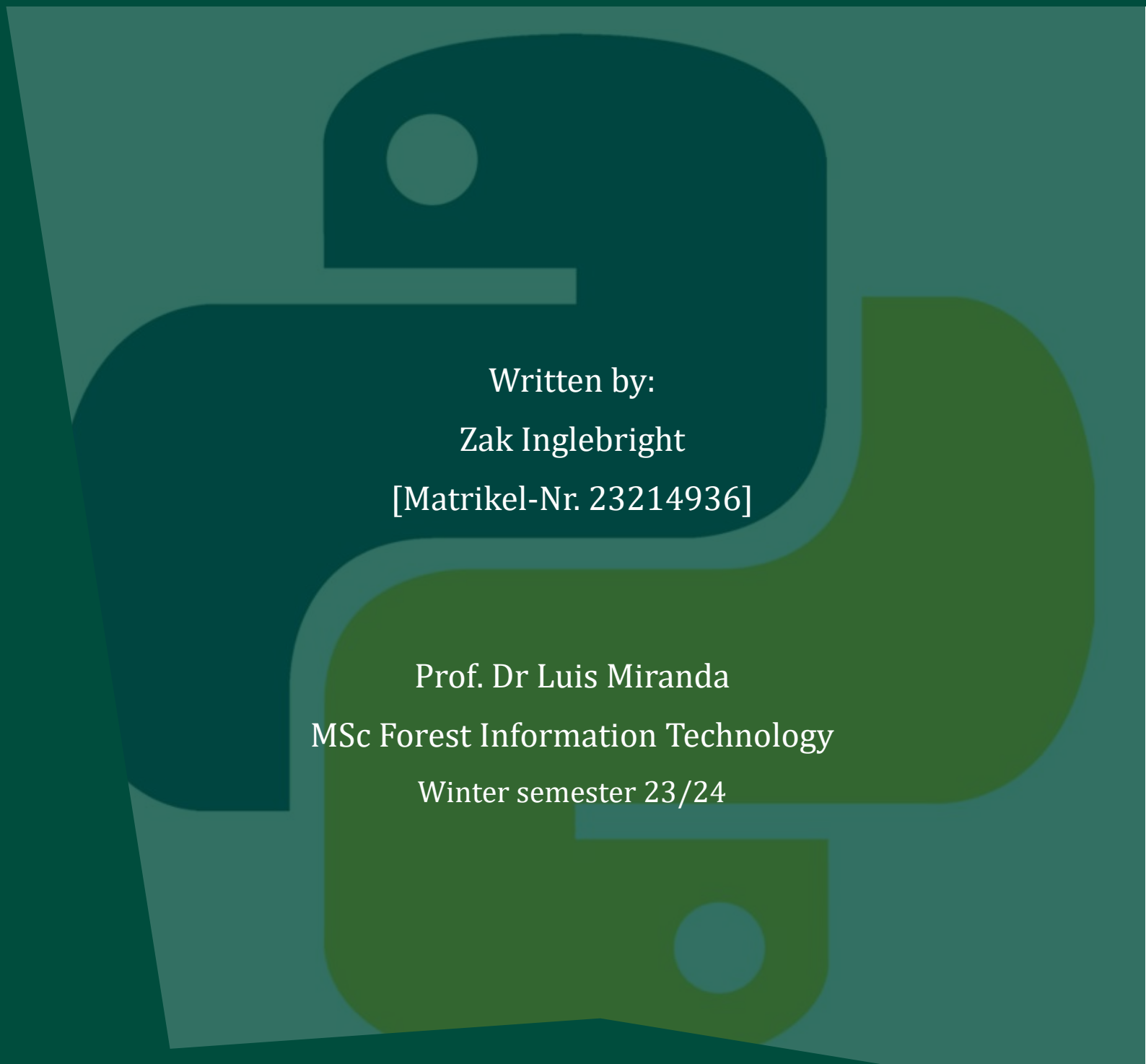
Hochschule
für nachhaltige Entwicklung
Eberswalde

Applied Programming in Forestry

Project: Python for Automation of LUCAS Dataset Classification for Land Use Analysis



Written by:

Zak Inglebright

[Matrikel-Nr. 23214936]

Prof. Dr Luis Miranda

MSc Forest Information Technology

Winter semester 23/24

Berlin, 12th February 2024

List of Contents

1 Introduction

The project leverages Python 3 to streamline the classification of the Land Use/Cover Area frame Survey (LUCAS) dataset, analysing land use across the European Union (EU). Inspired by "*Agroforestry as a Sustainable Land Use Option to Reduce Wildfire Risks in European Mediterranean Areas*" by Damianidis et al. (2021), it categorises land into distinct areas: forest, shrubland, grassland, arable, and livestock agroforestry. Initially conducted using Geographic Information System (GIS) software, the process is now enhanced through Python, incorporating a DASH-based dashboard for interactive visualisation. This tool allows for the easy input of LUCAS datasets from 2009 onwards, processing and displaying outcomes on the dashboard, hence addressing the critical need for efficient data analysis and visualisation in assessing land use change.

2 Material and Software Tools

This project rests on the analysis of LUCAS survey datasets sourced from the EUROSTAT database. These datasets, formatted as comma-separated values (CSV) files, contain various columns detailing land cover, geographical coordinates, and additional survey-specific information. To process the raw data, Python 3 programming language was implemented within the integrated development environment (IDE) of Visual Studio Code. This setup leveraged the functionalities of Python 3, together with an array of libraries such as Pandas for data manipulation of heterogeneous data types (McKinney, 2012), Plotly Express for data visualisation, and DASH for interactive dashboard applications, among other python packages. The end result in the functionality of the code is an interactive dashboard application, capable of rendering time-series analyses with dynamic visualisation capabilities.

2.1 Raw Data

2.1.1 LUCAS Survey Point Data

The Land Use/Cover Area frame Survey (LUCAS) is a dataset for understanding land use and land cover across the EU. Conducted every three or four years since 2006 by Eurostat, LUCAS provides comprehensive data on various land categories, including agricultural, forest, and urban areas. The survey involves a systematic sampling of points across the EU (Eurostat, 2024), offering insights into land use changes and patterns.

2.1.2 Country Code

A CSV file, containing a list of countries with their (ISO 3166-1) Alpha-2 code, Alpha-3 code, UN M49, average latitude and longitude coordinates (GitHub, 2020) was employed. This file served

to match the corresponding Alpha-2 codes with the LUCAS dataset to extract the full name of the countries. Thereby, improving the readability of the dataset in the dashboard application.

2.2 DASH for Interactive Dashboard Application

Dash, developed by Plotly, is an open-source Python framework for developing data visualisation and analytical web applications. It integrates Flask, React.js, and Plotly.js, enabling data scientists to build interactive user interfaces without a deep knowledge in web development (Plotly, 2024). The project utilises Dash to forge a user-friendly dashboard, offering interactive data engagement, such as filtering and selection features. It facilitates data exportation, accommodates new LUCAS dataset additions, and ensures adaptable data comparison.

3 Methods

The code automates LUCAS data processing from initial input to classification and visualisation of the land use and landcover (LULC) categories. It includes custom functions for data standardisation, classification based on LULC criteria, and the generation of an interactive dashboard to visualise the outputted datasets.

3.1 Classification & Transformation

The classification phase of the code utilises the `processLucasData` Function seen in **Figure 1**, which is comprised of several functions describe below.

3.1.1 Standardisation and Merging

The `standardiseColumn` function is crucial in preprocessing the LUCAS dataset for analysis. This function systematically renames columns, such as 'SURVEY_GRAZING' to 'LAND_MNGT', and removes unnecessary prefixes, creating a consistent naming convention across various datasets. This standardisation process extends to country data as well, where country codes are updated, and long-form country names are simplified, for instance, renaming *'United Kingdom of Great Britain and Northern Ireland'* to *'Great Britain'*. The function then merges the country information with the LUCAS data, providing a well-structured dataset that enhances its comprehensiveness for further analysis.

3.1.2 Classification

The `filterClasses` function, illustrated in **Figure 2**, categorises survey points into predefined LULC classes, using classification criteria from den Herder et al. (2017). This categorisation, encompassing classes like livestock, arable, forest, shrubland, and grassland, is crucial for analysing LULC, especially agroforestry trends across time and locations. Boolean indexing in this

function refines the dataset by applying conditions on primary and secondary land covers and management practices, facilitating quick and precise selection of data points for specific LULC classes.

An integral part of the process is the `generateClassIDs` function (**Figure 3**), which employs the classification framework by den Herder et al. (2017) to identify agroforestry systems. It uses string concatenations and the `range` function within a `for` loop to code each LULC class in a single line of code. For example, the class ID for livestock_LC1 is created by concatenating 'B', 'C', and 'D10', 'E10' with a sequence of numbers, generating unique identifiers for various livestock categories in the LUCAS dataset. This methodological approach enhances the dataset's utility for detailed LULC analysis, allowing for effective comparison and study of land use patterns.

3.1.3 Summarising the Data

The `createSummarisedTable` function is comprised of several functions used to create a summarised table containing the total number of LUCAS points in each class per country. Such functions include the `createBinaryClassColumn` function used to create columns in binary format to indicate the presence (1) or absence (0) of the specified classIDs. The `createFrequencyTable` function to summarise the total count of each class per country, and lastly the `createPercentageTable` function to show the distribution of each class per country as a percentage.

3.2 Visualisation by DASH

The callback graph depicted in **Figure 4** illustrates the dependencies and interactions between different components of the Dash application through callbacks. This structured data flow causes the users interaction to trigger data processing and subsequent updates to the visual components, enabling real-time data visualisation and interaction. **Figure 4** will serve to outline the methodology and summarise the key functionalities within the five hundred lines of code that makes up the dashboard application. It highlights how the different components are connected and interact with each other through callbacks. However, only the core processes that are essential to the operation of the dashboard will be discussed below.

3.2.1 Processing and Storing Uploaded Content

The `parse_contents` function reads, decodes, and extracts the year from filenames for raw data processing via the `processLucasData` function, as shown in **Figure 1**. This function processes uploaded data, producing lucas_df for web mapping and lucasTable for manipulation and display. A safeguard prevents duplicate file inputs by checking the 'countryYear' attribute in lucas_df

against the global DF1 before merging. The outcomes, lucas_df and lucasTable, are stored in **`storage-gdf`** and **`storage-table`** respectively, depicted in **Figure 2**.

3.2.2 Updating Graphics

The updated geospatial data in **`storage-gdf`** is used to update the dropdown options and map visualisation. Through the **`my-dropdown`** component, users can select data subsets, which **`my-map`** renders the subsetted dataset on an interactive map, providing a spatial context to the data (**Figure 5**). The **`stackedGraph`** component is updated via a separate callback, employing the **`storage-table`** data to generate a stacked bar graph that offers insight into the distribution of LULC classifications over a time distribution (**Figure 6**), as captured in the interactions of **Figure 4**.

4 Results & Discussion

The outcome of the project has showcased the analytical and computational efficiencies of Python 3 in the classification and analysis of LUCAS datasets. The previous methodology, outlined by Damianidis et al. (2020), was characterised by a manual and time-consuming approach via a GIS software. A prominent example of this is the **`parse_contents`** function, which, through encapsulation via object-oriented programming, simplifies data extraction (Zambelli et al., 2013), allowing for rapid parsing and year extraction from file names—a task more complex and slower in GIS. The **`generateClassIDs`** function (**Figure 3**) is critical for creating the distinct Class IDs for discerning the five LULC classes employed by Damianidis et al. (2020). Such a task, when executed in a GIS software, are notoriously time-intensive (Wang et al., 2019), due to the multitude of primary and secondary landcover IDs required. Conversely, with the functional ability of Python 3, the generation of Class IDs for each LULC class is executed through a single line of code. By leveraging concatenations and the **`range`** function, Python 3 adeptly assembles specific combinations into a single list via **`for`** loops with remarkable speed. This not only highlights Python 3 ability for efficient data manipulation but also represents a significant advance in the processing of environmental datasets.

The DASH dashboard's interactive capabilities enable users to engage with the data dynamically over a spatial and temporal domain. The application permits rapid selection and visual comparison of LULC classes across various EU countries, facilitated by the `my-dropdown` and `my-map` components. These tools alongside the functionalities of Plotly Express enables users to focus directly on the agroforestry classes, among others, allowing for an immediate comparative analysis that highlights spatial distribution of specific classes between various counties, as shown in **Figure 5**. Moreover, the stacked bar graph provides a temporal comparison

to the user within individual countries. For instance, examining Italy's data in **Figure 6** from 2009, 2012, and 2022 reveals insightful shifts in land use dynamics. The increasing percentage of forest cover from 31.3% in 2009 to 57.3% in 2022 suggests a trend towards forestation or natural succession, whereas arable land shows a minor decrease. The added ability to export tables corresponding to these visualisations, allows the user the opportunity to delve deeper into the data. They can apply additional analytical methods, conduct more advanced analyses, or integrate this data with other datasets for a more comprehensive study (Fischer & Hutchinson, 2018).This ability underscores the dashboard's role not only as a tool for immediate visual analysis but also as a foundation for extended research.

The integration of Boolean indexing in the `filterClasses` function (**Figure 2**) overcame the initial challenge of processing large datasets, a task which previously employed the `.iterrow()` method for filtering ClassIDs. The `.iterrow()` approach, while straightforward, is markedly slower and less efficient for large datasets, primarily due to its row-by-row processing nature (Ball & Rague, 2022). The use of Boolean indexing has transformed this process, allowing for the bulk filtering of data based on predefined ClassIDs. This method applies logical conditions to the dataset, significantly accelerating the classification process by eliminating the need for row-by-row examination. Such enhancement in data processing capability emphasises Python 3's ability to handle large number of environmental datasets with speed (Rajagopalan, 2021). This development not only speed up the classification of LULC classes but also enhances the users experience in the dashboard application when uploading large datasets, by accelerating the processing capacity. While the project significantly advances the processing and analysis of LUCAS datasets, it acknowledges certain limitations. These issues can be surmised by (Zhang, 2023) who stated, "*environmental data providing huge amounts of information, but it is complex to process due to the size, variety, and dynamic nature of the data*". This is because the format of LUCAS datasets varied considerably across the years requiring thorough modifications to the `standardiseColumn` function. This implies that surveys conducted post-2022 might not be compatible with the dashboard application due to variances in the dataset's naming conventions. An example of this is the 2022 datasets, where identifiers such as 'Survey_Grazing' have been introduced, compared to the attribute 'Grazing' used in the 2018 datasets.

5 Conclusion

The code has overcome a significant problem in geoinformatics, regarding the classification of environmental data, by providing an automated solution. For scientists and the public alike, it translates complex LUCAS datasets into accessible, interactive maps and graphs, directly

contributing to a better understanding of land use. This aids in environmental conservation, policy-making, and educational outreach by making data-driven insights readily available. Furthermore, the dashboard aligns with Damianidis et al. (2020) by affording a methodological framework that not only identifies but also classifies LUCAS data with precision. While Damianidis had his colleagues had to grapple with the lack of automation and the associated laboriousness in their GIS-based analysis, this project overcomes those constraints, presenting an innovative, user-friendly interface that integrates data analysis and visualisation, thereby enriching the study of agroforestry classes across the EU.

6 References

Ball, R., & Rague, B. (2022). Data Wrangling. *The Beginner's Guide to Data Science*, 31–68. https://doi.org/10.1007/978-3-031-07865-1_3

Damianidis, C., Santiago-Freijanes, J. J., den Herder, M., Burgess, P., Mosquera-Losada, M. R., Graves, A., Papadopoulos, A., Pisanelli, A., Camilli, F., Rois-Díaz, M., Kay, S., Palma, J. H. N., & Pantera, A. (2021). Agroforestry as a sustainable land use option to reduce wildfires risk in European Mediterranean areas. *Agroforestry Systems*, *95*(5), 919–929. https://doi.org/10.1007/S10457-020-00482-W/FIGURES/1

den Herder, M., Moreno, G., Mosquera-Losada, R. M., Palma, J. H. N., Sidiropoulou, A., Santiago Freijanes, J. J., Crous-Duran, J., Paulo, J. A., Tomé, M., Pantera, A., Papanastasis, V. P., Mantzanas, K., Pachana, P., Papadopoulos, A., Plieninger, T., & Burgess, P. J. (2017). Current extent and stratification of agroforestry in the European Union. *Agriculture, Ecosystems & Environment*, *241*, 121–132. https://doi.org/10.1016/J.AGEE.2017.03.005

Eurostat. (2024). *Overview - Land cover and use*. https://ec.europa.eu/eurostat/web/lucas/overview

Fischer, A. J., & Hutchinson, S. (2018). *Developing and evaluating a geographic information dashboard to improve spatial task performance*. https://krex.k-state.edu/handle/2097/38841

GitHub. (2020). *Countries with their (ISO 3166-1) Alpha-2 code, Alpha-3 code, UN M49, average latitude and longitude coordinates*. https://gist.github.com/tadast/8827699

McKinney, W. (2012). *pandas: powerful Python data analysis toolkit*. https://pandas.pydata.org/pandas-docs/version/0.7.3/pandas.pdf

Plotly. (2024). *Plotly Python Graphing Library*. https://plotly.com/python/

Rajagopalan, G. (2021). Prepping Your Data with Pandas. In *A Python Data Analyst's Toolkit* (pp. 147–241). Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-6399-0_6

Wang, S., Zhong, Y., & Wang, E. (2019). An integrated GIS platform architecture for spatiotemporal big data. *Future Generation Computer Systems*, *94*, 160–172. https://doi.org/10.1016/J.FUTURE.2018.10.034

Zambelli, P., Gebbert, S., & Ciolli, M. (2013). Pygrass: An Object Oriented Python Application Programming Interface (API) for Geographic Resources Analysis Support System (GRASS) Geographic Information System (GIS). *ISPRS International Journal of Geo-Information 2013, Vol. 2, Pages 201-219*, *2*(1), 201–219. https://doi.org/10.3390/IJGI2010201

Zhang, Z. (2023). Environmental Data Analysis. In *Environmental Data Analysis*. De Gruyter. https://doi.org/10.1515/9783111012681/HTML

7 Appendix

```python
def processLucasData(rawData, countryID, year):
    """
    Main function to process LUCAS data: standardises columns, filters by land use and cover classes,
    and generates summarised tables.
    """
    modifiedDF = rawData.copy()
    correctDF = standardiseColumn(modifiedDF, countryID)
    lucasData = filterLandUseCoverClasses(correctDF, year)
    lucasTable = createSummarisedTable(lucasData, year)

    return lucasData, lucasTable
```

Figure 1: Python function **`processLucasData`** defined to automate the processing of LUCAS datasets. The function performs three key operations: it creates a copy of the raw data, standardises column names to ensure consistency, filters data by LULC classes using the **`filterLandUseCoverClasses`** function, and generates summarised tables for further analysis. The outcome of the function is two structured dataframes, lucasData and lucasTable, which hold the processed information for subsequent use.

```python
def filterClasses(rawDF, classIDs, className, landMngt=2.0, option=0):
    """
    Generic filtering function to classify and return LULC classes into DataFrame, based on booleon indexing
    """
    columnName = ['LAT', 'LONG', 'LC1', 'LC2', 'LAND_MNGT']

    # Filtera Agroforestry classes
    if option == 0:
        if landMngt == 2.0:  # Arable agroforestry
            condition = (rawDF['LC1'].isin(classIDs[0])) & (rawDF['LC2'].isin(classIDs[1])) & (rawDF['LAND_MNGT'] == landMngt)
        else:            #  livestock agroforestry
            condition = (rawDF['LC1'].isin(classIDs)) & (rawDF['LAND_MNGT'] == landMngt)
    # Filters Non-Agroforestry classes
    else:
        condition = (rawDF['LC1'].isin(classIDs[0])) & (~rawDF['LC2'].isin(classIDs[1])) & (rawDF['LAND_MNGT'] == landMngt)

    filteredDF = rawDF.loc[condition, columnName].copy()
    filteredDF['CLASS'] = className

    return filteredDF
```

Figure 2: Python function **`filterClasses`**, demonstrating the use of Boolean indexing for classifying LULC data within a DataFrame. The function segregates data into LULC classes based on a combination of conditions applied to land cover columns 'LC1' and 'LC2', and the 'LAND_MNGT' attribute.

```python
def generateClassIDs():
    """
    Generates and returns lists of Class IDs for various land cover categories:
    livestock, arable, forest, shrubland, and grassland.
    """
    # Define Class IDs for different land use categories
    livestockClassIDs = ['B' + str(b) for b in range(71, 85)] + ['C' + str(c) for c in range(10, 34)] + ['D10', 'E10']
    arable_LC1 = livestockClassIDs[:-2] + ['D10']  # Shares some IDs with livestock, excludes the last two
    arable_LC2 = ['B' + str(i) for i in range(11, 55)]
    forest_LC1 = ['C10', 'C21', 'C22', 'C23', 'C31', 'C32', 'C33']
    shrubland_LC1 = ['D10', 'D20']
    grassland_LC1 = ['E10', 'E20', 'E30']

    arableClassIDs = [arable_LC1, arable_LC2]  # Combine both lists for arable agroforestry
    forestClassIDs = [forest_LC1,arable_LC2]
    shrublandClassIDs = [shrubland_LC1, arable_LC2]
    grasslandClassIDs = [grassland_LC1, arable_LC2]


    return livestockClassIDs, arableClassIDs, forestClassIDs, shrublandClassIDs, grasslandClassIDs
```

Figure 3: Python function **`generateClassIDs`**, delineating the creation of Class ID lists for the five LULC categories including: Livestock, Arable, Forest, Shrubland and Grassland classes.

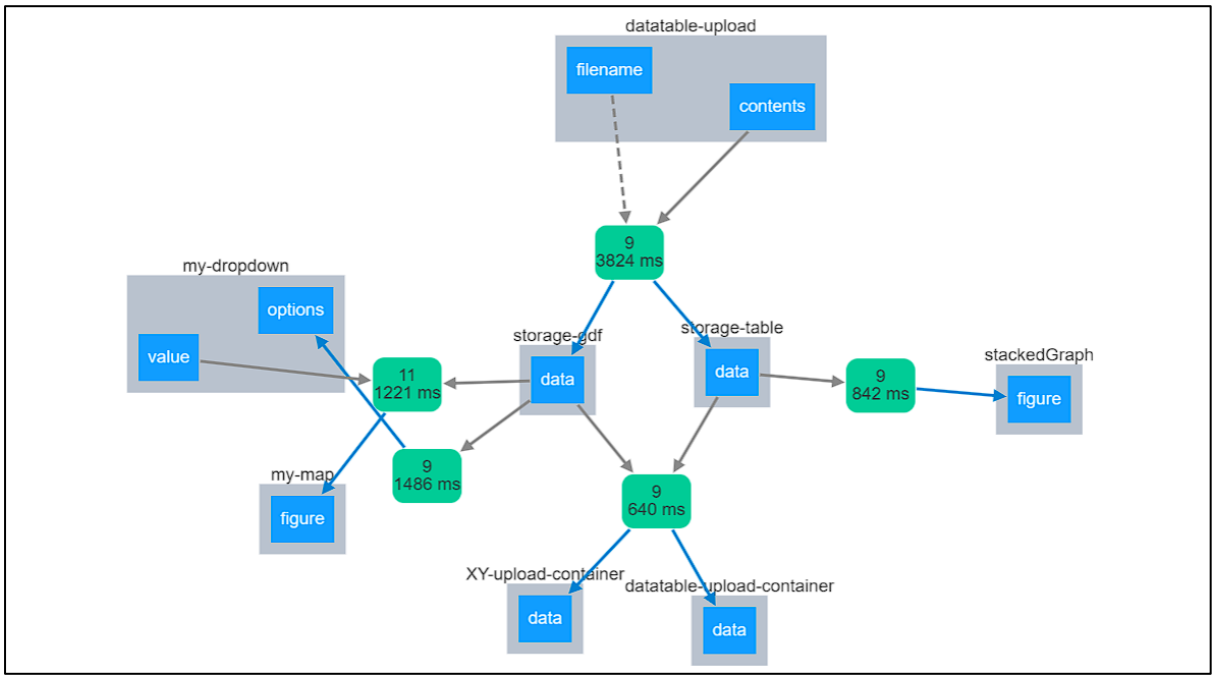


Figure 4: Callback graph utilised in interactive data applications. Green boxes denote computational nodes executing data transformations. Solid arrows show data flow direction, with outputs feeding into successive nodes. Dotted arrows indicate event-triggered processes. Blue boxes represent initial inputs or interfaces initiating the callback sequence. Arrow-labelled 'data' signifies the transmission of processed outputs, while adjacent numbers with milliseconds denote the operation sequence and execution time.

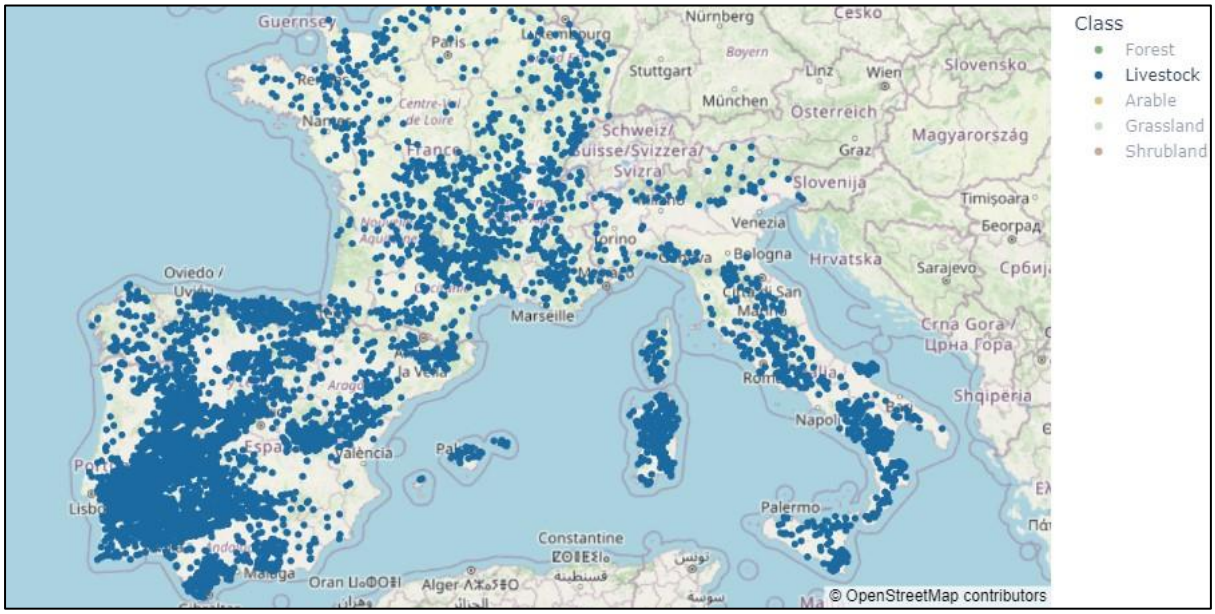


Figure 5: A map visualisation representing the distribution of the Livestock agroforestry class across the selected countries in the EU, derived from the LUCAS dataset.

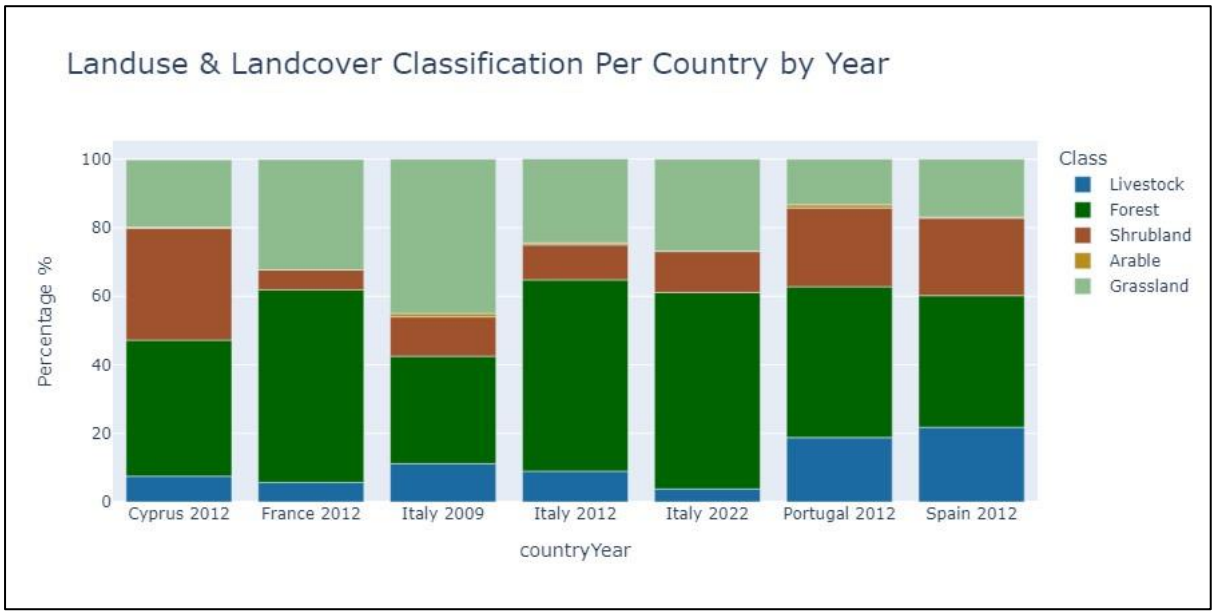


Figure 6: Stacked bar chart displaying the percentage distribution of LULC classifications per country by year, based on LUCAS survey data. Each bar segment represents one of five land cover classes: Livestock, Forest, Shrubland, Arable, and Grassland, illustrating changes in land composition over time for selected EU countries.