

Data Knowledge Discovery and Data Mining



Professor: Prof. Dr. SCHOMMER Christoph

Dataset: Diabetes

Students and tasks:

- Guillaume BALLINGER - Data cleaning
- Filipe DA SILVA - Analyse
- Esada LICINA - Visualisation

Table of contents

1. Data Cleaning

- 1.1 Multi-level grouping**
- 1.2 Drop column**
- 1.3 Column concatenation + additional attribute**
- 1.4 Row filtering**
- 1.5 Data Imputation**
- 1.6 Listwise deletion**
- 1.7 Overall reduction**

2. Data Analyse

- 2.1 Diabetes type 1**
- 2.2 Diabetes type 2**

3. Data Visualisation

- 3.1 The number of hospital visits based on the medication taken**
- 3.2 Checking with association**
 - 3.2.1 Flask program**
 - 3.2.2 Selecting method**
 - a. PYCHARM Code**
 - b. HTML code**
 - c. JAVA script**
 - 3.2.3 Searching method**
 - a. PYCHARM Code**
 - b. HTML code**
 - c. JAVA script**
 - 3.2.4 Using the table**

Tasks:

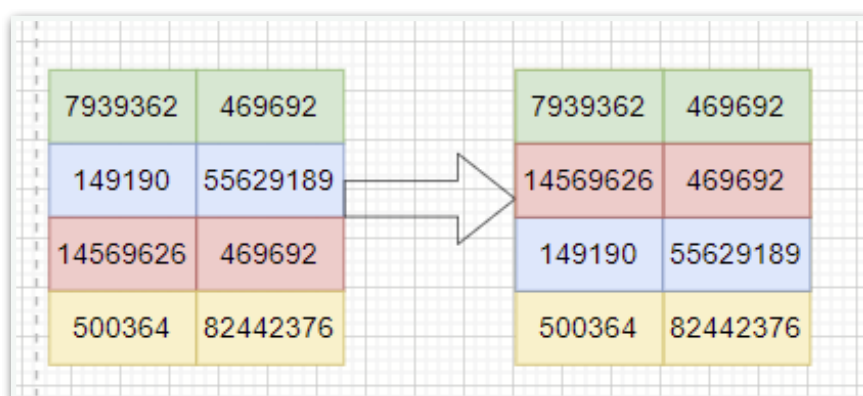
- There are several data sources available. Decide by yourself, which data you want to concern in your project (Question A).
- Download the data; decide and justify how you manage the data in your management system. (Question B).
- Prepare the data, for instance by inspecting + discussing each attribute by a visualisation, by repairing the data if needed, by adding additional attributes if beneficial, et cetera. Use the terminology and justify your decision (Question C).
- Analyse your data with an application of your choice: Association Discovery, Clustering, Classification, or another application (Question D).
- interpret and discuss the results (Question E) and proposed a resulting activity (Question F).

1. Data Cleaning

We took the dataset about diabetes. In order to make a meaningful analysis we had to clean and repair the data. We used pandas (python dataframe) to clean the data

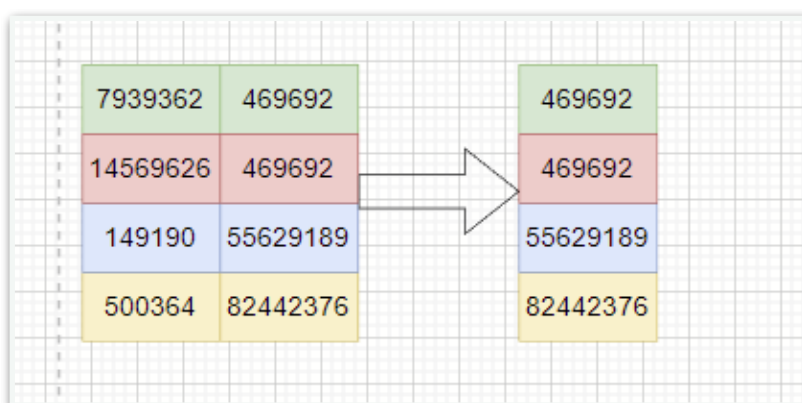
1.1 Multi-level grouping:

We noticed that the encounter ID(left) was based on the entrance time. While they didn't give an exact time they gave the order of entrance. Thus we first sorted by patient id(right) followed by their encounter id to have a chronological entrance.



1.2 Drop column:

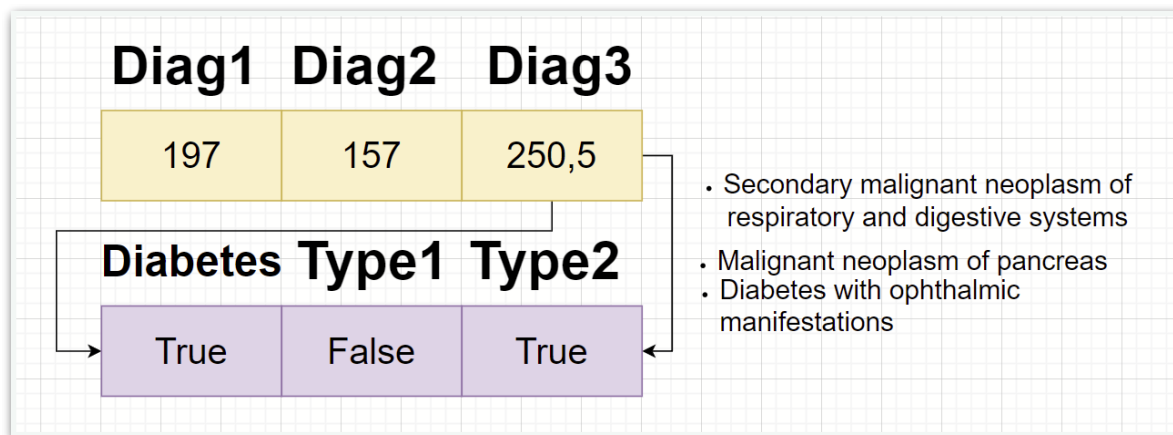
We removed all columns that were not relevant to our analysis: payer code, encounter id, time in hospital, number of lab procedures, outpatient, emergency, and diagnoses. A particular case is for citoglipton and examide which had only 1 single value inside. Which for the apriori algorithm would make it match 100% but would remain uninteresting in the relations.



1.3 Column concatenation + additional attribute:

Since we wanted to focus on diabetes only we searched through the diagnosis and searched the database to see which diabetes corresponded to type 1 or type 2. Everything that was not in the range of 249-251 would be excluded as these would be diagnoses of something else to diabetes. It could be related but no 100% guarantee. Thus we only kept

the range. We also created 3 new fields and dropped the diagnosis fields. Diabetes which is a boolean. Type 1 and Type 2. We dropped every patient that has never been diagnosed with diabetes. We kept the ones that had been diagnosed at least once since we could check for evolution.



1.4 Row filtering:

We dropped every row that contained IDs that were unknown. E.g admission source id 9 which is not available.

The diagram shows a table with two columns: 'Person ID' and 'Admission ID'. The table has four rows. The first three rows are highlighted in green, red, and blue respectively, and are shown in both the original and filtered versions. The fourth row, with Person ID 82442376 and Admission ID 9, is highlighted in yellow and is crossed out with a large 'X' in the original version, indicating it was filtered out. An arrow points from the original table to the filtered table, which contains only the first three rows.

Person ID	Admission ID
469692	1
469692	3
55629189	2
82442376	9

Person ID	Admission ID
469692	1
469692	3
55629189	2

1.5 Data Imputation:

The next step was to fill the empty fields in our dataset. Since we sorted the data we could easily repair it by correlation. The first one that we found was the race field. If it is missing and present in another field it is highly unlikely that the patient changed his race.

For the field specialist by using techniques in the statical analysis, we found out that every child (age [0-10]) was consulted by a Pediatrics. Thus this could also be filled.

The weight class was a little bit more special. Removing empty fields would have left us with less than 2000 entries which have rendered all of the machine-learning techniques useless (e.g: regression mean). Thus we opted to take the average weight by age. To

make sure that we would not have a bias later we added a field “weight_averaged” to reduce the weight of analysis towards this field.

The diagram illustrates listwise deletion. On the left, a table with columns 'Person ID' and 'Race' has three rows: (469692, Asian), (469692, ?), and (55629189, Caucasian). An arrow points to the right, where the same table is shown but the row with the missing value has been replaced by the row with the same 'Person ID' but a known 'Race' value. The resulting table has two rows: (469692, Asian) and (55629189, Caucasian).

Person ID	Race
469692	Asian
469692	?
55629189	Caucasian

Person ID	Race
469692	Asian
55629189	Caucasian

1.6 Listwise deletion:

Finally after all the repairs if one value was missing we removed the whole row.

The diagram illustrates overall reduction. On the left, a table with columns 'Person ID' and 'Race' has two rows: (469692, Asian) and (55629189, ?). An arrow points to the right, where the same table is shown but the row with the missing value has been removed. The resulting table has one row: (469692, Asian).

Person ID	Race
469692	Asian
55629189	?

Person ID	Admission ID
469692	Asian

1.7 Overall reduction:

We started with 101763 rows. After taking only diabetes cases into consideration we had 55440 rows. 56 out of 18464 weights could not be repaired and 46323 weights have been averaged. roughly 20k empty rows have been dropped which leaves us with a total of 40782 rows.

2. Data Analyse

When dealing with medical conditions, it is crucial to have high recall and precision accuracy to avoid false diagnoses. In this case, the recall for diabetes type 1 is quite low at only 12%. This means that out of all the actual cases of diabetes type 1, only 12% were correctly identified by the model. I will ref why in the below report.

2.1 Diabetes type 1 - Recall looks very low with 12% - it is linked to True Negative

Classification	Positive diagnose	Negative diagnose	Precision:	68%
Positive diagnose	2663	70	Recall:	12%
Negative diagnose	1243	20385	Accuracy:	95%

2.2 Diabetes type 2

Classification	Positive diagnose	Negative diagnose	Precision:	97%
Positive diagnose	14835	3213	Recall:	82%
Negative diagnose	385	3369	Accuracy:	83%

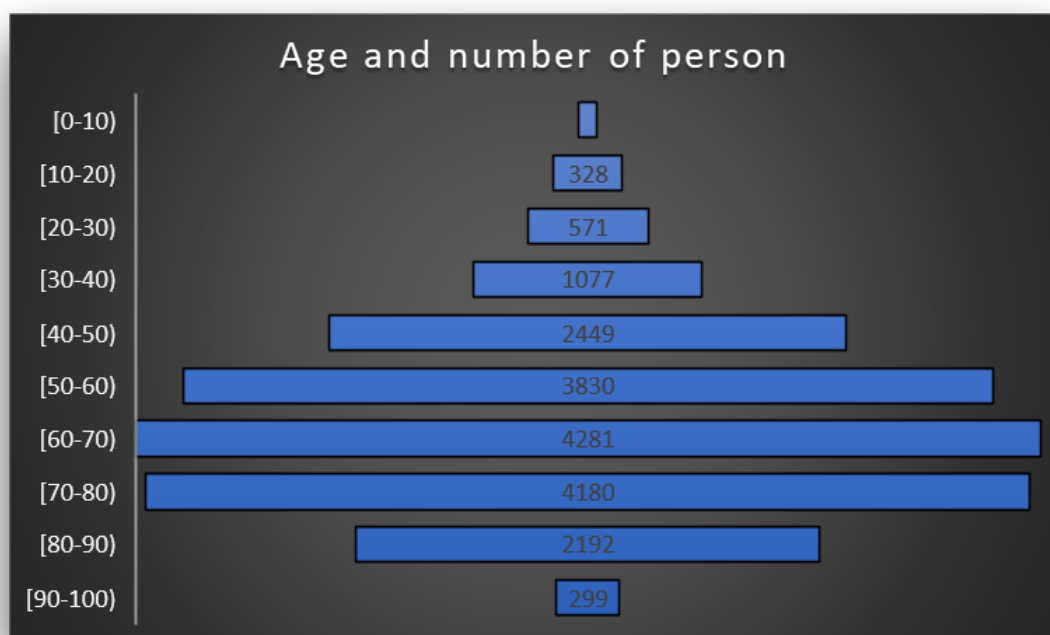
The precision rate of the model is 97%, indicating that for all the positive diagnoses made by the model, 97% were accurate. Moreover, the recall rate of the model is 82%, meaning that 82% of all the actual positive cases were correctly identified.

The precision and recall rates of the model are both high for the type 2, indicating its effectiveness in accurately diagnosing patients. The high precision rate suggests that when the model predicts positive for a patient for type 2 diabetes, there is a high probability that the patient has the condition. The high recall rate implies that the model is successful in detecting most of the positive cases, reducing the chances of false negatives.

In this case it is explained by The given data indicates that among the 40,782 tests conducted, 17,498 positive results were observed. Out of these positive results, 85% were classified as type 2 diabetes, while only 15% were identified as type 1 diabetes. This suggests that it is more likely for an individual to have type 2 diabetes than type 1 diabetes and age is an important factor.

Type 2 diabetes is more common than type 1 diabetes. It is usually diagnosed in adults and is characterized by insulin resistance, where the body does not use insulin effectively, leading to high blood sugar levels. On the other hand, type 1 diabetes is an autoimmune condition that usually affects children and young adults, where the immune system attacks and destroys insulin-producing cells in the pancreas.

The high prevalence of type 2 diabetes emphasizes the importance of adopting a healthy lifestyle, as seen in the data the type 2 is increasing with age, in the the data provided indicates that type 2 diabetes is more common than type 1 diabetes. Being aware of the risk factors and adopting a healthy lifestyle can help prevent the onset of type 2 diabetes and reduce the risk of complications associated with the disease. As indicated in the data set if you take the medication it is less that you get readmitted.



On the other hand, the precision accuracy of 68% means that out of all the positive diagnoses, 68% were correct. While this precision rate is not ideal, it is still much better than the recall rate. A high precision rate indicates that when the model diagnoses someone with diabetes type 1, there is a relatively high chance that the diagnosis is correct. Which is shown in the data that by testing as age progress you have a greater chance to get type 2 diabetes.

Given data represents the number of hospital visits based on the medication taken by the patients. The data is divided into different categories based on the percentage of adherence to medication, labeled as less <30 days, over >30 days, and NO. The data is further classified into ten intervals, each with a range of 10 years of adherence.

The results indicate that patients who adhere to medication have a significantly lower number of hospital visits than those who do not.

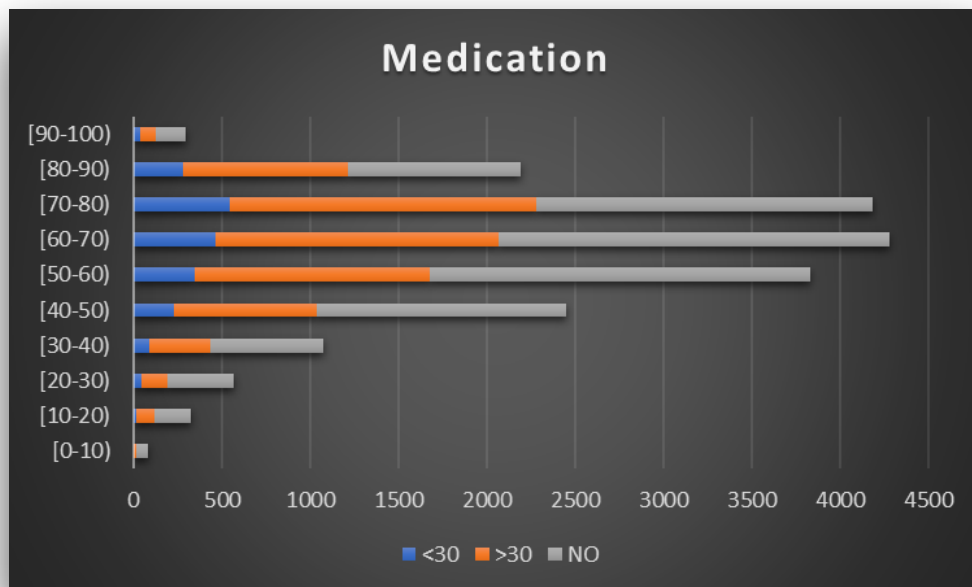
Notably, the data shows a clear trend of the number of hospital visits decreasing as the adherence levels to medication increases. The data further highlights that patients who had adherence levels between 50-90 years had a higher number of hospital visits than those who had adherence levels between 20-50 years.

Étiquettes de lignes	[0-10)	[10-20)	[20-30)	[30-40)	[40-50)	[50-60)	[60-70)	[70-80)	[80-90)	[90-100)
<30	1	20	49	89	228	346	465	546	281	38
>30	19	103	147	349	814	1332	1599	1731	936	90
NO	63	205	375	639	1407	2152	2217	1903	975	171

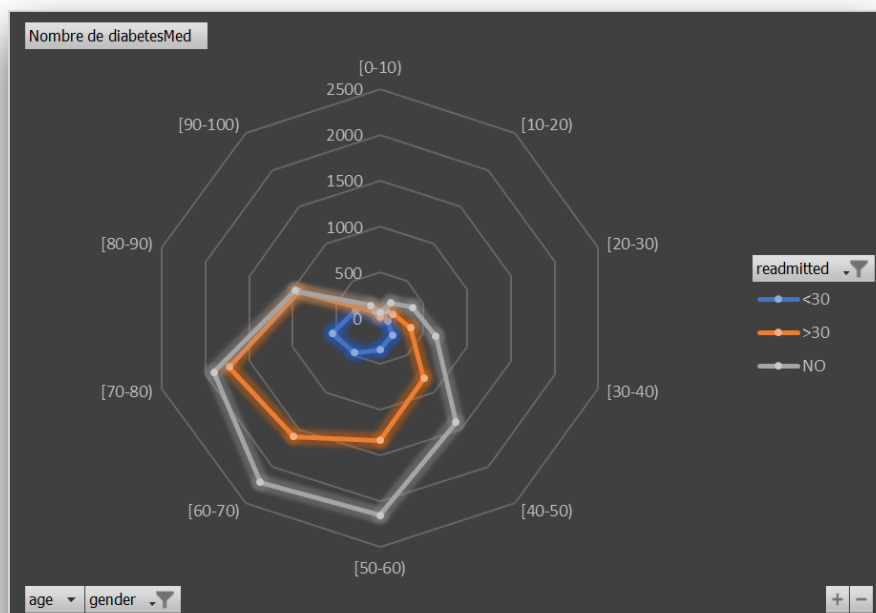
3. Data Visualisation

Several visuals were made and shown in the Analysis section using an Excel program. The report will now cover a bit of the medication visualisation and the association table, which was the most challenging and interesting section.

3.1 The number of hospital visits based on the medication taken



The below graph needs to be read clockwise. Outside of the graph is the age as the age increases as the number of hospitalizations are increasing. The data represents the number of hospital visits based on the medication taken.



In conclusion, the given data indicates that medication adherence has a significant impact on the number of hospital visits. Patients who adhere to medication have a lower number of hospital visits than those who do not. It is therefore crucial for healthcare providers to encourage patients to adhere to medication, which could result in better health outcomes and reduced healthcare costs.

3.2 Checking with association

We process a brute force and below are the examples of the results of association rule mining between the type of diabetes (type 1 or type 2), weight, and weight averaged for patients. The data includes the attributes (type of diabetes and weight), the attribute (weight and weight averaged), support (proportion of transactions containing both), confidence (the likelihood of the appearing in transactions containing), lift (the ratio of the observed support to what would be expected if the type and were independent), and frequency (the proportion of transactions containing the weights).

	A	B	C	D	E	F	G	H	I
1	lhs_attrit	lhs_value	rhs_attrit	rhs_value	support	confiden	lift	frequency	
70020	Type1	TRUE	weight	[50-75)	0.08437545	0.71942295	0.8835071971805709	0.10361960973259456	
70021	Type1	TRUE	weight	[25-50)	0.00573782	0.04892326	0.6520224815830570	0.7647058823529411	
70022	Type1	TRUE	weight	[75-100)	0.02479034	0.21137361	0.91034397266903600	0.1411419796174787	
70023	Type1	TRUE	weight	[0-25)	0.00237850	0.02028015	0.787681371524148	0.9238095238095239	
70590	Type2	TRUE	weight	[50-75)	0.48126624	0.80829420	0.9926479902461423	0.5910322813779813	
70591	Type2	TRUE	weight	[25-50)	0.00115246	0.00193559	0.25796482830015294	0.15359477124183007	
70592	Type2	TRUE	weight	[75-100)	0.11281938	0.18948192	0.788010180946200	0.6423286332542231	
70593	Type2	TRUE	weight	[0-25)	0.00017164	0.00028827	0.11196771270900255	0.06666666666666667	
71714									
71715									

3.2.1 Flask program

This program's goal is to utilise Python to construct a clear and structured table on the Flask server, allowing users to search for and choose particular values in each column. This allows for a more thorough knowledge of the dataset's linkages.

Python's Flask library contains important features such as the DataTable, which simplifies the coding while still providing necessary table display capabilities. Additional information, including code snippets and explanations, may be obtained on the DataTable website: <https://www.datatables.net/>.

Our Table Result:

Show <input type="text" value="10"/> entries							
Chose left attribute:	Filter lhs Values:	Chose right attribute:	Filter rhs Values:	Filter Support:	Filter Confidence:	Filter Lift:	Filter Frequency:
None		None					
A1Cresult	Norm	admission_type_id	2	0.009710166249816	0.2163934426229508	1.1598051487776555	0.0520436325404126
A1Cresult	>7	num_medications	15	0.0015202785542641	0.0451565914056809	0.8232347388048646	0.02771569065713
A1Cresult	>7	num_medications	16	0.0018880878819086	0.056081573197378	1.0662558126505688	0.0358974358974358
A1Cresult	>7	num_medications	17	0.0015202785542641	0.0451565914056809	0.987440273837256	0.0332439678284182
A1Cresult	>7	num_medications	18	0.0018635672600657	0.0553532410779315	1.326331302961342	0.044653349001175
A1Cresult	>7	num_medications	19	0.0012505517139914	0.0371449380917698	0.9569455876554376	0.0322173089071383
A1Cresult	>7	num_medications	20	0.0011769898484625	0.0349599417334304	1.0033331061032795	0.0337790288529204
A1Cresult	>7	num_medications	21	0.0011524692266195	0.0342316096139839	1.1556568735740849	0.0389072847682119
A1Cresult	>7	num_medications	22	0.0010298661174047	0.0305899490167516	1.1702807699823317	0.0393996247654784
A1Cresult	>7	num_medications	23	0.0006130155460742	0.0182083029861616	0.8428728857907448	0.0283768444948921

Showing 1 to 10 of 31,606 entries (filtered from 71,712 total entries)

Previous **1** 2 3 4 5 ... 3161 Next

The image depicts characteristics that are already present in the DataTable function. We only discuss the code's most critical and self-contained sections. The paper concludes with a quick overview on how to use the table.

3.2.2 Selecting method

If you click on the blue button, you get the sliding window with the list of attributes/ columns. We done it on this way such that the user can see which attributes/columns the dataset have to select. Just the attribute columns have the selecting type.

Chose left attribute:

None



Chose left attribute:

- ✓ None
- None
- race
- gender
- age
- weight
- admission_type_id
- discharge_disposition_id
- admission_source_id
- medical_specialty
- num_medications
- max_glu_serum
- A1Cresult

a. PYCHARM Code

```
with open("association2NoFilter.csv", "r") as f:
    df = pd.read_csv(f)

first = df["lhs_attribute"].unique()
second = df["rhs_attribute"].unique()
first = np.asarray(first)
second = np.asarray(second)

return render_template('server_table.html', title='Assosiation Table', l_column_names=first, r_column_names=second)
```

Using the Panda library, this code reads an association dataset file. It then uses the unique() function to eliminate duplication by extracting the unique values of a column and storing them in a Numpy array for usage in a sliding window.

Lastly, the code uses the Flask method render template to generate an HTML template called „server_table.html“ (). As parameters, the template is given a title string and the arrays, which will be utilised as column search values in the produced HTML.

```
if "latt" in req and req["latt"] is not None:
    leftAttribute = req["latt"]
if "ratt" in req and req["ratt"] is not None:
    righthAttribute = req["ratt"]

if leftAttribute != "None":
    data_filter = data_filter.query('lhs_attribute == "{}"'.format(leftAttribute))
else:
    data_filter = data_filter
if righthAttribute != "None":
    data_filter = data_filter.query('rhs_attribute == "{}"'.format(righthAttribute))
else:
    data_filter = data_filter
```

Afterwards we take the req variable which contains all the requests that can be made by a user. Then it checks if „latt“ (stands for the lhs_attribute) is contained in the dictionary of req and if something was selected to search for. If its true the parameters of „latt“ in req will be put inside a new variable which will be used to filter the data with the corresponding query of the the „latt“ parameters.

b. HTML code

```
<th>
  <label for="latt">Chose left attribute:</label>
  <form id="my-form1" action="{{ url_for('api_data') }}" method="post">
    <select id="latt" name="latt">
      <option value="None">None</option>
      {% for column in l_column_names %}
      <option value="{{ column }}" {% if column == l_column_name %}selected{% endif %}>{{ column }}</option>
      {% endfor %}
    </select>
  </form>
</th>
```

The form has a select element with an ID of "latt" and a name of "latt". The select element contains a default option of "None" and a for loop that iterates over the variable "l_column_names" and generates an option for each item in the list. The l_column_names variable was already created in the python code where we put the unique values of that column.

Then there is a conditional check to see if the current column of the list is the same as the previously selected column/attribute from the sliding window, and if so, it marks that option as selected.

The form's action attribute is set to the URL of the "api_data" endpoint. When the form is submitted, it sends a POST request to the specified endpoint with the selected value of the "latt" select element. In this way the lhs_attribute column can be filtered corresponding to the selected column/attribute.

c. JAVA script

```
$('#my-form1').on('change', function(event) {  
    event.preventDefault();  
    var formData = $(this).serialize();  
    $.ajax({  
        url: '/api/data',  
        data: formData,  
        dataType: 'json',  
        type: 'GET',  
        success: function(data) {  
            table.clear();  
            table.rows.add(data).draw();  
        },  
        error: function(xhr, ajaxOptions, thrownError) {  
            console.log(thrownError);  
        }  
    });  
});
```

This code handles the "change" event of a form with the ID "my-form1" using jQuery. When the form is updated, event.preventDefault is used to prevent the default submit action ().

After serializing the form data into a string with jQuery's serialize() method, an AJAX request is issued with the ajax() function. A GET request to the URL "/api/data" includes the serialized data, and the anticipated return type is JSON.

If the request is successful, the new server data is placed into an HTML table through the DataTables plugin. Previous data in the table is removed. The array of JSON objects containing the data rows that need to be added to the table is sent to the success callback method.

The error callback method reports the error message to the console if the request fails.

```
// Listen for the change event of the select element
$('#latt').on('change', function() {
    // Submit the form when the value changes
    $('#my-form1').submit();
});
```

Then we submit the form by the changed value.

3.2.3 Searching method

The use of this method is because we work here with numerical values and its easier to select the range of the values that we want because there are a lot of different ones. There is a big choice and it would be useless if we make it like the first method because we would get a large list of different values.

Filter Support:

a. PYCHARM code

```
search_support_ = request.args.get('support_search')
```

At first we get the user request that was typed into the search field and save it into a variable.

```
if search_support_ is not None or search_support != "":
    if search_support_ != "":
        try:
            data_filter = data_filter.query('support {}'.format(search_support_))
            search_support = search_support_
        except:
            try:
                data_filter = data_filter.query('support {}'.format(search_support))
            except:
                pass
    else:
        data_filter = data_filter
        search_support = ""
```

The function determines whether or not the `search_support_` or the `search_support` variables are empty or contain `None`. If it is not empty, it uses the `search_support_` value to filter the data. If the `search_support_` parameter is left blank, the data stays unfiltered.

b. HTML code

```
<th>
  <label for="support_search">Filter Support:</label>
  <input type="text" id="support_search" name="support_search" class="form-control form-control-sm">
</th>
```

Users can search for „support“ data using this form control by providing a search word in the input box. The search phrase used by the user while submitting the form may be recorded by the server-side code and used to filter the support data.

c. JAVA script

```
$('#support_search').on('keyup', function(event) {
  event.preventDefault();
  var formData = $(this).serialize();
  $.ajax({
    url: '/api/data',
    data: formData,
    dataType: 'json',
    type: 'GET',
    success: function(data) {
      table.clear();
      table.rows.add(data).draw();
    },
    error: function(xhr, ajaxOptions, thrownError) {
      console.log(thrownError);
    }
  });
});
```


Is almost the same as the previous one.

```
$('#support_search').on('keyup', function () {
  table.column('support:name').search($('#support_search').val()).draw()
});
```

Using this code, an input field with the ID `support search` gains an event listener. When a user writes something into this field, the table's data is filtered according to the value of the input, and the table is then redrawn to only display the rows that adhere to the search criteria.

3.2.4 Using the table

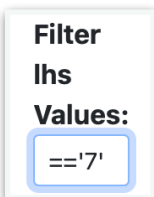
- Clicking on the blue arrow and a sliding window will appear where a value can be selected.



Chose left attribute:

None

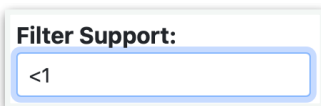
- The Filter lhs Values and Filter rhs Values search fields are different from the rest. The values of this column have multiple types (float, string..), that's why we need to enter, such that every value will be handled as String:
=='your search value'



Filter
lhs
Values:

=='7'

- The values of this column are all numerical values, so you can enter:
== any number or > or < any number



Filter Support:

<1