21.08.2019

Tomasz Boczek

Rekrutacja Junior Data Science

Table of contents

Task 1.

In order to find the brewery, which produces the strongest beers by ABV%, a simple solution was applied. The solution was based on a for-loop. The strongest beer was found to be produced in Schorschbräu, it reaches ABV of 57.7%. The code is attached below.

```
import csv

with open('piwo.csv', newline='', encoding='utf8') as datafile:
    piwo = csv.reader(datafile)

    strongest_brewery = [[]]
    highest_abv = 0

    forloop_counter = 0
    error_counter = 0
    for data_row in piwo:

        # The first row is a header:
        if forloop_counter == 0:
            forloop_counter = 1
            continue

        # Since the data file is huge, and there is no possibility to make sure
        there aren't any mistakes in the cells,
        # which will be used for numerical operations, it is advisable to manage
        the ValueError.
        try:

            # If the considered beer has higher abv than the highest found so far,
            it will overwrite the data in
            # the "strongest" list.
            if float(data_row[11]) > highest_abv:
                strongest_brewery = [[data_row[0], data_row[1], data_row[11]]]
                highest_abv = float(data_row[11])

            # If the considered beer has the same abv as the highest found so far,
            it will add another brewery to
            # the "strongest" list.
            elif float(data_row[11]) == highest_abv:

                already_on_list = False
                # It might though happen, the brewery is already in the list:
                for brewery in strongest_brewery:
                    if data_row[0] == brewery[0]:
                            already_on_list = True
                            break

                    # After checking, whether the particular brewery has been inscribed
                     to the list yet, the brewery can be
                    # saved in the list.
                    if not already_on_list:
                        strongest_brewery.append([data_row[0], data_row[1],
                                                  data_row[11]])

        except ValueError:
            error_counter += 1

    if len(strongest_brewery) > 1:
        print("The strongest beers are produced accordingly in: ")
        for brewery in strongest_brewery:
            print("Brewery " + brewery[1] + ", highest abv " + brewery[2])
    else:
        print("The strongest beer is produced in " +
              strongest_brewery[0][1] + ", the brewery ID: " +
              strongest_brewery[0][0] + ". The strongest beer produced there
               reaches ABV of " +
              strongest_brewery[0][2] + "%.")
```

Task 2.

I'd definitely start from picking the best beers according to the *review_overall*. After obtaining the list of such beers, I'd make another filtration according to some other indicator, for example taste or aroma. At the end, as it appears from the results, there are many beers on the list, which are perfect accordeing to all aspects. The program automatically chose 3 of them:

- Big Eddy Russian Imperial Stout, ID:39334
- Leglifter Light, ID: 16088
- Timmermans Kriek Lambic, ID: 6562

The fragment of code, from which the results were generated is attached below:

```
import csv

with open('piwo.csv', newline='', encoding='utf8') as datafile:
    piwo = csv.reader(datafile)

    # Taking into consideration the fact, that the value inscribed in column
    # "review_overall" is supposed to be
    # presumably most indicative, it will be used as the major determinant of the
    # recommendation. However, it might
    # happen, there will be too many beers, more than 3, with the same note.
    # Therefore, as the second determinant the
    # value of "review_taste" is going to be used, thirdly the "review_aroma" etc.

    recommended = [[]]
    highest_overall = 0
    forloop_counter = 0
    error_counter = 0

    for review_beer in piwo:

        # The first row is a header:
        if forloop_counter == 0:
            forloop_counter = 1
            continue

        # Since the data file is huge, and there is no possibility to make sure
        # there aren't any mistakes in the cells,
        # which will be used for numerical operations, it is advisable to manage
        # the ValueError.
        try:

            # If the beer has a higher overall note, the "recommended" list will be
             # overwritten.
            if float(review_beer[3]) > highest_overall:
                recommended = [[review_beer[12], review_beer[10], review_beer[3],
                                  review_beer[9], review_beer[4], review_beer[8]]]
                highest_overall = float(review_beer[3])

            # if the beer has the same overall note as the beers in the
             # "recommended" list,
            # it will be added to the list.
            elif float(review_beer[3]) == highest_overall:

                already_on_list = False
                # It might though happen, the beer is already in the list:
                for beer in recommended:
                    if review_beer[12] == beer[0]:
                        already_on_list = True
                        break
                if not already_on_list:
                    recommended.append([review_beer[12], review_beer[10],
                                       review_beer[3], review_beer[9],
                                       review_beer[4], review_beer[8]])

        except ValueError:
            error_counter += 1

    # Only 3 best beers must be chosen:
    if len(recommended) > 3:
        recommended = sorted(recommended, key=lambda beer: beer[3])
        recommended = sorted(recommended, key=lambda beer: beer[4])
        recommended = sorted(recommended, key=lambda beer: beer[5])
        recommended.reverse()

    count = 0

    print("Beers recommended: ")
    for r in recommended:
        if count < 3:
            print(r[1] + ", ID: " + r[0])
            count += 1
        else:
            break
```

Task 3.

In order to establish the most important factor in determining the overall quality of beer, a method based on average deviation was applied. Each beer has certain set of notes for particular aspects (taste, aroma, appearance etc.). The aspects differentiate among each other in case of the influence on the overall note. To compare this difference in influence, for each aspect of each beer in the list, the deviation from the overall note was measured. Afterwards, the average deviation for all aspects was calculated. The less the average deviation is, the more indicative the aspect is. As it appears from the results, the *taste* seems to be the most indicative feature of beer.

The results from the investigation:

Aspect	Aroma	Taste	Appearance	Palette
Deviation	0.43	0.31	0.47	0.37

The fragment of code, from which the results were generated is attached below:

```
import csv

# For this task, the idea is basically doing an investigation of the
average deviations of particular indicators
# from the overall review note. The bigger the deviation is, the less
significant the indicator is.
with open('piwo.csv', newline='', encoding='utf8') as datafile:
    piwo = csv.reader(datafile)

    # In the following variables the average deviation from the overall
    # note is going to be saved:
    aroma_dev = 0
    taste_dev = 0
    appearance_dev = 0
    palette_dev = 0

    counter = 0
    for review_beer in piwo:

        if counter == 0:
            counter += 1
            continue
        elif counter == 1:
            aroma_dev = abs(float(review_beer[3]) - float(review_beer[4]))
            taste_dev = abs(float(review_beer[3]) - float(review_beer[9]))
            appearance_dev = abs(float(review_beer[3]) -
            float(review_beer[5]))
            palette_dev = abs(float(review_beer[3]) -
            float(review_beer[8]))
            counter += 1
            continue

        aroma_dev = (aroma_dev * (counter - 1) + abs(float(review_beer[3])
        - float(review_beer[4]))) / counter
        taste_dev = (taste_dev * (counter - 1) + abs(float(review_beer[3])
        - float(review_beer[9]))) / counter
        appearance_dev = (appearance_dev * (counter - 1) +
        abs(float(review_beer[3]) - float(review_beer[5]))) / counter
        palette_dev = (palette_dev * (counter - 1) +
        abs(float(review_beer[3]) - float(review_beer[8]))) / counter

        counter += 1

    print(aroma_dev, taste_dev, appearance_dev, palette_dev)
```

Task 4.

Due to the fact, that you picked two objectives as your major ones, it is necessary to treat the task as an multi-objective exercise. The two objectives are aroma and appearance, the features of the beer. Since the question refers to the beer style, it was decided to segregate the beers according to the beer style. After doing so, the average values of the objectives (aroma and appearance) were calculated accordingly for beer styles. The figure 4.1 below shows the average values of aroma and appearance for particular kinds of beers.

As it appears from the figure 4.1, it is not possible to decide, which exactly style of beer is preferable for you. At least not before specyfing which one of th objectives is more valuable. Therefore, giving a single answer for this task wouldn't be reasonable.

For the purpose of such problems, a common method of giving a complete answer is finding a Pareto set. Pareto set contains of elements (beer styles in this case) which, according to the assumptions, are not dominated by any other in the considered set. Domination appears, when one of elements is simultaneously better regarding one of the objctives and is not worse regarding the other objectives. On the figure 4.2 the Pareto set is depicted. There are 3 beer styles, which are the best ones, according to the simplified assumptions.

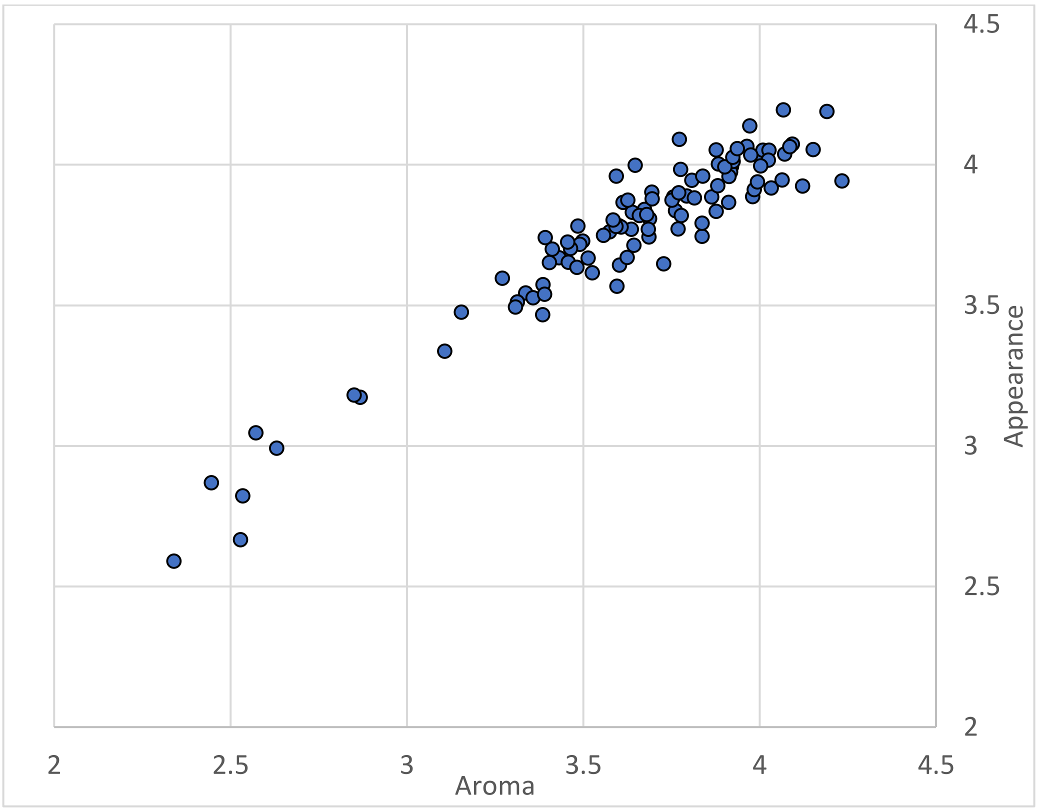

Figure 4.1. Average values of aroma and appearance for different kind of beer styles.

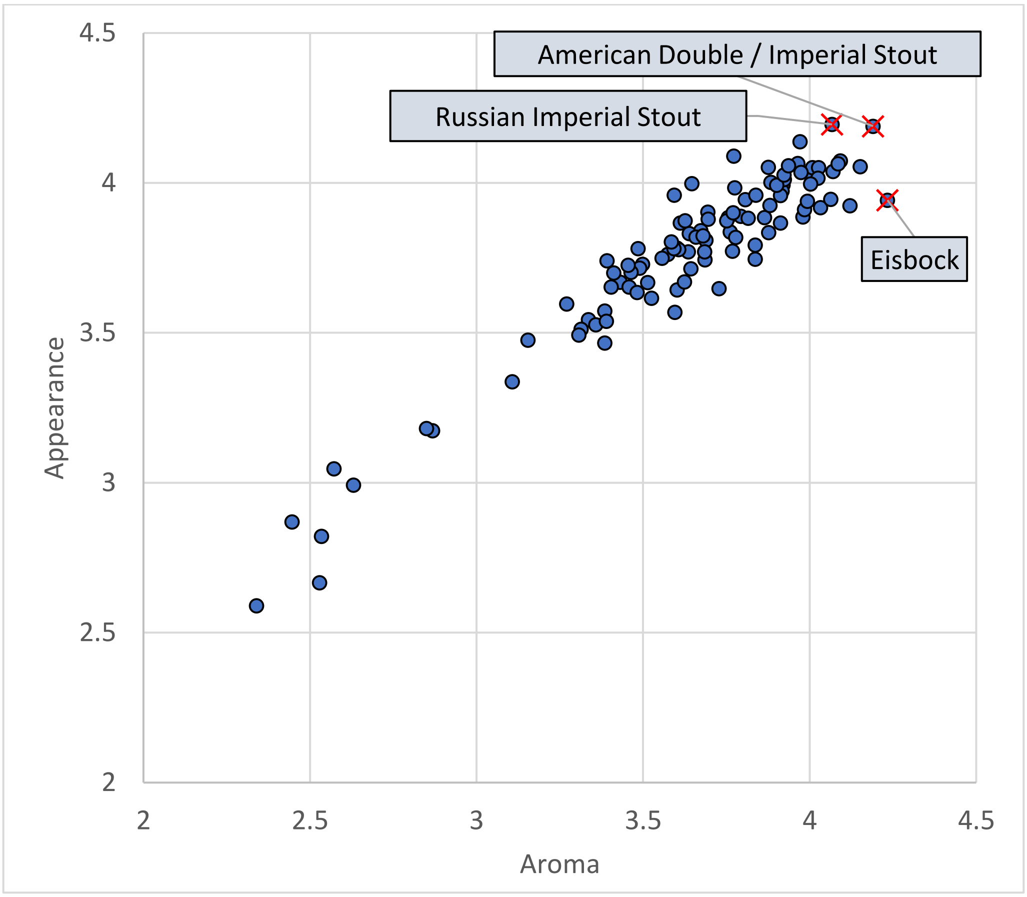

Figure 4.2. Average values of aroma and appearance – Pareto set.

```
import csv

import xlsxwriter as xlsxwriter

def is_dominated_by_any(beer_style, dominants):
    for dominant in dominants:
        if (beer_style[1] < dominant[1] and beer_style[2] <= dominant[2]) \
                or (beer_style[1] <= dominant[1] and beer_style[2] <
                    dominant[2]):
            return True
    return False


def front(results):

    if len(results) == 1:
        return results

    else:
        t = front(results[0:int(len(results)/2)])
        b = front(results[int(len(results)/2):len(results)])
        m = []  # list containing final results
        for member in b:
            if not is_dominated_by_any(member, t):
                m.append(member)
    return m + t
```

```python
with open('piwo.csv', newline='', encoding='utf8') as datafile:
    piwo = csv.reader(datafile)

    # The list "results" is going to contain a following set of data for
    # each beer style: average aroma,
    # average appearance, the list of aroma notes and the list of
    # appearance notes.
    results = []
    counter = 0
    for review_beer in piwo:

        if counter == 0:
            counter += 1
            continue

        if len(results) == 0:
            results.append([review_beer[7], float(review_beer[4]),
                           float(review_beer[5]), [float(review_beer[4])],
                           [float(review_beer[5])]])

        else:

            result_saved = False
            for beer_style in results:
                # if the beer style is already in the list of results:
                if beer_style[0] == review_beer[7]:
                    beer_style[1] = (beer_style[1] * len(beer_style[3]) +
                                     float(review_beer[4])) \
                                    / (len(beer_style[3]) + 1)
                    beer_style[2] = (beer_style[2] * len(beer_style[4]) +
                                     float(review_beer[5])) \
                                    / (len(beer_style[4]) + 1)
                    beer_style[3].append(float(review_beer[4]))
                    beer_style[4].append(float(review_beer[5]))
                    counter += 1
                    result_saved = True
                    break
            # if the beer style is new to the list of results:
            if not result_saved:
                results.append([review_beer[7], float(review_beer[4]),
                               float(review_beer[5]),
                               [float(review_beer[4])],
                               [float(review_beer[5])]])

        counter += 1

    # After the segregation is done, the list "results" is going to be
saved into the external xlsx file.
    workbook = xlsxwriter.Workbook("Task_4_results.xlsx")
    worksheet = workbook.add_worksheet("sheet")

    worksheet.write(0, 0, "Beer style")
    worksheet.write(0, 1, "Aroma - average")
    worksheet.write(0, 2, "Appearence - average")
    counter = 1
    for beer_style in results:
        worksheet.write(counter, 0, beer_style[0])
        worksheet.write(counter, 1, beer_style[1])
        worksheet.write(counter, 2, beer_style[2])
        counter += 1

    # After saving the results, it is advisable to pick the best fitting
    # set of beer styles. In order to do this,
    # a recursive function is going to be used in order to find the Pareto
    # set, which basically means finding the
    # set of the best beer styles among the entire set. Before using the
    # "front()" function, it is necessary
    # to sort the results.
    results = sorted(results, key=lambda beer_style: beer_style[1])
    results.reverse()
    pareto_set = front(results)
    worksheet.write(0, 4, "Beer style")
    worksheet.write(0, 5, "Aroma - average")
    worksheet.write(0, 6, "Appearence - average")
    counter = 1
    for beer_style in pareto_set:
        worksheet.write(counter, 4, beer_style[0])
        worksheet.write(counter, 5, beer_style[1])
        worksheet.write(counter, 6, beer_style[2])
        counter += 1

    workbook.close()
```