Aspect Extraction and Sentiment Analysis of Google Reviews

¶

Part1: Aspect Extraction

In [1]:

```
import nltk
from nltk.tokenize import RegexpTokenizer
from nltk.stem import WordNetLemmatizer,PorterStemmer
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords
import re
from nltk import FreqDist
lemmatizer = WordNetLemmatizer()
import pandas as pd
from textblob import TextBlob
import contractions
```

In [2]:

```
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
from nltk import word_tokenize, pos_tag
from IPython.display import display, HTML
df = pd.read_csv("RGE NY_sellenium.csv")
df = df[df["Review"].notna()]
df
def get_wordnet_pos(tag):
    if tag.startswith('J'):
        return wordnet.ADJ
    elif tag.startswith('V'):
        return wordnet.VERB
    elif tag.startswith('N'):
        return wordnet.NOUN
    elif tag.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN

def lemmatize_passage(text):
    words = word_tokenize(text)
    pos_tags = pos_tag(words)
    lemmatizer = WordNetLemmatizer()
    lemmatized_words = [lemmatizer.lemmatize(word, get_wordnet_pos(tag)) for word, tag in pos_tags]
    lemmatized_sentence = ' '.join(lemmatized_words)
    return lemmatized_sentence

df["Review"] = df["Review"].apply(lambda x:contractions.fix(x))
df["Review"] = df["Review"].apply(lambda x: re.sub(r"\brep\b", "representative", x))
df["Review"] = df["Review"].apply(lambda x: re.sub(r"\bbilling\b", "bill", x))
df["Review"] = df["Review"].apply(lambda x: re.sub(r"\bcompany\b", "rg&e", x))


df["Review"] = df["Review"].apply(lambda x: lemmatize_passage(x))
replacers = {'rge':'RG&E','RGE':'RG&E','rg & e':'RG&E','RG & E':'RG&E','Rge':'RG&E'}
df["Review"] = df["Review"].replace(replacers,regex=True)
display(HTML('<h1><p style="text-align:center">Customer reviews in tabular form</p></h1>'))
#print("Customer reviews in tabular form")
```

```
39 df.head(10)
```

Customer reviews in tabular form

Out[2]:

	User	Date	Review	Likes	Response	Response_Date	Rating
0	haru maru	2024-06-25	Prices be high any time i ask why i be give th...	Like	Hello. We're very sorry to hear about your exp...	a week ago	1
1	The Luxury Spa	2024-05-09	service be horrible I have never see a Utility...	3	Hello. We're very sorry to hear about your exp...	2 months ago	1
2	Sam Peachey	2024-06-09	If you have the option of use RG&E do not . I ...	5	We're very sorry to hear about your experience...	a month ago	1
3	Ms M	2024-06-09	They tell you to enter a meter read .. I do it...	3	We're very sorry to hear about your experience...	a month ago	2
4	Phoebe Hernick	2024-05-09	I schedule a meter reading , and the meter rea...	2	Hi Phoebe. If you need assistance, please cont...	2 months ago	1
5	Rochelle Marie	2024-01-09	This a ZERO star review . Worst RG&E ever . Ho...	16	NaN	NaN	1
6	Cakes Blue	2024-01-09	First three month of living at my new place	7	NaN	NaN	1
7	Nathan Scott	2024-03-09	My bill have go from about $ 80 a month to ove...	2	Hi Nathan. We're very sorry to hear about your...	4 months ago	1
8	Kim Ganley	2024-06-09	RG&E be terribly inefficient . I do not know h...	1	Hi Kim. We're very sorry to hear about your ex...	a month ago	1
9	MEHRI DALAI	2024-03-09	0 star ! This dumb , scamming donkey behind RG...	1	Hi Mehri. We're very sorry to hear about your ...	4 months ago	1

In [3]:

```
display(HTML('<h1><p style="text-align:center">Customer reviews in tabular form</p></h1>'))
display(HTML('<img src="test.png">'))
```

Customer reviews in tabular form

	User	Date	Review	Likes	Response	Response_Date	Rating
0	haru maru	2024-06-25	Prices be high any time i ask why i be give th...	Like	Hello. We're very sorry to hear about your exp...	a week ago	1
1	The Luxury Spa	2024-05-09	service be horrible I have never see a Utility...	3	Hello. We're very sorry to hear about your exp...	2 months ago	1
2	Sam Peachey	2024-06-09	If you have the option of use RG&E do not . I ...	5	We're very sorry to hear about your experience...	a month ago	1
3	Ms M	2024-06-09	They tell you to enter a meter read .. I do it...	3	We're very sorry to hear about your experience...	a month ago	2
4	Phoebe Hernick	2024-05-09	I schedule a meter reading , and the meter rea...	2	Hi Phoebe. If you need assistance, please cont...	2 months ago	1

In [4]:

```
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
def get_sentiment_vader(words):
    analyzer = SentimentIntensityAnalyzer()
    #print("GGGG",words)
    vs = analyzer.polarity_scores(words)

    return vs["compound"]

def get_sentiment_blob(words):
    blob = TextBlob(words)
    return(blob.sentiment.polarity)
```

In [5]:

```python
#!python -m spacy download en_core_web_lg
import nltk
#nltk.download('vader_lexicon')

import spacy
nlp = spacy.load("en_core_web_lg")

from nltk.sentiment.vader import SentimentIntensityAnalyzer
sid = SentimentIntensityAnalyzer()


def find_sentiment(doc):
    # find roots of all entities in the text
    ner_heads = {ent.root.idx: ent for ent in doc.ents}
    #print("AA",doc)
    #print("CCC",doc.ents)
    #print("BB",ner_heads)
    rule3_pairs = []
    for token in doc:
        children = token.children
        A = "999999"
        M = "999999"
        add_neg_pfx = False
        for child in children:
            if(child.dep_ == "nsubj" and not child.is_stop): # nsubj is nominal subject
                if child.idx in ner_heads:
                    A = ner_heads[child.idx].text
                else:
                    A = child.text
            if(child.dep_ == "acomp" and not child.is_stop): # acomp is adjectival complement
                M = child.text
            # example - 'this could have been better' -> (this, not better)
            if(child.dep_ == "aux" and child.tag_ == "MD"): # MD is modal auxiliary
                neg_prefix = "not"
                add_neg_pfx = True
            if(child.dep_ == "neg"): # neg is negation
                neg_prefix = child.text
                add_neg_pfx = True
        if (add_neg_pfx and M != "999999"):
            M = neg_prefix + " " + M
        if(A != "999999" and M != "999999"):
            #print("AA",doc)
            doc = str(doc)
```

```
            phrase = doc[doc.find(A):doc.find(M)]+M
            #print("PPP",phrase)
            #print("MM1",sid.polarity_scores(phrase)['compound'],sid.polarity_scores(M)['compound'],get_s
            #print("MM2",M,sid.polarity_scores(M)['compound'],get_sentiment_vader(M))
            #print(A,M)
            #print("*****************************************************************")
            if sid.polarity_scores(phrase)['compound']<0:
                sentiment = sid.polarity_scores(phrase)['compound']
            if sid.polarity_scores(M)['compound']<0:
                sentiment = sid.polarity_scores(M)['compound']
            if get_sentiment_vader(phrase)<0:
                sentiment = get_sentiment_vader(phrase)
            if get_sentiment_vader(M)<0:
                sentiment = get_sentiment_vader(M)
            else:
                sentiment = sid.polarity_scores(phrase)['compound']
            rule3_pairs.append((A, M, sentiment))
    return rule3_pairs

```

In [6]:

```
from nltk.tokenize import sent_tokenize
import numpy as np
aspect_data = {}
for i in df["Review"]:
    sent_tok = sent_tokenize(i)
    #print(sent_tok)
    for j in sent_tok:
        aspect = find_sentiment(nlp(j))
        if len(aspect)>0:
            #print(aspect)
            for i in range(len(aspect)):
                if aspect[i][0].lower() in aspect_data:
                    aspect_data[aspect[i][0].lower()] +=[aspect[i][1].lower()]
                else:
                    aspect_data[aspect[i][0].lower()] =[aspect[i][1].lower()]


#aspect_data
```

In [7]:

```
aspect_datas = sorted(aspect_data.items(), key= lambda x: len(x[1]), reverse=True)
#print(aspect_datas)
data = []
aspect_term =[]
from collections import Counter
for i in aspect_datas:
    #print(i[0])
    aspect_term.append(i[0])
    data.append(Counter(i[1]))
    #print(Counter(i[1]))
    #print(list(Counter(i[1]).keys()))
    #print('****************************')
```

Top Aspect phrases used by customers in the review:

1. RGE

2. Service

3. Bill

4. Representative

In [8]:

```
config = {
  'toImageButtonOptions': {
    'format': 'svg', # one of png, svg, jpeg, webp
    'filename': 'custom_image',
    'height': 500,
    'width': 700,
    'scale': 5 # Multiply title/legend/axis/canvas sizes by this factor
  },'modeBarButtonsToAdd': ['drawline',
                                        'drawopenpath',
                                        'drawclosedpath',
                                        'drawcircle',
                                        'drawrect',
                                        'eraseshape'
                                       ]}

```

In [9]:

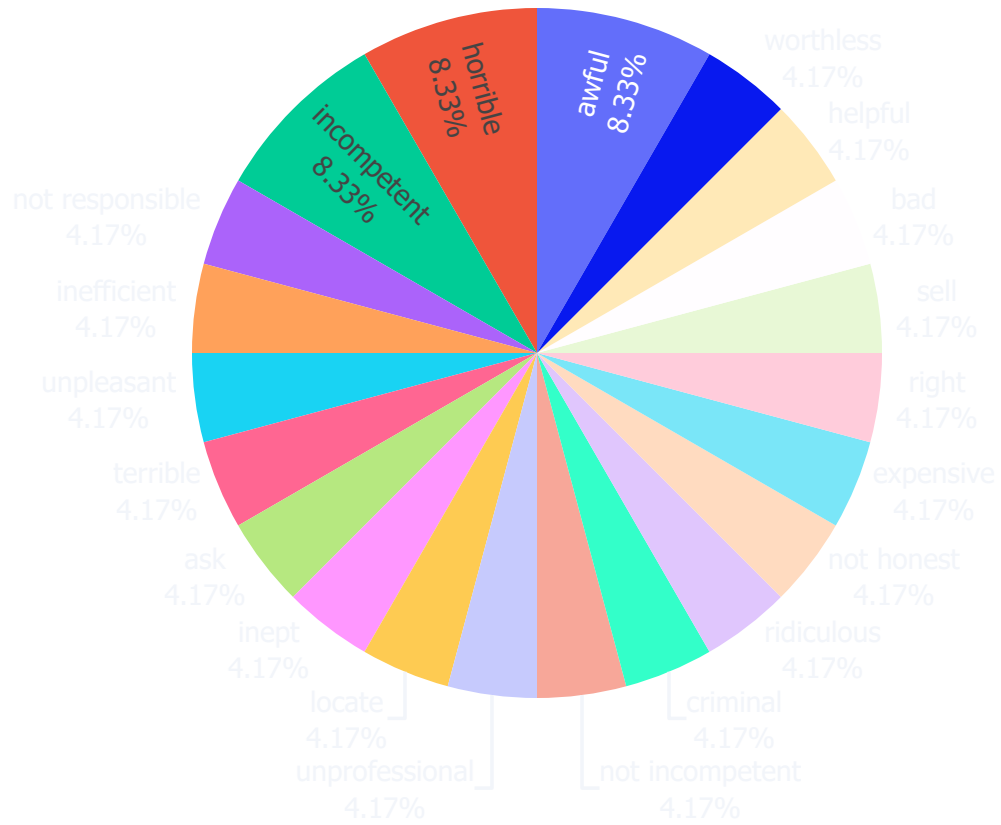
```
import plotly.express as px
import plotly.graph_objects as go
import matplotlib.pyplot as plt

k = 0 #defines which index in the list to plot
label = list(data[k].keys())
#rep = representative = {'friendly': 2, 'unprofessional': 2, 'able': 1, 'rude': 3, 'not helpful': 2, 'hum
val = list(data[k].values())
#label = list(rep.keys())
#val = list(rep.values())

#print(label)
val
fig = go.Figure(data=[go.Pie(labels=label, values=val, textinfo='label+percent',insidetextorientation='ra
fig.update_layout(legend=dict({'traceorder': 'normal'}),
                  legend_title_text="Description of term "+'"'+aspect_term[k].capitalize()+'"',
                  title ="Descriptions of the term "+'"'+aspect_term[k].capitalize()+'"')
fig.update_layout(template='plotly_dark',title_x=0.5)
fig.update_layout(showlegend=False)
fig.update_layout(title_text="Descriptions used for the term <span style='color:gold'>%s </span>"%aspect_
fig.update_layout(
    font_family="tahoma",
    font_size=14,
    legend_title_font_color="green"
)
#fig.write_image("name.svg")
print("EXAMPLE")
fig.show(config=config)
```

EXAMPLE

Descriptions used for the term RG&E

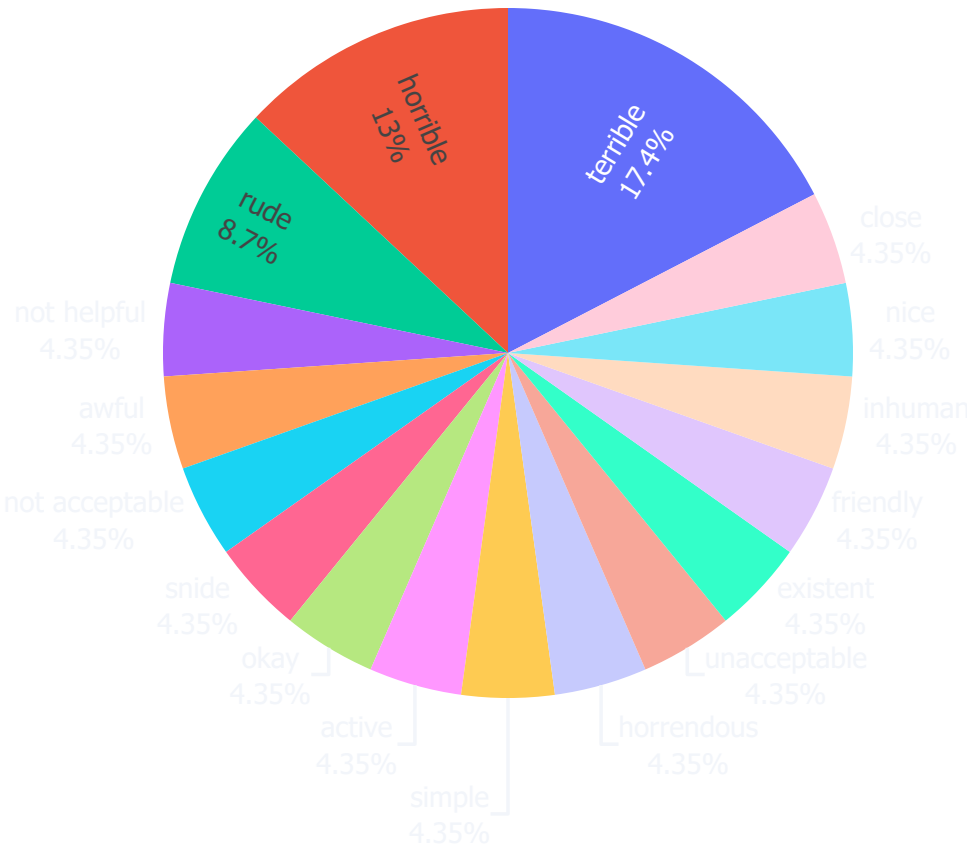


In [10]:

```
k = 1 #defines which index in the list to plot
label = list(data[k].keys())
#rep = representative = {'friendly': 2, 'unprofessional': 2, 'able': 1, 'rude': 3, 'not helpful': 2, 'hum
val = list(data[k].values())
#label = list(rep.keys())
#val = list(rep.values())

#print(label)
val
fig = go.Figure(data=[go.Pie(labels=label, values=val, textinfo='label+percent',insidetextorientation='ra
fig.update_layout(legend=dict({'traceorder': 'normal'}),
                  legend_title_text="Description of term "+'"'+aspect_term[k].capitalize()+'"',
                  title ="Descriptions of the term "+'"'+aspect_term[k].capitalize()+'"')
fig.update_layout(template='plotly_dark',title_x=0.5)
fig.update_layout(showlegend=False)
fig.update_layout(title_text="Descriptions used for the term <span style='color:gold'>%s </span>"%aspect_
fig.update_layout(
    font_family="tahoma",
    font_size=14,
    legend_title_font_color="green"
)
#fig.write_image("name.svg")
fig.show(config=config)
```

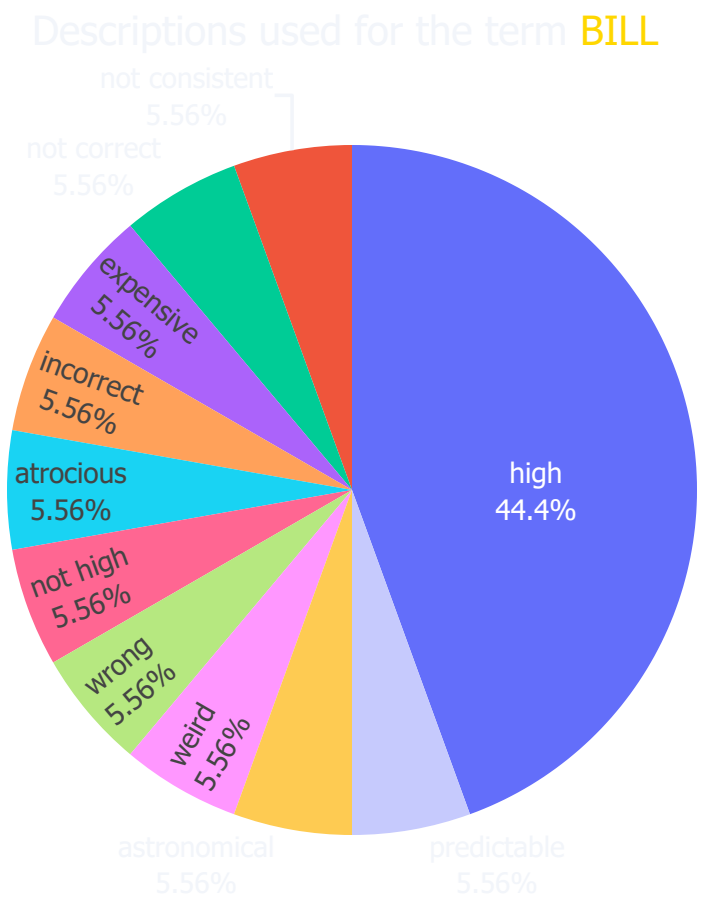
Descriptions used for the term SERVICE



In [11]:

```
k = 2 #defines which index in the list to plot
label = list(data[k].keys())
#rep = representative = {'friendly': 2, 'unprofessional': 2, 'able': 1, 'rude': 3, 'not helpful': 2, 'hum
val = list(data[k].values())
#label = list(rep.keys())
#val = list(rep.values())

#print(label)
val
fig = go.Figure(data=[go.Pie(labels=label, values=val, textinfo='label+percent',insidetextorientation='ra
fig.update_layout(legend=dict({'traceorder': 'normal'}),
                  legend_title_text="Description of term "+'"'+aspect_term[k].capitalize()+'"',
                  title ="Descriptions of the term "+'"'+aspect_term[k].capitalize()+'"')
fig.update_layout(template='plotly_dark',title_x=0.5)
fig.update_layout(showlegend=False)
fig.update_layout(title_text="Descriptions used for the term <span style='color:gold'>%s </span>"%aspect_
fig.update_layout(
    font_family="tahoma",
    font_size=14,
    legend_title_font_color="green"
)
#fig.write_image("name.svg")
fig.show(config=config)
```


Descriptions used for the term BILL
not consistent
5.56%
not correct
5.56%
expensive
5.56%
incorrect
5.56%
atrocious
5.56%
not high
5.56%
wrong
5.56%
weird
5.56%
high
44.4%
astronomical
5.56%
predictable
5.56%

In [12]:

```
k = 3 #defines which index in the list to plot
label = list(data[k].keys())
#rep = representative = {'friendly': 2, 'unprofessional': 2, 'able': 1, 'rude': 3, 'not helpful': 2, 'hum
val = list(data[k].values())
#label = list(rep.keys())
#val = list(rep.values())

#print(label)
val
fig = go.Figure(data=[go.Pie(labels=label, values=val, textinfo='label+percent',insidetextorientation='ra
fig.update_layout(legend=dict({'traceorder': 'normal'}),
                  legend_title_text="Description of term "+'"'+aspect_term[k].capitalize()+'"',
                  title ="Descriptions of the term "+'"'+aspect_term[k].capitalize()+'"')
fig.update_layout(template='plotly_dark',title_x=0.5)
fig.update_layout(showlegend=False)
fig.update_layout(title_text="Descriptions used for the term <span style='color:gold'>%s </span>"%aspect_
fig.update_layout(
    font_family="tahoma",
    font_size=14,
    legend_title_font_color="green"
)
#fig.write_image("name.svg")
fig.show(config=config)
```

Descriptions used for the term REPRESENTATIVE

Summary: the descriptions used for each term shows that customers are unhappy.

Aspect Sentiment analyzer

In [13]:

```
from nltk.tokenize import sent_tokenize
import numpy as np
aspect_sentiment = {}
for i in df["Review"]:
    sent_tok = sent_tokenize(i)
    #print(sent_tok)
    for j in sent_tok:
        aspect = find_sentiment(nlp(j))
        if len(aspect)>0:
            #print(aspect)
            for i in range(len(aspect)):
                #print(aspect[i][2])
                if aspect[i][0].lower() in aspect_sentiment:
                    #print("HERE1")
                    #print(aspect[i][2])
                    if float(aspect[i][2])>0:
                        #print("HERE2")
                        aspect_sentiment[aspect[i][0].lower()] +=["Positive"]
                    if float(aspect[i][2])<0:
                        aspect_sentiment[aspect[i][0].lower()] +=["Negative"]
                    else:
                        aspect_sentiment[aspect[i][0].lower()] +=["Neutral"]
                else:
                    if float(aspect[i][2])>0:
                        aspect_sentiment[aspect[i][0].lower()] =["Positive"]
                    if float(aspect[i][2])<0:
                        aspect_sentiment[aspect[i][0].lower()] =["Negative"]
                    else:
                        aspect_sentiment[aspect[i][0].lower()] =["Neutral"]


#aspect_sentiment
```

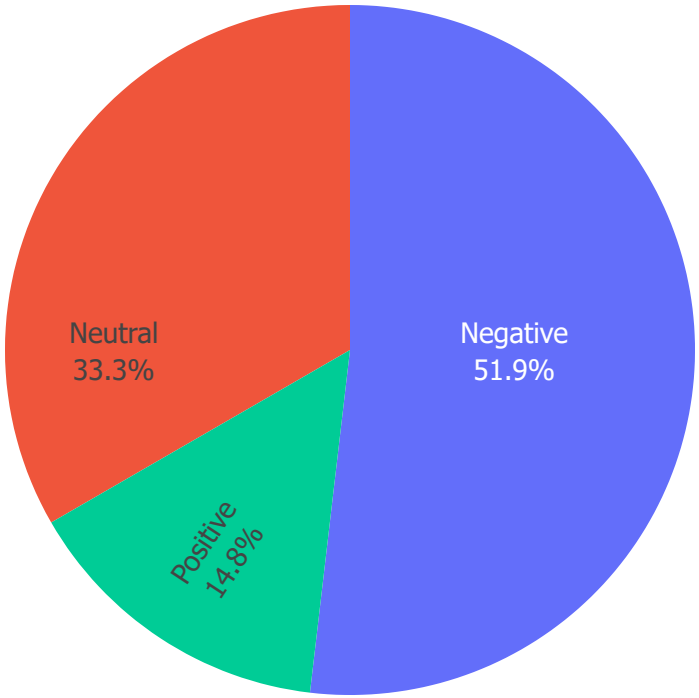
In [14]:

```
aspect_datas = sorted(aspect_sentiment.items(), key= lambda x: len(x[1]), reverse=True)
#print(aspect_datas)
data = []
aspect_term =[]
from collections import Counter
for i in aspect_datas:
    #print(i[0])
    aspect_term.append(i[0])
    data.append(Counter(i[1]))
    #print(Counter(i[1]))
    #print(list(Counter(i[1]).keys()))
    #print('****************************')
```

In [15]:

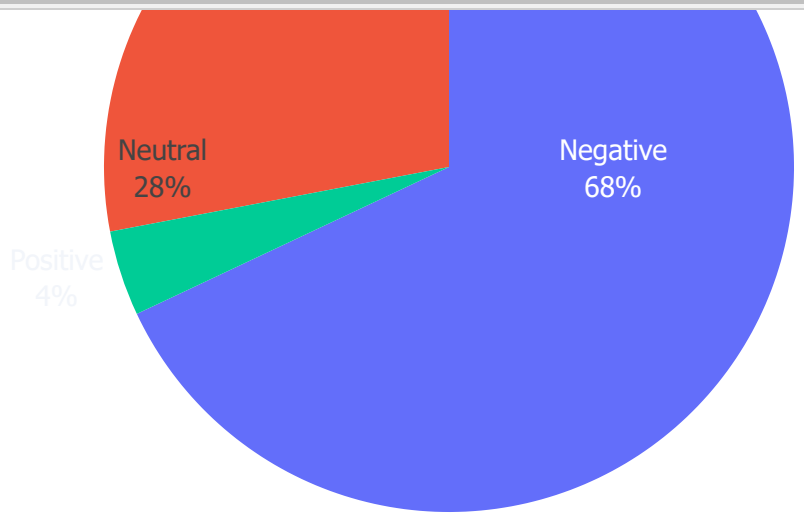
```
k = 0 #defines which index in the list to plot
label = list(data[k].keys())
val = list(data[k].values())
#print(label)
val
fig = go.Figure(data=[go.Pie(labels=label, values=val, textinfo='label+percent',insidetextorientation='ra
fig.update_layout(legend=dict({'traceorder': 'normal'}),legend_title_text='"'+aspect_term[k].capitalize()
fig.update_layout(template='plotly_dark',title_x=0.5)
fig.update_layout(showlegend=False)
fig.update_layout(title_text="Sentiments associated with the term <span style='color:gold'>%s </span>"%as
fig.update_layout(
    font_family="tahoma",
    font_size=14,
    legend_title_font_color="green"
)
#fig.write_image("name.svg")
fig.show(config=config)
```

Sentiments associated with the term SERVICE



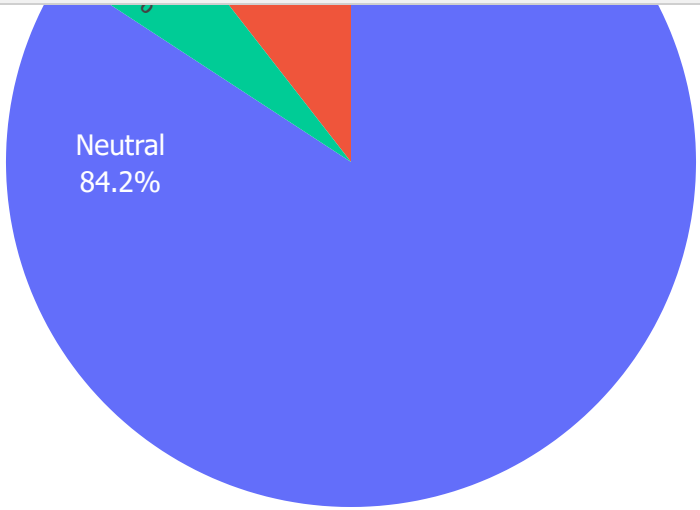
In [16]:

```
k = 1 #defines which index in the list to plot
label = list(data[k].keys())
val = list(data[k].values())
#print(label)
val
fig = go.Figure(data=[go.Pie(labels=label, values=val, textinfo='label+percent',insidetextorientation='ra
fig.update_layout(legend=dict({'traceorder': 'normal'}),legend_title_text='"'+aspect_term[k].capitalize()
fig.update_layout(template='plotly_dark',title_x=0.5)
fig.update_layout(showlegend=False)
fig.update_layout(title_text="Sentiments associated with the term <span style='color:gold'>%s </span>"%as
fig.update_layout(
    font_family="tahoma",
    font_size=14,
    legend_title_font_color="green"
)
#fig.write_image("name.svg")
fig.show(config=config)
```



In [17]:

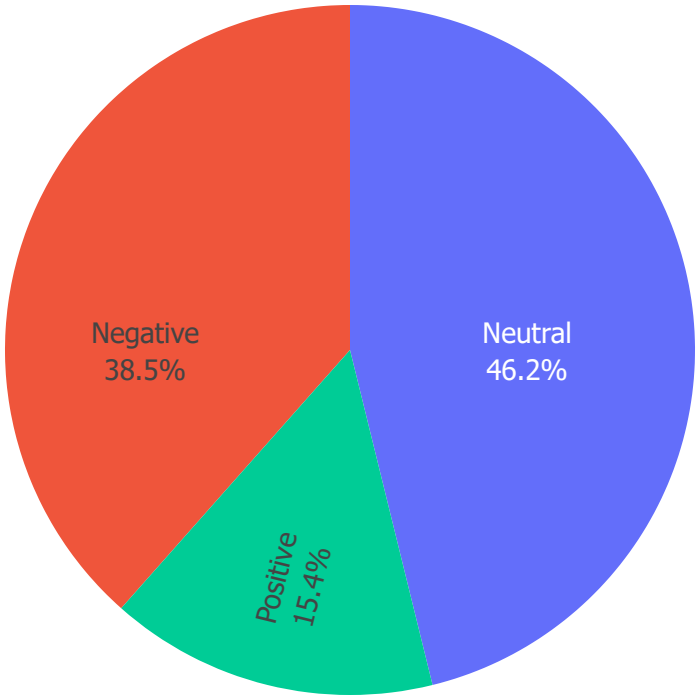
```
k = 2 #defines which index in the list to plot
label = list(data[k].keys())
val = list(data[k].values())
#print(label)
val
fig = go.Figure(data=[go.Pie(labels=label, values=val, textinfo='label+percent',insidetextorientation='ra
fig.update_layout(legend=dict({'traceorder': 'normal'}),
                  legend_title_text="Description of term "+'"'+aspect_term[k].capitalize()+'"',
                  title ="Descriptions of the term "+'"'+aspect_term[k].capitalize()+'"')
fig.update_layout(template='plotly_dark',title_x=0.5)
fig.update_layout(showlegend=False)
fig.update_layout(title_text="Distribution of the sentiments used for the term  <span style='color:gold'>
fig.update_layout(
    font_family="tahoma",
    font_size=14,
    legend_title_font_color="green"
)
#fig.write_image("name.svg")
fig.show(config=config)
```



In [18]:

```
k = 3 #defines which index in the list to plot
label = list(data[k].keys())
val = list(data[k].values())
#print(label)
val
fig = go.Figure(data=[go.Pie(labels=label, values=val, textinfo='label+percent',insidetextorientation='ra
fig.update_layout(legend=dict({'traceorder': 'normal'}),legend_title_text='"'+aspect_term[k].capitalize()
fig.update_layout(template='plotly_dark',title_x=0.5)
fig.update_layout(showlegend=False)
fig.update_layout(title_text="Sentiments associated with the term <span style='color:gold'>%s </span>"%as
fig.update_layout(
    font_family="tahoma",
    font_size=14,
    legend_title_font_color="green"
)
#fig.write_image("name.svg")
fig.show(config=config)
```

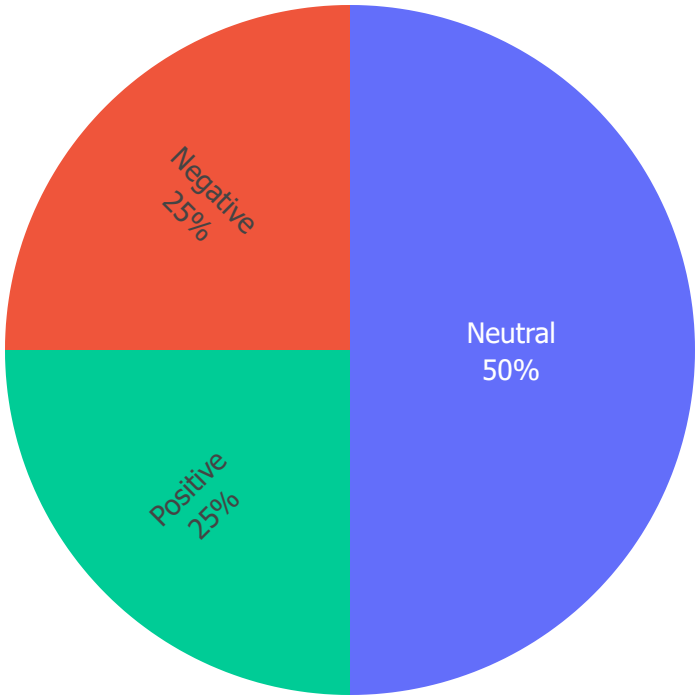
Sentiments associated with the term REPRESENTATIVE



In [19]:

```
k = 4 #defines which index in the list to plot
label = list(data[k].keys())
val = list(data[k].values())
#print(label)
val
fig = go.Figure(data=[go.Pie(labels=label, values=val, textinfo='label+percent',insidetextorientation='ra
fig.update_layout(legend=dict({'traceorder': 'normal'}),legend_title_text='"'+aspect_term[k].capitalize()
fig.update_layout(template='plotly_dark',title_x=0.5)
fig.update_layout(showlegend=False)
fig.update_layout(title_text="Sentiments associated with the term <span style='color:gold'>%s </span>"%as
fig.update_layout(
    font_family="tahoma",
    font_size=14,
    legend_title_font_color="green"
)
#fig.write_image("name.svg")
fig.show(config=config)
```

Sentiments associated with the term PEOPLE



Summary:

Sentiments associated with each aspect shows that there are more negative sentiments than positive sentiments for

In [25]:
```
1 !jupyter nbconvert --to slides  --no-input Aspect_based_sentiment_analyzer_using_multi_approach.ipynb
```

```
[NbConvertApp] Converting notebook Aspect_based_sentiment_analyzer_using_multi_approach.ipynb to slides
[NbConvertApp] Writing 4349003 bytes to Aspect_based_sentiment_analyzer_using_multi_approach.slides.html
```