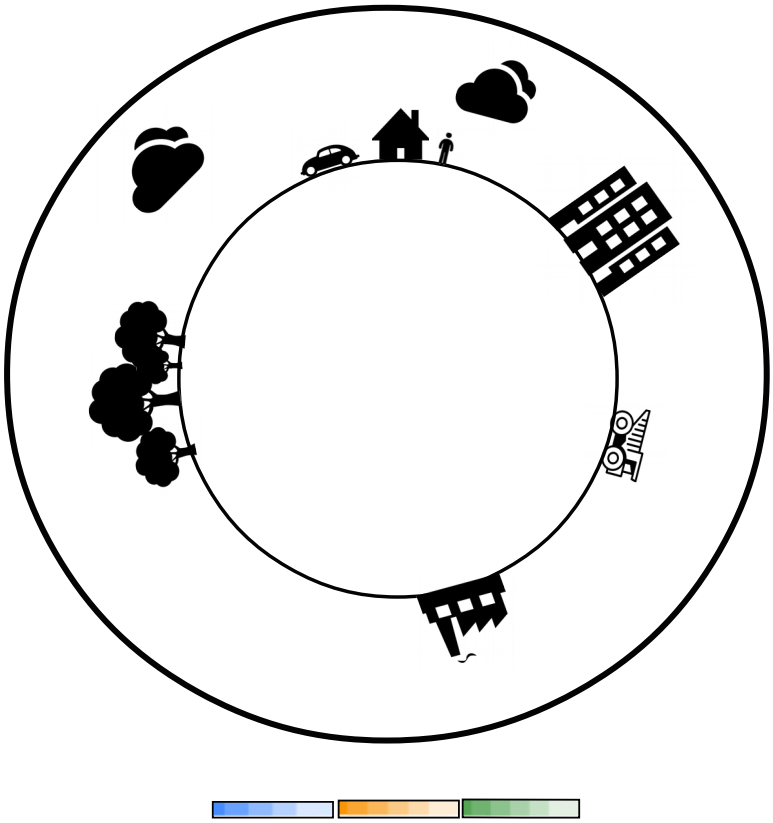
# ExioVisuals

Combined front-end & back-end development with Python/Django



*Name: Sidney Niccolson*
*Date: July 4th,  2016*
*Student number: s1650548 (Leiden) -  4452909 (Delft)*
*Field: Industrial Ecology*
*Mail: sidneyniccolson@gmail.com*
*Background: Bio-informatics*

# Table of Contents

# 1 Introduction

The goal of this project is to allow researchers to filter and visualize Environmental Input-Output (EIO) data in an convenient way using piecharts, treemaps, time-series and Sankey diagrams. There are various ways to achieve this and following the current trends of utilizing the benefits that come with the Internet, Web-based applications have become important. Django[1] is a Python based Web framework for building Web-applications. It was designed for making Web-application development easier and faster. Django uses a combination of Python code, JavaScript, templates based on HTML[2] and databases. In terms of data-visualization methods, there are many technologies available, however this project is mostly based on the relatively new visualization technologies D3plus[3] and NVD3[4]. This can be integrated in the Django framework and in addition a mySQL[5] and HDF5[6] database is built to handle data retrieval and data requests. Data-handling can be achieved by converting the EIO data to a HDF5 database. All concepts described above will be elaborated in following sections. Chapter 1 explains the core concepts that make up the basis of this project, while Chapter 2 elaborates on the translation from necessary data structures towards data visualizations. In Chapter 3 some examples of the implemented application's functionalities will be discussed.

## 1.1 Core concepts – A short summary

**Terminology**

- Applications: programs, services.
- Protocol: a set of defined rules for communication and representation of data.
- User/client: a person that makes use of applications and tools.
- Server: Web server. Computer that provides Web services or Web applications and therefore makes them accessible over the Internet.
- Web service: application that can be used over the Internet with application-application communication.
- Web application: application that can be used over the Internet with web browser-application communication. At a low-level Web services are Web applications.
- Web interface: user side presentation layer for invoking a Web application. Can be HTML but also JavaScript or similar.
- Function: a method inside a program that performs a specific task (process).

**System: Open Source, Linux and Apache**

Linux[7] is an open source Operating System (OS) wherein an operating system serves as the foundation for computer programs to run. The OS functions as a supervisor of power & memory and acts as an interface between the software applications & the hardware. By having open source software, the source code is freely available, in turn any user can contribute to the development of the OS. A compiler converts source code in computer understandable language. Windows and Mac are non-open source operating systems, hence the use and modifying of the OS is limited or restricted. In this project all software used is based on open source software packages meaning that all software can be shared under the right conditions. The web application is made accessible on the Web through a Linux-based server. In addition the Apache[8] software is installed on the server to manage direct communication between the Web application and users on the Internet (see section 1.3 for more detail).

**Programming Language: Python**

Python[9] is an popular programming language for developing software applications. A programming language is a human-understandable language that can be compiled to computer-understandable code. Python is a relatively quick development language, because a lot of elements are performed in the background. Other programming languages such as Java, C++ can possibly require more programming, because more code needs to be explicitly written down to explain the computer what to do with for instance the data types and resources of memory. Because source code can become fairly complex even with Python, using an Integrated Development Environment (IDE) is essential for development. IDE's can help with writing code and managing application dependencies. In this project PyCharm[10] was used as the IDE.

**User interface: HTML**

HyperText Markup Language (HTML) is an user side presentation layer. Web browsers can render HTML code and visualize this through web pages. HTML can be seen as part of the XML[11] based standards. XML based code is for exchange and storage of data in both a human and computer understandable way. XML principles are becoming more and more popular, because in order to achieve platform independence standardized communication is needed, and XML provides that. Unlike XML, HTML is only for the display of information and uses predefined tags (meta-data). Elements are the “bodies” of these tags, that is the actual information/data. For example a text document can contains the tag <html> as meta-data. Within the HTML you could have a tag called <body> in which you could place any text you want to show (actual information). At that point the meta-tag tells to any capable software that is going to use the text document to actually run it as HTML and show whats inside of the <body> tag.

**Inline programming language: JavaScript with JSON**

JavaScript is a way to program the behavior of web pages. HTML is considered static in the sense that the code is non-dynamic, it simply displays information and does not take information. JavaScript can be used inside HTML pages (hence the “inline”) to develop dynamic aspects of web pages, therefore taking user information such as a click on a web page and adjusting what is displayed. Javascript is used throughout this project, mainly JavaScript libraries such as D3plus[12], NVD3[13], Leaflet[14] and Jquery[15] (latter for dropdown menu's) .

JSON[16] is somewhat more abstract and is officially called a *language agnostic* data-interchange format. It is inspired by data interchange in JavaScript through JavaScript objects. Basically there are for example occasions in which data needs to be shared in different applications. As mentioned XML can be used for this, but JSON is lighter and is more close to data types in normal programming languages. JSON structures are used for communication between the mentioned JavaScript libraries and Python code. However it should be noted that Python dictionaries look a lot like JSON objects using them interchangeably may cause errors in some occasions.

**Styling and layout: CSS & Bootstrap**

CSS is a way to style your web site and is just like JavaScript invoked in HTML tags. In this application the CSS framework Twitter Bootstrap[17] is used. Bootstrap strong points are the capability to display information in grid structures, mobile device compatibility and bundles JavaScript for interactive drop-down menu's, tabs and navigation bars. Also stylized buttons, icons can be achieved through Bootstrap. Bootstrap can either be invoked in HTML with external stylesheets (basically the styling data resides on a cloud-server and is referenced in HTML) or can be downloaded and used locally (reference is in this case to the local directory).

**Communication: HTTP**

Communication between Web applications and users can be achieved via Internet protocols. HTTP (Hypertext Transfer Protocol) is one of those protocols for communication. HTTP uses other protocols named TCP/IP (Transmission Control Protocol/Internet Protocol). The IP provides addressing (every computer has a unique IP address). TCP provides a way for sending data in between applications over the Internet. Web applications uses HTTP as the protocol for communication in general. Clients can send HTTP requests in the browser and retrieve HTTP responses from Web applications.

**Dealing with data: Databases**

Data can easily become redundant or unclear when dealing with large amounts. Databases provide a way to structure, organize and query data in a standardized way. Databases have a long history of development and explaining the details is outside of the scope of this

report. However in this project the relational database management system mySQL is used. mySQL similar to other relational databases uses tables with predetermined rows and columns. Tables can be linked together through “keys” creating the relations between tables. Next to mySQL, HDF5 is used for the EXIOBASE data. HDF5 is a hierarchical organizational database. In short it is a data model for storing high volumes of data in a flexible/efficient manner. It contains folder-like containers and datasets that store data in matrices-like objects.

**Visualisations: D3(plus)/NVD3/Leaflet/GeoJson**

D3 is a JavaScript library for creating dynamic, interactive visualizations. D3plus is an extensions to the D3 library, this extension is mainly used throughout the project. NVD3 is a Django-wrapper which is basically reusable Python code that utilizes D3. Leaflet is a JavaScript library for developing user-friendly interactive maps. A combination of these three visualization libraries are used. GeoJson however is not a library, but are geographical JSON objects. This means that it contains geographical data that can be used by the Leaflet JavaScript library. This is valuable for mapping data to geographic maps.

**Combining it all: Web frameworks**

Web frameworks are designed to eliminate the overhead needed in web application development. In simple words without Web frameworks you have to explicitly code all aspects such as HTTP communication, python code that needs to interact with JavaScript code. Web development would take much more time and would be very difficult to achieve. As explained in the introduction Django is a Web framework for Python that can combine all these components described above.

## 1.2 Django infrastructure

In order to elaborate the functionalities of the web application, it is necessary to explain some of the mechanisms behind Django. There are 8 modular concepts of Django that are relevant to discuss in this section namely: Views, Templates, Forms, Models, Static files, Administration, Settings, URLs.

- **Views** can be seen as the heart of the Django application, this is the part where Python code resides and all run-time processes are handled.
- **Templates** is where all the HTML code resides, which will be served to the client side.
- **Forms** are abstractions how user data should look like and how it should be handled.
- **Models** reflect the infrastructure of the database used. Each model maps to a single database table.

- **Static files** is a directory accessible to the Django application in which CSS, JavaScript and images are served.
- **Administration** is the core management of the application itself and database once it is running on a server. Think of registration of users, adding database tables or removing tables.
- **Settings** is where configuration of Django takes place. In addition it tells a server where necessary files can be found and which database to use.
- **URLs** are addresses that explain where certain pages of a site are and redirects users to those pages. Django has a relatively simple URL mapping mechanism using regular expressions. The URLs are strongly linked to the Views as Views can either invoke templates via URLs or templates can invoke Views with POST/GET requests via URLs. See figure 1.1 for an overview of Django's infrastructure:

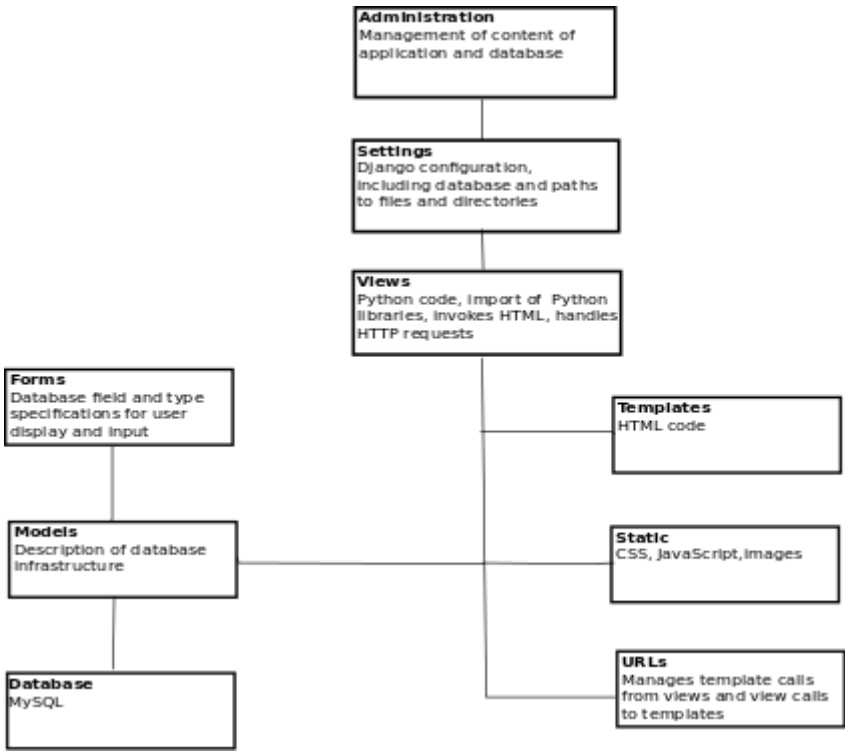



Figure 1.1: Django infrastructure

## 1.3 Django mechanisms & the Apache server

In section 1.2 the Django infrastructure is explained, for simplification we will now view the Django infrastructure as a black box in this section. As with all Web applications just like Django, the application needs to be deployed on a production server (e.g. Apache) eventually. Python is originally a programming language and was not built to program Web applications. Therefore certain protocols where developed in order for Python to communicate with Web servers and Web frameworks. Web Server Gateway Interface (WSGI)[18] is the common Python protocol and Django uses this as well. Apache on the

other hand is a piece of Web server software that serves any client HTTP requests and handles HTTP responses and can be installed on for instance a Linux Operating System. How is Apache linked to Django? Well this can be explained through the Apache's mod_wsgi module which hosts WSGI applications including Django. As mentioned in section 1.2 databases reside on the server as well. However it is common to separate the front-end application (Django) with the back-end (databases) on different servers. Therefore at the CML department in Leiden two servers are available. One called Ronin (for front-end) and the other called Akira (for back-end). These should be linked together in Django Settings. Below an overview of the communication between client, Apache, Django and database.

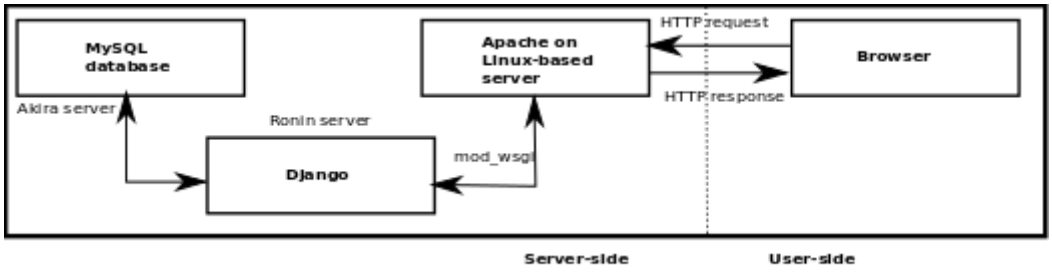



Figure 1.2: Communication between all relevant modules

Please refer to appendix 1 for system requirements of this application, appendix 2 for installation of Django on server and Apache configuration.

# 2 EXIOBASE & ExioVisuals goal

## 2.1 EXIOBASE

Input-Output analysis is a way of analyzing interdependence of an economies productive sectors[19]. Many macro-economics related questions can be answered with the use of IO analysis. For example questions like “what is the indirect impact of a change in supply or demand in sector A on the interrelated sector B” or “if the government gives subsidies over products what will be upstream and downstream effects?” These productive sectors can be viewed as industries in which companies produce certain goods. Some companies need goods from others or use primary inputs such as mining. All these aspects are reflected in IO tables that show interdependence of the economy. Extended Environmental Input-Output analysis (EIO) takes into account not only monetary or physical flows between industries, but also their related emissions. Therefore EIO provides ways of relating emissions to specific sectors and in addition aspects such as a change in demand can be assessed. See figure 2.1 for an example of flows of materials with regards to a given product.

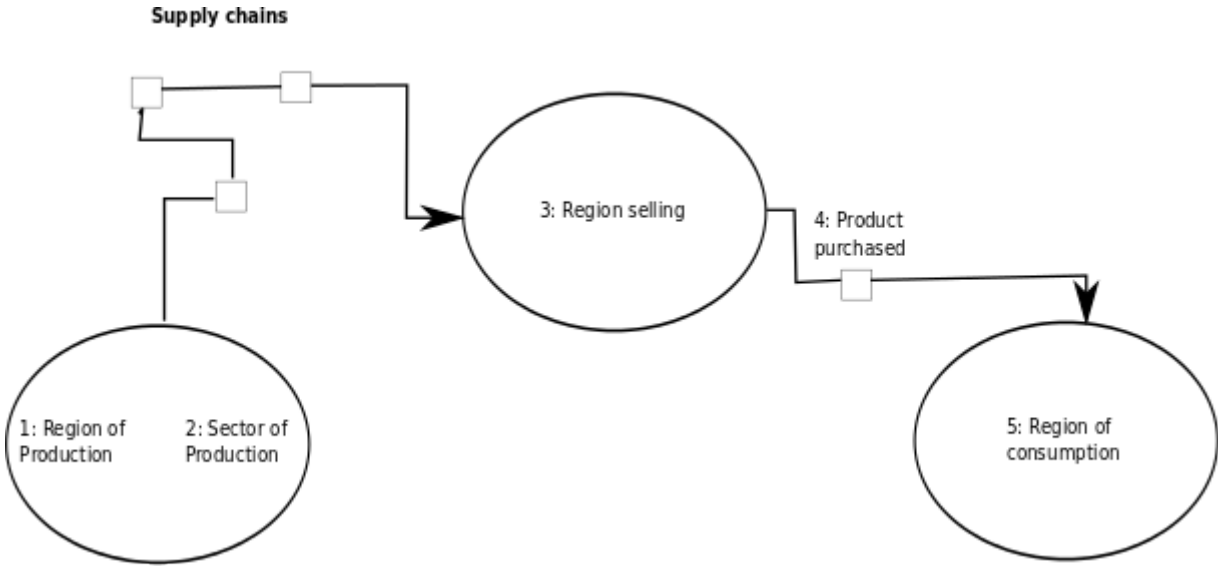



Figure 2.1: the flow of materials for a given product.

EXIOBASE is a global Multi-regional Environmentally Extended Input Ouput database (MR EE SUT/IOT) relating environmental impacts to the final consumptions of product groups. EXIOBASE uses the conventions of the System of National Accounts (SNA) in which the flow described in figure 2.1 can be assessed according to the following notation:

$$\Delta r = b^T (I-A)^{-1} y$$

wherein:

- Δr > changes in emissions
- b > environmental stressor
- $(I-A)^{-1}$ = Loentief inverse (L) > Economic structures and networks (total requirement matrix)
- y > demand

Data from the EXIOBASE will be used to render relevant visualizations.

## 2.2 ExioVisuals web application

Now that most relevant aspects are described, the exact functions of the web application can be described. Django is capable of handling client inputs and rendering plots. Ideally pre-calculated MR EE SUT/IOT tables reside on the database that can be queried by clients. Django than renders visuals to show to users. ExioVisuals is the Django based application developed that retrieves user inputs to visualize EXIOBASE data. As described in Chapter 1 the EXIOBASE is translated to HDF5 data hierarchies. mySQL is used to develop user selectable inputs (see section 2.3 for more detailed information). Below a short elaboration on the respective visualization types relevant for EIO data.

**Sankey Diagrams**

Sankey diagrams are flow diagrams that can show the flow of commodities between countries. Hereby analyses can be done on which commodities contribute to large emissions and which companies are involved. Below in figure 2.2 examples of such diagrams.

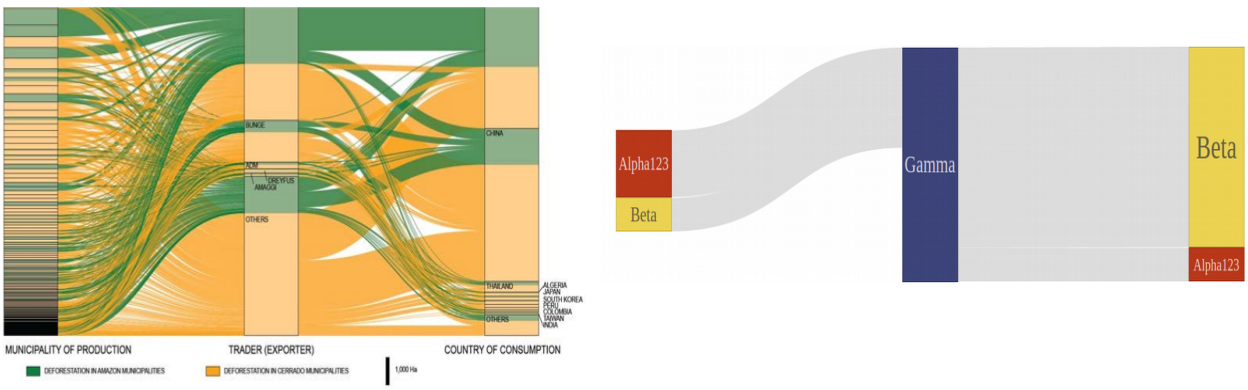



Figure 2.2: Sankey diagram. (left) An example of a Sankey diagram with real data (source: http://www.sei-international.org). (right) A dummy implementation in ExioVisuals using NVD3.

### Timeseries and piecharts

Timeseries are ways of plotting data over changes in time. For instance in EIO we can assess changes of environmental impacts such as GHG emissions of the production of a commodity in a specific country or for a couple of countries. Piecharts can represent for instance sugar cane production in countries and there contribution to global sugar cane production. Below examples of D3/NVD3 visualizations implementations with dummy data in Django.

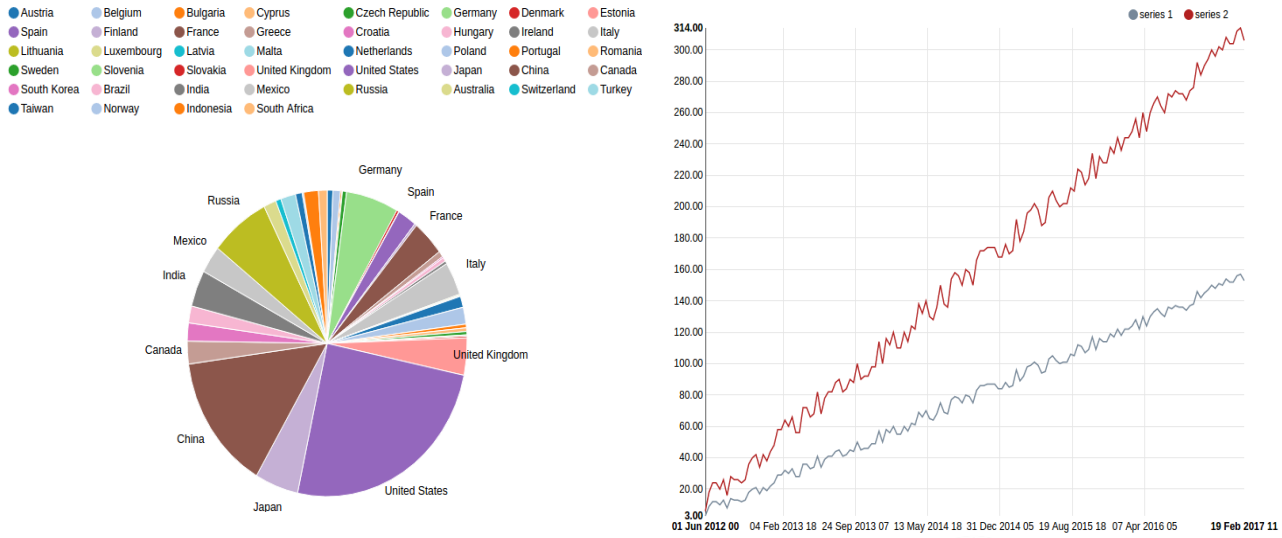



Figure 2.3: D3plus/NVD3 visualizations in ExioVisuals

### Treemaps

Treemaps can show visuals on for instance what is imported by country A from country B. Next to knowing what is imported we can also see what commodities have the largest distributions (see figure 2.4).

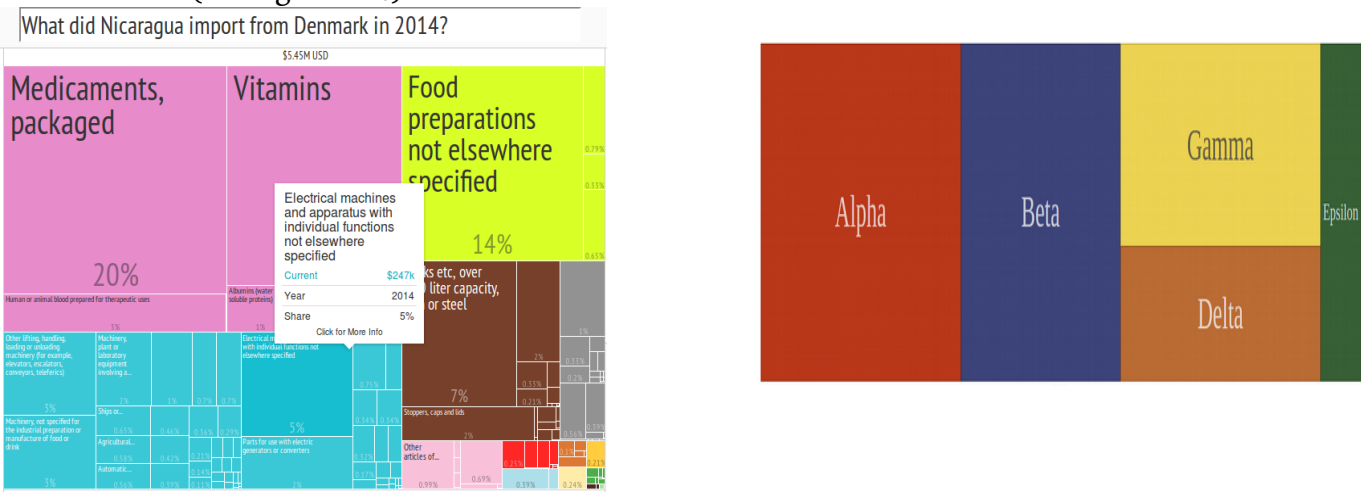



Figure 2.4: Treemaps. (left) Treemap on Nicaragua imports from Denmark (source: http://atlas.cid.harvard.edu/) (right) Treemap in ExioVisuals with dummy data

### Web mapping

Web mapping directly maps specific geographic data to region-divided world maps. This technique is commonly used in Geographical Information Systems (GIS) with the use of standardized geographic data infrastructure and plotting mechanisms. See below in figure 2.5 (left) an example from D3. Django-leaflet provides ways of webmapping (figure 2.5, right image).

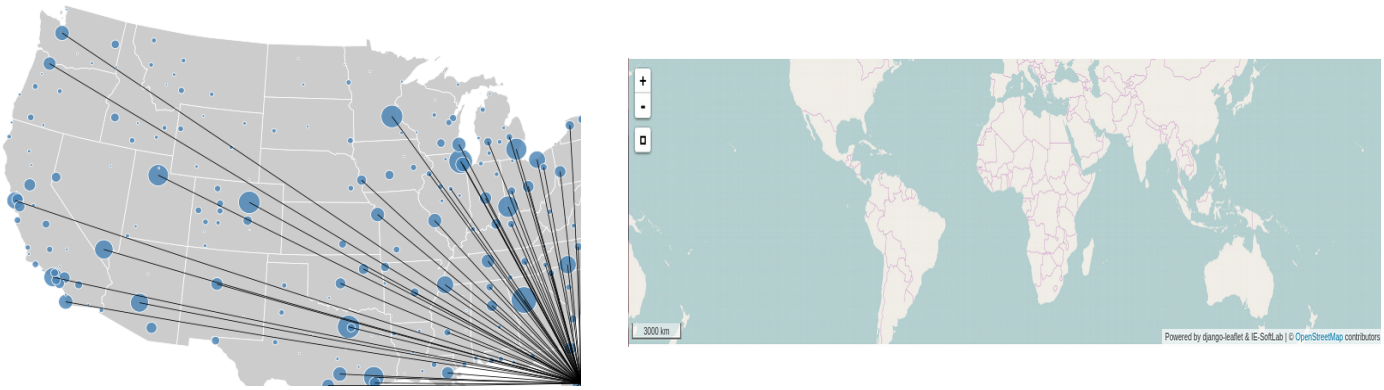



Figure 2.5: (left) Webmapping in D3. (right). Webmapping with Django-leaflet in ExioVisuals. The figure to the right is an empty skeleton, but if geographical data is supplied with GeoJson it possible to denote for example regions with specific colors.

## 2.3 Database development and structuring

### mySQL

As explained Django can be used to setup the basic skeleton of tables that an mySQL database should have. In addition initial data can be passed in the database with Python “populate” scripts. In this way the database can be used indirectly via Django providing ways to populate the database with values, setup the table structures and making sure the database stays consistent. For example with specifications in the Models such as for example fields that can only contain integers or cannot be NULL. Once the mySQL database is configured appropriately Django provides query mechanisms to gather data from the database.

The current lists of industries, sectors (200) and regions (49) from the EXIOBASE where implemented as tables in mySQL. Those lists where aligned with the NACE v1.1 (Statistical classification of economic activities in the European Community) standard for industries and sectors to create higher hierarchical levels. See figure 2.6 for rendering of sectors and regions as user input fields in ExioVisuals.

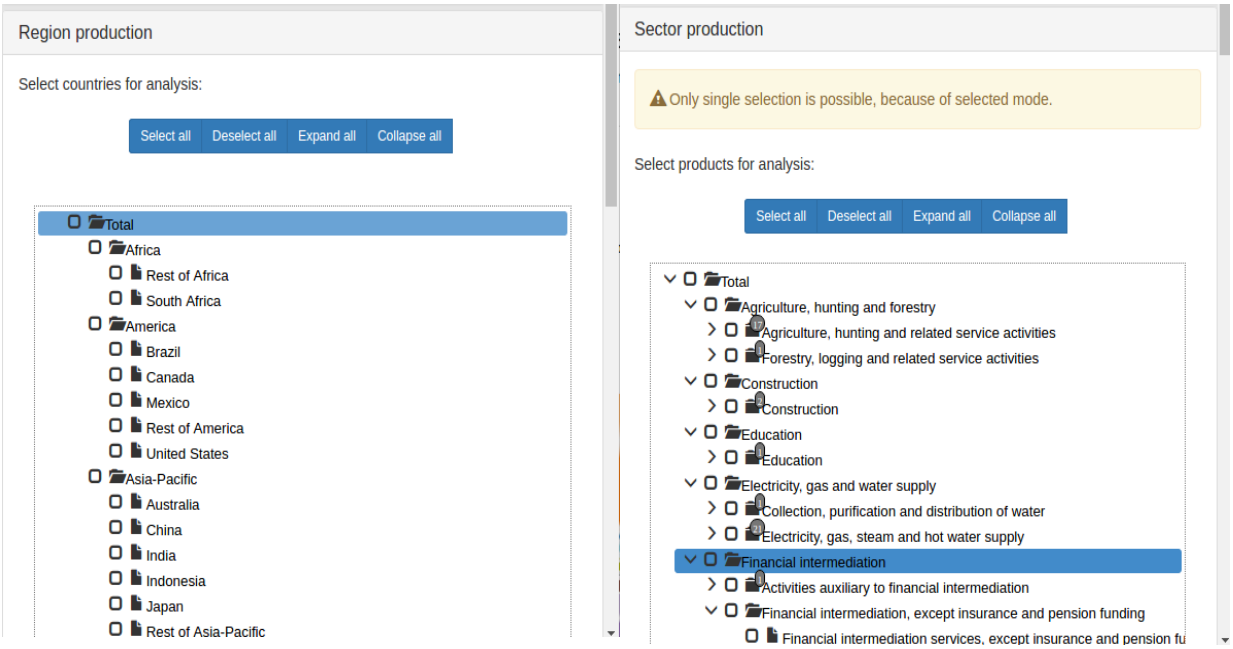



Figure 2.6: ExioVisuals user input menu. (left) the 49 regions of EXIOBASE can be selected. (right) 200 sectors according to NACE hierarchy

The tree-like structure that denotes the hierarchical levels is achieved with JavaScript's Fancytree library[20]. As the number of regions and sectors may change in time for EXIOBASE it is possible to easily update these tree-like structures with the implemented setup. This is achieved through the use of Comma Delimited Files (CSV) files that are created according to the respective regions and countries to populate the mySQL database. See figure 2.7 for a schematic overview of the process of populating and rendering the tree-like structure in the Django ExioVisuals application.

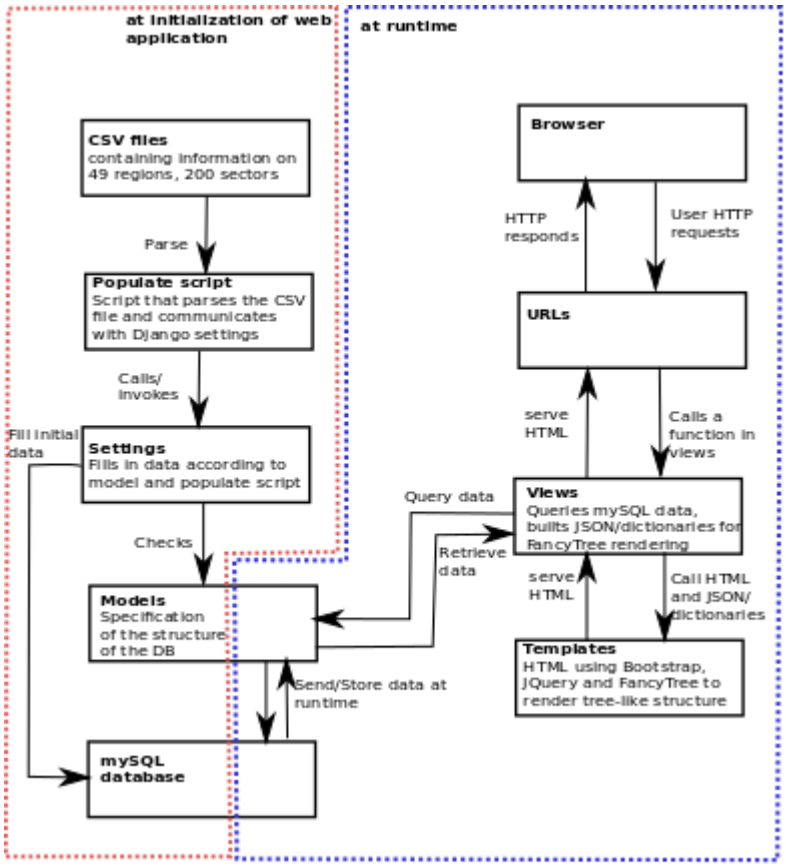



Figure 2.7: the creation of user input tree-like structures for regions and sectors

If there is a change in the regions or sectors, the respective CSV file can be updated and a new tree-like structure can be generated with the populate script. However the mySQL entries should be removed first before the populate script can use the CSV files to prevent duplicate data. Refer to appendix 3 for snapshots of the CSV files opened in a document software tool that can read CSV files. Refer to appendix 4 for mySQL database setup configurations including remote access and an example of a small populate script. In appendix 5 both populate scripts for sectors and regions can be found including the Django model needed to populate mySQL.

**HDF5**

HDF5 is used as the database for the actual EXIOBASE data. The implementation follows the usage of MATLAB[21] matrix/vector arrays pre-created with EXIOBASE data. These arrays contain the L matrix, b,y vectors and additionally the h vector which is the final demand categories vector. The version of EXIOBASE used includes data from 1995-2011. Each MATLAB file signifies a specific year. A python populate script is developed that (1) reads in these MATLAB files, (2) does needed calculations, (3) populates a HDF5 database. Step 2 embodies the calculation of the final demand and the footprint of the 49 regions and 200 sectors per specific year. H5py is the Python library used for this stage of the project, and enables the creation of the HDF5 database and querying of that database. See figure 2.8 for an overview of the process.

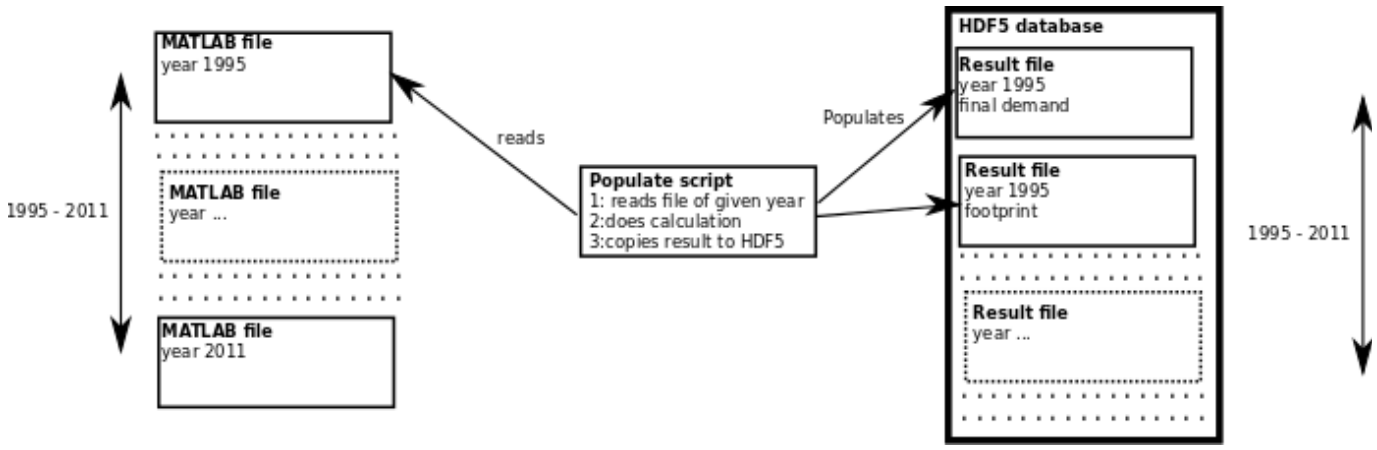



Figure 2.8: overview of HDF5 database creation. Each year has two files namely final demand and footprint.

The populate script created a compressed HDF5 database with multi-threading (simultaneous processing to speed up the performance), below in table 2.1 details on the runtime on creating the HDF5 database and it's final size.

Table 2.1: HDF5 database

| Max. memory usage | Time taken | Size of database | Matrix dimensions final demand of file per year | Matrix dimensions footprint of file per year |
|---|---|---|---|---|
| 3166.78 MB (approx. 3GB) | 4.6 hours | 8.3 GB | (17, 49) | (49, 200, 49, 200) |

For retrieving the memory usage the Python module memory profiler[22] was used in combination with a library called pdb. Logging was used to measure the time and a truncation and compression step was implemented to minimize the size. In order to analyze whether the MATLAB data generates the right HDF5 database containing the results, experiments and validation was necessary. Below in table 2 results from query experiments that was compared with Octave[23] (matrix computation software) output. The result file containing footprint data has 49 datasets according to importing regions (region of consumption). Each dataset has the matrix dimensions noted in table 1 last column. These dimensions signify respectively [region of production, sector of production, region selling, sector of consumption]. With these dimensions EIO research questions can answered through HDF5 queries (see Chapter 3 for an example).

Table 2.2: query experiments that matched the results in the program Octave

| Experiment | Octave query | HDF5 query coordinates | Dimensions of result | Output |
|---|---|---|---|---|
| Footprint of 1995 on import region 0 | 0,0,0:10,0,0:10,0 | Data[0,0:10,0,0:10] + initial traversal to 1995 and import region 0 | (10,10) | See appendix 6 |
| Footprint of 1995 on import region 0 | sum of all data | testSum = sum(sum(sum(sum(tmp))))) tmp = data[0:49,0:200,0:49,0:200]+ initial traversal to 1995 and import region 0 | absolute | 137624606477 |

Refer to appendix 7 for the developed HDF5 populate script. Below in figure 2.9 an overview of the process with respect to Django.

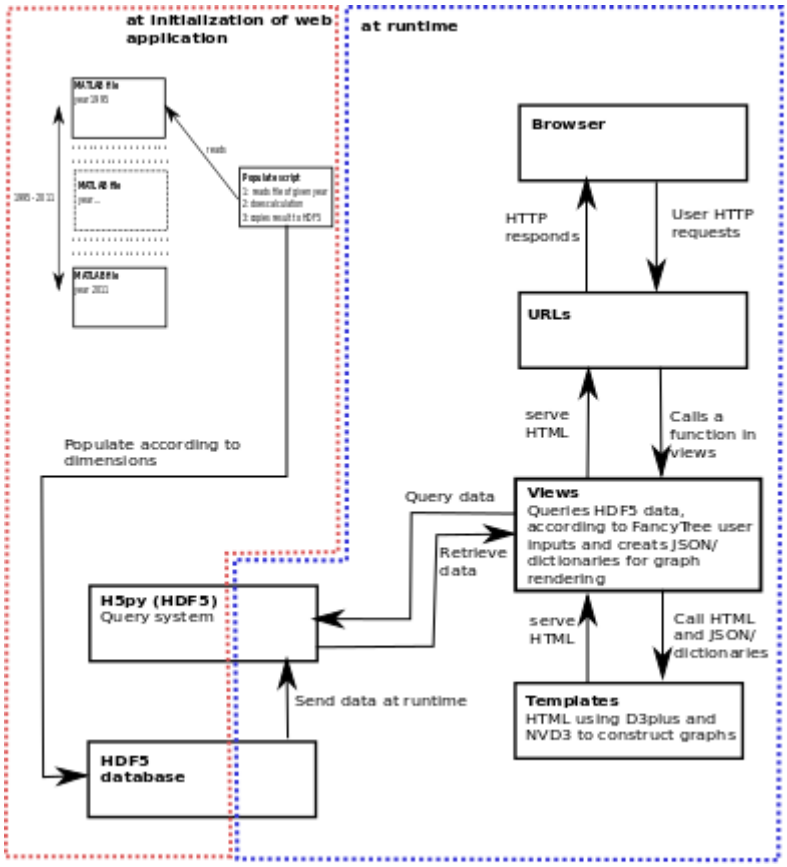



Figure 2.9: an overview of HDF5 in Django.

User inputs from FancyTree are used to create a query that includes sets of integers relating to coordinates in the HDF5 database. This query retrieves data and that is translated to JSON/dictionaries to construct graphs.

# 3 ExioVisuals functionalities

ExioVisuals is an effort to visualize complex environmental Input-Output Analysis results from EXIOBASE. The visualizations are according to user input and are interactive. This section explains the functionalities of the application in a tutorial-like manner. The web application can be found with the following link *cml.liacs.nl/softlab/exiovisuals* (user: cml, pass:VanSteenis). And the whole project is found on GitHub at:

*https://github.com/SidneyNiccolson/IEplatform/tree/master/IEMasterProject*

## 3.1 Homepage

On the homepage in the navigation bar five tabs are available in which each signifies a visualization type. See figure 3.1 below for a screen shot that shows the respective tabs.

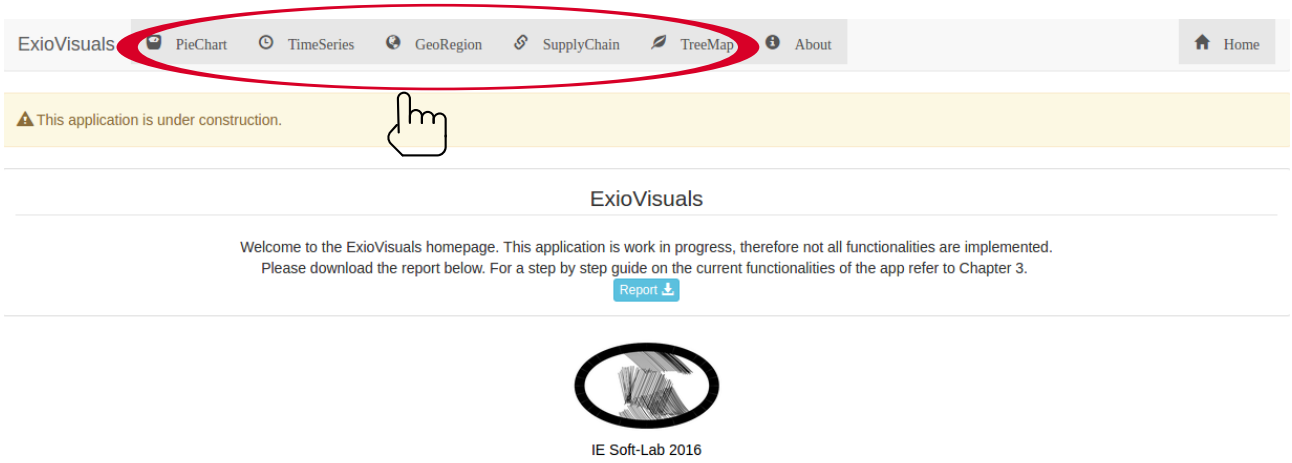



Figure 3.1: the homepage of ExioVisuals

- **PieChart**: which is aimed at visualizing the distribution of environmental footprints.
- **TimeSeries**: visualizes changes in time dimensions.
- **GeoRegion** (not functional yet): is developed to integrate the concept of Web mapping. Web mapping directly maps specific geographic data to region-divided world maps. This technique is commonly used in Geographical Information Systems (GIS) with the use of standardized geographic data infrastructure and plotting mechanisms.
- **SupplyChain** (not functional yet): focuses on flows in a Sankey Diagram manner.
- **TreeMap** (not functional yet): Similar to PieChart is focused on distribution analysis. In contrast to PieChart, this can be used to visualize multiple dimensions.

A click on a visualization type sends the user to the respective visualization page in which an additional navigation bar is shown. Each page of an visualization type has a set of three

global settings called “mode”, “problem”, “filter”, see image 3.2 below.

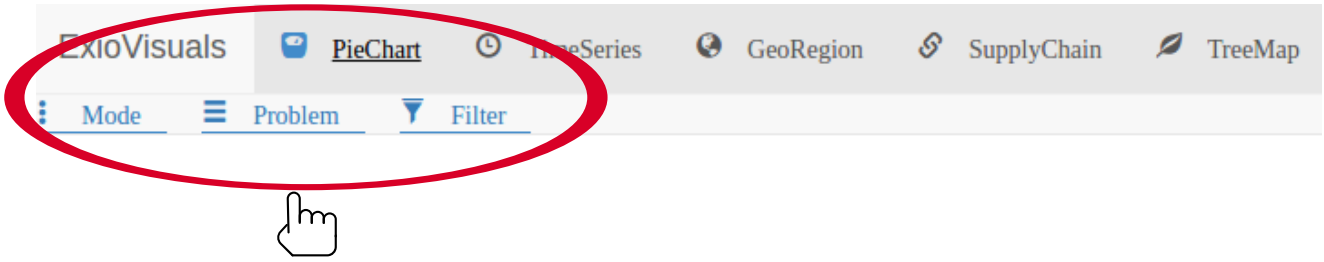



Figure 3.2: The three global settings at the visualization page of PieChart.

- **Mode (decomposition):** Serves as a measure of breakdown on the various dimensions the database contains. For example in the case of PieChart we want to break down to one dimension as piechart graphs can only visualize the distribution along one dimension. For other visualization types this may differ such as TimeSeries in which there is an additional year dimension.
- **Problem:** is a function that is not implemented yet. This allows for a breakdown of the environmental problem a researcher may wants to assess.
- **Filter:** takes inputs on what aspects the user want to filter according to the selected “Mode”. Filter is a dynamic setting that changes according to the “Mode” selection.

Clicking on one of them will render a drop-down menu (widget) in which user inputs can be given.

## 3.2 The PieChart visualization type and two cases

As of now the PieChart and TimeSeries are the only truly functional pieces of the application, therefore step by step examples will be shown. For the TimeSeries example refer to section 3.3. Two examples will be discussed in this section that elaborate the dynamics of the application for the PieChart. The PieChart example includes the following research questions:

- (1) What is the contribution of the various electricity generation types in China used for the construction of machinery and equipment in terms of absolute footprint in the year 2011?
- (2) What is the distribution of environmental footprints of electricity generation by coal generated in China, Indonesia, Japan and India in the year 2011 with respect to the construction of machinery and equipment exported by China and imported by Germany?

To answer these two questions we will go simultaneously through both questions in a stepwise manner. If the interrelations between flows of materials that make the absolute footprint are not clear, please refer back to figure 2.1 in Chapter 2.

**Step 1: Select your mode.** Six selections are possible namely decomposition of absolute footprint of consumption split by “Consumed product category”, “Consuming region”, ”Sector where impact occurs”,“Country where impact occurs”, ”Environmental pressure”, ”Year”. See image 3.3 below.

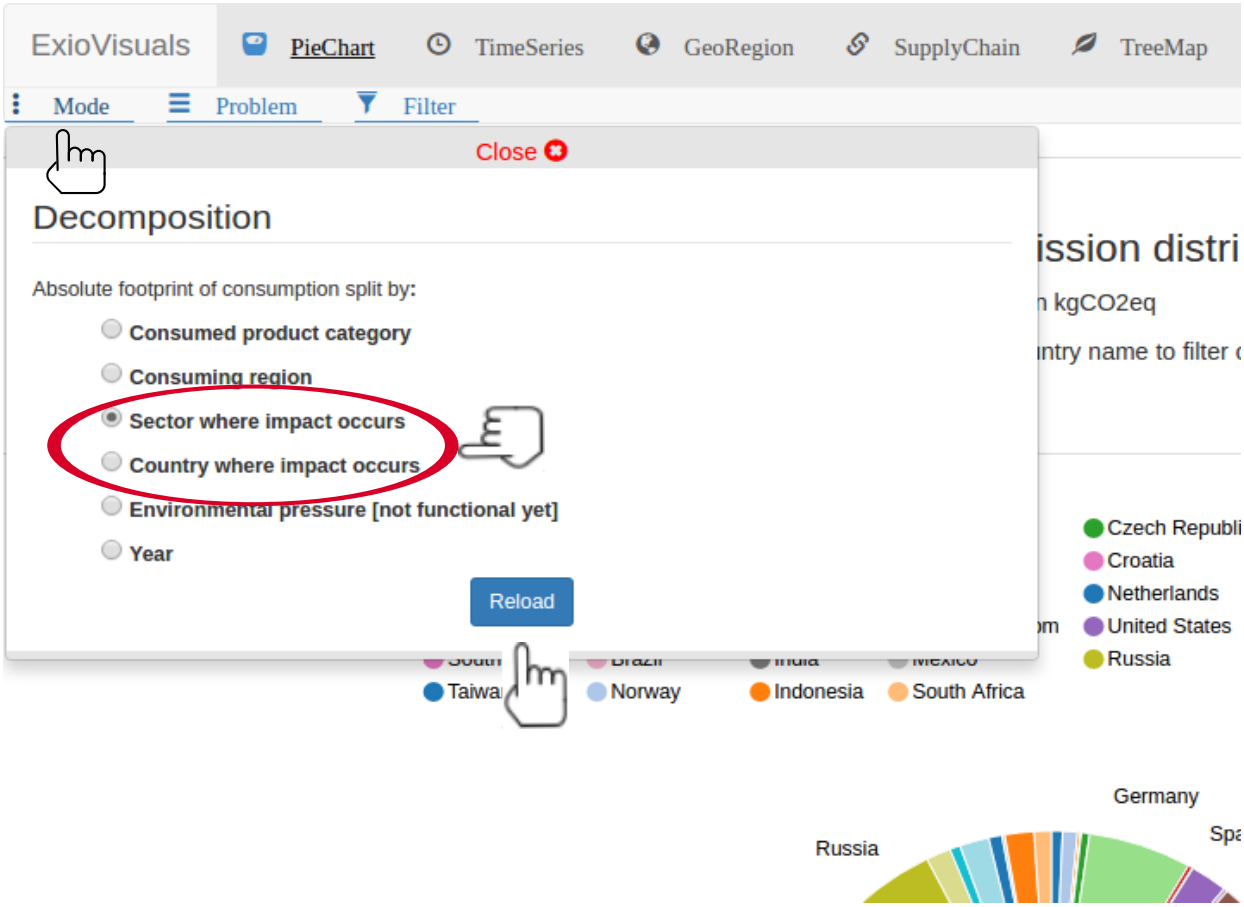



Figure 3.3: The drop-down menu of the setting “Mode”

As seen in figure 3.3, selections can be made that serves our step 1 for the research questions. We will select for question 1 “Sector where impact occurs” and for question 2 “Country where impact occurs”. Clicking upon the reload button (figure 3.3), will create selectable user inputs for the global setting “Filter” according to the selection made in “Mode”. A pop-up notification will be shown based on your selection. See figure 3.4.

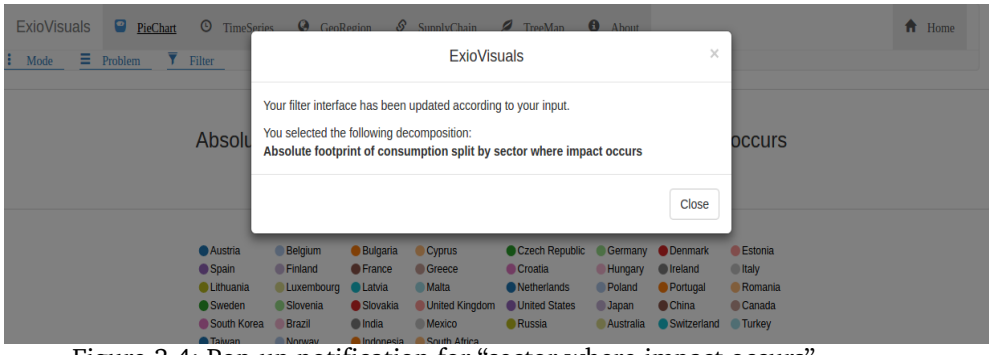



Figure 3.4: Pop-up notification for “sector where impact occurs”

**Step 2: Select filter settings.** Because piechart graphs have only one dimension to

visualize, limited selections can be made by the user. Below in figure 3.5 seven options that are available in general in the filter setting.

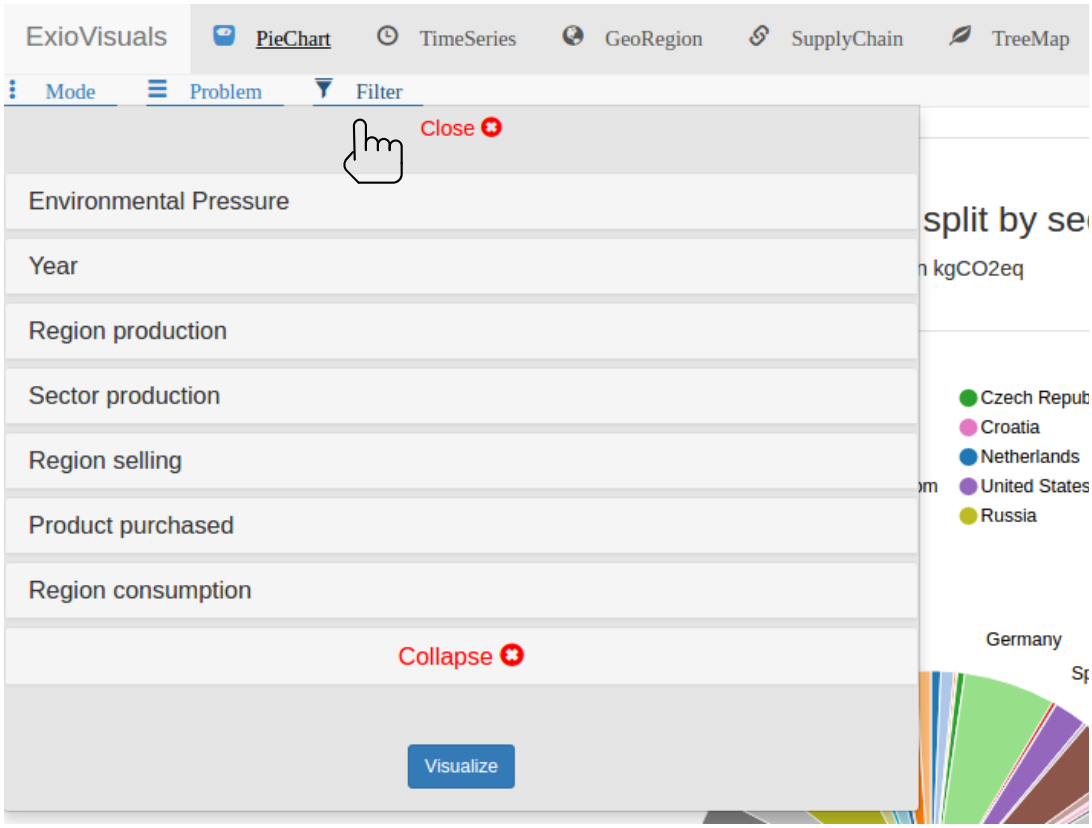



Figure 3.5: Drop-down menu for Filter setting, showcasing the seven options for input available.

According to our selection in “Mode”, most user inputs in the filter function will now only be capable to take a single selection except one option, hence the notion of having only one dimension for piecharts. Back to the two questions: “Sector where impact occurs” relates to “Sector of production” in the Filter menu and “Country where impact occurs” relates to “Region of production”. In the case of Q1 the “Sector of production” will be a multiple select setting. In the case of Q2 the “Region of production” will be a multiple select setting. However to answer our questions all options except environmental pressure needs to be filled in, below a translation to a short table for the settings for each question.

Table 3.1: Translation of research questions to parameters in the application

| | Q1 | Q2 |
|---|---|---|
| Year | 2011 | 2011 |
| Region of production | China | China, India, Indonesia, Japan |
| Sector of production | Electricity by coal, biomass and waste, gas, geothermal, hydro, nuclear, petroleum, solar photovoltaic, solar thermal, tide, wave, ocean, wind | Electricity by coal |
| Region selling | China | China |
| Product purchased | machinery and equipment | machinery and equipment |
| Region of consumption | China | Germany |

See image 3.6 below for a screen shot of the selection menu for both questions for “Sector of production”.

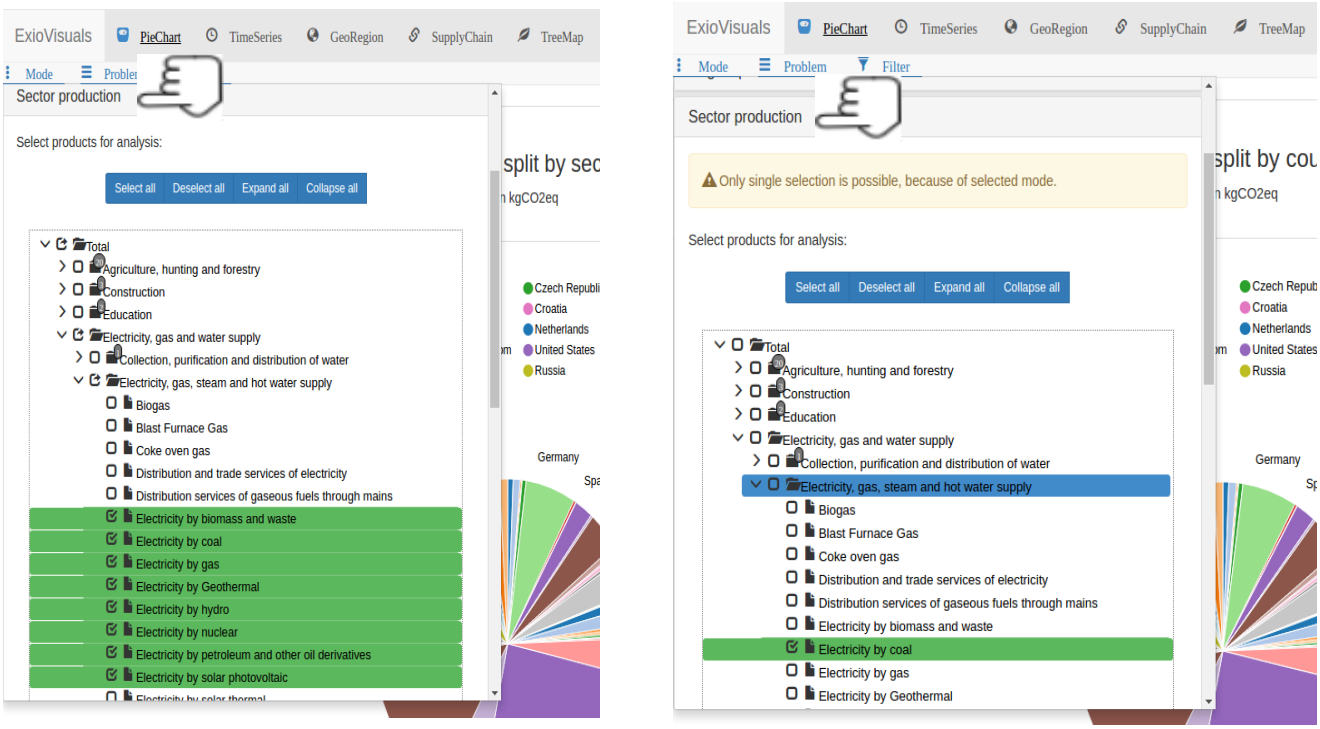



Figure 3.6: Sector of production selection inputs. (left) in the case of Q1, (right) in the case of Q2. In the right image it is visible that a small notification message is given that only a single selection can be made. In the case of Q2 it is not possible to select multiple sectors, because this dimension was not chosen in “Mode”.

After all the needed inputs are given by the user, he or she may click on the button “Visualize”, see figure 3.5 on the previous page. Based on the input a query is made and the results are passed on to a graphics library (D3plus/NVD3) to generate the graphs. See figure 3.7 for the visualization result for Q1 and in figure 3.8 the result for Q2.


Figure 3.7: The generated piechart based on Q1.

Absolute footprint of consumption split by country where impact occurs

Plotted in kgCO2eq

| Type | Query |
|---|---|
| Select mode | Absolute footprint of consumption split by country where impact occurs |
| Environmental pressure | [] |
| Year | 2011 |
| Region of production | ['Japan', 'China', 'India', 'Indonesia'] |
| Sector of production | ['Electricity by coal'] |
| Region selling | ['China'] |
| Region of consumption | ['Germany'] |
| Sector of consumption | ['Machinery and equipment n.e.c.'] |

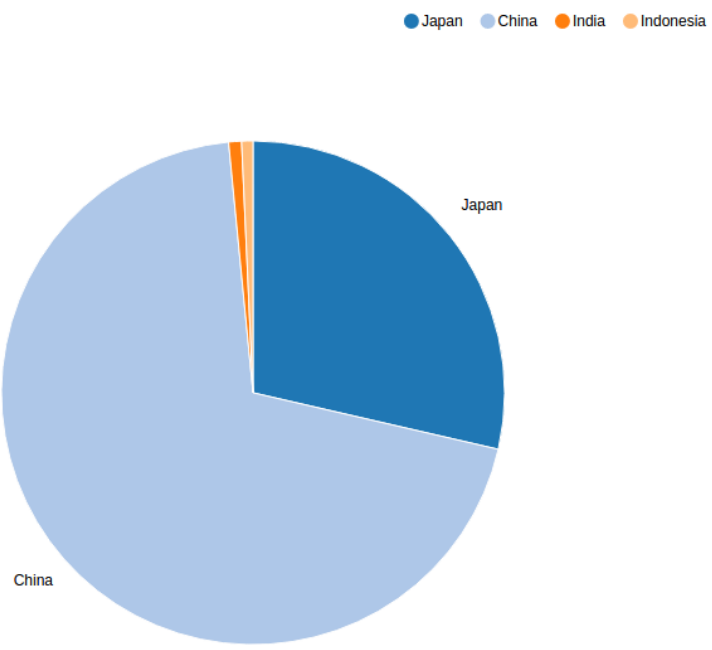



Figure 3.8: The generated piechart based on Q2.

As mentioned the graphic libraries are also interactive. Lets take the example of Q1, electricity by coal has by far the largest contribution and in 2$^{nd}$ place electricity by wind. Double-clicking on these two inputs in the legend above the graph will omit this contribution and a new piechart is made. See figure 3.9.

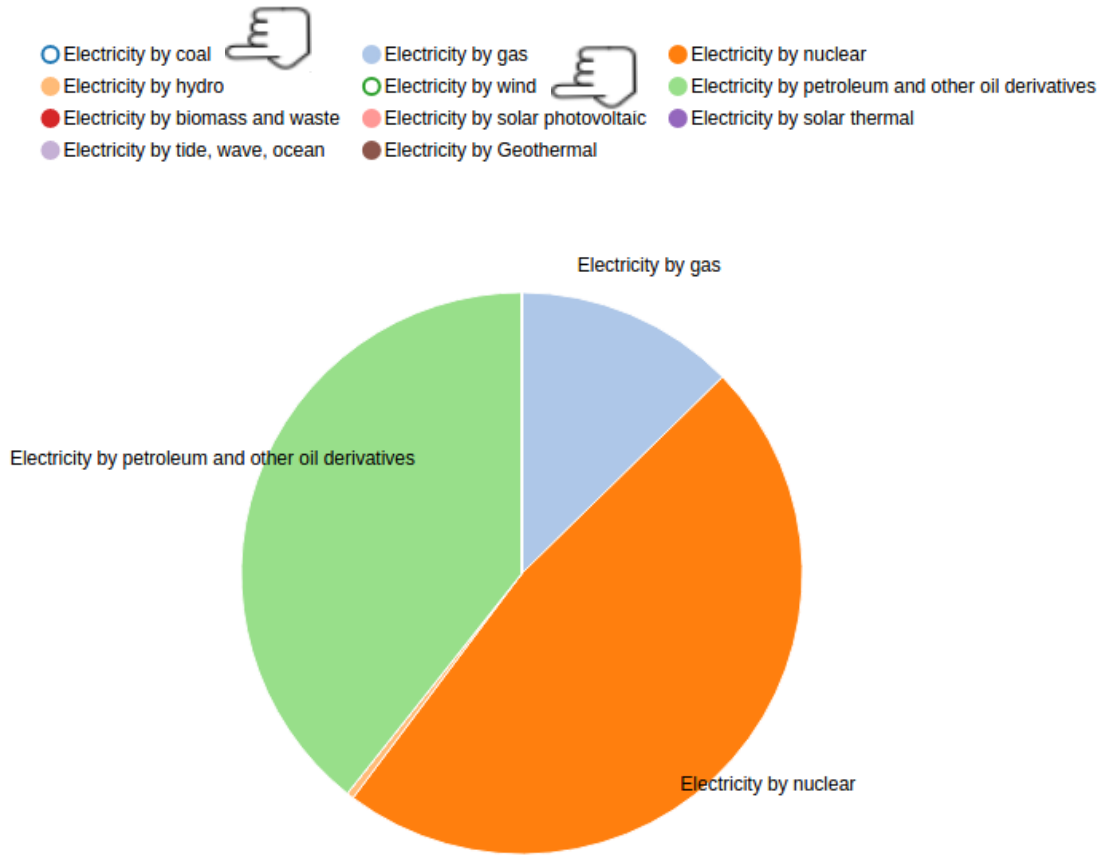



Figure 3.9: Interactively adjusted graph.

To recap, two research questions are used to generate visualizations. In “Mode” we made two selections, dynamically changing the “Filter”. We selected our inputs/parameters and generated the graphs. For the PieChart other “Modes” are available that can answer different questions related to consumption in importing regions and sectors. In section 3.3 a last example will be given for another visualization type the TimeSeries. In section 3.4 final remarks are made on the functionalities and future implementations.

## 3.3 TimeSeries

In the PieChart visualization type it is possible to select a decomposition in the dimension year. As discussed in Chapter 2 there are in total 17 years (1995-2011) in EXIOBASE. It makes more sense to visualize a dimension with respect to years in the TimeSeries visualization type than in a piechart. Therefore upon selecting in “Mode” the year, the user is automatically redirected to the TimeSeries tab. See figure 3.10.

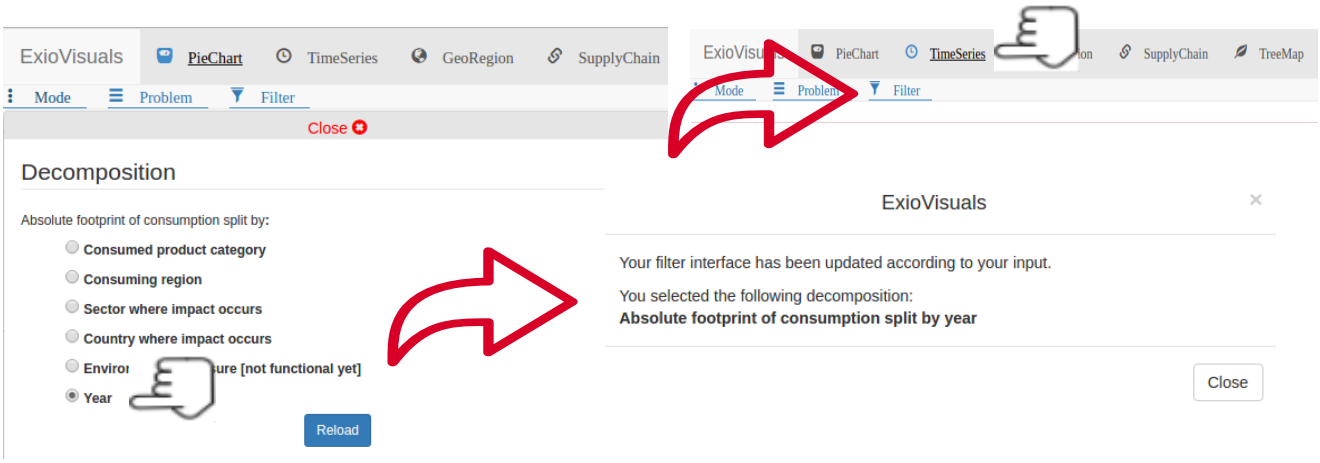



Figure 3.10: Automatic redirection from PieChart to TimeSeries if year is selected in mode.

Now that the redirection has taken place, the user can fill in the “Filter”. As discussed in section 3.2 the “Filter” is adjusted to the selection in “Mode”. In this case it is not different only that the user now resides in the visualization type TimeSeries and year is now open for multiple selections. The following research question is developed to walk through the TimeSeries type:

- What has been the changes in footprint for electricity by coal for the production of machinery and equipment in China over the last 17 years?

We construct the following table.

Table 3.2: TimeSeries example input

| | Q |
|---|---|
| Year | 1995 to 2011 |
| Region of production | China |
| Sector of production | Electricity by coal |
| Region selling | China |
| Product purchased | machinery and equipment |
| Region of consumption | China |

The year is now a setting that allows multiple selections and all other settings are restricted to one selection only. See image 3.11 below for a screen shot of the generated graphs according to these inputs.

Absolute footprint of consumption split by year

Plotted in kgCO2eq

| Type | Query |
|---|---|
| Select mode | Absolute footprint of consumption split by year |
| Environmental pressure | [] |
| Year | ['1995', '1996', '1997', '1998', '1999', '2000', '2001', '2002', '2003', '2004', '2005', '2006', '2007', '2008', '2009', '2010', '2011'] |
| Region of production | ['China'] |
| Sector of production | ['Electricity by coal'] |
| Region selling | ['China'] |
| Region of consumption | ['China'] |
| Sector of consumption | ['Machinery and equipment n.e.c.'] |

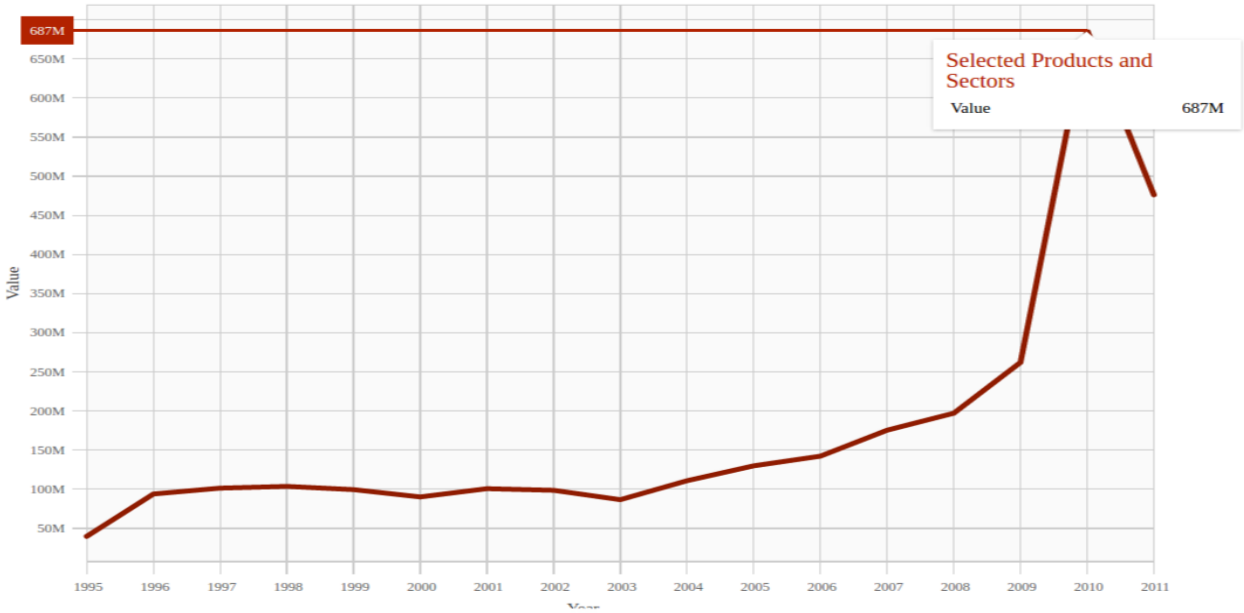



Figure 3.11: TimeSeries result. Each year is a selectable point, in this example 2010 has been chosen and a pop-up emerges showing additional information.

## 3.4 Future implementations and research

ExioVisuals development is still in progress, a possible important addition is the breakdown on environmental pressures and problems. More from an abstract level, additional research should be put into what is really important to visualize from EXIOBASE, what data and what dimensions. In turn the question should be answered on what are the most suitable graphs to generate from this data. Several functions need to be implemented on the short term such as implementing the functionality to select higher hierarchical levels for the EXIOBASE sectors and regions (NACE). Lastly in the future more recent data may be added for the web application. ExioVisuals is a proof of concept that assists in the development of a simple to use platform for EIO visualizations without the requirement from users to have programming experience. In turn we hope to achieve through visualizations a better understanding of environmental footprints of complex macro-economics.

# 4 References

[1]: https://www.djangoproject.com/

[2]: http://www.xmlobjective.com/what-is-the-difference-between-xml-and-html/

[3]: http://d3plus.org/

[4]: https://github.com/areski/django-nvd3

[5]: https://www.mysql.com/

[6]: https://www.hdfgroup.org/HDF5/

[7]: https://www.linux.com/

[8]: www.apache.org

[9]: https://www.python.org/

[10]: https://www.jetbrains.com/pycharm/

[11]: http://www.w3schools.com/xml/,

[12]: http://d3plus.org/

[13]: https://github.com/areski/django-nvd3

[14]: https://pypi.python.org/pypi/django-leaflet

[15]: https://jquery.com/

[16]: http://www.w3schools.com/js/

[17]: http://getbootstrap.com/

[18]: https://www.fullstackpython.com/wsgi-servers.html

[19]: http://www.EXIOBASE.eu/

[20]: https://github.com/mar10/fancytree/wiki

[21]: https://pypi.python.org/pypi/memory_profiler

# Appendix 1: System requirements

| Software | Version | Documentation |
|---|---|---|
| Django ** | 1.9.2 | Home laptop/desktop for development: https://www.digitalocean.com/community/tutorials/how-to-install-the-django-web-framework-on-ubuntu-14-04<br><br>Ubuntu with Apache server: https://www.digitalocean.com/community/tutorials/how-to-serve-django-applications-with-apache-and-mod_wsgi-on-ubuntu-14-04 |
| Apache** | 2.x | *https://httpd.apache.org* |
| Python** | 3.4.3 | *https://www.**python**.org* |
| django-nvd3** | 0.9.7 | https://github.com/areski/django-nvd3 |
| django-bower** | 5.1.0 | https://github.com/areski/django-nvd3 |
| Python-nvd3** | 0.14.2 | https://github.com/areski/python-nvd3 |
| Twitter bootstrap* | 3.3.6 | http://getbootstrap.com/ |
| Jquery* | 2.2.1 or 2.1.1 | https://jquery.com/ |
| South** | 1.0.2 | https://south.readthedocs.org/en/latest/ |
| django-leaflet** | 0.18.0 | https://pypi.python.org/pypi/django-leaflet |
| django-geojson** | 2.9.0 | https://pypi.python.org/pypi/django-geojson |
| JsonField** | 1.3.0 | Check geojson/leaflet documentations |
| mySQL | 5.5.47 | https://www.digitalocean.com/community/tutorials/how-to-use-mysql-or-mariadb-with-your-django-application-on-ubuntu-14-04 |
| D3, D3plus | v2 | *https://d3js.org/* & *http://bl.ocks.org/cfergus/3921009*<br>*http://d3plus.org/* |
| Django-mptt | | Install with pip |
| Django-extensions | | Install with pip |
| django-jquery-ui | | Install with pip |
| django-bootstrap-form | | Install with pip |
| Pillow | | Install with pip |
| H5py | 2.6.0 | http://www.h5py.org/ |

*twitter bootstrap and Jquery are called from the online repository and used inline in the HTML's.

** All to be installed on production server (either system-wide or advisably in a virtual environment)

# Appendix 2: Installation of Django on server and Apache configuration

Installation of Django on server and setting up a virtual environment based on the following documentation:

https://www.digitalocean.com/community/tutorials/how-to-serve-django-applications-with-apache-and-mod_wsgi-on-ubuntu-14-04

The basic idea is that you need mod_wsgi as the interface that connects python code with the Apache server. On the server this file: `/etc/apache2/sites-available/000-default.conf` is the most important for configuration linking django with Apache. Below how the 000-default.conf file should look like. Virtualhost is the port on which you run the app, all the "auth" is just .htaccess configuration, however WSGIscript alias is important! Because that is where it is accessible so <ipadress>/<alias> will run the django app, the rest such as <directory> just tells Apache where to find the files:

```
<VirtualHost *:80>
        # The ServerName directive sets the request scheme, hostname and port that
        # the server uses to identify itself. This is used when creating
        # redirection URLs. In the context of virtual hosts, the ServerName
        # specifies what hostname must appear in the request's Host: header to
        # match this virtual host. For the default virtual host (this file) this
        # value is not decisive as it is used as a last resort host regardless.
        # However, you must set it for any further virtual host explicitly.
        # ServerName

        ServerAdmin webmaster@localhost
        DocumentRoot /home/www

        # Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
        # error, crit, alert, emerg.
        # It is also possible to configure the loglevel for particular
        # modules, e.g.
        #LogLevel info ssl:warn

        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined

        # For most configuration files from conf-available/, which are
        # enabled or disabled at a global level, it is possible to
        # include a line for only one particular virtual host. For example the
        # following line enables the CGI configuration for this host only
        # after it has been globally disabled with "a2disconf".
        #Include conf-available/serve-cgi-bin.conf

#Password protection for the IElab Wiki

<Directory "/home/www/dokuwiki">
                AuthType Basic
                AuthName "Restricted Content"
                AuthUserFile /etc/apache2/.htpasswd
                Require valid-user
        </Directory>

        Alias /static /home/sidney/IEMasterProject/static/assets
        <Directory /home/sidney/IEMasterProject/static/assets>
                AuthType Basic
                AuthName "Restricted Content"
                AuthUserFile /etc/apache2/.htpasswd
```

```
                Require valid-user
        </Directory>
        <Directory /home/sidney/IEMasterProject/IEMasterProject>
                AuthType Basic
                AuthName "Restricted Content"
                AuthUserFile /etc/apache2/.htpasswd
                Require valid-user
                <Files wsgi.py>
                        AuthType Basic
                        AuthName "Restricted Content"
                        AuthUserFile /etc/apache2/.htpasswd
                         Require valid-user

               </Files>
        </Directory>

        WSGIDaemonProcess IEMasterProject python-
path=/home/sidney/IEMasterProject:/home/sidney/Env/IEMasterproject/lib/python3.4/site-packages
        WSGIProcessGroup IEMasterProject
        WSGIScriptAlias /softLab /home/sidney/IEMasterProject/IEMasterProject/wsgi.py
        # WSGIApplicationGroup %{GLOBAL}


</VirtualHost>

# vim: syntax=apache ts=4 sw=4 sts=4 sr noet
```

Below commands that can be used:

```
#adjusting apache configuration:
sudo nano /etc/apache2/sites-available/000-default.conf
#updating the configuration
sudo a2ensite 000-default.conf
#restart the server
sudo service apache2 restart
```

# Installation of prerequisites from sources

Depending on where you would install it ether system-wide or virtual (advised), here are some examples of pip3 installations:

```
$pip3 install python-nvd3
$pip3 install --upgrade python-nvd3
$pip3 install django-bower
$pip3 install django-leaflet
$pip3 install django-nvd3
$pip3 install django_nvd3
$pip3 install "django-geojson [field]"
$pip3 install south
```

# Installation of prerequisites Quick and Dirty

```
# go to the master project of your Django development project (where manage.py exists)
$pip3 freeze > requirements.txt
#to install the requirements do
$pip3 install requirements.txt
```

## Updating Django on server

Adjusting things in urls.py, views.py or settings.py will not be updated automatically. A graceful restart of Apache is needed (see section on configuring Apache). Adjusting things in templates can be done readily without a restart of Apache.

## Uploading and downloading of files over the server

GitHub can be used to upload your project to “the cloud”. This can be handy for sharing your projects. But as well as for getting your project from development on your own computer to a production server. Be careful with updating folders as you may loose data either local or cloud. Below some commands that can be used (make sure you downloaded GitHub):

```
#follow these commands ones you have Django project
#make sure you are outside of the django project folder by one step.
#create a local repository from a folder
$git init
#clone local repository to cloud
$git clone username@host:/path/to/repository
#propose a change
$git add <yourproject>
#initialize the change
$git commit -m "Commit message"
#upload to cloud
$git push -u origin master
#download from cloud
$git clone https://github.com/<username>/<repository>
```

Next to GitHub the Secure Copy Protocal (SCP) may be used to upload and download files to the server. Below some commands for sending and retrieving files.

```
#Sending:
sudo scp -oProxyCommand="ssh -W %h:%p s1650548@sshgw.leidenuniv.nl"
<path_to_file_to_be_send>  <user>@ronin.liacs.nl:<folder>

#Retrieving:
sudo scp -oProxyCommand="ssh -W %h:%p s1650548@sshgw.leidenuniv.nl"
<user>@ronin.liacs.nl:<path_to_file_to_be_send> ./

Retrieving folders:
scp with added -r parameter
```

# Appendix 3: CSV files used for constructing user input options

Below a snapshot of the top part of the 200 sectors with additionally the NACE higher hierarchical levels.

| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| 1 | **Name** | **Code** | **Id** | **Parent_Id** | **Position (loc** | **Level** | **Parent (local** |
| 2 | Total | total | 1 | NULL | 1 | 0 | 0 |
| 3 | Agriculture | hunting and forestry | A | 2 | 1 | 1 | 1 |
| 4 | Fishing | B | 3 | 1 | 2 | 1 | 1 |
| 5 | Mining and quarrying | C | 4 | 1 | 3 | 1 | 1 |
| 6 | Manufacturing | D | 5 | 1 | 4 | 1 | 1 |
| 7 | Electricity | gas and water supply | E | 6 | 1 | 5 | 1 |
| 8 | Construction | F | 7 | 1 | 6 | 1 | 1 |
| 9 | Trade | G | 8 | 1 | 7 | 1 | 1 |
| 10 | Hotels and restaurants | H | 9 | 1 | 8 | 1 | 1 |
| 11 | Transport | storage and communication | I | 10 | 1 | 9 | 1 |
| 12 | Financial intermediation | J | 11 | 1 | 10 | 1 | 1 |
| 13 | Real estate | renting and business activities | K | 12 | 1 | 11 | 1 |
| 14 | Public administration | L | 13 | 1 | 12 | 1 | 1 |
| 15 | Education | M | 14 | 1 | 13 | 1 | 1 |
| 16 | Health and social work | N | 15 | 1 | 14 | 1 | 1 |
| 17 | Other service activities | OPQ | 16 | 1 | 15 | 1 | 1 |
| 18 | Agriculture | hunting and related service activities | p01 | 17 | 2 | 1 | 2 |
| 19 | Forestry | logging and related service activities | p02 | 18 | 2 | 2 | 2 |
| 20 | Fishing | operating of fish hatcheries and fish farms; se | p05 | 19 | 3 | 3 | 2 |
| 21 | Mining of coal and lignite; extraction of peat | p10 | 20 | 4 | 4 | 2 | 3 |
| 22 | Extraction of crude petroleum and natural gas; service activities incidental to o | p11 | 21 | 4 | 5 | 2 | 3 |
| 23 | Mining of uranium and thorium ores | p12 | 22 | 4 | 6 | 2 | 3 |
| 24 | Mining of metal ores | p13 | 23 | 4 | 7 | 2 | 3 |
| 25 | Other mining and quarrying | p14 | 24 | 4 | 8 | 2 | 3 |
| 26 | Manufacture of food products and beverages | p15 | 25 | 5 | 9 | 2 | 4 |
| 27 | Manufacture of tobacco products | p16 | 26 | 5 | 10 | 2 | 4 |
| 28 | Manufacture of textiles | p17 | 27 | 5 | 11 | 2 | 4 |
| 29 | Manufacture of wearing apparel; dressing and dyeing of fur | p18 | 28 | 5 | 12 | 2 | 4 |
| 30 | Tanning and dressing of leather; manufacture of luggage | handbags | saddlery | harness and footwea | p19 | 29 | 5 |
| 31 | Manufacture of wood and of products of wood and cork | except furniture; manufacture of articles of str | p20 | 30 | 5 | 14 | 2 |
| 32 | Manufacture of pulp | paper and paper products | p21 | 31 | 5 | 15 | 2 |
| 33 | Publishing | printing and reproduction of recorded media | p22 | 32 | 5 | 16 | 2 |
| 34 | Manufacture of coke | refined petroleum products and nuclear fuels | p23 | 33 | 5 | 17 | 2 |
| 35 | Manufacture of chemicals and chemical products | p24 | 34 | 5 | 18 | 2 | 4 |
| 36 | Manufacture of rubber and plastic products | p25 | 35 | 5 | 19 | 2 | 4 |
| 37 | Manufacture of other non-metallic mineral products | p26 | 36 | 5 | 20 | 2 | 4 |
| 38 | Manufacture of basic metals | p27 | 37 | 5 | 21 | 2 | 4 |
| 39 | Manufacture of fabricated metal products | except machinery and equipment | p28 | 38 | 5 | 22 | 2 |
| 40 | Manufacture of machinery and equipment n.e.c. | p29 | 39 | 5 | 23 | 2 | 4 |
| 41 | Manufacture of office machinery and computers | p30 | 40 | 5 | 24 | 2 | 4 |
| 42 | Manufacture of electrical machinery and apparatus n.e.c. | p31 | 41 | 5 | 25 | 2 | 4 |
| 43 | Manufacture of radio | television and communication equipment and | p32 | 42 | 5 | 26 | 2 |
| 44 | Manufacture of medical | precision and optical instruments | watches and clocks | p33 | 43 | 5 | 27 |
| 45 | Manufacture of motor vehicles | trailers and semi-trailers | p34 | 44 | 5 | 28 | 2 |
| 46 | Manufacture of other transport equipment | p35 | 45 | 5 | 29 | 2 | 4 |
| 47 | Manufacture of furniture; manufacturing n.e.c. | p36 | 46 | 5 | 30 | 2 | 4 |

Below a snapshot of the top part of the 49 regions

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | Name | Code | Id | Parent_Id | Local | Level |
| 2 | Total | total | 1 | NULL | 0 | 0 |
| 3 | Europe | T_WE | 2 | 1 | 0 | 1 |
| 4 | Middle-East | T_WI | 3 | 1 | 1 | 1 |
| 5 | America | T_WA | 4 | 1 | 2 | 1 |
| 6 | Asia-Pacific | T_WA | 5 | 1 | 3 | 1 |
| 7 | Africa | WF | 6 | 1 | 4 | 1 |
| 8 | Austria | AT | 8 | 2 | 0 | 2 |
| 9 | Belgium | BE | 9 | 2 | 1 | 2 |
| 10 | Bulgaria | BG | 10 | 2 | 2 | 2 |
| 11 | Cyprus | CY | 11 | 3 | 3 | 2 |
| 12 | Czech Republic | CZ | 12 | 2 | 4 | 2 |
| 13 | Germany | DE | 13 | 2 | 5 | 2 |
| 14 | Denmark | DK | 14 | 2 | 6 | 2 |
| 15 | Estonia | EE | 15 | 2 | 7 | 2 |
| 16 | Spain | ES | 16 | 2 | 8 | 2 |
| 17 | Finland | FI | 17 | 2 | 9 | 2 |
| 18 | France | FR | 18 | 2 | 10 | 2 |
| 19 | Greece | GR | 19 | 2 | 11 | 2 |
| 20 | Croatia | HR | 20 | 2 | 12 | 2 |
| 21 | Hungary | HU | 21 | 2 | 13 | 2 |
| 22 | Ireland | IE | 22 | 2 | 14 | 2 |
| 23 | Italy | IT | 23 | 2 | 15 | 2 |
| 24 | Lithuania | LT | 24 | 2 | 16 | 2 |
| 25 | Luxembourg | LU | 25 | 2 | 17 | 2 |
| 26 | Latvia | LV | 26 | 2 | 18 | 2 |
| 27 | Malta | MT | 27 | 2 | 19 | 2 |
| 28 | Netherlands | NL | 28 | 2 | 20 | 2 |
| 29 | Poland | PL | 29 | 2 | 21 | 2 |
| 30 | Portugal | PT | 30 | 2 | 22 | 2 |
| 31 | Romania | RO | 31 | 2 | 23 | 2 |
| 32 | Sweden | SE | 32 | 2 | 24 | 2 |
| 33 | Slovenia | SI | 33 | 2 | 25 | 2 |
| 34 | Slovakia | SK | 34 | 2 | 26 | 2 |
| 35 | United Kingdom | GB | 35 | 2 | 27 | 2 |
| 36 | United States | US | 36 | 4 | 28 | 2 |
| 37 | Japan | JP | 37 | 5 | 29 | 2 |
| 38 | China | CN | 38 | 5 | 30 | 2 |
| 39 | Canada | CA | 39 | 4 | 31 | 2 |
| 40 | South Korea | KR | 40 | 5 | 32 | 2 |
| 41 | Brazil | BR | 41 | 4 | 33 | 2 |
| 42 | India | IN | 42 | 5 | 34 | 2 |
| 43 | Mexico | MX | 43 | 4 | 35 | 2 |
| 44 | Russia | RU | 44 | 2 | 36 | 2 |
| 45 | Australia | AU | 45 | 5 | 37 | 2 |
| 46 | Switzerland | CH | 46 | 2 | 38 | 2 |
| 47 | Turkey | TR | 47 | 3 | 39 | 2 |

# Appendix 4: Database setup and initialization

## Database setup

Developed by this documentation: https://www.digitalocean.com/community/tutorials/how-to-use-mysql-or-mariadb-with-your-django-application-on-ubuntu-14-04/. Relevant commands (not to be followed directly, follow the documentation!):

```
#install dependencies
$sudo apt-get install python-pip python-dev mysql-server libmysqlclient-dev

##Reinitilizing the database and administration
#clear the database (for instance when you changed something in the models.py)
python3 manage.py flush
#collect files such as static files from the apps and put them in a single location (may not be
that #useful depending on your configuration)
python3 manage.py collectstatic
#create the administrator of the database. This is really important to get into the administration
#page of the whole Django project.
python3 manage.py createsuperuser
#DATABASE CHMOD STUFF (PERMISSIONS)
chown www-data:www-data /home/username/<Django  project>
chown www-data:www-data /home/username/<database>

#migrate/update to remote mySQL
python3 manage.py makemigrations
python3 manage.py migrate
```

Commands used in mySQL:

```
#create database entry
CREATE DATABASE <databaseEntryName> CHARACTER SET UTF8;
#create user and password
CREATE USER <userName>@'%' IDENTIFIED BY '<password>';
#grant privileges
GRANT ALL PRIVILEGES ON <databaseEntryName>.* TO <userName>@'%';
#update privileges
FLUSH PRIVILEGES;
```

## Population of database

Population of database is not that straight forward. I will give an example in which I parsed an CSV file and then populate a database with a certain structure defined in the following models.py:

```
#make structure of ghgEmission database tables
class GhgEmissions(models.Model):
     #has a countryname
    label = models.CharField(max_length=128, unique=True)
    #has a countrycode
    code = models.CharField(max_length=128, unique=True)
    #has a absolute value
    absolute = models.FloatField()
    #mandatory
    def __unicode__(self):
        return self.label
```

Populate script:

```
import os
import sys
import django
#stuff for newer django !


#open the Exiobase file
def getfile():
    #open the file
    #*** > give the path to the file.
    f=open('/home/chai/Downloads/final_dem.csv', 'r')

    #get the content
    F=f.read()
    #split (make an array where each element is determined by an enter)
    U = F.split('\n')







    #Create empty list !!!!!!! THIS IS THE WORKING LIST OF LIST WE NEED FOR EVERYTHING !!
    L = []

    #fill the empty list with the data (this time split even further by tabs)
    for line in U:
        L.append(line.split('#'))
    #we only need to get co2 emissions and country names, so lets do that!
    useFul = L[:11]
    useFul.pop(0)
    useFul.pop(0)
    useFul.pop(0)
    useFul.pop(0)
    useFul.pop(0)
    useFul.pop(0)
    useFul.pop(0)
    useFul.pop(0)
    for x in useFul:
        x.pop(0)
        x.pop(0)
        x.pop(0)


    return useFul


#interface
listInList = getfile()



def populate():
        #below an example to fill in 1 entry
        #add_data("America", "US", 80.5)

        absDataFloat = []
        # get all data from parsed csv
        codeData = listInList[0]
        labelData = listInList[1]
        absData = listInList[2]

        codes = []
        lst2=[]
        # convert to get [[countryA,codeA,absA]...[..]]
        for i,x in enumerate(absData):
                lst2.append([item[i] for item in listInList])
        for x in lst2:
                #!!!!Now start adding the data iteratively
                add_data(x[1],x[0],x[2])
        print(codes)
        print('Done!')
```

```
#function that adds the data (offcourse)
def add_data(label, code, absolute):
    e, created = GhgEmissions.objects.get_or_create(label=label, code=code, absolute=absolute)

    return e


# Start execution here!
if __name__ == '__main__':
    print ("Starting IEMasterProject population script...")


    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'IEMasterProject.settings')
    django.setup()
    from ExioVisuals.models import GhgEmissions
    populate()
```

Now call the script within the masterproject (where manage.py resides) with the following command: $python3 exioPopulate.py

## Access  database from terminal

If you would like to access the database from outside of the Django application do the following:

```
#access mySQL;
mysql -u root -p

#show databases
show databases;

#use a specific one:
USE <databaseName>;

#to see all database tables:
SHOW tables;

#retrieve values from the database:
SELECT * FROM <tableName>;
```

## Remote access configuration for databases

The mysql configuration file needs to be adjusted with setting a bind-adress. Update the /etc/mysql/my.cnf file to contain this line (adjust or uncomment the bind-adress line):

```
bind-address            = <adress of server itself>
```

Restart the database: $sudo service mysql restart

In django setting.py file insert the following by adjusting the Databases line:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': '<databaseEntryName>',
        'USER': '<databaseUser>',
        'PASSWORD': '<UserPassword>',
        'HOST': '<hostIPaddress>',
        'PORT': '3306',
    }
}
```

## Updating Django on server

Adjusting things in urls.py, views.py or settings.py will not be updated automatically. A graceful restart of Apache is needed (see section on configuring Apache). Adjusting things in templates can be done readily without a restart of Apache.

# Appendix 5: populate scripts for EXIOBASE regions and sectors and the Django model

Populate script for regions:

```
import os
import sys
import django
import numpy as np

from string import ascii_uppercase
from collections import OrderedDict

od = OrderedDict((ch, idx) for idx, ch in enumerate(ascii_uppercase, 2))

#stuff for newer django !
#open the Exiobase file
def getfile():
    #open the file
    #*** > give the path to the file.

    #************************************NACE
    #OPEN NACE DATASET FOR SECTION DATA
    f=open('data/countryTree_exiovisuals.csv', 'r')


    #get the content
    F=f.read()

    #split (make an array where each element is determined by an enter)
    U = F.split('\n')







    #Create empty list !!!!!!! THIS IS THE WORKING LIST OF LIST WE NEED FOR EVERYTHING !!
    data = []


    #fill the empty list with the data (this time split even further by tabs)
    for line in U:
        data.append(line.split(','))






    return data



    #***generate tree for products
```

```
def populate(data):


    print("***Adding products***")
    lwst_level = False
    #add total parent
    add_parent(data[0][2], data[0][0], data[0][1], data[0][4], data[0][5], lwst_level)
    #remove total from data
    data.pop(0)


    #add children now
    for x in data:
        try:
            parent_id = Country.objects.get(id=x[3])
            id = x[2]

            name = x[0]
            print(name)
            code = x[1]
            local = x[4]
            level=  x[5]
            #the lowest level is 2 in this csv file
            if int(level) == 2:
                lwst_level = True
            #print(lws_level)
            add_child(parent_id, id, name, code, local, level, lwst_level)

        except:
            pass


    #go through the data
    print('***Done!! All countries have been added***')

#function that adds the data (offcourse)
def add_child(parent, id, name, slug, local, lvl, lwst_level):
    e, created = Country.objects.get_or_create(parent=parent, id=id, name=name, slug=slug,
local=local, lvl=lvl, lwst_level=lwst_level)

    return e

def add_parent(id,name, slug, local, lvl, lwst_level):
    e, created = Country.objects.get_or_create(id=id,name=name, slug=slug, local=local, lvl=lvl,
lwst_level=lwst_level)

    return e



# Start execution here!
if __name__ == '__main__':
    print ("Starting IEMasterProject population script...")


    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'IEMasterProject.settings')
    django.setup()
    from ExioVisuals.models import Country
    print("***Opening file****")
    #open the file
    data = getfile()
    #remove first entry
    data.pop(0)
    print("***Header removed****")
    #invoke function to populate the database
    populate(data)
```

Populate script for sectors:

```
import os
import sys
```

```
import django
import numpy as np




from string import ascii_uppercase
from collections import OrderedDict

od = OrderedDict((ch, idx) for idx, ch in enumerate(ascii_uppercase, 2))

#stuff for newer django !
#open the Exiobase file
def getfile():
    #open the file
    #*** > give the path to the file.

    #************************************NACE
    #OPEN NACE DATASET FOR SECTION DATA
    f=open('data/productTree_exiovisuals.csv', 'r')


    #get the content
    F=f.read()

    #split (make an array where each element is determined by an enter)
    U = F.split('\n')







    #Create empty list !!!!!!! THIS IS THE WORKING LIST OF LIST WE NEED FOR EVERYTHING !!
    data = []


    #fill the empty list with the data (this time split even further by tabs)
    for line in U:
        data.append(line.split('#'))






    return data




    #***generate tree for products
def populate(data):


    print("***Adding products***")
    lwst_level = False
    #add total parent
    add_parent(data[0][2], data[0][0], data[0][1], data[0][4], data[0][5], lwst_level)
    #remove total from data
    data.pop(0)


    #add children now
    for x in data:
        try:
            parent_id = Product.objects.get(id=x[3])
            id = x[2]

            name = x[0]
            print(name)
```

```
            code = x[1]
            local = x[4]
            level=  x[5]
            #the lowest level is 2 in this csv file
            if int(level) == 3:
                lwst_level = True

            add_child(parent_id, id, name, code, local, level, lwst_level)

        except:
            pass


    #go through the data
    print('***Done!! All products have been added***')

#function that adds the data (offcourse)
def add_child(parent, id, name, slug, local, lvl,lwst_level):
    e, created = Product.objects.get_or_create(parent=parent, id=id, name=name, slug=slug,
local=local, lvl=lvl,lwst_level=lwst_level)

    return e

def add_parent(id,name, slug,  local, lvl,lwst_level):
    e, created = Product.objects.get_or_create(id=id,name=name, slug=slug, local=local,
lvl=lvl,lwst_level=lwst_level)

    return e



# Start execution here!
if __name__ == '__main__':
    print ("Starting IEMasterProject population script...")


    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'IEMasterProject.settings')
    django.setup()
    from ExioVisuals.models import Product
    print("***Opening file****")
    #open the file
    data = getfile()
    #remove first entry
    data.pop(0)
    print("***Header removed****")
    #invoke function to populate the database
    populate(data)
```

Django models.py (it includes additional code for other parts of ExioVisuals such as creating the FancyTrees out of the mySQL data):

```
from djgeojson.fields import PointField
from django.db import models
from djgeojson.fields import PolygonField
from mptt.models import MPTTModel, TreeForeignKey


# models.py
from djgeojson.fields import PointField
from django.db import models
from django.contrib.auth.models import User
from urllib.parse import urljoin

from django.db import models
from django.core.urlresolvers import reverse
from django_extensions.db.fields import AutoSlugField
import mptt

#from urlparse import urljoin
```

```
#make structure of ghgEmission database tables
class GhgEmissions(models.Model):
     #has a countryname
    label = models.CharField(max_length=128, unique=True)
    #has a countrycode
    code = models.CharField(max_length=128, unique=True)
    #has a absolute value
    absolute = models.FloatField()



    #mandatory
    def __unicode__(self):
        return self.label

class years(models.Model):
     #has a countryname
    years = models.CharField(max_length=128, unique=True)

    #mandatory
    def __unicode__(self):
        return self.years



class Product(models.Model):
    parent = models.ForeignKey('self',
                               null=True,
                               blank=True,
                               related_name='children')

    name = models.CharField(max_length=200)
    local = models.CharField(max_length=200, default="none")
    lvl = models.CharField(max_length=200, default="none")
    slug = models.CharField(max_length=100)
    url = models.TextField(editable=False)
    lwst_level = models.BooleanField(default=False)

    class Meta:
        verbose_name_plural = "FancyTreeProduct"
        unique_together = (("name", "slug", "parent"), )
        ordering = ("tree_id", "lft")

    def __str__(self):
        return self.url

    def save(self, force_insert=False, force_update=False, **kwargs):
        super(Product, self).save(
            force_insert=force_insert,
            force_update=force_update,
            **kwargs)
        self.update_url()

    def get_tree(self, *args):
        """
        Return the tree structure for this element
        """
        level_representation = "--"
        if self.level == 0:
            node = "| "
        else:
            node = "+ "
        _tree_structure = node + level_representation * self.level
        return _tree_structure
    get_tree.short_description = 'tree'

    def get_repr(self, *args):
        """
        Return the branch representation for this element
        """
        level_representation = "--"
        if self.level == 0:
```

```
            node = "| "
        else:
            node = "+ "
        _tree_structure = node + level_representation * self.level + ' ' + self.name
        return _tree_structure
    get_repr.short_description = 'representation'

    def tree_order(self):
        return str(self.tree_id) + str(self.lft)

    def update_url(self):
        """
        Updates the url for this Product and all children Categories.
        """
        url = urljoin(getattr(self.parent, 'url', '') + '/', self.slug)
        if url != self.url:
            self.url = url
            self.save()

            for child in self.get_children():
                child.update_url()



class Country(models.Model):
    parent = models.ForeignKey('self',
                               null=True,
                               blank=True,
                               related_name='children')

    name = models.CharField(max_length=200)
    local = models.CharField(max_length=200)
    lvl = models.CharField(max_length=200)

    slug = models.CharField(max_length=100)
    url = models.TextField(editable=False)
    lwst_level = models.BooleanField(default=False)

    class Meta:
        verbose_name_plural = "FancyTreeCountry"
        unique_together = (("name", "slug", "parent"), )
        ordering = ("tree_id", "lft")

    def __str__(self):
        return self.url

    def save(self, force_insert=False, force_update=False, **kwargs):
        super(Country, self).save(
            force_insert=force_insert,
            force_update=force_update,
            **kwargs)
        self.update_url()

    def get_tree(self, *args):
        """
        Return the tree structure for this element
        """
        level_representation = "--"
        if self.level == 0:
            node = "| "
        else:
            node = "+ "
        _tree_structure = node + level_representation * self.level
        return _tree_structure
    get_tree.short_description = 'tree'

    def get_repr(self, *args):
        """
        Return the branch representation for this element
        """
        level_representation = "--"
        if self.level == 0:
```

```
            node = "| "
        else:
            node = "+ "
        _tree_structure = node + level_representation * self.level + ' ' + self.name
        return _tree_structure
    get_repr.short_description = 'representation'

    def tree_order(self):
        return str(self.tree_id) + str(self.lft)

    def update_url(self):
        """
        Updates the url for this Country and all children Categories.
        """
        url = urljoin(getattr(self.parent, 'url', '') + '/', self.slug)
        if url != self.url:
            self.url = url
            self.save()

            for child in self.get_children():
                child.update_url()


class Substance(models.Model):
    parent = models.ForeignKey('self',
                               null=True,
                               blank=True,
                               related_name='children')

    name = models.CharField(max_length=200)

    slug = models.CharField(max_length=100)
    url = models.TextField(editable=False)


    class Meta:
        verbose_name_plural = "FancyTreeSubstance"
        unique_together = (("name", "slug", "parent"), )
        ordering = ("tree_id", "lft")

    def __str__(self):
        return self.url

    def save(self, force_insert=False, force_update=False, **kwargs):
        super(Substance, self).save(
            force_insert=force_insert,
            force_update=force_update,
            **kwargs)
        self.update_url()

    def get_tree(self, *args):
        """
        Return the tree structure for this element
        """
        level_representation = "--"
        if self.level == 0:
            node = "| "
        else:
            node = "+ "
        _tree_structure = node + level_representation * self.level
        return _tree_structure
    get_tree.short_description = 'tree'

    def get_repr(self, *args):
        """
        Return the branch representation for this element
        """
        level_representation = "--"
        if self.level == 0:
            node = "| "
        else:
            node = "+ "
```

```
        _tree_structure = node + level_representation * self.level + ' ' + self.name
        return _tree_structure
    get_repr.short_description = 'representation'

    def tree_order(self):
        return str(self.tree_id) + str(self.lft)

    def update_url(self):
        """
        Updates the url for this Country and all children Categories.
        """
        url = urljoin(getattr(self.parent, 'url', '') + '/', self.slug)
        if url != self.url:
            self.url = url
            self.save()

            for child in self.get_children():
                child.update_url()

mptt.register(Product, order_insertion_by=['name'])
mptt.register(Country, order_insertion_by=['name'])
mptt.register(Substance, order_insertion_by=['name'])


class Selection (models.Model):
    products = models.ManyToManyField(Product)

    def __str__(self):
        return self.name

    def get_absolute_url(self):
        return reverse('selection', args=[self.pk])


class Selection2 (models.Model):
    countries = models.ManyToManyField(Country)

    def __str__(self):
        return self.name

    def get_absolute_url(self):
        return reverse('selection', args=[self.pk])

class Selection3 (models.Model):
    substances = models.ManyToManyField(Substance)

    def __str__(self):
        return self.name

    def get_absolute_url(self):
        return reverse('selection', args=[self.pk])
```

# Appendix 6: HDF5 query result

```
[[  0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
    0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
    0.00000000e+00   0.00000000e+00]
 [  0.00000000e+00   3.09065680e+07   8.10149597e+02   3.70676855e+03
    3.00668518e+02   0.00000000e+00   0.00000000e+00   3.82752393e+03
    1.70379625e+05   1.18919462e+06]
 [  0.00000000e+00   1.50051880e+03   9.76299920e+07   1.33837207e+04
    1.21801733e+03   3.86304810e+02   0.00000000e+00   4.62124219e+03
    5.70674688e+05   5.10223600e+06]
 [  0.00000000e+00   3.61889008e+02   6.78491577e+02   1.87869968e+08
    3.39871429e+02   0.00000000e+00   0.00000000e+00   0.00000000e+00
    1.70656938e+05   4.97005219e+05]
 [  0.00000000e+00   2.18485229e+02   4.40763062e+02   1.88814673e+03
    2.89938260e+07   0.00000000e+00   0.00000000e+00   0.00000000e+00
    8.17705156e+04   2.63953844e+05]
 [  0.00000000e+00   1.49467987e+02   2.91416168e+02   1.35993384e+03
    0.00000000e+00   1.17978470e+07   0.00000000e+00   0.00000000e+00
    1.05082305e+05   3.15857219e+05]
 [  0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
    0.00000000e+00   0.00000000e+00   3.84935219e+05   0.00000000e+00
    2.55878784e+02   7.96578064e+02]
 [  0.00000000e+00   2.05908096e+02   4.23549286e+02   4.87790688e+05
    1.63413589e+02   0.00000000e+00   0.00000000e+00   5.45781950e+06
    7.90101875e+05   6.02262438e+05]
 [  0.00000000e+00   4.06189844e+04   8.69029844e+04   2.54700250e+05
    2.91821875e+04   1.31892568e+04   1.70570142e+03   5.48452148e+03
    2.73519008e+08   3.64109900e+06]
 [  0.00000000e+00   4.46351044e+02   8.19665283e+02   3.67326172e+03
    4.23433990e+02   0.00000000e+00   0.00000000e+00   0.00000000e+00
    1.19116320e+05   4.56007360e+07]]
```

# Appendix 7: HDF5 populate script

```
import numpy as np,time, scipy.io, h5py, os, psutil, resource, pdb, subprocess
from memory_profiler import profile
from threading import Thread
#metacode

@profile
def finished():
#start merging importing regions to one file called <year>_combined.hdf5
    ssst0 = time.time()
    year = np.arange(1995,2012)
    path = "/home/sidney/experiments/"
    for i, x in enumerate(year):
        for j in range(49):
            #yearStr = ' '.join(map(str,year))
            #print(i)
            #print(j)
            print(x)
            subprocess.call(['h5copy', '-i', path+str(x)+"_region"+str(j)+"_Footprint.hdf5",'-s',
'/Footprint', '-o', str(x)+'_combined.hdf5', '-d', '/region'+str(j)])

            #subprocess.call(['h5copy', '-i', '{0}','-s', '/Footprint', '-o', '{1}_combined.hdf5',
'-d', '/region{2}'.format(path+str(x$

            print("Success in combining files")
                #pdb.set_trace()
                #del(bDiagVector)
                #del(yDiagVector)

            #f.close()
        #remove the region filed, as at this point there is a combined file of them
        crop = 'rm -rf '+ path+str(x)+'_region*'
        print(crop)
        subprocess.call(str(crop), shell=True)

    ssst1 = time.time()
    ssstotal = ssst1-ssst0
    print("\ntotal time of merging files: "+str(ssstotal))
    #f.flush()

    #f.close()
    #log.close()


    print("finished")

def mult_diag(d, mtx, left=True):
    """Multiply a full matrix by a diagonal matrix.
    This function should always be faster than dot.

    Input:
      d -- 1D (N,) array (contains the diagonal elements)
      mtx -- 2D (N,N) array

    Output:
      mult_diag(d, mts, left=True) == dot(diag(d), mtx)
      mult_diag(d, mts, left=False) == dot(mtx, diag(d))
    """
    if left:
        return (d*mtx.T).T
    else:
        return d*mtx


#create hdf file
#create a new file and this object serves a the first starting point


    #group = f.create_group('a_group')
#f = h5py.File("/home/chai/data/built_exio.hdf5", "w")
```

```
@profile
def populate():
    #create log file

    #create year ranges and totals
    year =  [
                "1995",
                "1996",
            "1997",
            "1998",
            "1999",
            "2000",
            "2001",
            "2002",
            "2003",
            "2004",
            "2005",
            "2006",
            "2007",
            "2008",
             "2009",
            "2010",
            "2011",
                ];
    year = np.arange(1995,2012)
    #path = "/home/chai/data/"
    path = "/home/sidney/experiments/"
    #year = np.arange(1995,1996)
    #log = open('/home/sidney/experiments/properlog.txt', 'w')

    log = open(path+'v4_2log.txt', 'w')
    #create regions and sectors ranges
    #number of years
    nt = 17
    #number of regions
    nr = 49
    #number of sectors
    ns = 200
    #number of final demand categories
    nf = 1
    #households
    nh = 7
    #0.0000000000001 sort of to check for zero's
    accuracy = 1e-16

    #with h5py.File('/home/sidney/experiments/proper.hdf5','w') as f:
    #with h5py.File('/home/chai/data/v3built_exio.hdf5','w') as f:

    #create hdf dataset with dimension nt * nr * ns * nr * ns * nr for footprint of consumption
    #and a smaller one for emissions of final demand just with nt * ns

        #footprint_dset = f.create_dataset('Footprint', (nt, nr, ns, nr, ns, nr), dtype = 'f',
compression="gzip", compression_opts=9)



#total of two datasets

    #populate

    print("*** Starting script ***")
    log.write("*** Starting script ***\n")
    mem = resource.getrusage(resource.RUSAGE_SELF).ru_maxrss  / 1000
    print( "start of script the usage is (in MB):",mem)
    log.write("start of script the usage is (in MB):%s" % str(mem))
    #run loop for year
    sst0 = time.time()
    for i, x in enumerate(year):

    #read stuff from matlab binary
        print("*** Loading file of year: "+ str(x)+ " ***")
        log.write("\n*** Loading file of year: "+ str(x)+ " ***\n")
```

```
        t0 = time.time()

        mat = scipy.io.loadmat('/data/exiovisuals/sys_leo_%s.mat' % x) #%insert i in name
        #mat = scipy.io.loadmat('/data/exiovisuals/sys_leo_%s.mat' % x) #%insert i in name

        print("*** Retrieving arrays y,b,h,leo ***",
resource.getrusage(resource.RUSAGE_SELF).ru_maxrss  / 1000)
        log.write("*** Retrieving arrays y,b,h,leo ***\n")


        y = mat['io']['y'][0][0];
        leo = mat['io']['leo'][0][0];
        b = mat['io']['b'][0][0];
        h = mat['io']['h'][0][0];

    #THIS IS A FIX BECAUSE EMISSIONS FROM HOUSEHOLDS
    #ITERATE OVER FINAL DEMAND CATEGORIES!!!!
        h = np.reshape(h, (nr, nh))[:,0]

        h = h.reshape(1,nr)
        mem = resource.getrusage(resource.RUSAGE_SELF).ru_maxrss  / 1000
        print( "after retrieval of arrays the usage is (MB):",mem)
        log.write("after retrieval of arrays the usage is (MB):%s \n" % str(mem))

        t1 = time.time()
        total = t1-t0
        print("Loading time taken: "+str(total))
        log.write("Loading time taken: "+str(total)+ "\n")

        t0 = time.time()
    #populating the final demand emissions database
        print("*** Populating final demand emissions db of year: "+ str(x)+ " ***")
        log.write("*** Populating final demand emissions db of year: "+ str(x)+ " ***\n")

        f = h5py.File(path+'%s_finalDemand.hdf5' % x ,'w')
        finalDemand_dset = f.create_dataset('Final_demand', (nt,nr),dtype='f')
        #try to fill in the final demand dataset with at coordinate i the array
        finalDemand_dset[i,:] = h


        t1 = time.time()
        total = t1-t0
        print("Final demand population time taken: "+str(total))
        log.write("Final demand population time taken: "+str(total)+"\n")

        print("*** Populating footprint db of year: "+ str(x)+ " ***")
        log.write("*** Populating footprint db of year: "+ str(x)+ " ***\n")

        t0 = time.time()
    #iterating over import region


        def myfunc(j):



            mem = resource.getrusage(resource.RUSAGE_SELF).ru_maxrss  / 1000
            print( "before calculating the usage is (MB):",mem)
            log.write("before calculating the usage is (MB):%s \n" % str(mem))
            foot = mult_diag(y[:,j],leo, left=False)
            foot = mult_diag(b[0],foot, left=True)
            mem = resource.getrusage(resource.RUSAGE_SELF).ru_maxrss  / 1000
            print( "after calculating the usage is (MB):",mem)
            log.write("after calculating the usage is (MB):%s \n" % str(mem))
            vpercent = 99.9
            vtot = sum(sum(abs(foot)))
            vmax = np.max(foot)

            for k in range(50):
                foot_tmp1 = foot * (foot > (vmax * 10 ** (-1 * k)))
                vtmp = sum(sum(abs(foot_tmp1)))
```

```
                if (vtmp/vtot*100 > vpercent):
                    ktmp = k
                    break


            print("ktmp")
            log.write(str(ktmp)+ " "+ str(vtmp/vtot*100)+ " "+ str(100*vtmp / vtot))
            log.flush()
            print(ktmp)
            print(sum(sum(foot_tmp1)))
            foot = foot_tmp1

            foot_tmp = np.reshape(foot,[nr,ns,nr,ns])

            #do the multiplication of leontief times the y diagonal
            f2 = h5py.File(path+'{0}_region{1}_Footprint.hdf5'.format(x, j) ,'w')

            temperal = ('Footprint_dset_%s' % str(j))
            print(temperal)
            temperal = f2.create_dataset('Footprint', (nr, ns, nr, ns), dtype = 'f',
compression="gzip", compression_opts=9 )

            temperal[:,:,:,:] = foot_tmp

        #do multithreading but do not do al 49 threads at the same time
        for j in range(10):
            t = Thread(target=myfunc, args=(j,))
            t.start()
            t.join()
        for j in range(10):
            j = j+10
            t = Thread(target=myfunc, args=(j,))
            t.start()
            t.join()
        for j in range(10):
            j = j+20

            t = Thread(target=myfunc, args=(j,))
            t.start()
            t.join()

        for j in range(10):
            j = j+30

            t = Thread(target=myfunc, args=(j,))
            t.start()
            t.join()

        for j in range(9):
            j = j+40
            t = Thread(target=myfunc, args=(j,))
            t.start()
            t.join()

        #total time of calculating and putting into a file:
        sst1 = time.time()
        sstotal = sst1-sst0
        log.write("\ntotal time of calculating and putting into a file: "+str(sstotal))

        f.flush()
        log.flush()

        f.close()


populate()
finished()
```