A Technical Report on Geospatial Data Analysis and Interactive Visualization: Mapping German Districts with Python, Streamlit, and Plotly

Section I: Foundations of Geospatial Data

The Data Model: A Foundational Choice for Spatial Analysis

The practice of geospatial data analysis requires a fundamental understanding of how geographic information is structured and stored. The digital representation of the Earth's surface is primarily managed through two distinct data models: vector and raster.[1] The selection between these two models is a critical initial step in any geospatial project, as it dictates the optimal tools and analytical approaches for the task at hand. This choice is determined by the nature of the geographic features being analyzed—specifically, whether they are discrete or continuous.

The vector data model is built upon a system of points, lines, and polygons to define discrete locations and features on the Earth's surface. Points are used to represent single, specific locations, such as the centroid of a city or the coordinates of a single building. Lines, composed of a series of connected points, are ideal for mapping linear features like roads, rivers, or utility networks. Polygons are closed shapes used to define areas, such as property boundaries, lakes, or, in the context of this report, administrative districts.[2] A notable characteristic of vector data is its resolution, which is flexible and independent of scale. This allows vector-based features to maintain their sharp, precise detail when a user zooms in or out, and this is achieved without a significant increase in file size.[2] For mapping clearly defined, political, or administrative boundaries, the vector model is the superior choice because it accurately and precisely delineates these discrete features. The widely accepted

adage, "Raster is faster but vector is corrector," aptly summarizes this distinction.[1] For projects that require precision in boundary definition, such as those involving governmental or political territories, the vector model's emphasis on correctness is paramount.

In contrast, the raster data model represents geographic space as a grid of uniformly sized cells, or pixels.[2] Each individual pixel in this grid stores a single value that represents a specific attribute of the area it covers, such as temperature, elevation, or color.[2] This pixel-based structure makes raster data exceptionally well-suited for representing continuous surfaces and phenomena that change gradually over space, such as a temperature gradient or the slope of a mountain. Common examples of raster data include Digital Elevation Models (DEM) and satellite imagery, which provide a pixel-by-pixel view of the terrain or land cover.[2] A key trade-off with the raster model is the direct relationship between resolution and file size; a higher resolution (smaller pixel size) captures more detail but results in a dramatically larger data file.[2] While highly effective for certain types of spatial analysis, such as terrain or flood modeling, the raster model is an inefficient and often inaccurate choice for mapping discrete, non-continuous features like administrative boundaries.

The core purpose of this report is to create a map of German districts (Kreise), which are defined by clear, political boundaries. The use of a vector data model is not merely a preference but a professional necessity. Representing these administrative units with a raster grid would introduce inherent inaccuracies at their borders, which is unacceptable for a map intended to reflect official jurisdictions. The fundamental decision to use vector data for this project dictates the entire technology stack and workflow that follows, ensuring that the final application is built on a foundation of data integrity and geographic accuracy.

The GeoJSON Format: The Lingua Franca of the Web for Geospatial Data

Having established the vector data model as the correct choice for this project, the next step is to select a specific, industry-standard file format for the geographic data. GeoJSON stands out as an optimal choice, particularly given the objective of creating an interactive web-based map. GeoJSON is an open-standard format built upon the JavaScript Object Notation (JSON), designed specifically for encoding simple geographic features.[3] Its web-native, text-based structure makes it exceptionally versatile and easy to integrate into modern data pipelines.

The internal structure of a GeoJSON file is a key to its power. The data is organized into a FeatureCollection, which is a list of individual Feature objects. Each Feature contains two primary components: a geometry field and a properties field. The geometry field holds the spatial information, defining the shape of the feature using coordinates (e.g., Point, LineString,

or Polygon types). The properties field is a key-value store that contains all the non-spatial attributes associated with that geographic feature, such as its name, population, or a unique identification code.[5]

The seamless compatibility of the GeoJSON format with the Python ecosystem is a key reason for its selection in this workflow. GeoJSON's structured, JSON-based nature aligns perfectly with Python's data manipulation libraries. For example, the geopandas library can read a GeoJSON file and automatically parse its contents into a GeoDataFrame, where the properties become the tabular columns and the geometry field is converted into a specialized column for spatial operations.[6] This transformation is crucial as it allows a user to treat the geographic data just like any other tabular data, enabling data-joining and filtering operations. Furthermore, the format's inherent compatibility with web technologies means that libraries like Plotly can directly consume a GeoJSON object—either as a file or in-memory dictionary—and render it into an interactive map without the need for complex intermediate conversions. This creates a highly efficient, end-to-end data pipeline with minimal friction, a significant advantage over other formats like Shapefiles, which often require a more complex parsing process.

The Python Geospatial Stack: GeoPandas, Plotly, and Streamlit

The construction of an interactive geospatial dashboard, as outlined in the user's request, is best achieved by leveraging a set of specialized Python libraries, each fulfilling a distinct but complementary role. This professional-grade approach, often referred to as a "best-of-breed" architecture, divides the project into three logical stages: data preparation, visualization, and application delivery. The selected libraries—GeoPandas, Plotly Express, and Streamlit—form a robust and efficient pipeline for this task.

At the base of this pipeline is **GeoPandas**, the central engine for geospatial data manipulation and analysis. GeoPandas extends the functionality of the ubiquitous pandas library, allowing it to perform spatial operations on geometric data types. The core data structures are the GeoDataFrame (a subclass of pandas.DataFrame) and the GeoSeries (a subclass of pandas.Series), which are designed to seamlessly integrate tabular and geometric data.[6] The library relies on a powerful underlying stack of open-source geospatial libraries, including

shapely for geometric operations, fiona for file reading and writing, and pyproj for managing Coordinate Reference Systems (CRS).[8] In this project, the primary role of GeoPandas is to ingest the GeoJSON data, inspect its properties, and, most critically, perform the data merge with the statistical data. While GeoPandas can create static maps for exploratory data analysis, its true power lies in its data-wrangling capabilities.

For the visualization component, **Plotly Express** is the ideal choice. As a high-level API for the Plotly graphing library, it provides a simple and concise way to generate a wide range of interactive charts and graphs, including the choropleth map required for this project.[9] A key distinction between Plotly and other plotting libraries like Matplotlib is its native support for interactivity. Plotly's charts are dynamic by default, allowing users to zoom, pan, and hover over data points to reveal additional information without any extra code.[9] This interactivity is a core requirement for a modern geospatial dashboard and is a significant improvement over static maps, which provide limited exploration capabilities.[10]

The final piece of the architecture is **Streamlit**, the application builder that transforms the Python script into a shareable web application. Streamlit is revered for its low-code philosophy, which allows data scientists and analysts to build and deploy interactive web apps using pure Python, without requiring any knowledge of frontend technologies like HTML, CSS, or JavaScript.[12] The library provides a suite of intuitive widgets, such as sliders and selectboxes, that can be used to control the data and visualizations in real time.[14] The real power of Streamlit lies in its ability to manage the interactive loop between user input and visualization updates, which will be central to the dashboard's functionality.

The synergy of these three libraries is not a coincidence. GeoPandas provides the foundational data manipulation, preparing a clean, joined GeoDataFrame. Plotly takes this prepared data and renders a beautiful, interactive map. Streamlit wraps the entire process, providing a user interface and a reactive framework to connect user input to the data and the map. The combination of these libraries creates a cohesive, highly effective solution that is greater than the sum of its parts.

Section II: Data Acquisition and Harmonization

Sourcing and Analyzing the Geographic Data (GeoJSON)

The first step in building the interactive map is to acquire the geographic data for the German districts. For this project, a GeoJSON file is the ideal source. A suitable dataset containing the geographic boundaries of German districts (Kreise) is available from open data portals, such as the georef-germany-kreis dataset on the opendatasoft platform.[16] Public repositories on platforms like GitHub also provide similar datasets.[17] GeoPandas simplifies the data acquisition process by allowing a direct read from a web URL, eliminating the need for a

manual download. The

geopandas.read_file() function can handle a URL as an input, which streamlines the data loading process.[19]

Upon ingesting the data, a critical step is to analyze the GeoDataFrame to understand its structure and content. A GeoDataFrame, a specialized pandas.DataFrame, contains a geometry column that holds the polygon shapes for each district and additional columns that correspond to the properties field in the GeoJSON file.[6] A careful inspection of these properties is essential to identify a reliable, unique identifier for each district. The

georef-germany-kreis dataset contains several attributes, including Kreis name (e.g., "Kreis Coesfeld") and a more formal code, the Amtlicher Regionalschlüssel (ARS).[16]

The professional approach to this project requires the use of the Amtlicher Regionalschlüssel (ARS) as the canonical join key. While a name-based join on a field like Kreis name might seem intuitive, it is prone to data integrity issues. The term "district" in Germany can be a Kreis, Landkreis (rural district), or Kreisfreie Stadt (urban district).[16] A simple string match on the name can lead to ambiguity and misalignments, for example, between the city of Munich (

Kreisfreie Stadt München) and the surrounding rural district (Landkreis München). The ARS is a standardized numeric identifier that provides a stable, unambiguous reference for each district, making it the only reliable choice for a data merge.[18] By using this unique key, the analyst ensures that the geographic boundaries are correctly and consistently associated with the statistical data, thereby guaranteeing the accuracy of the final visualization.

Identifying and Preparing the Statistical Data (Population)

The next step is to acquire a dataset containing statistical information that can be mapped to the geographic boundaries. Population data is a common choice for this type of visualization. Research into open data sources for German district-level population statistics reveals a common challenge: a clean, universally available CSV or Excel file is not always a direct download option from official sources.[23] However, public information from German government agencies, such as the Federal Statistical Office (

Statistisches Bundesamt), often contains the required data embedded within other documents. For instance, police crime statistics reports (Polizeiliche Kriminalstatistik) published by the Federal Criminal Police Office (Bundeskriminalamt or BKA) include detailed population data by district (Kreis and Kreisfreie Stadt), critically, along with their Amtlicher

Regionalschlüssel (ARS).[22]

This is a scenario frequently encountered in real-world data projects, where data is not provided in a perfectly clean, machine-readable format. Instead of halting the project, a data professional would adapt their approach. The solution involves extracting the necessary information, either by manually creating a small, representative sample or by using more advanced data extraction techniques to programmatically parse the document. For the purpose of this report, a sample pandas.DataFrame can be created to represent the population data, with columns for the ARS key, the district name, and the total population. This pragmatic approach allows the project to proceed and demonstrates a core skill in data analysis: solving real-world data acquisition challenges.

The Critical Merge: Joining Geographic and Statistical Data

With both the geographic and statistical datasets prepared, the crucial step of joining them can be performed. This process unites the spatial information (the polygons) with the non-spatial attributes (the population figures) into a single, cohesive GeoDataFrame, which can then be visualized. Since GeoPandas is built as an extension of pandas, a GeoDataFrame inherits all the functionality of a standard DataFrame.[6] This means that the familiar

pandas.merge() function can be used to perform the join.

The join operation must be executed on a common, unique identifier present in both datasets. As previously established, the Amtlicher Regionalschlüssel (ARS) is the correct key for this purpose, guaranteeing a one-to-one match and preventing data misalignment. A left join is the most appropriate merge type for this task, as it ensures that all geographic features (German districts) are retained in the final GeoDataFrame, even if some of them happen to be missing a corresponding record in the statistical data.[26] The final output of this step is a single GeoDataFrame that contains all the geographic polygons along with their associated population data, perfectly aligned and ready for visualization.

To illustrate the technical foundation of this join, the following table summarizes the data schema for the key fields in each dataset:

Dataset	Key Field Name	Data Type	Example Value	Source Reference
GeoJSON	ARS	Integer	01001	[18]

(GeoDataFrame)				
Population (DataFrame)	ARS	Integer	01001	[22]

This table serves as a blueprint for the join operation, making the crucial relationship between the two datasets explicit. The successful execution of this data-merging step is the cornerstone of the entire project, as a map with misaligned data is not only inaccurate but also misleading. This process underpins the integrity of the final visualization.

Section III: Constructing the Interactive Map

Choropleth Maps: The Visual Language of Geospatial Data

The central objective of this project is to create a compelling and informative visualization that communicates the distribution of population across Germany's districts. The most effective tool for this task is a choropleth map. A choropleth map is a thematic map in which divided geographic regions, such as countries, states, or in this case, districts, are shaded or colored in proportion to the value of a single data variable.[9] In this context, the variable is population.

The power of a choropleth map extends beyond its aesthetic appeal; it is a powerful analytical tool. By using a sequential color scale—typically ranging from a light color for low values to a dark color for high values—it allows a user to instantly identify geographical patterns and trends.[10] For instance, areas with high population density will appear as darker-colored polygons, while sparsely populated areas will be lighter. This visual representation enables the rapid identification of population clusters, regional disparities, or demographic hotspots that would be difficult to discern from a simple table of numbers.[10] Effective choropleth map design involves a careful choice of color scale and clear labeling to ensure that the visualization is not only visually appealing but also a transparent and accurate representation of the data.[9]

Building the Map with Plotly Express

Plotly Express provides a high-level, user-friendly interface for creating complex visualizations with minimal code. The library's px.choropleth() function is specifically designed to handle the creation of interactive choropleth maps with ease. The function automates the complex process of joining geographic boundaries with tabular data and rendering them into a web-native, interactive map.[9]

The function requires several key parameters to function correctly:

- data_frame: This is the prepared GeoDataFrame that contains both the geographic boundaries and the statistical data.
- geojson: The GeoJSON object itself. Plotly can accept a GeoJSON object in a dictionary format, which means the GeoDataFrame can be converted to this format in memory, creating an efficient, in-memory data pipeline that avoids writing temporary files to disk.
- locations: The name of the column in the data_frame that holds the unique identifier, which in this case is the ARS.[9]
- color: The name of the column containing the numerical values to be visualized, which is the population data. This parameter dictates the shading of each polygon.[9]
- hover_data: A list of column names whose values should be displayed when a user hovers their cursor over a specific district on the map. This is a crucial element of the map's interactivity, providing the user with on-demand access to additional information, such as the district's name and its exact population figure.[9]

The px.choropleth() function also handles the base map configuration and projection automatically. The generated map is interactive by default, allowing users to zoom and pan to explore the data at different levels of detail, a functionality that is essential for a compelling geospatial dashboard.[9]

The following table summarizes the key parameters for using px.choropleth() in the context of this project:

Parameter	Description	Required Input	Source Reference
geojson	The GeoJSON object containing the geographic shapes.	GeoJSON Dictionary or URL	[5]
locations	The column with the unique join key.	Column Name (e.g., 'ARS')	[9]

color	The column with the data for the color scale.	Column Name (e.g., 'Population')	[9]
hover_data	Columns to display in the hover tooltip.	List of Column Names	[9]

By following this approach, a professional can transform the prepared data into a visually rich and interactive map with a single, elegant line of code.

Section IV: Developing the Streamlit Dashboard

From Script to Interactive Application

Streamlit is the final piece of the technology stack, serving as the bridge between a data script and a fully functional, user-facing web application. Its design philosophy is centered on simplicity: the entire application is defined by a single Python script that is executed from top to bottom on every user interaction, such as a button click or a filter selection.[12] This straightforward model makes it exceptionally easy for data scientists to build interactive prototypes and dashboards without needing to delve into the complexities of traditional web development frameworks.

However, a top-to-bottom re-run on every interaction presents a performance challenge, especially for applications that require loading large datasets. To address this, Streamlit provides a powerful caching mechanism via the @st.cache_data decorator.[14] When this decorator is placed before a function, Streamlit will execute the function only once and store its output in a local cache. On subsequent re-runs of the script, if the function's inputs and code have not changed, Streamlit retrieves the data from the cache instead of re-executing the function. This capability is critical for optimizing the dashboard's performance, as it ensures that the time-consuming data acquisition and preparation steps are performed only once when the application starts or when the data source changes.[14] This combination of a simple, reactive loop with intelligent caching is what makes Streamlit a highly effective tool for rapid application development in the data science domain.

Designing the Dashboard Layout and Logic

A well-designed dashboard provides a clean and intuitive user experience by separating the controls from the main content. Streamlit offers simple layout components to achieve this. The st.sidebar function creates a dedicated panel on the left side of the screen, which is the perfect place to house interactive widgets and filters. The main area of the application can then be used to display the core visualization, which in this case is the interactive choropleth map.

The central piece of functionality for the dashboard is the interactive filter. A st.selectbox widget is an ideal component for this purpose, as it presents a dropdown menu of options to the user.[15] The options for the selectbox can be dynamically populated from a list of the district names (

Kreis name or GEN) in the GeoDataFrame. When a user selects a district from the dropdown, Streamlit's reactive loop is triggered. The selected name is passed back to the script, which uses it to filter the GeoDataFrame.[15] The filtered data is then passed to the

px.choropleth function, which redraws the map with the new selection highlighted, thereby completing the interactive feedback loop. This seamless connection between user input, data manipulation, and visualization is the core logic that powers the dashboard.

Final Code Assembly and Deployment

The culmination of this workflow is the final, complete Python script. This script orchestrates all the components discussed in this report, from data acquisition and preparation to visualization and interactive control.

The script begins with the necessary library imports (streamlit, pandas, geopandas, and plotly.express). The data loading function, responsible for acquiring and merging the GeoJSON and population data, is defined and decorated with @st.cache_data to ensure optimal performance. The main body of the script sets up the Streamlit application, defining the layout with a sidebar for user controls. The st.selectbox widget is added to the sidebar, allowing the user to select a district from the GeoDataFrame's Kreis name column. The script then uses this selection to filter the GeoDataFrame. Finally, the px.choropleth function is called with the filtered data and the appropriate parameters, and the resulting interactive map is rendered in the main content area using st.plotly_chart.

Once the script is complete, running the application is a straightforward process. A user can

execute the script from the command line using the command streamlit run <script_name.py>. This command launches a local web server and opens the interactive dashboard in a web browser, making the application immediately accessible for exploration. The final output is not just a static map but a dynamic, shareable data product that can be deployed to the web, ready for a wider audience to use and interact with.

Conclusion

The development of an interactive geospatial dashboard for German districts serves as a powerful case study for a professional data analytics workflow. The project begins with a foundational, expert-level decision on the data model—choosing vector over raster for its precision in mapping administrative boundaries. The subsequent selection of the GeoJSON format aligns perfectly with the chosen Python stack, creating a seamless data pipeline.

The report highlights how a professional would handle the realities of imperfect data by sourcing information from a pragmatic source and using a canonical identifier, the Amtlicher Regionalschlüssel (ARS), to ensure data integrity during the critical merge. This is a fundamental step that guarantees the accuracy of the final visualization.

The final product, an interactive choropleth map, is built using Plotly Express, a library renowned for its dynamic, web-native output. This map is then integrated into a full-fledged web application using Streamlit. The application's core functionality, a reactive loop that updates the map based on user selections from a selectbox widget, is made performant and responsive through Streamlit's caching mechanism.

This project is a clear demonstration of the power and synergy of the modern Python ecosystem for data science. It shows how the specialized capabilities of GeoPandas (data manipulation), Plotly Express (visualization), and Streamlit (application building) can be combined to solve a complex, real-world problem and deliver a comprehensive, shareable, and fully functional data product.

Referenzen

1. www.esri.com, Zugriff am September 11, 2025, https://www.esri.com/content/dam/esrisites/en-us/media/pdf/teach-with-gis/raster-faster.pdf
2. Understanding raster and vector geospatial data - Birdi Blog, Zugriff am September 11, 2025, https://www.birdi.io/blog-post/understanding-raster-and-vector-geospatial-data
3. en.wikipedia.org, Zugriff am September 11, 2025, https://en.wikipedia.org/wiki/GeoJSON

4. GeoJSON—ArcGIS Online Help | Documentation, Zugriff am September 11, 2025, https://doc.arcgis.com/en/arcgis-online/reference/geojson.htm
5. Choropleth maps in Python - Plotly, Zugriff am September 11, 2025, https://plotly.com/python/choropleth-maps/
6. Introduction to GeoPandas, Zugriff am September 11, 2025, https://geopandas.org/en/stable/getting_started/introduction.html
7. Geopandas: an introduction - Automating GIS Processes, Zugriff am September 11, 2025, https://autogis-site.readthedocs.io/en/latest/lessons/lesson-2/geopandas-an-introduction.html
8. Working with Geospatial Data in Python - GeeksforGeeks, Zugriff am September 11, 2025, https://www.geeksforgeeks.org/python/working-with-geospatial-data-in-python/
9. How to create a choropleth map with Plotly Express in Python - Educative.io, Zugriff am September 11, 2025, https://www.educative.io/answers/how-to-create-a-choropleth-map-with-plotly-express-in-python
10. Choropleth Maps using Plotly in Python - GeeksforGeeks, Zugriff am September 11, 2025, https://www.geeksforgeeks.org/python/choropleth-maps-using-plotly-in-python/
11. Choropleth map - Python Graph Gallery, Zugriff am September 11, 2025, https://python-graph-gallery.com/choropleth-map/
12. Streamlit • A faster way to build and share data apps, Zugriff am September 11, 2025, https://streamlit.io/
13. Building a dashboard in Python using Streamlit, Zugriff am September 11, 2025, https://blog.streamlit.io/crafting-a-dashboard-app-in-python-using-streamlit/
14. Create an app - Streamlit Docs, Zugriff am September 11, 2025, https://docs.streamlit.io/get-started/tutorials/create-an-app
15. [Explained] Streamlit Selectbox: Usage, Parameters, and Examples - Kanaries Docs, Zugriff am September 11, 2025, https://docs.kanaries.net/topics/Streamlit/streamlit-selectbox
16. Kreise - Germany — Data hub, Zugriff am September 11, 2025, https://data.opendatasoft.com/explore/dataset/georef-germany-kreis@public/export/
17. SBejga/germany-administrative-geojson - GitHub, Zugriff am September 11, 2025, https://github.com/SBejga/germany-administrative-geojson
18. Praesklepios/geoGermany: geojson files of the administrative regions of Germany - GitHub, Zugriff am September 11, 2025, https://github.com/Praesklepios/geoGermany
19. Reading and Writing Files — GeoPandas 0.7.0 documentation, Zugriff am September 11, 2025, https://geopandas.org/en/v0.7.0/io.html
20. Reading and writing files — GeoPandas 1.0.1+0.g747d66e.dirty documentation, Zugriff am September 11, 2025, https://geopandas.org/en/v1.0.1/io.html
21. Districts of Germany - Wikipedia, Zugriff am September 11, 2025, https://en.wikipedia.org/wiki/Districts_of_Germany

22. Bevölkerungsdaten insgesamt Bereich: Kreise und kreisfreie Städte Quelle: Statistisches Bundesamt Berichtsjahr: 2020 V1.0 erst - BKA, Zugriff am September 11, 2025, https://www.bka.de/SharedDocs/Downloads/DE/Publikationen/PolizeilicheKriminalstatistik/2020/Sonst_Tabellen/04-KR-BV-insg_pdf.pdf?__blob=publicationFile&v=5
23. Maps and geodata - German Federal Statistical Office - Statistisches Bundesamt, Zugriff am September 11, 2025, https://www.destatis.de/EN/Service/OpenData/maps-geodata.html
24. Bevölkerungsdaten insgesamt Bereich: Kreise und kreisfreie Städte Quelle: Statistisches Bundesamt Berichtsjahr: 2023 V1.0 erst - BKA, Zugriff am September 11, 2025, https://www.bka.de/SharedDocs/Downloads/DE/Publikationen/PolizeilicheKriminalstatistik/2023/Sonst_Tabellen/04-KR-BV-insg_pdf.pdf?__blob=publicationFile&v=4
25. GeoPandas Tutorial: An Introduction to Geospatial Analysis - DataCamp, Zugriff am September 11, 2025, https://www.datacamp.com/tutorial/geopandas-tutorial-geospatial-analysis
26. Spatial Joins — GeoPandas 1.1.1+0.ge9b58ce.dirty documentation, Zugriff am September 11, 2025, https://geopandas.org/en/stable/gallery/spatial_joins.html
27. plotly.graph_objects.Choropleth — 6.3.0 documentation, Zugriff am September 11, 2025, https://plotly.github.io/plotly.py-docs/generated/plotly.graph_objects.Choropleth.html
28. How to make a culture map - Streamlit Blog, Zugriff am September 11, 2025, https://blog.streamlit.io/how-to-make-a-culture-map/
29. How to filter data frame via a column *and* have a check box column to filter rows, Zugriff am September 11, 2025, https://discuss.streamlit.io/t/how-to-filter-data-frame-via-a-column-and-have-a-check-box-column-to-filter-rows/48089