Data Visualisation
Assignment 1
Document Submission Cover Sheet

Please ensure that you complete all relevant sections below and that you make this the ***first page*** *of your assignment submission.*

For submission, you are required to put all aspects of the assignment *into one document, including the cover sheer below and then to submit the assignment on Moodle in the relevant module location.*

The assignment must be completed on or before the designated submission date and time. Students are recommended to keep a copy of the assignment, as submitted.

Please complete the following, in advance of submission.

If for some reason you cannot complete this form to attach to your assessment, please hand write the details, photograph and upload to Moodle.

Student Number: A00325359	**Programme Title:** MSC Software with AI
Programme Year: 1	**Module Title:** Data Visualisation
Lecturer Name: Mark Daly	
Visualisation Title: Iceland Economic Dashboard	

Declaration of Authenticity: (Please Sign)
By uploading this document, I confirm that I have read and understood the assignment instructions and that the attached assignment is original (save for the visualisation being analysed) and represents all my own work.

andrew Nolan.

Table of Contents

Iceland Economic Dashboard

1. Introduction

1.1. API

The `api.py` file defines the `StatisticsIcelandAPI` class, which acts as a wrapper around the Statistics Iceland API. It's designed to manage endpoints, handle requests responsibly, and log activity to avoid exceeding rate limits.

1.1.1 How Endpoint Management Works

The class maintains a dictionary of API endpoints using `APINode` objects (defined in `constants.py`). You can add these endpoints either individually or in bulk using the `add_endpoint` and `add_endpoints` methods. This setup makes it easy to organise and manage which parts of the API you're accessing.

1.1.2 Handling Requests & Rate Limiting

The core of the request process is handled by a private method called `_request`. It does the following:

- Checks if the number of recent requests is within the allowed limit using the `RequestLogger.queryRemaining` method from `utils.py`.
- Raises a `TooManyRequestsError` (from `exceptions.py`) if the limit is exceeded.
- Sends a POST request asking for data in CSV format.
- Logs each request in a local SQLite database to track usage.

The public `request` method then goes through all the endpoints you've added, uses `_request` to query each one, and returns the responses in a dictionary.

Rate limiting is enforced through a 10-second window, allowing a maximum of 10 requests. This prevents overloading the API and helps stay within its usage guidelines.

1.1.3 Type Safety and Code Quality

To keep the code robust and avoid bugs, the class uses the `beartype` decorator. This enforces strict type-checking on function inputs and outputs, making the code easier to maintain and safer to use.

1.1.4 Database Integration

The system connects to a local SQLite database (configured via `SOURCE.DATA.DB.str` in `constants.py`). This is where each request is logged, which supports both auditing and the rate-limiting feature. The database plays a key role in managing how often the API is accessed.

1.1.5 Supporting Files

- **`constants.py`**:
  Defines the `APINode` class (which stores endpoint info), rate limit settings, and file paths like the database location.
- **`utils.py`**:
  Contains the `RequestLogger` class for interacting with the database, as well as helpers like `convert_to_df` to turn API responses into DataFrames.
- **`exceptions.py`**:
  Defines custom exceptions, including `TooManyRequestsError` to handle cases where API limits are exceeded.

1.1.6 Putting It All Together

In practice, here's how it works:

1. You initialize the API object, which sets up access to the database.
2. You add the endpoints you need.
3. When you call `request()`, the system checks rate limits, queries each endpoint, logs the requests, and returns the results.

The overall design is modular, efficient, and compliant with the API's rules. It also emphasizes clean structure and type safety, making the code easier to work with and extend.

1.2. Background processes and database

The `daemon.py` file handles all the behind-the-scenes data refreshing. It defines a class called `DataRefresher`, which runs in the background and keeps the SQLite database up to date by regularly pulling data from the Statistics Iceland API. This is done using Python’s `multiprocessing.Process`, so it runs separately from the main dashboard app without slowing anything down.

What `DataRefresher` Does

When you start a `DataRefresher`, you give it:

- an API endpoint (`APINode`) to pull data from
- a table name for where that data should go in the database
- how long it should wait between updates (`sleep_period`)
- an optional delay before starting (to stagger requests)
- an event (`alive_event`) that lets you stop it cleanly

Once it starts, it waits (if needed), sets up the API connection, and enters a loop:

1. It pulls data from the endpoint.
2. Cleans it up and turns it into a Pandas DataFrame.
3. Overwrites the old table in the database with the new data.
4. Logs the update and sleeps until it's time to go again.

If anything goes wrong during that loop, it logs the error and tries again after a pause.

The helper method `processResponse` makes sure the data is cleaned and typed properly.

How It Connects to the Dashboard

Even though you won't see `DataRefresher` mentioned directly in `app.py`, it's a key part of the whole system. While the Dash app focuses on layout and user interaction, the `DataRefresher` keeps the underlying database current. For example:

- Modules like `state_of_the_economy` and `iceland_through_time` pull their data directly from the database tables populated by `DataRefresher`.
- Dash's `dcc.Interval` components make sure the UI is updated periodically, so users always see the latest figures.

This setup makes the app more modular and scalable-data fetching is handled separately from the user interface, which keeps things snappy.

Typical Flow

Let's say you're tracking population data:

- A `DataRefresher` is launched to watch that API endpoint.
- It pulls fresh numbers every so often and replaces the data in the "population" table.
- The dashboard reads from that table and updates the graphs.
- The user sees real-time figures, without knowing anything is happening in the background.

Summary

`daemon.py` provides the backend logic that keeps your data fresh and your dashboard reliable. By offloading the API calls and data processing to background processes, the app runs smoother and scales better as more endpoints are added.

1.3. Functionality of Dashboard

The Dashboard is built with Dash and Plotly, and it pulls in live data from the local Database to give a snapshot of how the country's economy is doing. The app breaks that info down into graphs and metrics that update regularly and are easy to interact with. It's split into two main tabs: the first shows headline figures like GDP, population, CPI, and net migration; the second takes a longer view and tracks how these indicators have changed over time.

The data itself isn’t fetched directly every time the user interacts with the dashboard - instead, background processes (defined in `daemon.py`) pull fresh data in every so often and update a local SQLite database. The dashboard just reads from that.

The updates are handled by Dash's `dcc.Interval`, which runs quietly in the background and triggers the necessary callbacks to re-render the graphs. For example, most of the charts get refreshed every 3 seconds. There’s also an interval tied to the population slider so that the animation works smoothly when someone hits play.

Tabs at the top let you switch between the main views - one is called "Overview", the other is "Iceland Through Time".

In the Overview tab, the dashboard queries the database for the most recent CPI, population, migration, and GDP figures. Those queries are done using functions like `getCPI()` or `getGDP()` from the `state_of_the_economy.py` file. The same tab also includes pie charts for things like employment breakdown and citizenship distribution - again, these pull straight from the database.

The "Iceland Through Time" tab is about trends. You can pick an economic variable (like CPI or GDP) and get a time-series chart of it. These are built using helper functions like `charts_consumerPriceIndex()` or `charts_EmploymentBySector()` that fetch and plot the relevant data.

There are also some more niche population charts here: one compares native vs non-native population; another shows births, deaths, and net migration over time. The population pyramid is a horizontal bar chart that updates when you move a slider, or you can hit play to animate through the years. There’s also a chart just for the non-binary population.

All the data comes from a single local SQLite file (`database.db`). That file is kept fresh by the `DataRefresher` background processes in `daemon.py`. Each one is responsible for hitting an API endpoint, cleaning the data, and writing it to the correct table. The dashboard itself never talks to the API - it just reads whatever’s in the database.

In terms of user experience, everything is interactive - dropdowns, sliders, play buttons. The dashboard keeps itself up-to-date, so you don’t need to reload the page or press anything to see the latest figures.

All in all, the dashboard is a clean way to keep track of Iceland’s economy using live data in a format that’s easy to understand. The separation between background data collection and front-end display helps keep things snappy and modular.

1.4. Deployment

The Dashboard was deployed and made live using render.com. You can view the dashboard on the web here.

1.5. GitHub

You may view the GitHub repository here. There are commits dating back to the beginning of April.

2. Datasets

In the /data folder you will find some csvs. I retrieved them from the below endpoints using the following code.

```
        resp = requests.post(endpoint.str,
                             json = {
                                "query": [],
                                "response": {
                                    "format": "csv"
                                }
                            }
                        )
```

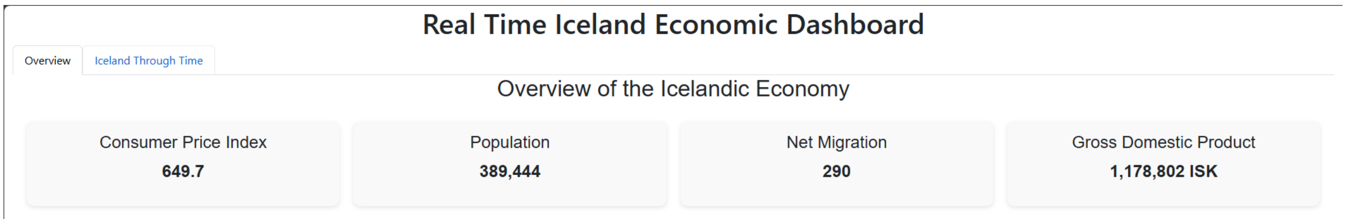
https://px.hagstofa.is/pxen/api/v1/en/Efnahagur/visitolur/1_vnv/1_vnv/VIS01000.px
https://px.hagstofa.is/pxen/api/v1/en/Efnahagur/vinnumagnogframleidni/vinnumagn/THJ11001.px
https://px.hagstofa.is/pxen/api/v1/en/Efnahagur/thjodhagsreikningar/landsframl/2_landsframleidsla_arsfj/THJ01601.px
https://px.hagstofa.is/pxen/api/v1/en/Efnahagur/thjodhagsreikningar/fjarmalareikningar/peningamal/PEN01101.px
https://px.hagstofa.is/pxen/api/v1/en/Ibuar/mannfjoldi/1_yfirlit/arsfjordungstolur/MAN10002.px
https://px.hagstofa.is/pxen/api/v1/en/Ibuar/mannfjoldi/1_yfirlit/arsfjordungstolur/MAN10001.px
https://px.hagstofa.is/pxen/api/v1/en/Ibuar/mannfjoldi/1_yfirlit/yfirlit_mannfjolda/MAN00101.px

Consumer_price_index_and_changes__base_1988_100.csv
Number_of_employed_persons__jobs_and_hours_worked_by_economic_activity_and_quarters__1991_to_2024.csv
Quarterly_GDP_1995_to_2024.csv
Weighted_average_interest_rates_of_commercial_banks_1960_to_2016.csv
Births__deaths_and_migration_by_sex_and_citizenship__NUTS3_regions_and_quarters_2011_to_2024.csv
Population_by_municipality__sex__citizenship_and_quarters_2011_to_2024.csv
Population_by_sex_and_age_1841_to_2025.csv

The endpoints correspond to the csvs in the same order. The datasets are too large to show here.

3. Visualisations and Analysis

The dashboard is split into two sections - one for an at-a-glance summary of Iceland's economy right now, and one that looks back at how things have changed over time. It's designed to let the viewer explore economic and demographic trends without needing to dig through raw tables or spreadsheets.
The first tab, called "Overview," gives a snapshot of the present. At the top, there's a set of headline figures - CPI, population, net migration, and GDP. They're updated automatically from the database, and I've kept them text-based so they're easy to skim.

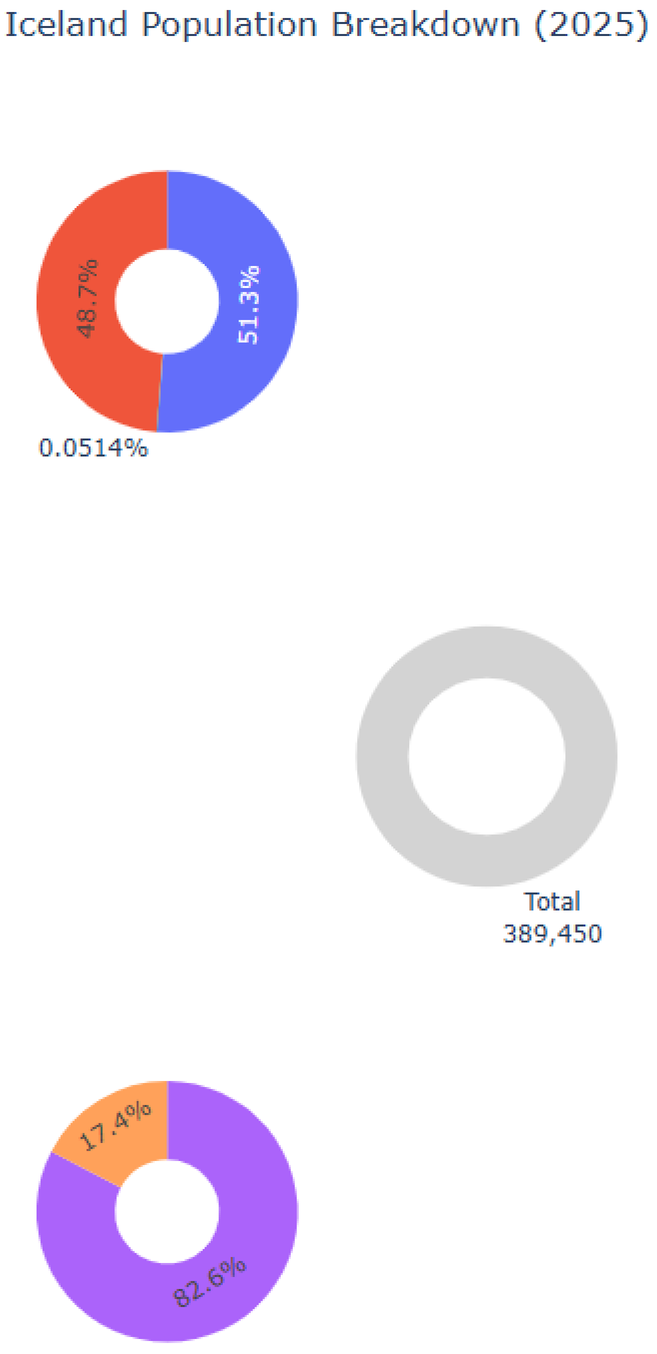

Real Time Iceland Economic Dashboard

Overview | Iceland Through Time

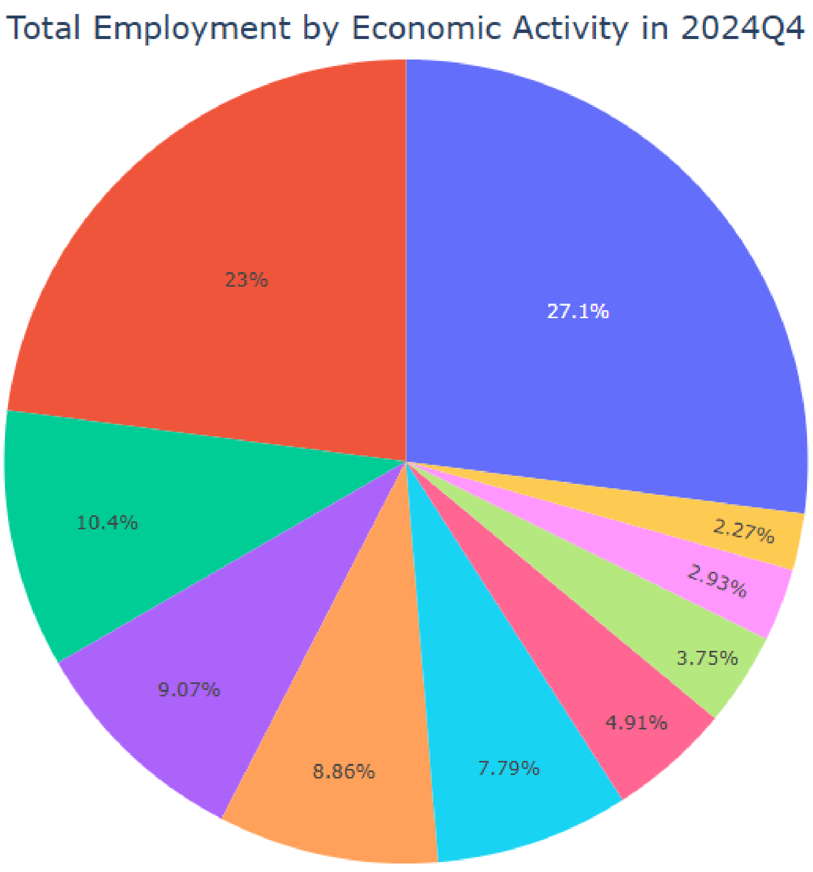
Overview of the Icelandic Economy

Consumer Price Index	Population	Net Migration	Gross Domestic Product
649.7	389,444	290	1,178,802 ISK

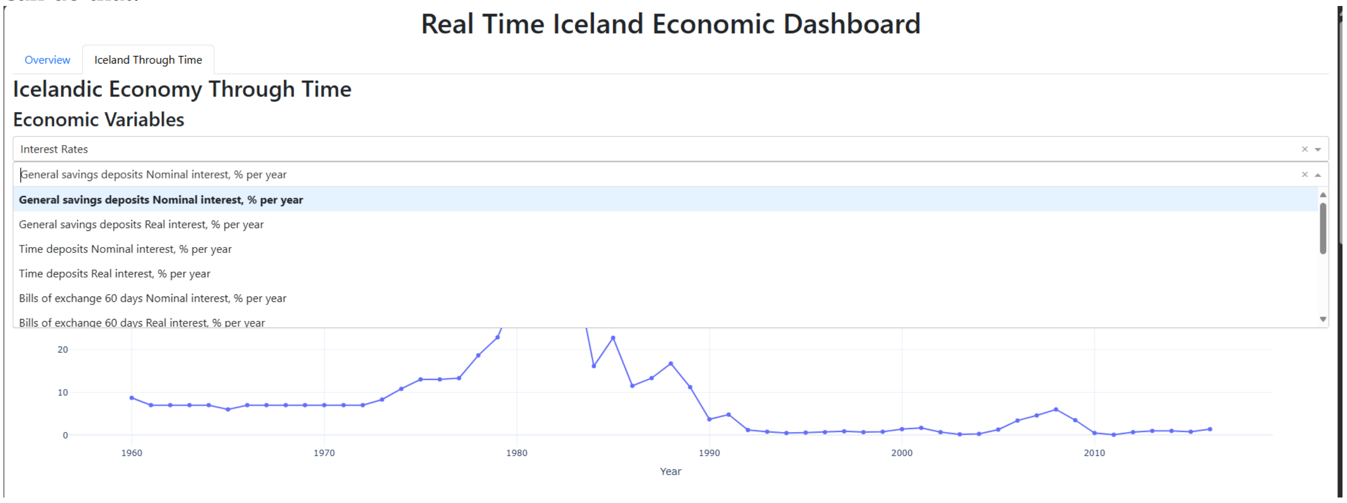
Underneath that, there's a breakdown of the population by gender and nationality. I used a stacked pie chart here, it works for this kind of categorical, proportional data. It's meant to give a feel for the demographic split in a single glance.



There's also another pie chart that shows employment by sector - tourism, fishing, services, and so on. It's based on the latest quarter available. It doesn't have a legend as I feel that it overcrowds the page, since the dashboard is interactive, the user can hover over different sections and callout appears.



The second tab, “Iceland Through Time,” is where the historical data lives. The first visual is a line graph that show how variables have shifted - GDP, CPI, interest rates, and employment trends. You can choose specific subcategories from a dropdown, so if you want to isolate, say, interest rates in real terms instead of nominal, you can do that.



Real Time Iceland Economic Dashboard

Overview | Iceland Through Time

Icelandic Economy Through Time

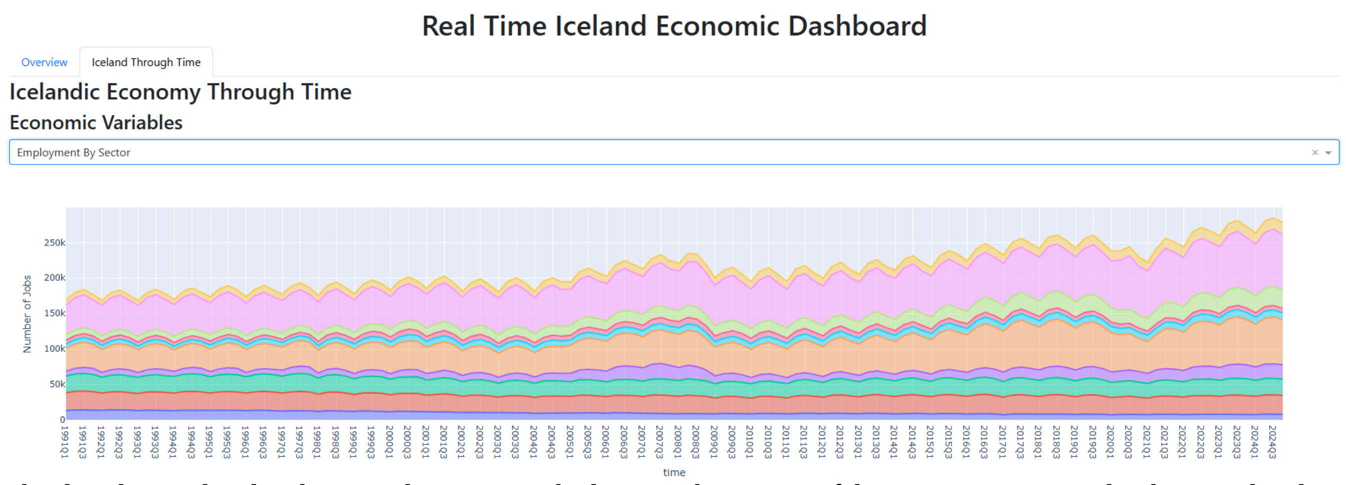
Economic Variables

Interest Rates

General savings deposits Nominal interest, % per year

General savings deposits Nominal interest, % per year
General savings deposits Real interest, % per year
Time deposits Nominal interest, % per year
Time deposits Real interest, % per year
Bills of exchange 60 days Nominal interest, % per year
Bills of exchange 60 days Real interest, % per year

The Employment by Sector graph doesn’t have any subcategories, so the secondary dropdown disappears.



Real Time Iceland Economic Dashboard

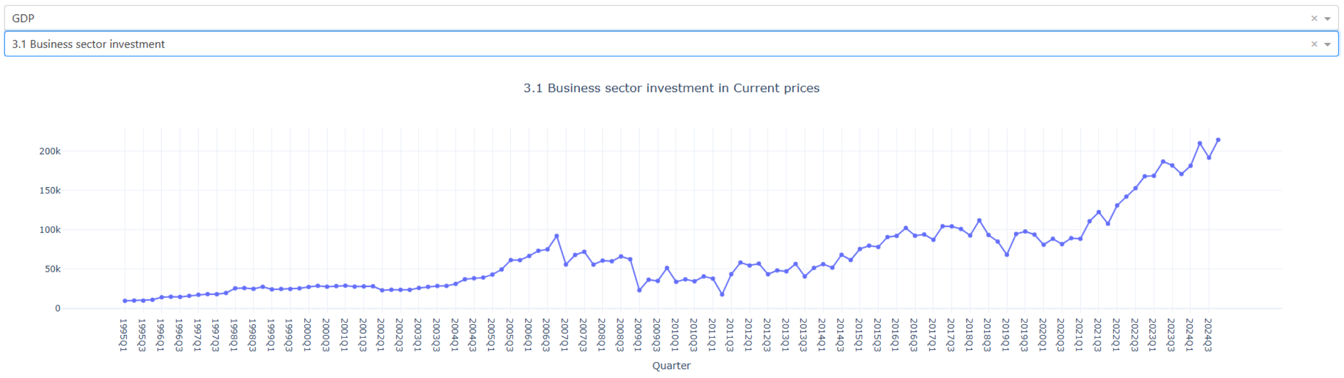
Overview | Iceland Through Time

Icelandic Economy Through Time

Economic Variables

Employment By Sector

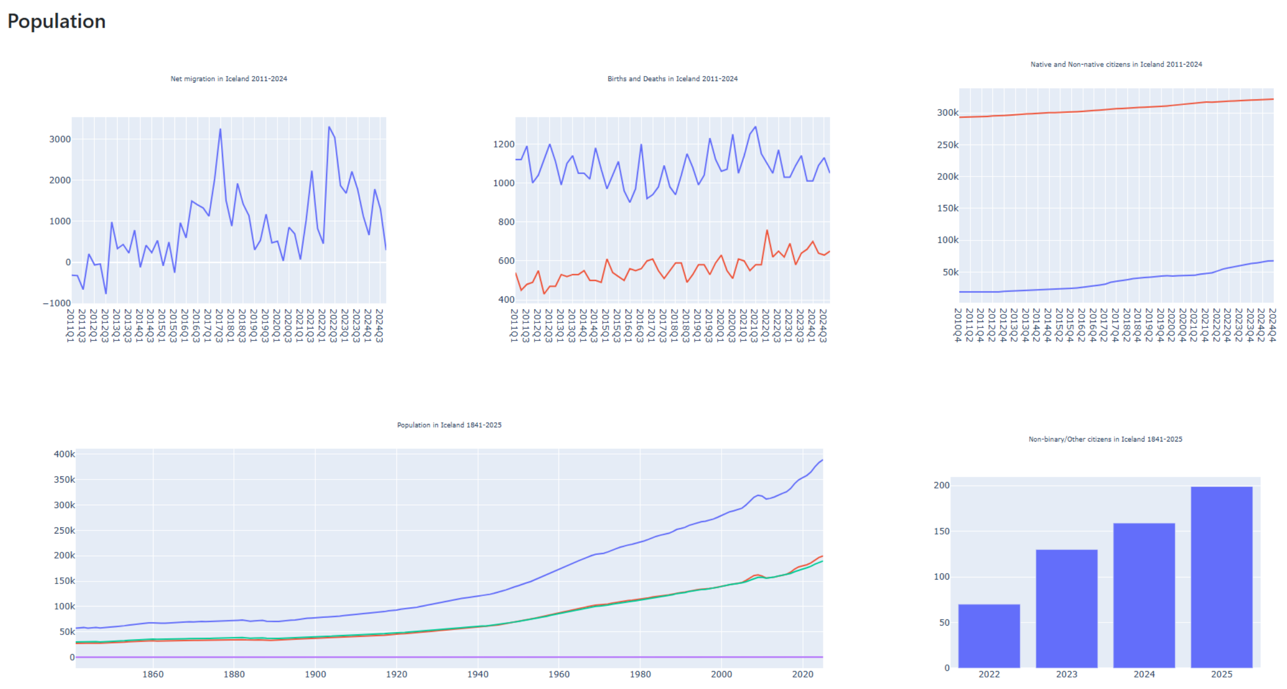
The dropdowns decide what visual appears in the line graph section. I felt it was important to let the user decide what to see, rather than invading them with an overcrowded dashboard.



GDP

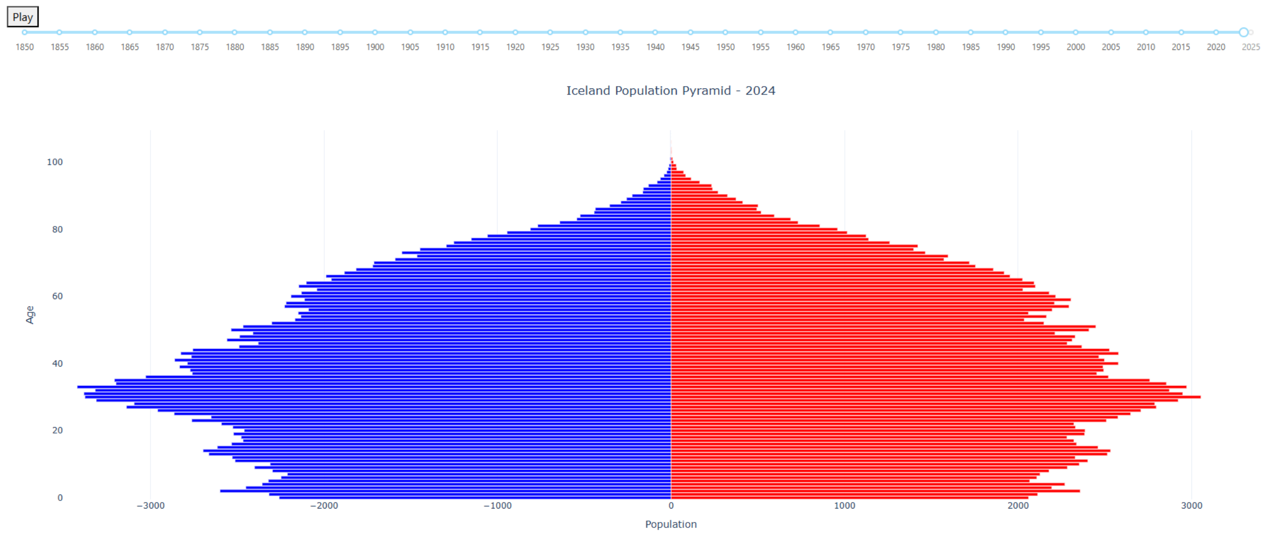
3.1 Business sector investment

For demographic change, there's a whole set of visuals. One line chart tracks net migration - how many people are coming and going. Another compares births and deaths. There's also a chart showing Iceland's total population all the way back to 1841.The non-binary population is tracked separately with a bar chart - the data is limited, but I included it to reflect that it's now being recorded. And finally, there's population pyramid that shows age structure by gender, which you can scroll through by year.



Population

Net migration in Iceland 2011-2024
Births and Deaths in Iceland 2011-2024
Native and Non-native citizens in Iceland 2011-2024
Population in Iceland 1841-2025
Non-binary/Other citizens in Iceland 1841-2025



Notice the play button above which allows the user to play it as an animation.

The main takeaways:
From the economic side: the CPI metric gives a quick sense of inflation - if it's rising, living costs are going up. GDP shows whether the economy is growing or not. A steady rise generally means stability, while sudden dips or spikes might suggest external shocks. The employment breakdown shows what Iceland leans on most - industries like tourism and fishing are major players.

On the demographic side: the population pyramid makes it easy to spot patterns, like an aging population or gender imbalances. A narrowing base (younger people) with a broadening top (older people) might point to future issues with workforce size or dependency ratios. Net migration trends often follow economic conditions - surges in immigration could be tied to job availability or policy shifts. You can also see the baby-boomers being born as you play the animation, which I enjoyed. I find it quite worrying that there are more 30–40-year-olds than 0–10-year-olds. When those 30–40-year-olds retire, there may not be enough taxpayers to support them.

The long-term population graph shows where Iceland's come from - periods of fast growth, times of stagnation. And the births vs deaths chart helps flag whether the country is growing naturally or relying more on migration. As for why I chose these chart types - line charts make the most sense for anything time-based. They show trends, dips, and growth clearly. Pie charts are decent for proportion-based snapshots when the categories are simple and distinct. Bar charts work best when you want to compare clear-cut categories. And the population pyramid is just the clearest way to show age structure - it's a format built for that purpose.
That said, there are a few limitations worth flagging. The dataset has some holes - for example, the non-binary data only exists for recent years, so we can't do long-term analysis there. Similarly, certain economic indicators (like interest rates) don't go as far back as others. Some of the data, like historical population, is static and doesn't update in real time

Overall, the dashboard's job is to make Iceland's economic and demographic data easier to understand. The goal wasn't just to display the numbers but to help viewers see the patterns - where things are steady, where they're shifting, and what that might mean going forward. There's still room to improve, especially as more detailed or recent data becomes available, but I think the current setup strikes a good balance between clarity, depth, and usability.

4. Conclusion

The way the dashboard was put together came down to making the data clear, honest, and useful. Every visual was chosen with a specific reason in mind - whether it was to show a shift over time, break something down into categories, or just give someone a quick snapshot of where things stand right now. Iceland's economy and population data are nuanced, and the dashboard reflects that. Some things are better shown with lines, others with bars or pies, and sometimes it's just best to show the number outright.

The choices weren't just aesthetic - they were shaped by what the data actually allows for. For example, the long-term population trend makes sense as a line because it shows steady change. The population pyramid works because it gives structure to age and gender data in a way that's instantly readable. Where data was limited - like with the non-binary population - the visuals were kept simple and direct so they wouldn't overstate the message. At its core, this dashboard isn't about being flashy - it's about communicating. It's meant to be easy to read, but not oversimplified. That balance is what guided every decision. Of course, there are limits in the data itself, and those were kept in mind throughout. Where granularity was missing, or the data couldn't go back far enough, the dashboard avoids pretending otherwise. The goal was always to stay grounded in what the data could actually support.
The visuals and analysis are built to guide the user through that story without ever getting in the way of it.

5. References

- Statistics Iceland. Official Statistics Portal [Internet]. Reykjavík: Statistics Iceland; [cited 2025 May 4]. Available from: https://www.statice.is/
- Statistics Iceland. StatBank – Data Query Interface [Internet]. Reykjavík: Statistics Iceland; [cited 2025 May 4]. Available from: https://www.statice.is/stat-bank
- Statistics Iceland. PX-Web API Endpoint [Internet]. Reykjavík: Statistics Iceland; [cited 2025 May 4]. Available from: https://px.hagstofa.is/pxen/pxweb/en/
- Statistics Sweden. API for the Statistical Database – Open Data API [Internet]. Stockholm: Statistics Sweden; [cited 2025 May 4]. Available from: https://www.scb.se/en/services/open-data-api/api-for-the-statistical-database/

The first one is a frontend that uses the same API as me, the second one is like an interactive way of requesting data – it helps the user build queries, it shows all possible endpoints. The third one is the endpoint requested from. The fourth one is where I got the documentation from for the API. You will find the documentation pdf in the reports folder.

6. Third-Party Code.

I didn't use any third-party code. I used the libraries (directly and indirectly) specified in the requirements.txt file.