INTRODUCTION

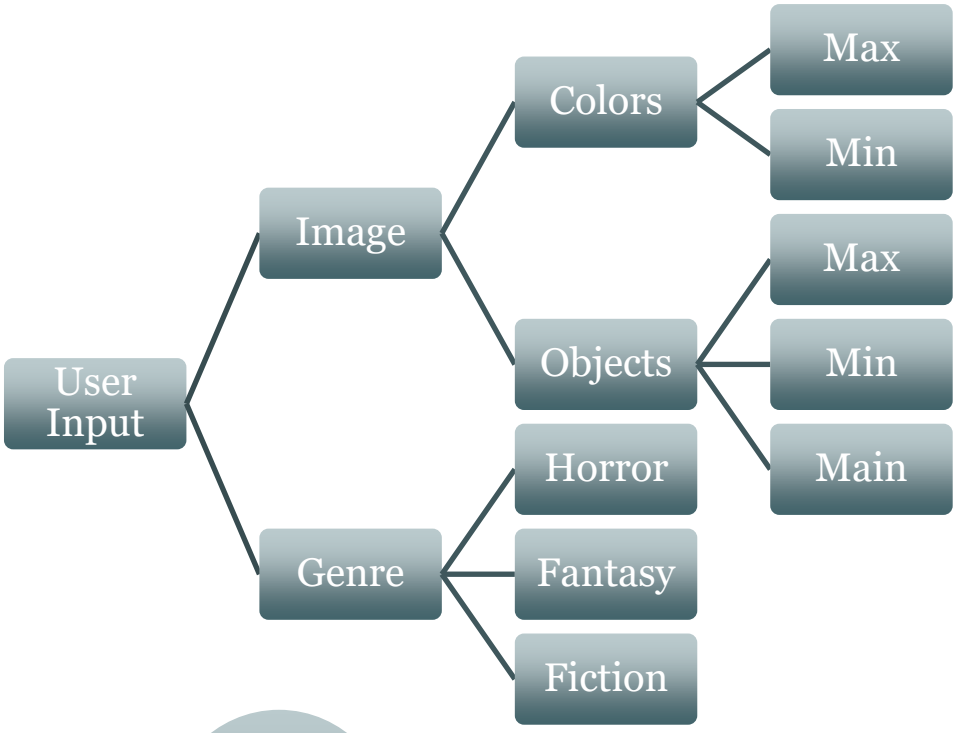
"Every picture tells a story"

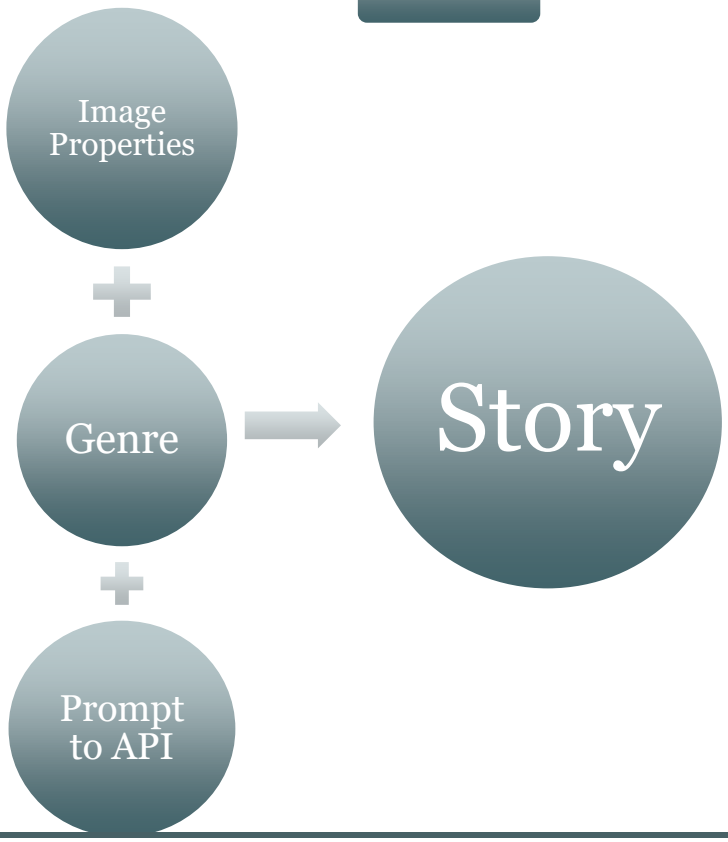
- Anastasia Hollings

Can we tell yours?

'MysticVisions' is a webapp that generates automated stories for any given image.

HOW DOES IT WORK?





CODE BREAKDOWN:

The code is a Flask web application that generates a story based on a given genre and an image.
The web application accepts an image and genre from a web form and saves the image to the server. The image is then processed to determine the most common color and the least common color in the image. The object detection function is called to determine the main object, most common object, and least common object in the image. A prompt is then generated based on the genre, the most and least common color, and the objects in the image. The prompt is used to generate a story with OpenAI's language generation model "text-davinci-003". Finally, the generated story is returned to the user on the index page of the website.

1. object_detection() function identifies the main object, maximum repeated objects and least appearing objects in the image

2. color_detection() function identifies the maximum and minimum spread colours in the image

3. generate_prompt() function generates customized prompt for each genre using the obtained image properties.

4. And Finally, generate_story() function is the main function. It calls all the other functions and using OpenAI's GPT 3's API Model *text-davinci-003* generates the output story

DEEP DIVE INTO CODE:

1. **object_detection()**

   FCOS modal is used which is faster and efficient than YOLO. Specifically, ‘fcos_resnet50_fpn’ backbone is used.
   This function performs object detection on an image using the FCOS (Fully Convolutional One-Stage) object detection model.

   The image is read and transformed using the "read_image" and "to_pil_image" functions from the torchvision library.

   The model is then loaded using the "fcos_resnet50_fpn" function, with the weights set to the default value. The model is set to evaluation mode using the "model.eval()" method.
   Label set is created and the object with highest confidence score is set to main_obj, the object occurring most times in the set is max_obj and least is min_obj

2. **color_detection()**

   The color_detection function uses the Python Imaging Library (PIL) to open the image and resize it to reduce the number of pixels. It then uses the load() function to get all the pixels in the image, and uses a list comprehension to get the color of each pixel.

   The function then uses the collections.Counter class to count the number of occurrences of each color in the image, and uses the most_common() method to get the most common color and least_common() method to get the least common color.

   Then, it uses the webcolors library to convert the RGB color to a human-readable format.

   It has closest_color function to find nearest color using 'Euclidean distance' if color is not found in the webcolors.CSS3_HEX_TO_NAMES_MAP dictionary.

3. **generate_story()**

generate_story() function creates a completion request using OpenAI's API, specifically the **openai.Completion** class.

The following parameters are being passed to the **create** method:

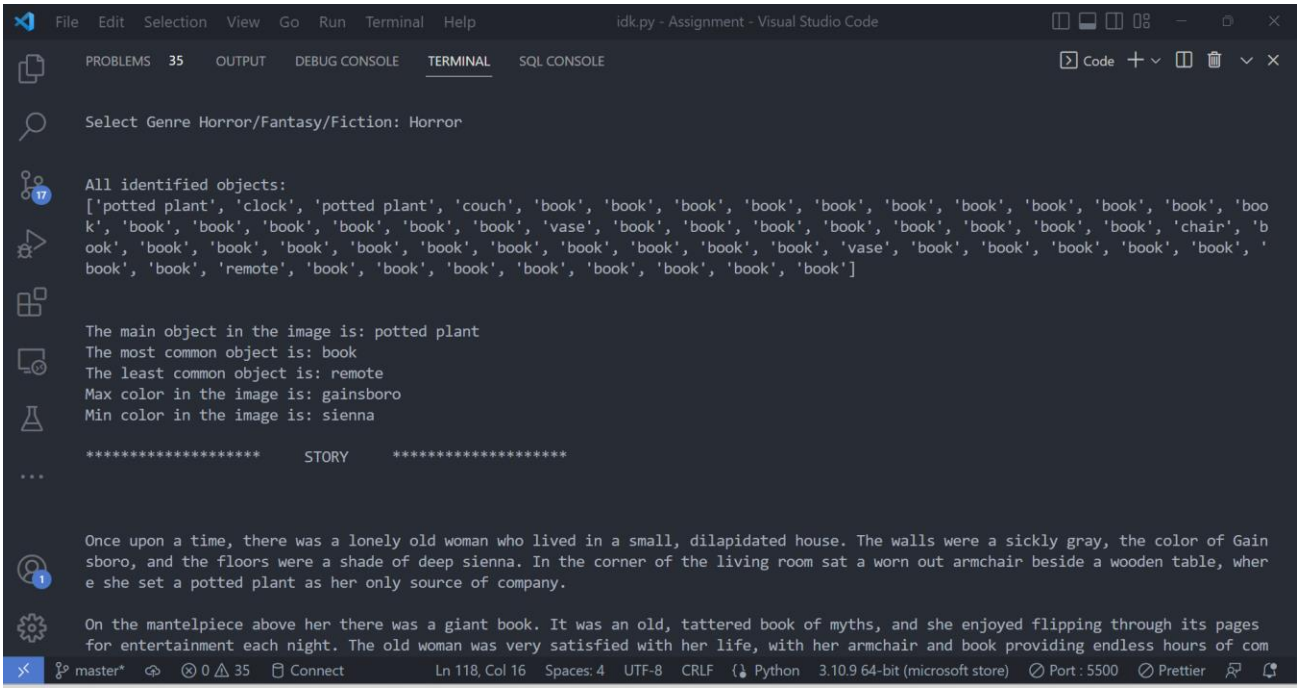
- **engine**: The language model to use for generating the completion, specified by the **model** variable.
- **prompt**: The prompt, or initial text, for the completion, specified by the **prompt** variable.
- **max_tokens**: The maximum number of tokens, or individual elements in the input sequence, to generate in the completion. The value is set to 1024.
- **n**: The number of completions to generate, set to 1.
- **stop**: A sequence of tokens at which the completion should stop generating tokens, set to **None**.
- **temperature**: The temperature, a value used to control the randomness of the generated tokens, set to 1. More temperature means more random

This code does not actually execute the request, it only creates the request object. The request would need to be sent using **. execute()** on the created object.

EXPERIMENTAL RESULTS:

IMAGE:

GENRE: HORROR



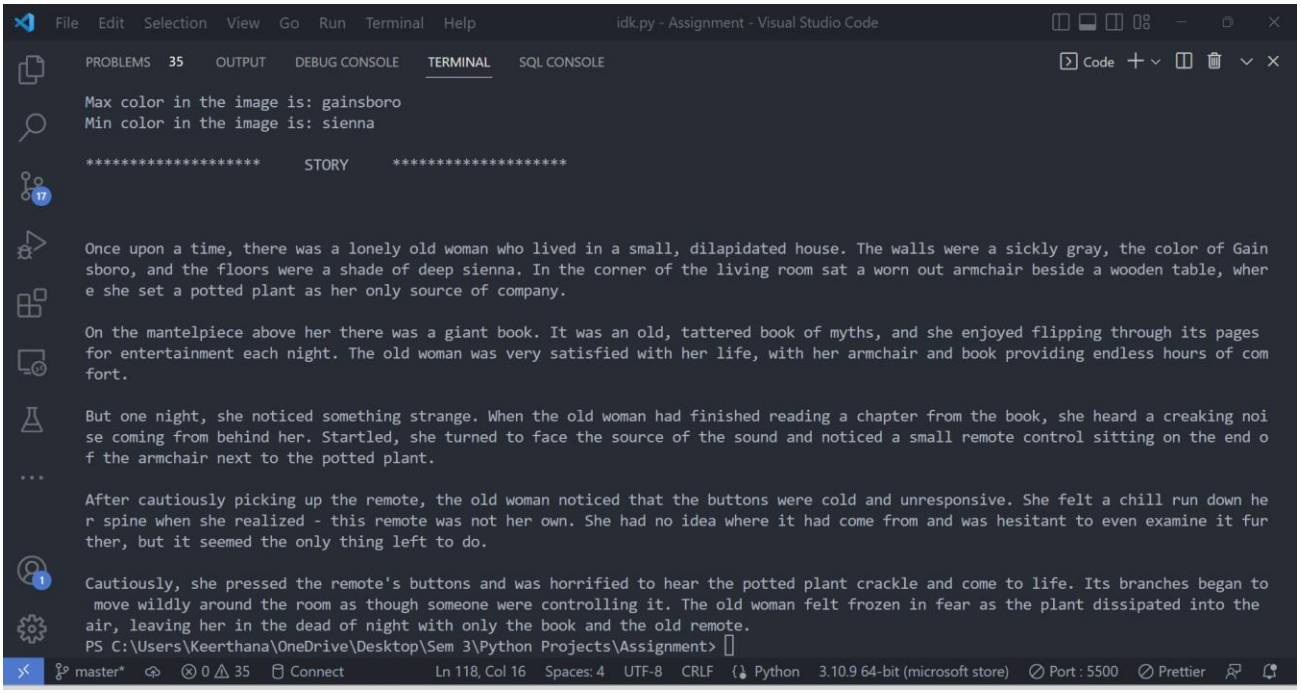
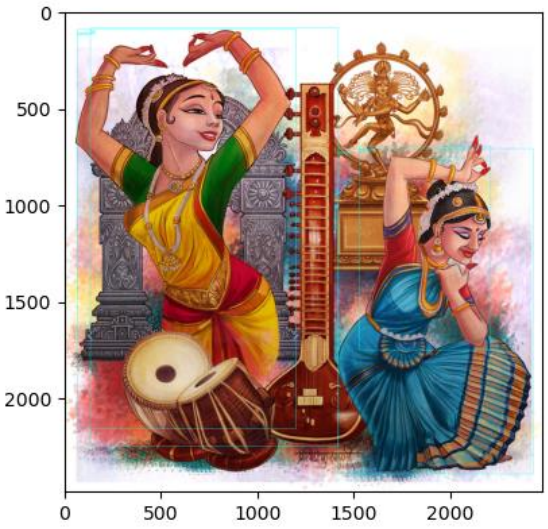
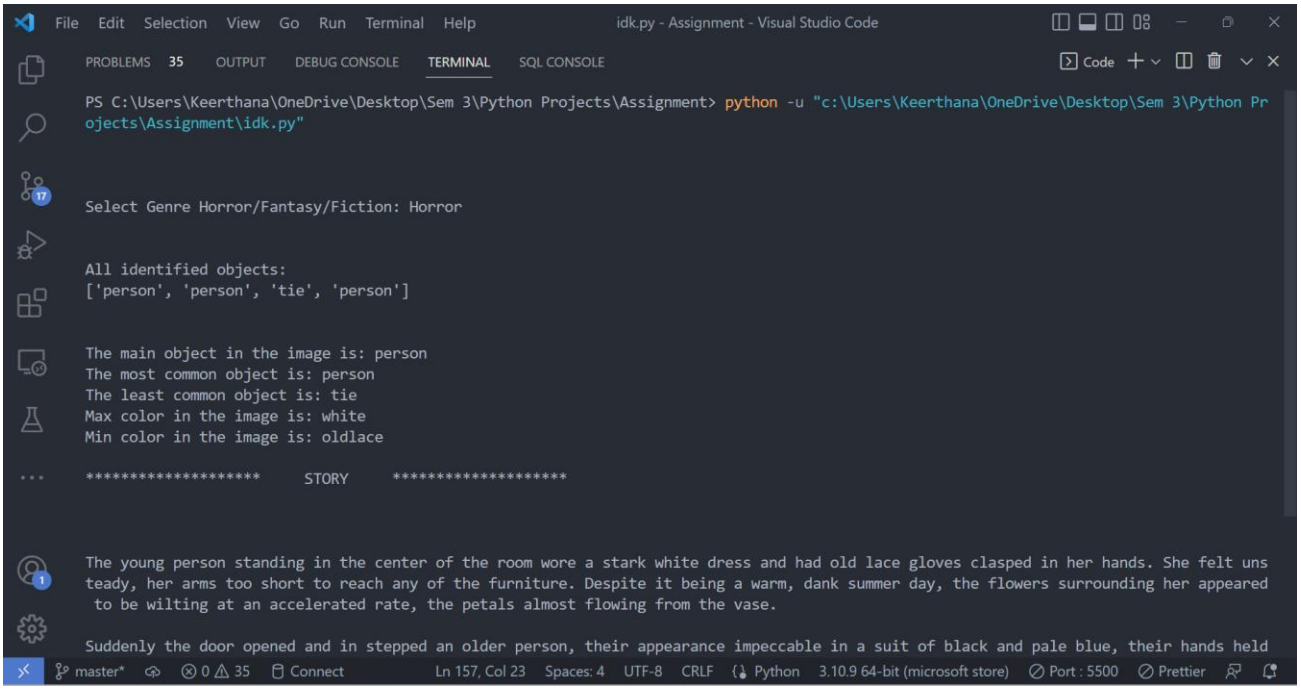
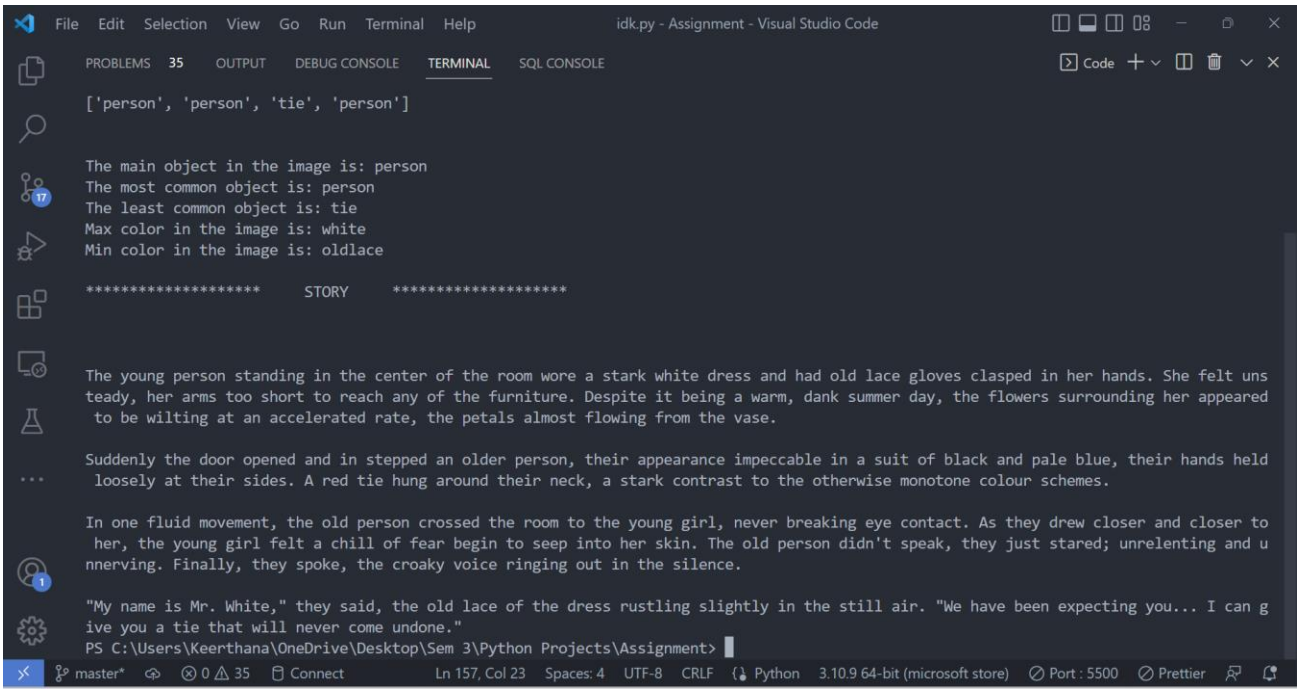


IMAGE:

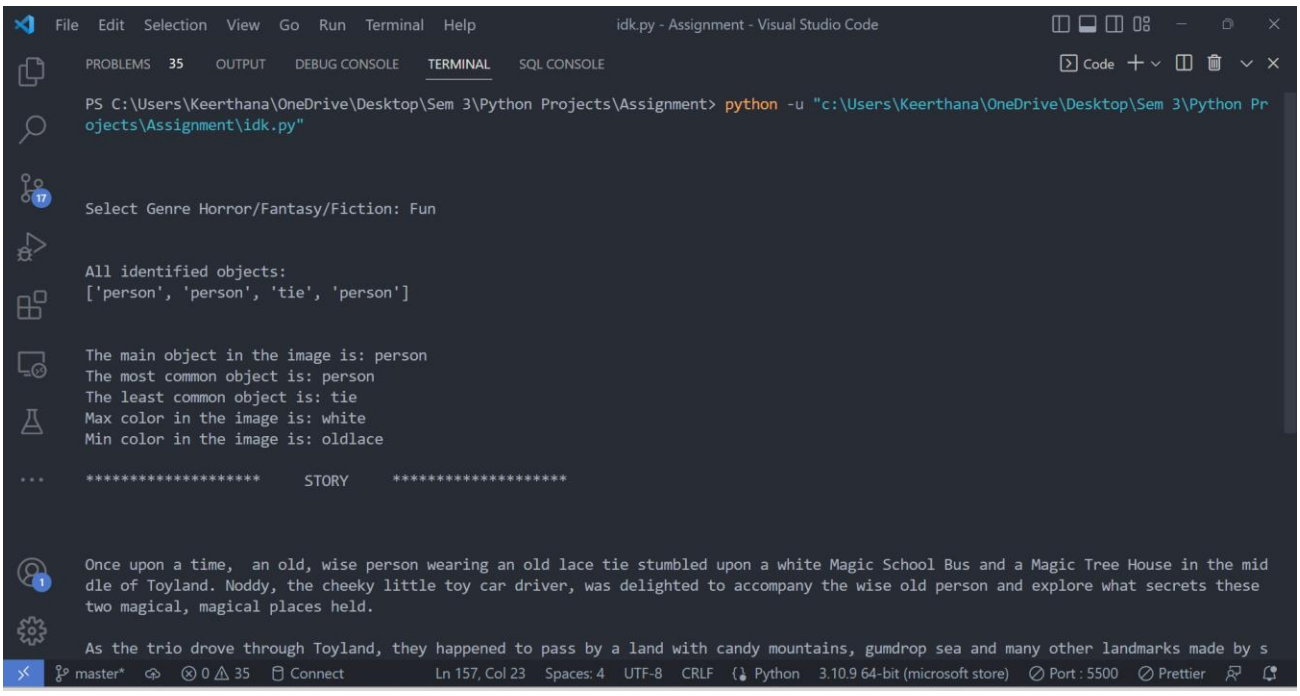


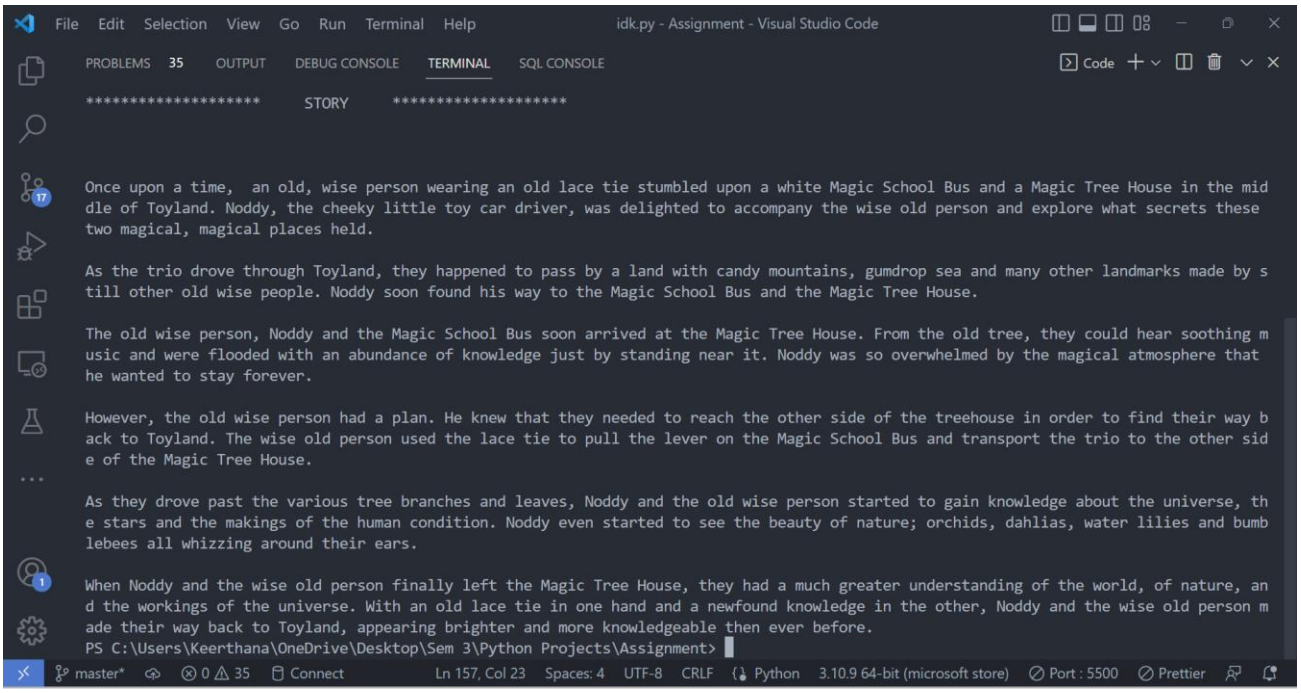
GENRE: HORROR



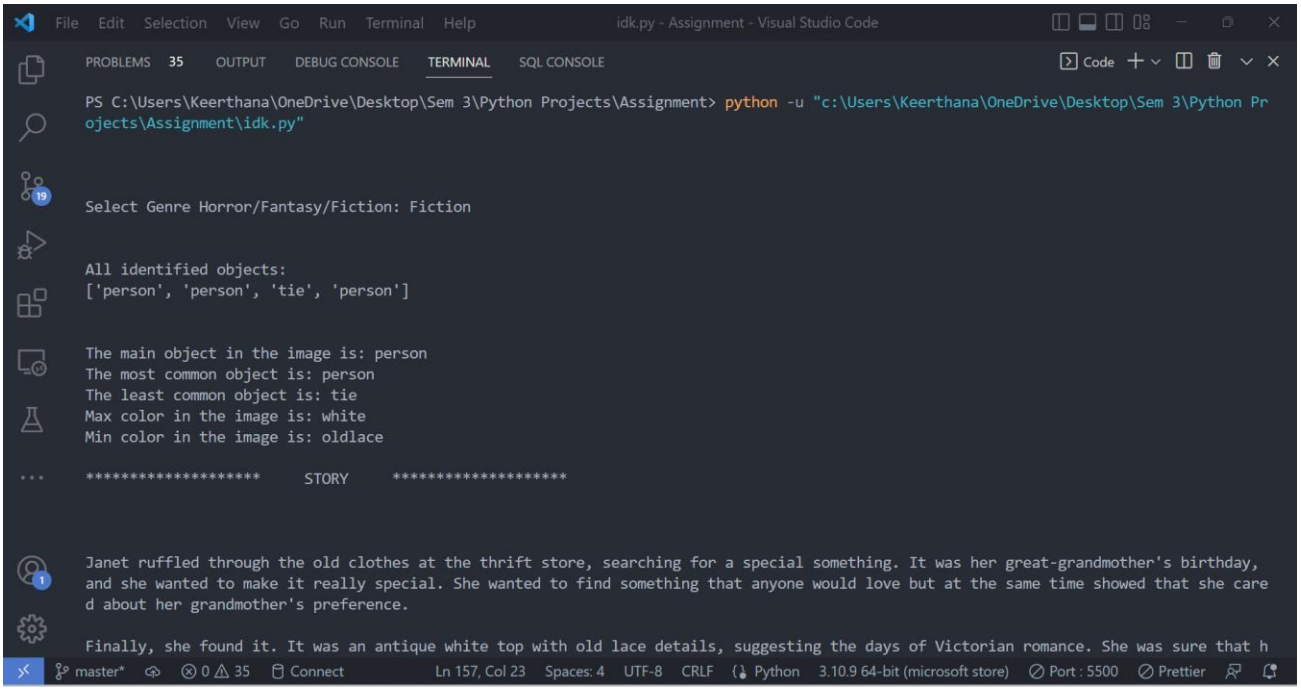


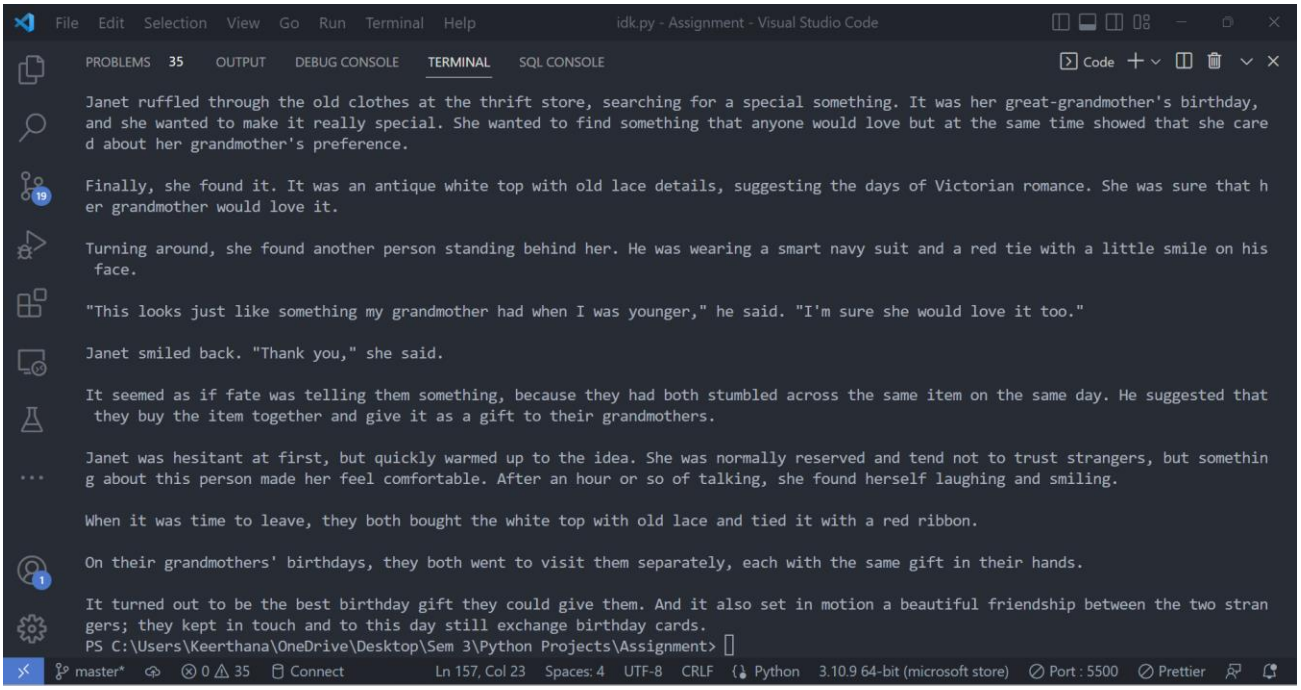
GENRE: FUN
(If any option other than given three are selected, a fun story based on novels 'Magic School Bus', 'Noddy Goes to Toyland' and 'Magic Tree House' is generated.)



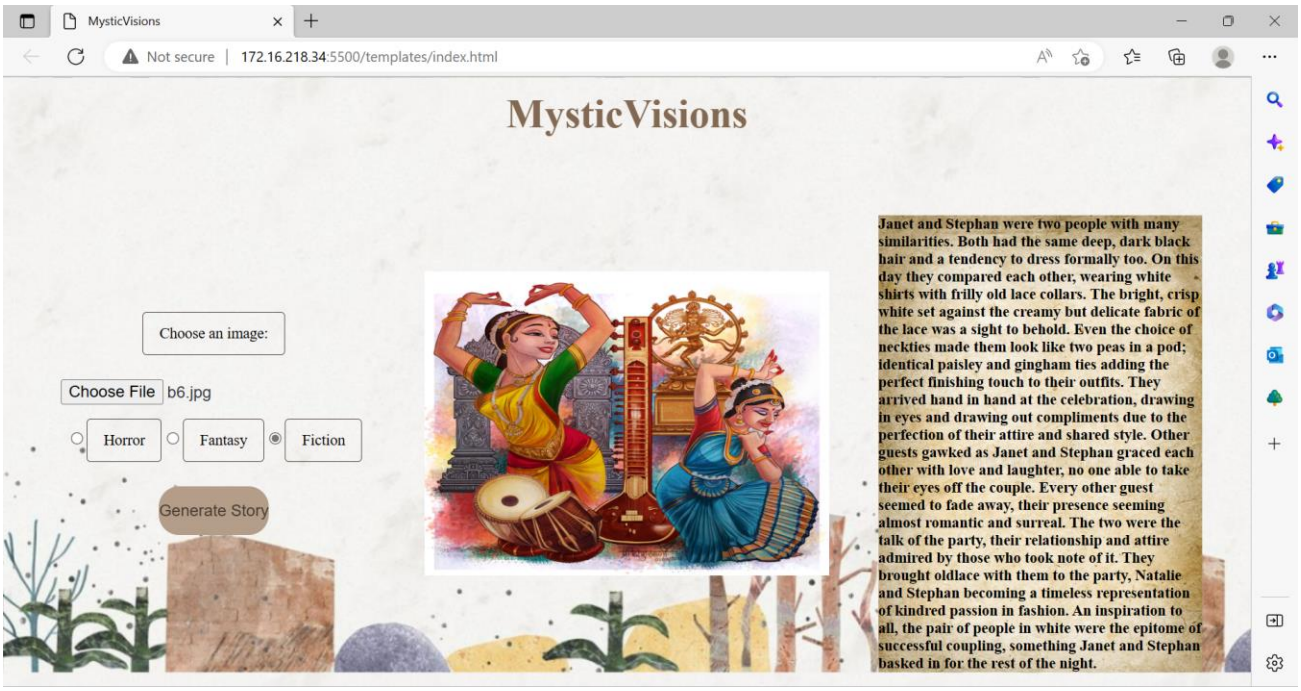


GENRE: FICTION





FINAL OUTPUT PAGE:



CONCLUSION:

As shown above ‘MysticVisions’ works flexibly with different images and same genre, same image and different genre or both different.
It can handle any kinds of file upload errors or diverse images and generate a ‘Mystic’ story for the user.

RELATED STUDIES AND REFERENCES:

- OPENAI MODAL USAGE: https://github.com/openai/openai-quickstart-python
- NPM DOCUMENTATION: https://docs.npmjs.com/getting-started
- FLASK DOCUMENTATION: https://flask.palletsprojects.com/en/2.2.x/quickstart/#a-minimal-application
- FLASK HTTP METHODS: https://www.youtube.com/watch?v=9MHYHgh4jYc&list=RDCMUC4JX40jDee_tINbkjycV4Sg&start_radio=1&t=617s
- FLASK BLUEPRINTS: https://www.youtube.com/watch?v=WteIH6J9v64&list=PLzMcBGfZo4-n4vJJybUVV3Un_NFS5EOgX&index=10
- UPLOAD LOCAL IMAGE WITH PYTHON AND FLASK: https://tutorial101.blogspot.com/2021/04/python-flask-upload-and-display-image.html
- FORMS IN FLASK: https://vegibit.com/how-to-use-forms-in-python-flask/#:~:text=Data%20associated%20with%20an%20HTML,args.&text=The%20code%20just%20above%20uses,%3E%2C%20which%20exists%20in%20home.

- OPENAI API USAGE BLOGS:
  1. https://medium.com/nerd-for-tech/create-ai-application-in-minutes-with-openai-api-5e84bd3ec5d0
  2. https://accessibleai.dev/post/generating_text_with_gpt_and_python/

- OPENAI API DOCUMENTATION: https://beta.openai.com/docs/api-reference/edits/create
- SIMILAR APP WITH ML MODAL: https://huggingface.co/spaces/bipin/image2story
- SITES REFERRED FOR OBJECT DETECTION FUNCTION:

1. https://towardsdatascience.com/object-detection-with-10-lines-of-code-d6cb4d86f606
2. https://towardsdatascience.com/object-detection-and-tracking-in-pytorch-b3cf1a696a98
3. https://towardsdatascience.com/step-by-step-r-cnn-implementation-from-scratch-in-python-e97101ccde55
4. https://pjreddie.com/darknet/yolo/
5. https://medium.com/python-in-plain-english/image-captioning-with-an-end-to-end-transformer-network-8f39e1438cd4
6. https://keras.io/examples/vision/image_captioning/
7. https://www.geeksforgeeks.org/image-captioning-using-python/
8. https://github.com/achen353/Image-Caption-Generator
9. https://www.topcoder.com/thrive/articles/python-for-image-recognition-opencv
10. https://developers.google.com/mediapipe/solutions/vision/object_detector/python
11. https://dontrepeatyourself.org/post/yolov4-custom-object-detection-with-opencv-and-python/
12. https://github.com/chandravenky/Computer-Vision---Object-Detection-in-Python