The Journal of Open Source Software

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

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

1 University of California, Berkeley, California, United States of America

DOI: 10.21105/joss.05593

Software

- Review
- Repository
- Archive

Editor: Kevin M. Moerman

Reviewers:

- @Kevin-Mattheus-Moerman

Submitted: 01 June 2023
Published: 28 June 2023

License
Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License (CC BY 4.0).

Summary

NOAA AFSC's Groundfish Assessment Program produces longitudinal catch data which support ocean health research and fisheries management (Fisheries, n.d.). These "hauls" report in what quantities and locations bottom trawl surveys find different marine species along with environmental conditions at the time and place of observation (Heifetz, 2002). Increasing usability for communities of diverse programming experience, Pyafscgap.org offers query language compilation, memory-efficient algorithms for "zero-catch" inference, and interactive visual analytics for these economically and scientifically important GAP datasets. Altogether, this research toolset supports investigatory tasks across survey programs' locations and broadens access through game and information design.

Statement of need

Pyafscgap.org reduces barriers for use of NOAA AFSC RACE GAP[1] data, offering:

- Improved developer usability.
- Memory-efficient algorithms for zero catch inference.
- Zero-code visualization tools.

Altogether, these open source tools extend the reach and approachability of GAP's multiple survey programs to support analysis like longitudinal catch per unit effort (CPUE) in context of environmental changes (Pottinger, 2023b).

Developer usability

Working with these data requires knowledge of tools outside the Python "standard toolset" like closed-source ORDS query language ("Oracle Rest Data Services," 2022). While the `afscgap` package offers easier access to the official REST service, it also crucially offers ORDS compilation, documented types, and lazy access to these large datasets. Together, these tools enable Python developers to efficiently use familiar patterns to interact with these data: type checking, standard documentation, and compatibility with common Python data-related libraries.

Record inference

Surveys on their own within the API struggle supporting some investigations as they provide "presence-only" data (Kenney & Roberson, 2022). For example, the API may readily yield total mass of Pacific cod but not its geohash-aggregated CPUE (Niemeyer, 2008).

[1] Groundfish Assessment Program in the Resource Assessment and Conservation Engineering Division of the National Oceanic and Atmospheric Administration's Alaska Fisheries Science Center

$$CPUE_{species} = \frac{m_{species}}{A_{swept}}$$

Metrics like CPUE need "absence data" or hauls in which the species was not recorded. This package can efficiently infer those results (Pottinger, 2023b).

Broad accessibility

Though the afscgap Python package makes GAP catch information more accessible, the data's size and complexity complicates comparative analysis between species, years, and/or geographic areas (Pottinger, 2023b). Without deep developer experience, it may still be difficult to get started even with scientific background. To address a broader audience, this project offers visualization on top of afscgap with CSV and Python code export as a bridge to further analysis.

Functions

This project improves accessibility of GAP data and offers approachable tools to kickstart analysis.

Efficient facade

The afscgap library manages significant complexity to offer a simple familiar interface to Python developers:

- Lazy "generator iterables" increase accessibility by encapsulating logic for memory-efficient pagination and "data munging" behind Python-standard iterators (Hunner, 2019).
- Decorators adapt diverse structures to common interfaces in zero catch data, offering polymorphism that helps to reduce the complexity of code using the library (Shvets, 2023a).
- Providing a single object entry-point into the library, a "facade" frees users from needing deep understanding of the library's types and transparently compiles "standard" Python types to Oracle REST Data Service queries (Shvets, 2023b).

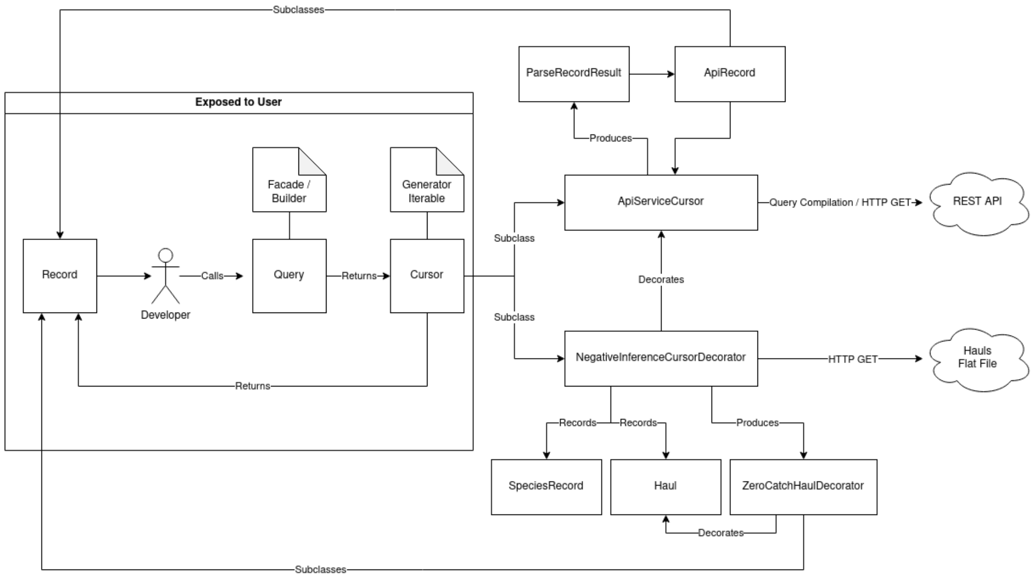


Figure 1: Diagram of afscgap.

Zero catch inference

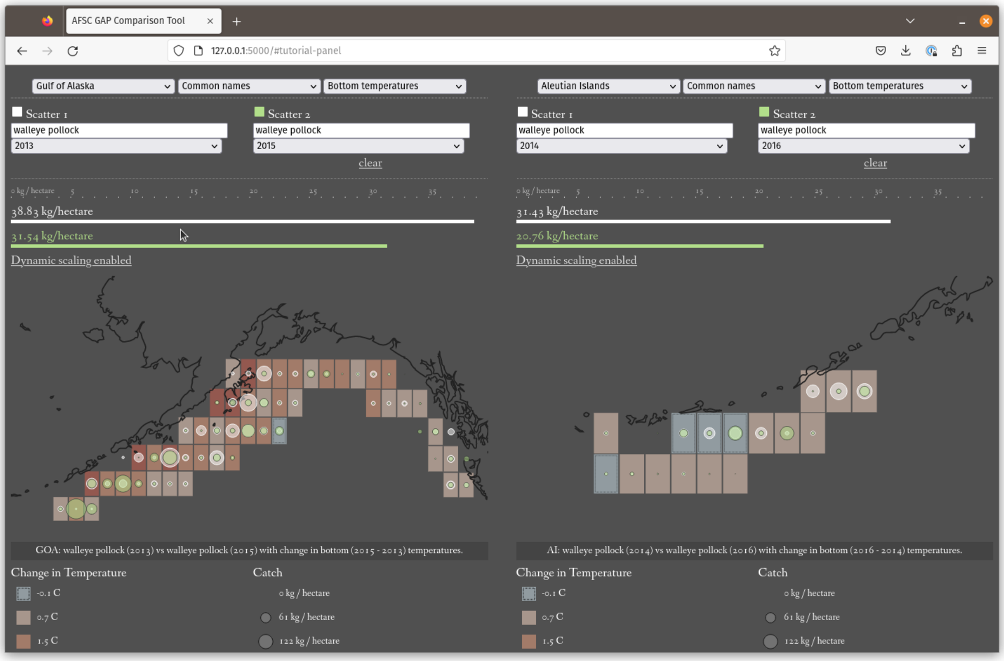
"Zero catch" inference enables a broader range of analysis with the following algorithm:

- Lazily paginate while records remain available from the API service.
  - Record species and hauls observed from API-returned results.
  - Return records as available.
- Lazily 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 a record from the iterator.

Note `afscgap` performs Python-emulation of ORDS filters on inferred records.

Visualization

These complex data require technical sophistication to navigate and, to increase accessibility, visualization tools help start temporal, spatial, and species comparisons with deep linking, coordinated highlighting, separated color channels, summary statistics, and side-by-side display (Few, 2010).



GOA: walleye pollock (2013) vs walleye pollock (2015) with change in bottom (2015 - 2013) temperatures.

AI: walleye pollock (2014) vs walleye pollock (2016) with change in bottom (2016 - 2014) temperatures.

Figure 2: Visualization screenshot.

To support learning this UI, an optional introduction sequence tutorializes a "real" analysis via Hayashida design (Brown, 2015; Nutt & Hayashida, 2012):

- **Introduction**: The tool shows information about Pacific cod with pre-filled controls used to achieve that analysis gradually fading in, asking the user for minor modifications.
- **Development**: Using the mechanics introduced moments prior, the tool invites the user to change the analysis to compare different regions.
- **Twist**: Enabling overlays on the same display, the user leverages mechanics they just exercised in a now more complex interface.
- **Conclusion**: The visualization invites the user to demonstrate skills acquired in a new problem.

This visualization also serves as a starting point for continued analysis by generating either CSV or Python code to take work into other tools.

In addition to use in a graduate classroom setting, five individuals with relevant background offered feedback on this open source visualization with four aided by a think-aloud prompt[2] (Lewis, 1982).

Limitations

As further documented in the repository (Pottinger, 2023a), these tools:

- Run single-threaded and synchronous.
- Aggregate hauls as points in visualization due to data limitation.
- Ignore hauls if entirely excluded by NOAA.

Acknowledgments

Thanks to:

- Runtime dependencies: ColorBrewer, D3, Flask, Geolib, Requests, Toolz, and Papa Parse (Bostock & Contributors, 2023; Brewer et al., 2013; Holt, 2023; Joy & Rivard, 2021; Mönnich et al., 2023; Reitz, 2023; Rocklin et al., 2022).
- Library advice: Carl Boettiger, Fernando Perez, and PyOpenSci reviewers.
- Visualization advice: Magali de Bruyn, Brookie Guzder-Williams, Angela Hayes, David Joy, and Maya Weltman-Fahs.
- Lewis Barnett, Emily Markowitz, and Ciera Martinez for general guidance.
- Draw.io for diagrams (JGraph & draw.io, 2023).

Project of The Eric and Wendy Schmidt Center for Data Science and the Environment at UC Berkeley.

References

Bostock, M., & Contributors, D. (2023). Data-driven documents 7.8.2. In *D3.js*. Mike Bostock. https://d3js.org/

Brewer, C., Harrower, M., Sheesley, B., Woodruff, A., & Heyman, D. (2013). Colorbrewer 2.0. In *ColorBrewer*. The Pennsylvania State University. https://colorbrewer2.org

Brown, M. (2015). Super mario 3D world's 4 step level design. In *Game Maker's Toolkit*. YouTube. https://www.youtube.com/watch?v=dBmIkEvEBtA

Few, S. (2010). Coordinated highlighting in context. In *Visual Business Intelligence Newsletter*. Perceptual Edge. https://www.perceptualedge.com/articles/visual_business_intelligence/coordinated_highlighting_in_context.pdf

Fisheries, N. (n.d.). *Groundfish assessment program*. National Oceanic; Atmospheric Administration. https://www.fisheries.noaa.gov/contact/groundfish-assessment-program

Heifetz, J. (2002). Coral in alaska: Distribution, abundance, and species associations. *Hydrobiologia*, *471*(1/3), 19–28. https://doi.org/10.1023/a:1016528631593

Holt, M. (2023). Papa parse - powerful CSV parser for JavaScript. In *Papa Parse*. Matt Holt. https://www.papaparse.com/

[2]Discussion limited to tool-specific needs assessment / quality improvement, collecting information about the tool and not individuals. IRB questionnaire on file finds "project does not constitute human subjects research" and review is not required.

Hunner, T. (2019). Lazy looping in python: Making and using generators and iterators. In *Pycon 2019*. Python Software Foundation. https://pycon2019.trey.io/index.html

JGraph, & draw.io. (2023). Jgraph/drawio: Draw.io is a JavaScript, client-side editor for general diagramming and whiteboarding. In *GitHub*. GitHub, Inc. https://github.com/jgraph/drawio

Joy, A., & Rivard, É. (2021). Joyanujoy/geolib: Python geohash library. In *GitHub*. GitHub Inc. https://github.com/joyanujoy/geolib

Kenney, H., & Roberson, N. (2022). AFSC/race/gap: Racebase database. In *InPort*. Fisheries Information System. https://www.fisheries.noaa.gov/inport/item/22008.

Lewis, C. (1982). *Using the "thinking-aloud" method in cognitive interface design*. IBM TJ Watson Research Center. https://dominoweb.draco.res.ibm.com/2513e349e05372cc852574ec0051eea4.html

Mönnich, A., Ronacher, A., Lord, D., Li, G., Bronson, J., Unterwaditzer, M., Jones, P., & Contributors, F. (2023). Welcome to flask. In *Pallets Projects*. Pallets. https://flask.palletsprojects.com/en/2.2.x/

Niemeyer, G. (2008). Geohash.org is public! In *Labix Blog*. Labix. https://blog.labix.org/2008/02/26/geohashorg-is-public

Nutt, C., & Hayashida, K. (2012). The structure of fun: Learning from super mario 3D land's director. In *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). In *Oracle Help Center*. Oracle. https://docs.oracle.com/en/database/oracle/oracle-rest-data-services/

Pottinger, A. S. (2023a). AFSC GAP viz README.md. In *GitHub*. GitHub, Inc. https://github.com/SchmidtDSE/afscgap/blob/main/afscgapviz/README.md

Pottinger, A. S. (2023b). Cod AFSC GAP example. In *MyBinder*. The Binder Team. https://hub.gke2.mybinder.org/user/schmidtdse-afscgap-ia1m4wd3/notebooks/index.ipynb

Pottinger, A. S. (2023c). Python NOAA AFSC GAP tools (downloads). In *AFSC GAP Tools for Python*. University of California Berkeley. https://pyafscgap.org/#downloads

Reitz, K. (2023). HTTP for humans. In *Requests*. Requests Project. https://docs.python-requests.org/en/latest/index.html

Rocklin, M., Jacobsen, J., & Contributors, T. (2022). Pytoolz API documentation. In *Read the Docs*. Read the Docs, Inc. https://toolz.readthedocs.io/en/latest/

Shvets, A. (2023a). Decorator. In *Refactoring.Guru*. Refactoring.Guru. https://refactoring.guru/design-patterns/decorator

Shvets, A. (2023b). Facade. In *Refactoring.Guru*. Refactoring.Guru. https://refactoring.guru/design-patterns/facade