

# Makeup Ingredient Analyzer

By: Alondra Sanchez & Esmeralda Melendez



<https://github.com/ksu-is/makeupanalyzer>



# INTRODUCTION

The Makeup Ingredient Analyzer is a tool-like bot designed to help users understand the composition of their cosmetic products and pore-clogging ingredients. Analyzing the ingredients listed on makeup labels provides detailed information about each ingredient, including its purpose (e.g., moisturizer, tint), potential benefits, possible risks (e.g., pore cloggers or irritants), and product-to-product & product-to-skin match (e.g., silicone vs. water-based and dry vs. oily skin type, respectively). The tool aims to promote safer and more informed cosmetic choices by educating users about what they're applying to their skin.

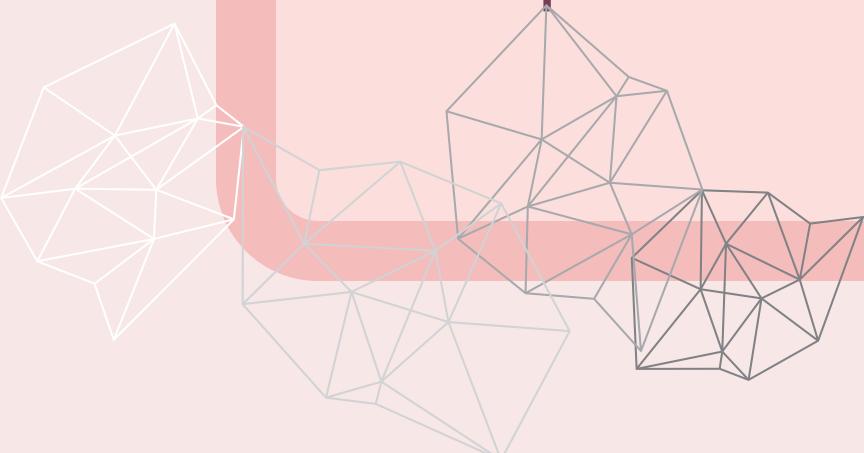
## Packages Used

Telegram, Google Cloud Vision, Pandas, OS

# BACKGROUND

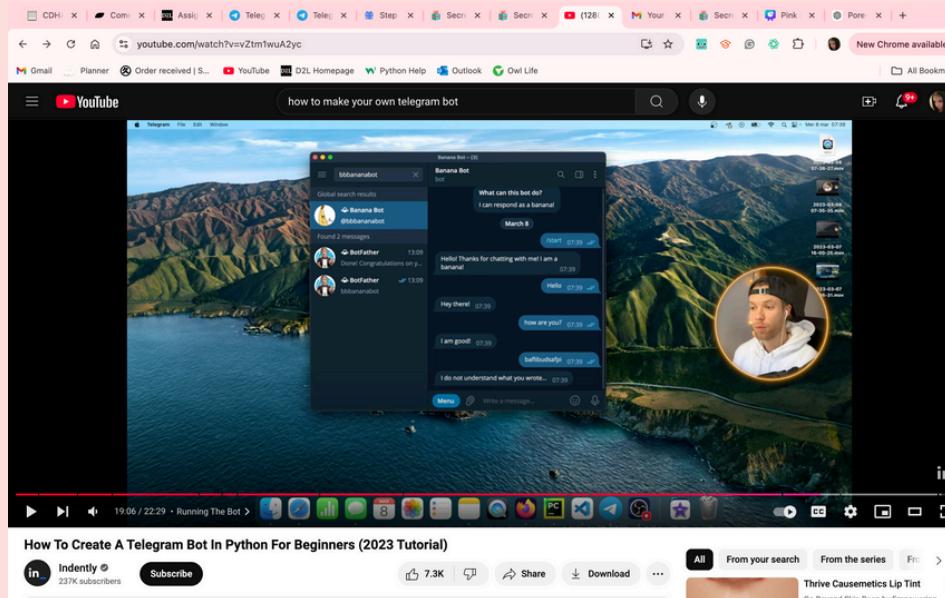
## What was the idea?

The idea stemmed from the need to decrease the amount of time, money and effort to compare a list of ingredients. We were inspired by immense list of cosmetic variations between brands and the struggles with acne limiting makeup choices. Not knowing what or how the products are causing acne is the basis for the program. Use this Bot not only for makeup products but skincare products and hair products as well. Have fun and may your best makeup routine shine!

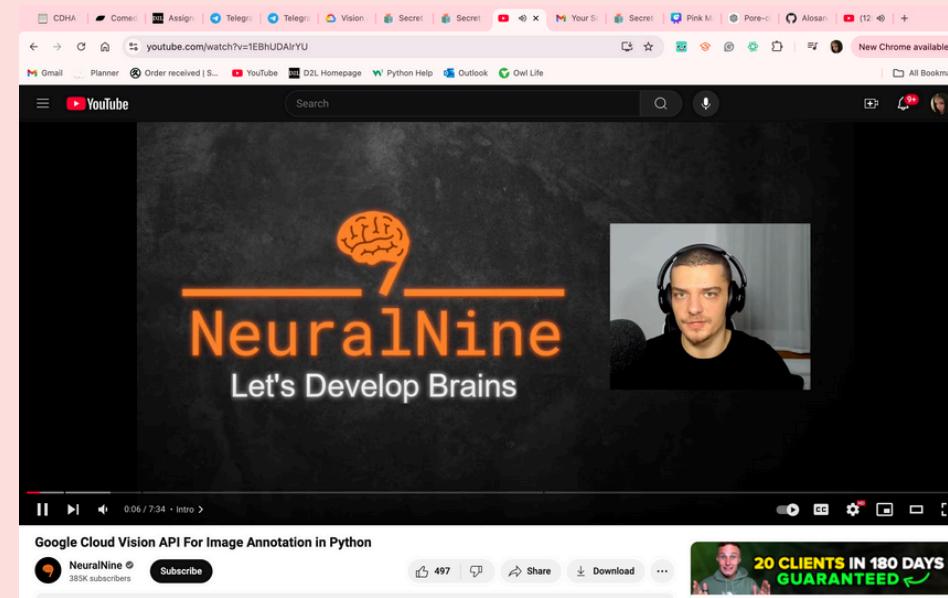


# REFERENCES AND SOURCE CODE

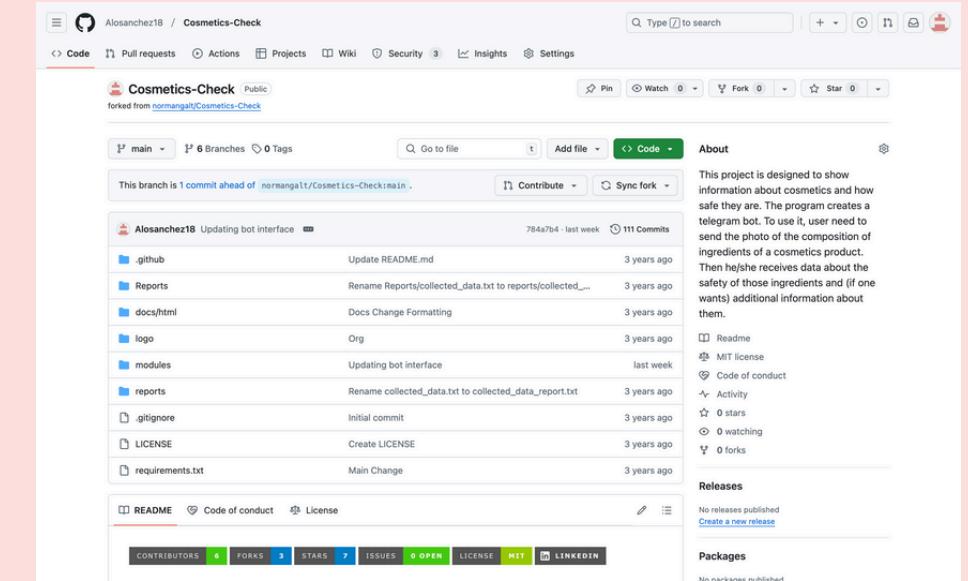
What source code & tutorials did we find?



Telegram Tutorial



Google Cloud Vision Tutorial



Cosmetics Checker

# DEVELOPMENT

What surprised you? How did the scope change? Why?

- Number of packages to use.
- How Telegram allows you to easily create a bot
- Even though it was easy to envision how it COULD come together, we had to replan to change the scope of the project
  - Instead of the bot being able to do two different tasks, our focus became one.

What roadblocks did you face? How did you overcome them?

- We began with a small list to get the interface started as turned in on Sprint two but how we soon realized having a spreadsheet would have been better
- We also found lists of acne-aggravating ingredients but not a corresponding list as to why
- We had to compile descriptions as to why to overcome the issue.
- Because the list of ingredients is almost 500, we only got to about 50% before moving on to another step.
- Not understanding quite what each point in the reference code meant. Using CHAT to interpret to better gage where the changes were going to be.

# DEVELOPMENT

This is was our main code before  
before.

The screenshot shows a code editor interface with the following details:

- File Explorer (Left):** Shows project files including `ProjectRoadMap.md`, `bot_interface.py`, `Main.py`, and `vision_api.py`.
- Search Bar (Top):** Contains the text "makeupanalyzer".
- Code Editor (Main Area):** Displays Python code for analyzing ingredients. The code defines a dictionary of pore-clogging ingredients and a function to analyze a list of ingredients, printing results to the console.
- Output/Console (Right):** Shows the execution results of the code, including the pore-clogging ingredients and the analysis output.

```
import pandas as pd
import os

#here i am asking for the specific database
file_name = "ingredient_makeup.xlsx"
file_path = r'C:/Users/alondra/Downloads'
file = os.path.join(file_path,file_name)

df = pd.read_excel(file) #use pandas on it

# This is a simple code that we will expand the project on.

pore_clogging_ingredients = {
    "coconut oil": "Coconut oil is pore-clogging because the molecules are too big for your pore size, especially if you already struggle with acne.",
    "lanolin": "Lanolin is highly comedogenic and can trap dirt and bacteria in your pores, leading to breakouts.",
    "isopropyl myristate": "Isopropyl myristate clogs pores and exacerbates acne-prone skin.",
    "wheat germ oil": "Wheat germ oil has a high comedogenic rating and can block pores, causing acne.",
    "algea extract": "Algae extract can be too rich for sensitive or acne-prone skin, leading to clogged pores."
}

# Making a function to analyze ingredients
def analyze_ingredients(ingredient_list):
    print("Analyzing ingredients: \n")
    for ingredient in ingredient_list:
        ingredient_lower = ingredient.lower()
        if ingredient_lower in pore_clogging_ingredients:
            print(ingredient.capitalize() + " is not good for your skin.")
            print("Reason: " + pore_clogging_ingredients[ingredient_lower] + "\n")
        else:
            print(ingredient.capitalize() + " appears to be safe for your skin.\n")
    print("Analysis complete! *-* ; -_- *-* ,> \nYou are now on your way to clearer skin! *-*... o(gvg)o ...-*")"

# Example list of ingredients to analyze including both good and bad
ingredient_list = [
    "Coconut Oil",
    "Aloe Vera",
    "Lanolin",
    "Vitamin E",
    "Isopropyl Myristate"
]

# Calling it
analyze_ingredients(ingredient_list)
```

This is how our list looks like now.

# BOT\_INTERFACE.PY

All the packages are gathered in the interface. It is the “main” code where it all happens. This handles the behavior of the bot.

The screenshot shows a code editor interface with the following details:

- Explorer View:** Shows the project structure with files like `bot_interface.py`, `Main.py`, and `dataset_ADT.py`.
- Search Bar:** Contains the text `makeupalyzer`.
- Code Editor:** Displays the `bot_interface.py` file content.
- Right Panel:** Shows a preview of the code and some status information.

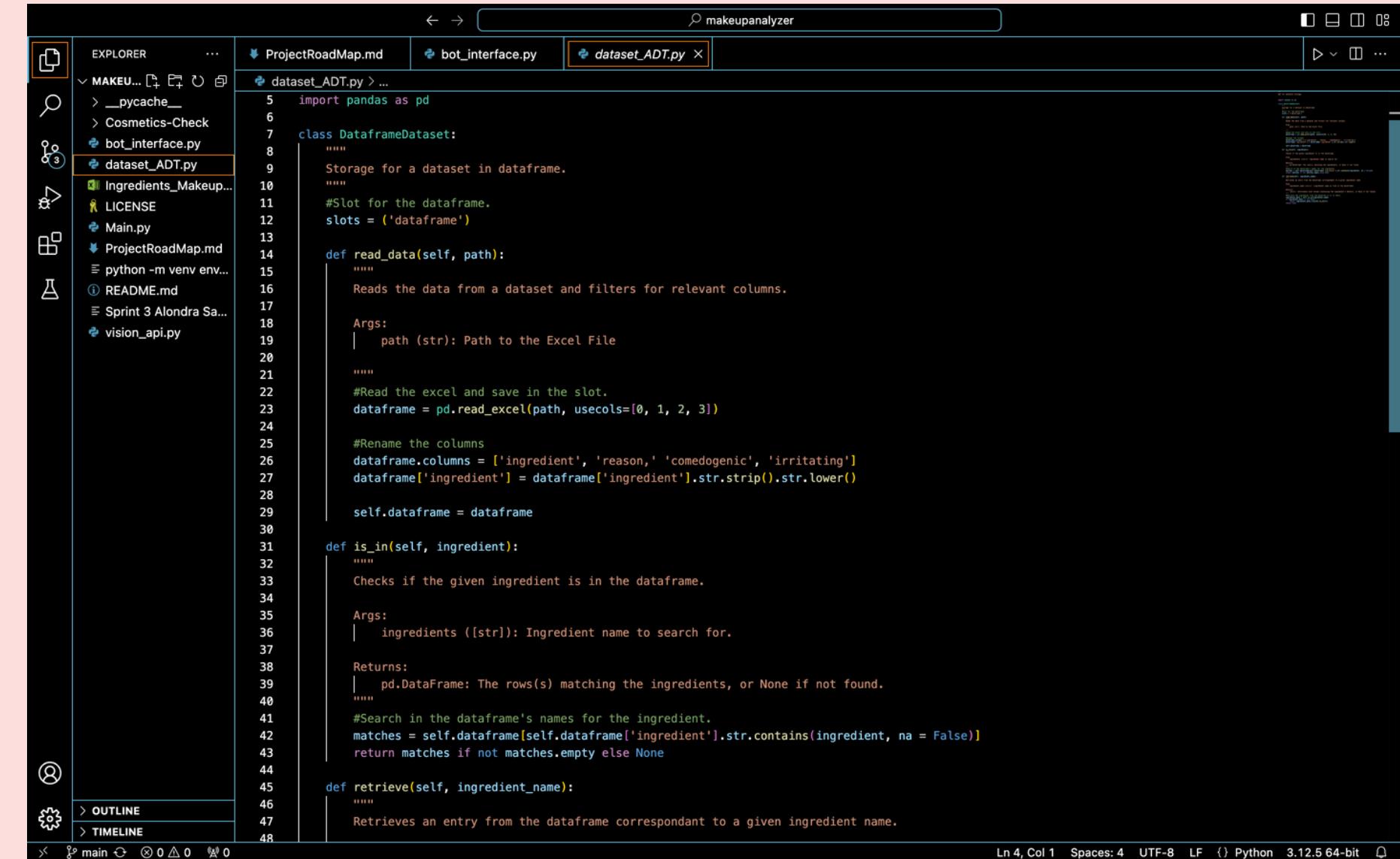
```
1 """
2     Telegram Bot Interface.
3 """
4
5 #Import the needed libraries
6 import telebot
7 from telebot import types
8 from vision_api import Cream
9 from dataset_ADT import DataframeDataset
10
11 #Load environment variables from .env file
12 API_KEY = '8083137281:AAGQiXiBuXorqrVt3QV7hi5W_g5UeefP0WI'
13
14 #has now been updated with our actual API key
15 ingredients = None
16
17
18 #Create and activate bot-service instance.
19 bot = telebot.TeleBot(API_KEY)
20
21 #Markups for the interface.
22 markup_for_yes_no = types.InlineKeyboardMarkup()
23
24 item_yes = types.InlineKeyboardButton(text='YES', callback_data='yes')
25 item_no = types.InlineKeyboardButton(text='NO', callback_data='no')
26
27 markup_for_yes_no.add(item_yes, item_no)
28
29 #Create a 'Cream' ADT-service instance.
30 cream_processor = Cream()
31
32 #Create storage-service ADTs for the datasets.
33 pore_clogging_dataset = DataframeDataset()
34 pore_clogging_dataset.read_data('Ingredients_Makeup.xlsx')
35
36 #Global storage for demand
37 ingredients = None
38
39 #Handle the initial message.
40 @bot.message_handler(commands=['start'])
41 def start(message):
42     """
43         Start the dialog with the bot.
44 """
```

The screenshot shows a code editor interface with the following details:

- Left Sidebar:** Includes icons for Explorer, Search, Find, Project, Main.py, ProjectRoadMap.md, README.md, Sprint 3 Alondra Sa..., and vision\_api.py.
- Top Bar:** Shows file tabs for "ProjectRoadMap.md", "bot\_interface.py" (which is the active tab), and "Main.py".
- Code Area:** Displays Python code for a bot interface. The code defines two functions: `start` and `add_photo`.
  - `start` handles the '/start' command, starting a dialog with the user. It sends a welcome message and provides instructions for using the service, including sending a photo and choosing options based on user responses.
  - `add_photo` handles the '/sendphoto' command, receiving a photo from the user and prompting them to send it.
- Right Side:** A large panel on the right displays a terminal window with multiple lines of text, likely logs or command-line output.

# DATASET\_ADT.PY

The reason for setting it up this way without the direct path here is that the code can be more versatile for future reference if we need to update the list.



```
import pandas as pd

class DataframeDataset:
    """
    Storage for a dataset in dataframe.
    """
    #Slot for the dataframe.
    slots = ('dataframe')

    def read_data(self, path):
        """
        Reads the data from a dataset and filters for relevant columns.

        Args:
            | path (str): Path to the Excel File
        """
        #Read the excel and save in the slot.
        dataframe = pd.read_excel(path, usecols=[0, 1, 2, 3])

        #Rename the columns
        dataframe.columns = ['ingredient', 'reason', 'comedogenic', 'irritating']
        dataframe['ingredient'] = dataframe['ingredient'].str.strip().str.lower()

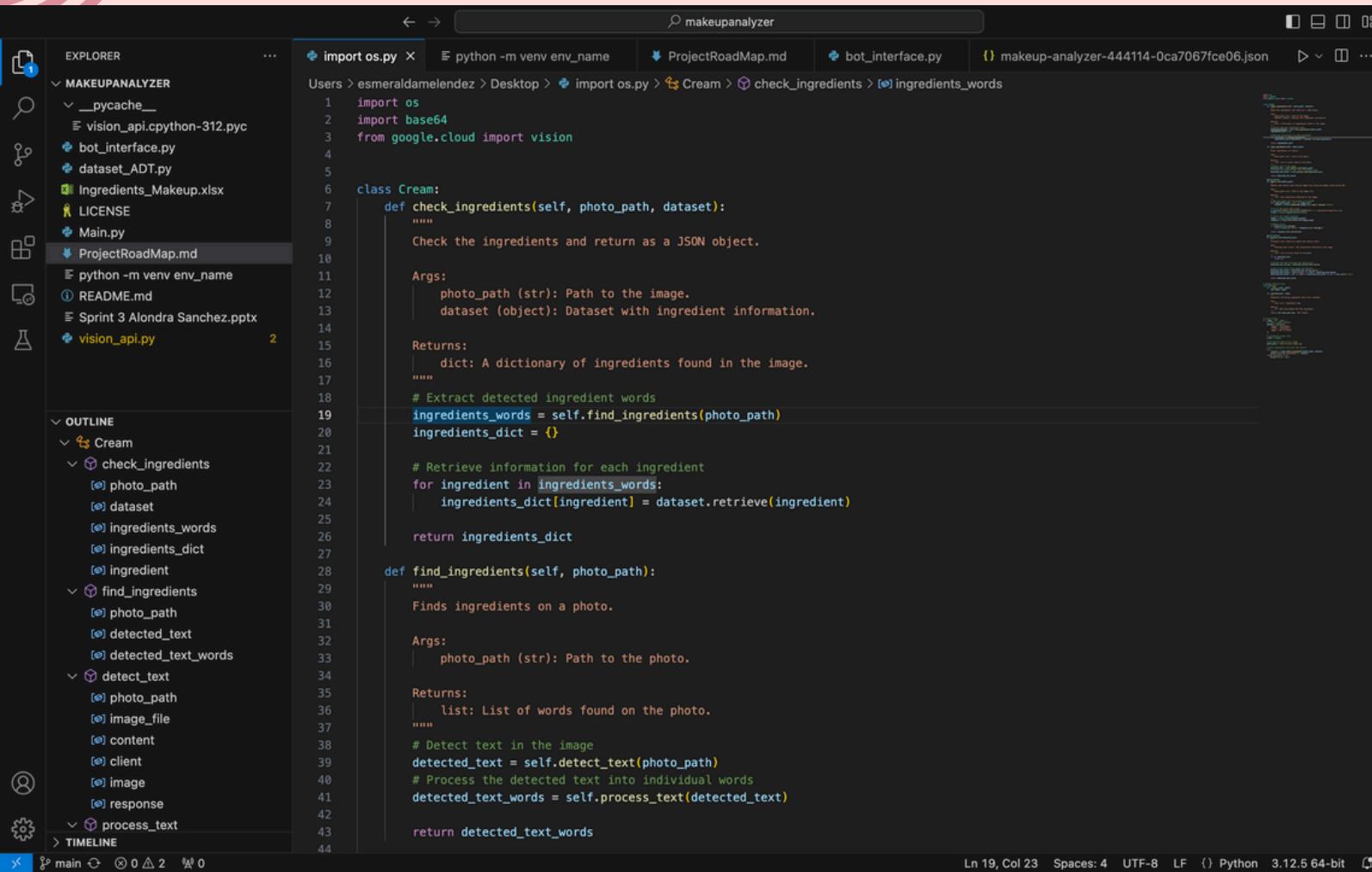
        self.dataframe = dataframe

    def is_in(self, ingredient):
        """
        Checks if the given ingredient is in the dataframe.

        Args:
            | ingredients ([str]): Ingredient name to search for.
        Returns:
            | pd.DataFrame: The rows(s) matching the ingredients, or None if not found.
        """
        #Search in the dataframe's names for the ingredient.
        matches = self.dataframe[self.dataframe['ingredient'].str.contains(ingredient, na = False)]
        return matches if not matches.empty else None

    def retrieve(self, ingredient_name):
        """
        Retrieves an entry from the dataframe correspondant to a given ingredient name.
        
```

# VISION\_API.PY



The screenshot shows a code editor window with several files open. The main file is `vision_api.py`, which contains code for a makeup analyzer. The code uses the Google Cloud Vision API to extract ingredients from images. It includes imports for `os`, `base64`, and `vision`. The `Cream` class has methods for checking ingredients and finding ingredients in a photo. The `check_ingredients` method retrieves information for each ingredient, while `find_ingredients` finds ingredients on a photo. The `detect_text` method detects text in a photo, and the `process_text` method processes the detected text into individual words. The code also includes documentation for the methods.

```
import os
import base64
from google.cloud import vision

class Cream:
    def check_ingredients(self, photo_path, dataset):
        """
        Check the ingredients and return as a JSON object.

        Args:
            photo_path (str): Path to the image.
            dataset (object): Dataset with ingredient information.

        Returns:
            dict: A dictionary of ingredients found in the image.
        """
        # Extract detected ingredient words
        ingredients_words = self.find_ingredients(photo_path)
        ingredients_dict = {}

        # Retrieve information for each ingredient
        for ingredient in ingredients_words:
            ingredients_dict[ingredient] = dataset.retrieve(ingredient)

        return ingredients_dict

    def find_ingredients(self, photo_path):
        """
        Finds ingredients on a photo.

        Args:
            photo_path (str): Path to the photo.

        Returns:
            list: List of words found on the photo.
        """
        # Detect text in the image
        detected_text = self.detect_text(photo_path)
        # Process the detected text into individual words
        detected_text_words = self.process_text(detected_text)

        return detected_text_words

    def detect_text(self, photo_path):
        """
        Finds text on a photo.

        Args:
            photo_path (str): Path to the photo.

        Returns:
            list: List of words found on the photo.
        """
        # Detect text in the image
        detected_text = self.detect_text(photo_path)
        # Process the detected text into individual words
        detected_text_words = self.process_text(detected_text)

        return detected_text_words

    def process_text(self, detected_text):
        """
        Processes the detected text into individual words.

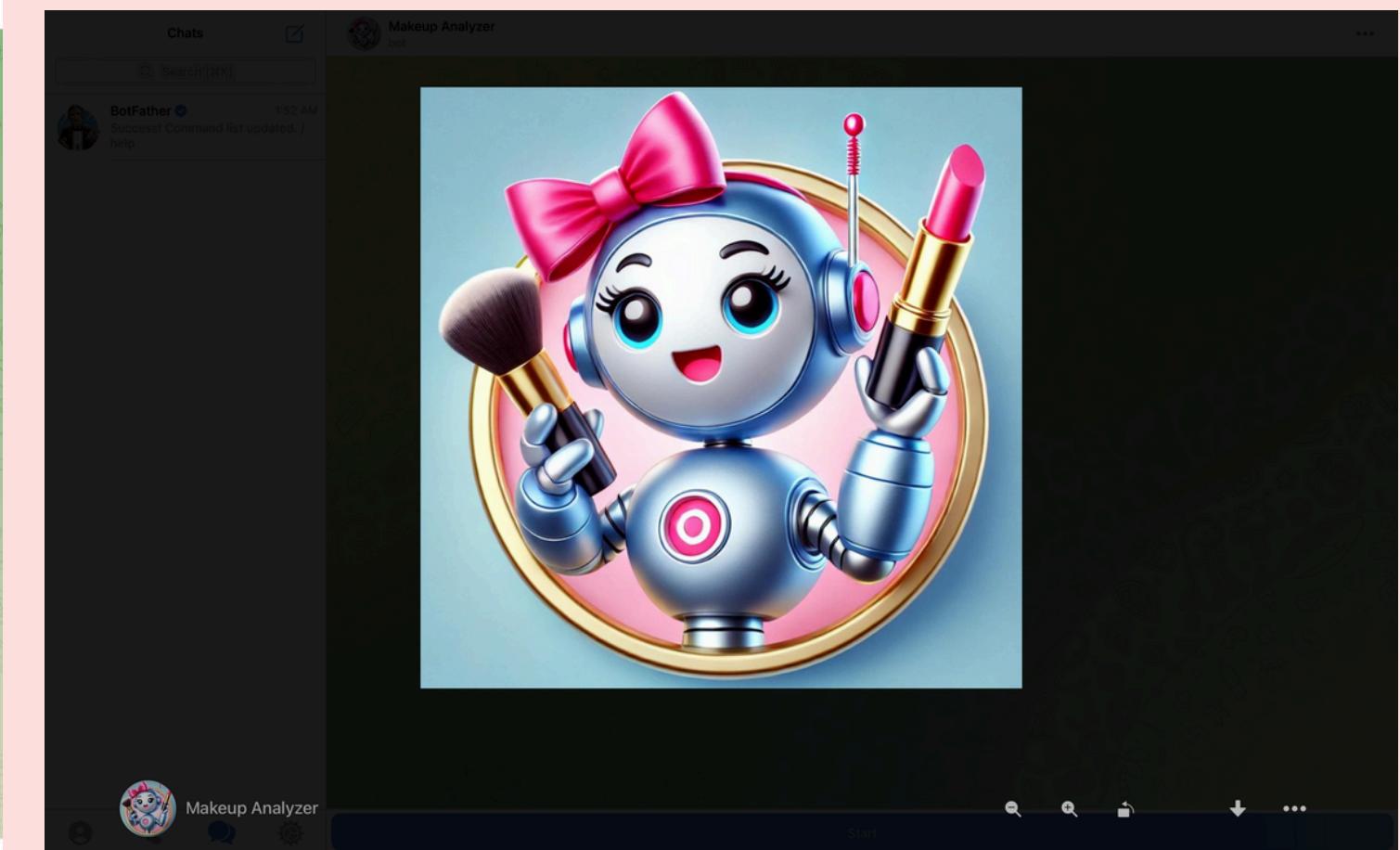
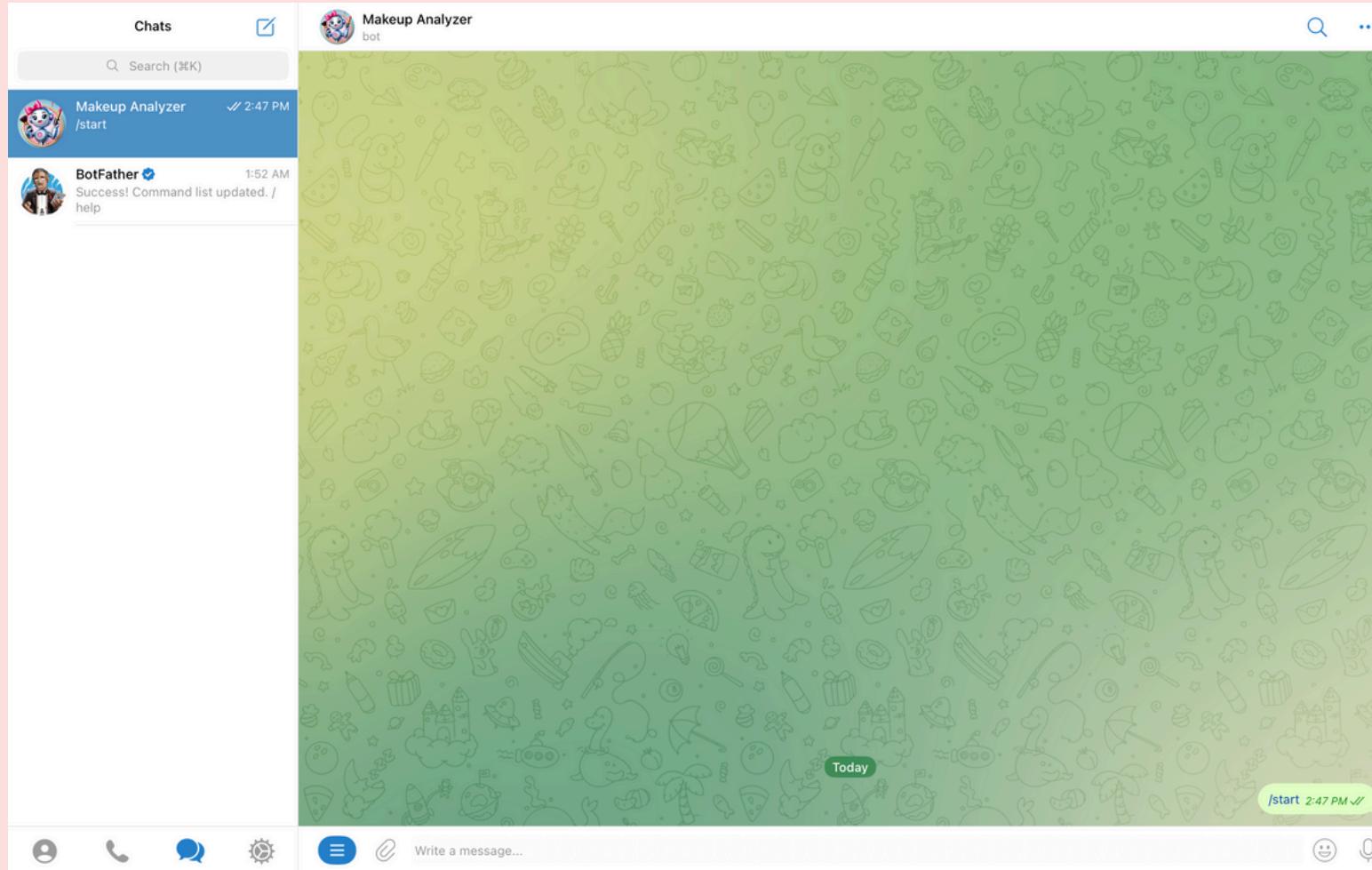
        Args:
            detected_text (list): List of words found on the photo.

        Returns:
            list: List of words found on the photo.
        """
        # Implement word processing logic here
        return detected_text
```



The Google Cloud Vision API simplifies image analysis by providing powerful pre-trained machine learning models. It enables developers to extract meaningful information from images, solving problems across a variety of industries efficiently. By providing a code the machine will generate an image analysis of what is being portrayed in the photo.

# NOW LETS RUN IT....



@makeup\_analyzer\_bot chatbox

## *Foundations*

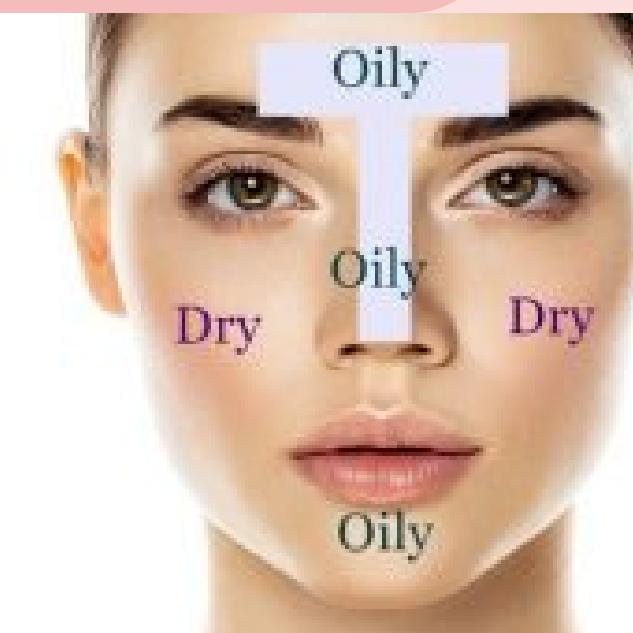
Water Based    Silicone Based



# CONCLUSION

It did not work....

Before we would be able to make anything happen, we would have to go in and make improvements to the development, the data set, and the various different interfaces we are trying to run. The next steps would be to get the product to product compatibility code implemented and working. We also would want the Bot to recommend products for both types of analyzes.



**THANK YOU**