typical Selenium setup opens a clean browser profile every time, which LinkedIn sees as a brand-new device. That's a fast way to get flagged. To avoid that, the app reuses the same Chrome profile between sessions. It also captures the li_at session cookie the first time you log in and automatically reuses it after that. No need to log in again manually, and the behavior stays consistent with what LinkedIn expects from a real user.

"Random Skip" - simulating choosiness in outreach`
People don't send invites to every profile they see, and bots that do stand out fast. To help with that, the outreach script skips around 1 in every 5 profiles at random. This makes it look like you're being selective instead of just blasting requests, which helps it blend in better.

Helper Functions throughout the code
To make every interaction look human, I added small helper functions that scroll randomly, hover over things, pause here and there, or take quick detours to unrelated pages. It's basically fake browsing. These small behaviors help make the automation less obvious and reduce the chances of LinkedIn catching on.



Feature Engineering Hypotheses | Who will accept?

Using the detection-avoidance tricks mentioned earlier, the pipeline is able to extract and parse the raw HTML from each profile. At this stage, I had the full text of every profile — but raw text isn't immediately useful for modeling. I had to figure out how to turn that into structured inputs the model could actually learn from.

There was also the issue of sample size. I didn't have thousands of connections to work with. Thanks to LinkedIn's weekly invite limits (and a looming deadline), I was capped at around 200 total invites. So I needed to be smart about feature design — focusing on simple, meaningful signals that could work well even with limited data.

With that in mind, I started engineering features. My first goal was to turn messy text into quantifiable traits. Here's what I extracted:

What did I extract?

Connection_Count
How many connections the user has. Basic, but important. This was central to a few of my hypotheses.

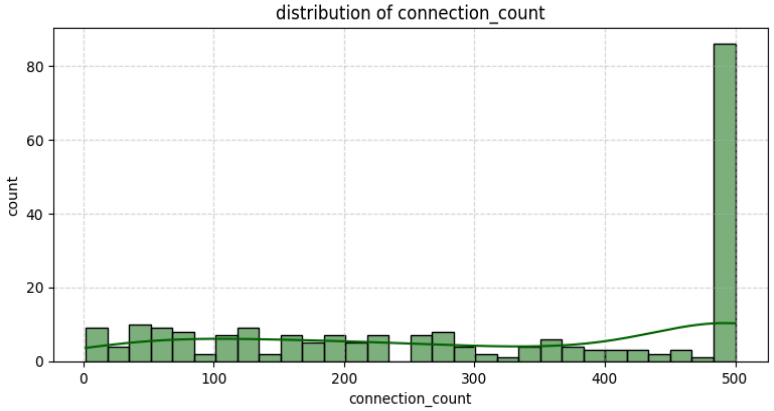
N_experiences
The number of work experiences, internships, courses, or projects listed. A rough proxy for how "busy" or detailed someone's profile is.

Len_raw_text
How much someone actually wrote on their profile. A zero-word, empty shell? Low effort. A full breakdown of every role with a mini essay? High effort. This acts as a loose signal for engagement or effort.

Same_Interest_Score
This one checks how many times a profile matches a list of preset interest keywords (e.g., data science, machine learning, finance). It's customizable in the config, so it adapts to whatever field you're targeting. The pipeline dynamically calculates this for each profile

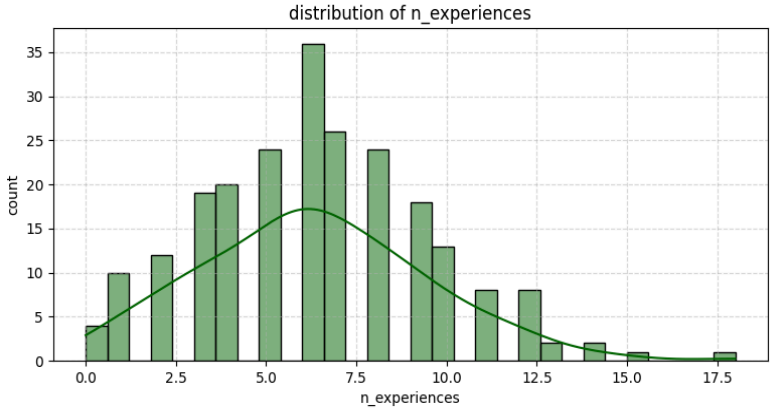
I've included distributions of each of these on the next page to give you a feel for how they're spread across the dataset. That said, raw numbers like these don't always play well with small datasets. With only ~200 max sent invites to work with, I needed to make the model as robust as possible. So I transformed these continuous values into categorical features tied directly to my hypotheses — which I'll explain next.

Simple Univariate Analysis



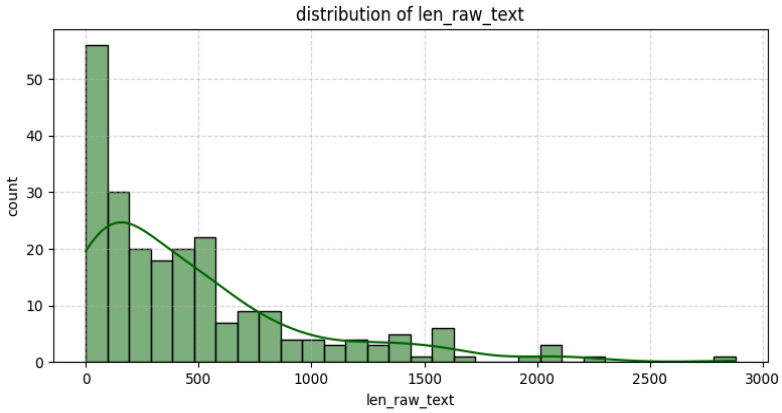
connection_count

Shape: Very right-skewed (positive skew)
Key observation: Huge spike at **500** (likely due to LinkedIn capping visible counts at "500+")
Interpretation: Most users have low to moderate connection counts, but a large chunk appear maxed at 500+. Could indicate either power users or LinkedIn display limitation.



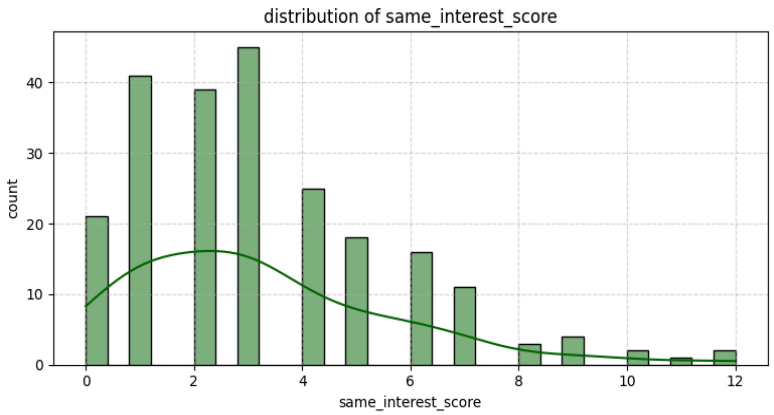
n_experiences

Shape: Slightly right-skewed but relatively symmetric
Central tendency: Peak around 5–7 experiences
Interpretation: Most users have some professional experience listed, with a few outliers having many roles.
Implication: This is a nice, usable feature. Could be interpreted as a signal of professional maturity or profile richness



len_raw_text

Shape: Strong right skew
Key observation: Long tail — most users have very short text (e.g., barebones profiles), while a few have rich, detailed profiles.
Interpretation: Possibly a proxy for profile effort or engagement.



same_interest_score

Shape: Right-skewed
Peak: Around 1–3 common interest matches
Interpretation: Most users only share 1–3 keywords with me; fewer have strong overlap, naturally though this depends on nicheness of interest words.

My Hypotheses

I'm testing five core hypotheses. Two stem from some light game theory: (1) users with fewer connections may accept more requests due to coordination dynamics, and (2) those nearing 500+ may connect more readily to boost visible status. Two others are based on logical assumptions: (3) shared interests (via keyword overlap) increase connection likelihood, and (4) profile effort—measured by how much someone has written—may signal engagement and responsiveness. Finally, (5) I'm exploring gender as a potential factor, treating it as an open-ended variable for investigation.

Hypothesis 1: Coordination-Game

When I had one lonely connection, I would've accepted a request from literally *anyone* just to boost my visibility. That got me thinking — maybe people with low connection counts feel the same. We both benefit from connecting… but only if the other accepts.

This idea maps neatly onto a basic coordination game from game theory:

Two low-connection users (A and B). User A sends a connection request.
User B decides whether to accept or ignore.

	ACCEPT	IGNORE
P1(SENDS REQUEST)	(+1, +1) both gain visibility	(0, 0) No connection formed

Two low-connection users (A and B). User A sends a request. If User B accepts, they both gain visibility and credibility. If they don't, nothing happens. Since there's no real cost to accepting, mutual cooperation (accepting) becomes the rational strategy.

The key assumption here is that low-connection users are *active enough* to actually see and accept requests. Totally inactive accounts would ruin that logic. So to filter for the likely-active low-connection crowd, I targeted users with **20 to 100 connections**.

I captured this with a dummy variable equal to 1 if the user is in that range, 0 otherwise. This turned out to be one of the strongest predictors in the final model.

Hypothesis 2: Status seeking behaviour

On LinkedIn, something magical happens at 500 connections: the number stops counting outwardly, you just look established, I'm pretty sure amongst real LinkedIn users it's something of a badge of honour. For users sitting in the 450–499 range, every connection gets them closer to that public milestone.

This creates a different kind of game, one rooted in status-seeking and minimal friction.

Two users (A and B). User A sends a connection request. User B has to to respond

	ACCEPT	IGNORE
P1(SENDS REQUEST)	(+1, +1) More visibility, Closer to 500+	(0, 0) No connection formed,

In this version of the game, the user with 450+ connections doesn't even care *who* I am. The payoff is a social proof that their whole network will see. As long as my profile doesn't look spammy, they'll likely accept just to inflate their number.

These users are also highly valuable. Once connected, they unlock a much larger set of second-degree profiles, which means better search results, more scraping potential, and eventually more targeted outreach. It's like opening up advanced filters for free.

I represent this hypothesis as a dummy variable equal to 1 for users between 450-500 connections.

Hypothesis 3: Similar likes and Interests

People are naturally drawn to those who resemble themselves — a phenomenon known as the similarity-attraction effect. Whether it's shared interests, a common university, or both listing "data science" in your headline, even small overlaps can create a sense of familiarity. On LinkedIn, these subtle signals often shape who decides to hit "Accept." This hypothesis tests whether shared traits increase the likelihood of forming new connections in cold outreach. This feature was 1 when a profile shared a number of specified interest keywords, for me this was data stuff. But you can change this to whatever floats your boat in the config section.

Hypothesis 4: Active, Detailed Profiles

Profiles that are more detailed, with written summaries, filled-out experiences, and thoughtful headlines suggest a user who takes LinkedIn seriously. This level of effort often correlates with higher platform engagement. From a behavioral standpoint, more engaged users are not only more likely to check their notifications but also more open to expanding their network. This hypothesis assumes that profile completeness acts as a proxy for responsiveness, increasing the baseline likelihood of accepting a cold connection request. Here I set this feature to 1 when a profile's raw text length was above the median.

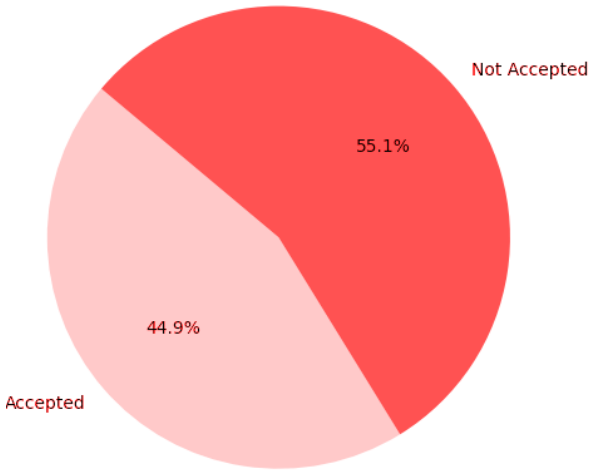
Exploratory: Gender as a Variable

This isn't a formal hypothesis, but more of an exploratory angle. I'll use the Python library gender-guesser, which maps first names to likely genders (e.g., "Jessica" → female, "Michael" → male), to tag profiles accordingly. The goal is to see whether there are observable differences in acceptance rates across inferred genders, or whether the model picks up gender as a meaningful predictor. This was represented as the column likely_female, where 1 represents gender-guesser thinks someone is a woman.

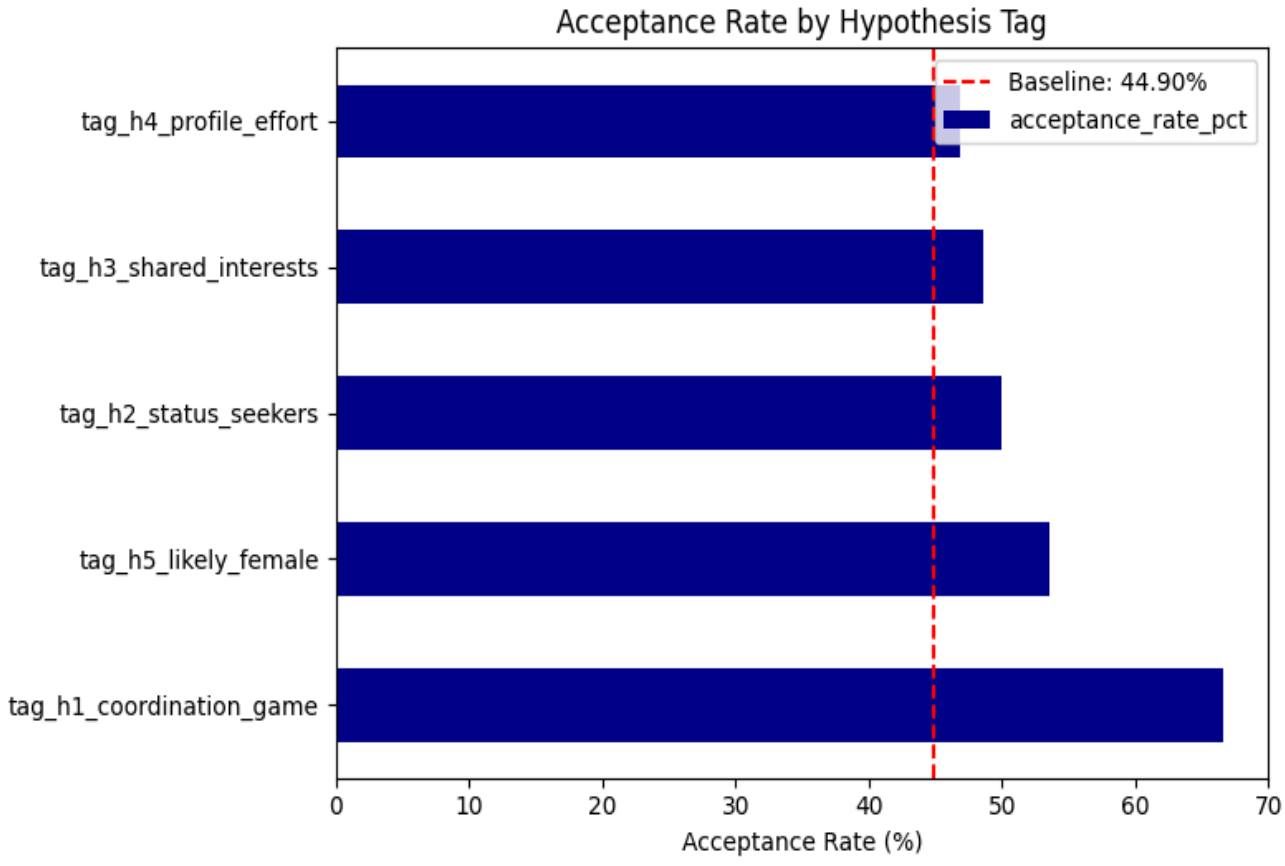
Analysis of Hypotheses | Were any of them right?

In this section I outline differences in acceptance rates between the tags, identify any potential multicollinearity, and conclude with the feature importances (how important each hypothesis the model deemed to be in determining acceptance)

Firstly let's observe the baseline for the overall sample. At 44.9% acceptance it's okay, but not great, at this rate following LinkedIns soft limit of 100 invites a week we'd reach 500+ in ~3 months. Could we make that happen faster? In half that time maybe?



Almost definitely with the right model, because I identified features that show clear predictive power. Here's what I found



Which Hypothesis performed the best?

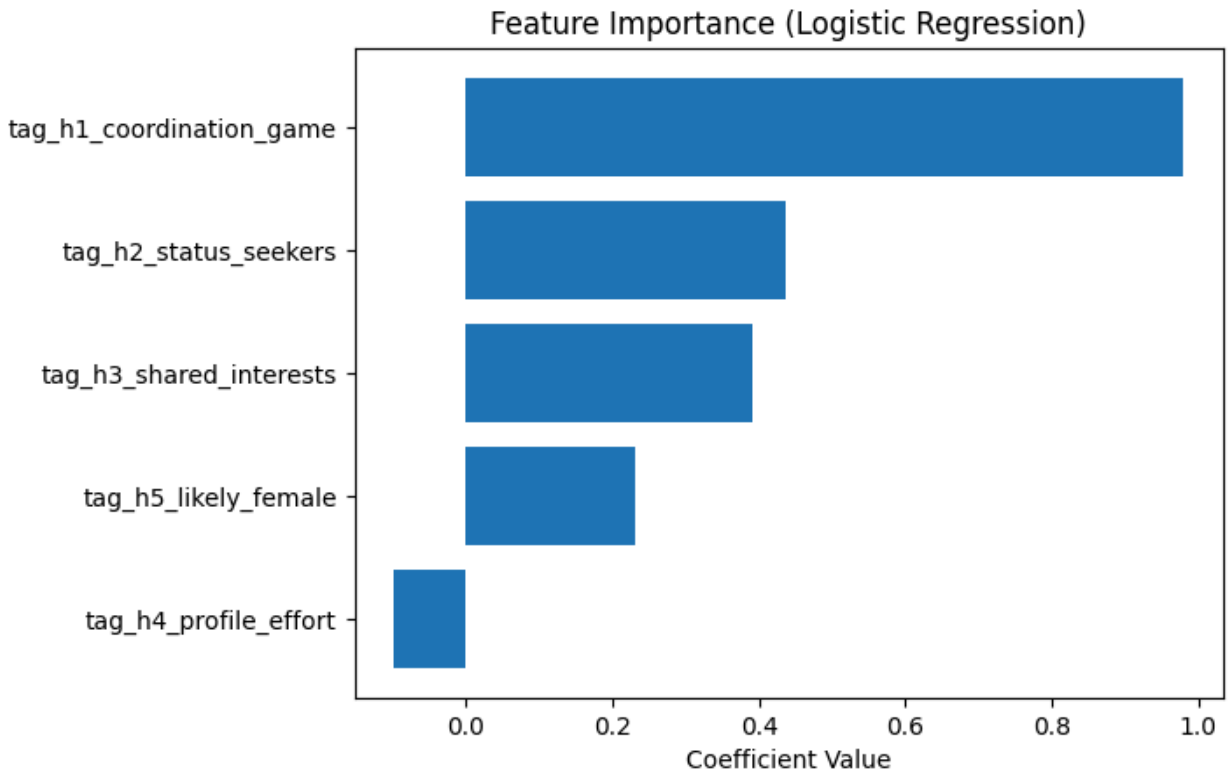
H1: Coordination Game
Highest performing tag with an acceptance rate close to **67%**.
This supports the idea that users with very few connections are more likely to accept requests—possibly due to coordination motives or desire to grow their network.

H5: Likely Female
Second-highest acceptance rate at around **55%**, suggesting a demographic signal may influence likelihood of acceptance—though this should be interpreted carefully to avoid biased targeting.

H2: Status Seekers and **H3: Shared Interests**
Both showed acceptance rates in the low 50s, modestly outperforming baseline. This suggests moderate predictive value, particularly for shared industry or career overlap.

H4: Profile Effort
Performed the weakest among hypotheses that beat the baseline, just above **baseline**. This suggests that detailed profiles alone may not strongly influence acceptance likelihood.

What does our baseline model think?



Interpreting Feature Importances

H1: Coordination Game

Strongest predictor with the highest coefficient (~0.97) in the logistic regression model. The model heavily relied on this tag, indicating that low-connection users were significantly more likely to accept requests.

H2: Status Seekers and H3: Shared Interests

Moderate coefficients (~0.45 and ~0.40), showing these features contribute meaningfully to the model's predictions. The model found some predictive value in users near 500 connections and those with overlapping interests.

H5: Likely Female

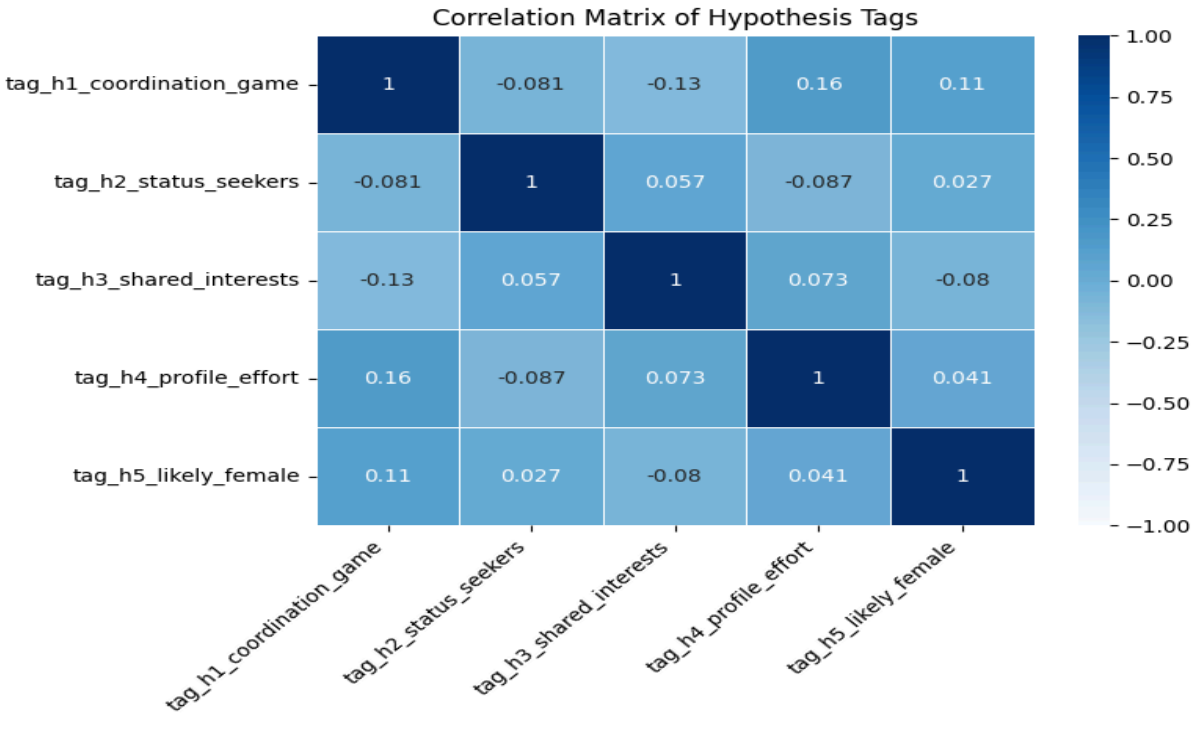
Lower importance (~0.25) than its raw acceptance rate suggested. The model considered this tag somewhat predictive but gave it less weight when adjusting for other variables.

H4: Profile Effort

Lowest coefficient (~0.10), indicating that profile detail and completeness had the least predictive power. This aligns with its weaker real-world performance.

Is multicollinearity likely?

Multicollinearity happens when features overlap too much. For example, if status seekers almost always have high profile effort, the model might struggle to figure out which feature is actually driving the outcome. It becomes harder to say what's really predictive. A quick way to check for this is to plot a correlation heatmap of the features. The heatmap below shows low correlations across the board, meaning our hypothesis tags are mostly independent.So for this sample, multicollinearity doesn't look like a major issue.



Baseline Model Performance | Is there a real predictive edge here?

Class	Precision	Recall	F1-Score
0 (Not Accepted)	0.62	0.83	0.71
1 (Accepted)	0.75	0.5	0.6
Accuracy	0.67	0.67	0.67
Macro Avg	0.69	0.67	0.66
Weighted Avg	0.69	0.67	0.66

To understand how well the model performs, we look beyond simple accuracy and assess three core metrics: precision, recall, and the F1-score.

Baseline Acceptance Rate

Before modeling, the cold connection acceptance rate sat at 44.9%. This is our baseline, this is what we'd expect if we sent invites randomly without any filtering or prediction. Any improvement over this rate suggests the model is successfully identifying better-than-random targets.

Model Accuracy: 67%

The model correctly predicts acceptance or rejection 67% of the time. That’s a 22 percentage point improvement over the 45% baseline. But accuracy alone doesn’t tell the whole story, especially in a case like this with class imbalance (only 45% of users accept).

Precision: 75% (Accepted Class)

When the model says someone will accept a request, it's right 75% of the time.
This is a significant jump over the 45% baseline, it means that the model helps prioritise people who are much more likely to accept, reducing wasted invites.

Recall: 50% (Accepted Class)

The model correctly identifies half of all people who actually would have accepted. This means it's conservative, it doesn't catch every potential acceptor, but the ones it does flag are highly likely to follow through. Is this a massive issue? Not really we can continually scrape more and more profiles, hence for our use case I think precision matters far more than recall.

F1 Score: 0.60 (Accepted Class)

This metric balances precision and recall. A score of 0.60 shows a solid tradeoff, especially considering the limited dataset (~150 samples). The model favors quality over quantity, which makes sense in a setting where you have a limited number of weekly invites.

In Conclusion

The model substantially outperforms random guessing.

It is high-precision, helping you make the most of your limited outreach quota.

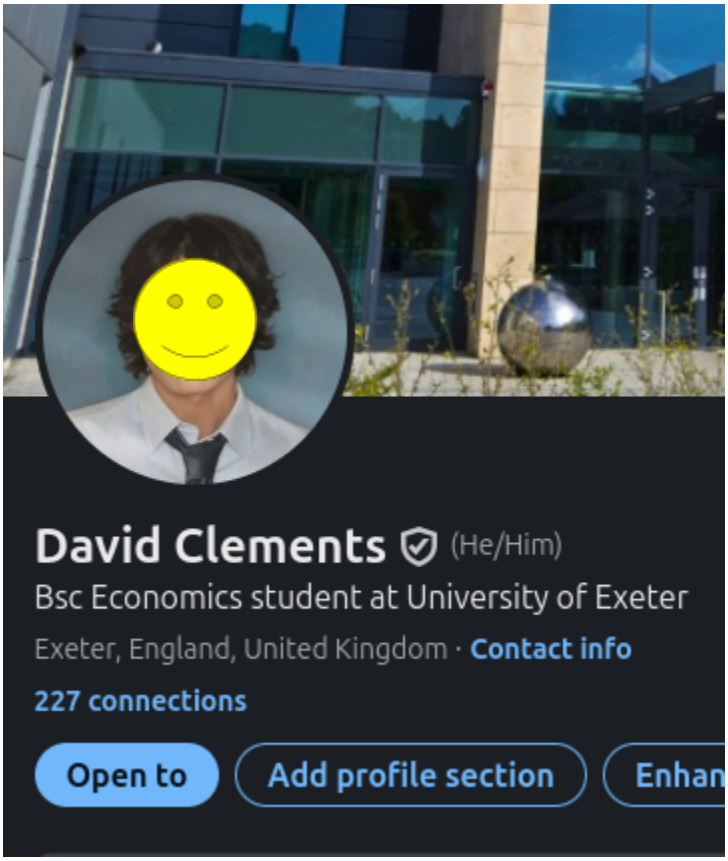
The conservative behavior (lower recall) is a tradeoff that works well when you're selecting from an effectively infinite pool, it's better to send fewer but smarter invites, as you have 100 weekly invites but there's an endless pool of people to send them to.

This model has a practical real application, and should be used as part of the automation pipeline to improve outreach. From comparing the baseline acceptance rate to the model informed acceptance rate, we would see a decline in 40% of the time it takes a new account to reach that shiny 500+ credibility marker.

Ethical and practical considerations

I chose not to include any database of scraped LinkedIn profiles in the GitHub repo. While the data was publicly accessible, something felt a little off about publishing a compendium of my peers profiles online — especially without their knowledge. That gut feeling felt like enough of a red flag. Even if not strictly against LinkedIn's terms, I figured it would likely cross some ethical line in the context of academic work. So I left it out.

All the code is included and documented, so the project is still fully reproducible with your own data, which you can scrape with the provided main.py script or more simply the standalone binary on linux, and the video demo which i have hopefully included a link to will show the core functionality and the setup from install to sending!



2 weeks in it's been pretty good
(higher invite limits because I'm using the Premium LinkedIn trial which gives you a limit ~150)
Following this projection I'll reach that great 500+ badge in a month max? Sweet :)

Potential Updates | What could I add or fix?