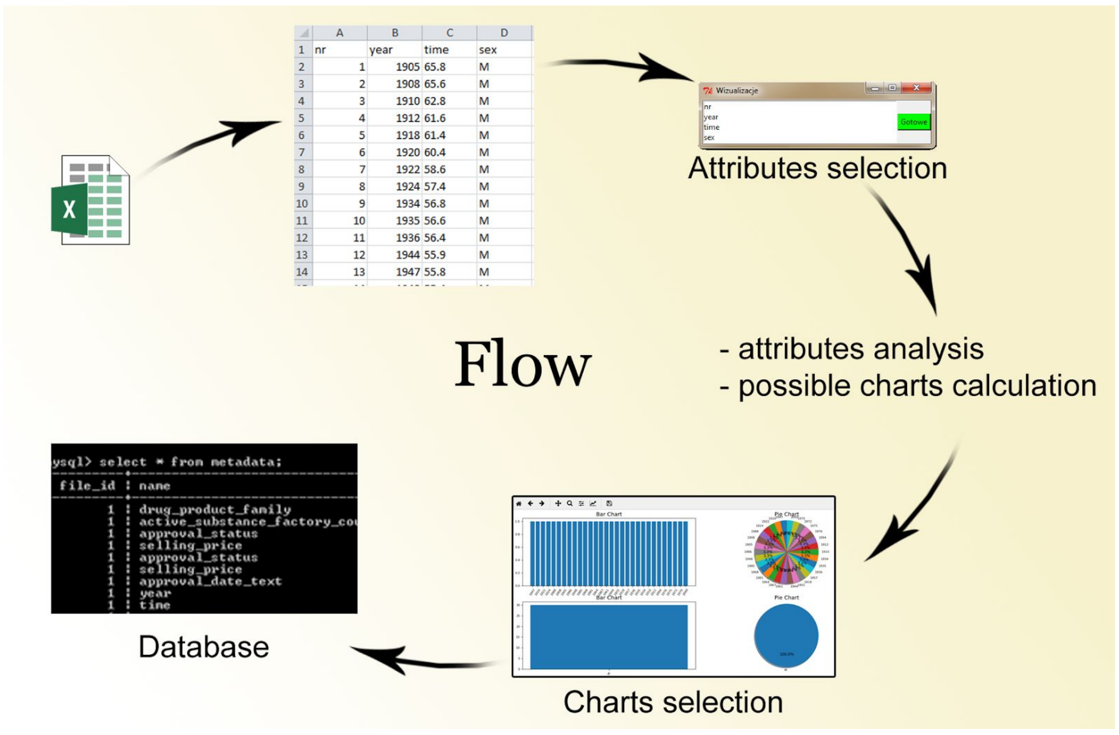
Data visualizer with DB recording feature - semester project

Authors:
Jędrzej Kubiak
Mateusz Półtorak

Introduction

‘Data visualizer with DB recording feature’ is a semi-automated tool for users who want to visualise data from excel files. Application allows you to choose columns which you want to project on diagrams. You can also choose the diagram you like the most - your choice will be written to the database. This feature can be used in future to create fully-automated application via machine learning technique.



	A	B	C	D
1	nr	year	time	sex
2	1	1905	65.8	M
3	2	1908	65.6	M
4	3	1910	62.8	M
5	4	1912	61.6	M
6	5	1918	61.4	M
7	6	1920	60.4	M
8	7	1922	58.6	M
9	8	1924	57.4	M
10	9	1934	56.8	M
11	10	1935	56.6	M
12	11	1936	56.4	M
13	12	1944	55.9	M
14	13	1947	55.8	M

```
ysql> select * from metadata;
----------+-----------------------------
 file_id | name
----------+-----------------------------
       1 | drug_product_family
       1 | active_substance_factory_cou
       1 | approval_status
       1 | selling_price
       1 | approval_status
       1 | selling_price
       1 | approval_date_text
       1 | year
       1 | time
```

Application Overview

Goals

The overriding goal for ‘Data visualizer with DB recording feature’ is to visualize data in quick and simple way. The idea is that generated diagrams are reasonable and useful (in analytical meaning), with as little as possible participation of user. Unlike traditional visualization products, our program allows users to provide reliable diagrams without necessity of mastering complicated visualization applications.
The following goal is to create database which collect name of the best diagram - chosen by the user and its metadata. This database was designed to enhance application, we expect that gathered data can be used in ‘machine learning’ technology. This application enhanced with ‘machine learning’ technology would be much more precise.

Architecture and Design

Attribute class

The class named ‘Attribute’ is responsible for metadata representation. It consist of four essential description structures:

- type
- nominal
- array of raw data
- dictionary of values

1. **Type** (Numeric | Categorical | Date)

- Numeric type is a generalized representation of all numerical types. It stands for integers, decimals, floats, etc.
- Categorical type is used when the data column consists of labels, such as (for example) colors, cities, names.
- Date is detected only when .xlsx file contains a data label (for attribute) in itself.

2. **Nominal** (Monomial | Binomial | Polynomial)

Nominal variable describes the number of unique values that the attribute has. It is segmented for three ranges: <1>, <2> and (2;inf+>. These ranges correspond to Nominal string values.

- Monomial - stands for attributes with only one value.
- Binomial - represents columns with two types of values, like “man/woman”, “accepted/rejected”
- Polynomial - any quantity that is greater than 2.

3. **Array of raw data**

The array of raw data stores list of column records. These records can be different than original .xlsx records because of filtering process which is executed in previous stage (before creating attribute class). Order of records is preserved. Arrays of data are mainly used for creating double charts, where program needs to know record’s positions in array.

4. **Dictionary of values**

Main mechanism of records representation, simple dictionary with {key:value} tuples. Dictionaries are passed directly to chart creating methods. Number of keys is connected with number of distinct values in attribute.
Algorithm for dictionaries creation:
d{} is a dictionary

```
def attribute_to_dict(data):
  d = {}
  for record in data:
     try:
        d[record] += 1
     except KeyError:
        d[record] = 1
  return d
```

Important program mechanics

- Reading values and clearing columns

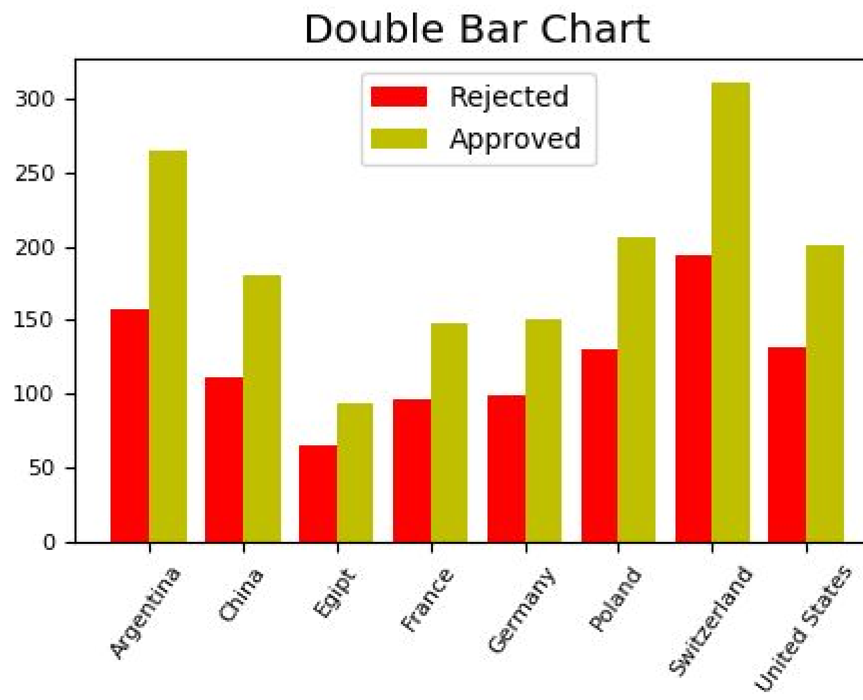
Before '*Attribute Class*' objects are created, each tuple of selected column is checked for consistency of data. The main goal is to get rid of missing values.
There are two cases and two different ways to reject wrong records. It depends on number of selected attributes in previous stage.
This process is executed by *read_values(selection)* function, where argument *selection* is a list of selected columns.

a) If user chooses only one column for analysis, *read_values* method eliminates all records with NULL value.
b) If user chooses two columns (col1, col2) where each column has records r_1 to r_n, *read_values* function removes all pairs of (col1(r_i), col2(r_i)) records that meet the condition (x or y) = NULL.

The cleared columns are processed into '*Attribute Class*' objects.

- *oneDict_twoLists()* function

In most cases, graphs are constructed from simple dictionaries {key;value} (as shown in the previous section). Graph values are dictionary.values() and graph labels are made from dictionary.keys() list.
Making a graph that describes two attributes (where at least one is binomial), requires knowledge of the correlation between them, so two separate dictionaries are not enough.



In this case, a dictionary with pair of keys is constructed. {key1, key2; value}

```
try:
  self.dd[(record1, record2)] += 1
except KeyError:
  self.dd[(record1, record2)] = 1
```

Double graphs requires two lists of values. One for class A, and one for class B (in example above, class A is “Rejected” and class B is “Approved”). This is stage of the process where function *oneDict_twoLists()* is used. It returns two lists with values, one list with keys and two class names (for class A, class B).

oneDict_twoLists() algorithm:

```
def onedict_twolists(d):     # argument d is a dictionary with double keys
  d1, d2 = {}, {}
  class1 = d.keys()[0][1]        # defining name of the first class
  class2 = ‘’
  for key in d.keys():
      if key[1] == class1:
         d1[key[0]] = d[key]
      else:
         class2 = key[1]# definition of the second class
         d2[key[0]] = d[key]
   # some key values can be equal zero, so they don't append in dictionary. We look for
these keys and we generate missing data
   for key in d1.keys():
      if key not in d2.keys():
         d2[key] = 0
   for key in d2.keys():
      if key not in d1.keys():
         d1[key] = 0
   list1, list2 = [], []               # here began lists creation
   keys = sorted(d1.keys())      # keys sorting
   for key in keys:
      list1.append(d1[key])
      list2.append(d2[key])
   return list1, list2, class1, class2, keys
```

- Graph selection

For charts selection, two functions were applied. The first method analyzes the possibilities, the second creates graphs.

analyse()
[chartList]
createCharts(   )

'analyse()' function checks types and nominals of *'Attribute Class'* objects. It fills *'chartList'* table with charts tags. Then, *'chartList'* table is used as a parameter for *'createCharts()'* function.

Database

Application connects to MySQL relational database management system. Database consists of three tables, with the same relationships as shown below:

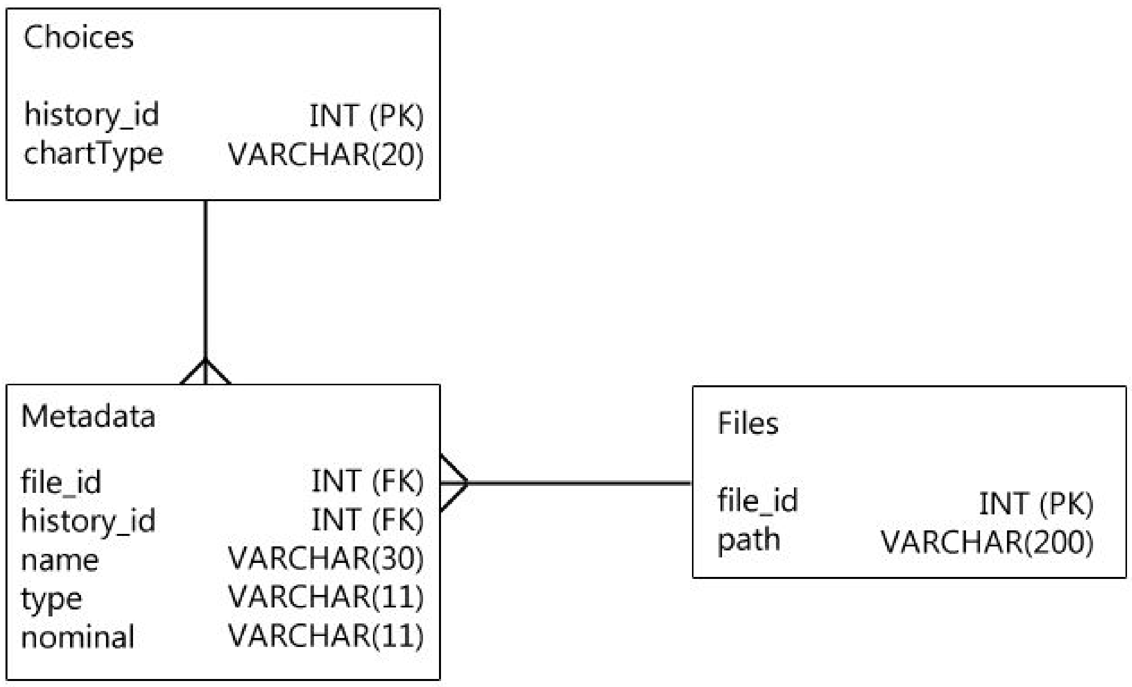


Table "Files" stores informations about loaded .xslx files. Every record is unique, therefore there cannot be two records with the same path. IDs are generated artificially, they increases as the number of data tuples increases.

Table "Choices" creates new records when user selects specific charts. "History_id" column is incremented automatically with every new selection. Every choose has it's own history_id. "chartType" specifies the type of choosen chart.
Example: *"User have selected attribute 'place_of_birth' and the proper chart was a bar chart. It's 144th time when user selects a chart."*
One record will be added to table:

history_id	**chartType**
144	Bar chart

Table "Metadata" represents selected attributes characteristics. There are informations about attribute name, type and nominal. Furthermore, for each selection, foreign keys are created. Keys are connected with tables "files" and "choices". Each choose has it's own set of attributes metadata and each file is linked with many "metadata" records. It implies that each file is in relation with many "choices" rows.

Environment and libraries

All code is written in Python 2.7
Graphs are generated with Matplotlib and Pandas libraries.
For GUI, we used TkInter library.
In addition, some calculations are done with Numpy functions.

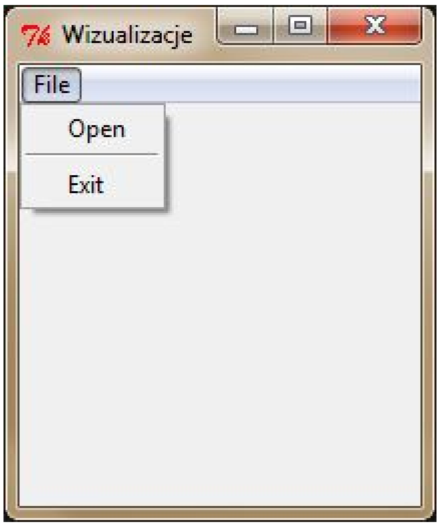
Application was tested on platform containing:

- Intel Core i3-4000M CPU @ 2.4GHz
- 4 GB RAM
- Windows 7 Service Pack 1
- JetBrains PyCharm Community Edition

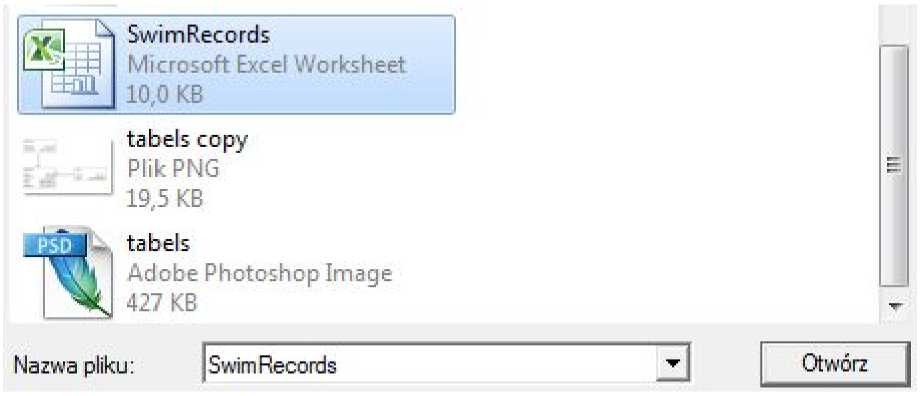
User guide

The interface itself is very simple and intuitive. Despite this, a user guide is always required. This section describes the steps for navigating the interface.

1. The first initialization view is an empty window with file selection option.



2. In this example, user’s choice is the excel file with SwimRecords data.

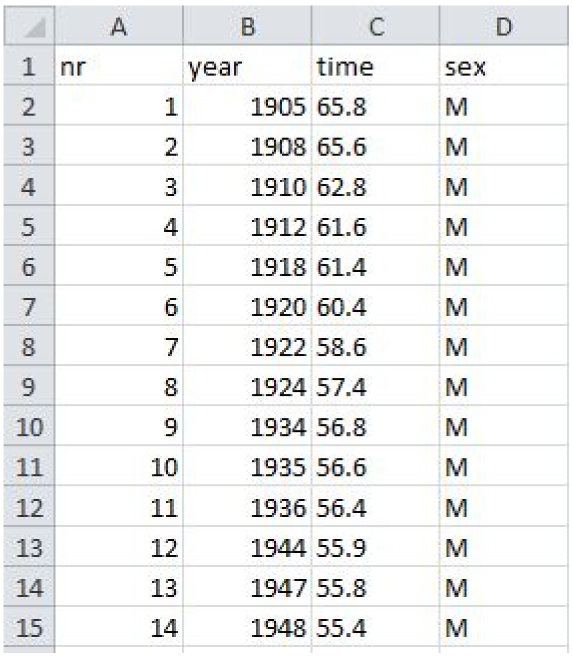

SwimRecords
Microsoft Excel Worksheet
10,0 KB

tabels copy
Plik PNG
19,5 KB

tabels
Adobe Photoshop Image
427 KB

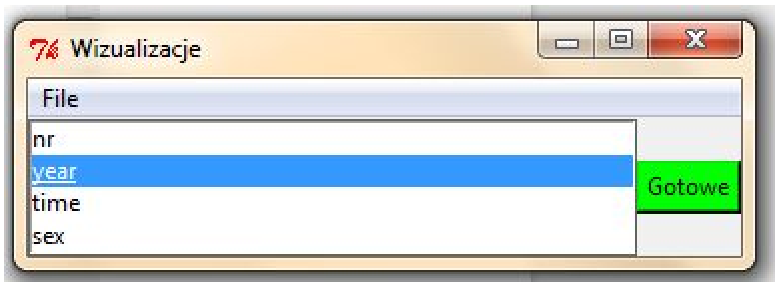
Nazwa pliku: SwimRecords

3. Data inside .xslx file looks like below.

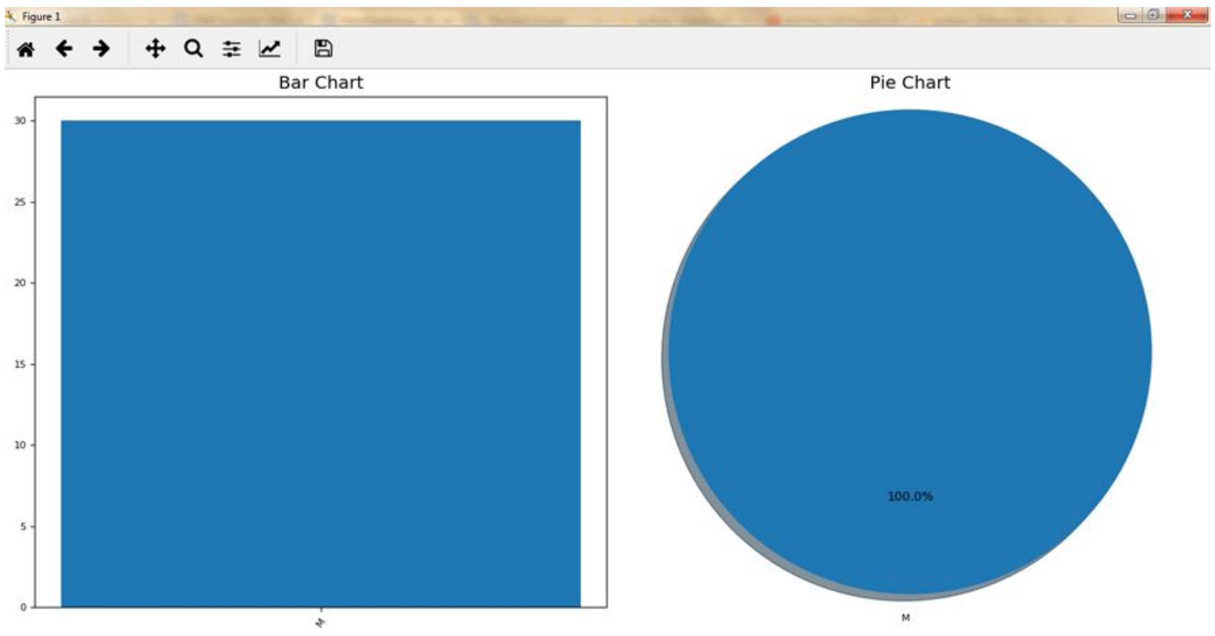


	A	B	C	D
1	nr	year	time	sex
2	1	1905	65.8	M
3	2	1908	65.6	M
4	3	1910	62.8	M
5	4	1912	61.6	M
6	5	1918	61.4	M
7	6	1920	60.4	M
8	7	1922	58.6	M
9	8	1924	57.4	M
10	9	1934	56.8	M
11	10	1935	56.6	M
12	11	1936	56.4	M
13	12	1944	55.9	M
14	13	1947	55.8	M
15	14	1948	55.4	M

4. After file selection, all possible attributes (columns) appear in the window. User has to choose one or two columns.



5. After selection and “Gotowe” click, graphs are presented. In this case, the user has chosen “sex” attribute which stands for gender. We can see that all swim records belong to men. Now, user has to click on interesting chart. Selection will be stored in DB.



Conclusion

Altogether:

- we created efficient application for data visualization,
- user participation in data visualization process is reduced to simple activity,
- program can be enhanced with ‘machine learning’ feature, then some of the data would be visualized without user participation,
- program can be enhanced with more types of charts (in actual version there are six types).