Politechnika Wrocławska
Wydział Informatyki i Zarządzania

Raport

Aplikacja analizująca dane o wiadomościach wysłanych w serwisie Facebook

Przedmiot:	Języki Skryptowe Laboratorium
Imię i nazwisko autora:	Konrad Liuras
Nr indeksu:	246752
Semestr studiów:	5
Data ukończenia pracy:	Styczeń 2021 r.
Prowadzący laboratorium:	Mgr inż. Natalia Piórkowska

HR EXCELLENCE IN RESEARCH
Politechnika Wrocławska
Wybrzeże Wyspiańskiego 27
50-370 Wrocław
wiz.pwr.edu.pl

Spis treści

Raport 1
1. Cel projektu 4
2. Wymagania projektowe 4
Podstawowe wymagania funkcjonalne: 4
Podstawowe wymagania niefunkcjonalne: 4
3. Opis implementacji i zastosowanych rozwiązań 5
Fragmenty kodu - najważniejsze funkcje 6
Generowanie wszystkich statystyk dla wszystkich konwersacji: 6
Przekazywanie danych do wygenerowania „chmury słów” dla uczestników pojedynczej konwersacji: 7
Przekazywanie danych do wygenerowania „chmury słów” dla pojedynczej osoby ze wszystkich konwersacji: 8
Tworzenie i wyświetlanie tabeli z dwudziestoma najczęściej używanymi słowami przez osobę/konwersację: 9
Generowanie wykresów z zależnością wysłanych wiadomości od miesiąca dla uczestników wybranej konwersacji: 10
Zliczanie wszystkich informacji o wybranej konwersacji: 11
Generowanie i wyświetlanie „chmury słów” na podstawie przekazanych danych: 12
Zliczanie wszystkich informacji o słowach użytych przez podaną osobę we wszystkich konwersacjach: 12
Zbiór funkcji odpowiedzialnych za generowanie modelu do klasyfikacji wiadomości: 13
4. Opis działania i prezentacja interfejsu 15
Opis sposobu instalacji i uruchomienia aplikacji 15
Opis działania 15
Screeny przedstawiające działanie aplikacji. 16
Pierwsza zakładka - Wordcloud 16
Ekran startowy aplikacji – pierwsza zakładka: 16
Stan po zmianie szablonu na loc.png: 16
Aplikacja po zmianie ścieżki pojedynczej konwersacji (pasek statusu): 17
Wygenerowane „chmur słów” oraz tabel 20 najczęstszych słów dla powyższych ustawień: 17
Aplikacja po zmianie czcionki (pasek statusu): 18
Wygenerowane „chmury słów” oraz tabel 20 najczęstszych słów dla powyższych ustawień dla pojedynczego uczestnika: 18
Wyświetlenie historii ustawień, dla których generowane były chmury: 19
Wyświetlenie zapisanych szablonów ustawień: 19
Druga zakładka – Your stats 20
Druga zakładka przed podjęciem na niej jakichkolwiek akcji: 20
Aplikacja po wybraniu opcji analizy plików Messengera (pojawia się możliwość zapisanej wcześniej analizy): 20
Wyświetlenie rankingu przeanalizowanych konwersacji: 21
Fragment pliku z zapisanym rankingiem konwersacji (po wciśnięciu przycisku „Save ranking”): 21

Wyświetlenie szczegółów konwersacji: 22
Zapisanie szczegółów do pliku .txt: 22
Zapisany plik .txt: 22
Zapisanie szczegółów do pliku .csv: 23
Zapisany plik .csv: 23
Dane z pliku otwarte w Excelu: 23
5. Podsumowanie 24
Podsumowanie wykonania projektu 24
6. Literatura 24

1. Cel projektu

Celem projektu było wytworzenie aplikacji, która pozwoli przeanalizować dogłębnie (m.in. pod kątem nacechowania emocjonalnego) prywatne wiadomości wymienione przez użytkownika za pośrednictwem serwisu Facebook, sporządzić z zebranych danych liczne wykresy oraz wygenerować spersonalizowaną „chmurę słów" składającą się z najczęściej występujących słów.

2. Wymagania projektowe

Podstawowe wymagania funkcjonalne:

- Aplikacja pozwala wygenerować, wyświetlić oraz zapisać „chmurę słów" złożoną z najczęściej używanych słów w wybranej konwersacji
- Aplikacja pozwala wybrać rodzaj (oddzielna, wspólna) „chmury słów" dla pojedynczej konwersacji
- Aplikacja pozwala wygenerować, wyświetlić oraz zapisać „chmurę słów" złożoną z najczęściej używanych słów przez wybranego uczestnika konwersacji we wszystkich konwersacjach
- Aplikacja pozwala dostosować czcionkę, wzór i kolor „chmury" oraz minimalną długość słowa, które może się znaleźć w „chmurze"
- Aplikacja pozwala zapisywać, wczytywać, usuwać i przeglądać szablony ustawień szczegółów „chmury słów"
- Aplikacja pozwala wyświetlić podsumowanie (procent wszystkich wiadomości i liczba wiadomości w konwersacji) wszystkich konwersacji
- Aplikacja pozwala wyświetlić i zapisać (w pliku .txt lub .csv) szczegóły (liczba i średnia długość wiadomości dla każdego uczestnika konwersacji) wybranej, przeanalizowanej wcześniej konwersacji
- Aplikacja pozwala wyświetlić i zapisać wykresy średniego czasu odpowiedzi (dla konwersacji dwuosobowych) oraz liczby wiadomości wysłanych przez każdego uczestnika konwersacji w zależności od miesiąca
- Aplikacja pozwala wyświetlić ranking pięciu najczęściej używanych reakcji dla każdego uczestnika wybranej konwersacji wraz z liczbą użyć
- Aplikacja pozwala wyświetlić jaki procent wiadomości w konwersacji został sklasyfikowany przez przygotowany model do każdej z predefiniowanych grup: szczęście, rozbawienie, neutralne, smutek, złość

Podstawowe wymagania niefunkcjonalne:

- Aplikacja implementowana jest w środowisku Python 3.8
- Interfejs wykonany został przy użyciu biblioteki tkinter
- Do wyświetlania wykresów wykorzystywana jest biblioteka matplotlib
- Aplikacja działa z użyciem pobranych uprzednio danych z serwisu Facebook
- Aplikacja działa na komputerze użytkownika

3. Opis implementacji i zastosowanych rozwiązań

(Schemat plików oraz zestaw wykorzystywanych bibliotek może ulec zmianie)

Aplikacja była testowana przy użyciu interpretera Python 3.8. Została ona napisana w języku python3 z wykorzystaniem importowanych do aplikacji bibliotek: os, numpy, json, fnmatch, wordcloud, copy, random, matplotlib.pyplot, PIL.Image, io, tkinter, pickle, requests, re, pandas, scipy, stempel, content, csv

Część odpowiedzialna za logikę przetwarzania wiadomości zawiera się w pliku: wordcloudAPP.py
Część odpowiedzialna za interfejs graficzny składa się z pliku gui.py
Część odpowiedzialna za analizę nacechowania wiadomości znajduje się w plikach: text_analyzer.py oraz content.py
W kilku katalogach znajduje się dodatkowa zawartość, wspomagająca działanie programu:
Clouds – zapisane „chmury słów”
Masks – przygotowane szablony kształtów „chmur słów”
Fonts – czcionki
W pliku „history.pkl” zapisywana jest historia wygenerowanych chmur
W pliku „templates.pkl” zapisywane są szablony generowania chmur
W pliku „Help.txt” znajduje się pomoc do pobrania informacji o swoich konwersacjach z Facebooka
W pliku „config.txt” znajduje się informacja o tym czy jest to pierwsze uruchomienie aplikacji

Fragmenty kodu - najważniejsze funkcje

(Opis funkcji i walidacja kodu może ulec zmianie)

Generowanie wszystkich statystyk dla wszystkich konwersacji:

```
def create_messenger_statistics(inbox_path):
    """
        Generate all statistics for every conversation in inbox_path

        :param inbox_path: path of inbox directory, containing all conversation directories
        :return: dictionary containing all details for all conversations
    """
    stats_by_conv = {}

    msg_by_month = {}
    msg_count = {}
    msg_length = {}
    response_times_by_months = {}
    all_reactions = {}
    all_messages = {}
    try:
        for _, dirs, _ in os.walk(inbox_path):
            for dir in dirs:    # For every directory (conversation) in inbox_path directory
                for _, _, filenames in os.walk("%s/%s" % (inbox_path, dir)):
                    for filename in filenames:  # For every file in conversation directory
                        if fnmatch.fnmatch(filename, 'message*'):
                            _, new_by_month, new_responses, new_mess_count, new_mess_length, new_reactions, new_messages = \
                                count_file("%s/%s" % ("%s/%s" % (inbox_path, dir), filename), 0) # Actual counting function

                            if len(msg_by_month) == 0:
                                msg_by_month = new_by_month
                            else:
                                merge_dictionaries(msg_by_month, new_by_month)
                            if len(response_times_by_months) == 0:
                                response_times_by_months = new_responses
                            else:
                                merge_dictionaries(response_times_by_months, new_responses)
                            if len(all_reactions) == 0:
                                all_reactions = new_reactions
                            else:
                                merge_dictionaries(all_reactions, new_reactions)
                            if len(msg_count) == 0:
                                msg_count = new_mess_count
                            else:
                                add_dicionaries(msg_count, new_mess_count)
                             if len(msg_count) == 0:
                                 msg_length = new_mess_length
                             else:
                                 add_dicionaries(msg_length, new_mess_length)
                             if len(all_messages) == 0:
                                 all_messages = new_messages
                             else:
                                 for part, msges in new_messages.items():
                                     if part in all_messages:
                                         all_messages[part] += msges
                                     else:
                                         all_messages[part] = msges
                    print("Directory:", dir)
                    for participant, dates in response_times_by_months.items():   # Getting average response time
                        for date in dates:
                            dates[date] = int(sum(dates[date]) / max(1, len(dates[date])))
                    for part in all_reactions:  # Sorting reactions by amounts descending
                        all_reactions[part] = {k: v for k, v in
                                               sorted(all_reactions[part].items(), key=lambda item: item[1], reverse=True)}
                    stats_by_conv[dir] = {"response": copy.deepcopy(response_times_by_months),
                                          "msg_count": copy.deepcopy(msg_count),
                                          "msg_by_month": copy.deepcopy(msg_by_month),
                                          "msg_length": copy.deepcopy(msg_length),
                                          "reactions": copy.deepcopy(all_reactions),
                                          "messages": copy.deepcopy(all_messages)}  # Packing conversations details

                    response_times_by_months.clear()
                    all_reactions.clear()
                    msg_count.clear()
                    msg_by_month.clear()
                    msg_length.clear()
                    all_messages.clear()
    except FileNotFoundError as e:
        print(e, "Wrong path")
        return None

    return {k: v for k, v in
            sorted(stats_by_conv.items(), key=lambda conv: sum(conv[1]["msg_count"].values()), reverse=True)}
```

Przekazywanie danych do wygenerowania „chmury słów” dla uczestników pojedynczej konwersacji:

```
def create_single_conversation_wordcloud(separated, all, single_conv_path, image, min_length, font, color):
    """
        Generate wordcloud from one conversation

        :param color: color name
        :param font: font path
        :param min_length: minimum word's length to be counted
        :param image: path to mask image
        :param single_conv_path: path of conversation directory, containing all messages
        :param all: if 1, wordcloud made of all words in conversation is generated
        :param separated: if 1 every participant will have it's own wordcloud generated
        :return: dictionary containing all details for all conversations
    """
    dictionaries = {}
    try:
        for _, _, filenames in os.walk(single_conv_path):
            for filename in filenames:
                if fnmatch.fnmatch(filename, 'message*'):
                    new_dic, _, _, _, _, _, _ = count_file(
                        "%s/%s" % (single_conv_path, filename), min_length)
                    if len(dictionaries) == 0:
                        dictionaries = new_dic
                    else:
                        merge_dictionaries(dictionaries, new_dic)
    except FileNotFoundError as e:
        print(e, "Wrong directory")
        return None
    if dictionaries == {}:
        print("No data - single conversation")
        return None
    if separated == 1:
        for key in dictionaries:  # Sorting words in dictionaries by amounts
            word_cloud(dictionaries[key], key, image, 0, font, color)
            dic_sorted = {k: v for k, v in sorted(dictionaries[key].items(), key=lambda item: item[1], reverse=True)}
            most_freq(dic_sorted, key, min_length)
    if all == 1:
        all_dic = merge_all_dictionaries(dictionaries)
        word_cloud(all_dic, "All participants", image, 0, font, color)
        all_dic_sorted = {k: v for k, v in sorted(all_dic.items(), key=lambda item: item[1], reverse=True)}
        most_freq(all_dic_sorted, "ALL PARTICIPANTS", min_length)
```

Przekazywanie danych do wygenerowania „chmury słów” dla pojedynczej osoby ze wszystkich konwersacji:

```
def create_single_participant_wordcloud(single_person_path, name, image, min_length, font, color):
    """
        Generate wordcloud from all words used by 'name' in all conversations in 'inbox' path

        :param name: participant's name
        :param color: color name
        :param font: font path
        :param min_length: minimum word's length to be counted
        :param image: path to mask image
        :param single_person_path: path of inbox directory, containing all conversation directories
        :return: dictionary containing all details for all conversations
    """
    dictionaries = {}
    try:
        for _, dirs, _ in os.walk(single_person_path):
            for dir in dirs:
                for _, dirs2, filenames in os.walk("%s/%s" % (single_person_path, dir)):
                    for filename in filenames:
                        if fnmatch.fnmatch(filename, 'message*'):
                            newDic = count_file_one_person("%s/%s/%s" % (single_person_path, dir, filename), name,
                                                           min_length)  # Counting function
                            print("%s/%s/%s" % (single_person_path, dir, filename))
                            if len(dictionaries) == 0:
                                dictionaries = newDic
                            else:
                                merge_dictionaries(dictionaries, newDic)
    except FileNotFoundError as e:
        print(e, "Wrong directory")
        return None
    if dictionaries == {}:
        print("No data - single person")
        return None
    for key in dictionaries:  # Sorting words in dictionaries by amounts
        word_cloud(dictionaries[key], key, image, 0, font, color)
        dic = {k: v for k, v in sorted(dictionaries[key].items(), key=lambda item: item[1], reverse=True)}
        most_freq(dic, key, min_length)
```

Tworzenie i wyświetlanie tabeli z dwudziestoma najczęściej używanymi słowami przez osobę/konwersację:

```
def most_freq(dictionary, name, min_length):
    data = []
    for key in dictionary:
        data.append([key, dictionary[key]])
    number_of_words = 0
    for key in dictionary:
        number_of_words += dictionary[key]
    columns = ('word', 'frequency')
    i = 0
    cell_text = []
    for row in range(20):  # Appending top 20 words
        i += 1
        try:
            cell_text.append(data[row])
            if i > 20:
                break
        except:
            cell_text.append(["", ""])

    row_labels = list(range(1, 21))
    fig, ax = plt.subplots()

    ax.xaxis.set_visible(False)  # Hiding x axis
    ax.yaxis.set_visible(False)  # Hiding y axis
    the_table = ax.table(cellText=cell_text,
                         colWidths=[0.2] * 3,
                         colLabels=columns,
                         rowLabels=row_labels,
                         loc='center')  # Creating table with data
    the_table.auto_set_font_size(False)

    plt.tick_params(axis='x', which='both', bottom=False, top=False, labelbottom=False)
    plt.tick_params(axis='y', which='both', right=False, left=False, labelleft=False)
    for pos in ['right', 'top', 'bottom', 'left']:
        plt.gca().spines[pos].set_visible(False)

    plt.title("TOP 20 MOST FREQUENTLY USED WORDS by %s\nTotal number of words longer than %s characters: %s" % (
        name, min_length - 1, str(number_of_words)))  # Setting title
    plt.draw()  # Show plot
    plt.waitforbuttonpress(0)
```

Generowanie wykresów z zależnością wysłanych wiadomości od miesiąca dla uczestników wybranej konwersacji:

```
def create_graph(dates_dictionaries, title_name, yaxis_name):
    def dic_to_matrix(dic):...

    def sort_matrix_date(matrix):...

    dictionaries = {}
    for person in dates_dictionaries:
        dictionaries[person] = sort_matrix_date(dic_to_matrix(dates_dictionaries[person]))

    months = []
    all_messages = {}
    people = []
    for person in dictionaries:
        for num_tuple in dictionaries[person]:
            months.append("%s/%s" % (num_tuple[0], num_tuple[1][-2:]))
        break

    for person in dictionaries:
        people.append(person)
        all_messages[person] = []
        for number in dictionaries[person]:
            all_messages[person].append(number[2])

    print("Months: ", months)
    # print(yaxis_name, ": ", all_messages)
    """ x = np.arange(len(months))
    width = 0.35"""
    X = np.arange(len(months))
    fig, ax = plt.subplots()

    colors = ['b', 'g', 'r', 'y', 'purple', 'black']
    xs = [0.00]
    for i in range(0, len(people)):
        xs.append((i + 1) / (len(people) + 1))
    # xs=[0.00,0.10,0.20,0.30,0.40,0.50]
    i = 0
    rects_array = []
    for person in all_messages:
        rects_array.append(ax.bar(X + 0.1 + xs[i], all_messages[person], width=xs[1], label=people[i]))
        i += 1
    ax.set_ylabel(yaxis_name)
    ax.set_xlabel('Months')
    ax.set_title(title_name)
    ax.set_xticks(X + xs[len(people) // 2])
    ax.set_xticklabels(months, rotation=270)  # obrócone etykiety miesiecy
    ax.legend()

    def autolabel(rects):...


    plt.show()
```

Zliczanie wszystkich informacji o wybranej konwersacji:

```python
def count_file(file_arg, min_length):
    participants_dict = {}  # All participants
    msg_by_month_file = {}  # Messages by month
    response_times = {}
    last_messages_time = {}
    messages_count = {}
    total_messages_length = {}
    reactions = {}
    all_messages = {}
    count_response = True
    with open(file_arg, 'r', encoding="utf8") as file:
        filedata = file.read()
        data = json.loads(filedata)  # load json data from file

    for name in data["participants"]:
        encoded_name = name["name"].encode('raw_unicode_escape').decode('utf8')
        participants_dict[encoded_name] = {}
        msg_by_month_file[encoded_name] = {}
        last_messages_time[encoded_name] = 0
        response_times[encoded_name] = {}
        messages_count[encoded_name] = 0
        total_messages_length[encoded_name] = 0
        reactions[encoded_name] = {}
        all_messages[encoded_name] = []

    if len(participants_dict) != 2:
        count_response = False

    for message in data["messages"]:
        if message["sender_name"].encode('raw_unicode_escape').decode('utf8') in participants_dict:
            sender = message["sender_name"].encode('raw_unicode_escape').decode('utf8')
            ################### START RESPONSE TIME ####################
            if count_response:
                for key in participants_dict.keys():
                    if key != sender:
                        receiver = key
                time = message["timestamp_ms"] // 1000

            msg_date = msg_to_date(message)

            #  Check if tht's a response
            for someone in msg_by_month_file:
                if msg_date not in msg_by_month_file[someone]:
                    msg_by_month_file[someone][msg_date] = 0
                    if count_response:
                        response_times[someone][msg_date] = []

            msg_by_month_file[sender][msg_date] += 1
            if count_response:
                if last_messages_time[sender] == max(
                        last_messages_time.values()) and 0 not in last_messages_time.values():
                    if last_messages_time[receiver] - time < 60 * 60 * HOURS:
                        response_times[receiver][msg_date].append(last_messages_time[receiver] - time)
                last_messages_time[sender] = time
            ################### END RESPONSE TIME ####################
            if "reactions" in message:
                for reaction in message["reactions"]:
                    reaction_emoji = reaction["reaction"]

                    actor = reaction["actor"].encode('raw_unicode_escape').decode('utf8')
                    if actor in reactions:
                        if reaction_emoji in reactions[actor]:
                            reactions[actor][reaction_emoji] += 1
                        else:
                            reactions[actor][reaction_emoji] = 1

            if "content" in message:
                all_messages[sender].append(message["content"])
                messages_count[sender] += 1
                total_messages_length[sender] += len(message["content"])

                for word in re.split('[ ,.|\*|\n| |\u00e2\u0080\u009d|\u00e2\u0080\u009e|:\)|\)', message["content"]):
                    word = word.encode('raw_unicode_escape').decode('utf8')
                    if not curse_words_filter(word.lower()) and len(word) >= min_length:
                        if word.lower() not in participants_dict[sender]:
                            participants_dict[sender][word.lower()] = 1
                        else:
                            participants_dict[sender][word.lower()] += 1
    return participants_dict, msg_by_month_file, response_times, messages_count, total_messages_length, reactions, all_messages
```

Generowanie i wyświetlanie „chmury słów” na podstawie przekazanych danych:

```
def word_cloud(frequencies, title, mask_img, font, given_color):
    global COLOR
    COLOR = given_color
    fig = plt.figure(num="%s' WordCloud" % (title))
    mask = np.array(PIL.Image.open(mask_img))
    image_colors = ImageColorGenerator(mask, default_color=(0, 0, 0))
    wordcloud = WordCloud(font_path=font,
                          relative_scaling=1.0, mask=mask, background_color="white").generate_from_frequencies(
                          frequencies)
    plt.title(title)
    plt.rcParams.update({"text.color": "black"})
    plt.rcParams['image.cmap'] = 'gray'

    if COLOR == "Random":
        plt.imshow(wordcloud, interpolation='bilinear')
    elif COLOR == "From mask":
        plt.imshow(wordcloud.recolor(color_func=image_colors), interpolation='bilinear')
    else:
        plt.imshow(wordcloud.recolor(color_func=color_to_hsl), interpolation='bilinear')
    plt.axis("off")
    plt.savefig("Clouds/%s.png" % title, format="png")  # Save in 'Clouds' directory
    plt.draw()
    plt.waitforbuttonpress(0)
    return wordcloud
```

Zliczanie wszystkich informacji o słowach użytych przez podaną osobę we wszystkich konwersacjach:

```
def count_file_one_person(file_arg, one_name, min_length):
    participants_dict = {}
    with open(file_arg, 'r', encoding="utf8") as file:
        file_data = file.read()
        data = json.loads(file_data)
    for name in data["participants"]:  # For every participant
        if name["name"].encode('raw_unicode_escape').decode('utf8') == one_name:
            participants_dict[name["name"].encode('raw_unicode_escape').decode('utf8')] = {}

    for message in data["messages"]:
        if message["sender_name"].encode('raw_unicode_escape').decode('utf8') in participants_dict:
            sender = message["sender_name"].encode('raw_unicode_escape').decode('utf8')
            if "content" in message:  # If it's an actual message
                for word in re.split(' |  |\*|\n| |\u00e2\u0080\u009d|\u00e2\u0080\u009e|:\)|\)', message["content"]):
                    word = word.encode('raw_unicode_escape').decode('utf8')
                    if not curse_words_filter(word.lower()):  # If not a curse word :/
                        if word.lower() not in participants_dict[sender] and len(
                                word) > min_length and sender == one_name:
                            participants_dict[sender][word.lower()] = 1

    return participants_dict
```

Zbiór funkcji odpowiedzialnych za generowanie modelu do klasyfikacji wiadomości:

```python
def hamming_distance(X, X_train):
    """
    Zwróć odległość Hamminga dla obiektów ze zbioru *X* od obiektów z *X_train*.

    :param X: zbiór porównywanych obiektów N1xD
    :param X_train: zbiór obiektów do których porównujemy N2xD
    :return: macierz odległości pomiędzy obiektami z "X" i "X_train" N1xN2
    """

    X = X.toarray()
    X_train = X_train.toarray()

    X_train = X_train.transpose()

    outArr = X.astype(np.uint8) @ X_train.astype(np.uint8)
    outArr2 = (~X).astype(np.uint8) @ (~X_train).astype(np.uint8)
    arr = outArr + outArr2

    return np.subtract(np.uint8(X_train.shape[0]), arr)
```

```python
def sort_train_labels_knn(Dist, y):
    """
    Posortuj etykiety klas danych treningowych *y* względem prawdopodobieństw
    zawartych w macierzy *Dist*.

    :param Dist: macierz odległości pomiędzy obiektami z "X" i "X_train" N1xN2
    :param y: wektor etykiet o długości N2
    :return: macierz etykiet klas posortowana względem wartości podobieństw
        odpowiadającego wiersza macierzy Dist N1xN2

    Do sortowania użyj algorytmu mergesort.
    """
    w = Dist.argsort(kind='mergesort')
    return y[w]
```

```python
def p_y_x_knn(y, k):
    """
    Wyznacz rozkład prawdopodobieństwa p(y|x) każdej z klas dla obiektów
    ze zbioru testowego wykorzystując klasyfikator KNN wyuczony na danych
    treningowych.

    :param y: macierz posortowanych etykiet dla danych treningowych N1xN2
    :param k: liczba najbliższych sąsiadów dla KNN
    :return: macierz prawdopodobieństw p(y|x) dla obiektów z "X" N1xM
    """
    print("Shape 0:", np.shape(y)[0], "Shape 1:", np.shape(y)[1])

    points_number = 5               # liczba możliwych klas
    result_matrix = []
    for i in range(np.shape(y)[0]):
        helper = []
        for j in range(k):
            helper.append(y[i][j])
        line = np.bincount(helper, None, points_number)
        result_matrix.append([line[0] / k, line[1] / k, line[2] / k, line[3] / k, line[4] / k])
    return result_matrix
```

```python
def model_selection_knn(X_val, X_train, y_val, y_train, k_values):
    # y_val=np.array(y_val, dtype=np.uint8)
    # y_train=np.array(y_train, dtype=np.uint8)
    """
    Wylicz błąd dla różnych wartości *k*. Dokonaj selekcji modelu KNN
    wyznaczając najlepszą wartość *k*, tj. taką, dla której wartość błędu jest
    najniższa.

    :param X_val: zbiór danych walidacyjnych N1xD
    :param X_train: zbiór danych treningowych N2xD
    :param y_val: etykiety klas dla danych walidacyjnych 1xN1
    :param y_train: etykiety klas dla danych treningowych 1xN2
    :param k_values: wartości parametru k, które mają zostać sprawdzone
    :return: krotka (best_error, best_k, errors), gdzie "best_error" to
        najniższy osiągnięty błąd, "best_k" to "k" dla którego błąd był
        najniższy, a "errors" - lista wartości błędów dla kolejnych
        "k" z "k_values"
    """
    k = len(k_values)
    errors = []
    Dist = hamming_distance(X_val, X_train)
    ksort = sort_train_labels_knn(Dist, y_train)
    for i in range(0, k):
        error = classification_error(p_y_x_knn(ksort, k_values[i]), y_val)
        errors.append(error)
    best_error = min(errors)
    best_k = k_values[np.argmin(errors)]
    return best_error, best_k, errors
def classification_error(p_y_x, y_true):
    """
    Wyznacz błąd klasyfikacji.

    :param p_y_x: macierz przewidywanych prawdopodobieństw - każdy wiersz
        macierzy reprezentuje rozkład p(y|x) NxM
    :param y_true: zbiór rzeczywistych etykiet klas 1xN
    :return: błąd klasyfikacji
    """
    n = len(p_y_x)
    m = len(p_y_x[0])
    res = 0
    for i in range(0, n):
        if (m - np.argmax(p_y_x[i][::-1]) - 1) != y_true[i]:
            res += 1
    return res / n
```

4. Opis działania i prezentacja interfejsu

Opis sposobu instalacji i uruchomienia aplikacji

- do wypełnienia

Opis działania

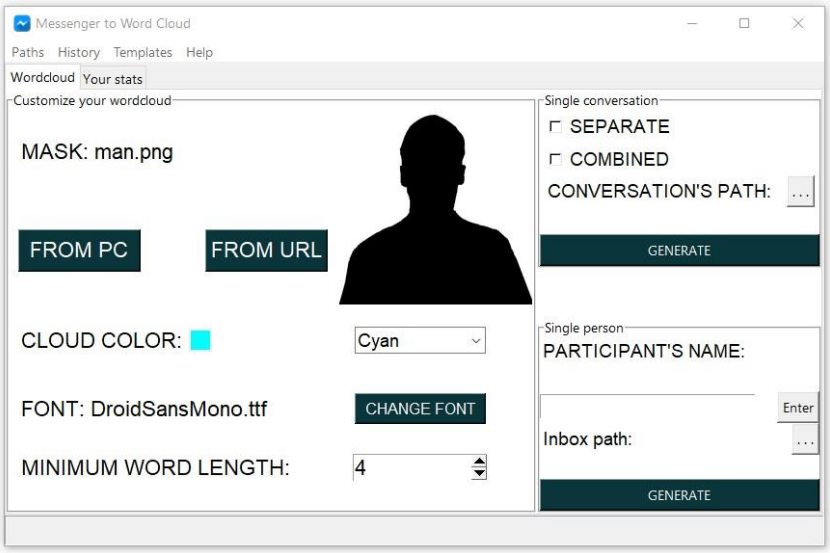
Aplikacja, korzystając z pobranych wcześniej z serwisu Facebook danych o przeprowadzonych rozmowach, pozwala na wygenerowanie „chmury" z najczęściej używanych słów w wybranej konwersacji lub we wszystkich konwersacjach dla podanego uczestnika. Dodatkowo umożliwia zapisywanie, wczytywanie i usuwanie szablonów oraz historii. Druga zakładka umożliwia użytkownikowi przeanalizowanie szczegółów wszystkich konwersacji i przedstawia wyniki w postaci rankingu. Po wejściu w szczegóły konwersacji, pojawia się okno z posortowanymi malejąco według liczby wiadomości informacjami o uczestniku, średniej długości jego wiadomości oraz liczbie wysłanych wiadomości w konwersacji. Z poziomu tego widoku, użytkownik może:

- zapisać plik ze szczegółami w formacie .txt,
- zapisać plik ze szczegółami w formacie.csv,
- wyświetlić najczęściej używane przez uczestników konwersacji reakcje,
- wyświetlić analizę semantyczną wiadomości każdego z uczestników (procentowy udział wiadomości w 5 ustalonych klasach: szczęście, rozbawienie, neutralne, smutek, złość)
- wyświetlić wykres średniego czasu odpowiedzi na wiadomości z podziałem na miesiące (tylko dla konwersacji dwóch uczestników)
- wyświetlić wykres liczby wiadomości wysłanych przez każdego z uczestników konwersacji w zależności od miesiąca

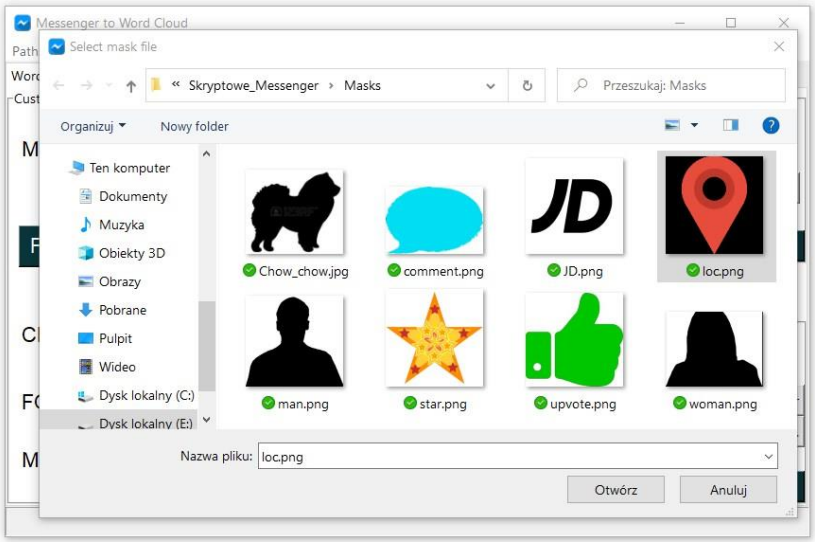
Screeny przedstawiające działanie aplikacji.

Pierwsza zakładka - Wordcloud

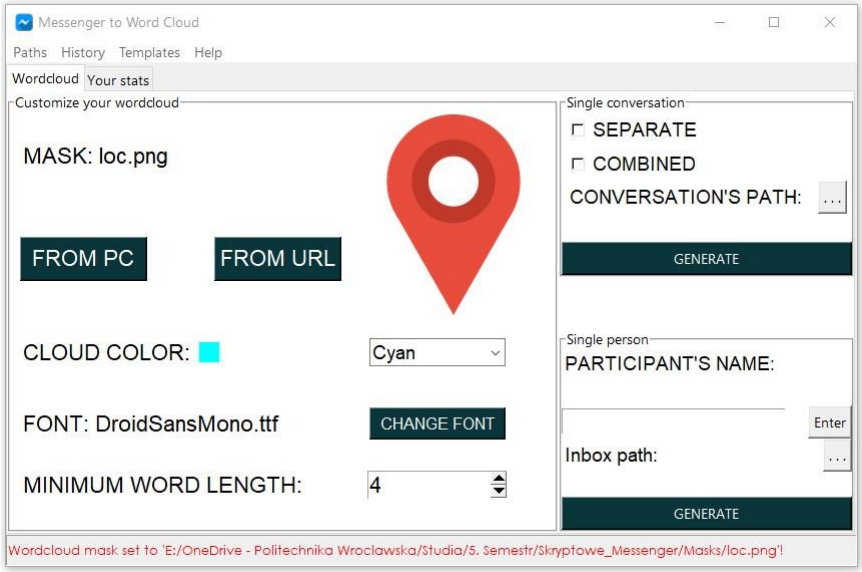
Ekran startowy aplikacji – pierwsza zakładka:



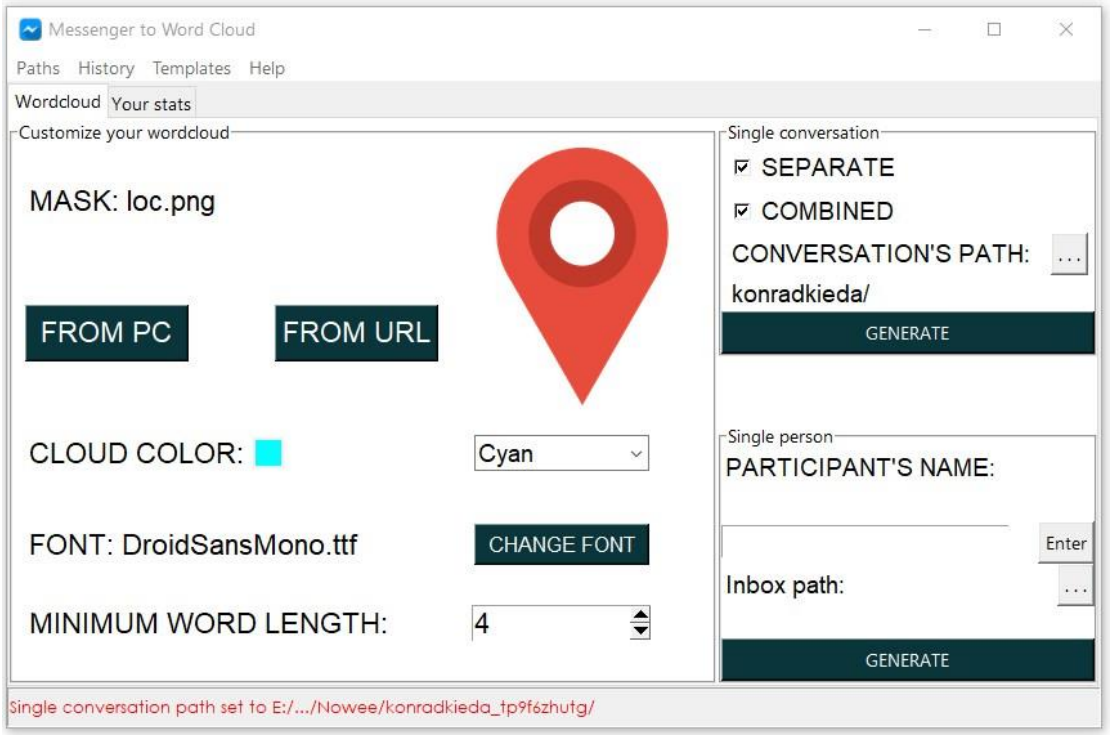
Zmiana szablonu chmury:



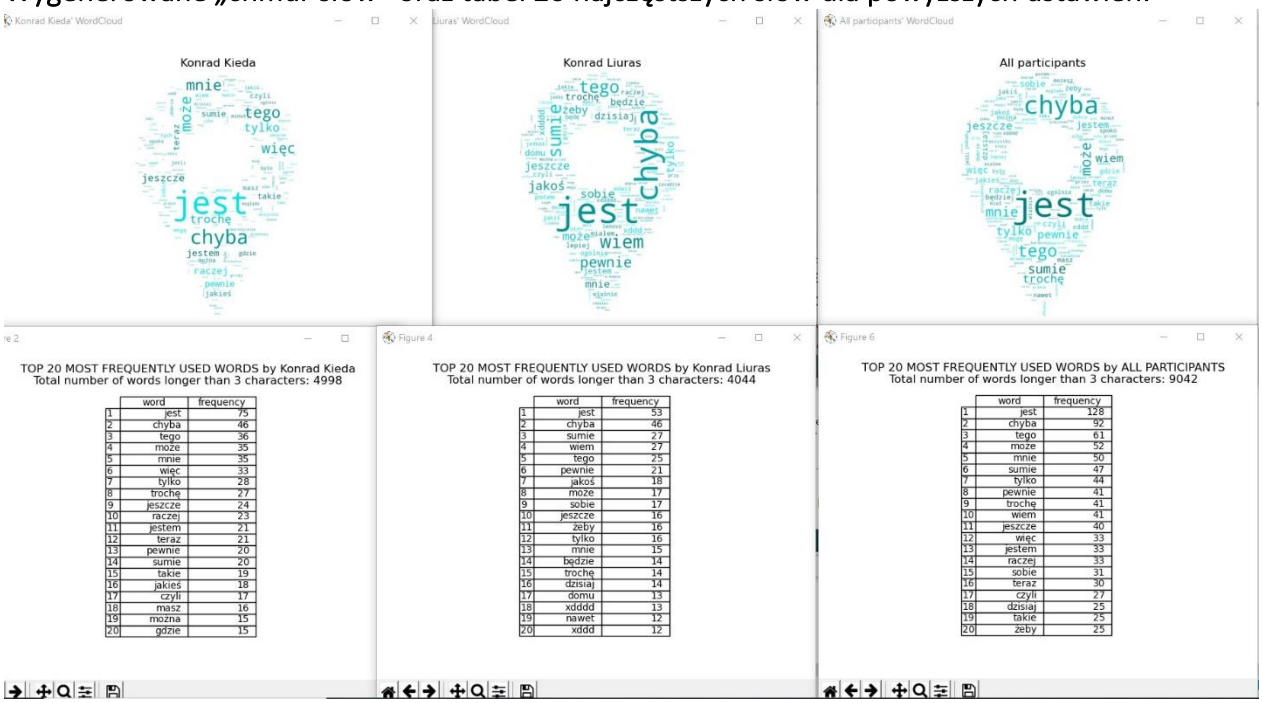
Stan po zmianie szablonu na loc.png:



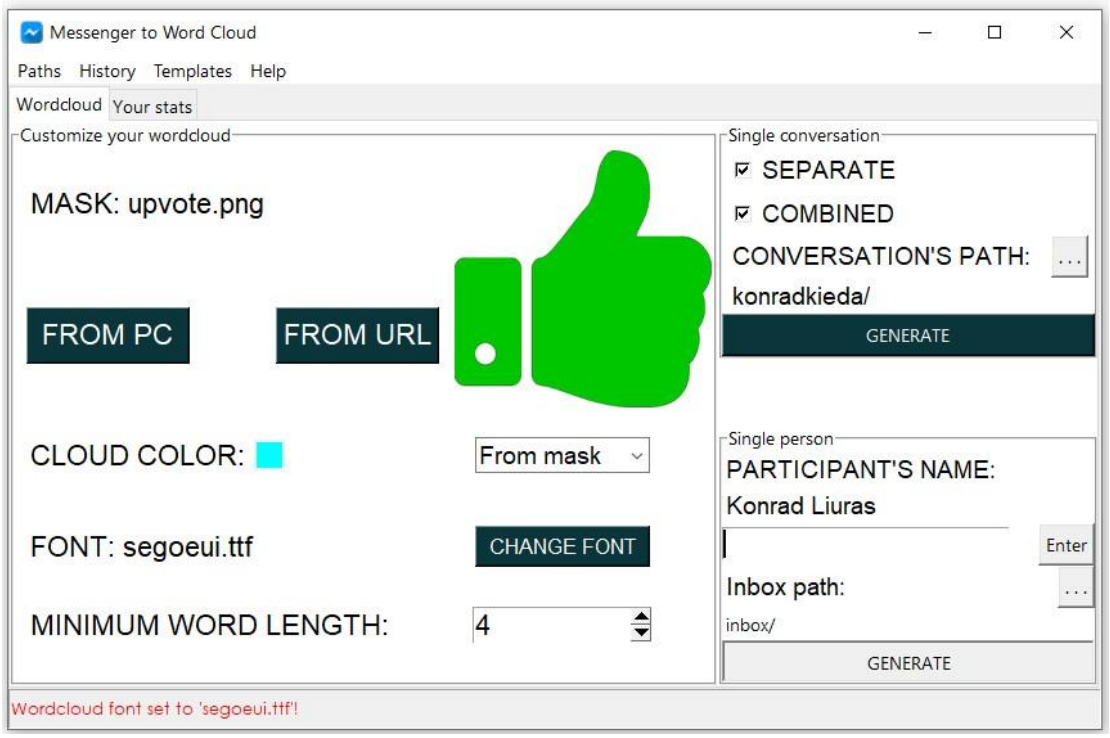
Aplikacja po zmianie ścieżki pojedynczej konwersacji (pasek statusu):



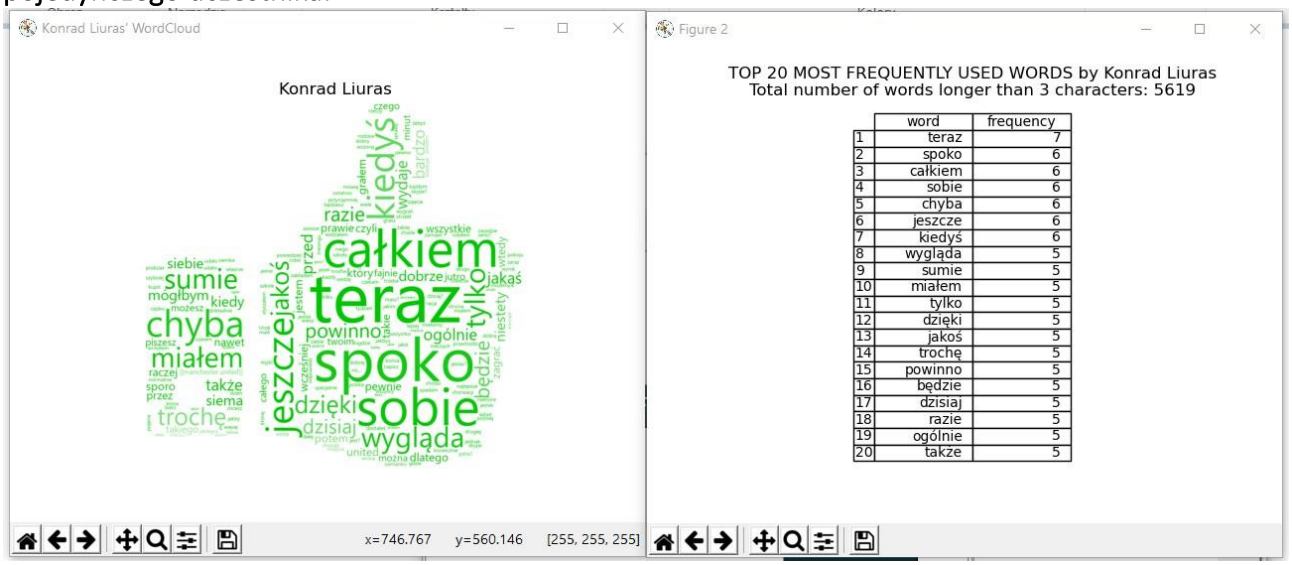
Wygenerowane „chmur słów” oraz tabel 20 najczęstszych słów dla powyższych ustawień:



Aplikacja po zmianie czcionki (pasek statusu):

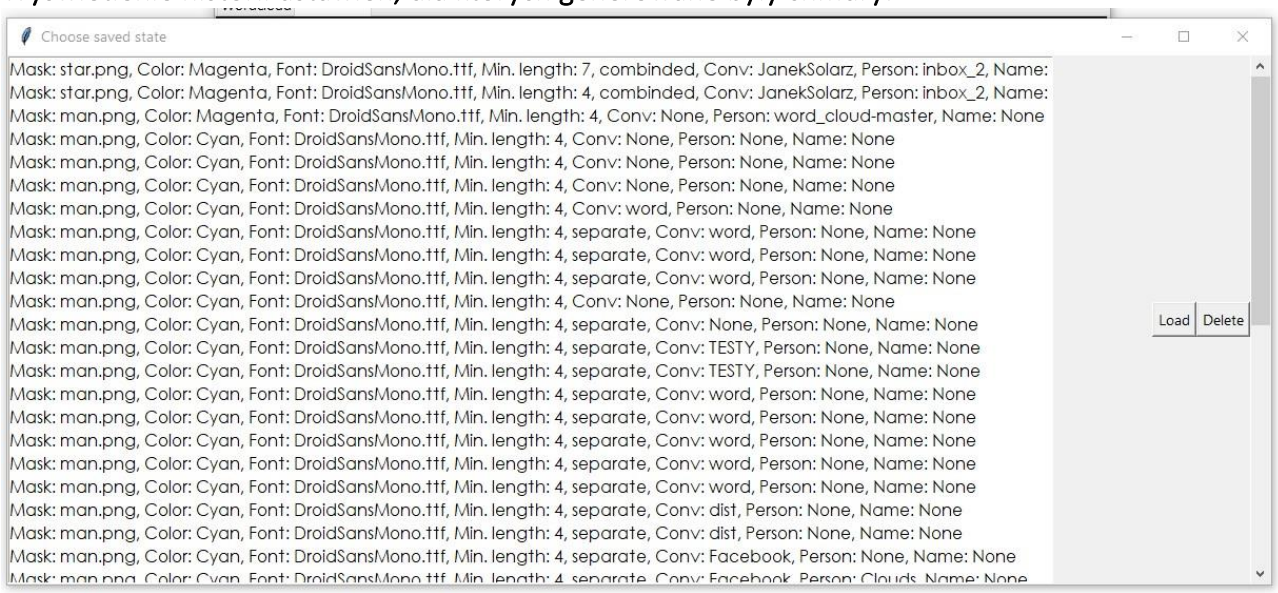


Wygenerowane „chmury słów” oraz tabel 20 najczęstszych słów dla powyższych ustawień dla pojedynczego uczestnika:

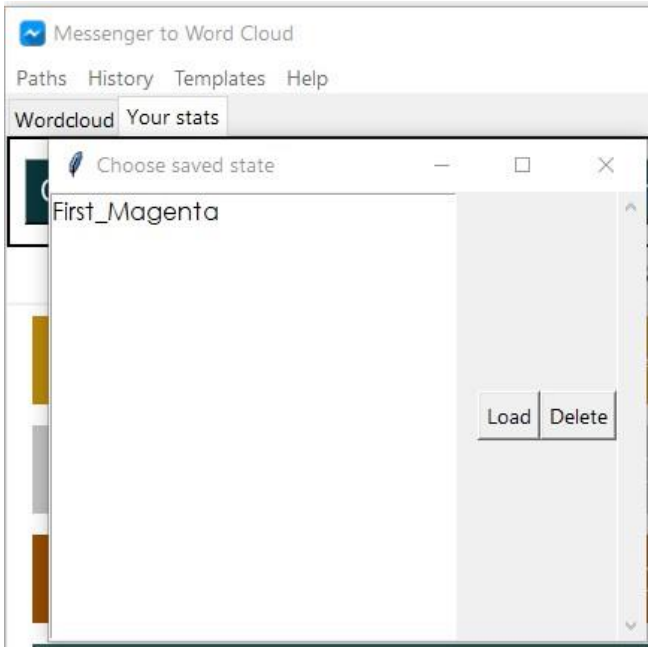


	word	frequency
1	teraz	7
2	spoko	6
3	całkiem	6
4	sobie	6
5	chyba	6
6	jeszcze	6
7	kiedyś	6
8	wygląda	5
9	sumie	5
10	miałem	5
11	tylko	5
12	dzięki	5
13	jakoś	5
14	trochę	5
15	powinno	5
16	będzie	5
17	dzisiaj	5
18	razie	5
19	ogólnie	5
20	takze	5

Wyświetlenie historii ustawień, dla których generowane były chmury:

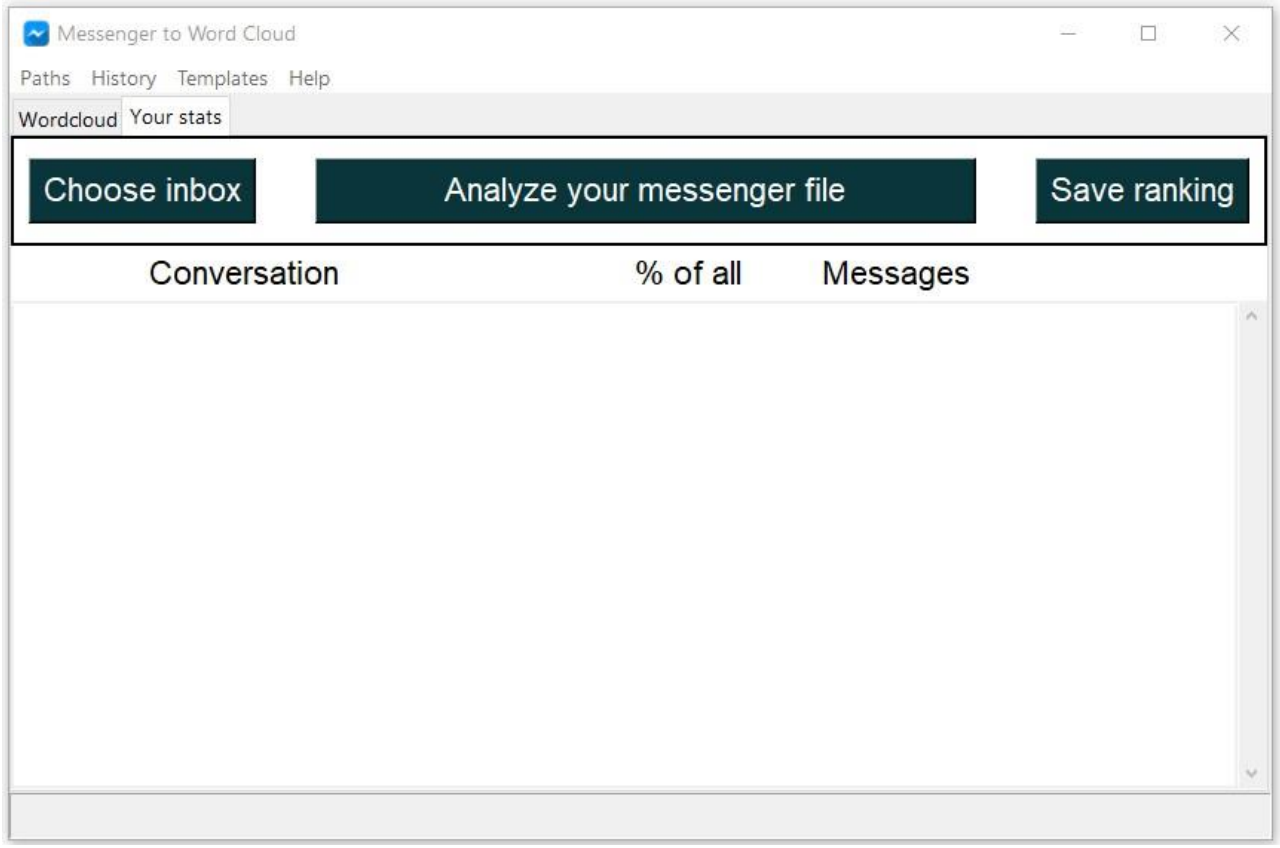


Wyświetlenie zapisanych szablonów ustawień:

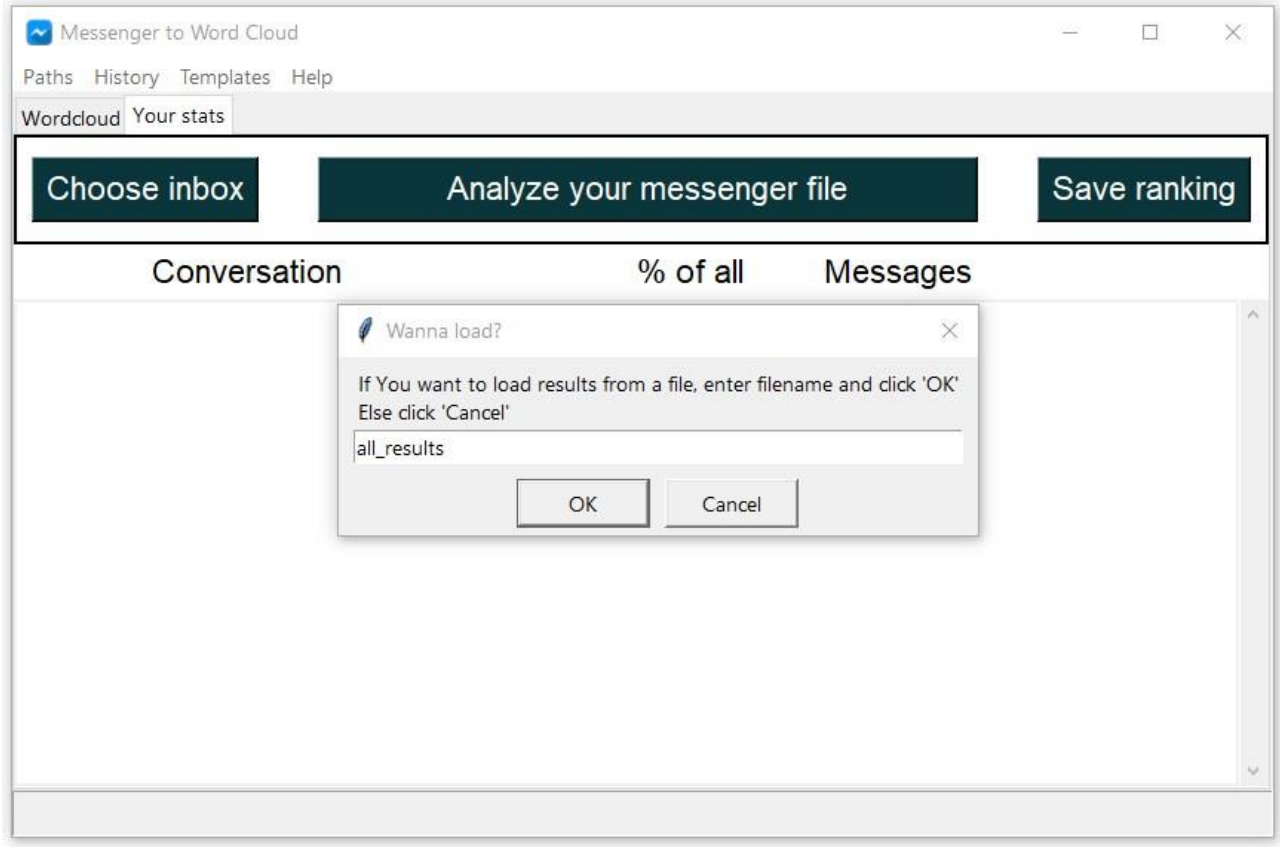


Druga zakładka – Your stats

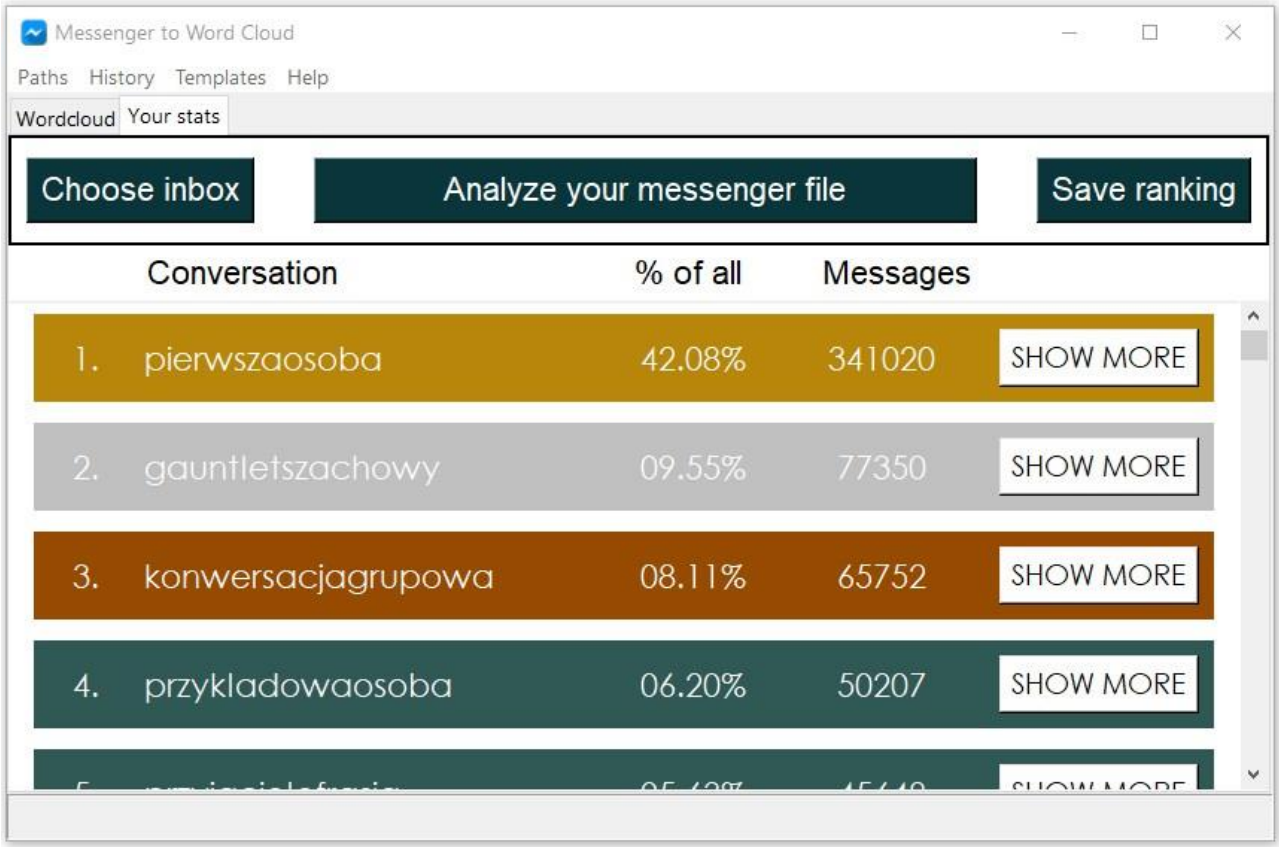
Druga zakładka przed podjęciem na niej jakichkolwiek akcji:



Aplikacja po wybraniu opcji analizy plików Messengera (pojawia się możliwość zapisanej wcześniej analizy):



Wyświetlenie rankingu przeanalizowanych konwersacji:



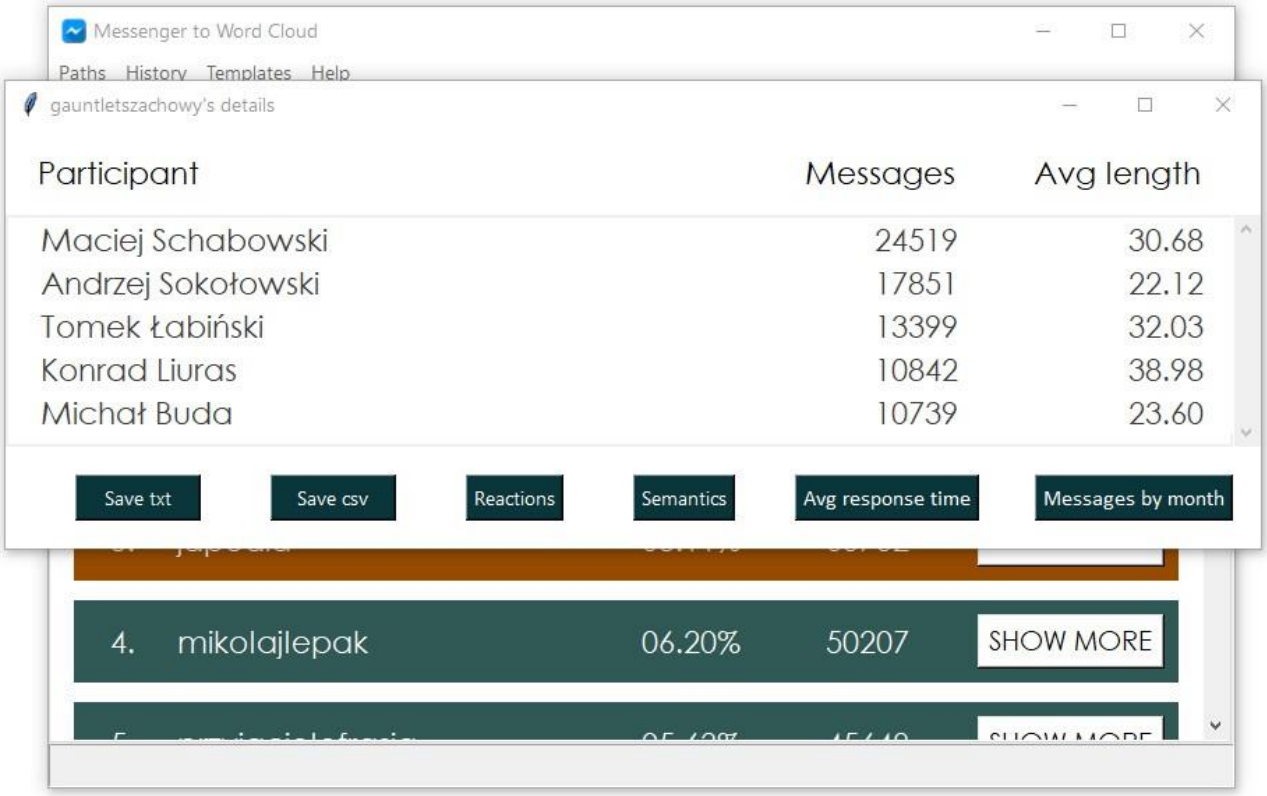
Fragment pliku z zapisanym rankingiem konwersacji (po wciśnięciu przycisku „Save ranking”):

*saved_ranking_test.txt — Notatnik

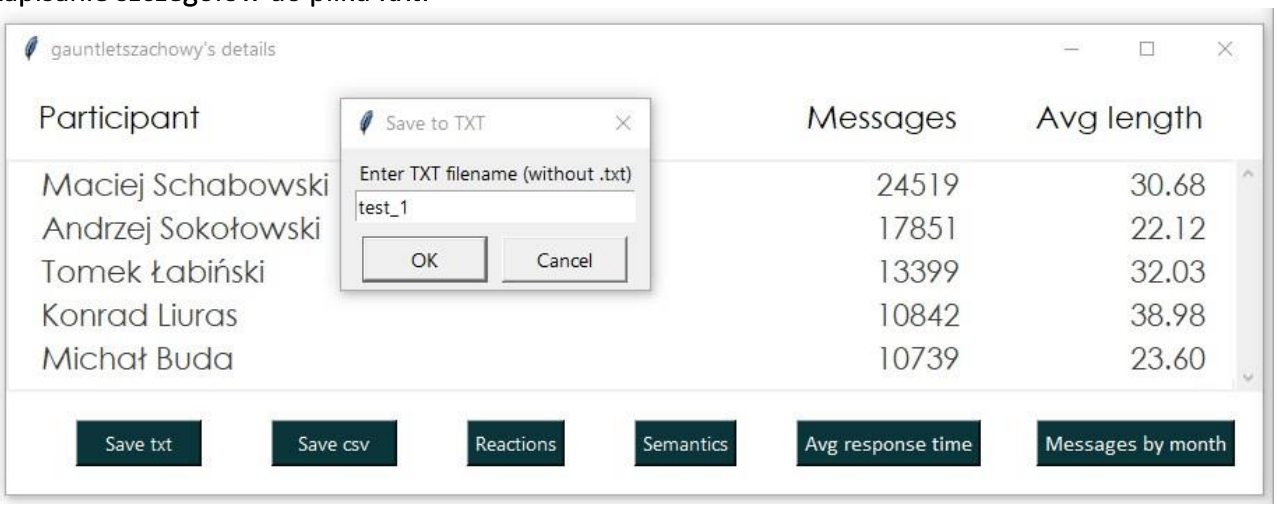
Plik Edycja Format Widok Pomoc

```
Conversation name                                                    |Messages       |% of all messages
-------------------------------------------------------------------------------------
pierwszaosoba                                                        |341020         |42.08%
gauntletszachowy                                                     |77350          |09.55%
konwersacjagrupowa                                                   |65752          |08.11%
przykladowaosoba                                                     |50207          |06.20%
przyjacielefrasia                                                    |45648          |05.63%
```

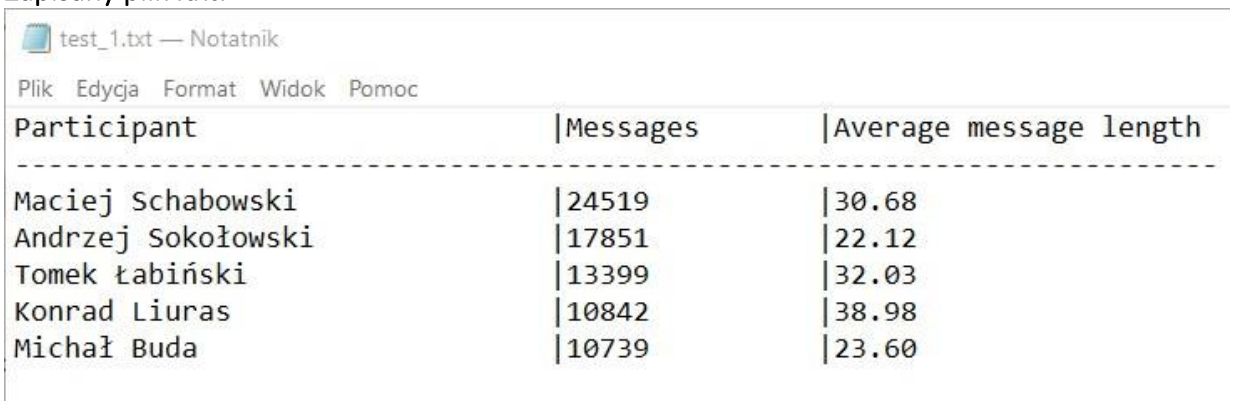
Wyświetlenie szczegółów konwersacji:



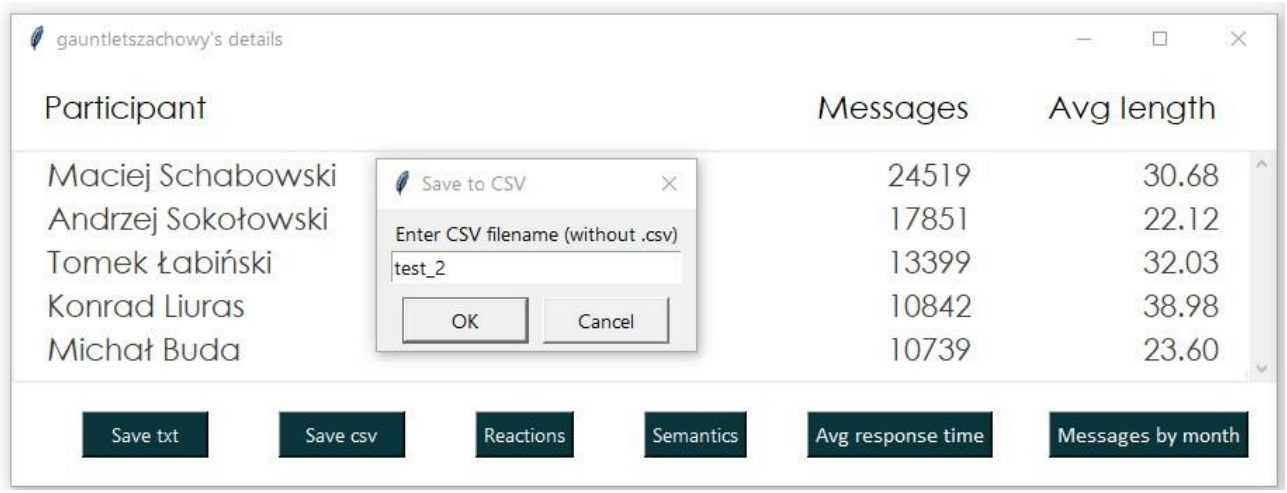
Zapisanie szczegółów do pliku .txt:



Zapisany plik .txt:

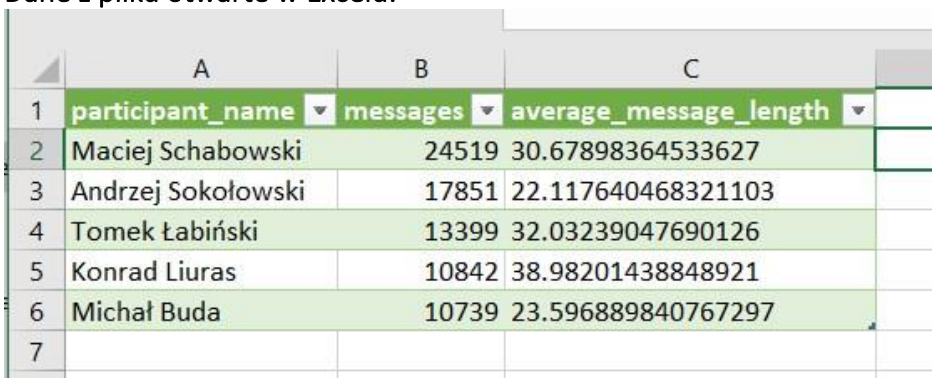


Zapisanie szczegółów do pliku .csv:



Zapisany plik .csv:

Dane z pliku otwarte w Excelu:



	A	B	C
1	participant_name	messages	average_message_length
2	Maciej Schabowski	24519	30.67898364533627
3	Andrzej Sokołowski	17851	22.117640468321103
4	Tomek Łabiński	13399	32.03239047690126
5	Konrad Liuras	10842	38.98201438848921
6	Michał Buda	10739	23.596889840767297
7			

5. Podsumowanie

Podsumowanie wykonania projektu

Projekt spełnia wszystkie założone wymagania funkcjonalne i realizuje wymagania kursu Języki Skryptowe. Aplikacja jest zaimplementowana w języku Python 3, a jej interfejs graficzny zrealizowany jest przy pomocy biblioteki tkinter. Zapamiętywane przez użytkownika dane są przechowywane w plikach „.pkl” w folderze z aplikacją. Raporty wiadomości można wygenerować dla wybranej konwersacji lub dla wszystkich konwersacji. Do dopracowania pozostały części odpowiedzialne za reprezentację listy najczęściej używanych reakcji oraz listy rozkładu wiadomości w ramach pięciu zdefiniowanych klas. Dla rzetelniejszych wyników analizy semantycznej, należałoby też poprawić jakoś modelu, który służy do klasyfikacji wiadomości.

6. Literatura

Dokumentacja Python: https://docs.python.org/3.8/
https://pypi.org/project/pystempel/
https://github.com/dzieciou/pystempel
Wykłady zamieszczone na ePortalu PWr