Pyafscgap.org: Open source multi-modal Python-based tools for NOAA AFSC RACE GAP

A Samuel Pottinger[1,*] Giulia Zarpellon[1]

21 March 2023

[1] University of California, Berkeley

[*] Correspondence: A Samuel Pottinger <sam.pottinger@berkeley.edu>

Summary

The Resource Assessment and Conservation Engineering Division of the National Oceanic and Atmospheric Administration's Alaska Fisheries Science Center (NOAA AFSC RACE) runs the Groundfish Assessment Program (GAP) which produces longitudinal catch data (Fisheries, n.d.). These "hauls" report where species are found in the ocean during bottom trawl surveys and in what quantities, empowering ocean health research and fisheries management (Heifetz 2002). Increasing accessibility of these important data (RACEBASE) through a suite of tools for individuals of diverse programming experience, Pyafscgap.org offers not just easier access to the REST API through query compilation but provides both memory-efficient algorithms for "zero-catch inference" and interactive visual analytics tools (Kenney and Roberson 2022; "Oracle Rest Data Services" 2022). Altogether, this toolset supports investigatory tasks not easily executable using the API service alone and, leveraging game and information design, offers these data to a broader audience.

Statement of need

Pyafscgap.org reduces barriers for use of GAP data, offering open source solutions for addressing the dataset's presence-only nature, use of proprietary databases, and size / complexity (Kenney and Roberson 2022).

Developer needs

The `afscgap` package offers easier developer access to the official REST service with automated pagination, query language compilation, and documented types. Together, these tools enable Python developers to use familiar patterns to

interact with these data like type checking, standard documentation, and other Python data-related libraries while freeing programs from structures specific to closed-source Oracle REST Data Services (ORDS) ("Oracle Rest Data Services" 2022).

That being said, access to the API alone cannot support some investigations as the API provides "presence-only" data (Kenney and Roberson 2022). Many types of analysis (like geohash-aggregated species catch per unit effort) require information not just about where a species was present but also where it was not (Niemeyer 2008). In other words, while the presence-only dataset may provide a total weight or count, the total area swept for a region may not necessarily be easily available but required (Pottinger 2023b). The `afscgap` Python package can, with memory efficiency in sometimes millions of data points, algorithmically infer those needed "zero catch" records.

General public needs

Though the `afscgap` Python package makes GAP catch data more accessible to developers, the size and complexity of this dataset requires non-trivial engineering for comparative analysis between species, years, and/or geographic areas (Pottinger 2023b). Therefore, this project also offers visualization tools sitting on top of `afscgap` to begin investigations. Employing information and game design, this tool lowers usability barriers to address broader audiences. Furthermore, it offers both Python code generation as a bridge to `afscgap` and CSV export for those at home in other tools.

Functions

This project's design aims to improve accessibility of GAP catch data, democratizing developer access to the sophisticated methods required to interact with these data and offering inclusive approachable tools to kickstart analysis.

Lazy querying facade

Starting with the `afscgap` library, lazy "generator iterables" increase accessibility of the data by encapsulating logic for memory-efficient pagination and "data munging" behind a familiar interface (Hunner 2019). Furthermore, to support zero catch data, decorators adapt diverse structures to common interfaces to free client code from understanding the full complexities of `afscgap`'s type system (Shvets 2023a).

Finally, offering a single function entry-point into the library with keywords for complex use, this "facade" approach allows the user to interact with these systems without requiring client code to reflect deep understanding of the library's mechanics, a goal furthered by compilation of "standard" Python types to Oracle REST Data Service queries (Shvets 2023b).

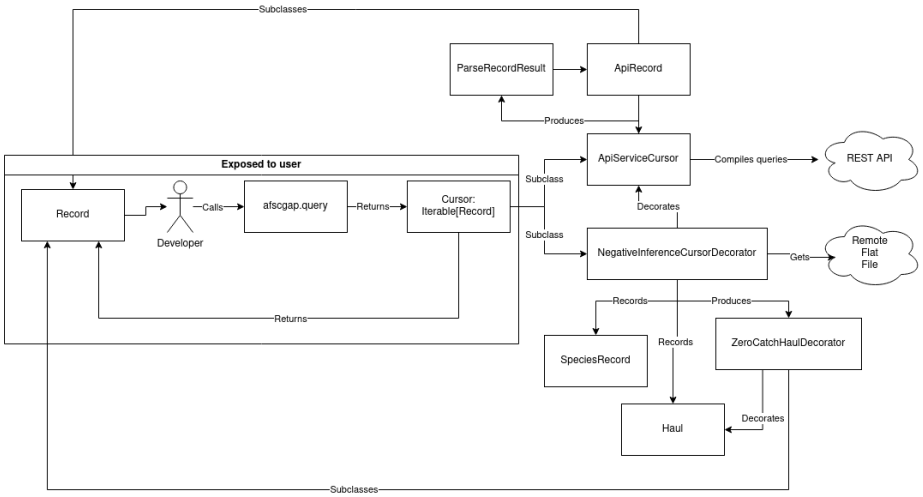

Figure 1: Diagram of simplified afscgap operation ("Jgraph/Drawio: Draw.io Is a Javascript, Client-Side Editor for General Diagramming and Whiteboarding" 2023).

Zero catch inference

"Negative" or "zero catch" uses the following algorithm:

- Paginate while records remain available from the API service.
  - Record species and hauls observed from API-returned results.
  - Return records as available.
- Generate inferred records after API exhaustion.
  - For each species observed in API results, check if it had a record for each haul in a hauls flat file (Pottinger 2023c).
  - For any hauls without the species, produce an 0 catch record from the iterator.

Note that, in adddition to compiling ORDS queries, those queries are also emulated in Python to filter inferred records.

Visualization

Despite these developer-focused tools, zero catch inference may expand this dataset into the millions, demanding technical sophistication to navigate. To further increase accessibility, this project offers visualization tools for temporal, spatial, and species comparisons.

However, building competency in sophisticated interfaces presents user experience challenges and, to that end, this project interprets Hayashida level design via Mark Brown's formalization into an in-tool introduction sequence that directs

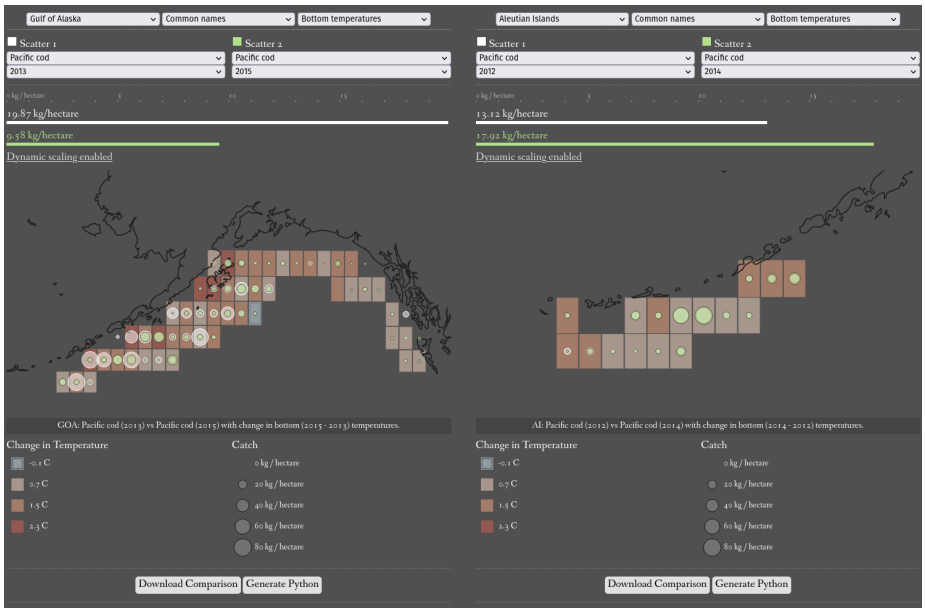

Figure 2: Screenshot of the visualization tool.

the player through a "real" analysis (Nutt and Hayashida 2012; Brown 2015):

- **Introduction**: The player sees information about Pacific cod with prefilled elements used to achieve that analysis gradually fading in.
- **Development**: The tool invites the player using the mechanics introduced moments prior to change the analysis to compare different regions with temperature data.
- **Twist**: Overlays on the same display are enabled, allowing the player to leverage mechanics they just exercised in a now more complex interface.
- **Conclusion**: The tool ends by giving the player an opportunity to demonstrate all of the skills acquired in a new problem.

Finally, while this interface uses game / information design techniques to offer an accessible on-ramp to quickly learn a sophisticated interface, it is also designed as a starting point for continued analysis by generating either CSV or Python code to "take out" work into other tools. Examined via Thinking-aloud Method (Lewis 1982).

Limitations

This library focuses on single-threaded non-asynchoronous utilization and recognizes that aggregation of hauls happens on a latitude / longitude point due to dataset limitations which may cause some approximation in regional CPUE as documented in the visualization's README (Pottinger 2023a).

Acknowledgements

Thanks Carl Boettiger and Fernando Perez for advice in the library. Thanks also to Maya Weltman-Fahs, Brookie Guzder-Williams, and Magali de Bruyn for advice on visualizations. Project of the The Eric and Wendy Schmidt Center for Data Science and the Environment at University of California Berkeley where Kevin Koy is Executive Director. Though the project lists full library credits in its README, authors want to thank runtime dependencies ColorBrewer, D3, Flask, Geolib, Requests, Toolz, and Papa Parse (Brewer et al. 2013; Bostock and Contributors 2023; Mönnich et al. 2023; Joy and Rivard 2021; Reitz 2023; Rocklin, Jacobsen, and Contributors 2022; Holt 2023).

References

Bostock, Mike, and D3 Contributors. 2023. "Data-Driven Documents 7.8.2." *D3.js*. Mike Bostock. https://d3js.org/.

Brewer, Cynthia, Mark Harrower, Ben Sheesley, Andy Woodruff, and David Heyman. 2013. "Colorbrewer 2.0." *ColorBrewer*. The Pennsylvania State University. https://colorbrewer2.org.

Brown, Mark. 2015. "Super Mario 3D World's 4 Step Level Design." *Game Maker's Toolkit*. YouTube. https://www.youtube.com/watch?v=dBmIkEvEBtA.

Fisheries, NOAA. n.d. "Groundfish Assessment Program." National Oceanic; Atmospheric Administration. https://www.fisheries.noaa.gov/contact/groundfish-assessment-program.

Heifetz, Jonathan. 2002. "Coral in Alaska: Distribution, Abundance, and Species Associations." *Hydrobiologia* 471 (1/3): 19–28. https://doi.org/10.1023/a:1016528631593.

Holt, Matt. 2023. "Papa Parse - Powerful Csv Parser for Javascript." *Papa Parse*. Matt Holt. https://www.papaparse.com/.

Hunner, Trey. 2019. "Lazy Looping in Python: Making and Using Generators and Iterators." *Pycon 2019*. Python Software Foundation. https://pycon2019.trey.io/index.html.

"Jgraph/Drawio: Draw.io Is a Javascript, Client-Side Editor for General Diagramming and Whiteboarding." 2023. *GitHub*. GitHub, Inc. https://github.com/jgraph/drawio.

Joy, Anu, and Éloi Rivard. 2021. "Joyanujoy/Geolib: Python Geohash Library." *GitHub*. GitHub Inc. https://github.com/joyanujoy/geolib.

Kenney, Heather, and Nancy Roberson. 2022. "AFSC/Race/Gap: Racebase Database." *InPort*. Fisheries Information System. https://www.fisheries.noaa.gov/inport/item/22008.

Lewis, Clayton. 1982. “Using the "Thinking-Aloud" Method in Cognitive Interface Design.” IBM TJ Watson Research Center. https://dominoweb.draco.res.ibm.com/2513e349e05372cc852574ec0051eea4.html.

Mönnich, Adrian, Armin Ronacher, David Lord, Grey Li, Joshua Bronson, Markus Unterwaditzer, Philip Jones, and Flask Contributors. 2023. “Welcome to Flask.” *Pallets Projects*. Pallets. https://flask.palletsprojects.com/en/2.2.x/.

Niemeyer, Gustavo. 2008. “Geohash.org Is Public!” *Labix Blog*. Labix. https://blog.labix.org/2008/02/26/geohashorg-is-public.

Nutt, Christian, and Koichi Hayashida. 2012. “The Structure of Fun: Learning from Super Mario 3D Land’s Director.” *Game Developer*. Informa. https://www.gamedeveloper.com/design/the-structure-of-fun-learning-from-i-super-mario-3d-land-i-s-director.

“Oracle Rest Data Services.” 2022. *Oracle Help Center*. Oracle. https://docs.oracle.com/en/database/oracle/oracle-rest-data-services/.

Pottinger, A S. 2023a. “AFSC Gap Viz Readme.md.” *GitHub*. GitHub, Inc. https://github.com/SchmidtDSE/afscgap/blob/main/afscgapviz/README.md.

———. 2023b. “Cod Afsc Gap Example.” *MyBinder*. The Binder Team. https://hub.gke2.mybinder.org/user/schmidtdse-afscgap-ia1m4wd3/notebooks/index.ipynb.

———. 2023c. “Python Noaa Afsc Gap Tools (Downloads).” *AFSC GAP Tools for Python*. University of California Berkeley. https://pyafscgap.org/#downloads.

Reitz, Kenneth. 2023. “HTTP for Humans.” *Requests*. Requests Project. https://docs.python-requests.org/en/latest/index.html.

Rocklin, Matthew, John Jacobsen, and Toolz Contributors. 2022. “Pytoolz Api Documentation.” *Read the Docs*. Read the Docs, Inc. https://toolz.readthedocs.io/en/latest/.

Shvets, Alexander. 2023a. “Decorator.” *Refactoring.Guru*. Refactoring.Guru. https://refactoring.guru/design-patterns/decorator.

———. 2023b. “Facade.” *Refactoring.Guru*. Refactoring.Guru. https://refactoring.guru/design-patterns/facade.