University of Zurich[UZH]

Dynamic and Distributed Information Systems

Data-Oriented Programming – HS18

Extra Exercise: Basic Data Processing

Submission Details

- **Submission Format:** ZIP file which includes all Python code scripts for this exercise and two .csv files for trian and test data sets; and, 2) a .txt file with your first name, last name and student identification number (Matrikelnummer)
- **Submission Deadline:** 23:59 11.11.2017.
- The name of the zip file should have the following format: olatusername_dop_exercise.zip, e.g. *example_dop_exercise.zip*.
- Please divide your code into properly identifiable files which can be assigned to each question.
- Please submit the assignment **by email** to: `ashena@ifi.uzh.ch` and put "DOP submission" in the subject

1 Task: Data Exploration *(4.5 Points)*

1.1 Assignment

As future data scientists, when you get your hands on a new data set, your first job is to explore it and make yourself familiar with its structure and content: Can you picture the data set as a table? If so, what are the entities (rows) and attributes (columns)? What is the data type of each attribute (numerical, nominal, or ordinal)? What are the range and distribution of each attribute? The first part of the exercise focuses on answering these questions.

The Data Set The data set for this exercise contains real data from Titanic. If this is your first time hearing about Titanic, it was the largest passenger liner of its time, 1912. But, unfortunately, it sank due to the collision with an iceberg and a tragedy happened by the death of its 1502 out of 2224 passengers and crew. It is highly recommended that you watch the movie with the same title, Titanic directed by James Cameron.

The data is provided in comma separated values (CSV) format, titanic.csv file. It contains data for 887 of the real Titanic passengers. Each row represents one person. The columns describe different attributes about the person that are described in the table below.

Attribute	Description
`Survived`	Whether the passenger survived or not
`Pclass`	The passenger's class
`Name`	The name recorded for the passenger
`Sex`	The gender of the passenger
`Age`	The age of the passenger
`Fare`	The fair that the passenger paid

a) Read and Display Data from the CSV File *(1.5 Point)*

First, we want to see the records in the CSV file. In this example the data set is small and can be explored in a text editor. But in real use-cases such files are often much larger and can contain many gigabytes of data. Therefore, it makes sense to be able to investigate the data programmatically.

Your task: Extend the implementation of the function `read_csv`. It should read the file `titanic.csv` line-by-line and return the first (or the last) `n` lines as a python list. Write the function in such a way that it stops processing as soon as `n` lines have been collected. Add an extra feature to the function, so it also returns the starting line in the file (typially the header). This feature becomes handy in coming sections.

Below you find a skeleton implementation of this function that you can extend:

- `file_name` specifies the path including the file name to the CSV file.
- `n` specifies the maximum number of lines which will be returned. If `n` is bigger than the number of existing rows in the file, function should return all content of the file.
- `mode` indicates whether the `n` lines should be retrieved from the head (`mode=0`) or the tail (`mode=1`) of the file. `mode=2` means only the header should be returned.
- `lines` is a python list which contains the up to `n` strings; the strings represent the lines contained in `file_name` excluding the header.

```python
import os


def read_csv(file_name, n, mode):
    lines = list()
    # your implementation here
        #if mode == 0:
            #read from head

        #if mode == 1:
            # read from tail

        #if mode == 2:
            # read headings

    return lines


if __name__ == '__main__':
    # this code is only run when the script is executed, e.g. via: python
        task_xx_x.py
    data_file_name = os.path.join("..", "data", "titanic.csv")
    # print heading
    print("".join(read_csv(data_file_name, 0, 2)))
    # print 5 lines from head
    print("".join(read_csv(data_file_name, 5, 1)))
    # print 5 lines from tail
    print("".join(read_csv(data_file_name, 5, 2)))
```

Listing 1: task_01_1.py

Hints

- You may find reveresed() built-in function from Python very useful for this task https://docs.python.org/3/library/functions.html#reversed.

  Below shows the sample outputs when running the function three times with `mode=2`, `0`, and `1`, and `n=5`:

```
Survived,Pclass,Name,Sex,Age,Fare
No,3,Mrs. William (Margaret Norton) Rice,female,39,29.125
No,2,Rev. Juozas Montvila,male,,13
Yes,1,Miss. Margaret Edith Graham,female,500,30
No,3,Miss. Catherine Helen Johnston,female,7,23.45
Yes,1,Mr. Karl Howell Behr,male,26,30
No,3,Mr. Patrick Dooley,male,32,7.75
Survived,Pclass,Name,Sex,Age,Fare
```

b) Statistics for Numerical Data *(1.5 Points)*

The result of the previous task shows what the data records look like and perhaps you can already spot some missing or invalid values for `Age` attribute. But in order to assess the quality of the data it is better you further investigate. You may have realized some attributes contain numbers like `Age`, `Pclass`, and `Fair` and others contain text like `Survived`, `Name`, and `Sex`. In this section, you are going to investigate the numeric attributes by computing:

1. the range of numeric attributes by finding their minimum and maximum values
2. the average of the numeric attributes

Your task: Write the function `attribute_stat` which has two inputs, `data` and `index`. `data` is a list of all rows in `titanic.csv` basically the output of `csv_read` function from the previous task. `index`, starting from zero, is the index of the column of corresponding attribute (`0:Survived`, `1:Pclass`, `2:Name`, `3:Sex`, `4:Age`, `5:Fare`). The function returns the minimum, maximum, and average of the given attribute. Your function should check the type of the attribute value and returns `None` if the type is not numeric.
Extend the function `attribute_stat` in the following python script:

```
1  import os
2
3
4  def read_csv(file_name, n, mode):
5      lines = list()
6      # your implementation from task_01_1
7      return lines
8
9
10 def isfloat(str):
11     try:
12         float(str)
13         return True
14     except ValueError:
15         return False
16
17
18 def attribute_stat(index, data):
19
20     # if attribute is numerical:
21     # return min_value, max_value, mean_value
22     # else:
23     # return None
24
25
26 if __name__ == '__main__':
27     # this code is only run when the script is executed, e.g. via:
           python task_xx_x.py
28     data_file_name = os.path.join("..", "data", "titanic.csv")
29     # headings
30     headings = [heading.strip('\r\n') for heading in "".join(read_csv(
           data_file_name, 0, 2)).split(",")]
```

```
31    # data
32    data = read_csv(data_file_name, 1000, 0) # n =1000 is bigger than
          the number of rows in the file, so all the lines
33    # will be returned!
34
35    # if you were not able to write read_csv, then uncomment the
          following lines:
36    # data = list()
37    # with open(data_file_name) as file_handle:
38    # for line in file_handle:
39    # data.append(line)
40    # headings = [heading.strip('\r\n') for heading in "".join(data.
          pop(0)).split(",")]
41    column_index = 0
42    for heading in headings:
43       attributeStat = attribute_stat(column_index, data)
44       if attributeStat is None:
45          print("{} => It doesn't look like that values of this column
                are numeric.".format(
46             headings[column_index]))
47       else:
48          print("{} => min={}, max={}, mean={}".format(
49             headings[column_index], attributeStat[0], attributeStat
                  [1], attributeStat[2]))
50       column_index += 1
```

Listing 2: task_01_2.py

Note:

- If you could not solve the previous assignment, you can find guidance as comment in the provided python script, `task_01_2`.
- Function `isfloat` is provided that you can utilize it in your code.
- If you remember from the output of assignment 1.1.a, there are missing values for column age. Typically, they are empty strings like "" in python. Fortunately, function `isfloat` returns false for an empty string.

The script should produce the following:

```
1  Survived => It doesn't look like that values of this column are
       numeric.
2  Pclass => min=1.0, max=3.0, mean=2.30552423901
3  Name => It doesn't look like that values of this column are numeric.
4  Sex => It doesn't look like that values of this column are numeric.
5  Age => min=0.42, max=500.0, mean=32.2221599045
6  Fare => min=0.0, max=512.3292, mean=32.3054201804
```

This output is very interesting! You see age ranges from 0.42 to 500. The lower bound makes sense as there might be infants in the ship. But, the upper bound indicates that there are some invalid data for the age. Another interesting point is that `attribute_stat` has

returned valid output for `Pclass`. But, in reality the average of the passenger class doesn't mean anything. Although `Pclass` has numerical value, it is a categorical data. More than average, frequency of each category can be useful.

Hints

- You can use the `sum`, `max`, and `min` built-in functions in python to get the sum of a list and the maximum and minimum values of a list. These functions require the list elements to be a numeric type in order to work as expected.
- `None` is a special value in python. Here, we can use it to indicate that computation was not possible on a given attribute. Comparisons with None will always return False and mathematical operations involving None will fail.

c) Learn About Nominal Attribute *(1.5 Points)*

From the result of the previous task, we couldn't obtain useful information about the nominal attributes, `Survived`, `Pclass`, and `Sex`. In this section, you are going to investigate the distribution of these attributes. Basically, you are going to find the answers of these questions: What are the values of a given nominal attribute and how many times a value occurs in the data?

Your task: Extend the script below by implementing a body for function `attribute_count` which has two inputs, `data` and `index`. `data` is a list containing all rows in `titanic.csv`, the output of `csv_read` function from the first task. `index`, starting from zero, is the index of the column of corresponding attribute (for this task we are only interested in `0:Survived`, `1:Pclass`, and `3:Sex`). The function returns a python dictionary where the keys are the distinct attribute values and the associated values are the number of times that value occurs in the data set.

You should implement this function using the `Counter` type from the python `collections` module. This type is a specialised dictionary for counting. You can work with it like with normal dictionaries but it offers some additional functionality too. Learn about how to use this type in the official documentation (https://docs.python.org/3/library/collections.html#collections.Counter).

Extend the function `attribute_counts` in the following python script:

```
1  import os
2  from collections import Counter
3
4
5  def read_csv(file_name, n, mode):
6
7      lines = list()
8      # your implementation from task_01_1
9      return lines
10
11
12
13 def attribute_counts(index, data):
14     counter = Counter()
15     # your implementation here
16     return counter
17
18
19 if __name__ == '__main__':
20     # this code is only run when the script is executed, e.g. via:
           python task_xx_x.py
21     data_file_name = os.path.join("..", "data", "titanic.csv")
22     # headings
23     headings = [heading.strip('\r\n') for heading in "".join(read_csv(
           data_file_name, 0, 2)).split(",")]
24     # data
```

```
25    data = read_csv(data_file_name, 1000, 0) # n =1000 is bigger than
          the number of rows in the file, so all the lines
26    # will be returned!
27
28    # if you were not able to write read_csv, then uncomment the
          following lines:
29    # data = list()
30    # with open(data_file_name) as file_handle:
31    # for line in file_handle:
32    # data.append(line)
33    # headings = [heading.strip('\r\n') for heading in "".join(data.
          pop(0)).split(",")]
34
35    print("Survived", attribute_counts(0, data))
36    print("Pclass",attribute_counts(1, data))
37    print("Sex",attribute_counts(3, data))
```

Listing 3: task_01_3.py

Note:

- If you could not solve the first assignment, you can find guidance as comment in the provided python script, `task_01_3`.

The script should produce the following:

```
1  ('Survived', Counter({'No': 545, 'Yes': 342}))
2  ('Pclass', Counter({'3': 487, '1': 216, '2': 184}))
3  ('Sex', Counter({'male': 573, 'female': 314}))
```

1.2 Remarks

1. https://docs.python.org/3/library/functions.html#reversed
2. https://docs.python.org/3/library/collections.html#collections.Counter

2 Task: Data Cleaning *(5.5 Points)*

2.1 Assignment

We have seen that the age attribute has some missing and invalid values. For example, have look at two following rows from the data set:

```
Survived,Pclass,Name,Sex,Age,Fare
No,2,Rev. Juozas Montvila,male,,13
Yes,1,Miss. Margaret Edith Graham,female,500,30
```

Rev. Montivla's age is missing (there is no value provided for it.), and Miss Graham's age is reported as 500!
You are going to learn how to handle invalid entries and missing values in assignments 2.1.a and 2.1.b respectively.

a) Discard Records with Invalid Age *(1 Points)*

Let's consider the acceptable range for human age is 0 to 99. Below you will find a program, very similar to assignment 1.1.a, that reads the data set line by line and decides whether the current line should be discarded or appended to the final list.
Your task: Your task is to extend the following script by writing a body for function `is_age_valid` which gets a record as string (one row of a data set) and returns false if the age is less than 0 or bigger than 99. Pass the cleaned data set to your `attribute_stat` to see whether invalid ages are removed correctly.

Note:

- None or empty values for age are not invalid but missing. Therefore, keep them while cleaning the data.
- In some cases, data analyst may decide to change invalid values to None instead of discarding. This decision depends on the application and the means of data collection. Here, we assume that if ,in a record, an attribute is invalid other attributes may also have imprecise values.

Extend the following script by writing a body for function `is_age_valid`:

```
import os


AGE_INDEX = 4


def is_float(value):
    try:
        float(value)
        return True
    except ValueError:
        return False



```

```python
def attribute_stat(index, data):
    # if attribute is numerical:
    # return min_value, max_value, mean_value
    # else:
    # return None


def is_age_valid(record):

    # your implementation here
    return True

def clean_data(file_name):
    cleaned_records = list()
    with open(file_name) as file_handle:
        for row in file_handle:
            if is_age_valid(row):
                cleaned_records.append(row)

    return cleaned_records


if __name__ == '__main__':
    # this code is only run when the script is executed, e.g. via:
        python task_xx_x.py
    data_file_name = os.path.join("..", "data", "titanic.csv")
    cleaned_data = clean_data(data_file_name)
    print(attribute_stat(4, cleaned_data))
```

Listing 4: task_01_2.py

The script should produce the following:

```
(0.42, 99.0, 29.55924096385542)
```

b) Missing Data Imputation *(2.5 Points)*

One way to handle missing values is to find a substitute for them. For example, you can fill missing values with the mean (or median in case of discrete values) for that attribute in the data set. It is very important that by this replacement you don't change the overall distribution of the data. Therefore, it is a good practice to look at the distribution of data before and after replacement. Usually, when data distribution is normal or close to normal, filling missing values with the mean is one of the common approaches.

Your task: Extend the provided script by writing a body for functions: `histogram_age` and `replace_missing_data`.
`histogram_age` is a function that gets your cleaned data from the previous step as input, and returns a python dictionary. Each key in the dictionary indicates a range of 10 for age attribute. For each range (key), this function counts the number of passengers whose age

lies in that range. In other words, the value of each key shows the frequency of that age range. The output of the function on cleaned data should be like this:

```
{'20-29': 274, '70-79': 6, '50-59': 47, '90-99': 1, '40-49': 107,
    '60-69': 24, '10-19': 120, '80-89': 1, '0-9': 68, '30-39': 182}
```

Note:

- Function `text_histogram_age` gets the dictionary returned by `histogram_age` and plots the distribution of age data.
- Please, don't change the provided dictionary initialization. Otherwise, you can't use `text_histogram_age` to visualize the distribution of the data.

`replace_missing_data` function gets your cleaned data from the previous step and a value as inputs, and returns a new data set in which the missing values are replaced with the given value.

Note:

- The statistics of age data and an alternative code for cleaned data is given, so students that couldn't solve the previous assignments don't face problems.

Run the script and observe how the distribution of attribute changes when missing values are replaced with the mean, maximum, or minimum.

```
import os

AGE_INDEX = 4
AGE_AVERAGE = 29.55
AGE_MINIMUM = 0.42
AGE_MAXIMUM = 99


def replace_missing_data(substitute, data):
    replaced_data = list()
    # Your implementation here

    return replaced_data


def text_histogram_age(hist):
    keys = ["0-9", "10-19", "20-29", "30-39", "40-49", "50-59", "60-69
        ", "70-79", "80-89", "90-99"]
    for key in keys:
        if len(key) < 5:
            row_hist = " " + key + " : "
        else:
            row_hist = key + ": "
        for x in range(0, hist[key] / 4):
            row_hist += "#"
        row_hist += str(hist[key])
        print(row_hist)
```

```
27
28
29  def histogram_age(data):
30      histogram = {"0-9": 0, "10-19": 0, "20-29": 0, "30-39": 0, "40-49"
            : 0,
31                   "50-59": 0, "60-69": 0, "70-79": 0, "80-89": 0, "90-99":
                         0}
32      # Your implementation here
33      return histogram
34
35
36  if __name__ == '__main__':
37      # this code is only run when the script is executed, e.g. via:
            python task_xx_x.py
38      data_file_name = os.path.join("..", "data", "titanic.csv")
39      cleaned_data = clean_data(data_file_name)
40
41      # if you were not able to write clean_data, then uncomment the
            following lines:
42      # cleaned_data = list()
43      # with open("..\data\cleaned_titanic.csv") as file_handle:
44      # for line in file_handle:
45      # cleaned_data.append(line)
46      print("Distribution of cleaned data without replacing missing
            values:")
47      print(histogram_age(cleaned_data))
48      text_histogram_age(histogram_age(cleaned_data))
49      print("Distribution of cleaned data after replacing missing values
             with mean:")
50      replaced_cleaned_data = replace_missing_data(AGE_AVERAGE,
            cleaned_data)
51      print(histogram_age(replaced_cleaned_data))
52      text_histogram_age(histogram_age(replaced_cleaned_data))
53      #run again by replacing missing values with minimum and maximum as
             well.
```

Listing 5: task_02_2.py

Hints

- See https://docs.python.org/3/library/stdtypes.html#string-methods for the documentation of string manipulation methods.

c) Write Cleaned Data with No Missing Value to File *(2 Point)*

After preprocessing and cleaning te data, data scientists often divide the data in two sets known as train and test. They investigate the train set thoroughly and obtain statistical model based on the data. Then, they evaluate their model on test data.

Your task: Consult the documentation for the `csv` module and complete the following script. Your code selects randomly 20% of the records in the final data set (cleaned and with no missing value) and write them in a file named `titanic_test.csv`. The remaining records that are not selected should be written in a separate file named `titanic_train.csv`.

Note:

- The cleaned and with no missing value data is provided, so students that couldn't complete previous assignments don't face problems.

```
1  import os
2
3  AGE_AVERAGE = 29.55
4
5  def is_float(value):
6      try:
7          float(value)
8          return True
9      except ValueError:
10         return False
11
12
13 def is_age_valid(record):
14     # your implementation here
15     return True
16
17
18 def clean_data(file_name):
19     cleaned_records = list()
20     with open(file_name) as file_handle:
21         for row in file_handle:
22             if is_age_valid(row):
23                 cleaned_records.append(row)
24
25     return cleaned_records
26
27
28 def replace_missing_data(substitute, data):
29     replaced_data = list()
30     # Your implementation here
31
32     return replaced_data
33
34 if __name__ == '__main__':
35     # this code is only run when the script is executed, e.g. via:
           python task_xx_x.py
36     data_file_name = os.path.join("..", "data", "titanic.csv")
37     cleaned_data = clean_data(data_file_name)
38     replaced_cleaned_data = replace_missing_data(AGE_AVERAGE,
           cleaned_data)
```

```
39
40      # # if you were not able to write replace_missing_data, then
            uncomment the following lines:
41      # replaced_cleaned_data = list()
42      # with open("..\data\\replaced_cleaned_titanic.csv") as
            file_handle:
43      # for line in file_handle:
44      # cleaned_data.append(line)
```

Listing 6: task_02_2.py

Hints

- See https://docs.python.org/3/library/random.html for the documentation of python random generator.

2.2 Remarks

1. https://docs.python.org/3/library/stdtypes.html#string-methods
2. https://docs.python.org/3/library/random.html