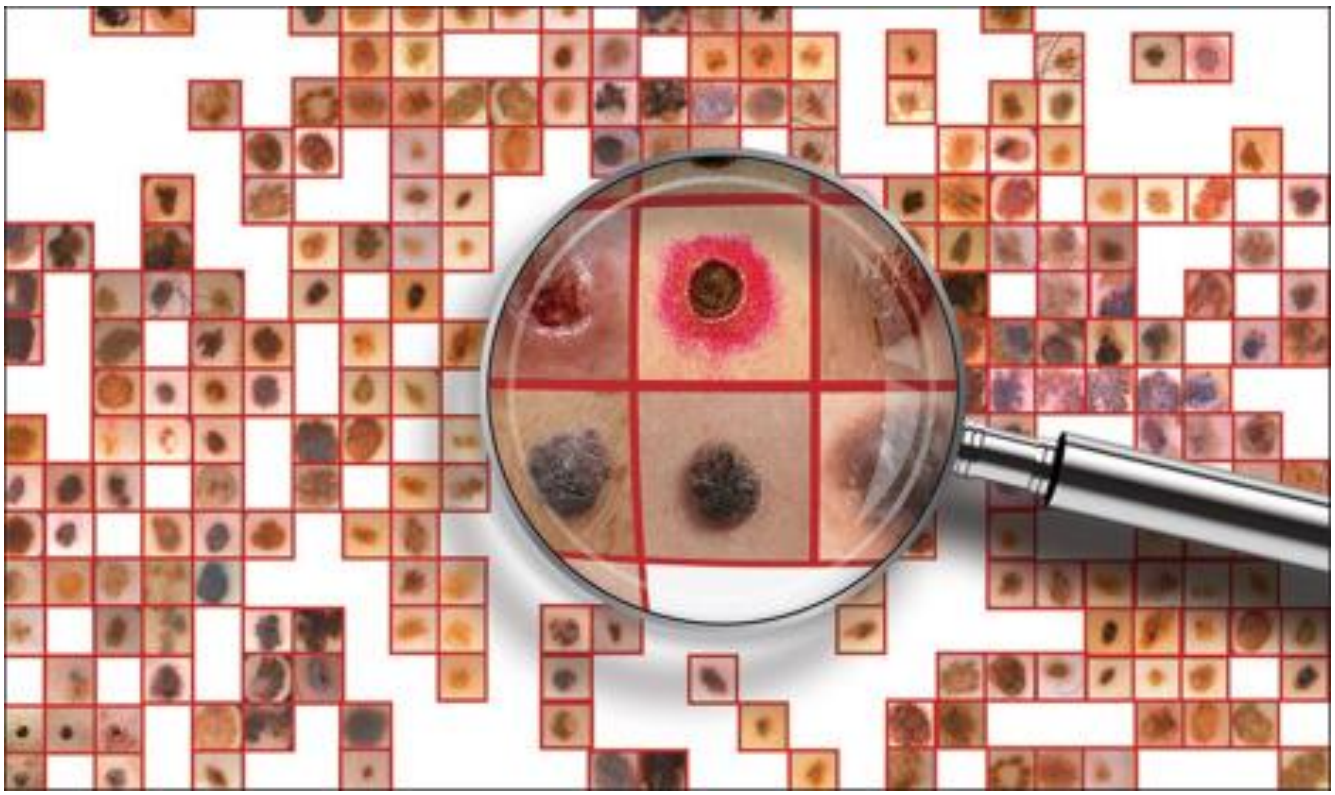


# Skin cancer detection

DEPI graduation project


Done by:

- Ahmad Mohamed Abdel Galeel
- Amira Emad Abdel Wahab

# Table of content

# 1. Introduction

Detecting skin cancer early can be a matter of life or death, yet accurately diagnosing it remains a challenge for even the most trained eye. What if we could harness the power of machine learning to assist in this critical task? This project takes on that challenge by developing a cutting-edge system that analyzes both skin images and patient metadata to predict the likelihood of skin cancer.

By leveraging advanced machine learning techniques and deploying the model on AWS, this project empowers healthcare professionals with an intuitive tool that offers fast, reliable predictions. Through a simple web interface, users can upload an image of a skin lesion and/or provide patient information. In seconds, the system delivers a diagnosis — either reinforcing a clinical decision or guiding further investigation. This documentation will walk through how the project was built, the data behind it, the preprocessing steps, the machine learning model, and the deployment process.

# 2. Dataset

The dataset used in this project comes from the **ISIC 2024** archive, which is a globally recognized repository for skin lesion images. It contains diagnostically labeled skin lesion images along with comprehensive metadata. The combination of image and metadata allows for a more powerful machine-learning model, using both visual and patient data to improve skin cancer detection accuracy.

## 2.1 Dataset Composition

The dataset consists of:

- **401,059 images** of skin lesions.

  Each image in the dataset is diagnostically labeled with a binary classification:
  0: Benign (non-cancerous)
  1: Malignant (cancerous)

- **Metadata** fields describing patient demographics, lesion characteristics, and additional technical information.

Each category below provides a view of either the lesion's physical appearance, location, or model-derived information, contributing to a more comprehensive understanding of the data. The color and texture information help differentiate healthy tissue from potential malignancies, while the shape and border characteristics can reveal irregularities typical of cancerous growths.

| | |
|---|---|
| **Patient Information**<br>These fields describe general details about the patient. | |
| isic_id | Unique case identifier. |
| patient_id | Unique patient identifier. |
| age_approx | Approximate age of the patient at the time of imaging. |
| sex | Sex of the patient. |
| anatom_site_general | General location of the lesion on the patient's body. |
| **Lesion Characteristics**<br>These fields provide detailed information about the lesion's physical features. | |
| clin_size_long_diam_mm | Maximum diameter of the lesion (in mm). |
| tbp_lv_A | A color channel inside the lesion. |
| tbp_lv_Aext | A color channel outside the lesion. |
| tbp_lv_B | B color channel inside the lesion. |
| tbp_lv_Bext | B color channel outside the lesion. |
| **Color and Texture Information**<br>These fields describe the lesion's color and contrast compared to the surrounding skin. | |
| tbp_lv_C<br>tbp_lv_Cext | Chroma inside and outside the lesion. |
| tbp_lv_H<br>tbp_lv_Hext | Hue inside and outside the lesion. |
| tbp_lv_L<br>tbp_lv_Lext | Lightness inside and outside the lesion. |

| | |
|---|---|
| tbp_lv_deltaA<br>tbp_lv_deltaB<br>tbp_lv_deltaL | Contrast between the inside and outside of the lesion. |
| tbp_lv_deltaLBnorm | Contrast between the lesion and surrounding skin. |
| **Shape and Size Information**<br>These fields describe the geometry of the lesion. | |
| tbp_lv_areaMM2 | Area of the lesion ($mm^2$) |
| tbp_lv_area_perim_ratio | Ratio of the perimeter to area, indicating shape irregularity. |
| tbp_lv_minorAxisMM | Smallest lesion diameter (mm) |
| tbp_lv_perimeterMM | Perimeter of the lesion. |
| **Border and Asymmetry Characteristics**<br>These fields describe the lesion's border regularity and asymmetry. | |
| tbp_lv_norm_border | Border irregularity score (0–10) |
| tbp_lv_symm_2axis<br>tbp_lv_symm_2axis_angle | Border symmetry and angle measurements. |

## 3. Data pre-processing

Data preprocessing is a crucial step in preparing the dataset for training a machine learning model. In this project, preprocessing involves handling both the image data and the accompanying metadata.

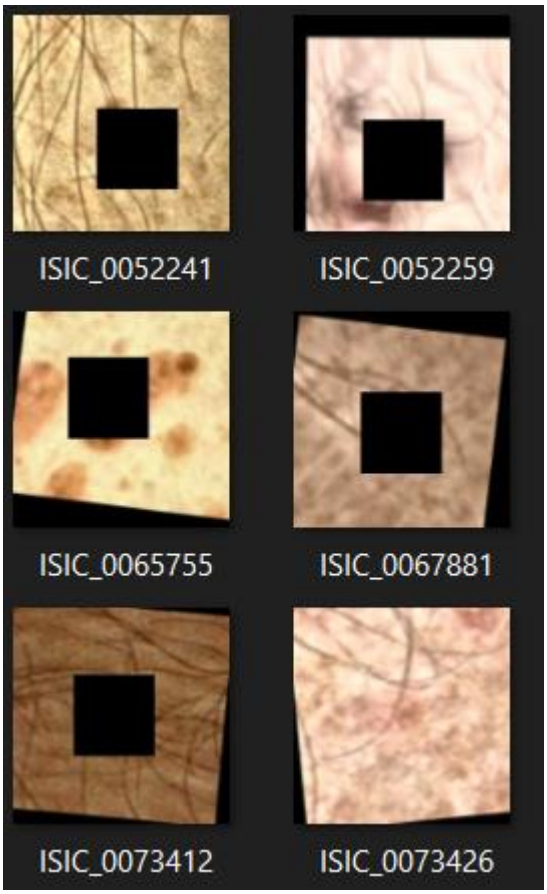
### 3.1 Images augmentation

In this project, we used the **Albumentations** library for image augmentation, applying a series of transformations to simulate real-world conditions such as flipping, brightness adjustments, noise addition, and distortions.

```
# Set the image size for resizing during augmentation
image_size = 224
```

```
# Define the set of augmentation transformations to apply
transforms_train = A.Compose([
    A.Transpose(p=0.5),  # Randomly transpose the image (swap rows and columns)
    A.VerticalFlip(p=0.5),  # Random vertical flip
    A.HorizontalFlip(p=0.5),  # Random horizontal flip
    A.RandomBrightnessContrast(brightness_limit=0.2, p=0.75),  # Adjust brightness and contrast
    A.OneOf([  # Apply one of the following blur/noise augmentations
        A.MotionBlur(blur_limit=5),
        A.MedianBlur(blur_limit=5),
        A.GaussianBlur(blur_limit=5),
        A.GaussNoise(var_limit=(5.0, 30.0)),
    ], p=0.7),
    A.OneOf([  # Apply one of the following distortion transformations
        A.OpticalDistortion(distort_limit=1.0),
        A.GridDistortion(num_steps=5, distort_limit=1.),
        A.ElasticTransform(alpha=3),
    ], p=0.7),
    A.CLAHE(clip_limit=4.0, p=0.7),  # Apply CLAHE (Contrast Limited Adaptive Histogram Equalization)
    A.HueSaturationValue(hue_shift_limit=10, sat_shift_limit=20, val_shift_limit=10, p=0.5),  # Adjust color properties
    A.ShiftScaleRotate(shift_limit=0.1, scale_limit=0.1, rotate_limit=15, border_mode=0, p=0.85),  # Shift, scale, and rotate the image
    A.Resize(image_size, image_size),  # Resize image to 224x224
    A.CoarseDropout(max_holes=1, max_height=int(image_size * 0.375), max_width=int(image_size * 0.375), fill_value=0, p=0.7),  # Randomly mask out part of the image
])
```

Sample of images before and after augmentation:

| Before | After |
| --- | --- |
|  |  |



## 3.2 Metadata pre-processing

This section covers how we transformed raw metadata into features that are more suitable for machine learning models. We also handle missing data and transform categorical variables into a format suitable for training the model.

- **Feature Engineering**
  We create additional features to capture patterns in the data. These features include ratios, contrasts, and composite indices to provide more insight into the lesion characteristics.

```
# Function to engineer new features from the existing metadata
def feature_engineering(df):
    # New features based on existing columns
    df["lesion_size_ratio"] = df["tbp_lv_minorAxisMM"] /
df["clin_size_long_diam_mm"]
    df["lesion_shape_index"] = df["tbp_lv_areaMM2"] /
(df["tbp_lv_perimeterMM"] ** 2)
    df["hue_contrast"] = (df["tbp_lv_H"] - df["tbp_lv_Hext"]).abs()
    df["luminance_contrast"] = (df["tbp_lv_L"] -
df["tbp_lv_Lext"]).abs()
    df["lesion_color_difference"] = np.sqrt(df["tbp_lv_deltaA"] ** 2 +
df["tbp_lv_deltaB"] ** 2 + df["tbp_lv_deltaL"] ** 2)
    # Additional complex features
    df["color_uniformity"] = df["tbp_lv_color_std_mean"] /
df["tbp_lv_radial_color_std_max"]
    df["3d_position_distance"] = np.sqrt(df["tbp_lv_x"] ** 2 +
df["tbp_lv_y"] ** 2 + df["tbp_lv_z"] ** 2)
    df["perimeter_to_area_ratio"] = df["tbp_lv_perimeterMM"] /
df["tbp_lv_areaMM2"]

    # Return the updated dataframe and new feature column names
    new_num_cols = [
        "lesion_size_ratio", "lesion_shape_index", "hue_contrast",
        "luminance_contrast", "lesion_color_difference",
"color_uniformity",
        "3d_position_distance", "perimeter_to_area_ratio"
    ]
    return df, new_num_cols

# Apply feature engineering
df_train, new_num_cols = feature_engineering(df_train)
```

- **Handling Missing Values**
  Missing numerical values are filled using the median, ensuring the dataset is complete and ready for model training.

```
# Numerical columns to fill missing values with the median
num_cols = [
    'age_approx', 'clin_size_long_diam_mm', 'tbp_lv_A', 'tbp_lv_Aext',
    'tbp_lv_B', 'tbp_lv_Bext', 'tbp_lv_H', 'tbp_lv_Hext', 'tbp_lv_L',
    'tbp_lv_Lext', 'tbp_lv_areaMM2', 'tbp_lv_minorAxisMM',
```

```
'tbp_lv_perimeterMM',
]

# Fill missing numerical values with the median
df_train[num_cols] =
df_train[num_cols].fillna(df_train[num_cols].median())
```

- **Dropping Irrelevant Columns**
  We drop metadata columns that are irrelevant to our analysis to simplify the dataset.

```
# Drop irrelevant columns
columns_to_drop = ['image_type', 'attribution', 'copyright_license',
'tbp_lv_location', 'iddx_5']

df_train.drop(columns=columns_to_drop, inplace=True)
```

- **One-Hot Encoding for Categorical Data**
  We transform categorical variables into one-hot encoded features, making them suitable for machine learning models.

```
# Categorical columns to be one-hot encoded
cat_cols_1hot = ["tbp_tile_type", "anatom_site_general",
"tbp_lv_location_simple"]

# One-Hot Encoding for categorical variables
encoder = OneHotEncoder(sparse=False, handle_unknown='ignore')
encoder.fit(df[cat_cols_1hot])

new_cat_cols=encoder.get_feature_names_out(cat_cols_1hot))
df[new_cat_cols] = encoder.transform(df[cat_cols_1hot])

# onehot encoding to sex column
df['sex'] = df['sex'].map({'male': 1, 'female': 0})
df['sex'] = df['sex'].fillna(-1)
```

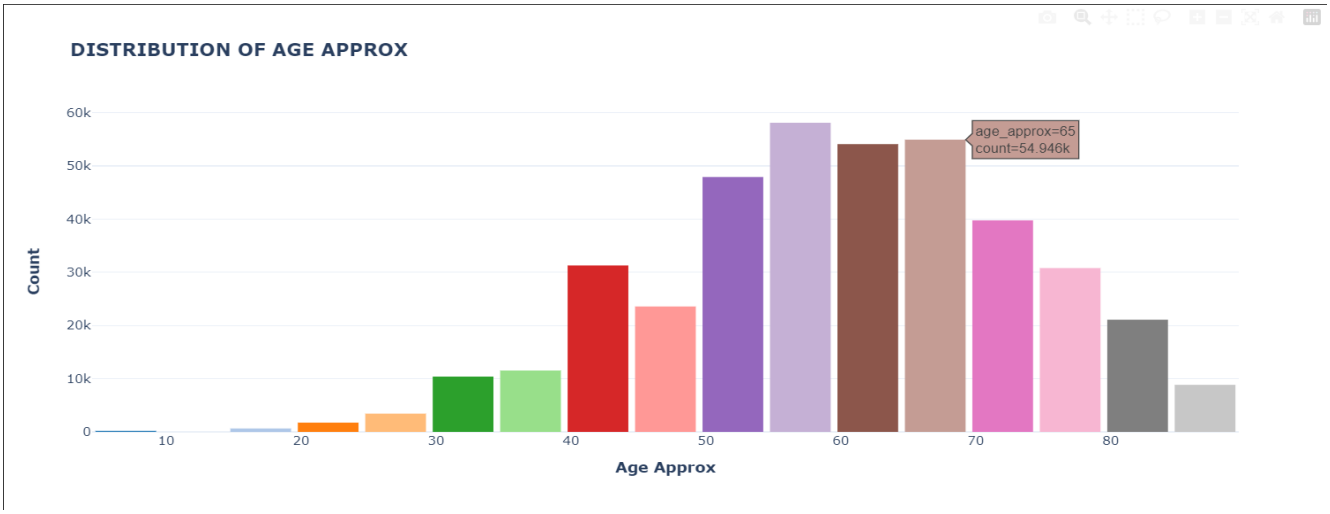
- **Handling NAN values for Categorical Data**
  We fill Nan values with 'unknown' in category columns, making them suitable for machine learning models.

```
# Now fill NaN values with 'unknown' in caegory columns
    df['patient_id'] = df['patient_id'].fillna('unknown')
    df['lesion_id'] = df['lesion_id'].fillna('unknown')
    df['iddx_full'] = df['iddx_full'].fillna('unknown')
    df['iddx_1'] = df['iddx_1'].fillna('unknown')
    df['iddx_2'] = df['iddx_2'].fillna('unknown')
    df['iddx_3'] = df['iddx_3'].fillna('unknown')
    df['iddx_4'] = df['iddx_4'].fillna('unknown')
    df['mel_mitotic_index'] = df['mel_mitotic_index'].fillna('unknown')
```

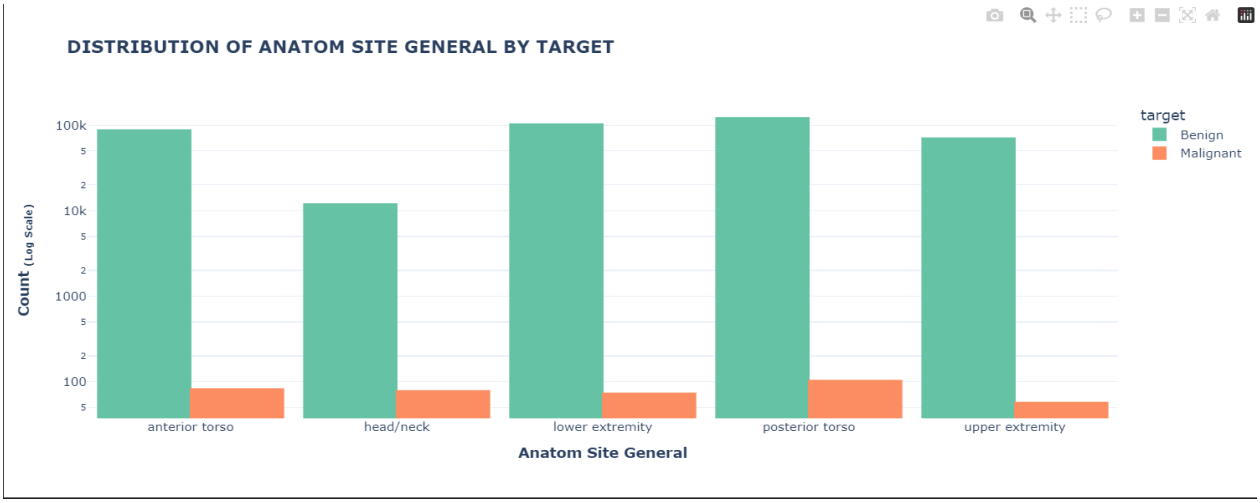
# 4. Descriptive analysis

This section presents a descriptive analysis of the dataset, including the distribution of data and the probability of melanoma with respect to age and sex.

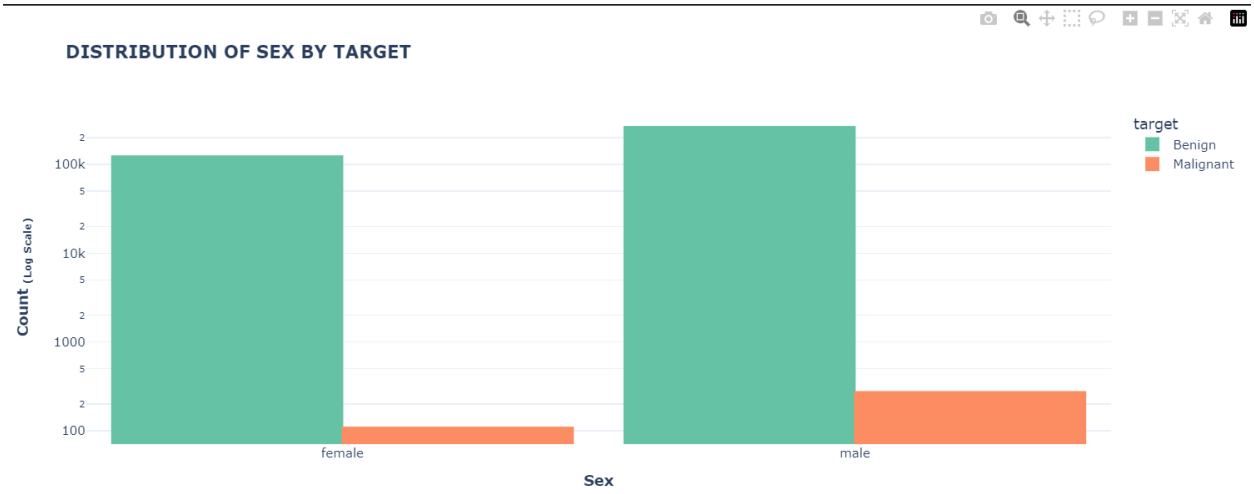
## 4.1 DISTRIBUTION OF AGE APPROX



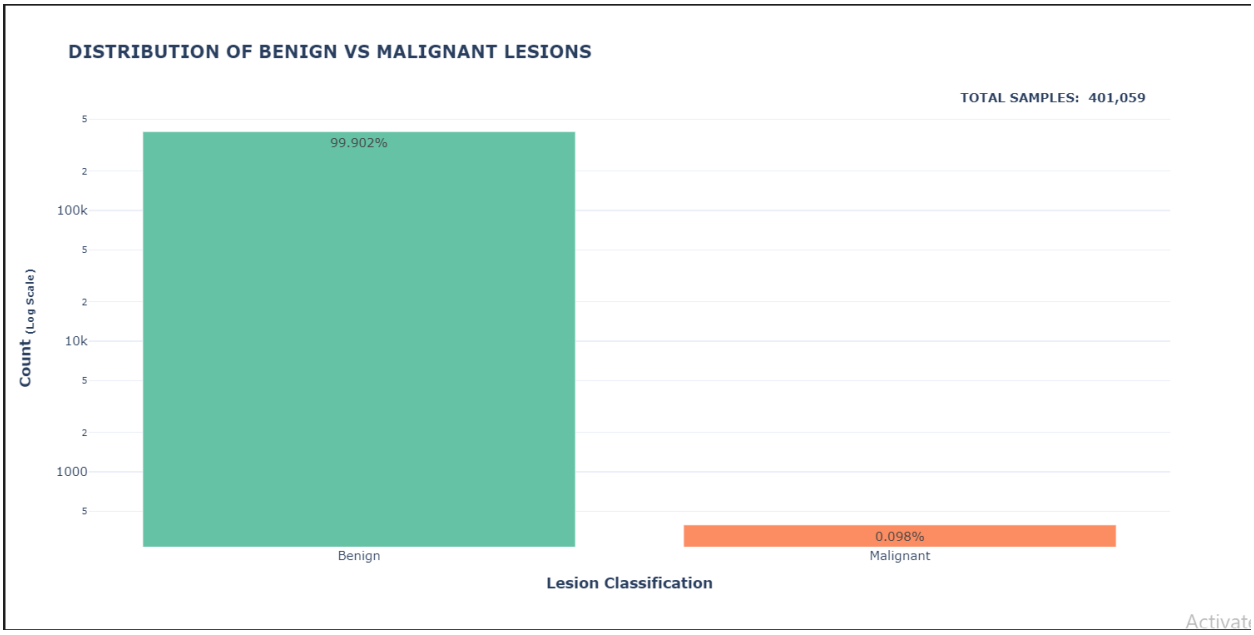
## 4.2 DISTRIBUTION OF ANATOM SITE GENERAL BY TARGET



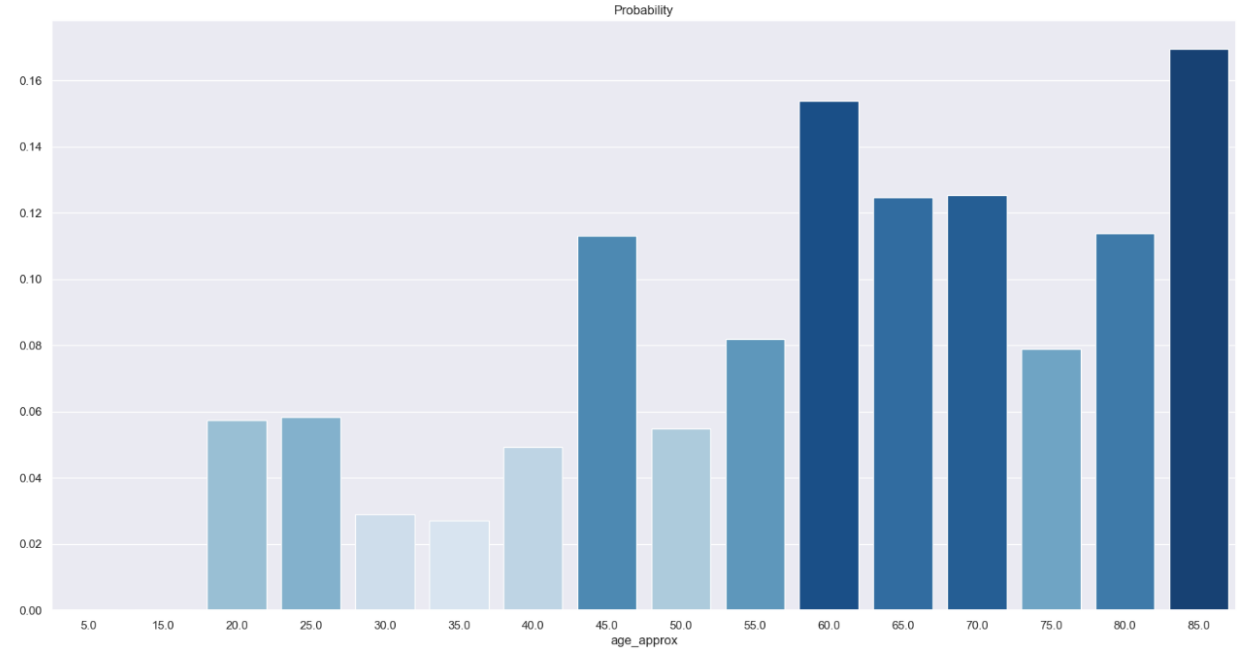

## 4.3 DISTRIBUTION OF SEX BY TARGET



## 4.4 DISTRIBUTION OF BENGIN VS. MALIGNANT LESIONS

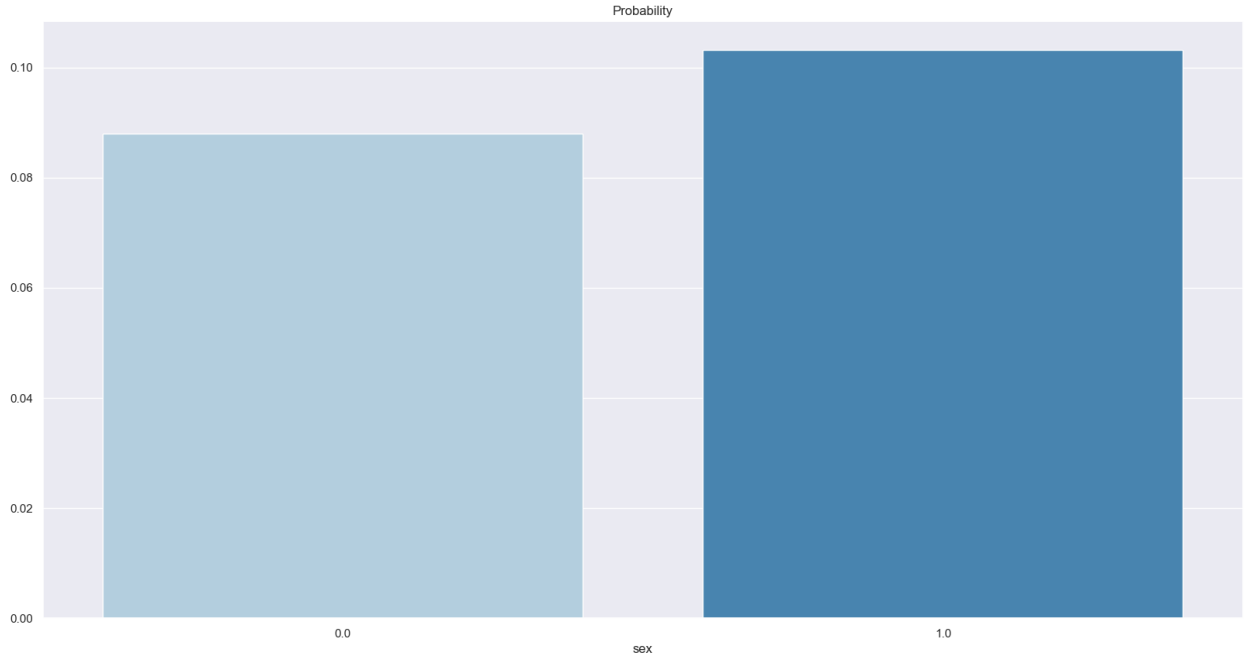



## 4.5 Probability Of Melanoma By Age. Approx



## 4.6 Probability Of Melanoma By Sex

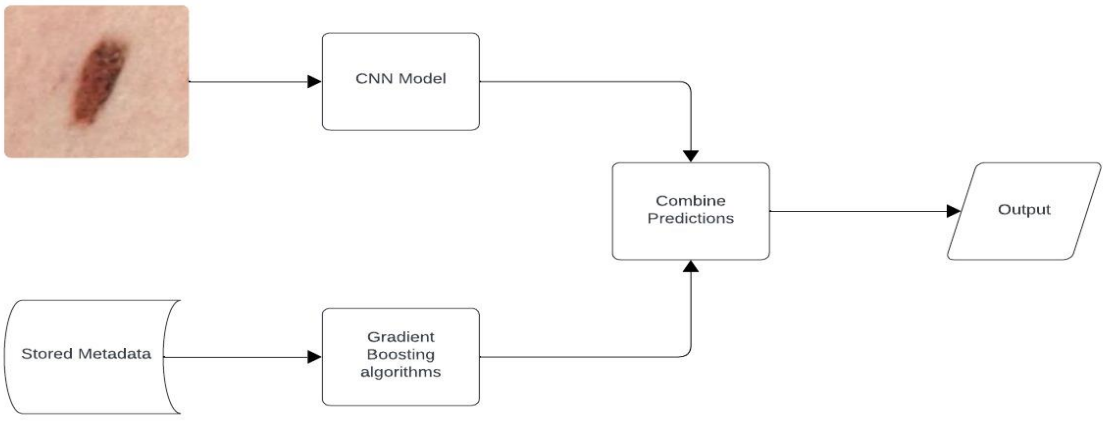
Where {'Male': 1, 'Female': 0}




# 5. Machine learning model

This section describes the architecture and process of our machine-learning model, which integrates image data and structured metadata to predict the likelihood of skin cancer lesions. The model comprises two primary components:

- **Convolutional Neural Network (CNN) for Image Data**: This component processes the image of the lesion to extract relevant features.
- **Gradient Boosting Algorithm for Metadata**: This component utilizes structured metadata associated with the lesion to enhance the predictive capabilities.



## 5.1 CNN for Image Data

The CNN is designed to extract features from the input lesion images. We decided to use **EfficientNet_B0** in particular because:

- **Its ability to perform well in limited-resource environments** as it employs a unique compound scaling method that balances the model's depth, width, and resolution. This allows for optimized performance without unnecessarily increasing computational costs.
- It achieves competitive accuracy on benchmarks while **using significantly fewer parameters** compared to other architectures

```
class EfficientNetTrainer:
    def __init__(self, num_classes, learning_rate=1e-3,
checkpoint_path='efficientnet_checkpoint.pth'):
        # Load EfficientNet with pre-trained weights
        self.model = models.efficientnet_b0(pretrained=True)

        # Replace the last fully connected layer to match the number of
classes
        self.model.classifier[1] =
nn.Linear(self.model.classifier[1].in_features, num_classes)

        # Move the model to the selected device (GPU/CPU)
        self.device = torch.device('cuda' if torch.cuda.is_available() else
'cpu')
        self.model = self.model.to(self.device)

        # Define loss function and optimizer
        self.criterion = nn.CrossEntropyLoss()
        self.optimizer = optim.Adam(self.model.parameters(),
lr=learning_rate)

        # Initialize tracking variables for loss plotting and checkpointing
        self.train_losses = []
        self.val_accuracies = []
        self.checkpoint_path = checkpoint_path
```

## 5.2 Gradient Boosting Algorithm for Metadata

Using a gradient boosting algorithm effectively captures complex relationships within the metadata features by doing the following:

- **Boosting Iterations** that involves sequentially fitting weak learners to minimize prediction errors, where each new learner aims to correct the mistakes of the previous ones.
- Then **Output Layer** combines the predictions of these weak learners into a final prediction, leading to improved accuracy and robustness compared to individual weak model

Here is the implementation Example of the gradient boosting model using CatBoost:

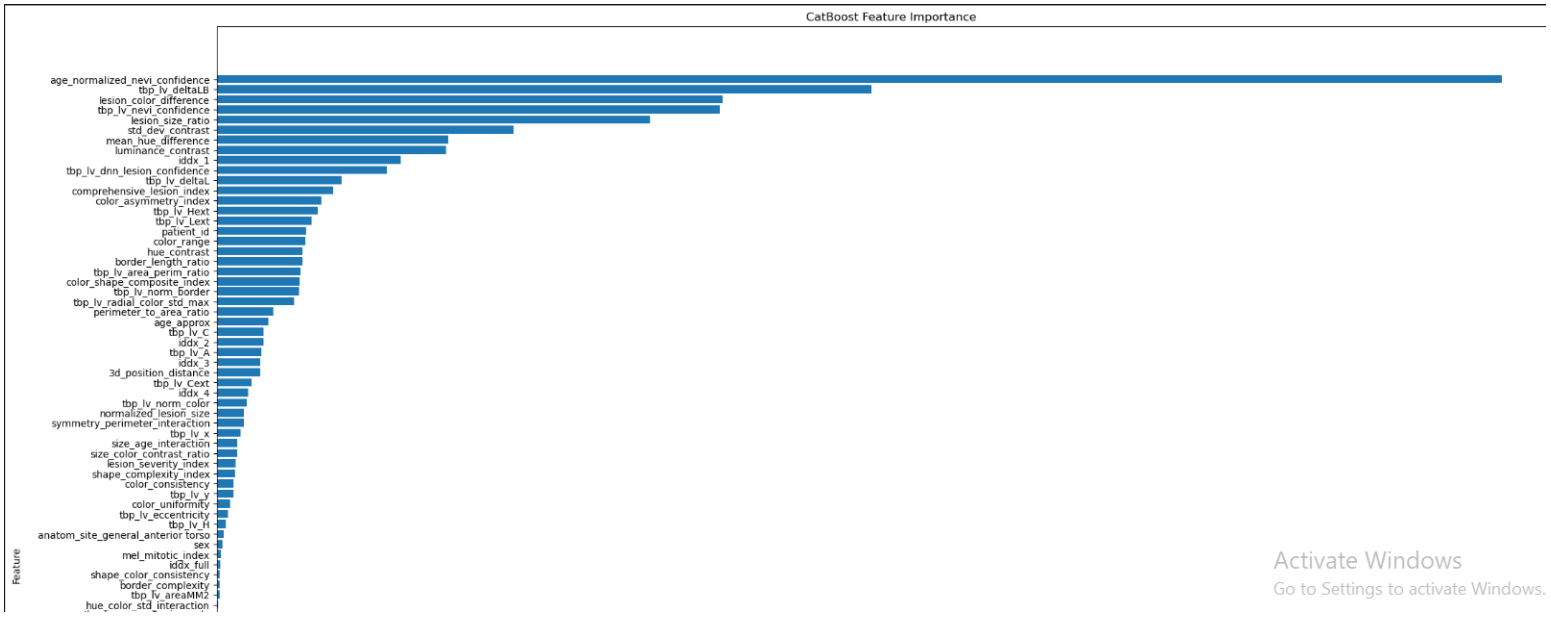
```
# Create a CatBoost Pool for training
train_pool = Pool(data=X_train, label=y_train,
cat_features=categorical_features)

test_pool = Pool(data=X_test, label=y_test,
cat_features=categorical_features)

# Initialize the CatBoost model
model = CatBoostClassifier(iterations=100,  # Adjust as needed
        max_depth=7,
        learning_rate=0.01,
        # min_data_in_leaf=24,
        eval_metric='Logloss',  # Main metric
        use_best_model=True,
        custom_metric=['F1', 'AUC'],  # Additional metrics
    )
)

# Train the model
model.fit(train_pool)

# Make predictions
y_pred = model.predict(test_pool)
```

## 5.3 Handling Imbalanced Data

To tackle data imbalance as we saw in chart ***Distribution o bengin VS. malignant lesions***, two main approaches were used to split data:

1. **Initial Split with train_test_split**:
    - Initially, I used `train_test_split` to divide the dataset into training and validation sets.
    - After training with this split, I plotted feature importance to understand the model's initial perception of key features.
    - This method, however, risked over-representing certain features due to imbalances, leading to unreliable importance weights.

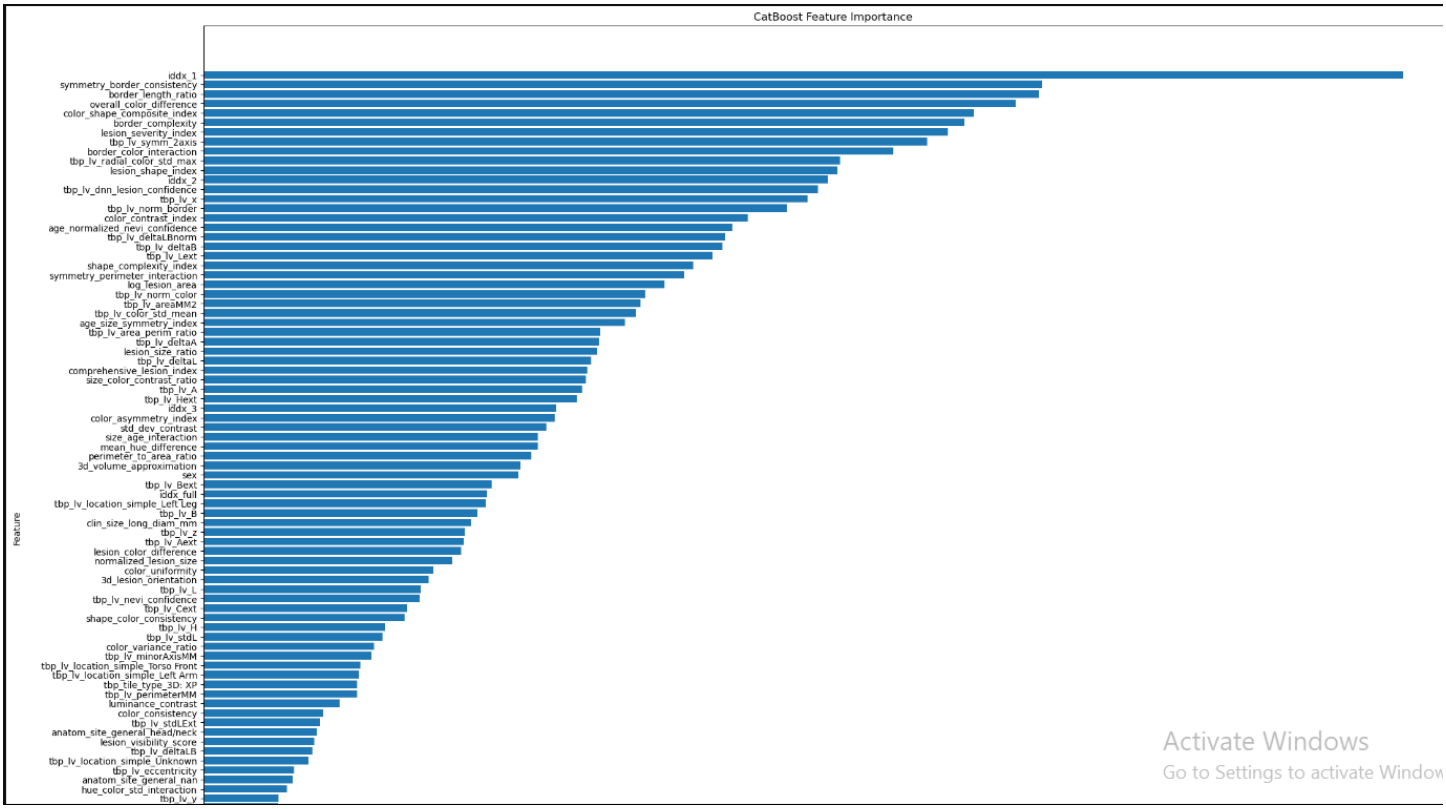
Feature importance using initial Split with train_test_split






2. **Stratified Group K-Fold with Patient IDs**:
    - To achieve more balanced training, I switched to a `StratifiedGroupKFold` approach, ensuring that each fold had a similar distribution of classes and was grouped by patient ID to prevent data leakage.
    - Using this approach, I plotted feature importance again. The K-Fold split led to a more evenly distributed feature importance, highlighting a wider range of significant features and reducing the bias toward over-represented samples.

## Feature importance using initial Split with Stratified Group K-Fold with Patient IDs




In comparing the CatBoost feature importance before and after applying StratifiedGroupKFold, there are several observations:

1. **Stability of Feature Rankings**: After using K-fold, the feature importances tend to be more evenly distributed, especially among top features.
2. **Key Feature Consistency**: In both visualizations, certain features like "age_normalized_nevi_confidence" and "lesion_color_difference" remain highly important, but their relative importance appears more balanced after K-fold, suggesting that their dominance might have been exaggerated in a single train-test split.

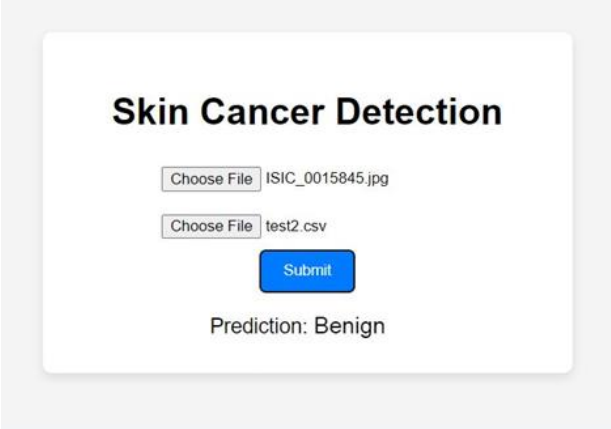
## 5.4 Combining Predictions

The outputs from both the CNN and Gradient Boosting models are combined to make a final prediction regarding the likelihood of skin cancer.

# 6. Deployment and user interaction

The skin cancer detection system is deployed on AWS, utilizing several core services to handle image and metadata inputs through a webpage, process them through machine learning models, and return a prediction.

## 6.1 User interface

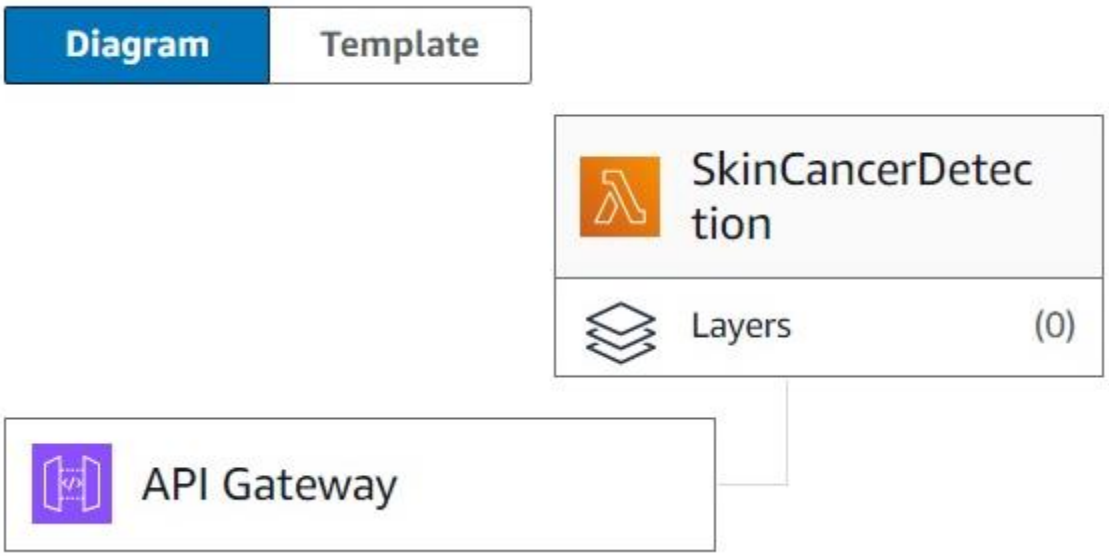




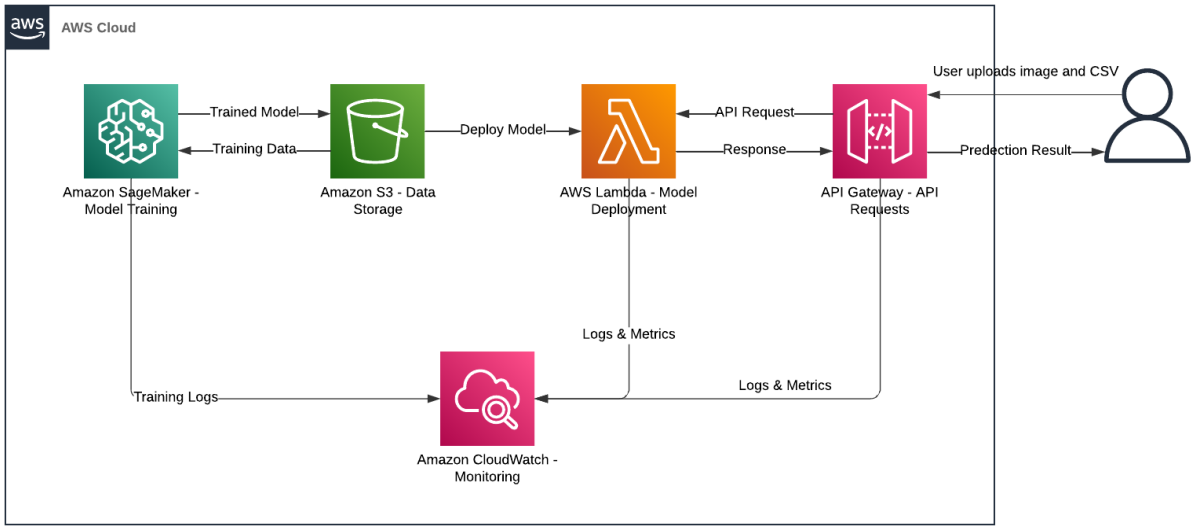
## 6.2 Workflow Summary

1- The Lambda function, triggered by an API Gateway request, receives base64-encoded inputs.
2- Decoding and preprocessing the data.
3- Loading and running two models for image and metadata analysis.
4- Combining the results through weighted averaging.

5- Sending back a JSON response with the final prediction.




## 6.3 Workflow Diagram



## 7. Data and Code Access

- **Project source code: https://shorturl.at/szCTj**
- **Dataset: https://shorturl.at/uZAC5**
- **Pre-processed dataset: https://shorturl.at/8q4y8**