TLDR - Code

Helpers.py

```
## Imports
import bs4 as bs
import urllib.request
import re
import nltk
import nltk.tokenize
from nltk.corpus import stopwords
from nltk.tokenize import sent_tokenize, word_tokenize
import heapq
import math
import numpy as np
stop_words = set(stopwords.words('english'))
## IDF Calculation
def idf(sent_list, word_sent):
    return math.log(len(sent_list)/word_sent)
## Pull Wikipedia Text
def pull_text(article_url):

    ## Read in article
    scraped_data = urllib.request.urlopen(article_url)
    article = scraped_data.read()
    parsed_article = bs.BeautifulSoup(article,'lxml')
    paragraphs = parsed_article.find_all('p')
    article_text = ""

    for p in paragraphs:
        article_text += p.text

    return article_text
## Format Text
def fix_it_up(article_text):

    ## Remove square brackets and extra spaces
    article_text = re.sub(r'\[[0-9]*\]', ' ', article_text)
    article_text = re.sub(r'\s+', ' ', article_text) # Remove extra space
    ## Remove everything else
    formatted_article_text = re.sub('[^a-zA-Z]', ' ', article_text)
    formatted_article_text = re.sub(r'\s+', ' ', formatted_article_text)
    return formatted_article_text, article_text
## Tokenize Sentences, Find Word Frequency
def sentence_tokenize(formatted_article_text, article_text, stop_words = stop_words):

    ## Tokenize the sentences in the OG article text, initialize stopwords
    sentence_list = nltk.sent_tokenize(article_text)
    stp_wds = stop_words
    word_frequencies = {} # A dictionary of words and how often they show up
```

```
    ## Fill up word_freq dict with (you guessed it) word frequencies
    for word in nltk.word_tokenize(formatted_article_text):
        if word not in stp_wds:
            if word not in word_frequencies.keys():
                word_frequencies[word] = 1
            else:
                word_frequencies[word] += 1
    return word_frequencies, sentence_list
def word_idf_create(sent_list, word_freq):
    word_idf = {}
    for word in word_freq.keys():
        word_idf[word] = idf(sent_list, word_freq[word])
    return word_idf
## Find IDF Values for Sentences
def sent_idf_create(sent_list, word_idf):
    sent_vec = [word_tokenize(val) for val in sent_list]
    sent_idf = {}
    for sent in sent_vec:
        sent_counter = 0.0
        sent_idx = sent_vec.index(sent)
        for word in sent:
            if word in word_idf.keys():
                sent_counter += word_idf[word]
        sent_trueVal = sent_list[sent_idx]
        sent_idf[sent_trueVal] = sent_counter
    return sent_idf
## Sort Top N IDF Values
def top_n(sent_idf, num_sents):
    all_sents = list(sent_idf.keys())
    all_stats = list(sent_idf.values())
    final_sents = []
    top_idx = list(np.argsort(all_stats)[-1*num_sents:])
    top_idx.sort()
    for idx in top_idx:
        final_sents.append(all_sents[idx])
    return final_sents
def wiki_to_sents(article_url, num_sents):
    article_text = pull_text(article_url)
    formatted_article_text, article_text = fix_it_up(article_text)
    word_freq, sent_list = sentence_tokenize(formatted_article_text, article_text, stop_words)
    word_F = word_idf_create(sent_list, word_freq)
    sent_F = sent_idf_create(sent_list, word_F)
    top_sents = top_n(sent_F, num_sents=num_sents)
    return top_sents
def text_to_sents(article_text, num_sents):
    formatted_article_text, article_text = fix_it_up(article_text)
    word_freq, sent_list = sentence_tokenize(formatted_article_text, article_text, stop_words)
    word_F = word_idf_create(sent_list, word_freq)
    sent_F = sent_idf_create(sent_list, word_F)
    top_sents = top_n(sent_F, num_sents=num_sents)
    return top_sents
```

App.py

```
from flask import Flask, render_template, url_for, request, redirect
from helpers import *
app = Flask(__name__)
@app.route('/')
def index():
    return render_template('index.html')
@app.route('/wiki', methods=['GET','POST'])
def wiki():
    return render_template('wiki.html')
@app.route('/text', methods=['GET','POST'])
def text():
    return render_template('text.html')
@app.route('/wiki_results', methods=['GET','POST'])
def wiki_results():
    input_text = request.form['url']
    input_sents = int(request.form['num_sents'])
    sentences = wiki_to_sents(input_text, num_sents=input_sents) ## Old function
    return render_template('wiki_results.html', sentences = sentences)
@app.route('/text_results', methods=['GET','POST'])
def text_results():
    input_text = request.form['text']
    input_sents = int(request.form['num_sents'])
    sentences = text_to_sents(input_text, num_sents=input_sents) ## Old function
    return render_template('text_results.html', sentences = sentences)
if __name__ == "__main__":
    app.run(debug=True)
```

Templates

Layout

```
<!DOCTYPE html>
<!--[if lt IE 7]>      <html class="no-js lt-ie9 lt-ie8 lt-ie7"> <![endif]-->
<!--[if IE 7]>         <html class="no-js lt-ie9 lt-ie8"> <![endif]-->
<!--[if IE 8]>         <html class="no-js lt-ie9"> <![endif]-->
<!--[if gt IE 8]>      <html class="no-js"> <!--<![endif]-->
<html>
    <head>
        <meta charset="utf-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <title>Text Summarizer</title>
        <meta name="description" content="">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <link rel="stylesheet" href="{{ url_for('static', filename='css/main.css') }}">
    </head>
    <body>
        <div class="home">
```

```
            <header>TL; DR</header>
            {% block content %}{% endblock %}
            <script src="" async defer></script>
        </div>
    </body>
</html>
```

Index

```
{% extends "layout.html" %}
{% block content %}
<p>
    <a href="{{ url_for('wiki') }}">Click this for wiki link</a>
    <a href="{{ url_for('text') }}">Click this for text</a>
</p>
{% endblock content %}
```

Text

```
{% extends "layout.html" %}
{% block content %}
<p>
    <form method='POST' action="{{ url_for('text_results') }}">
        Text: <textarea name="text" id="text" cols="30" rows="10"></textarea>  number of sentences:
<input type="int", name='num_sents', placeholder="number of sentences">
        <input type="submit" , value='post it yo' >
    </form>
</p>
{% endblock content %}
```

Text Results

```
{% extends "layout.html" %}
{% block content %}
    <ul>
        {% for sentence in sentences: %}
        <li>{{ sentence }}</li>
        {% endfor %}
    </ul>
{% endblock content %}
```

Wiki

```
{% extends "layout.html" %}
{% block content %}
<p>
    <form method='POST' action="{{ url_for('wiki_results') }}">
        URL: <input type="text", name='url', placeholder="enter a URL">  number of sentences:  <input
type="int", name='num_sents', placeholder="number of sentences">
```

```
        <input type="submit" , value='post it yo' >
    </form>
</p>
{% endblock content %}
```

Wiki Results

```
{% extends "layout.html" %}
{% block content %}
    <div class="container">
        <ul class="menu">
            {% for sentence in sentences: %}
            <li>{{ sentence }}</li>
            <p>.</p>
            {% endfor %}
        </ul>
    </div>
{% endblock content %}
```

Writeup Outline

Intro/Motive

Term Frequency-Inverse Document Frequency

Term Frequency-Inverse Document Frequency (TF-IDF) is a numerical statistic that reflects the importance of a term within a document and across a collection of documents. It is commonly used in information retrieval and text mining tasks.

The mathematical formula for TF-IDF is calculated as follows:

```
TF-IDF(t, d, D) = TF(t, d) * IDF(t, D)
```

Where:
- **TF**(t, d) represents the Term Frequency of term **t** within document **d**.
- **IDF**(t, D) represents the Inverse Document Frequency of term **t** across the document collection **D**.

To calculate **TF**(t, d), you can use different methods, but a common one is the raw term frequency:

```
TF(t, d) = # occurences OF term t IN doc d
```

To calculate **IDF**(t, D), you can use the following formula:

```
IDF(t, D) = log(N / (1 + DF(t, D)))
```

Where:
- **N** represents the total number of documents in the collection **D**.
- **DF**(t, D) represents the Document Frequency of term **t** within the document collection **D**, i.e., the number of documents in which the term **t** appears.

Notice that **DF** is inversely related to **IDF** - this is why our measure is called *inverse* document frequency.

Another important distinction to make here is that document frequency measures the amount of *documents* the term shows up in, whereas term frequency simply deals with the number of times that term appears in *one particulardocument*, **d**, which is a *subset* of the entire collection, **D**.

For our intents and purposes, a "document" is a paragraph, and a "document" collection is the entire block of text or the Wikipedia article we're dealing with.

Therefore, one can think of term frequency of a term as the number of times it shows up in a paragraph, and IDF as an inverse measure of the number of times the term shows up in the text as a whole. It's important to note that because the TF-IDF function is asymptotic at 1 and curves down, it prevents words that are more general (and thus tend show up in a lot of a paragraph, but also a lot of paragraphs) from being inflection points of summarization. This allows more important key words to be prioritized.

In more succinct terms, if you were looking to summarize an article on President Biden, this algorithm would prioritize "Biden" over "President". While the latter probably shows up just as much as the former, "Biden" is more important in the context of the article, and thus it stands to reason that "Biden" probably shows up in more paragraphs.

By multiplying the term frequency and the inverse document frequency, TF-IDF assigns a weight to each term in a document, which indicates its significance in that specific document and the entire collection.

Please note that the above formulas represent the general calculation of TF-IDF, but there are variations and alternative formulations depending on specific requirements or implementations.

The Process

To be honest, the most difficult thing about this project was finding the right algorithm. Everything else was straightforward, with the biggest hurdle - the challenge of encoding all of these words - was handled by the strength of NLTK.

NLTK

Natural Language Toolkit, or NLTK, is a python library capable of powerful tokenization algorithms. Commonly used for sentiment analysis and sentence recognition technology, NLTK allows its user to draw connections between words and sentences that would otherwise only exist abstractly.

Beautiful Soup