Faculty of Engineering and Technology

Department of Electrical and Computer Engineering

Intelligent Systems Lab

ENCS5141

Assignment #1: Case Study 3.1

The Impact of Data Preprocessing on Machine Learning Models.

Prepared by: Maysam Khatib 1190207.

Section: 3

Instructor: Dr. Mohammad Jubran.

TA: Eng. Hanan Awawdeh.

Date: 29/11/2023.

Abstract

This case study aims to prepare a dataset which is the Penguins dataset for machine learning analysis by applying essential data preprocessing steps to it.

The primary objective is to navigate through the various stages of data preprocessing, including initial data exploration, handling data quality issues such as outliers and missing values, and determining suitable methods to handle them. Additionally, these stages involve selecting the most relevant features from the dataset through analysis based on criteria such as variance and correlation methods, encoding non-numerical features, splitting the dataset into a training set for model training and a testing set for evaluating the model's performance, scaling and normalizing numerical features to a common scale, and handling high-dimensional data by applying dimensionality reduction techniques such as Principal Component Analysis (PCA). This case study further includes training a Random Forest Classifier model to evaluate the data preprocessing effectiveness.

The report navigates through these preprocessing stages, focusing on the reasoning behind each one, and underscores the importance of appropriate data preparation for enhancing the performance of machine learning models.

Table of Contents

Abstract 2

Table of Figures 4

Table of Tables 4

Introduction 5

- Impact of Preprocessing 5
- Missing Values and Outliers 5
- Feature Relevance and Selection 6
- Encoding Features 6
- Scaling Numerical Features 6
- Dimensionality Reduction 6

Procedure and Discussion 7

1. Dataset Loading 7
2. Initial Data Exploration 8
3. Data Quality Issues 9
   - 3.1. Missing Values 9
   - 3.2. Outliers 11
4. Features Relevance and Selections 14
5. Encoding Categorical Columns 14
6. Splitting Dataset 18
7. Scaling Numerical Features 18
8. Applying Dimensionality Reduction Techniques 19
9. Training and Evaluating the Model 21

Conclusion 23

Table of Figures

Figure 1. Outliers Analysis Plots..... 12
Figure 2. Original and Scaled Features Distribution 19

Table of Tables

Table 1. Rows from Penguins dataset..... 8
Table 2. Data frame Basic Statistics..... 9
Table 3. Data Frame after Encoding Categorical Columns..... 15
Table 4. New Data Frame with only the numerical columns 15
Table 5. Features Variances..... 16
Table 6. Data Frame after Removing 'sex_encoded' Feature..... 18
Table 7. PCA X_train Data Frame 20
Table 8. PCA X_test Data Frame..... 20

Introduction

In the field of machine learning, the key to achieving effective model performance lies in the quality and preparation of the input dataset. This case study focuses on the primary stages of data preprocessing applied to the **Penguins** dataset, a collection that includes information about various penguin species, their physical characteristics, and the regions where they inhabit.

The preprocessing of the Penguins dataset is imperative to extract meaningful insights and relationships, aiming for optimal performance in the ensuing machine learning model. Throughout this case study, we navigate through the data preprocessing stages, addressing challenges such as missing values, outliers, categorical features, and high dimensionality, which contribute to the complexity of the dataset.

• Impact of Preprocessing

One of the keys factors in the effectiveness of machine learning models is data preprocessing. It is not just data cleaning; rather, it is the first step toward improving generalization and making accurate predictions. By simplifying and structuring the raw data, we pave the way for a more effective learning process.

The impact of preprocessing is as follows:

1- Enhanced Model Accuracy: Preprocessing improves the dataset's quality, which improves generalization and produces predictions that are more accurate. Data that is clear and organized is essential to the performance of machine learning models.

2- Appropriate Use of Features: Preprocessing ensures the model uses the most pertinent data from the dataset by using methods like feature relevance analysis and encoding. Better ability to make decisions and an efficient learning process are the effects of this.

3- Improved Robustness of Model: Preprocessing helps to create a reliable and robust dataset by resolving issues like outliers and inconsistent data. Thus, the model is strengthened against any dangers and guaranteed to operate consistently and effectively.

• Missing Values and Outliers

The existence of missing values and outliers in a dataset impacts the performance of machine learning model. Missing values can produce biased or inaccurate results, while outliers can have a significant impact on the model's performance. Either removal or imputation techniques are used to deal with missing values; in these cases, the missing entries are estimated using the information that already exists. These imputation techniques are mean imputation, median imputation, or more complex techniques like regression imputation. Options for handling outliers include transformation, removal, or applying strong statistical techniques like the interquartile range (IQR).

- ## Feature Relevance and Selection

Building effective models requires carefully going through a dataset to find and select the most relevant features. The importance of each individual feature in enhancing the prediction ability of the model is referred to as feature relevance. Making the most relevant feature selections is essential for improving model accuracy, simplifying model interpretation, and reducing overfitting risk.

To find and choose the most informative features, a variety of strategies are used, including variance analysis, correlation analyses, and complex methods like Recursive Feature Elimination.

- ## Encoding Features

Encoding non-numerical properties of the dataset is an essential step in getting it ready for machine learning algorithms. It is necessary to transform categorical variables—like species, sex, and island in Penguins dataset—into a numerical representation that models can understand. Typical encoding methods include one-hot encoding, which generates binary columns for every category, and label encoding, which assigns a unique integer to each category. The machine learning algorithm's requirements and the type of categorical data determine which approach is best.

- ## Scaling Numerical Features

Scaling numerical features is important to ensure consistent scaling across variables. This step becomes particularly important when numerical features have different ranges. Scaling techniques, such as Min-Max scaling or standardization, transform numerical features to a common scale, preventing certain features from dominating others during model training. By bringing all features to a comparable range, scaling contributes to improved model performance and stability.

- ## Dimensionality Reduction

A lot of real-world datasets have a high dimensionality because of the number of features. To solve this problem, dimensionality reduction approaches minimize the number of features while maintaining the necessary data. A popular technique that converts the original characteristics into a new set of uncorrelated variables (principal components) is called principal component analysis (PCA). PCA helps to overcome the difficulties posed by high-dimensional data by allowing for a more simplified dataset by keeping the most important components.

Procedure and Discussion

In this case study, you will perform essential data preprocessing steps on the Penguins dataset. The dataset contains information about different species of penguins, including their physical characteristics and the region where they were observed. Your goal is to prepare the dataset for machine learning analysis. Follow these steps:

1. Load the penguins dataset using the code snippet provided below.

2. Perform initial data exploration to understand the dataset's structure, features, and any missing values. Summarize the dataset's statistics and gain insights into the data.

3. Address any data quality issues, such as missing values and outliers. Decide on an appropriate strategy for handling missing data, such as imputation or removal of rows/columns.

4. Analyze the relevance of each feature for your machine learning task by using the learned use feature selection techniques.

5. If the dataset contains categorical variables, encode them into a numerical format suitable for machine learning models.

6. Split the dataset into training and testing subsets to evaluate the performance of your machine learning models.

7. Scale or normalize the numerical features to ensure consistent scaling across variables.

8. Apply suitable dimensionality reduction techniques to reduce the size of the data while preserving important information.

9. Validate your preprocessing pipeline by training and evaluating a machine learning model, such as the Random Forest model, on the preprocessed data. Compare the results to the model trained on the raw data (before feature filtering, transformation, and reduction) to ensure that preprocessing has improved model performance.

1. Dataset Loading

The dataset was loaded successfully from seaborn library as shown in the code below:

```
# Import the 'load_dataset' function from seaborn to load the penguins dataset
from seaborn import load_dataset

# Load the penguins dataset and store it in the 'df' DataFrame
df = load_dataset('penguins')

# Display the first few rows of the DataFrame to get an initial look at the data
df.head()
```

The first few rows from the data frame was shown using df.head() as shown in table 1 below.

Table 1. Rows from Penguins dataset

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	Male
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	Female
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	Female
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	Female

2. Initial Data Exploration

First, the information about the data was displayed as shown in the code and output below. The dataset consists of 7 columns, and 344 entries (rows).

As noticed, there are missing values in a few columns. In particular, the counts of non-null values for 'bill_length_mm,' 'bill_depth_mm,' 'flipper_length_mm,' 'body_mass_g,' and 'sex' are lower than the number of entries (344). The null values will be handled in the next subsection.

```
# Display summary of information about the DataFrame
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 344 entries, 0 to 343
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   species            344 non-null    object 
 1   island             344 non-null    object 
 2   bill_length_mm     342 non-null    float64
 3   bill_depth_mm      342 non-null    float64
 4   flipper_length_mm  342 non-null    float64
 5   body_mass_g        342 non-null    float64
 6   sex                333 non-null    object 
dtypes: float64(4), object(3)
      memory usage: 18.9+ KB
```

Then, the statistics of the numerical columns in the data frame were displayed as we seen in table 2. The basic statistics were measured such as the number of non-null values (count), the mean value in the column, the standard deviation which indicates the spread and variability in the data, and other basic statistics.

```
# Display the dataset's statistics
df.describe()
```

Table 2. Data frame Basic Statistics

	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
count	342.000000	342.000000	342.000000	342.000000
mean	43.921930	17.151170	200.915205	4201.754386
std	5.459584	1.974793	14.061714	801.954536
min	32.100000	13.100000	172.000000	2700.000000
25%	39.225000	15.600000	190.000000	3550.000000
50%	44.450000	17.300000	197.000000	4050.000000
75%	48.500000	18.700000	213.000000	4750.000000
max	59.600000	21.500000	231.000000	6300.000000

3. Data Quality Issues

3.1.Missing Values

First, the rows containing missing values, and the number of them were displayed as we can see below. Out of the 344 entries, 11 rows contain missing values, constituting approximately 3.2% of the dataset. The affected columns include 'bill_length_mm' 'bill_depth_mm', 'flipper_length_mm', 'body_mass_g', and 'sex'.

```
# Display the rows contained missing values
print("The rows contained missing values:\n")
print(df[df.isnull().any(axis=1)])
print("-----------------------------------------")
# Display the number of missing values
print(f"The number of missing values in each
column:\n{df.isnull().sum()}")
print("-----------------------------------------")
total_missing_values = df.isnull().any(axis=1).sum()
percantage = total_missing_values*100/df.shape[0]
print(f"The total number of rows with missing values =
{total_missing_values}, with a percantage = {percantage}%")
```

```
The rows contained missing values:

    species     island  bill_length_mm  bill_depth_mm  flipper_length_mm
\
3    Adelie  Torgersen             NaN            NaN                NaN
8    Adelie  Torgersen            34.1           18.1              193.0
9    Adelie  Torgersen            42.0           20.2              190.0
10   Adelie  Torgersen            37.8           17.1              186.0
11   Adelie  Torgersen            37.8           17.3              180.0
47   Adelie      Dream            37.5           18.9              179.0
246  Gentoo     Biscoe            44.5           14.3              216.0
286  Gentoo     Biscoe            46.2           14.4              214.0
324  Gentoo     Biscoe            47.3           13.8              216.0
336  Gentoo     Biscoe            44.5           15.7              217.0
339  Gentoo     Biscoe             NaN            NaN                NaN

     body_mass_g  sex
3            NaN  NaN
8         3475.0  NaN
9         4250.0  NaN
10        3300.0  NaN
11        3700.0  NaN
47        2975.0  NaN
246       4100.0  NaN
286       4650.0  NaN
324       4725.0  NaN
336       4875.0  NaN
339          NaN  NaN
----------------------------------------
The number of missing values in each column:
species               0
island                0
bill_length_mm        2
bill_depth_mm         2
flipper_length_mm     2
body_mass_g           2
sex                  11
dtype: int64
----------------------------------------
The total number of rows with missing values = 11, with a percantage =
3.197674418604651%
```

There are many ways to handle these missing values, either drop these rows or replace them (Imputation Strategy). The imputation technique was used because I want to retain as much data as possible and sometime removing rows with missing values would lead to a reduction in the dataset size, potentially resulting in loss of valuable information.

As shown in the code below, missing values have been imputed using different strategies for numerical and categorical columns:

- For numerical columns ("bill_length_mm", "bill_depth_mm", "flipper_length_mm", "body_mass_g"), missing values are replaced with the mean value of each respective column.
- For categorical columns ("species", "island", "sex"), missing values are replaced with the mode value, which represents the most frequent value in each respective column.

As shown in the output below, there are no missing values after imputation.

```
# Replace null values with the mean value of the nemarical columns

numarical_columns = ["bill_length_mm", "bill_depth_mm",
"flipper_length_mm", "body_mass_g"]
for col in numarical_columns:
  df[col].fillna(df[col].mean(), inplace=True)

# Replace null values with the mode value (the most frequented value) of
the categorical columns
categorical_columns = ["species", "island", "sex"]
for col in categorical_columns:
  df[col].fillna(df[col].mode().iloc[0], inplace=True)

# We can instead delete the rows with null values using this:
# df.dropna(inplace=True)

print(f"The number of missing values after
imputation:\n{df.isnull().sum()}")
```

```
The number of missing values after imputation:

species              0
island               0
bill_length_mm       0
bill_depth_mm        0
flipper_length_mm    0
body_mass_g          0
sex                  0
dtype: int64
```

3.2.Outliers

To see if there are any outliers, a comprehensive analysis of the columns was conducted. The analysis included visualizations through box plots for numerical columns and count plots for categorical columns giving information on the distribution and existence of possible outliers.

As shown in figure 1, there are no outliers on the numerical columns based on Interquartile Range (IQR) method which is used in boxplot.

```
import seaborn as sns
import matplotlib.pyplot as plt

# plot each column to see if there are any outliers
plt.figure(figsize=(15, 8))

for i, col in enumerate(numarical_columns, start=1):
  plt.subplot(2, 4, i)
  plt.title(col)
  sns.boxplot(data=df, x=col, color='g' if i%2==1 else 'r')

for i, col in enumerate(categorical_columns, start=5):
  plt.subplot(2, 4, i)
  plt.title(col)
  sns.countplot(data=df, x=col)

plt.tight_layout()
plt.show()
```

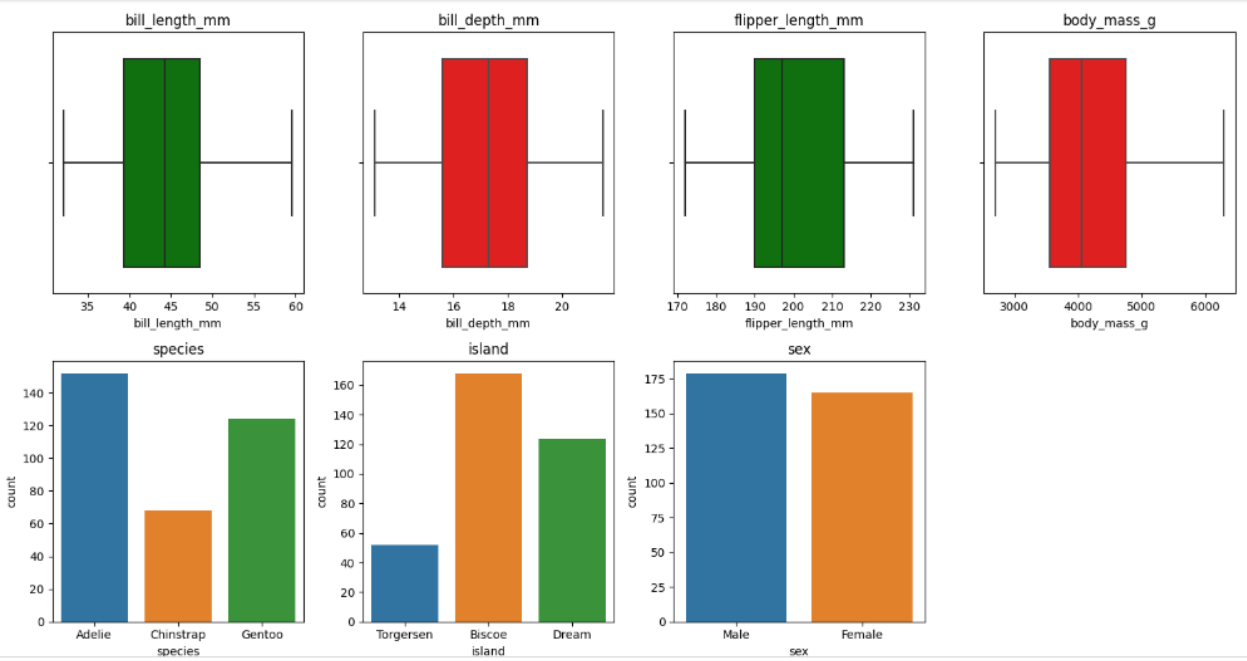

Figure 1. Outliers Analysis Plots

Also, the number of outliers was calculated using two methods: Interquartile Range (IQR) and z-score.

For each numerical column, the IQR method was implemented by calculating the 25th and 75th percentiles. Outliers were identified as values falling below 1.5 times the IQR below the 25th percentile or above 1.5 times the IQR above the 75th percentile.

Simultaneously, the z-score method was applied, wherein values more or less than 2.5 standard deviations from the mean were considered outliers.

```
# find the number of outliers based on Interquartile Range IQR and z-score
for col in numarical_columns:
  # calculating outliers number based on IQR
  percentile75 = df[col].quantile(0.75)
  percentile25 = df[col].quantile(0.25)
  IQR = percentile75 - percentile25
  upper_iqr = percentile75 + 1.5*IQR
  lower_iqr = percentile25 - 1.5*IQR
  n_oultiers_iqr = df[(df[col] < lower_iqr) | (df[col] >
upper_iqr)].shape[0]

  # calculating outliers number based on z-score
  mean = df[col].mean()
  std = df[col].std()
  upper_z = mean + 2.5*std
  lower_z = mean - 2.5*std
  n_oultiers_z = df[(df[col] < lower_z) | (df[col] > upper_z)].shape[0]
  print(f'for column: {col}')
  print(f'# of outliars based on IQR = {n_oultiers_iqr}')
  if(n_oultiers_iqr > 0):
    print("outliers values:")
    print(df.loc[(df[col] < lower_iqr) | (df[col] > upper_iqr),
[col]].values)
  print(f'# of outliars based on z-score = {n_oultiers_z}')
  if(n_oultiers_z > 0):
    print("outliers values:")
    print(df.loc[(df[col] < lower_z) | (df[col] > upper_z), [col]].values)
  print("")
```

- For IQR method: no outliers were detected in any of the numerical columns.
- For Z-score method: 2 outliers were observed with values 58.0 and 59.6 in ‘bill_length_mm’ column, and 1 outlier was observed with value 6300.0 in ‘body_mass_g’ column.

I decided not to remove the outliers because of their limited number so it will not have a big impact on the other data. Also, they were only detected by z-score not both methods.

```
for column: bill_length_mm
# of outliars based on IQR = 0
# of outliars based on z-score = 2
outliers values:
[[58. ]
 [59.6]]

for column: bill_depth_mm
# of outliars based on IQR = 0
# of outliars based on z-score = 0

for column: flipper_length_mm
# of outliars based on IQR = 0
# of outliars based on z-score = 0

for column: body_mass_g
# of outliars based on IQR = 0
# of outliars based on z-score = 1
outliers values:
[[6300.]]
```

4. Features Relevance and Selections

5. Encoding Categorical Columns

First, the categorical columns in the dataset were encoded so we can analyze its relevance. Also, encoding is an essential process for preparing the dataset for machine learning models. This transformation ensures that machine learning algorithms can effectively interpret and utilize these variables.

The encoding was done by the LabelEncoder from the scikit-learn library to transform categorical columns which are species, island, and sex into their corresponding numerical representations. The newly encoded columns were then appended to the dataset.

```
# the dataset contains categorical variables, so we have to encode them
first
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()

for col in categorical_columns:
  df[f'{col}_encoded'] = label_encoder.fit_transform(df[col])
  numarical_columns.append(f'{col}_encoded')

df.head()
```

Table 3. Data Frame after Encoding Categorical Columns

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex	species_encoded	island_encoded	sex_encoded
0	Adelie	Torgersen	39.10000	18.70000	181.000000	3750.000000	Male	0	2	1
1	Adelie	Torgersen	39.50000	17.40000	186.000000	3800.000000	Female	0	2	0
2	Adelie	Torgersen	40.30000	18.00000	195.000000	3250.000000	Female	0	2	0
3	Adelie	Torgersen	43.92193	17.15117	200.915205	4201.754386	Male	0	2	1
4	Adelie	Torgersen	36.70000	19.30000	193.000000	3450.000000	Female	0	2	0

As shown above in table 3, the categorical columns were encoded, and each category was represented by a specific number. For example, in sex column, Male category is represented by 0, while Female represented by 1.

Then, a new data frame with only the numerical and the encoded columns was created as shown in table 4 below.

```
df_encoded = df[numarical_columns]

df_encoded.head()
```

Table 4. New Data Frame with only the numerical columns

	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	species_encoded	island_encoded	sex_encoded
0	39.10000	18.70000	181.000000	3750.000000	0	2	1
1	39.50000	17.40000	186.000000	3800.000000	0	2	0
2	40.30000	18.00000	195.000000	3250.000000	0	2	0
3	43.92193	17.15117	200.915205	4201.754386	0	2	1
4	36.70000	19.30000	193.000000	3450.000000	0	2	0

Next, two different methods were used for evaluating each feature's relevance: variance analysis and correlation analysis.

- Variance Analysis

  A VarianceThreshold selector was employed for evaluating the variance of each numerical feature. Features with low variance, indicating minimal variation in their values, were considered less informative, and excluding them can lead to more efficient models. The threshold for feature exclusion was set at 0, meaning no features were initially excluded. The resulting dataset, containing features with high variance, was then obtained.

```
# analyze the relevance of each feature using varience
from sklearn.feature_selection import VarianceThreshold
import pandas as pd
```

```
# Initialize the VarianceThreshold selector with a threshold 0 to not
exclude anything yet
selector = VarianceThreshold(threshold=0)

# Fit the selector to the data
selector = selector.fit(df_encoded)

# Apply the selector to the data
df_high_variance = selector.transform(df_encoded)

# Variance of the data
variances = selector.variances_

variance_df = pd.DataFrame({'Column Name': numarical_columns, 'Variance':
variances})

variance_df
```

Table 5. Features Variances

	Column Name	Variance
0	bill_length_mm	27.500000
1	bill_depth_mm	3.865798
2	flipper_length_mm	59.000000
3	body_mass_g	3600.000000
4	species_encoded	0.795700
5	island_encoded	0.525825
6	sex_encoded	0.249586

As shown in table 5 above, ‘bill_length_mm' and 'flipper_length_mm' columns show higher variances, reflecting a greater variability in these attributes. Conversely, 'sex_encoded' shows lower variance, indicating that its impact on the dataset's overall variability may be limited.

- Correlation Analysis:

  The correlation between each feature and the target variable ('species_encoded') was calculated using the correlation coefficient to understand the linear relationships between the features and the target. The 'corrwith' method from pandas was employed to compute the correlation coefficients. The absolute values of correlations were taken, to focus on the strength not the direction of the relationships.

```
# analyze the relevance of each feature using correlations

features = df.drop(['species_encoded'], axis=1)
target = df['species_encoded']

feature_correlations = features.corrwith(target).abs()

feature_correlations
```

```
bill_length_mm       0.728674
bill_depth_mm        0.741335
flipper_length_mm    0.851160
body_mass_g          0.747726
island_encoded       0.635659
sex_encoded          0.010240
```

From the output above, it's clear that the features 'flipper_length_mm,' 'bill_depth_mm,' 'body_mass_g,', 'bill_length_mm', and 'island_encoded' show relatively high positive correlations with the target variable. This means that these features are more strongly related to the target. On Conversely, 'sex_encoded' shows a low correlation, indicating a weaker linear relationship with the target.

So based on both variance and correlation analysis, the feature 'sex_encoded' was identified as having low relevance and was removed from the dataset to streamline the feature set. The new data frame is shown in table 6 below.

```
# remove the sex feature because it has the smallest value of each of
correlation and variance

df_encoded.drop(['sex_encoded'], axis=1, inplace=True)
df_encoded.head()
```

Table 6. Data Frame after Removing 'sex_encoded' Feature

	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	species_encoded	island_encoded
0	39.10000	18.70000	181.000000	3750.000000	0	2
1	39.50000	17.40000	186.000000	3800.000000	0	2
2	40.30000	18.00000	195.000000	3250.000000	0	2
3	43.92193	17.15117	200.915205	4201.754386	0	2
4	36.70000	19.30000	193.000000	3450.000000	0	2

6. Splitting Dataset

The dataset was split into training and testing sets. The splitting strategy is performed to ensure a realistic assessment of model generalization on unseen data. The features matrix X is defined by excluding the target variable 'species_encoded' from the data frame, while the target variable y is set as the species_encoded column. The test size was set to be 30% of the dataset size. X_train, and y_train are the training features and the target respectively, which will be used on training the model. While X_test, and y_test are the testing features and target respectively, and they will be used to evaluate the performance of the model by comparing the predicted y from applying the test features, with the real output which is y_test.

```
from sklearn.model_selection import train_test_split

X = df_encoded.drop(['species_encoded'], axis=1)
y = df_encoded['species_encoded']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=95)
```

7. Scaling Numerical Features

To ensure consistent scaling across numerical variables, the Min-Max Scaling technique is applied using the MinMaxScaler. Min-Max Scaling transforms the data into a consistent range between 0 and 1, preventing features with larger scales from dominating the learning process. The scaler was fitted on the training data X_train to learn the scaling parameters and then applied to both the training and testing sets.

```
# apply the MinMaxScalar

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Histograms were plotted as shown in figure 2 below to compare the distribution of original and scaled values for each numerical feature.

```
# plot the original and scaled X_train
fig, axs = plt.subplots(figsize=(18, 5),ncols=5,nrows=2)
for i, col in enumerate(X.columns):
  sns.histplot(x=X_train[col], ax=axs[0,i], color='b',
kde=True);axs[0,i].set_title(f"Original {col}")
  sns.histplot(x=X_train_scaled[:, i], ax=axs[1,i], color='r',
kde=True);axs[1,i].set_title(f"Scalled {col}")

fig.subplots_adjust(hspace=1)
```

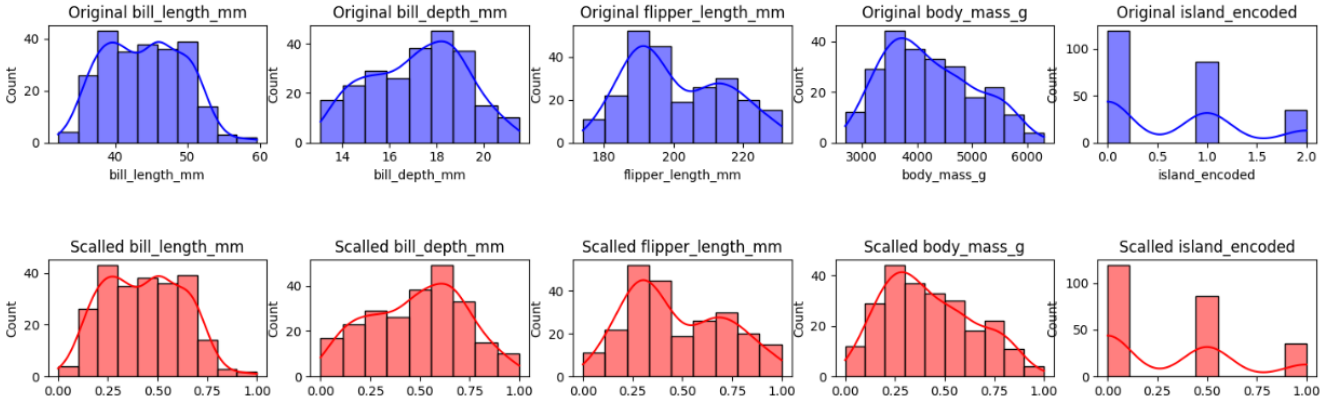

Figure 2. Original and Scaled Features Distribution

As noticed from figure 2 above, the shape and characteristics of the distributions before and after the scaling are still the same, contributing to the reliability of subsequent model training and evaluation.

8. Applying Dimensionality Reduction Techniques

To reduce the high dimensionality of the data since it has 5 features, the Principal Component Analysis (PCA) was applied. The number of components used to the PCA was 4 components. The PCA first was fitted on the training data X_train and then applied to both the training and testing sets. The explained variance ratio for each principal component was displayed.

```
from sklearn.decomposition import PCA

X_train, X_test = X_train_scaled, X_test_scaled
pca = PCA(n_components=4)
pca_X_train = pca.fit_transform(X_train)
pca_X_test = pca.transform(X_test)
```

```
print(f"PCA components' explained variance ratio are:
{pca.explained_variance_ratio_}")
```

```
PCA components' explained variance ratio are: [0.64935062 0.18625237
0.10120047 0.04492102]
```

PCA components' explained variance ratio represents the percentage of variance retained by each principal component. As seen in the output, the first component contributes almost 64.94% of the variance, and the first four components together contributes for nearly 99.15% of the variance, which means that even if the number of features is reduced, there is a sufficient amount of information in the components.

Below, PCA train and test data frames are shown.

```
pca_train_df = pd.DataFrame(pca_X_train)
pca_train_df.head()
```

Table 7. PCA X_train Data Frame

	0	1	2	3
0	-0.586231	0.053876	-0.269822	-0.015825
1	0.298754	0.109630	-0.118591	0.039950
2	0.286678	-0.182075	0.345504	0.206561
3	0.124784	0.653080	-0.010966	0.080850
4	0.271184	-0.015260	0.031238	0.206238

```
pca_test_df = pd.DataFrame(pca_X_test)
pca_test_df.head()
```

Table 8. PCA X_test Data Frame

	0	1	2	3
0	0.667565	-0.186492	-0.263780	-0.027812
1	-0.625790	-0.179056	0.050009	-0.266668
2	0.415893	0.276731	-0.173639	-0.106978
3	0.748180	-0.122641	-0.223612	-0.014092
4	0.725870	-0.259876	-0.066332	-0.157803

9. Training and Evaluating the Model

The Random Forest model was trained on preprocessed data. After that, the model was used to predict y. The accuracy was evaluated by accuracy_score function, and as shown below, the accuracy on the preprocessed data was almost 99.04%. This high accuracy shows that the model can generate accurate predictions.

```
# training the model on the preprocessed data
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

classifier = RandomForestClassifier(random_state=42)
classifier.fit(pca_X_train, y_train)
y_pred = classifier.predict(pca_X_test)
preprocessing_accuracy = accuracy_score(y_test, y_pred)

print(f'The accuracy on the preprocessed data = {preprocessing_accuracy}')
```

```
The accuracy on the preprocessed data = 0.9903846153846154
```

To compare the model's performance with and without the preprocessing pipeline, the model was trained on the raw data from the same dataset. The classifier does not take null or non-numerical values, so the rows that contain null values were removed, and the non-numerical columns were encoded.

```
# training the model on the raw data
df_raw = load_dataset('penguins')

# delete raws that contain null values because the classifier does not
accept null values
df_raw.dropna(axis=0, inplace=True)

# encode the non numerical values so the classifier can take them
df_raw['island_encoded'] = label_encoder.fit_transform(df_raw['island'])
df_raw['sex_encoded'] = label_encoder.fit_transform(df_raw['sex'])
df_raw['species_encoded'] = label_encoder.fit_transform(df_raw['species'])

df_raw.drop(['island', 'sex', 'species'], axis=1, inplace=True)
```

The raw data was split into training and testing sets, the test size was also 30%. The model was trained on the training raw data and used to predict y by using the testing raw data. The accuracy was evaluated, and it was 98%.

```
X_raw = df_raw.drop(['species_encoded'], axis=1)
y_raw = df_raw['species_encoded']

X_train_raw, X_test_raw, y_train_raw, y_test_raw = train_test_split(X_raw,
y_raw, test_size=0.3, random_state=95)

classifier_raw = RandomForestClassifier(random_state=42)
classifier_raw.fit(X_train_raw, y_train_raw)
y_pred_raw = classifier_raw.predict(X_test_raw)
accuracy_raw = accuracy_score(y_test_raw, y_pred_raw)

print(f'The accuracy on the raw data = {accuracy_raw}')
```

```
The accuracy on the raw data = 0.98
```

The preprocessed data produced an accuracy of 0.99, which was higher than the raw data's 0.98 accuracy. This enhancement means that the Random Forest model's performance has been positively affected by the preprocessing pipeline. The higher accuracy on the preprocessed data shows the effectiveness of preprocessing techniques in enhancing the model's predictive capabilities.

Conclusion

In conclusion, the thorough analysis and preprocessing of the penguin dataset have led to significant results and enhanced the performance of machine learning models. We have improved the data's effectiveness for training models by taking many steps like handling missing values, encoding categorical variables, using feature selection, and dimensionality reduction techniques. The validation process demonstrated the effectiveness of the preprocessing pipeline by comparing the accuracy of a Random Forest model on preprocessed and raw data; the preprocessed data produced a higher accuracy of 0.99 compared to 0.98 on the raw data. This demonstrates how essential effective preprocessing techniques are to maximize model results.

The process of transforming variables, resolving data quality issues, handling outliers, and doing data exploration has not only made the dataset ready for modeling but also expanded our comprehension of the characteristics of the data. This case study highlights the importance of a thoughtful and systematic approach to data preprocessing for effective machine learning endeavors.