Uncovering Factors Linked to Hair Loss: A Data-Driven Analysis

Erika Li

Introduction

The dataset used in this project, taken from the Hair Health Prediction Dataset on Kaggle, contains various features related to hair health, including genetic factors, stress levels, age, smoking habits, nutritional deficiencies, and more. The goal of this analysis is to explore the relationship between these factors and the occurrence of hair loss. The dataset is particularly interesting because it allows us to understand how different lifestyle and environmental factors may influence hair health, which could be beneficial for people seeking preventative measures or treatments.

Hypothesis

My hypothesis is that lifestyle and genetic factors, such as family history, stress, smoking, and age have a significant impact on the likelihood of experiencing hair loss. I hypothesize that it is possible to build a classifier that is able to use an individual's data, including lifestyle and genetic factors, to predict their hair loss. Given the increasing prevalence of hair loss, particularly in younger populations, understanding these factors could provide insights into preventive measures.

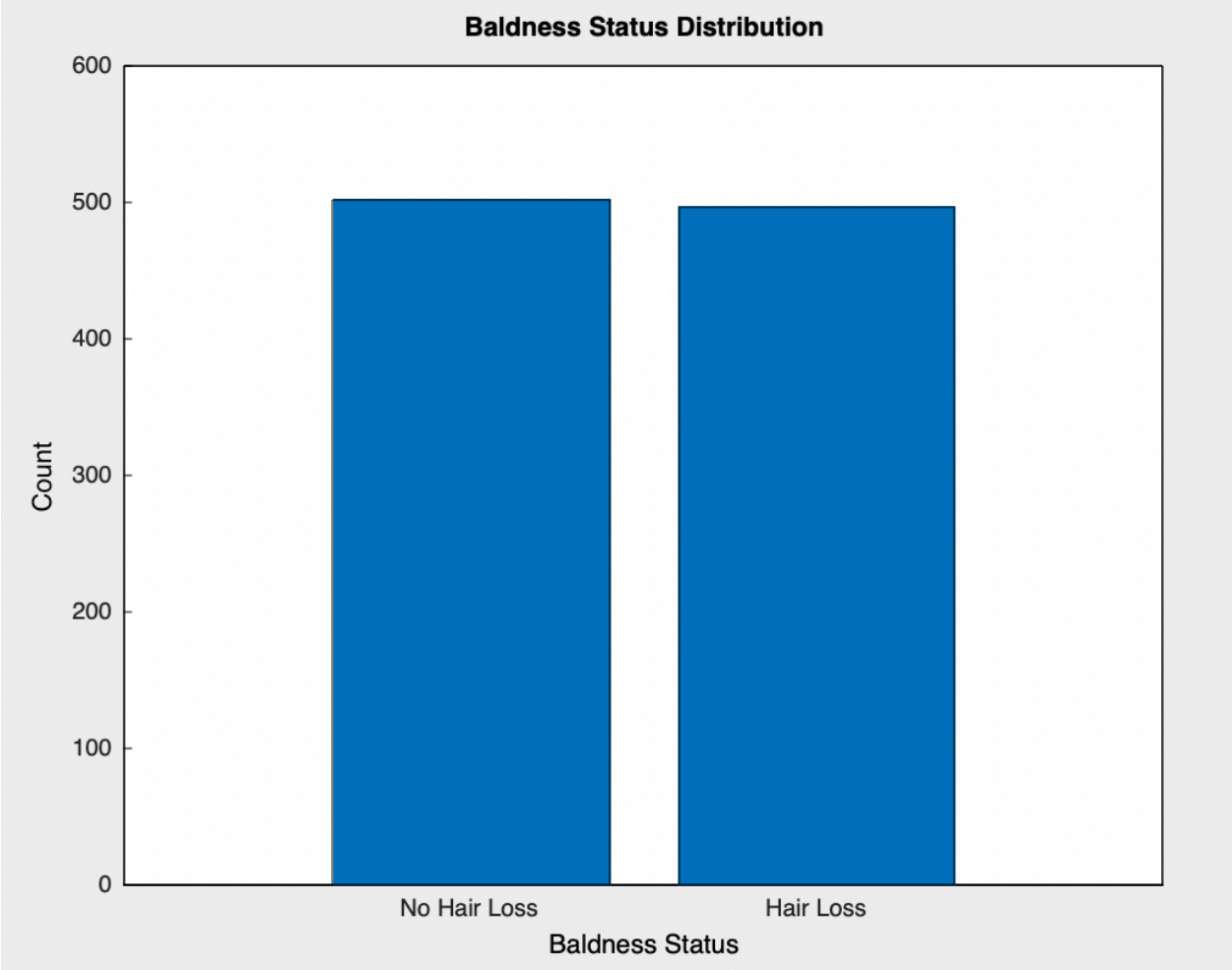
Data Analysis Techniques

The initial exploratory analysis of the dataset will involve bar plots/stacked bar plots, along with histograms, so that the distribution of the data collected can be visualized. I will then apply a linear support vector machine classifier to model the relationship between different features and hair loss. This model will predict the likelihood of hair loss based on these factors. The primary goal of these analyses is to uncover patterns or trends that could provide more clarity on how lifestyle and genetic factors contribute to hair loss.

Plots and Results

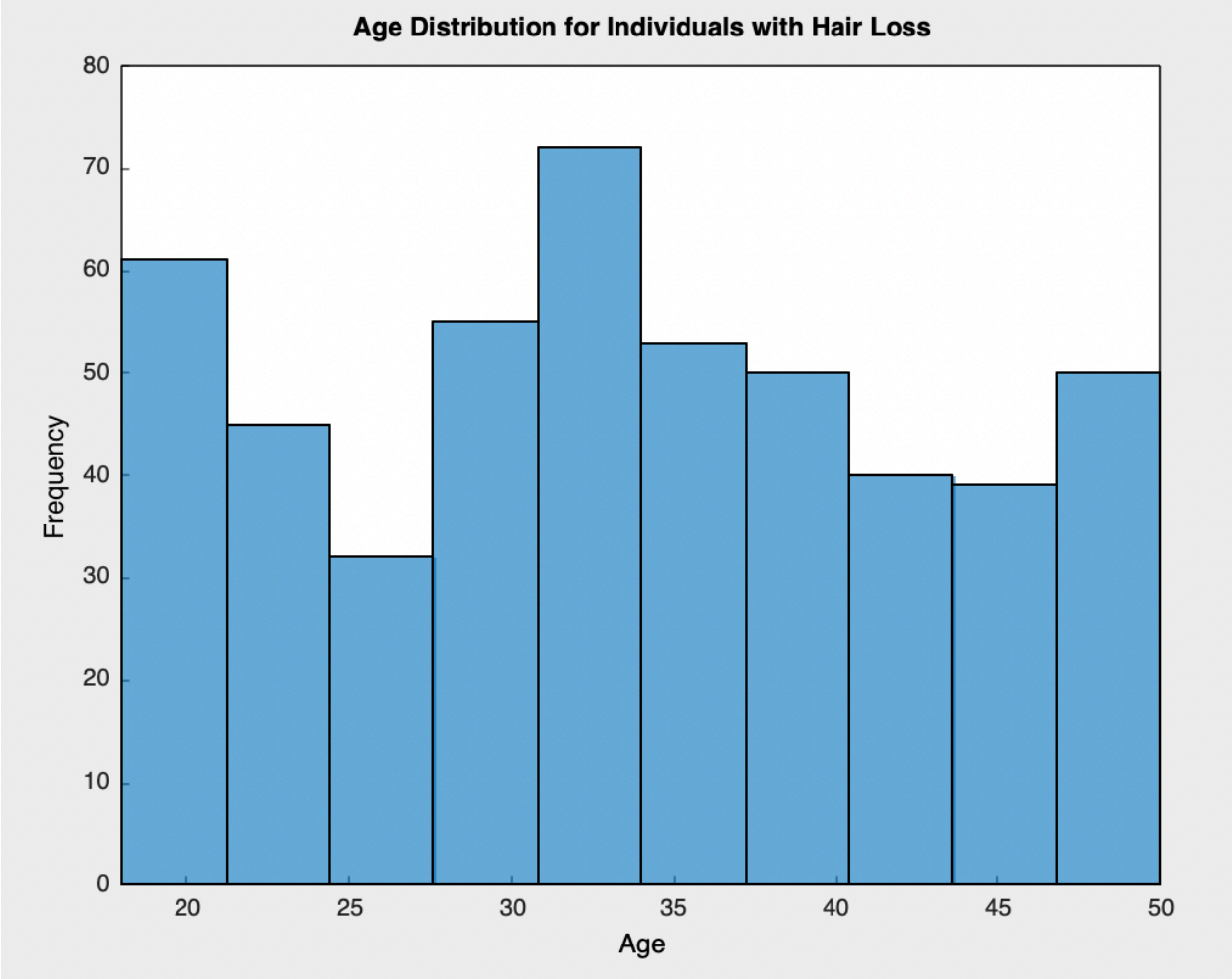
The first plot created shows the distribution of hair health status (hair loss or no hair loss). From the plot, we can see the proportion of individuals in the dataset who have experienced hair loss. The dataset is split nearly evenly between the two groups, with 502 individuals without hair loss, and 497 individuals without. This makes this dataset a good candidate to train a classifier model,

as the distribution of hair loss status is even, and won't induce bias into the algorithm.



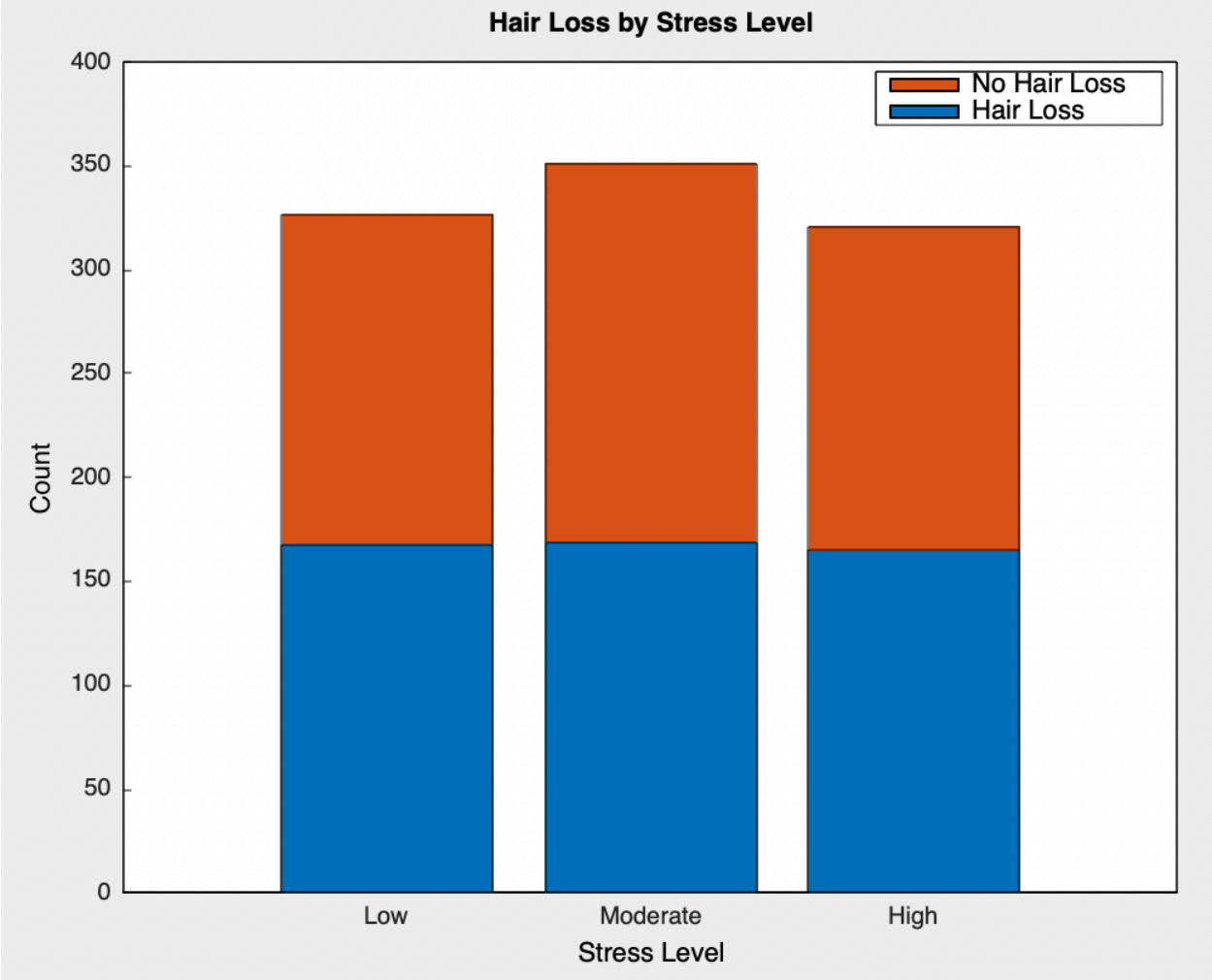
In the next plot, I aimed to examine the age distribution of the dataset, as hair loss is typically positively correlated with age. This histogram displays the age distribution for individuals with hair loss. It helps identify whether age is a significant factor in the likelihood of experiencing hair loss. This plot indicates that this dataset is widely spread, with individuals ranging from 18 years to 50 years all experiencing hair loss. Because the distribution of ages was so spread out, it

was hard to say whether age would be a useful predictor for hair loss in this dataset.



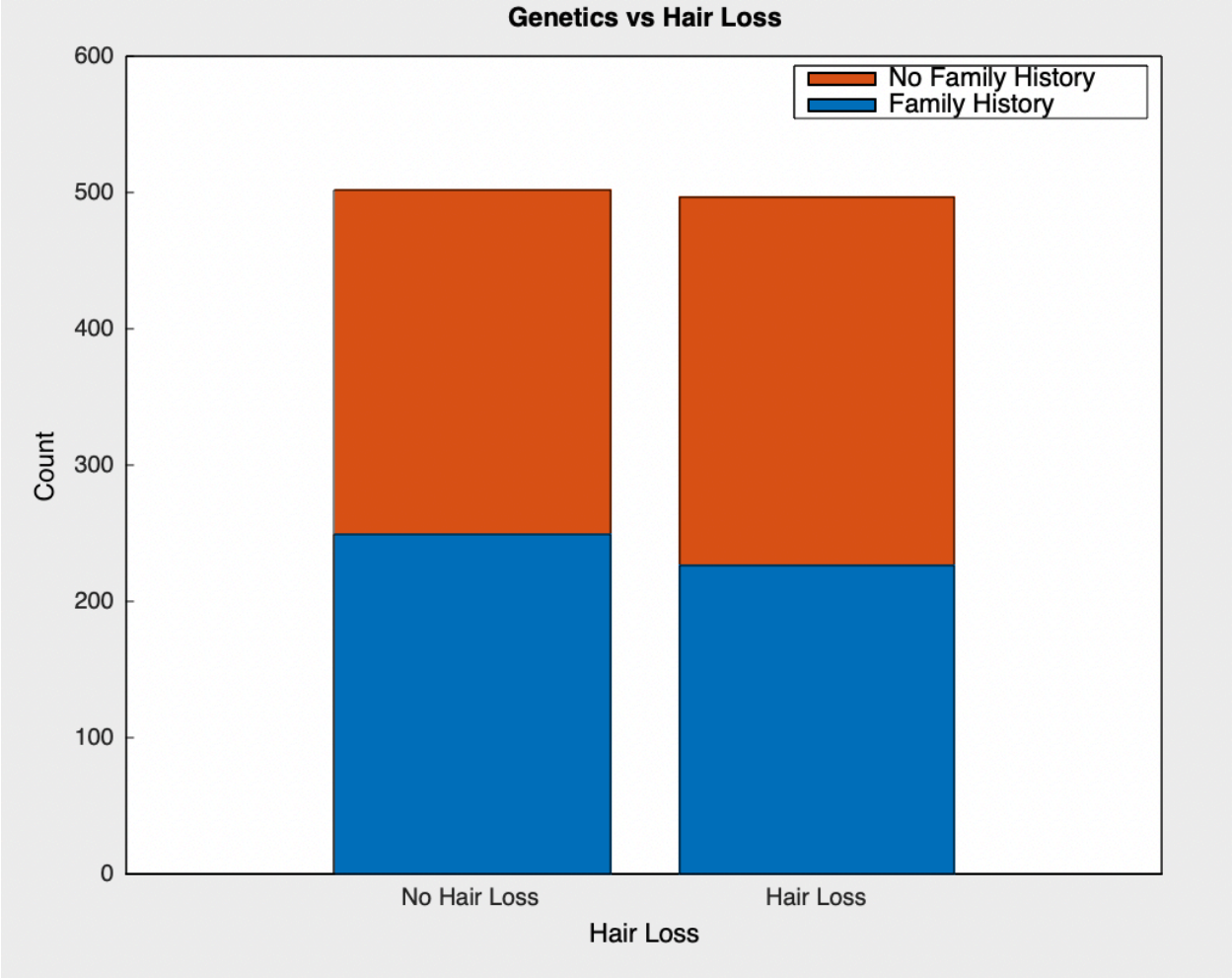
The next plots made were stacked bar plots, visualizing how different factors correlate with the occurrence of hair loss. This first plot visualizes how different levels of stress (Low, Moderate, High) affect the occurrence of hair loss. The stacked bars show the number of people with and without hair loss at each stress level, providing a visual representation of the relationship between stress and hair loss. It appears that individuals with and without hair loss have relatively similar stress levels - although high stress is known to correlate with hair loss, this dataset sees

no difference in hair loss status for levels of stress.



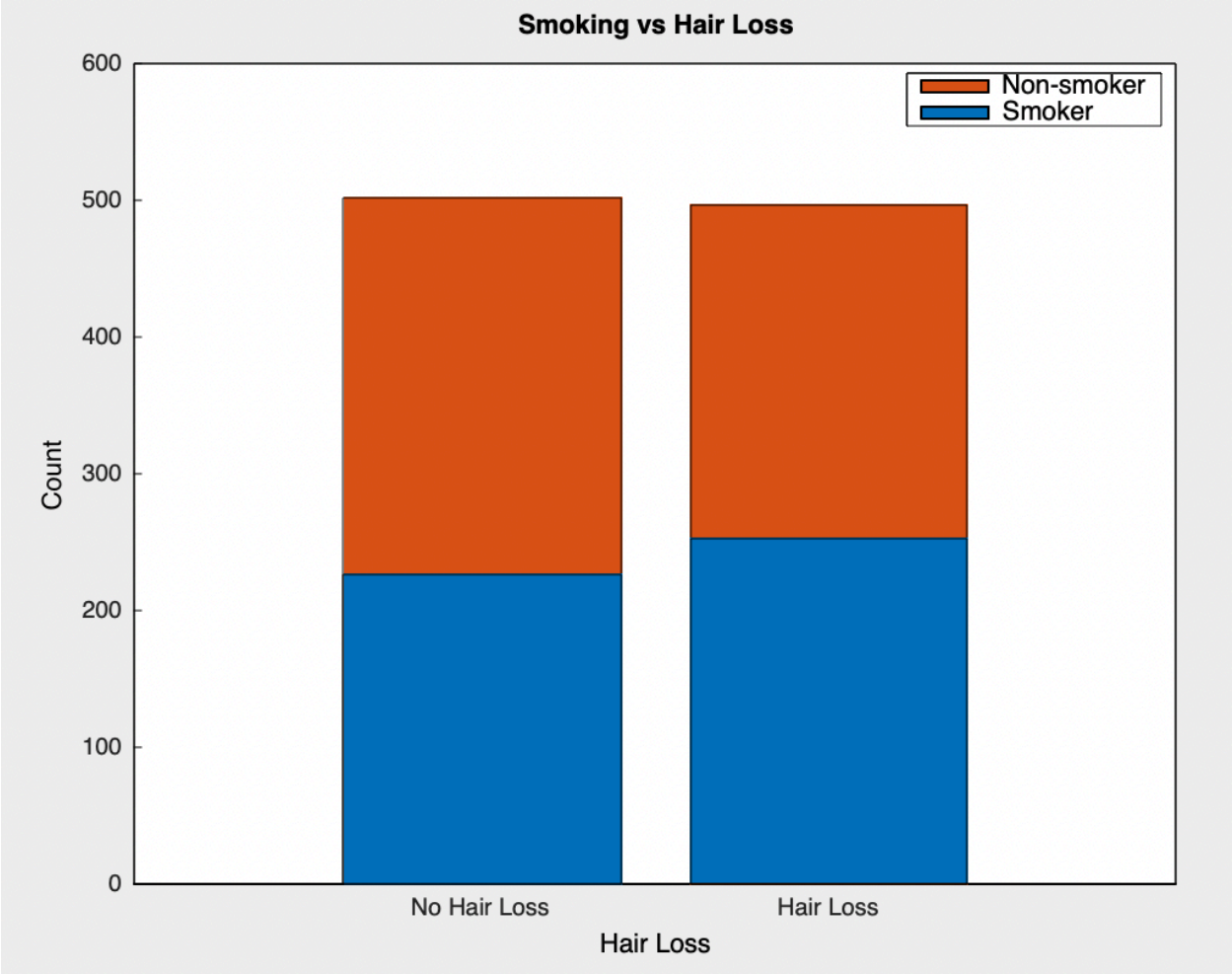
The next plot examines the relationship between genetics and hair loss, specifically whether or not an individual has a family history of hair loss. It can be seen that individuals with a family history of hair loss were nearly evenly split between having hair loss versus not, indicating that

there was no significant relationship between family history and hair loss in this data.



This next plot examines the relationship between smoking and hair loss. Smoking is known to cause damage to hair and is a risk factor for hair loss. This plot indicates that individuals who are smokers are slightly more likely to experience hair loss, but the difference in magnitude is

relatively small.



The initial data exploration revealed a limited amount of information about whether different individual factors would affect hair loss, as many groups were split evenly on hair loss status for each risk factor. In order to better understand whether certain factors are predictive of hair loss, I decided to create an SVM classifier, a supervised machine learning algorithm, that would help test my hypothesis that hair loss can be predicted based on different individual health features.

The SVM classifier trained on the dataset predicts hair loss based on features family history, hormonal changes, environmental factors, smoking, hair care, and weight loss. These factors were chosen as they were binary variables (Yes/No). The model's decision boundary is analyzed to assess how accurately it predicts hair loss, and predictions are made on a separate test set. A confusion matrix of true positives, true negatives, false positives, and false negatives, was also computed, along with an accuracy score. The SVM classifier outputted the following for accuracy status: 38 true positives, 6 true negatives, 35 false positives, and 1 false negative, leading to an overall accuracy of 0.55.

Conclusion

In this analysis, I explored various factors that may be linked to hair loss, including stress, smoking, family history, and age, using data from the Hair Health Prediction Dataset. Through visualizations and the application of a linear support vector machine (SVM) classifier, I was able to identify potential relationships between these factors and the occurrence of hair loss.

The exploratory data analysis revealed some key insights. Although stress and family history did not show a strong correlation with hair loss in this dataset, smoking appeared to have a modest influence, with smokers being slightly more likely to experience hair loss. However, the age distribution was broad, making it difficult to conclude whether age was a reliable predictor for hair loss.

The SVM yielded low predictive power of hair status, with an accuracy of only 55%. The high number of false positives indicates that this SVM is over-predicting instances of hair loss, suggesting that it is biased in some way, as the number of false negatives (1) was significantly lower. This could be due to many different factors, but due to my initial data exploration, I believe that the SVM's poor performance is due to the fact that the features selected for training the model might not provide enough discriminative power to differentiate between the two classes. Irrelevant or noisy features could mislead the SVM. This could be due to the complexity of hair loss, which may involve other factors not captured in the dataset.

It was also difficult to incorporate other factors into the SVM that may hold more predictive power. Some other features in the dataset that may have been useful were medical conditions, nutritional deficiencies, and current medications. I chose not to include these as they were either difficult to quantify or required more advanced preprocessing to make them suitable for the SVM model, as many were not inherently numeric.

From this analysis, I could not confirm my hypothesis that it is possible to build a classifier using the features selected from the Hair Health Preidction dataset. However, there are numerous possible future directions for this project. Moving forward, improving the accuracy of the classifier may involve incorporating additional features or applying more sophisticated machine learning models. Furthermore, a deeper exploration of how non-binary variables, such as nutritional deficiencies or environmental factors, influence hair loss could provide more context and enhance the model's predictive power. Overall, this analysis offers a foundation for understanding the multifaceted nature of hair loss and highlights areas where further research and data collection are needed.

Code Explanation

The exploratory data analysis part of this project is housed in projectpt1.cpp. This C++ program processes a CSV file containing data about individuals' hair health (e.g., hair loss status, age, stress levels, genetics, and smoking habits). It then generates various plots using the

matplotplusplus library, such as bar plots, histograms, and stacked bar plots, to visualize the relationships between these factors and hair loss.

The program includes the following key steps:

1. Reading CSV Data: The readCSV function reads the data from a CSV file and stores it in a 2D vector (vector<vector<string>>).
2. Counting Occurrences: The countOccurrences function counts occurrences of values in a specific column of the CSV data.
3. Generating Plots: The program generates a bar plot for baldness status distribution, a histogram for the age distribution of individuals with hair loss, and stacked bar plots for the relationships between stress levels, genetics, and smoking with hair loss.

The second part of this project involves building an SVM classifier.

This C++ program is designed to predict the occurrence of hair loss based on several health-related factors, using a machine learning model—specifically, a Support Vector Machine (SVM). The model is trained on data read from a CSV file, which contains various health indicators such as genetics, stress, smoking habits, and others. The prediction task is framed as a binary classification problem, where the goal is to determine if a person experiences hair loss (yes/no).

The program contains the following elements:

1. A helper function is defined to convert categorical values such as "Yes" and "No" into binary numeric values (1 for "Yes" and 0 for "No"). This function is crucial for transforming categorical data into a format suitable for machine learning models.
2. In the main function, the program handles the logic for reading the data, processing it, training the SVM model, and making predictions. This function ties together all the steps required to perform the machine learning task.
3. The program opens the CSV file (hair_health.csv) and skips the first line, which is the header. It then reads the remaining lines, each of which represents one individual's health data.
4. For each line in the CSV file, the program extracts various features such as genetics, stress levels, smoking habits, and more. These features are stored in string variables. The program then converts them into numerical values, preparing them for use in the machine learning model.
5. The categorical features (e.g., genetics, hormonal changes, smoking) are converted into binary values using the helper function. These converted values are stored in the sample_type matrix, which represents the feature vector for each individual.

6. The program reads the hair loss data, which indicates whether an individual experienced hair loss. This data is converted into a binary label (1 for hair loss, 0 for no hair loss) and added to the labels vector.
7. The program sets up the SVM model by selecting a linear kernel and using the Dlib library's svm_c_trainer to train the model. The model is trained using the feature vectors (from samples) and the corresponding labels (from labels).
8. After training the model, the program makes predictions on the input data. It compares the predicted labels with the actual labels to build a confusion matrix. The matrix tracks the number of true positives, true negatives, false positives, and false negatives, which are used to evaluate the performance of the model. It also calculates the accuracy of the model, which is the proportion of correct predictions (both true positives and true negatives) relative to the total number of predictions.

Data Structures

Part 1:

vector<vector<string>> data: To store the CSV data. Each row in the CSV is a vector<string>, and the entire CSV is represented as a 2D vector where each entry corresponds to one row.

map<string, int> counts: To store and count occurrences of specific values in a column. For example, the countOccurrences function counts how many times a specific baldness status appears in the dataset.

map<string, vector<int>> hairLossByStress/geneticsByHairLoss/smokingByHairLoss: To group values by certain attributes (e.g., stress level, genetics, smoking) and count occurrences based on conditions like hair loss.

Part 2:

typedef dlib::matrix<double, 7, 1> sample_type: This is an alias for a Dlib matrix, which holds the feature vector for each sample. Each sample contains 7 features, representing various health factors such as genetics, stress, and smoking.

std::vector<sample_type> samples: A vector of feature vectors (matrices) where each vector corresponds to an individual's health data.

std::vector<double> labels: A vector that stores the binary labels indicating whether an individual experienced hair loss (1 for "Yes", 0 for "No").

Instructions for Data Download and Code Compilation

Part 1:

Download the Hair Health Prediction dataset from Kaggle, and save it as a csv in the same directory as projectpt1.cpp. Ensure the matplotplusplus library is installed. Compile with the line clang++ projectpt1.cpp -o projectpt1 -std=c++17 -I/opt/homebrew/include/ -L /opt/homebrew/lib -l matplot, and run the program with ./projectpt1

Part 2:

Similarly, download the dataset, and ensure that the Dlib library is installed. Compile with the line clang++ projectpt2.cpp -std=c++17 -I/opt/homebrew/include/ -L/opt/homebrew/lib -l dlib -lblas -o projectpt2, and run the program with ./projectpt2

Code

projectpt1.cpp

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <string>
#include <map>
#include <matplot/matplot.h>

using namespace std;
using namespace matplot;

// Function to read CSV data
vector<vector<string>> readCSV(const string& filename) {
    vector<vector<string>> data;
    ifstream file(filename);
    string line;

    while (getline(file, line)) {
        stringstream ss(line);
        string cell;
        vector<string> row;
        while (getline(ss, cell, ',')) {
            row.push_back(cell);
        }
        data.push_back(row);
    }
    return data;
}

// Function to count occurrences for a column (e.g., baldness status)
map<string, int> countOccurrences(const vector<vector<string>>& data, size_t
colIndex) {
    map<string, int> counts;
    for (size_t i = 1; i < data.size(); ++i) { // Start at 1 to skip header
        if (colIndex < data[i].size()) {
            counts[data[i][colIndex]]++;
        }
    }
    return counts;
}
```

```
int main() {
    string filename = "hair_health.csv";
    auto data = readCSV(filename);

    // Plot 1: Bar Plot for Baldness Status
    size_t baldnessColumnIndex = 12;
    auto baldnessCounts = countOccurrences(data, baldnessColumnIndex);

    vector<double> x; // Numeric indices for bars
    vector<double> values;
    vector<string> labels;

    size_t index = 0;
    for (const auto& pair : baldnessCounts) {
        x.push_back(index++); // Assign a numeric index
        values.push_back(pair.second);
        labels.push_back(pair.first);
    }

    auto bars = bar(x, values);
    xlabel("Baldness Status");
    ylabel("Count");
    title("Baldness Status Distribution");

    // Set the string labels on the x-axis
    gca()->x_axis().ticklabels({"No Hair Loss", "Hair Loss"});
    show();

    // clearing plot
    cla();
    auto ax = gca();
    ax->x_axis().ticklabels({});

    // Plot 2: Histogram for Age and Hair Loss
    vector<double> ageWithHairLoss;
    size_t ageIndex = 7, hairLossIndex = 12;

    for (size_t i = 1; i < data.size(); ++i) {
        if (ageIndex < data[i].size() && hairLossIndex < data[i].size()) {
            double age = stod(data[i][ageIndex]);
            int hairLoss = stoi(data[i][hairLossIndex]);
            if (hairLoss == 1) {
                ageWithHairLoss.push_back(age);
            }
        }
    }

    auto h1 = hist(ageWithHairLoss, 10);  // Create the histogram for age with hair
loss
    gca()->xlim({18, 50});  // Set x-axis range to match the age data range
    gca()->xticks({20, 25, 30, 35, 40, 45, 50});
    xlabel("Age");
    ylabel("Frequency");
    title("Age Distribution for Individuals with Hair Loss");
    show();

    // clear plot
    auto ax1 = gca();
    ax1->x_axis().ticklabels({});
    cla();
```

```
    // Plot 3: Stacked Bar Plot: Hair Loss by Stress Level
    size_t stressIndex = 6;
    map<string, vector<int>> hairLossByStress;

    // Populate hairLossByStress map
    for (size_t i = 1; i < data.size(); ++i) {
        if (stressIndex < data[i].size() && data[i].size() > 12) {
            string stressLevel = data[i][stressIndex];
            int hairLoss = stoi(data[i][12]); // Column index for "Hair Loss"
            hairLossByStress[stressLevel].push_back(hairLoss);
        }
    }

    // Prepare data for stacked bar plot
    vector<string> stressLevels = {"Low", "Moderate", "High"};
    vector<double> noHairLoss, hairLoss;

    for (const auto& level : stressLevels) {
        auto& hairLossValues = hairLossByStress[level];
        int countNo = count(hairLossValues.begin(), hairLossValues.end(), 0);
        int countYes = count(hairLossValues.begin(), hairLossValues.end(), 1);
        noHairLoss.push_back(countNo);
        hairLoss.push_back(countYes);
    }

    // Create a 2D vector where each row corresponds to a data series
    vector<vector<double>> Y = {noHairLoss, hairLoss};

    // Generate stacked bar plot
    auto h = barstacked(Y);
    gca()->x_axis().ticklabels(stressLevels); // Set x-axis labels to stress levels
    xlabel("Stress Level");
    ylabel("Count");
    title("Hair Loss by Stress Level");
    matplot::legend({"No Hair Loss", "Hair Loss"});
    show();

    // Clearing plot
    auto ax2 = gca();
    ax2->x_axis().ticklabels({});
    cla();

    // Plot 4: Stacked Bar Plot - Genetics vs Hair Loss
    size_t geneticsIndex = 1;

    // Map to store Genetics counts grouped by Hair Loss
    map<string, vector<int>> geneticsByHairLoss;

    // Populate the geneticsByHairLoss map
    for (size_t i = 1; i < data.size(); ++i) {
        if (hairLossIndex < data[i].size() && geneticsIndex < data[i].size()) {
            string hairLoss = data[i][hairLossIndex];
            string genetics = data[i][geneticsIndex];

            // Initialize if not already in the map
            if (geneticsByHairLoss.find(hairLoss) == geneticsByHairLoss.end()) {
                geneticsByHairLoss[hairLoss] = {0, 0}; // {No Genetics, Yes
Genetics}
            }

            // Increment counts based on Genetics (Yes/No)
```

```
            if (genetics == "Yes") {
                geneticsByHairLoss[hairLoss][1]++;
            } else if (genetics == "No") {
                geneticsByHairLoss[hairLoss][0]++;
            }
        }
    }

    // Prepare data for the stacked bar plot
    // Automatically determine Hair Loss categories from the map keys
    vector<string> hairLossCategories;
    for (const auto& [hairLoss, counts] : geneticsByHairLoss) {
        hairLossCategories.push_back(hairLoss);
    }

    // Prepare data for the stacked bar plot
    vector<double> noGenetics, yesGenetics;

    for (const auto& category : hairLossCategories) {
        // Extract counts for each category
        noGenetics.push_back(geneticsByHairLoss[category][0]);
        yesGenetics.push_back(geneticsByHairLoss[category][1]);
    }

    // Create a 2D vector for stacked bar plot
    vector<vector<double>> yVar = {noGenetics, yesGenetics};

    // Generate the stacked bar plot
    auto h2 = barstacked(yVar);
    vector<string> x_axis = {"No Hair Loss", "Hair Loss"};
    gca()->x_axis().ticklabels(x_axis); // Dynamic tick labels
    xlabel("Hair Loss");
    ylabel("Count");
    title("Genetics vs Hair Loss");
    matplot::legend({"No Family History", "Family History"});
    show();

    // Clearing plot
    auto ax3 = gca();
    ax3->x_axis().ticklabels({});
    cla();

    // Plot 5: Stacked Bar Plot - Hair Loss vs Smoking
    size_t smokingIndex = 10;

    // Map to store Smoking counts grouped by Hair Loss
    map<string, vector<int>> smokingByHairLoss;

    // Populate the smokingByHairLoss map
    for (size_t i = 1; i < data.size(); ++i) {
        if (hairLossIndex < data[i].size() && smokingIndex < data[i].size()) {
            string hairLoss = data[i][hairLossIndex];
            string smoking = data[i][smokingIndex];

            // Initialize if not already in the map
            if (smokingByHairLoss.find(hairLoss) == smokingByHairLoss.end()) {
                smokingByHairLoss[hairLoss] = {0, 0};
            }

            // Increment counts based on Smoking status (Yes/No)
            if (smoking == "Yes") {
                smokingByHairLoss[hairLoss][1]++;
```

```
            } else if (smoking == "No") {
                smokingByHairLoss[hairLoss][0]++;
            }
        }
    }

    // Prepare data for the stacked bar plot
    vector<double> noSmoke, yesSmoke;

    for (const auto& category : hairLossCategories) {
        // Extract counts for each category
        noSmoke.push_back(smokingByHairLoss[category][0]);
        yesSmoke.push_back(smokingByHairLoss[category][1]);
    }

    // Create a 2D vector for stacked bar plot
    vector<vector<double>> yVariable = {noSmoke, yesSmoke};

    // Generate the stacked bar plot
    auto h3 = barstacked(yVariable);
    gca()->x_axis().ticklabels(x_axis);
    xlabel("Hair Loss");
    ylabel("Count");
    title("Smoking vs Hair Loss");
    matplot::legend({"Non-smoker", "Smoker"});
    show();


    return 0;
}
```

projectpt2.cpp

```
#include <iostream>
#include <vector>
#include <fstream>
#include <dlib/svm.h>
#include <string>
#include <sstream>

using namespace std;
using namespace dlib;

// Convert 'Yes'/'No' to 1/0
int yesNoToInt(const string& s) {
    if(s == "Yes"){
        return 1;
    }
    else{
        return 0;
    }
}

int main(int argc, char* argv[])
{
    typedef dlib::matrix<double, 7, 1> sample_type; // Dlib matrix that holds a
feature vector for each sample

    std::vector<sample_type> samples; // Vector of feature vectors
    std::vector<double> labels; // Vector of labels of whether an individual has
```

```
hair loss (0/1)

    ifstream ifile("hair_health.csv");
    string temp;

    // Read the header
    getline(ifile, temp);

    typedef linear_kernel<sample_type> kernel_type;

    size_t line_number = 0;
    while (getline(ifile, temp)) {
        string id, genetics, hormonalChanges, medicalConditions, medications,
nutritionalDeficiencies, stress, age, hairCare, environmentalFactors, smoking,
weightLoss, hairLoss;

        stringstream ss(temp);
        getline(ss, id, ',');
        getline(ss, genetics, ',');
        getline(ss, hormonalChanges, ',');
        getline(ss, medicalConditions, ',');
        getline(ss, medications, ',');
        getline(ss, nutritionalDeficiencies, ',');
        getline(ss, stress, ',');
        getline(ss, age, ',');
        getline(ss, hairCare, ',');
        getline(ss, environmentalFactors, ',');
        getline(ss, smoking, ',');
        getline(ss, weightLoss, ',');
        getline(ss, hairLoss);

        // Keeping only binary variables
        sample_type sample;
        sample(1) = yesNoToInt(genetics);
        sample(2) = yesNoToInt(hormonalChanges);
        sample(3) = yesNoToInt(environmentalFactors);
        sample(4) = yesNoToInt(smoking);
        sample(5) = yesNoToInt(hairCare);
        sample(6) = yesNoToInt(weightLoss);

        double label;
        int hairLossInt = std::stoi(hairLoss);
        if (hairLossInt == 1) {
            label = 1.0;
        }
        else {
            label = 0.0;
        }
        samples.push_back(sample);
        labels.push_back(label);
    }

    // Train SVM
    svm_c_trainer<kernel_type> trainer;
    decision_function<kernel_type> dec_funct = trainer.train(samples, labels);

    // Use the trained model to predict
    std::vector<double> predictions;
    int tp = 0, tn = 0, fp = 0, fn = 0;

    // Loop over the samples to predict and update confusion matrix
    for (size_t i = 0; i < samples.size(); ++i) {
```

```
        double prediction = dec_funct(samples[i]);
        predictions.push_back(prediction);

        // Compare predicted label to actual label to update the confusion matrix
        if (prediction == 1.0 && labels[i] == 1.0) {
            tp++;  // True Positive
        } else if (prediction == 0.0 && labels[i] == 0.0) {
            tn++;  // True Negative
        } else if (prediction == 1.0 && labels[i] == 0.0) {
            fp++;  // False Positive
        } else if (prediction == 0.0 && labels[i] == 1.0) {
            fn++;  // False Negative
        }
    }

    // Print the confusion matrix
    cout << "Confusion Matrix: " << endl;
    cout << "True Positive (TP): " << tp << endl;
    cout << "True Negative (TN): " << tn << endl;
    cout << "False Positive (FP): " << fp << endl;
    cout << "False Negative (FN): " << fn << endl;

    double accuracy = double(tp + tn) / (tp + tn + fp + fn);
    cout << "Accuracy: " << accuracy << endl;

    return 0;
}
```