Analysis of Booking Cancellation Reasons using Python 'spaCy' Library

Imagine that you have an app that schedules telemedicine consultations with doctors. After making a booking, patients have the option to cancel it if they no longer require the services.

When making a cancellation, users are presented with several **cancelled_reason** on the app to select from (Table 1). If they select 'Others', they will have to elaborate in a text box, and this data is stored as a separate column under user_input_reason.

Say for example that on 1 May 2022, your company received the below 20 cancellations (Table 2).
The Business / Operations team wants to know how they can reduce the number of cancellations.

Table 1

id	Default cancelled_reason
1	Waited too long for booking to be confirmed
2	Unsuitable appointment time
3	Found a cheaper telehealth service
4	Emergency ambulance required
5	Made a double/wrong booking
6	Wrong details provided
7	Others (if so, please elaborate in the text box)

Table 2

id	cancelled_reason	user_input_reason
1	Waited too long for booking to be confirmed	
2	Waited too long for booking to be confirmed	
3	Waited too long for booking to be confirmed	
4	Unsuitable appointment time	
5	Found a cheaper telehealth service	
6	Unsuitable appointment time	
7	Found a cheaper telehealth service	
8	Emergency ambulance required	
9	Made a double/wrong booking	
10	Wrong details provided	
11	Others	Visiting clinic
12	Others	visiting a clinic instead
13	Others	I got assigned an earlier timeslot alrrady
14	Others	wrong booking
15	Others	Monitor for another day
16	Others	Sorry wrong date
17	Others	Waited for more than 25 mins and Dr has yet response
18	Others	feeling better already
19	Others	no longer need to see a dr
20	Others	Patient has passed away

Proposed Solution

First, we will need to review the open-ended responses and assess if there is a need to expand the list of default cancelled_reason on the app. If there are only 20 cancellations a day and we do not expect a high volume of open-ended responses, we can sift through the responses manually and either **1) classify them under one of the default cancelled_reason** or **2) create a new category of default cancelled_reason and classify them under that category.**

For example, we can classify the user_input_reason: 'wrong booking' under the cancelled_reason: 'Made a double/wrong booking'. Taking another example, we can create a new category of cancelled_reason: 'Visiting clinic' for the user_input_reason: 'Visiting clinic' and 'visiting a clinic instead'.

However, if the volume of cancellations is huge and it would require a lot of manual effort to sift through the open-ended responses, we can employ the help of **Python spaCy library to extract the most commonly used words/phrases** through rule-based matching (refer to **Annex A** below).

In [14]:
```
#Using Matcher to find phrases with the pattern of adjective->noun, verb->noun, verb->adposition, or verb->adjective
matcher = Matcher(nlp.vocab)
pattern = [{'POS':'ADJ'}, {'POS':'AUX', 'OP':'*'}, {'POS':'ADP', 'OP':'*'}, {'POS':'DET', 'OP':'*'},
           {'POS':'PROPN', 'OP':'*'}, {'POS':'NOUN'}]
pattern2 =  [{'POS':'VERB'}, {'POS':'AUX', 'OP':'*'}, {'POS':'ADP', 'OP':'*'}, {'POS':'DET', 'OP':'*'},
             {'POS':'PROPN', 'OP':'*'}, {'POS':'NOUN'}]
pattern3 =  [{'POS':'VERB'}, {'POS':'ADP'}]
pattern4 =  [{'POS':'VERB'}, {'POS':'ADJ'}]
matcher.add('ADJ_NOUN_PHRASE', [pattern])
matcher.add('VERB__NOUN_PHRASE', [pattern2])
matcher.add('VERB_ADP_PHRASE', [pattern3])
matcher.add('VERB_ADJ_PHRASE', [pattern4])
matches = matcher(doc)
phrases = []
for span in matches:
    phrases.append(doc[span[1]:span[2]])

#Find the most commonly used phrases
phrase_freq = Counter(phrases)
phrase_freq.most_common()
```

Out[14]:
```
[(Visiting clinic, 1),
 (visiting a clinic, 1),
 (earlier timeslot, 1),
 (wrong booking, 1),
 (wrong date, 1),
 (Waited for, 1),
 (passed away, 1)]
```

Thereafter, we can use **Python Regular Expression to sift out those user_input_reason that contains the most commonly used phrases** and re-classify them into existing/new cancelled_reason.

In [18]:

```
#We assess if each user_input_reason contains the most commonly used phrases.
#If so, we can categorise the user_input_reason into existing/new cancelled_reason.

for i in range(10, 20):
    if (re.search('clinic', df2['user_input_reason'][i], flags=re.IGNORECASE)):
        df2['cancelled_reason'][i] = 'Visiting clinic'
    elif (re.search('timeslot', df2['user_input_reason'][i], flags=re.IGNORECASE)):
        df2['cancelled_reason'][i] = 'Assigned another timeslot'
    elif (re.search('wrong booking|wrong date', df2['user_input_reason'][i], flags=re.IGNORECASE)):
        df2['cancelled_reason'][i] = 'Made a double/wrong booking'
    elif (re.search('waited', df2['user_input_reason'][i], flags=re.IGNORECASE)):
        df2['cancelled_reason'][i] = 'Waited too long for booking to be confirmed'
    elif (re.search('passed away', df2['user_input_reason'][i], flags=re.IGNORECASE)):
        df2['cancelled_reason'][i] = 'Patient passed away'
```

In [19]:

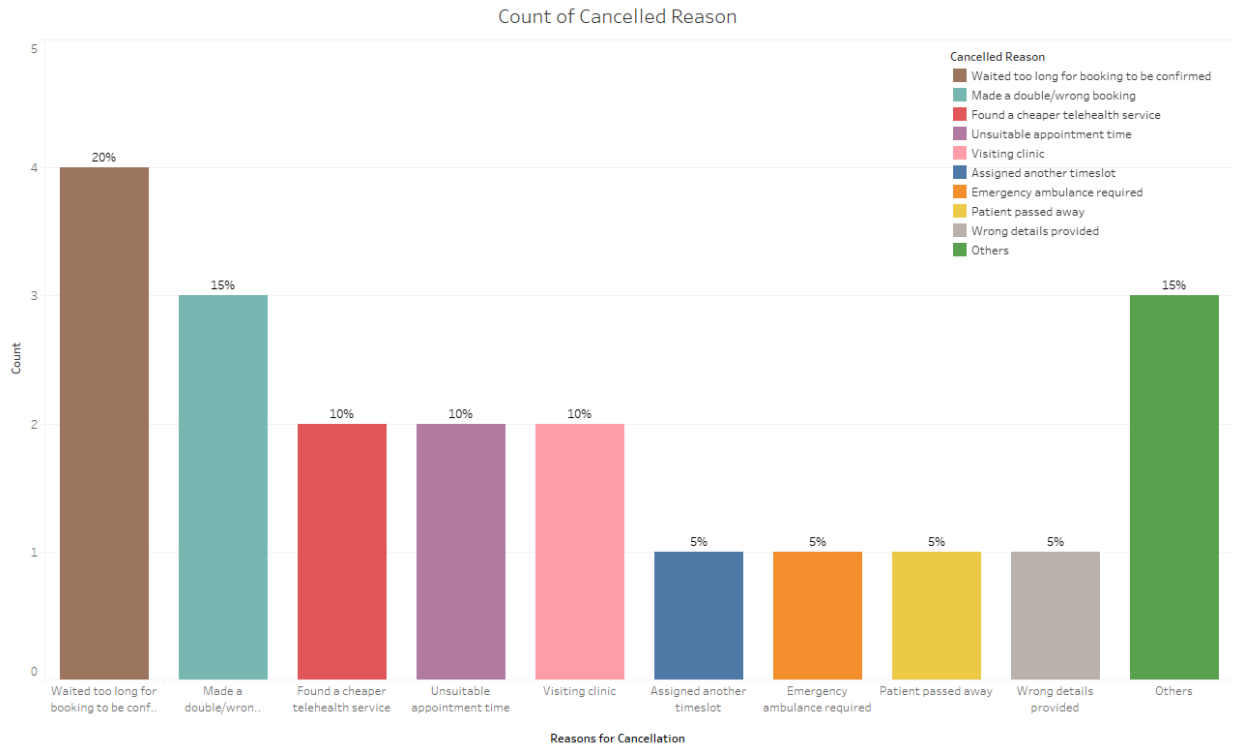
```
df2
```

Out[19]:

	id	cancelled_reason	user_input_reason
0	1	Waited too long for booking to be confirmed	NaN
1	2	Waited too long for booking to be confirmed	NaN
2	3	Waited too long for booking to be confirmed	NaN
3	4	Unsuitable appointment time	NaN
4	5	Found a cheaper telehealth service	NaN
5	6	Unsuitable appointment time	NaN
6	7	Found a cheaper telehealth service	NaN
7	8	Emergency ambulance required	NaN
8	9	Made a double/wrong booking	NaN
9	10	Wrong details provided	NaN
10	11	Visiting clinic	Visiting clinic
11	12	Visiting clinic	visiting a clinic instead
12	13	Assigned another timeslot	I got assigned an earlier timeslot alrrady
13	14	Made a double/wrong booking	wrong booking
14	15	Others	Monitor for another day
15	16	Made a double/wrong booking	Sorry wrong date
16	17	Waited too long for booking to be confirmed	Waited for more than 25 mins and Dr has yet re...
17	18	Others	feeling better already
18	19	Others	no longer need to see a dr
19	20	Patient passed away	Patient has passed away

As for the user_input_reason that we had yet to re-classify (e.g. Monitor for another day, feeling better already, etc.), we can manually go through and re-classify them as the volume would now be much less than before. We can also choose to leave the classification as 'Others' if we feel that the user_input_reason is rare and we do not wish to create a new classification just for that single user input.

We can visualise the most common cancellation reasons using a bar chart and share the common reasons with the Ops team for them to make improvements, where necessary.



As we identify new categories of cancelled_reason, we should remember to **update the default cancelled_reason in the app** so that as time goes by, we should receive less user_input_reason.

For cases where users feedback that they had waited too long for booking to be confirmed, we can **insert another field for users to input their waiting duration** so that we can conduct further analysis. For example, determine the average waiting time before users cancel their booking. This information would be useful for the Ops team to improve on their service level.

	Table 1
id	Default cancelled_reason
1	Waited too long for booking to be confirmed

Annex A

Extract the Most Commonly Used Words/Phrases Using Python Library – SpaCy

In [1]:
```
import numpy as np
import pandas as pd
from collections import Counter
import spacy
nlp = spacy.load('en_core_web_sm')

from spacy.matcher import Matcher
```

In [2]:
```
df = pd.read_excel('Q3_data.xlsx')
```

In [4]:
```
df.tail()
```

Out[4]:

	id	cancelled_reason	user_input_reason
15	16	Others	Sorry wrong date
16	17	Others	Waited for more than 25 mins and Dr has yet re...
17	18	Others	feeling better already
18	19	Others	no longer need to see a dr
19	20	Others	Patient has passed away

In [5]:
```
#For now, we are only concerned with the open-ended responses. Hence, we can drop the other rows

df = df.dropna(subset=['user_input_reason'])
```

In [6]:
```
df
```

Out[6]:

	id	cancelled_reason	user_input_reason
10	11	Others	Visiting clinic
11	12	Others	visiting a clinic instead
12	13	Others	I got assigned an earlier timeslot alrrady
13	14	Others	wrong booking
14	15	Others	Monitor for another day
15	16	Others	Sorry wrong date
16	17	Others	Waited for more than 25 mins and Dr has yet re...
17	18	Others	feeling better already
18	19	Others	no longer need to see a dr
19	20	Others	Patient has passed away

In [7]:
```
#Join all the open-ended responses into a single string for analysis as a whole

text = df.user_input_reason.str.cat(sep = ' ')
```

In [8]:
```
text
```

Out[8]:
```
'Visiting clinic visiting a clinic instead I got assigned an earlier timeslot alrrady\xa0 wrong booking\xa0 Monitor for another day Sorry wrong date Waited for more than 25 mins and Dr has yet response\xa0 feeling better already  no longer need to see a dr Patient has passed away'
```

In [9]:
```
#Process the text with nlp object. This will exclude stop words that do not contain much meaning
#and tag tokens (e.g. words, punctuation) to their part-of-speech (e.g. verb, pronoun, adjectives, etc.).

doc = nlp(text)
```

In [10]:
```
#For each token in the processed text, if token is not a stop word or a punctuation, append its base form into a list.

words = []

for token in doc:
    if (token is not token.is_stop) and (token is not token.is_punct):
        words.append(token.lemma_)
```

In [12]:
```
frequency = Counter(words)
```

In [13]:
```
#Find the most commonly used words

frequency.most_common()
```

Out[13]:
```
[('\xa0 ', 3),
 ('visit', 2),
 ('clinic', 2),
 ('a', 2),
 ('wrong', 2),
 ('for', 2),
 ('have', 2),
 ('instead', 1),
 ('I', 1),
```

In [11]:
```
words
```

Out[11]:
```
['visit',
 'clinic',
 'visit',
 'a',
 'clinic',
 'instead',
 'I',
 'got',
 'assign',
 'an',
 'early',
 'timeslot',
```

In [14]:
```
#Using Matcher to find phrases with the pattern of adjective->noun, verb->noun, verb->adposition, or verb->adjective
matcher = Matcher(nlp.vocab)
pattern = [{'POS':'ADJ'}, {'POS':'AUX', 'OP':'*'}, {'POS':'ADP', 'OP':'*'}, {'POS':'DET', 'OP':'*'},
           {'POS':'PROPN', 'OP':'*'}, {'POS':'NOUN'}]
pattern2 =  [{'POS':'VERB'}, {'POS':'AUX', 'OP':'*'}, {'POS':'ADP', 'OP':'*'}, {'POS':'DET', 'OP':'*'},
             {'POS':'PROPN', 'OP':'*'}, {'POS':'NOUN'}]
pattern3 =  [{'POS':'VERB'}, {'POS':'ADP'}]
pattern4 =  [{'POS':'VERB'}, {'POS':'ADJ'}]
matcher.add('ADJ_NOUN_PHRASE', [pattern])
matcher.add('VERB__NOUN_PHRASE', [pattern2])
matcher.add('VERB_ADP_PHRASE', [pattern3])
matcher.add('VERB_ADJ_PHRASE', [pattern4])
matches = matcher(doc)
phrases = []
for span in matches:
    phrases.append(doc[span[1]:span[2]])

#Find the most commonly used phrases
phrase_freq = Counter(phrases)
phrase_freq.most_common()
```

Out[14]:
```
[(Visiting clinic, 1),
 (visiting a clinic, 1),
 (earlier timeslot, 1),
 (wrong booking, 1),
 (wrong date, 1),
 (Waited for, 1),
 (passed away, 1)]
```

In [15]:
```
phrases
```

Out[15]:
```
[Visiting clinic,
 visiting a clinic,
 earlier timeslot,
 wrong booking,
 wrong date,
 Waited for,
 passed away]
```