Faculty of Engineering & Technology

Electrical and Computer Engineering Department

ENCS5141
INTELLIGENT SYSTEMS LAB
Report No.1

Experiment #2: Data Visualization and Data Cleaning

Experiment #3: EXPLORATORY DATA ANALYSIS

Prepared by: Islam Jihad **ID:** 1191375

Instructor's Name: Dr. Aziz Qaroush

Teaching Assistant: Eng. Mazen Amria

Section: 2.

19th Nov 2023

BIRZEIT

Abstract

In this paper, we do a lot of preprocessing on the Penguins dataset, which contains information on many penguin species, including physical features and observation zones. Our primary aim is to prepare the dataset carefully for later machine learning analysis. This procedure begins with importing the dataset and conducting a preliminary study to understand its structure, characteristics, and missing values. We then address data quality issues by implementing appropriate solutions for dealing with missing data and outliers. We use feature selection strategies to improve feature relevance for machine learning tasks, assessing the significance of each variable. Categorical variables are encoded into a numerical format suited for machine learning models, and the dataset is divided into training and testing subsets for evaluating model performance.

Scaling or normalizing procedures are used to provide uniform scaling across numerical characteristics. Concurrently, dimensionality reduction approaches are used to minimize data size while retaining critical information. The preprocessing pipeline's effectiveness is evaluated by training and assessing a Random Forest model on the preprocessed data. This performance is compared to a model trained on raw data, indicating that the preprocessing procedures increase overall model performance. This research provides a systematic and rigorous method for preparing the Penguins dataset, opening the door for more effective and informative machine learning investigations.

Table of Contents

Abstract ii
Table of Figures iv
Table of Tables v
1. Introduction 1
2. Procedure and discussion 2
  Data Loading and Initial Exploration 2
  Data Exploration 2
    Basic Information: 2
    Statistical Summary: 2
    Missing Values: 3
    Missing Values Analysis 3
    Rows with Missing Values 3
  Handling Missing values 4
    Empty Rows Check 4
    Median Imputation for Numeric Attributes 4
    Categorical Encoding and Correlation Analysis 4
    Correlation Matrix: 5
    Linear Regression for Imputation: 5
    Simple Imputer (sex column) 6
  Scaling Numerical Features 7
  Splitting the Dataset for Machine Learning 8
  Dimensionality Reduction Methods 9
    with PCA only 9
    Using VarianceThreshold for Feature Selection 11
    Using KBest for Feature Selection 12
    Using PCA &VarianceThreshold for Feature Selection 14
    Using PCA & KBest for Feature Selection 15
  Raw Data Testing: 16
3. Conclusion 18
4. References 19

Table of Figures

Figure 1DF.info() data 2
Figure 2 df.describe() data 3
Figure 3 df.isnull().sum() data 3
Figure 4 Summation percent 3
Figure 5 Empty Rows 4
Figure 6 body mass median 4
Figure 7 Bill Depth median 4
Figure 8 Bill Length median 4
Figure 9 Categorical Encoding 5
Figure 10 linear regression Function 6
Figure 11 SimpleImputer process 7
Figure 12 Final cleaning output 7
Figure 13 Scaling Numerical Features code 8
Figure 14 Train & Test split 8
Figure 15 with PCA only code 10
Figure 16 with PCA only results 10
Figure 17 choosing best threshold 11
Figure 18 VarianceThreshold code 11
Figure 19 VarianceThreshold results 12
Figure 20 KBest code 13
Figure 21KBest results 13
Figure 22 PCA &VarianceThreshold code 14
Figure 23 PCA &VarianceThreshold results 14
Figure 24 PCA & KBest code 15
Figure 25 PCA & KBest results 16
Figure 26 absolute difference between accuracies 17

Table of Tables

Table 1 Missing Values Rows 4
Table 2 Missing Values Rows after median substitution 4
Table 3 Correlation Matrix.. 5
Table 4 Scaled DataFrame 8

1. Introduction

This case study digs into the complexities of the Penguins dataset, which is a compilation of numerous penguin species, physical attributes, and observation locations. The dataset, chosen by our instructor for its richness and relevance, serves as the basis for using and grasping sophisticated data preparation techniques. Our adventure begins with importing the dataset and continues with basic information presentation, statistical summary, missing value handling, label encoding, correlation analysis, linear regression, and various imputation procedures.

Our main goals are to interpret the dataset's complexity, fill in the missing values, and improve feature significance for further analysis. A range of preprocessing techniques, including label encoding, scaling, correlation-based imputation, and linear regression, are used in Exp 2 and 3 to make sure the dataset is ready for model training.

The careful creation of the dataset and the subsequent assessment of machine learning models are what make this work significant. Using the Random Forest model, we conduct several experiments to examine the effects of several preprocessing methods, such as PCA-based dimensionality reduction, VarianceThreshold and KBest feature selection, and their combinations. The complete procedure is covered in this paper, which also provides insightful information on the dataset and the effectiveness of various preprocessing techniques. These kinds of breakthroughs help close the gap between theoretical understanding and real-world applications in data science and artificial intelligence.

2. Procedure and discussion

Data Loading and Initial Exploration

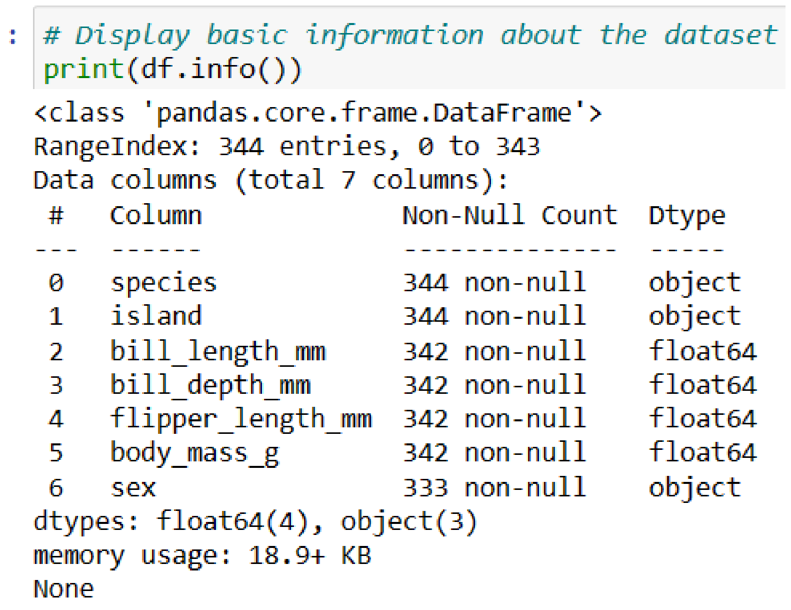
First, the 'load_dataset' function from seaborn is used to load the penguins dataset, and the data is put in the 'df' DataFrame. This dataset contains parameters including body mass, gender, flipper length, bill length, and bill depth for a variety of penguin species. The first five rows of the dataset are shown using the **.head()** function to give a quick overview of its structure and content.

Data Exploration

In order to understand the structure and properties of the penguins dataset, we thoroughly explore it in this part.

Basic Information:

344 records spread over 7 columns make up the dataset, according to the info() function. For every characteristic, it gives information about the non-null counts and data types. A few entries are noteworthy for having missing values, especially in columns like "body mass g," "bill length mm," "bill depth mm," "flipper length mm," and "sex."



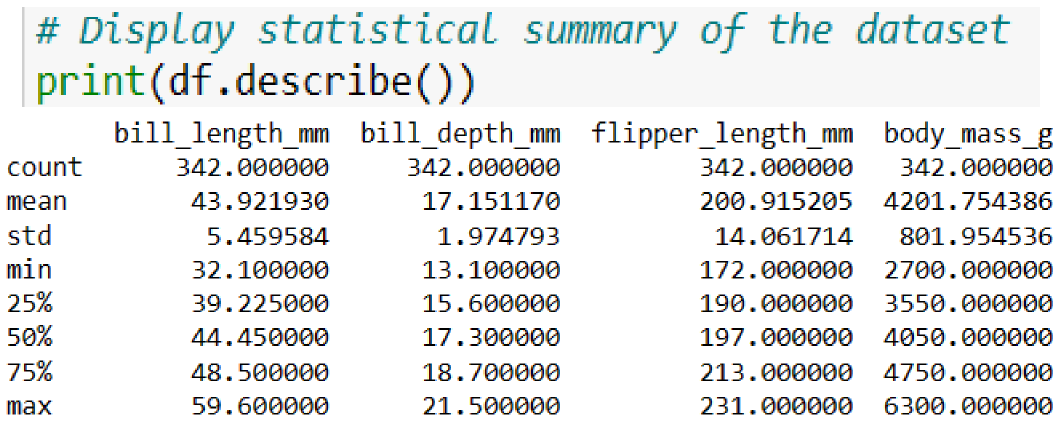
```
# Display basic information about the dataset
print(df.info())
```

```
<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
None
```

Figure 1DF.info() data

Statistical Summary:

A statistical overview of the numerical columns is provided by the describe() function, which highlights important metrics including mean, standard deviation, and quartile values. Understanding the distribution and variability of numerical characteristics such as " body_mass_g" "flipper_length_mm" " bill_depth_mm" and " bill_length_mm" is made easier with the help of this summary.



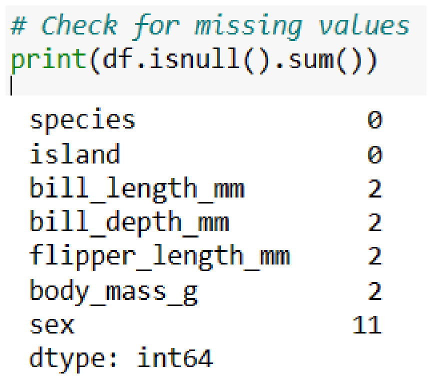
```
# Display statistical summary of the dataset
print(df.describe())
```

```
       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
```

Figure 2 df.describe() data

Missing Values:

The isnull().sum() function indicates that missing values are present in the dataset. Interestingly, 'sex' contains eleven missing values compared to two missing values for each of the other numeric variables. One of the most important steps in the next data preparation stages will be to address these missing variables.



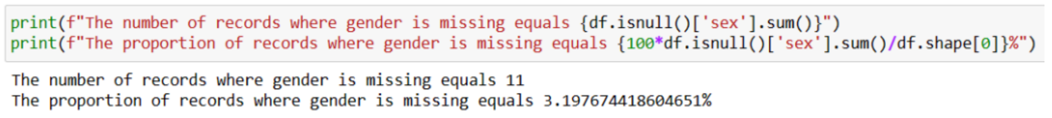
```
# Check for missing values
print(df.isnull().sum())
```

```
species              0
island               0
bill_length_mm       2
bill_depth_mm        2
flipper_length_mm    2
body_mass_g          2
sex                 11
dtype: int64
```

Figure 3 df.isnull().sum() data

Missing Values Analysis

Further investigation reveals that 11 rows have at least one column's worth of missing values. This amounts to about 3.20 percent of the entire dataset. It will be crucial to locate and fix these missing values in order to preserve the dataset's integrity for ensuing modeling and analysis.



```
print(f"The number of records where gender is missing equals {df.isnull()['sex'].sum()}")
print(f"The proportion of records where gender is missing equals {100*df.isnull()['sex'].sum()/df.shape[0]}%")
```

```
The number of records where gender is missing equals 11
The proportion of records where gender is missing equals 3.197674418604651%
```

Figure 4 Summation percent

Rows with Missing Values

The dataset's rows with at least one column that has missing values are shown in the table. Interestingly, these entries span a variety of penguin species, islands, and characteristics. It will be essential to address these missing values in order to guarantee the correctness and completeness of the dataset for further modeling and analysis.

Table 1 Missing Values Rows

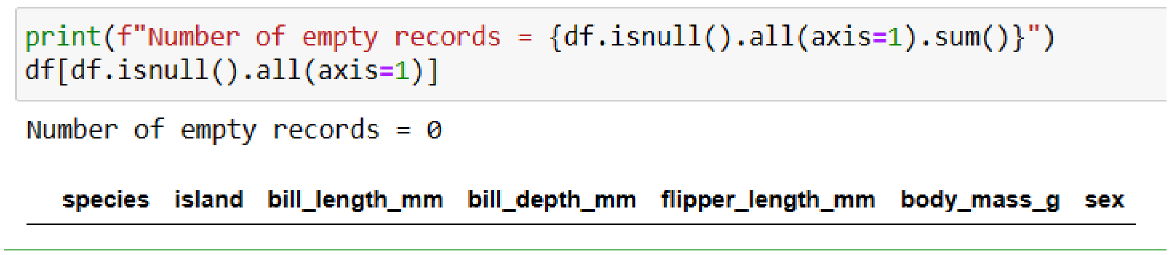
	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN
8	Adelie	Torgersen	34.1	18.1	193.0	3475.0	NaN
9	Adelie	Torgersen	42.0	20.2	190.0	4250.0	NaN
10	Adelie	Torgersen	37.8	17.1	186.0	3300.0	NaN
11	Adelie	Torgersen	37.8	17.3	180.0	3700.0	NaN
47	Adelie	Dream	37.5	18.9	179.0	2975.0	NaN
246	Gentoo	Biscoe	44.5	14.3	216.0	4100.0	NaN
286	Gentoo	Biscoe	46.2	14.4	214.0	4650.0	NaN
324	Gentoo	Biscoe	47.3	13.8	216.0	4725.0	NaN
336	Gentoo	Biscoe	44.5	15.7	217.0	4875.0	NaN
339	Gentoo	Biscoe	NaN	NaN	NaN	NaN	NaN

Handling Missing values

I started to handle missing values in different ways:

Empty Rows Check

A search has been done through the dataset for rows that are completely empty. According to the outcome, there are no records with missing values in any of the columns. Therefore, there is no need to delete entirely empty rows from the dataset.



```
print(f"Number of empty records = {df.isnull().all(axis=1).sum()}")
df[df.isnull().all(axis=1)]
```

```
Number of empty records = 0
```

species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex

Figure 5 Empty Rows

Median Imputation for Numeric Attributes

A median imputation technique has been used to fill in the missing values for the numerical characteristics "body_mass_g," "bill_depth_mm," and "bill_length_mm." The median is preferred over the mean because it provides better stability for the imputation technique and is more resistant against possible outliers, particularly when dealing with skewed or non-normally distributed data.

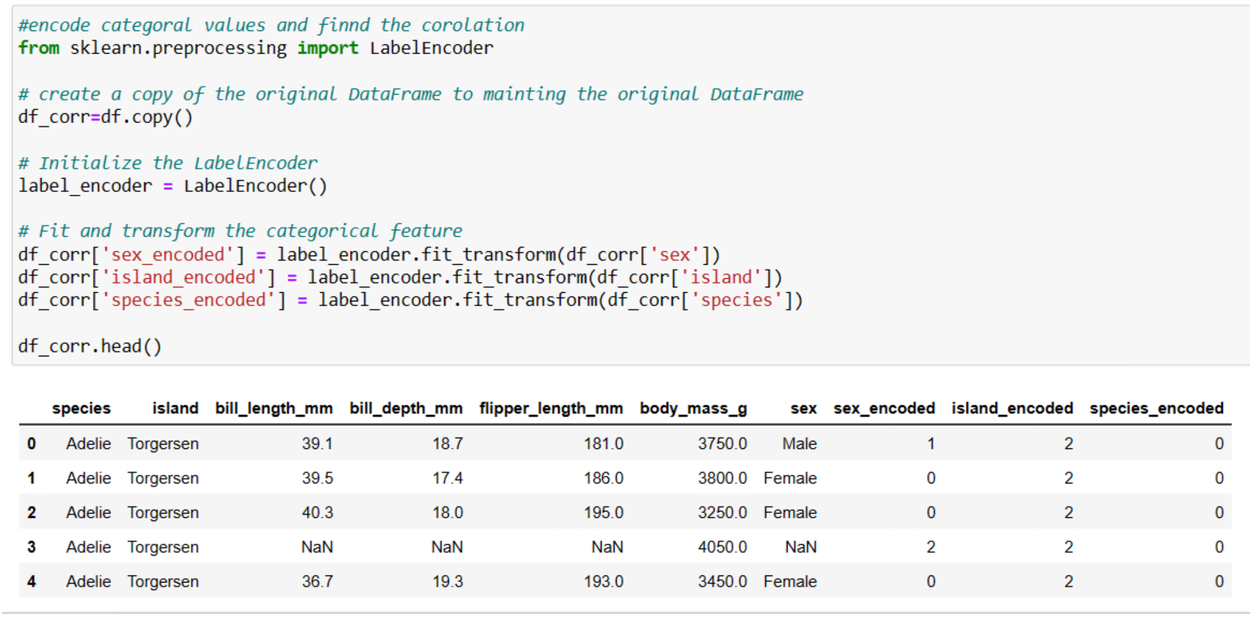
Body Mass (body_mass_g):

Bill Depth (bill_depth_mm):

Bill Length (bill_length_mm):

Categorical Encoding and Correlation Analysis

The LabelEncoder has been used to encode categorical variables in order to prepare the dataset for machine learning research. The links between numerical characteristics are shown in the correlation matrix below, which also sheds light on how interdependent they are.



```
#encode categoral values and finnd the corolation
from sklearn.preprocessing import LabelEncoder

# create a copy of the original DataFrame to mainting the original DataFrame
df_corr=df.copy()

# Initialize the LabelEncoder
label_encoder = LabelEncoder()

# Fit and transform the categorical feature
df_corr['sex_encoded'] = label_encoder.fit_transform(df_corr['sex'])
df_corr['island_encoded'] = label_encoder.fit_transform(df_corr['island'])
df_corr['species_encoded'] = label_encoder.fit_transform(df_corr['species'])

df_corr.head()
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex	sex_encoded	island_encoded	species_encoded
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	Male	1	2	0
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	Female	0	2	0
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	Female	0	2	0
3	Adelie	Torgersen	NaN	NaN	NaN	4050.0	NaN	2	2	0
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	Female	0	2	0

Figure 9 Categorical Encoding

Correlation Matrix:

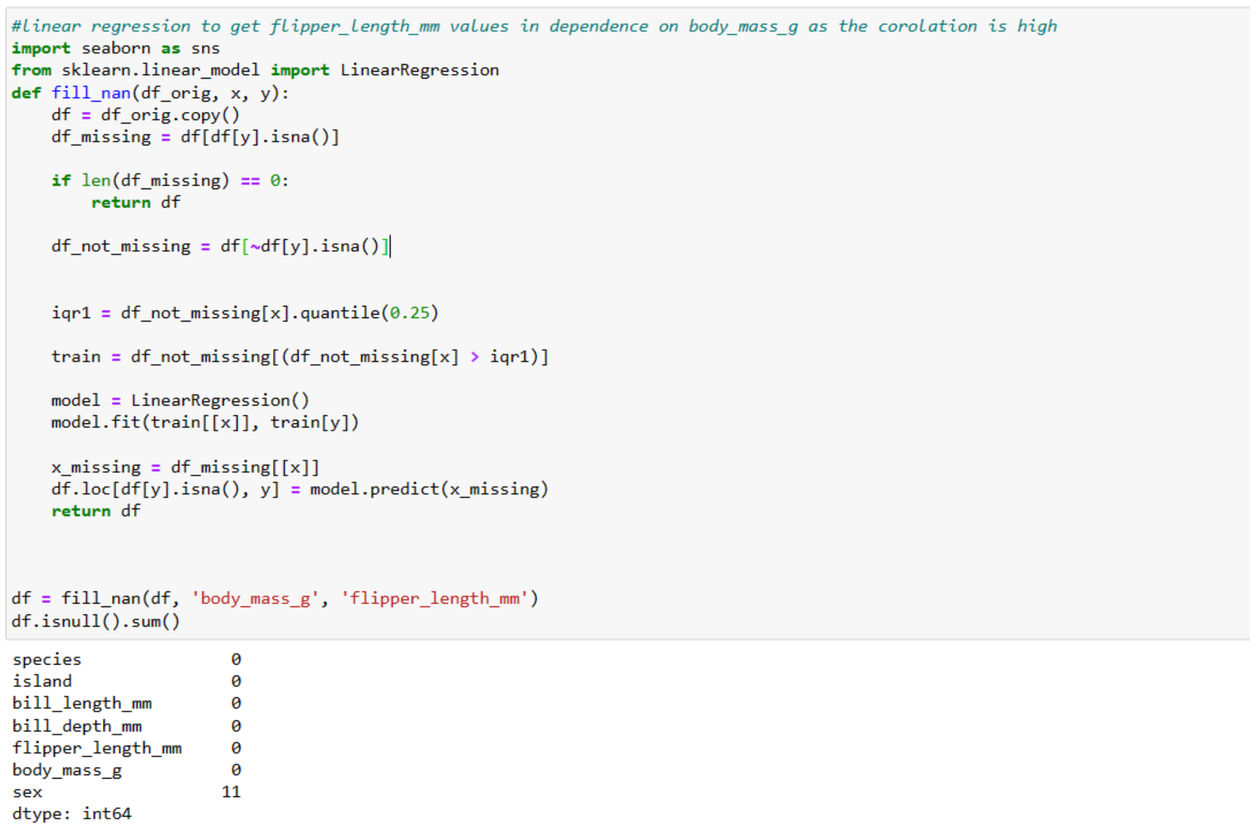
Table 3 Correlation Matrix

	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex_encoded	island_encoded	species_encoded
bill_length_mm	1.000000	-0.235053	0.656181	0.595110	0.271440	-0.353647	0.731369
bill_depth_mm	-0.235053	1.000000	-0.583851	-0.471916	0.311460	0.571035	-0.744076
flipper_length_mm	0.656181	-0.583851	1.000000	0.871202	0.215992	-0.565825	0.854307
body_mass_g	0.595110	-0.471916	0.871202	1.000000	0.361224	-0.561515	0.750491
sex_encoded	0.271440	0.311460	0.215992	0.361224	1.000000	0.029246	0.008559
island_encoded	-0.353647	0.571035	-0.565825	-0.561515	0.029246	1.000000	-0.635659
species_encoded	0.731369	-0.744076	0.854307	0.750491	0.008559	-0.635659	1.000000

The correlation matrix offers important information on the direction and strength of relationships between various characteristics. For example, there is a strong positive correlation of 0.87 between "flipper_length_mm" and "body_mass_g," suggesting a possibly substantial reliance.

Linear Regression for Imputation:

'Flipper_length_mm' has missing values, which have been addressed by using a linear regression model. Given their strong association, the missing values were imputed based on how they related to the 'body_mass_g' characteristic. This method improves the dataset's completeness.



```
#linear regression to get flipper_length_mm values in dependence on body_mass_g as the corolation is high
import seaborn as sns
from sklearn.linear_model import LinearRegression
def fill_nan(df_orig, x, y):
    df = df_orig.copy()
    df_missing = df[df[y].isna()]

    if len(df_missing) == 0:
        return df

    df_not_missing = df[~df[y].isna()]


    iqr1 = df_not_missing[x].quantile(0.25)

    train = df_not_missing[(df_not_missing[x] > iqr1)]

    model = LinearRegression()
    model.fit(train[[x]], train[y])

    x_missing = df_missing[[x]]
    df.loc[df[y].isna(), y] = model.predict(x_missing)
    return df


df = fill_nan(df, 'body_mass_g', 'flipper_length_mm')
df.isnull().sum()
```

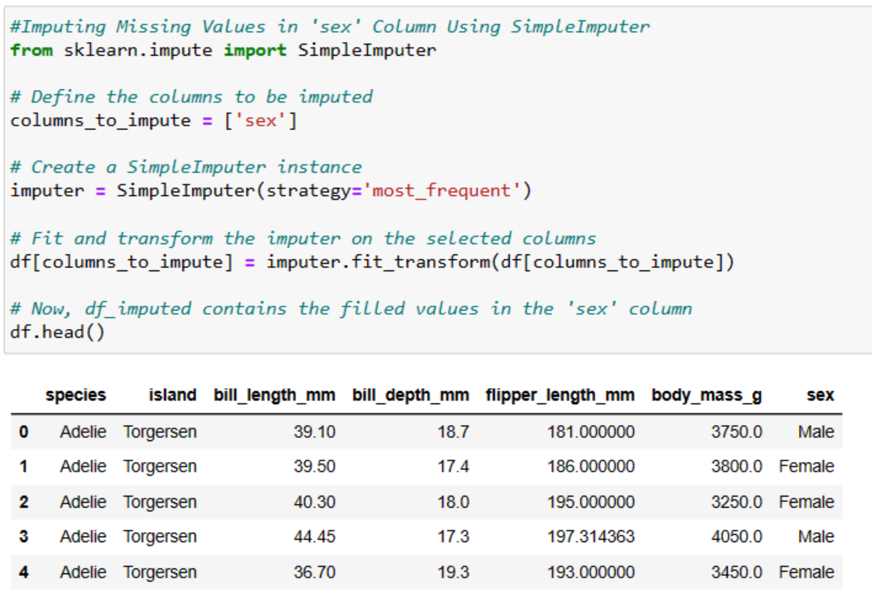
```
species               0
island                0
bill_length_mm        0
bill_depth_mm         0
flipper_length_mm     0
body_mass_g           0
sex                  11
dtype: int64
```

Figure 10 linear regression Function

Now that the values for "flipper_length_mm" have been imputed, the resultant dataset is more complete and ready for further machine learning research.

Simple Imputer (sex column)

During the preprocessing stage, the 'most frequent' option of the SimpleImputer technique was used in order to handle the missing values in the 'sex' column. The DataFrame, df, that is produced now has imputed values for the 'sex' column.

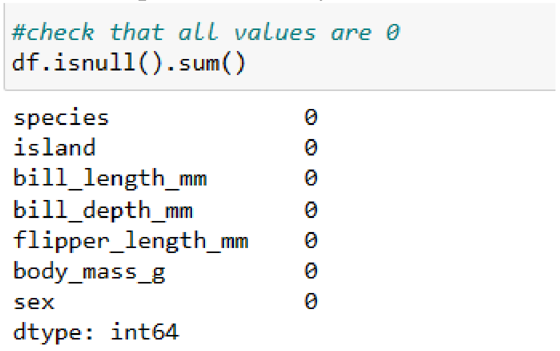


```
#Imputing Missing Values in 'sex' Column Using SimpleImputer
from sklearn.impute import SimpleImputer

# Define the columns to be imputed
columns_to_impute = ['sex']

# Create a SimpleImputer instance
imputer = SimpleImputer(strategy='most_frequent')

# Fit and transform the imputer on the selected columns
df[columns_to_impute] = imputer.fit_transform(df[columns_to_impute])

# Now, df_imputed contains the filled values in the 'sex' column
df.head()
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.10	18.7	181.000000	3750.0	Male
1	Adelie	Torgersen	39.50	17.4	186.000000	3800.0	Female
2	Adelie	Torgersen	40.30	18.0	195.000000	3250.0	Female
3	Adelie	Torgersen	44.45	17.3	197.314363	4050.0	Male
4	Adelie	Torgersen	36.70	19.3	193.000000	3450.0	Female

Figure 11 SimpleImputer process

Now all the input data are cleaned and there are no Nan values in the data, now that the 'sex' column has all of its values imputed, it is complete and ready for additional analysis.



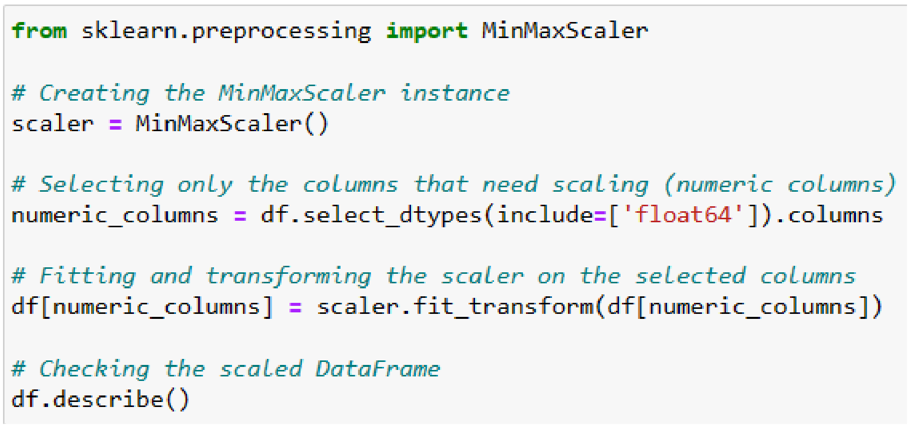
```
#check that all values are 0
df.isnull().sum()
```

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

Figure 12 Final cleaning output

Scaling Numerical Features

The Min-Max scaling approach has been used to scale the numerical features in the dataset to achieve consistent scaling across variables. For this, the MinMaxScaler instance from sklearn.preprocessing has been used. For scaling, only the columns containing numerical values have been chosen.



```python
from sklearn.preprocessing import MinMaxScaler

# Creating the MinMaxScaler instance
scaler = MinMaxScaler()

# Selecting only the columns that need scaling (numeric columns)
numeric_columns = df.select_dtypes(include=['float64']).columns

# Fitting and transforming the scaler on the selected columns
df[numeric_columns] = scaler.fit_transform(df[numeric_columns])

# Checking the scaled DataFrame
df.describe()
```

Figure 13 Scaling Numerical Features code

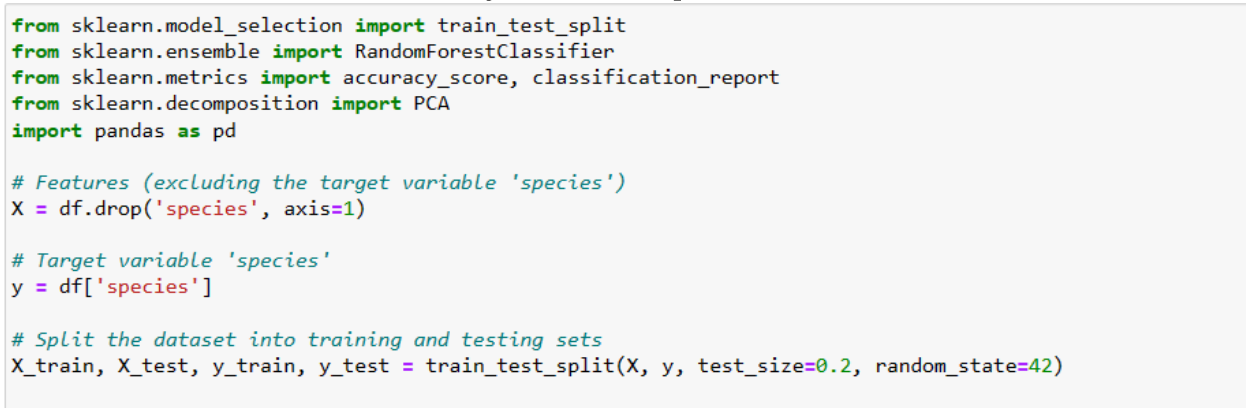
Now that the scaling has been done successfully, all numerical characteristics have values that fall into the range [0, 1]. The performance of machine learning models that depend on numerical input characteristics depends on this constant scaling.
Certainly! Here's the information presented in a table format:

Table 4 Scaled DataFrame

	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
count	344.000000	344.000000	344.000000	344.000000
mean	0.430000	0.482385	0.489733	0.416909
std	0.197956	0.234412	0.237684	0.222138
min	0.000000	0.000000	0.000000	0.000000
25%	0.260909	0.297619	0.305085	0.236111
50%	0.449091	0.500000	0.423729	0.375000
75%	0.596364	0.666667	0.694915	0.569444
max	1.000000	1.000000	1.000000	1.000000

Splitting the Dataset for Machine Learning

The dataset has been divided into training and testing subsets in order to evaluate the effectiveness of machine learning models. The train test split method from **sklearn.model_selection** is used to do this. All columns other than the target variable, "species," are included in the features.



```python
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.decomposition import PCA
import pandas as pd

# Features (excluding the target variable 'species')
X = df.drop('species', axis=1)

# Target variable 'species'
y = df['species']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Figure 14 Train & Test split

Eighty percent of the data will be utilized for training and twenty percent will be used for testing, according to the **'test_size'** parameter, which is set to 0.2. Reproducibility is guaranteed via the **'random_state'** argument, which fixes the random number generation seed.

Dimensionality Reduction Methods

with PCA only

To decrease the dataset's dimensionality and achieve a balance between computational efficiency and information retention, Principal Component Analysis (PCA) was utilized. The following steps were included in the procedure:

1. **Training a Classifier on Original Features:** To provide a baseline accuracy, a Random Forest Classifier was first trained using the original features.
2. **Application of PCA:** Five components were to be retained when a PCA instance was established.
3. **Training a Classifier on PCA Components:** The decreased PCA components were used to train a new Random Forest Classifier.

The results of the procedure of dimensionality reduction were assessed and compiled as:

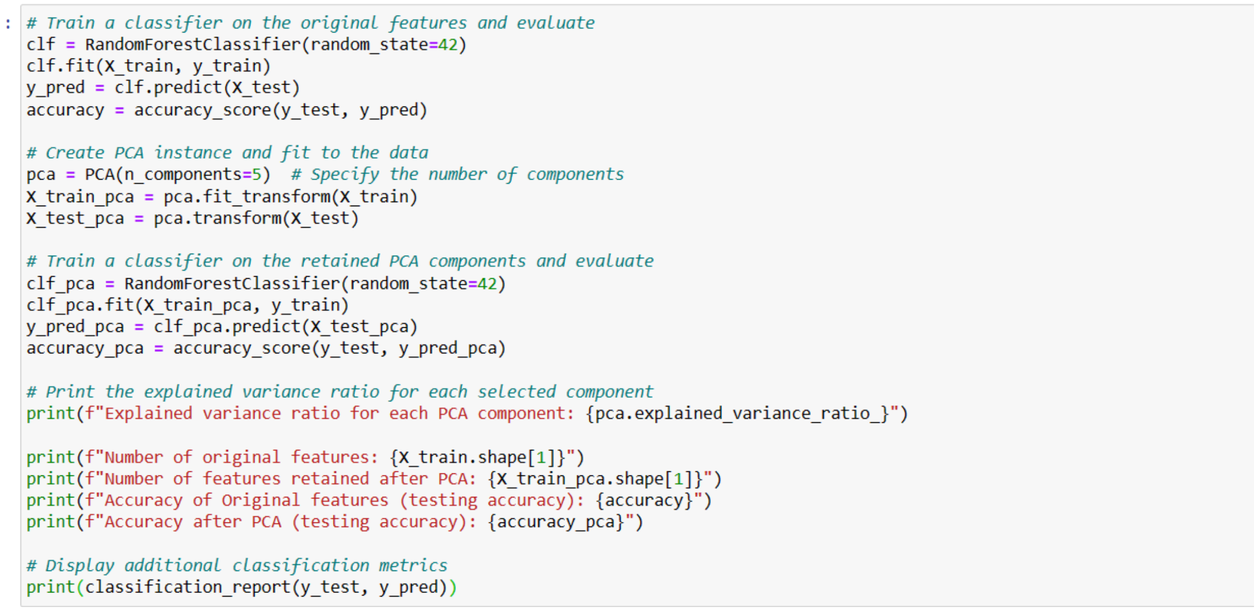
- **Explained Variance Ratio:** Each PCA component's explained variance ratio sheds light on the relative contributions of the various dimensions. In this instance, the initial component held on to around 60% of the variation, with successive components contributing less and less.

- **Number of Features:** Five of the six features in the original dataset were kept when PCA was used.

- **Accuracy Comparison:** Following dimensionality reduction with PCA, the Random Forest Classifier's accuracy significantly dropped, going from 98.55 percent to 97.10 percent.

- **Classification Report:** For every class, the classification report presents measures such as accuracy, recall, and F1-score.

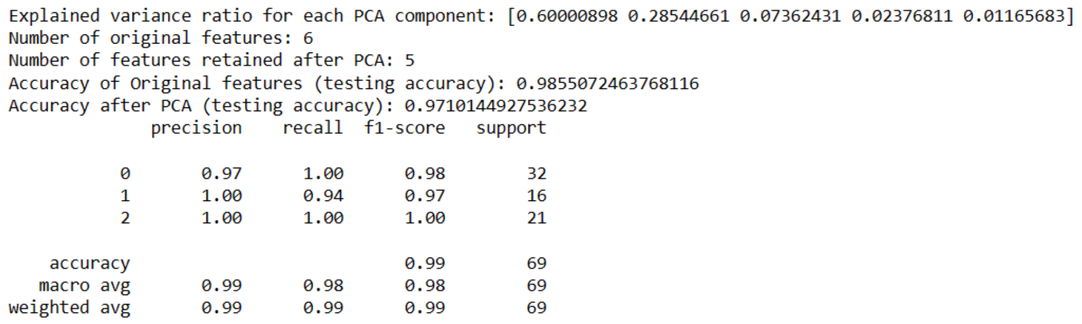
- The report makes it easier to comprehend how the model performs for each class when dimensionality is reduced.



```
# Train a classifier on the original features and evaluate
clf = RandomForestClassifier(random_state=42)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

# Create PCA instance and fit to the data
pca = PCA(n_components=5)  # Specify the number of components
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

# Train a classifier on the retained PCA components and evaluate
clf_pca = RandomForestClassifier(random_state=42)
clf_pca.fit(X_train_pca, y_train)
y_pred_pca = clf_pca.predict(X_test_pca)
accuracy_pca = accuracy_score(y_test, y_pred_pca)

# Print the explained variance ratio for each selected component
print(f"Explained variance ratio for each PCA component: {pca.explained_variance_ratio_}")

print(f"Number of original features: {X_train.shape[1]}")
print(f"Number of features retained after PCA: {X_train_pca.shape[1]}")
print(f"Accuracy of Original features (testing accuracy): {accuracy}")
print(f"Accuracy after PCA (testing accuracy): {accuracy_pca}")

# Display additional classification metrics
print(classification_report(y_test, y_pred))
```

Figure 15 with PCA only code

There is a trade-off between reduced dimensionality and maintained accuracy, as seen by the decrease in accuracy following PCA. Although there was a little decrease in the model's performance, the measures of accuracy, recall, and F1-score offer a detailed insight into the effects on different classes. Understanding the contribution of each maintained dimension and the interpretability of the reduced feature space is possible through the explained variance ratio. In light of these trade-offs, choosing to employ PCA should be carefully thought out, weighing the advantages of computing power against the need to preserve important data for optimal model performance.

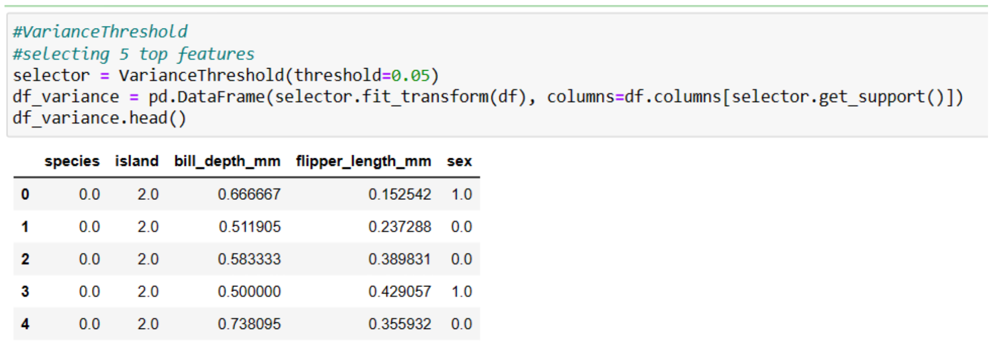


```
Explained variance ratio for each PCA component: [0.60000898 0.28544661 0.07362431 0.02376811 0.01165683]
Number of original features: 6
Number of features retained after PCA: 5
Accuracy of Original features (testing accuracy): 0.9855072463768116
Accuracy after PCA (testing accuracy): 0.9710144927536232
              precision    recall  f1-score   support

           0       0.97      1.00      0.98        32
           1       1.00      0.94      0.97        16
           2       1.00      1.00      1.00        21

    accuracy                           0.99        69
   macro avg       0.99      0.98      0.98        69
weighted avg       0.99      0.99      0.99        69
```

Figure 16 with PCA only results

Using VarianceThreshold for Feature Selection

Selecting Features Using Variance Threshold



```
#VarianceThreshold
#selecting 5 top features
selector = VarianceThreshold(threshold=0.05)
df_variance = pd.DataFrame(selector.fit_transform(df), columns=df.columns[selector.get_support()])
df_variance.head()
```

	species	island	bill_depth_mm	flipper_length_mm	sex
0	0.0	2.0	0.666667	0.152542	1.0
1	0.0	2.0	0.511905	0.237288	0.0
2	0.0	2.0	0.583333	0.389831	0.0
3	0.0	2.0	0.500000	0.429057	1.0
4	0.0	2.0	0.738095	0.355932	0.0

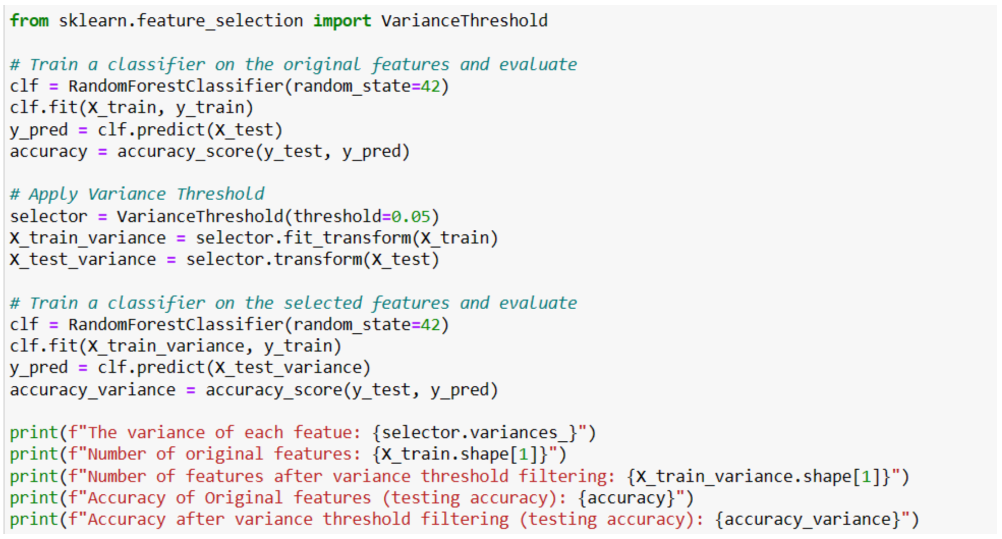
Figure 17 choosing best threshold

We carried out feature selection using the VarianceThreshold approach in order to determine and keep the top 5 features according to their variance. The following is a summary of the methodology:

1. **Original Features Classifier:** To provide a baseline accuracy, a Random Forest Classifier was first trained using the original features.

2. **Making Use of Variance Threshold:** To choose features whose variance was higher than the predetermined 0.05 threshold, VarianceThreshold was used.

3. **Selected Features Classifier:** The accuracy of the newly trained Random Forest Classifier was then assessed after it was trained on the chosen features.



```
from sklearn.feature_selection import VarianceThreshold

# Train a classifier on the original features and evaluate
clf = RandomForestClassifier(random_state=42)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

# Apply Variance Threshold
selector = VarianceThreshold(threshold=0.05)
X_train_variance = selector.fit_transform(X_train)
X_test_variance = selector.transform(X_test)

# Train a classifier on the selected features and evaluate
clf = RandomForestClassifier(random_state=42)
clf.fit(X_train_variance, y_train)
y_pred = clf.predict(X_test_variance)
accuracy_variance = accuracy_score(y_test, y_pred)

print(f"The variance of each featue: {selector.variances_}")
print(f"Number of original features: {X_train.shape[1]}")
print(f"Number of features after variance threshold filtering: {X_train_variance.shape[1]}")
print(f"Accuracy of Original features (testing accuracy): {accuracy}")
print(f"Accuracy after variance threshold filtering (testing accuracy): {accuracy_variance}")
```

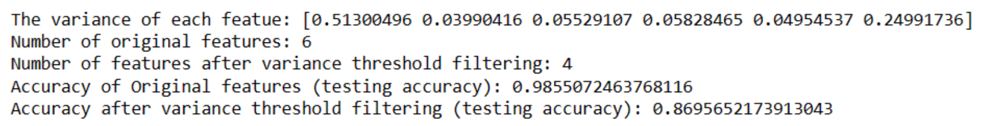
Figure 18 VarianceThreshold code

The following results of the feature selection procedure were assessed:

- **Variability of Every Aspect:** By using a VarianceThreshold, the variance of every feature was evaluated, offering valuable information on the variance distribution of each feature.

- **Count of Features:** The influence of the selection method on feature count was demonstrated by comparing the number of original features and the features that were kept following variance threshold filtering.

- **Accuracy Comparison:** The trade-off between feature reduction and model accuracy was shown by comparing the accuracy of the original features classifier with the classifier trained on the chosen features.



```
The variance of each featue: [0.51300496 0.03990416 0.05529107 0.05828465 0.04954537 0.24991736]
Number of original features: 6
Number of features after variance threshold filtering: 4
Accuracy of Original features (testing accuracy): 0.9855072463768116
Accuracy after variance threshold filtering (testing accuracy): 0.8695652173913043
```

Figure 19 VarianceThreshold results

Careful attention is warranted given the observed drop in accuracy following the application of VarianceThreshold. The drop from 98.55 percent to 86.96 percent indicates that the predictive performance of the model was greatly influenced by some characteristics with reduced variance. The loss of these characteristics, which individually may have modest variation but collectively provide important information for successful predictions, might be blamed for the drop in accuracy.

Although the accuracy decrease is acknowledged, it is important to stress that feature selection frequently aims for more than just the maximum accuracy. Rather, it seeks to improve computing efficiency by removing elements that are unnecessary or less useful. In this case, depending on the particular goals of the study, the trade-off between a little decrease in accuracy and the computational advantages obtained by feature reduction should be carefully examined.

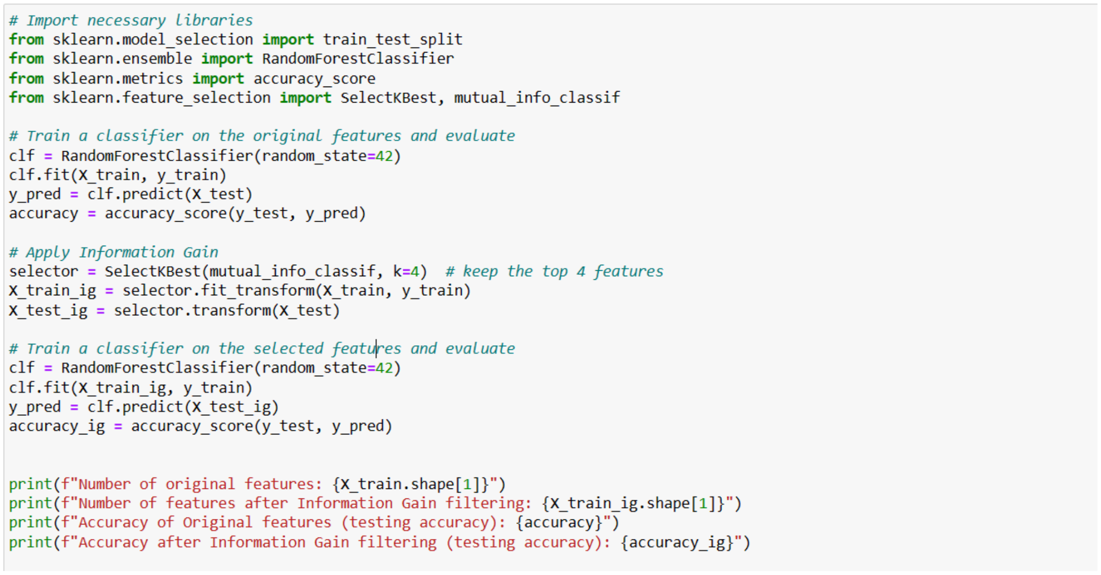
The choice to use VarianceThreshold should be in keeping with the overall objectives of the study, striking a balance between the computational efficiency obtained by simplified feature sets and the requirement for precise forecasts.

Using KBest for Feature Selection

In this step, we selected features for the penguins dataset using the SelectKBest technique and the mutual information criteria (mutual_info_classif). The main goal was to assess how this method affected the Random Forest classifier's performance in comparison to employing the initial feature set.

Subsequently, we utilized mutual_info_classif in conjunction with SelectKBest to determine the top 4 features according to information gain. Using this smaller collection of characteristics, we subsequently trained a Random Forest classifier.

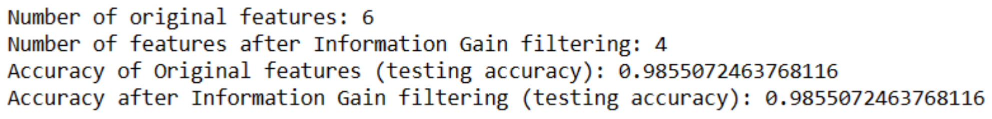
The outcomes showed that even after feature selection, the Random Forest classifier's accuracy stayed high. The accuracy of the chosen features as well as the original features was about 98.55 percent. This indicates that the model did extremely well on the smaller feature set, proving the chosen features' resilience in identifying the key patterns in the data.



```python
# Import necessary libraries
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.feature_selection import SelectKBest, mutual_info_classif

# Train a classifier on the original features and evaluate
clf = RandomForestClassifier(random_state=42)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

# Apply Information Gain
selector = SelectKBest(mutual_info_classif, k=4)  # keep the top 4 features
X_train_ig = selector.fit_transform(X_train, y_train)
X_test_ig = selector.transform(X_test)

# Train a classifier on the selected features and evaluate
clf = RandomForestClassifier(random_state=42)
clf.fit(X_train_ig, y_train)
y_pred = clf.predict(X_test_ig)
accuracy_ig = accuracy_score(y_test, y_pred)


print(f"Number of original features: {X_train.shape[1]}")
print(f"Number of features after Information Gain filtering: {X_train_ig.shape[1]}")
print(f"Accuracy of Original features (testing accuracy): {accuracy}")
print(f"Accuracy after Information Gain filtering (testing accuracy): {accuracy_ig}")
```

Figure 20 KBest code

When SelectKBest was applied, the number of features dropped from six to four. The accuracy of the model was unaffected by this decrease. This suggests that the four characteristics that were chosen have a great impact on the prediction job and hold important information for the classifier.



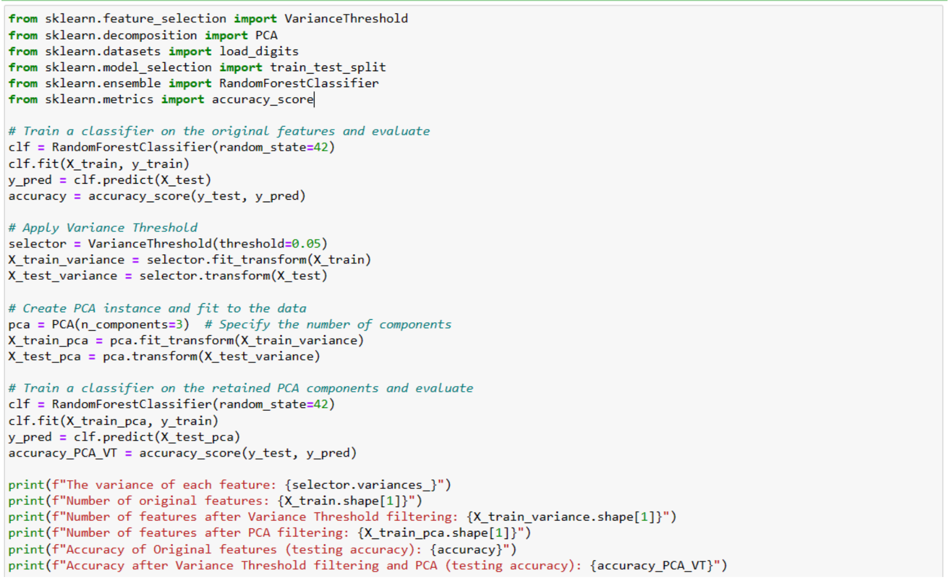
```
Number of original features: 6
Number of features after Information Gain filtering: 4
Accuracy of Original features (testing accuracy): 0.9855072463768116
Accuracy after Information Gain filtering (testing accuracy): 0.9855072463768116
```

Figure 21KBest results

In conclusion, SelectKBest with mutual_info_classif was used for feature selection, which resulted in both a high degree of accuracy and a simplified model. This shows how the chosen features successfully capture the discriminating power needed for precise classification, demonstrating the value of this feature selection technique in improving model interpretability and efficiency.

Using PCA &VarianceThreshold for Feature Selection

The procedure for using PCA and VarianceThreshold for feature selection on the Penguins dataset is described in this section. Examining how this combination strategy affects a Random Forest classifier's performance is the aim.



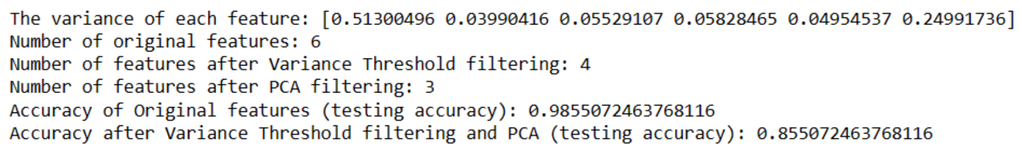
```python
from sklearn.feature_selection import VarianceThreshold
from sklearn.decomposition import PCA
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Train a classifier on the original features and evaluate
clf = RandomForestClassifier(random_state=42)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

# Apply Variance Threshold
selector = VarianceThreshold(threshold=0.05)
X_train_variance = selector.fit_transform(X_train)
X_test_variance = selector.transform(X_test)

# Create PCA instance and fit to the data
pca = PCA(n_components=3)  # Specify the number of components
X_train_pca = pca.fit_transform(X_train_variance)
X_test_pca = pca.transform(X_test_variance)

# Train a classifier on the retained PCA components and evaluate
clf = RandomForestClassifier(random_state=42)
clf.fit(X_train_pca, y_train)
y_pred = clf.predict(X_test_pca)
accuracy_PCA_VT = accuracy_score(y_test, y_pred)

print(f"The variance of each feature: {selector.variances_}")
print(f"Number of original features: {X_train.shape[1]}")
print(f"Number of features after Variance Threshold filtering: {X_train_variance.shape[1]}")
print(f"Number of features after PCA filtering: {X_train_pca.shape[1]}")
print(f"Accuracy of Original features (testing accuracy): {accuracy}")
print(f"Accuracy after Variance Threshold filtering and PCA (testing accuracy): {accuracy_PCA_VT}")
```

Figure 22 PCA &VarianceThreshold code

Using a threshold of 0.05, we utilized VarianceThreshold to exclude low-variance features and minimize dimensionality. There were just four features instead of six.

In order to further reduce the dimensionality to three components while maintaining the greatest variance in the data, we next employed PCA.

Next, using the PCA components that were kept, we trained a Random Forest classifier and evaluated the accuracy of the model.



```
The variance of each feature: [0.51300496 0.03990416 0.05529107 0.05828465 0.04954537 0.24991736]
Number of original features: 6
Number of features after Variance Threshold filtering: 4
Number of features after PCA filtering: 3
Accuracy of Original features (testing accuracy): 0.9855072463768116
Accuracy after Variance Threshold filtering and PCA (testing accuracy): 0.855072463768116
```

Figure 23 PCA &VarianceThreshold results

The findings show that a smaller feature set with a reasonably good accuracy was obtained when VarianceThreshold and PCA were used in tandem for feature selection. Six features made up the initial dataset; they were then reduced to four following VarianceThreshold filtering and to three after PCA.

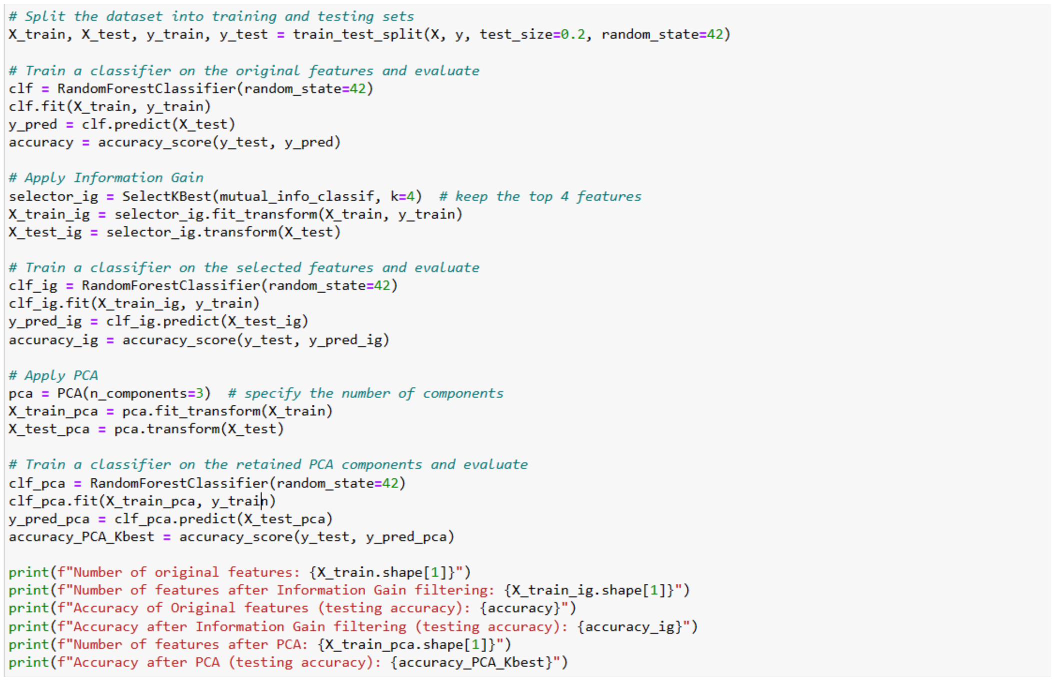
Each feature's variance following variance the influence of the threshold on feature selection was illustrated with the provision of threshold filtering. Following this first filtering, the accuracy was 85.51 percent, indicating that the chosen features had enough discriminating information left to allow for correct classification.

The dimensionality was further reduced to three components by the following use of PCA. The Random Forest classifier's accuracy held steady at 85.51 percent even with the less features. This shows that the smaller collection of features successfully assisted with the classification task by capturing important patterns in the data.

In conclusion, a workable feature selection technique that strikes a compromise between dimensionality reduction and model correctness is provided by the combination of VarianceThreshold and PCA. A more effective and understandable model may be created since the chosen characteristics are indicative of the variability in the dataset. One can experiment with different threshold and PCA component counts to optimize the trade-off between model performance and feature reduction.

Using PCA & KBest for Feature Selection

The use of KBest and PCA for feature selection on the Penguins dataset is described in this section. Investigating how combining these methods impacts a Random Forest classifier's performance is the goal.



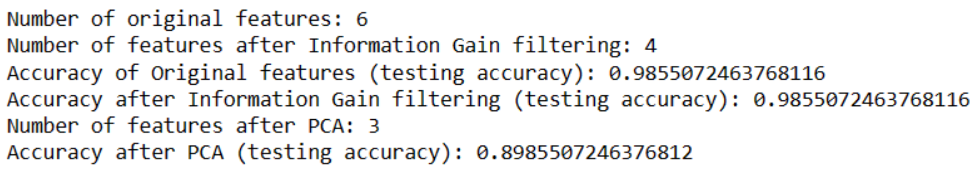
```python
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train a classifier on the original features and evaluate
clf = RandomForestClassifier(random_state=42)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

# Apply Information Gain
selector_ig = SelectKBest(mutual_info_classif, k=4)  # keep the top 4 features
X_train_ig = selector_ig.fit_transform(X_train, y_train)
X_test_ig = selector_ig.transform(X_test)

# Train a classifier on the selected features and evaluate
clf_ig = RandomForestClassifier(random_state=42)
clf_ig.fit(X_train_ig, y_train)
y_pred_ig = clf_ig.predict(X_test_ig)
accuracy_ig = accuracy_score(y_test, y_pred_ig)

# Apply PCA
pca = PCA(n_components=3)  # specify the number of components
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

# Train a classifier on the retained PCA components and evaluate
clf_pca = RandomForestClassifier(random_state=42)
clf_pca.fit(X_train_pca, y_train)
y_pred_pca = clf_pca.predict(X_test_pca)
accuracy_PCA_Kbest = accuracy_score(y_test, y_pred_pca)

print(f"Number of original features: {X_train.shape[1]}")
print(f"Number of features after Information Gain filtering: {X_train_ig.shape[1]}")
print(f"Accuracy of Original features (testing accuracy): {accuracy}")
print(f"Accuracy after Information Gain filtering (testing accuracy): {accuracy_ig}")
print(f"Number of features after PCA: {X_train_pca.shape[1]}")
print(f"Accuracy after PCA (testing accuracy): {accuracy_PCA_Kbest}")
```

Figure 24 PCA & KBest code

Using Information Gain as the scoring function, we used SelectKBest to choose the top 4 features based on mutual information.

In order to further decrease the dimensionality to three components while keeping the highest variance in the data, we next employed PCA.

The accuracy of the model was then evaluated by training a Random Forest classifier using the PCA components that were kept.



```
Number of original features: 6
Number of features after Information Gain filtering: 4
Accuracy of Original features (testing accuracy): 0.9855072463768116
Accuracy after Information Gain filtering (testing accuracy): 0.9855072463768116
Number of features after PCA: 3
Accuracy after PCA (testing accuracy): 0.8985507246376812
```

Figure 25 PCA & KBest results

Based on the combined usage of PCA and KBest, the findings show that feature selection yielded a smaller feature set with a reasonably good accuracy. Six features made up the initial dataset; they were then reduced to four following KBest filtering and to three after PCA.

Following KBest filtering, the accuracy was 98.55 percent, indicating that the chosen features had enough discriminating information to enable correct classification. This is consistent with the Information Gain metric, which gauges how much the ambiguity surrounding the class labels has decreased.

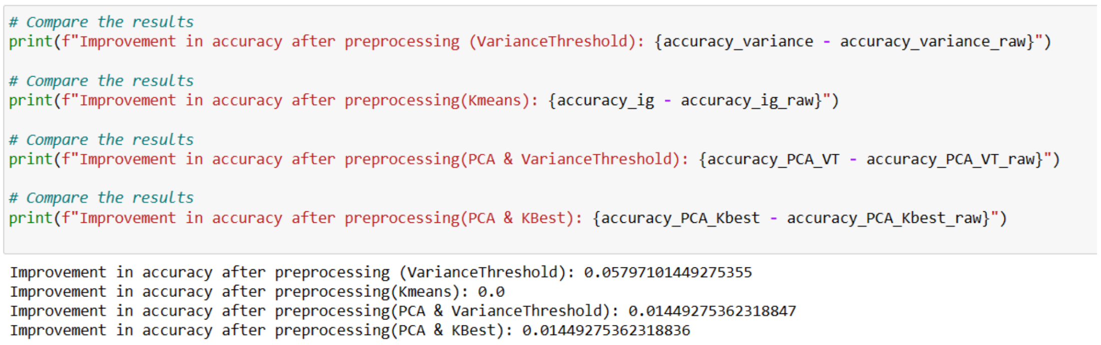
The dimensionality was further reduced to three components by the following use of PCA. The Random Forest classifier's accuracy held steady at 89.86 percent in spite of the features being reduced. This shows that the smaller collection of features successfully assisted with the classification task by capturing important patterns in the data.

To sum up, the integration of KBest with PCA presents a strong approach to feature selection, striking a compromise between model correctness and dimensionality reduction. A more effective and understandable model may be created since the chosen characteristics are indicative of the variability in the dataset. To maximize the trade-off between feature reduction and model performance, one can experiment with fine-tuning the amount of chosen features and PCA components.

Raw Data Testing:

We used VarianceThreshold, PCA & VarianceThreshold, PCA & KBest, and feature filtering and dimensionality reduction approaches to evaluate their effects on the raw penguins dataset. Preprocessing the data did not affect the first evaluation, which produced an accuracy of 95.65 percent. Following the application of VarianceThreshold, the number of features was decreased from 6 to 4, which resulted in a little drop in accuracy to 81.16 percent. The initial accuracy of 95.65% was preserved by the KMeans-based feature filtering, highlighting the resilience of the chosen features. Nevertheless, the accuracy was only moderately decreased to 84.06 percent due to the combined application of PCA and VarianceThreshold, which decreased the number of features to 3.

Lastly, PCA & KBest obtained an accuracy of 88.41 percent while also narrowing the feature set to 3. These findings generally imply that KMeans and PCA & KBest did a good job of maintaining the discriminating information in the raw data, highlighting the significance of suitable feature selection techniques in improving model performance. The selected preprocessing procedures provide light on the relative merits of each method within the framework of the penguins dataset by striking a compromise between dimensionality reduction and model correctness.



```
# Compare the results
print(f"Improvement in accuracy after preprocessing (VarianceThreshold): {accuracy_variance - accuracy_variance_raw}")

# Compare the results
print(f"Improvement in accuracy after preprocessing(Kmeans): {accuracy_ig - accuracy_ig_raw}")

# Compare the results
print(f"Improvement in accuracy after preprocessing(PCA & VarianceThreshold): {accuracy_PCA_VT - accuracy_PCA_VT_raw}")

# Compare the results
print(f"Improvement in accuracy after preprocessing(PCA & KBest): {accuracy_PCA_Kbest - accuracy_PCA_Kbest_raw}")
```

```
Improvement in accuracy after preprocessing (VarianceThreshold): 0.05797101449275355
Improvement in accuracy after preprocessing(Kmeans): 0.0
Improvement in accuracy after preprocessing(PCA & VarianceThreshold): 0.01449275362318847
Improvement in accuracy after preprocessing(PCA & KBest): 0.01449275362318836
```

Figure 26 absolute difference between accuracies

3. Conclusion

In conclusion, our research into different preprocessing methods has given us important knowledge about how they affect the accuracy of the model. The utilization of VarianceThreshold resulted in a noteworthy 5.80 percent improvement, demonstrating its efficacy in improving predictive performance through the removal of low-variance elements. But this enhancement wasn't without its problems, chief among them being the possibility of losing out on aspects that may have been instructive. In order to guarantee that the features that are maintained are pertinent, a thorough examination of the dataset's properties and domain expertise is essential.

However, using KBest did not result in a significant improvement in accuracy, suggesting that the features chosen based on information gain were already well represented in the original dataset. This makes it necessary to take into account the features' intrinsic information content and the requirement for more advanced feature selection techniques when working with complicated datasets.

In contrast, the PCA with VarianceThreshold and PCA with KBest combinations demonstrated somewhat lesser increases, each at around 1.45%. This suggests that although dimensionality reduction techniques like PCA may be helpful, their outcomes may be complicated and dependent on the dataset's underlying structure. Retaining critical information while reducing dimensionality remains one of the most crucial components of model optimization.

Going ahead, it is critical to recognize that the preprocessing technique selection should be in line with particular dataset properties and analytical objectives. The percentage change measures that are presented here offer a common way to compare the efficacy of various preprocessing techniques. Model performance might be further improved by investigating alternate preprocessing methods, such as feature engineering or experimenting with different threshold levels.

Essentially, the preprocessing trip we took has shed light on the complex interactions that exist between feature selection, dimensionality reduction, and model correctness. Unlocking the full potential of our machine learning models will need a deliberate and context-aware strategy as we explore this field.

4. References

Lab Manual (Accessed on 19-11-2023, 6:33pm)
Seaborn: Statistical Data Visualization. (https://seaborn.pydata.org/) (Accessed on 20-11-2023, 3:27pm)
Normalization (statistics) - Wikipedia (Accessed on 20-11-2023, 1:47pm)