```
# Visualization Libraries
from matplotlib.backends.backend_agg import FigureCanvasAgg as
FigureCanvas
from flask import Flask, render_template, request, send_file, Markup,
Response
import matplotlib.pyplot as plt
import seaborn as sns

# Sentiment Analysis
from textblob import TextBlob

# Scraping and data processing
from bs4 import BeautifulSoup
import pandas as pd

# Built-ins
import requests
import io
import time



#https://www.w3schools.com/python/gloss_python_global_variables.asp
#global variable resources
COMP_NAME = ''


# Keep Master Data_Frame stored in memory
DATA_FRAME = None




#help from Dr. Diederich and this tutorial
#https://www.tutorialspoint.com/flask/index.html

# Instantiate app
#builds web app all conection etc
app = Flask(__name__)
```

```
def scrape_reviews(url_template, tags):
    """ a function to scrape the reviews """
    global COMP_NAME
    global DATA_FRAME


    COMP_NAME = tags['company_name']



    # user agent from
https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/User-Agent/Firef
ox
    # need this to excess any webpage
    headers = {
        "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
AppleWebKit/605.1.15 (KHTML, like Gecko) Version/15.6.1 Safari/605.1.15"
    }



#https://stackoverflow.com/questions/72425082/scraping-reviews-from-multip
le-pages-in-r
    # creating the URL template based on the user input on index.html page
    site = ''
    if 'tripadvisor.com' in url_template:
        url_template = url_template + "-or{}-"
        site = 'trip'
    elif 'trustpilot.com' in url_template:
        url_template = url_template + '?page={}&sort=recency'
        site = 'trust'


    def page_counting(page_number, site):
        """ A function to help with page counting (different for each
website)"""

        if site == 'trip':
```

```
            return (page_number - 1) * 10
        elif site == 'trust':
            return page_number


    all_reviews_scraped = False
    page_number = 1

    # Initialize resulting list for data
    res=[]


    while not all_reviews_scraped:

        # Wait 3 seconds to prevent IP ban
        time.sleep(3)



        url = url_template.format(page_counting(page_number, site))
       # make a request to the website
        response = requests.get(url, headers=headers)
        html_content = response.text

        # BeautifulSoup only works with static webpages
        # https://www.zenrows.com/blog/selenium-vs-beautifulsoup

        # parse the HTML content with BeautifulSoup
        soup = BeautifulSoup(html_content, "html.parser")

        for box in soup.find_all(tags['box']['tag'],
tags['box'][list(tags['box'])[1]]):

            review_date = ''.join(filter(str.isdigit,
attribute_error_check(box.find(tags['dates']['tag'],
tags['dates'][list(tags['dates'])[1]])))) # more universal, allows me to
remove all text from this specific tag and only have the date


            if tags['yearorpage'] == 'by Year':
```

```
                if tags['years_scrapedto'] not in review_date:
                    review_text =
attribute_error_check(box.find(tags['reviews']['tag'],
tags['reviews'][list(tags['reviews'])[1]]))

                    res.append({"review date": review_date, "review text":
review_text})

                else:
                    all_reviews_scraped = True

            elif tags['yearorpage'] == 'by Pages':
                if int(tags['pages']) != page_number:
                    review_text =
attribute_error_check(box.find(tags['reviews']['tag'],
tags['reviews'][list(tags['reviews'])[1]]))

                    res.append({"review date": review_date, "review text":
review_text})

                else:
                    all_reviews_scraped = True


        # increment page number
        page_number += 1

    #creates dataframe to store in Global Variable
    DATA_FRAME = sentiment_processing(pd.DataFrame(res))



#https://sentry.io/answers/redirect-to-a-url-in-flask/#:~:text=Redirection
%20in%20Flask%20can%20be,same%20application%20and%20external%20websites.co
m

@app.route('/')
def index():
```

```
   print("PLZ")
   return render_template('index.html')

#connecting other html pages to flask
@app.route('/creator')
def creator():
    print("test")
    return render_template('creator.html')

#connecting other html pages to flask
@app.route('/research')
def research():
    print("test")
    return render_template('research.html')

#connecting other html pages to flask
@app.route('/contact')
def contact():
    print("test")
    return render_template('contact.html')


#https://www.geeksforgeeks.org/python-attributeerror/
#https://stackoverflow.com/questions/48838562/exception-handling-when-usin
g-pandas-apply
def attribute_error_check(fun):
    """ fun is a soup tag that has text, if there is no tag (i.e. does not
exist) it will return an AtrributeError thus a 'nan' value,
      otherwise, the text get's returned
    """

    try:
        return fun.text
    except AttributeError:
        return 'nan'


@app.route('/scraped', methods=['POST'])
def scraped():
    # Data frame stored in memory
```

```
    global DATA_FRAME

    if request.method == 'POST':

        url = request.form['url']
        company = request.form['company']

        review_box_tag = request.form['review_box_tag']
        review_box_att = request.form['review_box_att']
        review_box_att_val = request.form['review_box_att_val']

        review_date_tag = request.form['review_date_tag']
        review_date_att = request.form['review_date_att']
        review_date_att_val = request.form['review_date_att_val']

        review_text_tag = request.form['review_text_tag']
        review_text_att = request.form['review_text_att']
        review_text_att_val = request.form['review_text_att_val']


        year_or_pages = request.form['segment']

        pages_scraped = request.form['pages_scraped']

        years_scrapedTo = request.form['years_scrapedto']


        # organizing the inputs the user generates
        tags = {
            'company_name': company,
            'years_scrapedto': years_scrapedTo,
            'yearorpage': year_or_pages,
            'pages': pages_scraped,
            'box': {
                'tag': review_box_tag,
                review_box_att: review_box_att_val
            },
            'reviews': {
                'tag': review_text_tag,
                review_text_att: review_text_att_val
```

```
            },
            'dates': {
                'tag': review_date_tag,
                review_date_att: review_date_att_val
            }
        }

        scrape_reviews(url, tags)

    #checking the radio buttions
    #if nothing is NONE is clicked present the paragraphs which are the
reviews and tables
        if request.form.get('rating') == None or request.form['rating'] ==
'none':
           #new function call create_table this way I can have put the
table in if none is selected. Before I had the table showing up if
something was selected good, bad , etc
            table_html = create_html_table(DATA_FRAME)
            return render_template('scraped.html',html_paragraphs='',
review_type='No review type selected for preview', table_html=table_html)
        else:
            #getting reviews good, bad etc
            reviews_requested = request.form['rating']
           #if all is selected show all the reviews on bottom of screen
            if reviews_requested == 'all':
                df = DATA_FRAME
            else:
                #whatever is selected show that Dateframe and the title
                df = DATA_FRAME[DATA_FRAME['sentiment'] ==
reviews_requested.title()]
               #after scraping if there is no reviews then tell the users
            if len(df['review text']) == 0:
                return render_template('scraped.html', html_paragraphs='',
review_type='There are no reviews of type' + reviews_requested.title())
            else:
                #building every single review into paragraph form and make
it presentable on html web page
                #made zip to be part of the review text because Dr.
Diederich and Dr. McVey suggested it
```

```
                html_paragraphs = Markup(''.join(f"<p>{date}: {text}</p>"
for text, date in zip(df['review text'], df['review date'])))

                table_html = create_html_table(DATA_FRAME)

                #bring this info back to the scraped.html page
                return render_template('scraped.html',
html_paragraphs=html_paragraphs, review_type=reviews_requested.title() + '
Reviews:', table_html=table_html)


#https://thewebdev.info/2022/04/03/how-to-download-a-csv-file-on-clicking-
a-button-with-python-flask/
#another recommedation by Dr. McVey.....create a way to have the user be
able to download a csv file
@app.route('/download-csv')
def download_csv():
    """ this fuction allows the user to download the dataframe as a csv
file """

    csv_string = DATA_FRAME.to_csv(index=False)
    #tried with Dr. Diederich to get company part of csv download
    var = COMP_NAME

    return Response(csv_string, mimetype='text/csv',
headers={'Content-Disposition': 'attachment;filename=BusinessReport_' +
var + '.csv'})



def create_html_table(df):
    """ A function turn our dataframe into an HTML table """
    # resetting index - formulates data properly
    sentiment_counts = df['sentiment'].value_counts().reset_index()
    #counting up the values in the columns
    sentiment_counts.columns = ['sentiment', 'count']
```

```
#https://stackoverflow.com/questions/13315883/how-to-structure-data-to-eas
ily-build-html-tables-in-flask
    #markup part of flask libary - makes it prettier in html
    table_html = Markup(sentiment_counts.to_html())

    return table_html

#called after scraped_reviews is done
def sentiment_processing(df):
    """ This function will do the sentiment processing
    It will clean the data and run sentiment analysis to be used for
visualization later
    """

    # df = pd.read_csv("./reviews.csv")
    # df = read_dataframe()
    df['review date'] = df['review date'].astype(str)
    df['review date'] = df['review date'].str[-4:]

    polarity=[]
    subjectivity=[]
    sentiments=[]
    subjectives=[]
    for review in df['review text'].to_list():

        pol = TextBlob(review).sentiment.polarity
        sub = TextBlob(review).sentiment.subjectivity

      #my scoring
      #made sense in my mind
      #can change as the future project grows and changes
        if -1.0 <= pol < -.15:
            sent = 'Bad'
        elif pol < .05:
            sent = 'Neutral'
        elif pol < .45:
            sent = 'Okay'
        else:
            sent = 'Good'
```

```
      #can change as well
        if sub <=.45:
            subs = 'Factual'
        else:
            subs = 'Personal Opinion'

        subjectives.append(subs)

        sentiments.append(sent)

        # determining the Polarity
        polarity.append(pol)

        # determining the Subjectivity
        subjectivity.append(sub)

     #columns in dataframe
    df['polarity'] = polarity
    df['subjectivity'] = subjectivity
    df['sentiment'] = sentiments
    df['subjectivity score'] = subjectives

    # df.to_csv('testing.csv', index=False)
    return df



#https://stackoverflow.com/questions/66937279/plotting-sentiment-analysis-
over-time-in-python
#https://jakevdp.github.io/PythonDataScienceHandbook/04.14-visualization-w
ith-seaborn.html
@app.route('/visualizeData')
def visualizeData():
    global DATA_FRAME

    df = DATA_FRAME


    # canvas saved in memory for my graphs
```

```
    # size of each canva 10 * 10
    fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(10, 10))

    # set the colors of the wedges
    #sns set color for each plot
    #matplotlib - for first two
    sns.set_palette('Blues')

    # designing function within the visual data function
    # creates values I am using for my visuals
    def sentimentPie(df, score):
        """ A function to grab the values and words from specific columns
(score variable)"""

        vals = list(df[score].value_counts().to_dict().values())

        words = list(df[score].value_counts().to_dict().keys())

        return vals, words

#https://stackoverflow.com/questions/31460146/plotting-value-counts-in-sea
born-barplot
    # Create figure 1
    value_counts_1, words_1 = sentimentPie(df, 'sentiment')
    colors1 = sns.color_palette('Blues', len(words_1))
    wedges, texts, autotexts = axs[0][0].pie(value_counts_1,
wedgeprops=dict(width=0.5), startangle=-40, labels=words_1,
autopct='%1.1f%%', textprops=dict(color="k"), colors=colors1)
    axs[0][0].set_title('Percentage of review sentiment')
    axs[0][0].legend(wedges, words_1, loc='center')

    # Create figure 2
    value_counts_2, words_2 = sentimentPie(df, 'subjectivity score')
    colors2 = sns.color_palette('Greens', len(words_2))
    wedges, texts, autotexts = axs[0][1].pie(value_counts_2,
wedgeprops=dict(width=0.5), startangle=-40, labels=words_2,
autopct='%1.1f%%', textprops=dict(color="k"), colors=colors2)
    axs[0][1].set_title('Percentage of review opinion vs factual')
    axs[0][1].legend(wedges, words_2, loc='center')
```

```
    # Create figure 3
    sns.countplot(x='sentiment', hue='review date', data=df, ax=axs[1][0])
    axs[1][0].set_title('Countplot of review sentiment by year')

    # Create figure 4
    sns.countplot(x='review date', hue='sentiment', data=df, ax=axs[1][1])
    axs[1][1].set_title('Countplot of yearly reviews by sentiment')



    canvas=FigureCanvas(fig)
    img=io.BytesIO()
    fig.savefig(img, format='png')
    img.seek(0)

    return send_file(img, mimetype='img/png')
#put on a host
if __name__ == '__main__':
    app.run(host='localhost', port=5000, debug=True) # use when testing
locally

    #app.run(host='compsci04.snc.edu', port=3947, debug=True) # use when
running on compsci04
```