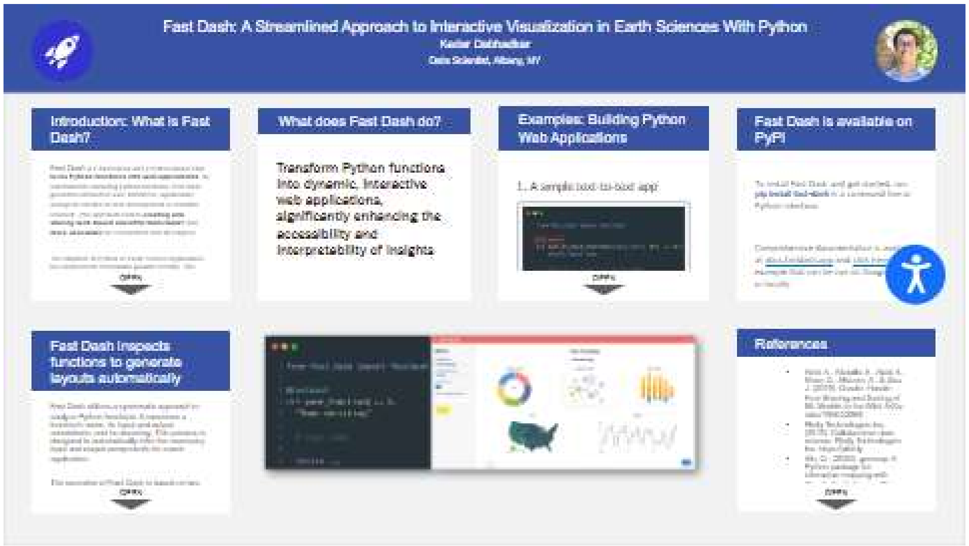
Fast Dash: A Streamlined Approach to Interactive Visualization in Earth Sciences With Python

Kedar Dabhadkar[1]

[1]Affiliation not available

January 13, 2024

Fast Dash: A Streamlined Approach to Interactive Visualization in Earth Sciences With Python



Fast Dash: A Streamlined Approach to Interactive Visualization in Earth Sciences With Python
Kedar Dabhadkar
Data Scientist, Albany, NY

Introduction: What is Fast Dash?

What does Fast Dash do?

Transform Python functions into dynamic, interactive web applications, significantly enhancing the accessibility and interpretability of insights

Examples: Building Python Web Applications

1. A simple text-to-text app

Fast Dash is available on PyPI

Fast Dash inspects functions to generate layouts automatically

References

Kedar Dabhadkar

Data Scientist, Albany, NY

PRESENTED AT:

AGU23 WIDE. OPEN. SCIENCE.

INTRODUCTION: WHAT IS FAST DASH?

Fast Dash is a framework and a Python library that **turns Python functions into web applications**. By automatically analyzing Python functions, Fast Dash generates interactive user interfaces, significantly easing the burden of web development in scientific research. This approach makes **creating and sharing web-based scientific tools faster** and **more accessible** for researchers and developers.

The adoption of Python in Earth Science applications has experienced remarkable growth recently. This surge is largely due to the integration of Python into various scientific, machine learning, and data visualization tools and libraries. This integration has made Python a preferred choice for Earth Scientists who wish to conduct complex experiments without the need to learn multiple programming languages.

However, Python's application in web development has not kept pace with its growth in other areas. This is primarily because Python was not originally designed with web development as a core focus. Popular frameworks like Django and Flask, while powerful, often require additional languages like HTML and JavaScript for front-end development, complicating the process for those focused on Python.

To address this challenge, frameworks such as Dash, Streamlit, and Gradio have been developed. These tools offer more Python-centric approaches to building web applications but often have steep learning curves. As a result, scientists and developers need to learn additional tools outside their primary scientific programming expertise to effectively share their work with a wider audience.

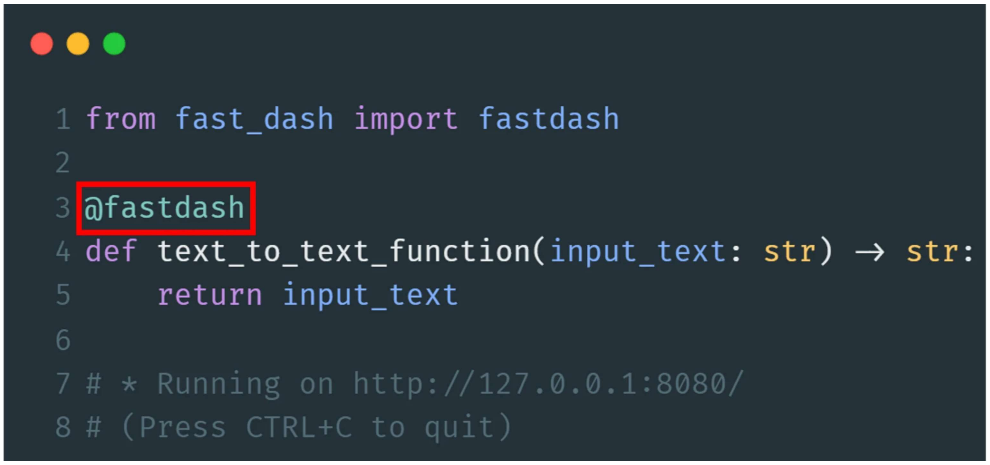
Fast Dash is introduced as a solution to this problem. It is currently based on Plotly's Dash but can be easily extended to a different web development framework.

WHAT DOES FAST DASH DO?

Transform Python functions into dynamic, interactive web applications, significantly enhancing the accessibility and interpretability of insights

EXAMPLES: BUILDING PYTHON WEB APPLICATIONS

1. A simple text-to-text app

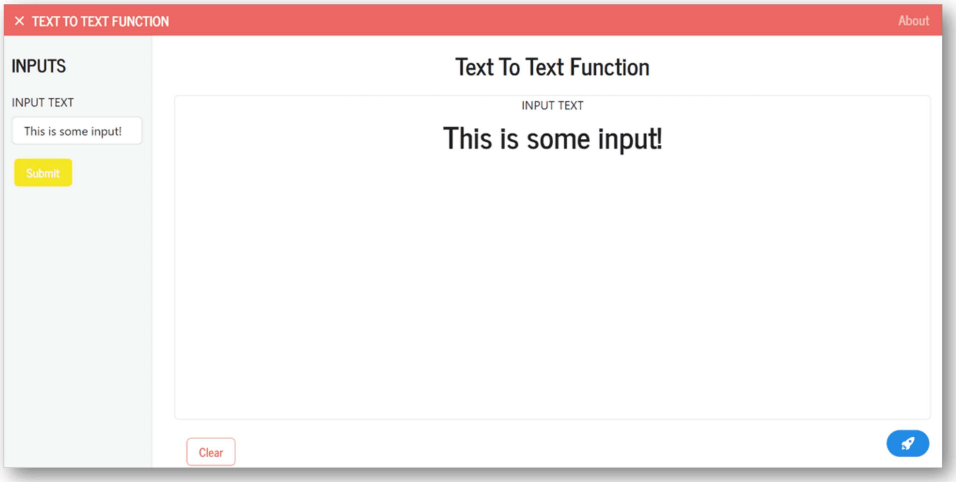


```python
from fast_dash import fastdash

@fastdash
def text_to_text_function(input_text: str) → str:
    return input_text

# * Running on http://127.0.0.1:8080/
# (Press CTRL+C to quit)
```

Figure 1. Python code to build a simple text-to-text web application

In this simple example, adding a "@fastdash" to your regular Python function and running the script deploys a simple Fast Dash app!

The result of running this function is:



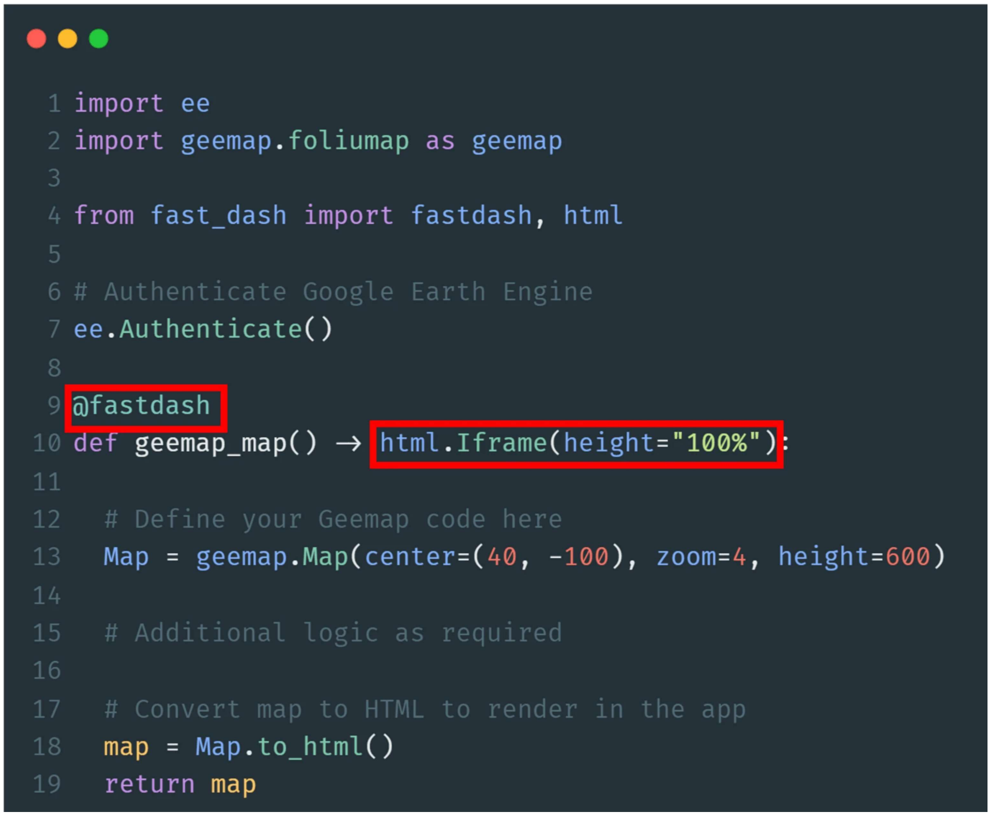
This is some input!

This is some input!

Figure 2. The result of running the Python code shown in Figure 1.

2. Deploying Geemap

Similarly, any map based on Geemap, which extends Google Earth Engine's functionality to a Python interface, can be deployed using the code shown in Figure 3.



```python
import ee
import geemap.foliumap as geemap

from fast_dash import fastdash, html

# Authenticate Google Earth Engine
ee.Authenticate()

@fastdash
def geemap_map() -> html.Iframe(height="100%"):

  # Define your Geemap code here
  Map = geemap.Map(center=(40, -100), zoom=4, height=600)

  # Additional logic as required

  # Convert map to HTML to render in the app
  map = Map.to_html()
  return map
```

Figure 3. Python code to build a simple Geemap-based mapping application

The result of running code shown in Figure 3 can be seen in Figure 4.

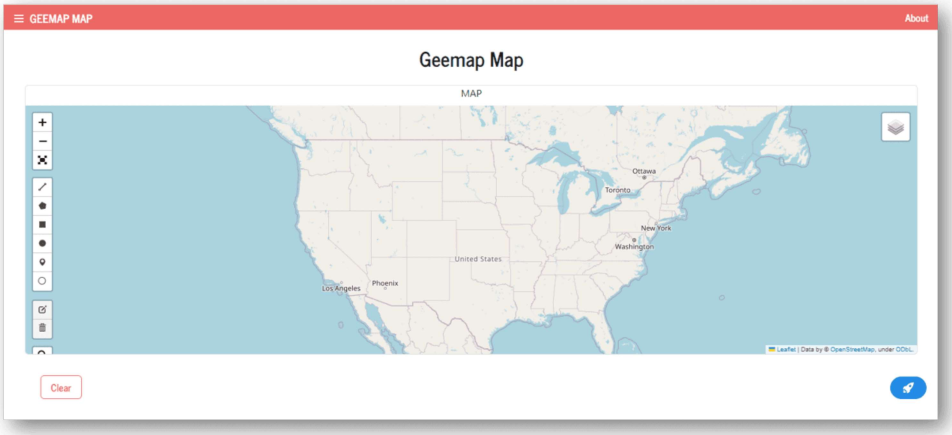


Figure 4. The result of running the code shown in Figure 3

FAST DASH IS AVAILABLE ON PYPI

To install Fast Dash and get started, run **pip install fast-dash** in a command line or Python interface.

Comprehensive documentation is available at docs.fastdash.app (https://docs.fastdash.app) and click here (https://docs.fastdash.app/Examples/01_simple_text_to_text/) for example that can be run on Google Colab or locally.

FAST DASH INSPECTS FUNCTIONS TO GENERATE LAYOUTS AUTOMATICALLY

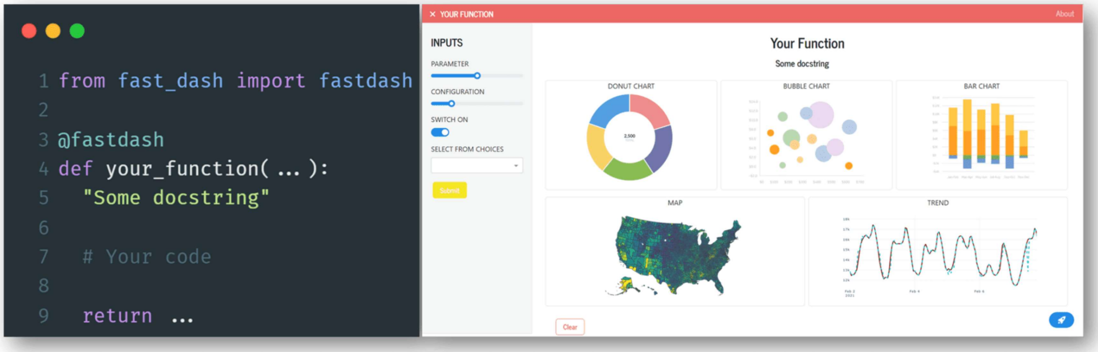
Fast Dash utilizes a systematic approach to analyze Python functions. It examines a function's name, its input and output annotations, and its docstring. This process is designed to automatically infer the necessary input and output components for a web application.

The operation of Fast Dash is based on two core principles:

1. **Origin of Web Applications**: It posits that every Python-based web application is derived from a single Python function. This principle emphasizes the potential of a Python function as the foundational element of web application development.
2. **Function Annotations as Information Source:** The framework recognizes that a well-annotated Python function contains all the essential information needed to create an interactive web application. Annotations provide insights into the function's purpose, inputs, outputs, and operational specifics.

In adherence to these principles, Fast Dash meticulously extracts details from a Python function. It analyzes aspects such as the function's name, its docstring, and the type hints associated with inputs and outputs. This comprehensive examination enables Fast Dash to automatically generate a user-friendly layout and create an interactive web application.

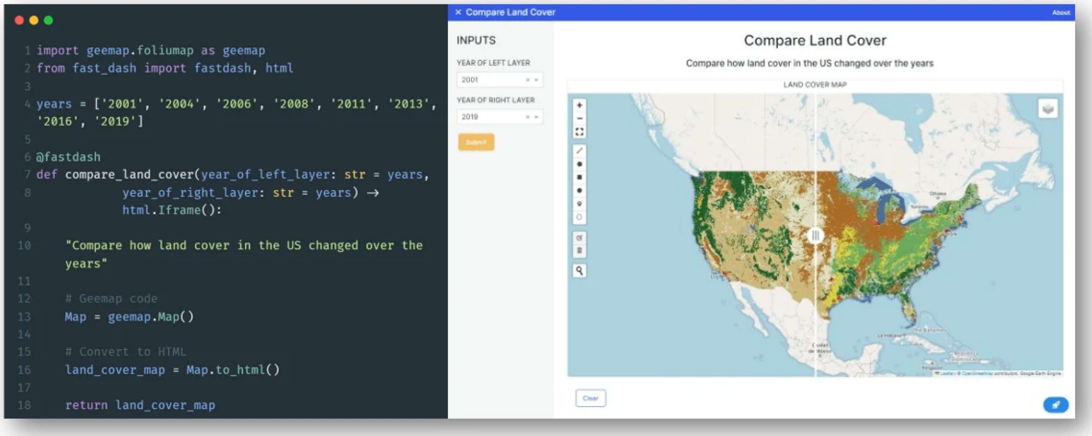
Moreover, the synergy between the callback function and the configuration settings of the Fast Dash app plays a pivotal role in determining the application's deployment and interactive capabilities. A function with detailed annotations often minimizes the need for extensive customization of app configurations, simplifying the development process and enhancing user interaction.



```python
from fast_dash import fastdash

@fastdash
def your_function( ... ):
  "Some docstring"

  # Your code

  return ...
```

Your Function

Some docstring



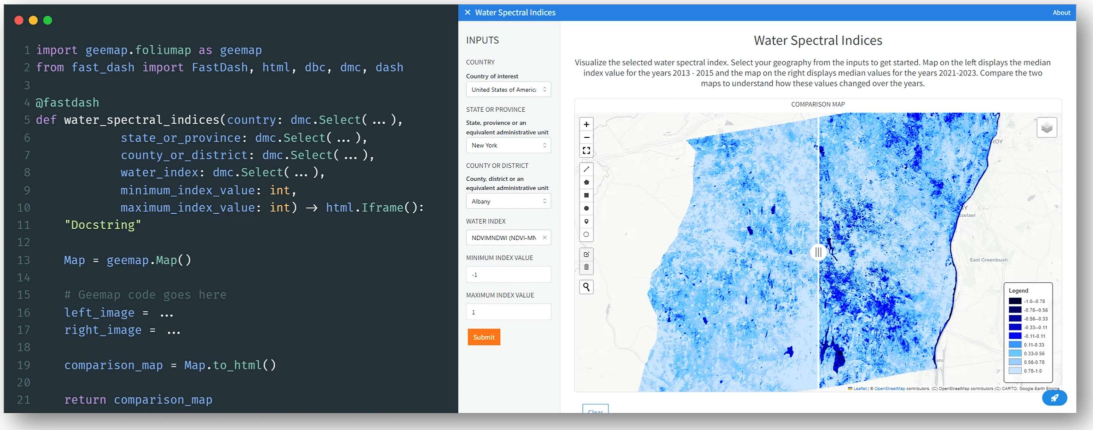
```python
import geemap.foliumap as geemap
from fast_dash import fastdash, html

years = ['2001', '2004', '2006', '2008', '2011', '2013',
'2016', '2019']

@fastdash
def compare_land_cover(year_of_left_layer: str = years,
            year_of_right_layer: str = years) →
            html.Iframe():

    "Compare how land cover in the US changed over the
    years"

    # Geemap code
    Map = geemap.Map()

    # Convert to HTML
    land_cover_map = Map.to_html()

    return land_cover_map
```

Compare Land Cover

Compare how land cover in the US changed over the years

YEAR OF LEFT LAYER: 2001

YEAR OF RIGHT LAYER: 2019



```python
import geemap.foliumap as geemap
from fast_dash import FastDash, html, dbc, dmc, dash

@fastdash
def water_spectral_indices(country: dmc.Select( … ),
            state_or_province: dmc.Select( … ),
            county_or_district: dmc.Select( … ),
            water_index: dmc.Select( … ),
            minimum_index_value: int,
            maximum_index_value: int) → html.Iframe():
    "Docstring"

    Map = geemap.Map()

    # Geemap code goes here
    left_image = …
    right_image = …

    comparison_map = Map.to_html()

    return comparison_map
```

✕ Water Spectral Indices About

INPUTS

COUNTRY
Country of interest: United States of Americ

STATE OR PROVINCE
State, provience or an equivalent administrative unit: New York

COUNTY OR DISTRICT
County, district or an equivalent administrative unit: Albany

WATER INDEX: NDVIMNDWI (NDVI-MN ✕

MINIMUM INDEX VALUE: -1

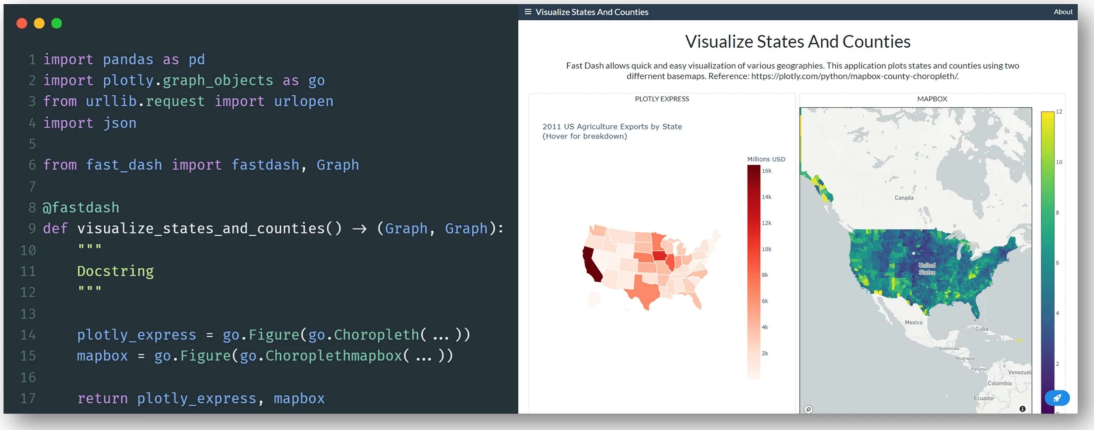
MAXIMUM INDEX VALUE: 1

Submit

Water Spectral Indices

Visualize the selected water spectral index. Select your geography from the inputs to get started. Map on the left displays the median index value for the years 2013 - 2015 and the map on the right displays median values for the years 2021-2023. Compare the two maps to understand how these values changed over the years.

COMPARISON MAP

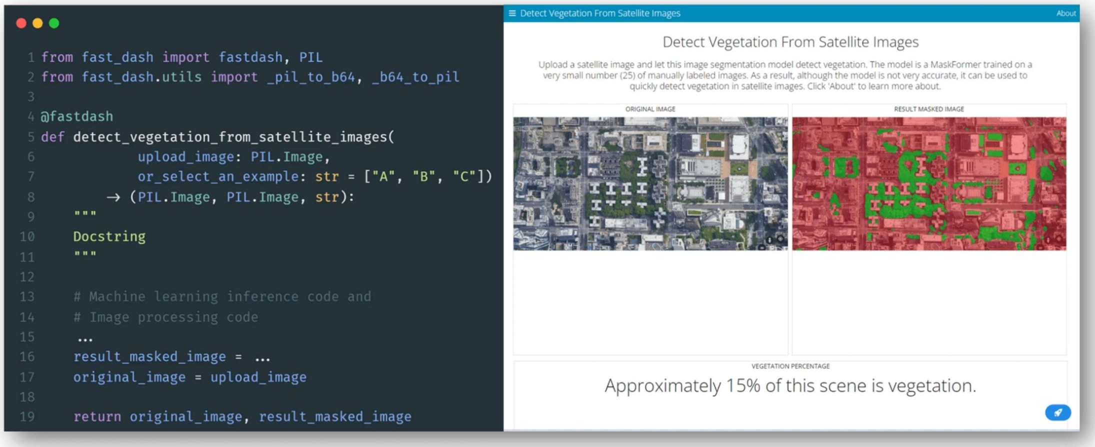


```python
import pandas as pd
import plotly.graph_objects as go
from urllib.request import urlopen
import json

from fast_dash import fastdash, Graph

@fastdash
def visualize_states_and_counties() → (Graph, Graph):
    """
    Docstring
    """

    plotly_express = go.Figure(go.Choropleth( … ))
    mapbox = go.Figure(go.Choroplethmapbox( … ))

    return plotly_express, mapbox
```

☰ Visualize States And Counties About

Visualize States And Counties

Fast Dash allows quick and easy visualization of various geographies. This application plots states and counties using two differnent basemaps. Reference: https://plotly.com/python/mapbox-county-choropleth/.



```python
from fast_dash import fastdash, PIL
from fast_dash.utils import _pil_to_b64, _b64_to_pil

@fastdash
def detect_vegetation_from_satellite_images(
            upload_image: PIL.Image,
            or_select_an_example: str = ["A", "B", "C"])
        → (PIL.Image, PIL.Image, str):
    """
    Docstring
    """

    # Machine learning inference code and
    # Image processing code
    …
    result_masked_image = …
    original_image = upload_image

    return original_image, result_masked_image
```

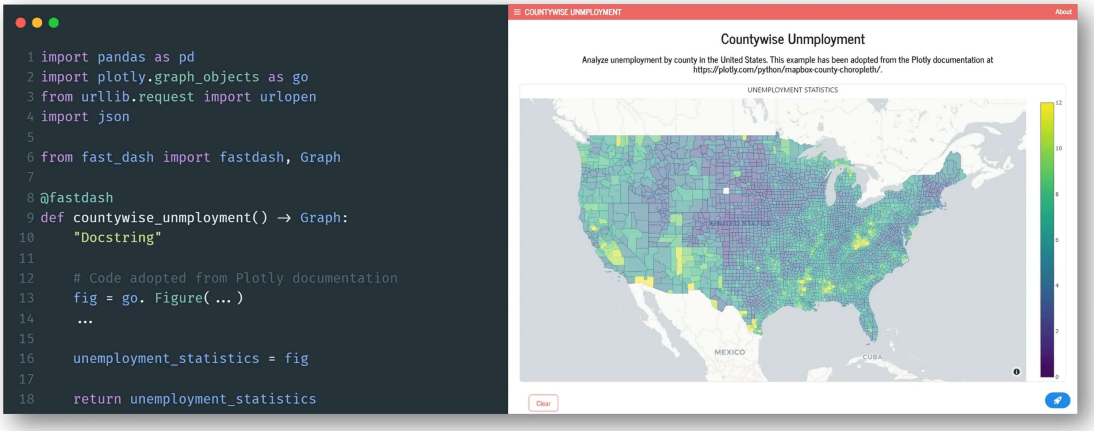
☰ Detect Vegetation From Satellite Images About

Detect Vegetation From Satellite Images

Upload a satellite image and let this image segmentation model detect vegetation. The model is a MaskFormer trained on a very small number (25) of manually labeled images. As a result, although the model is not very accurate, it can be used to quickly detect vegetation in satellite images. Click 'About' to learn more about.

VEGETATION PERCENTAGE

Approximately 15% of this scene is vegetation.

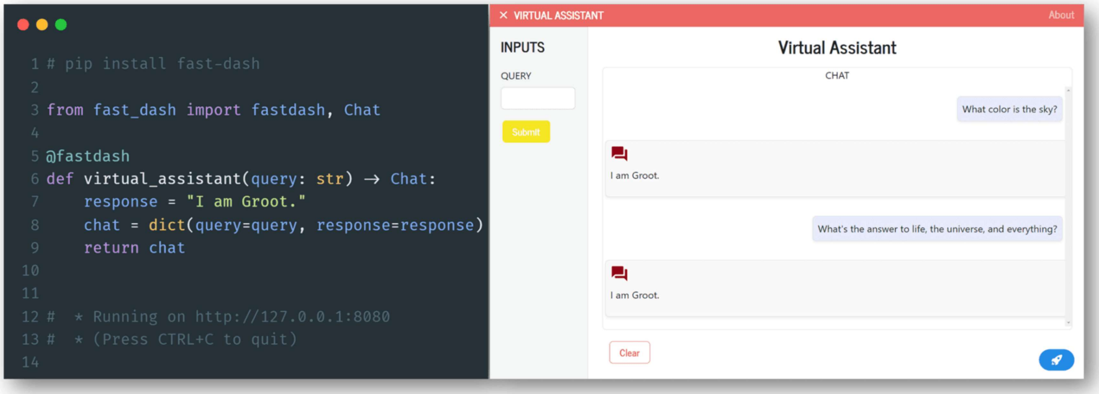


```python
import pandas as pd
import plotly.graph_objects as go
from urllib.request import urlopen
import json

from fast_dash import fastdash, Graph

@fastdash
def countywise_unmployment() → Graph:
    "Docstring"

    # Code adopted from Plotly documentation
    fig = go. Figure( … )
    …

    unemployment_statistics = fig

    return unemployment_statistics
```

☰ COUNTYWISE UNMPLOYMENT About

Countywise Unmployment

Analyze unemployment by county in the United States. This example has been adopted from the Plotly documentation at https://plotly.com/python/mapbox-county-choropleth/.



```
# pip install fast-dash

from fast_dash import fastdash, Chat

@fastdash
def virtual_assistant(query: str) -> Chat:
    response = "I am Groot."
    chat = dict(query=query, response=response)
    return chat


#  * Running on http://127.0.0.1:8080
#  * (Press CTRL+C to quit)

```

Virtual Assistant

CHAT

What color is the sky?

I am Groot.

What's the answer to life, the universe, and everything?

I am Groot.

REFERENCES

- Abid, A., Abdalla, A., Abid, A., Khan, D., Alfozan, A., & Zou, J. (2019). Gradio: Hassle-Free Sharing and Testing of ML Models in the Wild. ArXiv. /abs/1906.02569
- Plotly Technologies Inc. (2015). Collaborative data science. Plotly Technologies Inc. https://plot.ly
- Wu, Q., (2020). geemap: A Python package for interactive mapping with Google Earth Engine. The Journal of Open Source Software, 5(51), 2305. https://doi.org/10.21105/joss.02305
- Wu, Q., Lane, C. R., Li, X., Zhao, K., Zhou, Y., Clinton, N., DeVries, B., Golden, H. E., & Lang, M. W. (2019). Integrating LiDAR data and multi-temporal aerial imagery to map wetland inundation dynamics using Google Earth Engine. Remote Sensing of Environment, 228, 1-13. https://doi.org/10.1016/j.rse.2019.04.015 (pdf | source code)

AUTHOR INFO

Reach out on LinkedIn: https://linkedin.com/in/dkedar7

TRANSCRIPT

ABSTRACT

The exponential growth and complexity of Earth Science data call for innovative, adaptable, and user-friendly visualization tools. This study introduces 'Fast Dash', an open-source Python package that efficiently converts Python functions into interactive web applications, emphasizing geospatial mapping through integrating libraries such as Matplotlib, Geoplot, Plotly, and Leaflet.

The novelty of Fast Dash lies in its capacity to intuitively infer input and output layout components from Python function type hints. For instance, string type hints are transformed into text inputs, float hints manifest as sliders, and list hints become dropdowns. In a similar vein, the output layout is guided by the function outputs' type hints, with an option for visually appealing arrangements via mosaic ASCII art representations.

Grounded in the robust architecture of Plotly Dash and Flask, Fast Dash offers scalability, supports the visualization of large datasets via Datashader, and provides a streamlined process for deployment and sharing.

Fast Dash is underpinned by the understanding that well-annotated Python functions encapsulate all necessary information for designing interactive visualizations. In the domain of Earth observation, marked by a multitude of tools and a lack of standardized data visualization practices, Fast Dash presents a compelling solution. By significantly reducing the time between development and deployment, Fast Dash fosters a climate of accelerated innovation in Earth Science data visualization, further enhancing the interpretation and accessibility of complex data sets.