AUTOMATED WEB SCRAPING FOR LAND SALES DATA (ACRE TRADER)

Final Report

05/01/2025

Group 9: Jonathan Ivey, Craig Harriman, Phoebe Miller

DASC 49903 – Data Science Practicum II

Dr. Karl Schubert

Executive Summary

Acres.com, a land research platform, faces challenges with manually collecting land sale data from auction websites. It is a time-consuming, labor-intensive process, and our project addresses this inefficiency by developing an automated web scraping system that collects land sale information, predicts its relevance, and provides dashboards for monitoring data quality.

Our approach targets three high-volume software platforms: HiBid, Bid., and NextLot. By tailoring our solution to these specific structures, we developed a scraper with extensive coverage and relative simplicity. Our system combines Selenium and BeautifulSoup to balance functionality with usability while collecting key information such as auction dates, bid prices, property sizes, and descriptions.

To identify relevant land sales, we implemented a natural language processing model that assigns relevance scores to auction listings based on their descriptions. This model demonstrates strong correlation with human-verified relevance labels. Additionally, we created a descriptive dashboard that monitors scraper performance, identifies missing data patterns, and enables manual review of scraping results. Comprehensive documentation ensures the Acres.com team can operate, maintain, and improve our tool independently.

Our solution demonstrates significant efficiency improvements, collecting 1,042 unique listings from 13 websites in 1.5 hours—which is at least 9 times faster than the current manual process. This automated approach will ultimately help Acres.com expand their database of comparable land sales, free analyst time, and enable customers to make more confident land decisions.

Table of Contents

Introduction

Our industry partner is Acres.com. They are a land research platform created to help individuals make confident decisions about buying, selling, or investing in land. One of their

key features is a database of comparable land sales. However, many of these land sales are manually collected from online auction websites. That manual process is time and labor-intensive. Our project attempts to solve this problem by designing an automated system that scrapes land auctions, predicts how likely they are to be relevant to the business, and provides descriptive statistics to monitor data quality.

Understanding the Problem

Acres.com's database of comparable land sales (comps) is a key feature of their platform that provides customers with a unique resource for analyzing past land sales. However, it is currently hand-sourced from hundreds of disparate sources on the internet. Our goal is to develop an automated solution for retrieving relevant, actionable, and recent information from land sale sources. This solution will provide efficiencies, free time for data analysts, and help Acres.com grow their database. Through conversations with the Acres.com team, we distill that goal into a more specific deliverable of a web scraping tool with four key features. First, our scraper must target high-volume sources; there are hundreds of different website structures, but we need our scraper to focus on sources that will provide the most value. Second, the sales we collect must be relevant; auction websites often include listings for small objects or residential units, and we need a system for excluding those from the results. Third, our system must flag inconsistencies; no web scraping approach is perfect, and the Acres.com team needs to identify when manual intervention is needed. Finally, we must provide clear documentation; we will not be working on this project indefinitely, and Acres.com should be able to operate, maintain, and improve it

The Data

The Acres.com team provided us with a list of URLs for land auction websites that they wanted to scrape. They also provided specific information they would like extracted, including the auction completion date, winning bid price, property size, property description, tillable acres, source URL, and a screenshot of the sale. We analyzed these different sources, consulted with the Acres.com team, and noted essential characteristics. First, we identified that most sources followed a standard structure where they had pages listing completed auctions, and those pages linked to different pages for individual sales. This structure informed the final operation of our web scraper, which first collects sets of sale URLs and then scrapes them individually. We also noticed that many sales were multi-parcel. That means they have a single listing page and description but include multiple parcels of land that may sell for different prices. These multi-parcel sales were a challenging complication that our model would need to be able to handle. Finally, we identified a taxonomy of web page structures. Essentially, we found that there were three auction software services—HiBid, Auctioneer Software (referred to as Bid.), and NextLot—that accounted for most land auction websites. Individual land auction services pay these companies to run their websites; as a result, all websites with the same software services have similar structures. We took advantage of these similarities to simplify a scraper into a tool that only needs to handle three structures but can target high-volume sources.

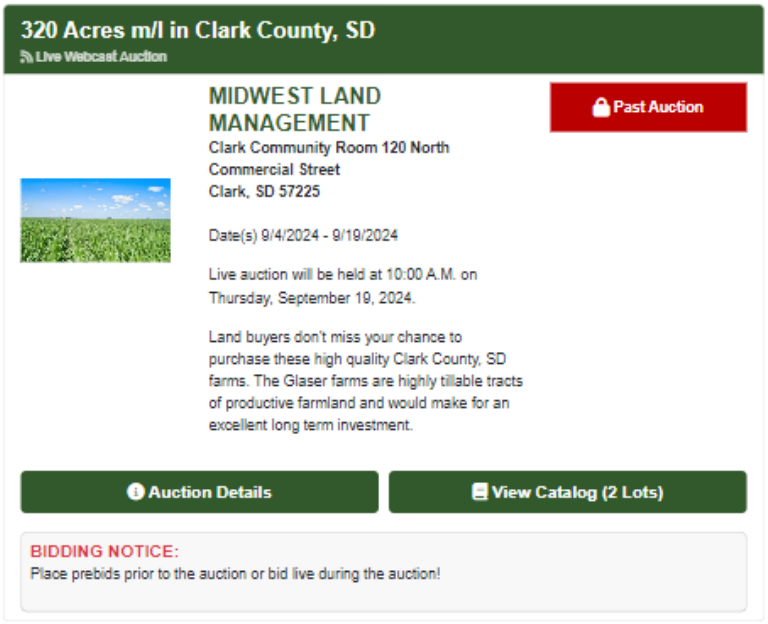


Figure 1: Example of land sale page from midwestlandmanagement.hibid.com

After identifying the three primary services, we performed further analysis of the structure of each website.

HiBid

HiBid was our primary source for initially testing because HiBid website have filters that allow us to scrape primarily land sales before we could predict relevance automatically. The main challenge with these sites were the function of the classes. Classes are an HTML attribute that we used initially used to locate most information on a webpage, but HiBid tended to have inconsistent classes. Learning mitigate this issues was challenging; however, it prepared us for when we encountered them in other sources.

Bid.

Bid. was a particularly unique source. Unlike the other sources, its webpages were dynamically loaded, so instead of the information being located inside the website's regular HTML, it was loaded using Javascript. This created a complicated challenge that we would eventually address using API endpoints.

NextLot

NextLot has the simplest structure of the three sources. It made the easiest to scrape, but it also made it far more likely to have missing fields because the sale records had more variance. Nextlot also has the most multi-parcel auctions, which require us to design systems to collect multiple sales from a single listing.

Methodology

Background & Previous Approaches

Web scraping is the automated extraction of data from websites. Rather than manually collecting information from web pages, which is tedious and time-consuming, web scrapers programmatically collect data with minimal human intervention. Web scrapers start by sending HTTP requests to websites to retrieve their content. After downloading it, the scraper parses HTML, XML, JavaScript or other components to create a structured representation. Then, that representation is queried with selector patterns (like CSS selectors, XPath expressions, or regular expressions) to extract specific elements of interest. Finally, that information transforms into a structured format like a CSV or JSON file.

There are many libraries designed for web scraping. We ultimately decided on combining Selenium and BeautifulSoup, but, we also considered Requests, LXML, Scrapy, and Puppeteer. Requests is a library that allows for sending HTTP requests; however, it cannot execute JavaScript, which is often necessary to scrape modern websites that use dynamically loaded content. LXML is a high-performance library that is more powerful than Requests; however, it has a similar limitation of being unable to handle dynamic content. Scrapy is a more comprehensive option with many of the features we would need; however, it has a steep learning curve and would complicate future maintenance for the Acres.com team. Finally, Puppeteer is a Node.js library that provides an API to control Chrome browsers, which, like Scrapy, is powerful but would add additional complexity. Because Puppeteer is a Node.js library, it would require a mixed technology stack, which would complicate future maintenance.

Web Scraper

Tools

For our web scraper, we combined the Selenium and BeautifulSoup libraries. Selenium is a popular web scraping library used for handling dynamic web pages. It can simulate a real user's behavior by interacting with JavaScript elements and navigating website structures. This is especially useful in our application because we work with a wide variety of websites, and some have information that is only accessible in dynamically loaded elements, which are loaded using JavaScript and not located in the html. Another benefit is Selenium's rendering capabilities. It can load a webpage to generate the same visuals that a user would see, and it has support for taking full page screenshots, which was an important requirement for this project.

Though Selenium has many great features, we wanted our solution to be flexible and easy to build upon for Acres.com. So, we chose to combine the more powerful Selenium package with the more accessible and popular BeautifulSoup package. BeautifulSoup is an easy-to-use library for parsing HTML documents that allows for cleaning and extracting data

already loaded in. Though it has fewer capabilities, its syntax is much simpler than Selenium, and because it is so popular, it has a wide variety of resources for beginners.

We built the primary infrastructure for retrieving and screenshotting webpages in Selenium because we do not expect Acres.com to need to modify that code. Then, we did most of the website-dependent scraping functions in BeautifulSoup so that they could be easily modified or extended. Finally, for websites that were too complex to use only BeautifulSoup, we used the more advanced features of Selenium.

Structure

Our scraper follows a streamline process with a common initial setup that branches based on the source being scraped. We begin by initializing our driver to simulate opening the auction in a web browser. Then, we do a preliminary setup that includes screenshotting the auction (in case it is removed in the future), determining the root of each URL, and pulling the pages full HTML.

After completing the common setup, we run scrapers tailored to the different sources to collect the URLs of individual sale pages from the listing set. Once they are collected, it runs an individual page scraper that is designed to gather information from individual auction pages. These are customized based on the source and rely on different techniques like using HTML classes to collect the information. One key component of these scrapers is the use of API endpoints. As discussed previously, some sources rely on dynamically loaded content, which was initially a major challenge. However, we were able to use API end points to send requests with unique IDs that populate the relevant information. Through research and experimentation, we found we could use these API endpoints to access information that would not have been available otherwise.

Relevance Predictor

A key requirement of the project is to identify relevant land sales. There are many irrelevant auctions for items like propane tanks and gold memorabilia on the websites we are scraping, and ideally, those would not be included in our results. However, the Acres.com team also informed us that their highest priority is having a low false-negative rate, meaning they would rather manually filter out irrelevant sales than potentially miss a relevant sale.

To address this need, we designed a land relevance predictor that uses natural language processing to analyze descriptions of collected auctions and output a score from 0 to 1 that identifies how likely it is to be relevant to the comp database. To achieve this, we take advantage of zero-shot predictions using generative large language models (LLMs). We set up a question-answer prompt using the question, “Is this a land sale?” Then, we take the probability that the word “yes” is predicted and normalize it with the probability that the word “no” is predicted. This gives us a score that we can use to sort the scraped pages by their probability of being relevant.

Using an LLM for this task provides much flexibility because the Acres.com team can modify the prompt to fit future business needs; however, it also poses implementation

challenges. Some of the smallest LLMs still have 7 billion parameters, but by using quantization techniques, we got a Mistral model to run in under 4 gigabytes of VRAM. Those techniques made it small enough to run on almost any reasonably equipped graphics card, and it can run faster than standard transformer-based classification models at inference time.

Descriptive Model

Another requirement of the project is to flag inconsistencies and make our tool adaptable to the needs of Acres.com. This requires an understanding of the operations of the scraper and the quality of the resulting data. To achieve that, we designed a dashboard to answer questions like:

- How many land sales are being collected?
- What data is missing?
- How does performance vary by domain?

We organized the dashboard into three main sections to address these questions: scraper overview, per-domain summary cards, and a raw listings viewer. The first section is an overview of all the scrapers, giving a high-level look at overall performance. It includes whole and partial scrapes metrics that indicate how many listings were scraped completely versus partially (with missing fields), a bar chart of scrape outcomes that compares these metrics, and a heatmap of missing fields the represents which fields are most frequently missing by domain (Figure x). It includes whole and partial scrapes metrics that indicate how many listings were scraped completely versus partially (with missing fields), a bar chart of scrape outcomes that compares these metrics, and a heatmap of missing fields that represents which fields are most frequently missing by domain.

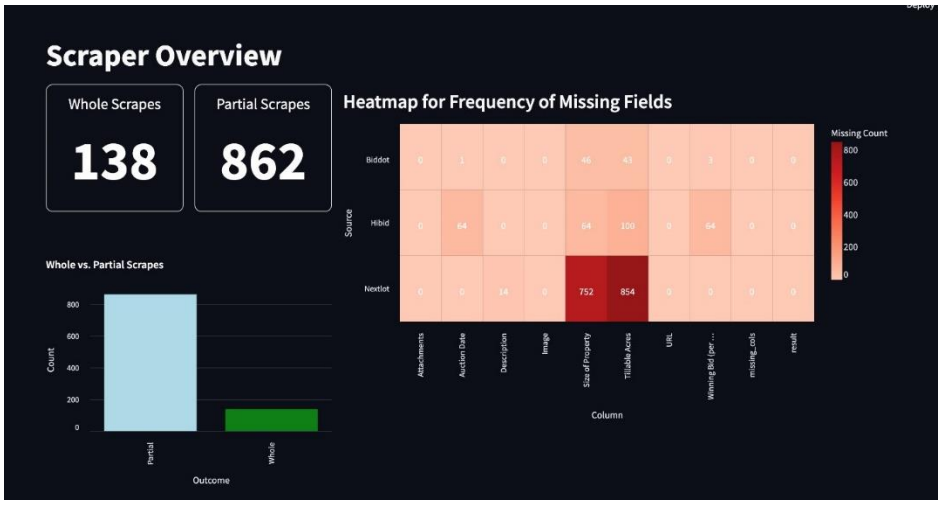


Figure 2: Scraper overview section with whole and partial scrapes metric, bar chart of scrape results, and heatmap of missing fields across domains

The second section gives a more granular breakdown for each domain via its own summary card. These cards include a pie chart of missing data proportions that gives a sense of relative data completeness, whole and partial scrape metrics, a bar chart of scrape

outcomes that compares those metrics, and a total pages scraped metric indicating how much raw data was gathered for each domain (Figure x). These cards include a pie chart of missing data proportions that gives a sense of relative data completeness, whole and partial scrape metrics, a bar chart of scrape outcomes that compares those metrics, and a total pages scraped metric indicating how much raw data was gathered for each domain. This modular approach facilitates quick performance comparisons across domains while allowing deeper inspection per source.

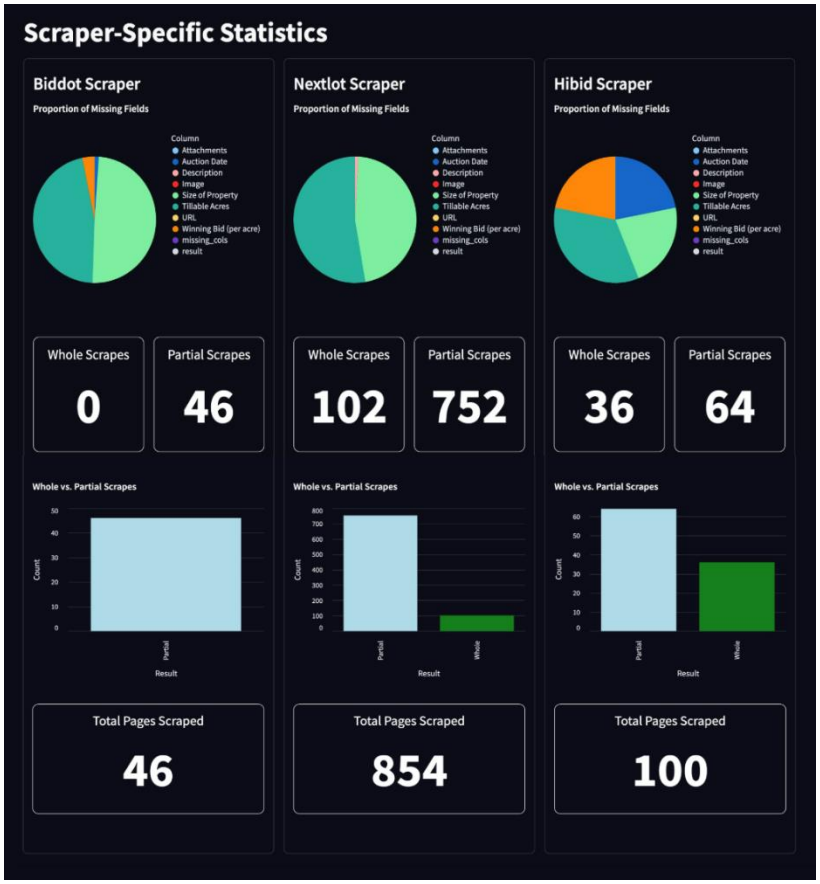


Figure 3: Per-domain summary cards with pie chart of missing fields, whole and partial scrapes metrics, bar chart of scrape results, and total pages scraped metric

Our industry partner requested the third and final section which includes a full, scrollable table of the scraped listings. This allows for manual review of edge cases or failed records and cross-referencing with source sites to verify individual entries if discrepancies are present.

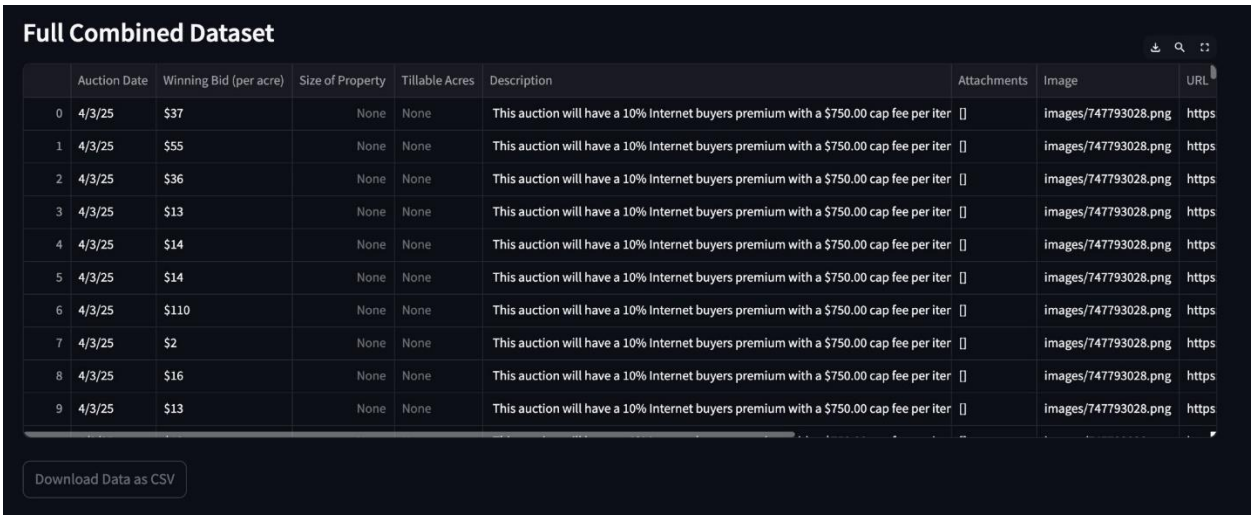


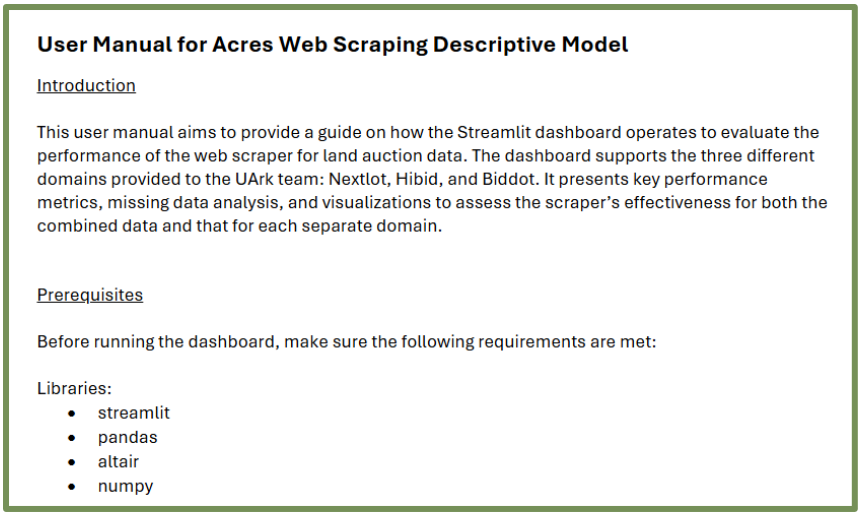
Figure 4: Table of all scraped listings for manual inspection

Alongside the request of our industry partner, we used Streamlit for its simplicity and seamless web app deployment capabilities, and we chose Altair for its aesthetic nature, declarative syntax, and strong integration with Pandas. Both tools enabled us to iterate quickly and focus on insight generation.

No web scraper is perfect, but our reporting will help Acres.com supplement the data collection process with their human capital, flag errors early to prevent downtime, and identify future areas for improvement. Ultimately, the dashboard acts as a health check over the entire scraping pipeline, enabling more trust in automation and faster identification of improvement needs.

Documentation

An important requirement for our web scraping tool is the ability to operate, maintain, and improve it over time. To support that goal, we wrote guides for the three key parts of our system: the web scraper, the relevance predictor, and the descriptive model. In each guide we include descriptions, operation requirements, instructions, and descriptions of the different sections of the code. Our descriptions cover the necessary inputs, parameters, and expected outputs. We cross-tested these guides with different team members to ensure they are interpretable to those less familiar with the specific code.



User Manual for Acres Web Scraping Descriptive Model

Introduction

This user manual aims to provide a guide on how the Streamlit dashboard operates to evaluate the performance of the web scraper for land auction data. The dashboard supports the three different domains provided to the UArk team: Nextlot, Hibid, and Biddot. It presents key performance metrics, missing data analysis, and visualizations to assess the scraper's effectiveness for both the combined data and that for each separate domain.

Prerequisites

Before running the dashboard, make sure the following requirements are met:

Libraries:

- streamlit
- pandas
- altair
- numpy

Figure 5: Excerpt of our guide for the descriptive model.

Results

Web Scraping

So, the most important problem that we wanted to know, how effective was our scraper at collecting listings? In our official testing of our scrapers, we collected from 13 different websites, gathering a total of 1042 unique listings. This process took about an hour and a half to collect which compared to their current process manually, drastically speeds up the process as far as data collection goes.

Based on the insights from our descriptive model, our scraper collected 138 complete listings and 862 partial listings, demonstrating a strong ability to gather large amounts of land sale data. Regarding the large portion of partial listings, the majority of those were solely missing the property size and tillable acres data, likely due to those listings not being land sales.

Among all scrapers, the HiBid scraper emerged as the most efficient, likely because it has a filtering option to view solely land sales. By comparison, the Nextlot scraper yielded many more whole scrapes, as we scraped the most auction listings from those sites. Essentially, each domain scraper is very valuable to the Acres team in offering them a significant amount of data much more efficiently than previous processes.

When benchmarked against the current manual process, which results in data from approximately 10.61 pages per hour, our tool proved to be at least nine times faster, with an average of over 95 auctions collected per hour. This figure is based solely on whole scrape results; when partial scrapes are taken into account, the overall efficiency rate of the tool is about 60 times faster than the manual process.

These results validate the scraper's performance and underscore its ability to significantly improve the speed and scale of data collection for AcreTrader while also highlighting areas for further refinement.

Relevance Predictor

To evaluate the effectiveness of the predictive model, we needed ground truth examples to compare to, so we manually reviewed 500 auctions from our three domains and gave them each a binary label indicating whether they are a relevant land sale. We did this in an iterative process that gave us an even 50/50 split for relevant and irrelevant sales. Then, we ran our predictive model on this test set and analyzed the scores it produced.

We found that if you treat the relevance scores as a classifier (where a score greater than 0.5 indicates relevance) our predictor has 83% accuracy on the test set. Notably, for the instances that are categorized incorrectly, the score predicted is within 0.1 on average. This shows the predictive model's scores strongly correlate with the true relevance of sales, and it will be a valuable tool for prioritizing relevant sales over irrelevant sales.

Challenges

Highly Variable Web Pages

When initially discussing the design of our web scraper, we wanted to create one scraper that could extract information from many websites. An almost immediate challenge with this idea was the variability in website structures. Each website the Acres team uses for their data has a unique layout with varied class names, tag structures, and element IDs. This lack of a standardized structure means that a scraper designed for one structure won't work on another without specific adjustments. Some websites also use nested HTML structures or elements that only appear after executing a specific action, like clicking a button, which adds complexity. Websites also tend to represent similar data differently. Examples would be dates appearing in various formats or numbers, including different currency symbols or separators. To overcome these challenges, we collaborated with the Acres.com team to create a list of the top three sources that sites use to auction land parcels and divided them among our members to make a scraper for each of the three sources. This allowed for specific adjustments each site might require, like varied class names and hidden information. Our final scraper will check the URL input to see which source it belongs to, then scrape the site with its related scraper.

Dynamically Loaded Content

Another major issue we encountered was struggling to scrape dynamically loaded content. Many websites use JavaScript to load or alter content dynamically, meaning other information may only appear after scrolling or clicking a button. This complicates the scraping, requiring additional programming to mimic the necessary actions. The Bid. sites were especially prone to this issue, as the attachments and auction dates were hidden within tabs that the scraper could not click. After further research and consulting with the Acres team and other mentors, we discovered that API endpoints could be used to solve this problem. Many websites use APIs to streamline dynamic content loading, sending requests to receive information in a structured format. Identifying these endpoints can simplify scraping by providing direct access to the underlying data without needing to

parse the HTML. These API endpoints also became beneficial when figuring out how to collect the URLs for these websites. Using these endpoints, we were able to locate the unique IDs for each auction and gather these URLs effectively.

Repetitious Text Data

When we initially designed the relevance predictor, we intended it to use a standard BERT-based classification model trained on examples of relevant and irrelevant land sales; however, we found unique characteristics of the data that caused overfitting. Many auction sites have highly repetitious descriptions that follow a formula, and training on these teaches the model to focus on the wrong aspects of the text. For example, initial modeling tests showed that simply including the words “for sale” could increase a description’s likelihood from 0.1 to nearly 0.85.

That level of instability is insufficient for the long-term operation of the model, so we designed a new architecture based on zero-shot predictions with a large language model. Implementing that model required using techniques like quantization to lower the memory usage to a reasonable level. However, it resulted in a more effective and adaptable model because its predictions can be tuned by simply changing the prompt.

Summary and Conclusions

Our primary goal was to develop an automated solution for retrieving relevant, actionable, and recent information from land sale sources.

Our tool accomplishes this by targeting the three most significant sources of land sale data and scraping them efficiently. With this process, the Acres team will be able to automate their data collection, as well as drastically speeding up the co up the collection process. Many of the complicated issues related to pulling the correct information are handled on the backend to streamline the process for the team to be able to run the scraper. To go with this, we have a user manual we have created if someone who is not familiar with the code needs to use it. While not perfect, this is a much better solution to the problem than how it was handled previously.

Beyond just collecting information, we designed a relevance prediction model to intelligently rank scraped auctions. Our predictor utilizes a pretrained generative language model to calculate relevance probabilities based on a customizable prompt. Our model predictions demonstrate a strong correlation with manually verified relevance indicators, correctly predicting 83% of sources with an average error of 0.1 on misclassifications. They can be used effectively to prioritize the most relevant land sales during the collection and verification process. We further optimized our model with quantization to ensure that it can run on hardware with limited video memory.

We also created our system with long-term adaptability in mind. It produces regular performance reports in a dashboard. Automatically visualizing missing fields and whole and partial scrapes for each scraper allows the Acres team to efficiently identify possible issues with a scraper and target specific areas of that scraper to update. On top of this, its

clear structure and documentation make it easy to maintain, scale, and hand off to future teams. Creating robust user manuals for the scraper, predictive model, and descriptive model ease the process of handing this project over to AcreTrader. Those unfamiliar with web scraping, LLMs, or Streamlit and Altair, or need help in upkeep with any part of our tool, can refer to the documentation to streamline their workflow.

Retrospective

Reflecting on our development process, we identified several practices that contributed to our success and where adjustments could have increased our efficiency and enhanced our collaboration.

Maintaining detailed notes ensured that everyone was aligned with and aware of their responsibilities, serving as a reference point for us to track progress and avoid doubling our efforts. Being consistent with this in our Trello board earlier could have made task assignments more accessible, but we could do so once we received feedback on our work with the project management tool.

Moreover, we found that seeking help from external sources saved time and effort. This was particularly prevalent in the issue of dynamically loaded content, where the solution came from Stack Overflow after significant time was devoted to individual research that did not lead to any solutions. We delayed seeking outside help initially, which cost us valuable time. In the future, we plan to reach out early when we are stuck on an issue to save our time and effort.

We also took advantage of each team member's specialization. Dividing tasks based on individual strengths increased our productivity and allowed for higher-quality results. We initially distributed tasks uniformly, which led to slower progress, but recognizing and assigning tasks based on our strengths earlier would have been beneficial. Each team member's unique experience played a key role, such as one member focusing on the structure of the scraper, another on dynamic content scraping, and the other on research.

Learning to plan for imperfection was another takeaway from this project. Since real-world data is rarely clean, we focused less on building a flawless system and more on creating one that is transparent and adaptable. By including fallback mechanisms and clear documentation, we managed challenges like filtering out non-land sales, handling missing data, and anticipating areas for future refinement.

Overall, our strengths included clear communication and collaboration, adaptability and willingness to pivot our approach, and not expecting perfection, which proved crucial for our success. We learned that having a proactive mindset can significantly improve group efficiency, and documentation and task-tracking tools are invaluable for staying focused and maintaining momentum.

Citations and Resources

AcreTrader Inc. “Farmland Investing Resources.” *AcreTrader*, acretrader.com/resources.

AcreTrader Inc. “Knowledge Base.” *Knowledge Base*, support.acres.com/.

Bhattarai, Aagaman. “Running a Streamlit Application in Google Colab.” *Medium*, Medium, 27 June 2024, medium.com/@aagamanbhattrai/running-a-streamlit-application-in-google-colab-1576b7188c87.

Heath, Anthony. “Web Scraping with Python: A Complete Step-by-Step Guide + Code.” *Medium*, Geek Culture, 29 Mar. 2023, medium.com/geekculture/web-scraping-with-python-a-complete-step-by-step-guide-code-5174e52340ea.

Laetsch, Thomas. “Web Scraping with Python Course.” *DataCamp*, www.datacamp.com/courses/web-scraping-with-python.

Nantasenamat, Chanin. “Building a Dashboard in Python Using Streamlit.” *Streamlit Blog*, Streamlit, 22 Jan. 2022, blog.streamlit.io/crafting-a-dashboard-app-in-python-using-streamlit/.

Saettele, Dylan. “Comp Sources & Initial Notes.”

Solanki, Kamlesh. “Web Scraping with Python: Beginner to Advanced.” *Medium*, Analytics Vidhya, 30 July 2021, medium.com/analytics-vidhya/web-scraping-with-python-beginner-to-advanced-10daaca021f3.

Streamlit Documentation. “Chart Elements.” *Streamlit Docs*, Streamlit, docs.streamlit.io/develop/api-reference/charts.

User: FrocketGaming. “Display a Created Heatmap on My Web App.” *Streamlit Discuss*, Streamlit, 26 Mar. 2023, discuss.streamlit.io/t/display-a-created-heatmap-on-my-web-app/40126.

User: x1337Loser. “Web scraping from a hidden tab on a site using python.” *StackOverflow,* 08 Nov. 2024, https://stackoverflow.com/questions/79171749/web-scraping-from-a-hidden-tab-on-a-site-using-python/79172488?noredirect=1#comment139647613_79172488