

“Using Data Science Techniques for SCIN Data Set”

Full Process Explanation

Data Science Life Cycle:-

1. Defining SCIN Data Set

2. Defines Global Parameters for GCP

3. Dataset Schema

4. Initialize Google Cloud Storage client to Load CSV label files

5. Display the random images

6. Identify Invalid Images

7. Reverse Engineering

8. Drop One Hot Encoded Columns

9. Feature Engineering

10. Impute Missing Values & Fix Unbalanced Data

11. Dropping Unneeded Columns

12. Rename the Columns and its Values

13. Data Visualization

14. Data Extraction

The Skin Condition Image Network (SCIN) dataset offers a diverse and representative collection of skin condition images, bridging important gaps for AI development, medical research, and equitable healthcare tools.



Notebook Link is [HERE](https://colab.research.google.com/drive/10CLCmWVpNP_b-JHTG8eyCZA0iZ1fuTZF?usp=sharing)

GitHub Repo is [HERE](https://github.com/sahermuhamed1/Predictive-Modeling-for-Skin-Condition-Image-Network-SCIN-/tree/main)



Author:

Eng. Saher Muhamed

1. **Defining SCIN Data Set**

Dataset composition

he SCIN dataset currently contains over 10,000 images of skin, nail, or hair conditions, directly contributed by individuals experiencing them. All contributions were made voluntarily with informed consent by individuals in the US, under an institutional-review board-approved study. To provide context for retrospective dermatologist labeling, contributors were asked to take images both close-up and from slightly further away. They were given the option to self-report demographic information and [tanning propensity](https://en.wikipedia.org/wiki/Fitzpatrick_scale) (self-reported Fitzpatrick Skin Type, i.e., sFST), and to describe the texture, duration, and symptoms related to their concern.

One to three dermatologists labeled each contribution with up to five dermatology conditions, along with a confidence score for each label. The SCIN dataset contains these individual labels, as well as an aggregated and weighted differential diagnosis derived from them that could be useful for model testing or training. These labels were assigned retrospectively and are not equivalent to a clinical diagnosis, but they allow us to compare the distribution of dermatology conditions in the SCIN dataset with existing datasets.

Fitzpatrick scale

Before work with SCIN dataset you need to know what is Fitzpatrick scale? The Fitzpatrick scale, also known as the Fitzpatrick skin type classification, is a system for classifying human skin color based on the skin's response to ultraviolet (UV) radiation exposure. Developed by dermatologist Thomas B. Fitzpatrick in 1975, this scale helps to categorize skin types and predict the risk of sunburn and skin cancer. It is widely used in dermatology and skincare to tailor treatments and preventive measures based on skin type. preventive measures based on skin type. To know more about Fitzpatrick scale you can click [HERE](https://en.wikipedia.org/wiki/Fitzpatrick_scale)

**Monk Skin Tone Scale**

The Monk Skin Tone Scale, also known as the Monk skin type classification, is a system developed for classifying human skin tones. It aims to provide a simple and practical way to categorize skin types based on their susceptibility to sunburn and skin cancer, similar to the Fitzpatrick scale. The Monk Skin Tone Scale typically includes a range of skin types, often categorized into several groups based on their characteristics, such as Light Skin Tones, Medium Skin Tone, Dark Skin Tones, and Very Dark Skin Tones. To know more about monk skin tone scale and why it is related to Fitzpatrick you can click [HERE](https://en.wikipedia.org/wiki/Monk_Skin_Tone_Scale)

# Dataset Schema:

Dataset on Google Cloud Storage

The data is stored in Google Cloud Storage and to access it click [HERE](https://console.cloud.google.com/storage/browser/dx-scin-public-data/dataset;tab=objects?pageState=(%22StorageObjectListTable%22:(%22f%22:%22%255B%255D%22))&prefix=&forceOnObjectsSortingFiltering=false). Check out the load notebook from [HERE](https://github.com/sahermuhamed1/Predictive-Modeling-for-Skin-Condition-Image-Network-SCIN-/blob/main/load_SCIN.ipynb) for a quick review of how to access the dataset and the Dataset Documentation for an overview of its schema from [HERE](https://github.com/sahermuhamed1/Predictive-Modeling-for-Skin-Condition-Image-Network-SCIN-/blob/main/Dataset%20Description).

Please note: This dataset contains images of medical conditions, some of which may be sensitive and/or graphic in nature.

Dataset Description

**scin\_cases.csv**

* **case\_id**: Identifier for the case.
* **source**: Value is "SCIN".
* **release**: Identifier for the release of the case, formatted as "major.minor.patch".
* **year**: Year closest to the bulk of the data released.
* **age\_group**: Mapped age ranges: AGE\_18\_TO\_29, AGE\_30\_TO\_39, etc.
* **sex\_at\_birth**: User-reported sex at birth: FEMALE, MALE, OTHER\_OR\_UNSPECIFIED.
* **fitzpatrick\_skin\_type**: User-reported skin type: FST1-6, NONE\_SELECTED.
* **race\_ethnicity\_\***: User-reported race and/or ethnicity demographic.
* **textures\_\***: User-reported skin condition textures.
* **body\_parts\_\***: User-reported affected body parts.
* **condition\_symptoms\_\***: User-reported symptoms related to the skin condition.
* **other\_symptoms\_\***: User-reported additional symptoms.
* **related\_category**: User-reported related categories.
* **condition\_duration**: User-reported duration of the skin condition.
* **image\_\d\_path**: Path to the image storage location.
* **image\_\d\_shot\_type**: Enum indicating image shot type.

**scin\_labels.csv**

* **case\_id**: Same as scin\_cases.csv.
* **dermatologist\_gradable\_for\_skin\_condition**: Label indicating if skin condition can be determined.
* **dermatologist\_skin\_condition\_label\_name**: List of condition names derived from dermatologist labels.
* **dermatologist\_skin\_condition\_confidence**: Confidence scores for dermatologist labels.
* **weighted\_skin\_condition\_label**: Final differential label generated from dermatologist labels.
* **dermatologist\_gradable\_for\_fitzpatrick\_skin\_type**: Label indicating if Fitzpatrick skin type can be estimated.
* **dermatologist\_fitzpatrick\_skin\_type\_label**: Dermatologist's estimated Fitzpatrick skin type.
* **gradable\_for\_monk\_skin\_tone\_india**: Label indicating if Monk skin tone label can be determined in India.
* **gradable\_for\_monk\_skin\_tone\_us**: Label indicating if Monk skin tone label can be determined in the US.
* **monk\_skin\_tone\_label\_india**: Monk skin tone label value in India.
* **monk\_skin\_tone\_label\_us**: Monk skin tone label value in the US.

# Defines Global Parameters for GCP:

Accessing the Dataset from Google Cloud Storage

To access the dataset from Google Cloud Storage you need to define a class called Globals with several parameters related to a Google Cloud Platform (GCP) project and Google Cloud Storage (GCS) bucket. These parameters include the GCP project name, GCS bucket name, paths to CSV files containing metadata and labels, and the directory containing images within the bucket.

The class also initializes some variables like a GCS storage client, bucket object, and DataFrames for the loaded CSV files. Finally, it prints out the values of some of the parameters defined in the class.

Overall, this code is setting up global parameters and configurations for a data science or machine learning project that involves accessing data stored in a GCS bucket within a GCP project.

Create our DataFrame

After initializes a GCS client and bucket, loads ‘metadata’ and ‘labels’ CSV files from the bucket into pandas DataFrames, and merges them based on 'case\_id', finally printing the length of the merged DataFrame.

1. **Initialize Google Cloud Storage client**

# Display the random images:

Display a random Images

We defines a functions to display images associated with a case ID and optionally print condition labels. It uses the Pillow library to handle images and Matplotlib to display them. The display\_image function loads an image from Google Cloud Storage (GCS) based on a provided image path and displays it using Matplotlib. The display\_images\_for\_case function selects a random case from a DataFrame (df) based on the provided case ID (if any) or selects a random case if none is provided. It then retrieves the image paths associated with that case and displays each image using the display\_image function. Additionally, it optionally prints condition labels associated with the case.

The code is designed to work with a DataFrame (df) that contains information about cases, including their IDs, image paths, and condition labels. It seems to be part of a larger project related to image classification or analysis, where cases are associated with images and corresponding condition labels.

Identify Invalid Images

We defines a function is\_valid\_image\_path to check if an image path is valid by attempting to open the image using Pillow. It then applies this function to each image path column in the DataFrame Globals.cases\_and\_labels\_df, adding a new column for each image path column to indicate whether the image path is valid or not. Finally, it prints the first few rows of the DataFrame with the validity columns.

The missing image path is : "dataset/images/-2243186711511406658.png

1. **Identify Invalid Images**

# Reverse Engineering:

Reverse Engineering

We defines a python script that reverses one-hot encoding by mapping binary encoded columns back to their original categorical values, iterating over specified groups and creating new DataFrame columns for each group's categorical values. It updates these columns with category names where the corresponding one-hot encoded column is 'YES', effectively restoring the original categorical values

1. **Drop One Hot Encoded Columns**

Dropping Encoded Columns

After I've reversed engineered the one-hot encoded columns to obtain a normal categorical column, I delete all the encoded columns because they don't provide meaningful information to me.

I drop the encoded columns by initializing a list that is initially empty. Then, I create a for loop that iterates over all the columns in my dataframe to check if the column name starts with any prefix in the one\_hot\_groups dictionary. If it does, I add it to the list. Finally, I drop all the columns in this list."

1. **Feature Engineering**

Feature Engineering Process

In this phase we calculates the most common value for each row across multiple sets of columns related to dermatologist gradings for skin conditions and Fitzpatrick skin types in a DataFrame. It defines a function to calculate the mode for each row and applies it to each set of columns using the apply function along the rows (axis=1) of the DataFrame, generating new columns with the most common values for each set of columns.

1. **Impute Missing Values & Fix Unbalanced Data**

Feature Engineering Process

First, I Encode categorical labels using LabelEncoder to convert them into numerical values .After this, I impute missing values in the encoded column with the most frequent value using SimpleImputer. Then, I decode the imputed numerical values back to their original categorical labels using LabelEncoder. Finally, I updated the DataFrame with the imputed categorical values in a new column.

1. **Dropping Unneeded Columns**

**Dropping Some Columns**

because they either contain mostly null values or are not relevant for your analysis. The indices correspond to the following columns:

* **‘case\_id’, source, release, year:** These seem to be identifiers or metadata that are not necessary for your analysis.
* **‘race\_ethnicity\_two\_or\_more\_after\_mitigation’**: Contains mostly null values.
* **‘dermatologist\_gradable\_for\_skin\_condition\_’:** These columns have many null values and might not be useful for your analysis.
* **‘dermatologist\_skin\_condition\_on\_label\_name’, ‘dermatologist\_skin\_condition\_confidence, weighted\_skin\_condition\_label’, ‘dermatologist\_fitzpatrick\_skin\_type\_label\_’:** Likely not relevant or redundant for your analysis.
* **‘gradable\_for\_monk\_skin\_tone\_’,’ monk\_skin\_tone\_label\_’:** These columns seem specific to certain analyses and are not relevant to your current task.

By dropping these columns, you're likely simplifying your dataset to focus on the most relevant features for your analysis, which can improve model performance and reduce computational overhead.

# Renaming columns:

Renaming columns and its values

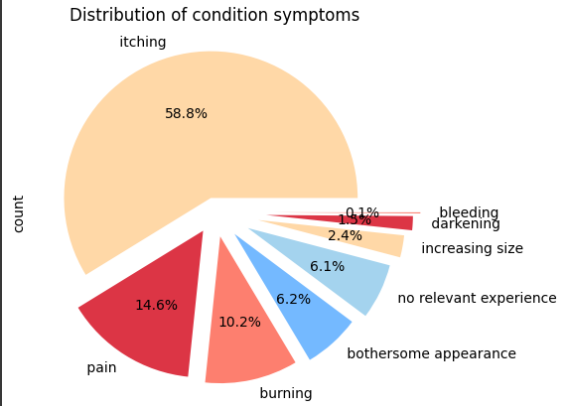
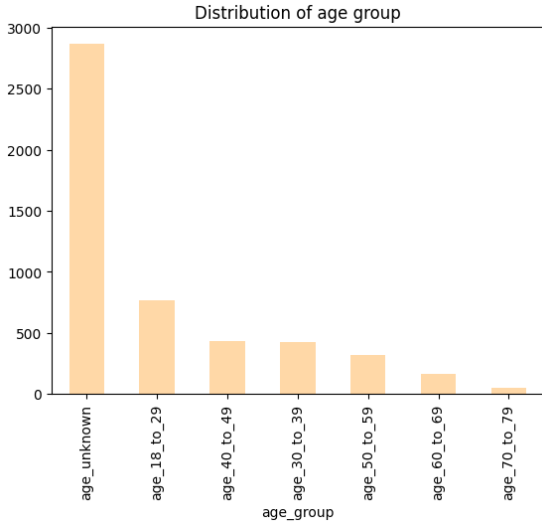
In this step, I rename the unclear column names to make them easier to recognize. I also rename values in some columns to clarify their meaning for the reader.

Example :

Rename columns to more descriptive names, e.g., 'fitzpatrick\_skin\_type' to 'fitzpatrick\_scale', and 'image\_1\_path' to 'image\_name'.

Split and clean the 'textures' column to remove the prefix and convert values to lowercase.

Replace underscores in the 'textures' column with spaces for readability.



Some Data Viz

In this step, I created visuals to gain insights, such as detecting data bias, identifying missing entries, and determining if any changes are needed in the DataFrame.

Visuals example:

1. **Data Visulization**

# Data Extraction:

Data Extraction

**This step contains:**

* Create a function to delete records from the DataFrame that don't have corresponding images in the image folder, based on the image\_name column.
* Validate that each record has an image in the images folder.
* Preprocess the image data.
* Get 15 valid sample records.

Create another DataFrame called Val\_samples to store these samples for further research.



“Thank You”

For Further Collaboration Projects or Research you can contact me:  
Email: [sahermuhamed176@gmail.com](mailto:sahermuhamed176@gmail.com)

Phone: +20 01068799655

ENG. **Saher Muhamed**

Data Scientist and Machine Lerning Engineer

