**Bitcot Technologies**

**Mini Project 1**

**Mini-Project #1: Markdown File Analyzer**

**Project Overview**

The Markdown File Analyzer is a simple Python program that scans .md files, providing a transparent summary of their content. It has word counting, detects headings (# through ###### using regex), follows links, and detects images using markdown and BeautifulSoup.

It also verifies links with requests and marks broken ones (HTTP 400+). It's hosted on GitHub (git clonehttps://github.com/tatedmaahi/miniprj1.git) and relies on requirements.txt dependencies (markdown, requests, beautifulsoup4). A .gitignore avoids garbage like venv/. Try it out with example.md and execute python analyzer.py <file.md> once set up!

**Files**

**.gitignore**

# Virtual environment

venv/

# Python cache

\_pycache\_/

\*.py[cod]

\*$py.class

# VS Code settings

.vscode/

# macOS system files (if using macOS too)

.DS\_Store

# Windows system files

Thumbs.db

# Environment variable files

.env

# Jupyter Notebook checkpoints (just in case)

.ipynb\_checkpoints/

# Logs or temporary files

\*.log

\*.bak

\*.swp

**README.md**

**# Markdown File Analyzer**

A simple tool to analyze .md files, counting words, headings, links, images, validating links and to Output a summary.

**## Features**

- Counts words, headings, links, and images.

- Checks for broken links.

- Outputs a summary report.

**## Installation**

```bash

git clone https://github.com/tatedmaahi/miniprj1.git

cd markdown-analyzer

pip install -r requirements.txt

**example.md**

**# Test File**

This is a test with 10 words.

**## Subheading**

[Google](https://google.com)

[Broken](https://nonexistent.example.com)

![Image](https://example.com/image.jpg)

**requirements.txt**

markdown

requests

beautifulsoup4

**analyzer.py**

import markdown

import re

import requests

from bs4 import BeautifulSoup

def analyze\_markdown(file\_path):

    """Analyze a Markdown file for content statistics (words, headings, links, images, broken links)."""

    try:

        with open(file\_path, 'r', encoding='utf-8') as f:

            content = f.read()

    except FileNotFoundError:

        raise FileNotFoundError(f"The file '{file\_path}' was not found.")

    except UnicodeDecodeError:

        raise UnicodeDecodeError(f"Unable to decode '{file\_path}' with UTF-8 encoding.")

    word\_count = len(content.split())

    heading\_count = len(re.findall(r'^#{1,6}\s', content, re.MULTILINE))

    link\_count = 0

    image\_count = 0

    broken\_links = []

    html = markdown.markdown(content)

    soup = BeautifulSoup(html, 'html.parser')

    links = soup.find\_all('a')

    link\_count = len(links)

    images = soup.find\_all('img')

    image\_count = len(images)

    for link in links:

        url = link.get('href')

        if url and url.startswith('http'):

            try:

                response = requests.head(url, timeout=5, allow\_redirects=True)

                if response.status\_code >= 400:

                    broken\_links.append(url)

            except requests.RequestException:

                broken\_links.append(url)

    return {

        'file': file\_path,

        'words': word\_count,

        'headings': heading\_count,

        'links': link\_count,

        'images': image\_count,

        'broken\_links': broken\_links

    }

def print\_report(report):

    """Print the analysis report in a formatted manner."""

    print(f"Analysis Report for {report['file']}:")

    print(f"- Words: {report['words']}")

    print(f"- Headings: {report['headings']}")

    print(f"- Links: {report['links']}")

    print(f"- Images: {report['images']}")

    if report['broken\_links']:

        print(f"- Broken Links: {', '.join(report['broken\_links'])}")

    else:

        print("- No broken links found.")

if \_\_name\_\_ == '\_\_main\_\_':

    import sys

    if len(sys.argv) != 2:

        print("Usage: python analyzer.py <file.md>")

    else:

        report = analyze\_markdown(sys.argv[1])

        print\_report(report)

**Tools And Functionality**

**Tools:**

This project is built with these tools:

* **Python**: The backbone of the script, where all the magic happens.
* **markdown**: A neat library that turns Markdown text into HTML so we can dig into it.
* **re (Regular Expressions)**: Python’s built-in tool for spotting Markdown headings (like # or ##) with some clever pattern matching.
* **requests**: Used to ping links and check if they’re working or broken.
* **BeautifulSoup**: A parser that sifts through HTML to find links (<a> tags) and images (<img> tags).
* **Git**: Keeps the project organized and hosted on GitHub for version control.
* **pip**: The trusty package manager that grabs all the dependencies listed in requirements.txt.

**Functionality;**

* **Content Breakdown**: Counts the words in your file (just splits the text) and spots headings (from # to ######) using regex.
* **Link & Image Hunting**: Turns Markdown into HTML with the markdown library, then uses BeautifulSoup to tally up links and images.
* **Broken Link Checker**: Fires off HTTP requests to test links. If a link returns a 400+ status code or fails, it’s flagged as broken.
* **Slick Report**: Spits out a tidy summary with the file name, word count, number of headings, links, images, and any broken links, thanks to the print\_report function.
* **Command-Line Vibes**: Run it from your terminal with python analyzer.py <file.md>. Mess up the command? It’ll let you know.
* **Error Handling**: Got a missing file or a weird encoding issue? It’ll catch those (like FileNotFoundError or UnicodeDecodeError) and give you a clear error message.

**Observation**

The Markdown File Analyzer is a Python tool that scans Markdown files to count words, headings, links, and images. It converts Markdown to HTML using the markdown library, parses content with BeautifulSoup, and checks for broken links via requests. A formatted report summarizes findings, and a command-line interface makes it user-friendly. With Git for version control and robust error handling, it’s perfect for analyzing Markdown content efficiently.

**Conclusion**

The Markdown File Analyzer is an efficient Python tool for analyzing .md files, offering word counts, heading detection, link and image tallying, and broken link identification. With a user-friendly command-line interface, robust error handling, and GitHub hosting, it leverages markdown, BeautifulSoup, and requests for reliable content parsing. Ideal for streamlined Markdown analysis, it delivers clear, formatted reports, making it a valuable asset for developers and content creators.