Vysoká škola ekonomická

Data-X (4IT439)

Penguin dataset: The new Iris

Course: Data-X (4IT439)

Semester: SS 2022/2023

Authors: Simona Dohová (dohs00), Veronika Vacková (vacv06), Barbora Dobrovolná (dobb00), Karolína Krutinová (volk11)

Content

Problem definition

The goal of this work is to evaluate penguin species using machine learning methods and data science.

In this paper, the Penguin dataset is used: The new Iris, which will be used to create a model that will predict what species the penguin is classified as.

First, we will look at understanding and exploration of data using graphs and statistics, then we pre-process the data to get the best results later in the modelling part. We will include also hyperparameter tuning.

To evaluate the performance of our models we will include metrics such as accuracy, precision, recall and F1-score in our analysis.

Finally, we will compare the models based on their accuracy score, interpretability, complexity, scalability and assumptions to choose the best model.

Data understanding and data preparation

Dataset

The dataset was submitted by the school, but we can find a very similar dataset on kaggle, specifically at this address: https://www.kaggle.com/code/parulpandey/penguin-dataset-the-new-iris.

The dataset consists of 7 independent variables and 1 dependent variable, and 363 observations. Dependent variable, that is categorical, represents a species of penguin.

Table with variables:

Species	Penguin species (Chinstrap, Adélie, or Gentoo)	object
Island	Island name (Dream, Torgersen, or Biscoe)	object
Bill_length_mm	bill length (mm)	float64
Bill_depth_mm	bill depth (mm)	float64
Flipper_length_mm	flipper length (mm)	float64
Body_mass_g	body mass (g)	float64
sex	penguin sex (male, female, nan)	object
year	Year	int

After taking a good look at the data structure, we transformed the categorical variables from object (or integer in the case of *year*) to category format.

Missing values and duplicates

When checking for missing values (see *Figure 1 Missing values*), we observed that 5 of 8 columns of the dataset contained NaNs. It was decided to remove observations that were missing more than 5 out of 7 explanatory variables including. Deleting the rows was chosen over imputation as there was only 5 such observations. Rows with 1 missing value will be imputed after splitting the data set into the training and testing part (see *Data for machine learning*).

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex	year	nans
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN	2007	5
287	Gentoo	Biscoe	NaN	NaN	NaN	NaN	NaN	2009	5
337	Chinstrap	Dream	NaN	NaN	NaN	NaN	NaN	2009	5
346	Chinstrap	Dream	NaN	NaN	NaN	NaN	NaN	2009	5
362	Chinstrap	Dream	NaN	NaN	NaN	NaN	NaN	2009	5
8	Adelie	Torgersen	34.10	18.10	193.00	3475.00	NaN	2007	1
9	Adelie	Torgersen	42.00	20.20	190.00	4250.00	NaN	2007	1
10	Adelie	Torgersen	37.80	17.10	186.00	3300.00	NaN	2007	1
11	Adelie	Torgersen	37.80	17.30	180.00	3700.00	NaN	2007	1
37	Adelie	Biscoe	37.50	18.60	NaN	3150.00	female	2007	1

Figure 1 Missing values

We also decided to delete the 11 duplicated values found in the data.

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex	year
0	Adelie	Torgersen	39.10	18.70	181.00	3750.00	male	2007
16	Adelie	Torgersen	39.10	18.70	181.00	3750.00	male	2007
19	Adelie	Torgersen	39.10	18.70	181.00	3750.00	male	2007

Figure 2 Duplicates

After making these changes, we were left with 347 observations in the dataset.

Descriptive statistics

This part shows the descriptive statistics divided into numeric and categorical variables.

	bill_length_mm	*bill_depth_mm*	*flipper_length_mm*	*body_mass_g*
count	347.0	347.0	346.0	347.0
mode	41.1	17.0	190.0	3800.0
median	44.5	17.3	197.0	4000.0
mean	43.94	17.17	200.81	4191.35
std	5.46	1.97	14.02	801.49
min	32.1	13.1	172.0	2700.0
25%	39.25	15.6	190.0	3550.0
50%	44.5	17.3	197.0	4000.0
75%	48.5	18.7	213.0	4750.0
max	59.6	21.5	231.0	6300.0

Figure 3 Descriptive statistics of numerical variables

	species	*island*	*sex*	*year*
Count	347	347	338	347
Unique	3	3	2	3
top	Adelie	Biscoe	male	2009
freq	153	169	170	119

Figure 4 Descriptive statistics of categorical variables

Data visualization

In every data analysis, it is important to visualize the data in order to gain more knowledge before preparing classification models.

Categorical variables

We first visualized the four categorical variables – *species, island, year, sex*.

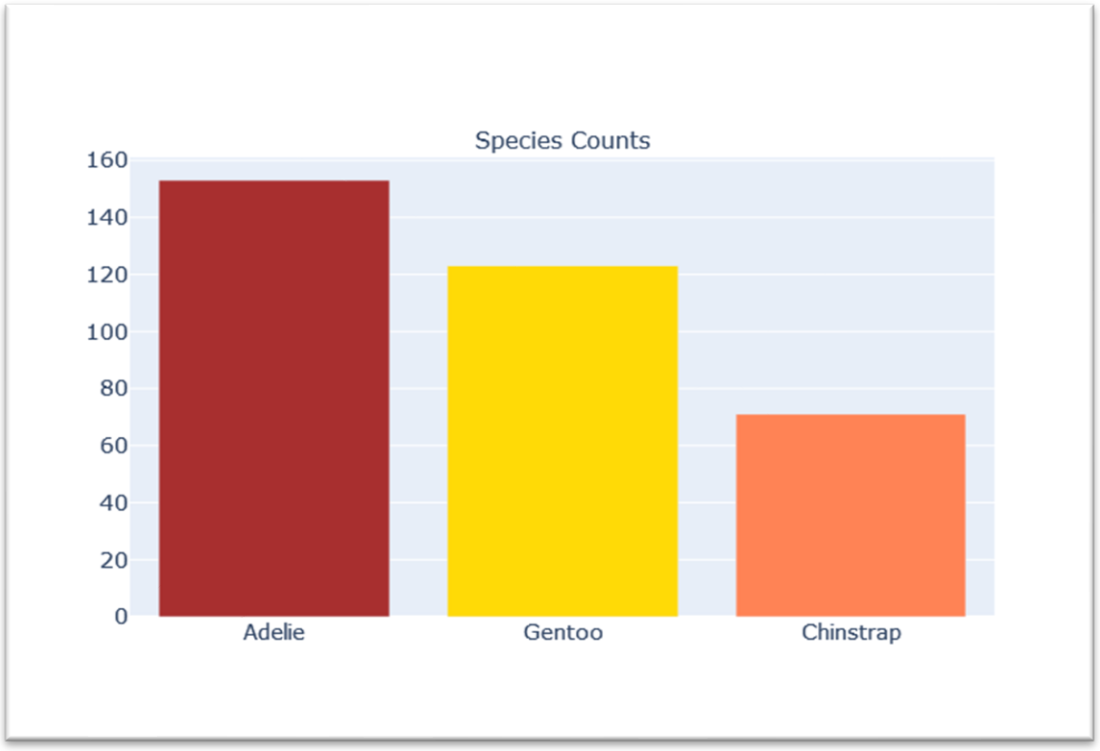

Figure 5 Species distribution

As it is visible from *Figure 5 Species distribution*, the distribution of the target variable seems to be slightly imbalanced. The Adelie is the majority specie with the count of 153 observations, followed by Gentoo (123 observations) and the third specie causing the major imbalance is Chinstrap specie with 71 observations count.

Cross-validation will be used to overcome the imbalance.

Depending on the performance of the classification models, it will be decided if an upsampling is necessary.

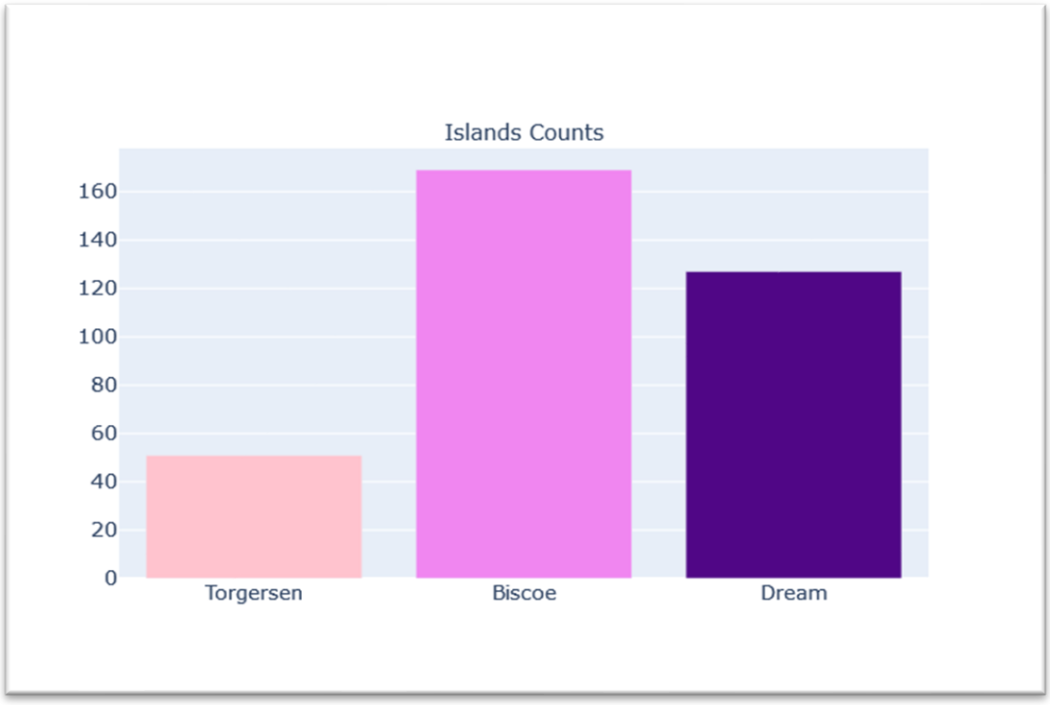

Figure 6 Islands distribution

We also looked at the distribution of the islands and the penguin drudges on them and found that the distribution was very uneven. In the dataset there are 51 observations from the Torgersen island, 169 observations from the Biscoe island and 127 observations from the Dream island.

After combining the information from bar chart for species and islands we obtained a graph showing species distribution per islands (see *Figure 7 Species distribution per islands*). It is visible that:

1. On the Torgersen island only the Adelie specie can be found,
2. Gentoo specie lives only on Biscoe island, and
3. Chinstrap species lives only on Dream island.

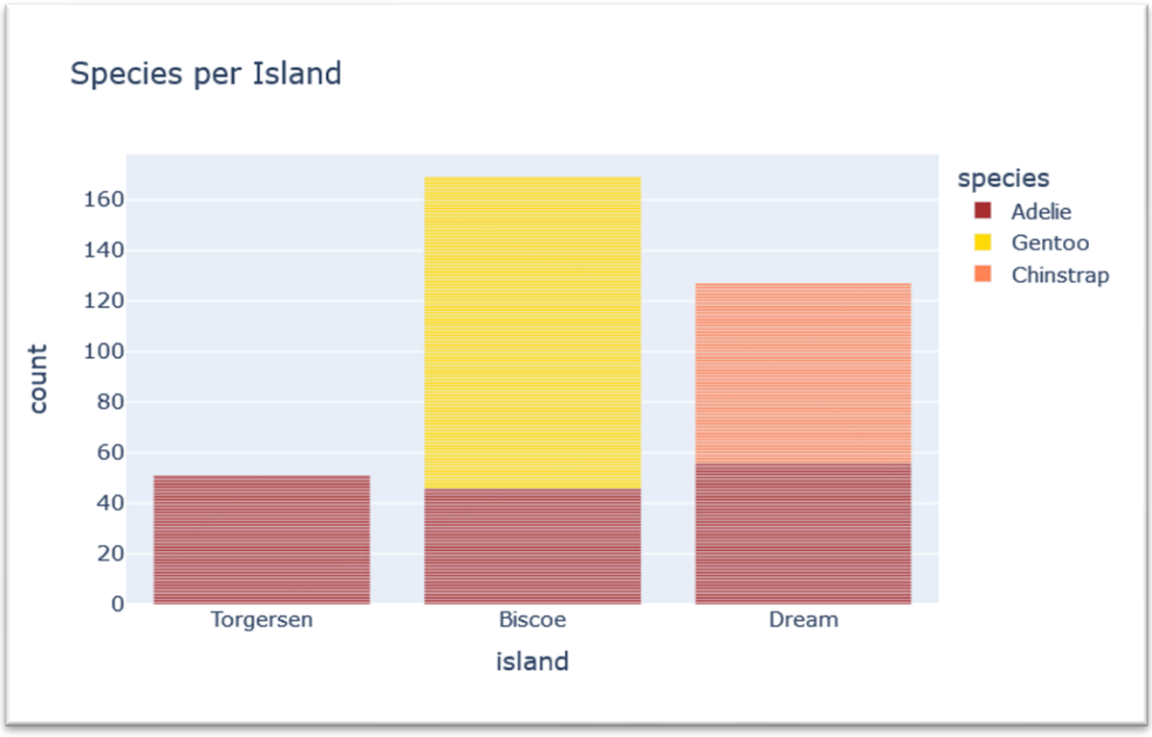

Figure 7 Species distribution per islands

The next variable needed to be plotted in a bar chart was the column *year*. Number of observations each year is balanced – in the data set there are 114 rows for the year 2007, the same amount for the year 2008, and slightly more (119) for 2009. This fact led us to future deletion of the column *year*.

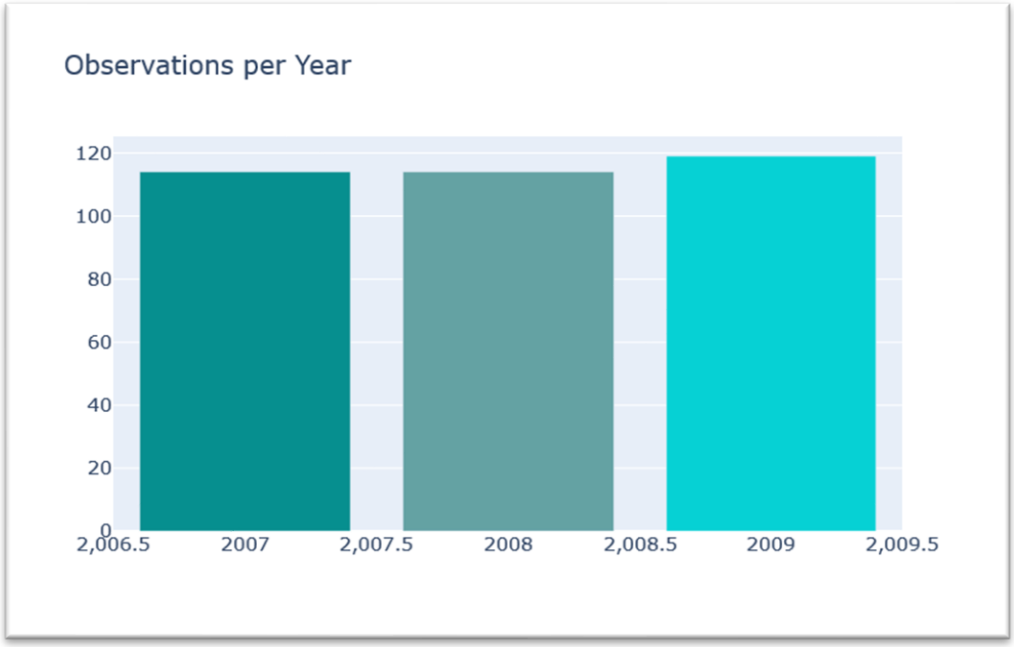

Figure 8 Years distribution

We visualized also the sex distribution (see *Figure 9 Sex distribution*). 170 observations contained male penguins and 168 females. There were also 9 rows with missing values that will be imputed before building the classification models.

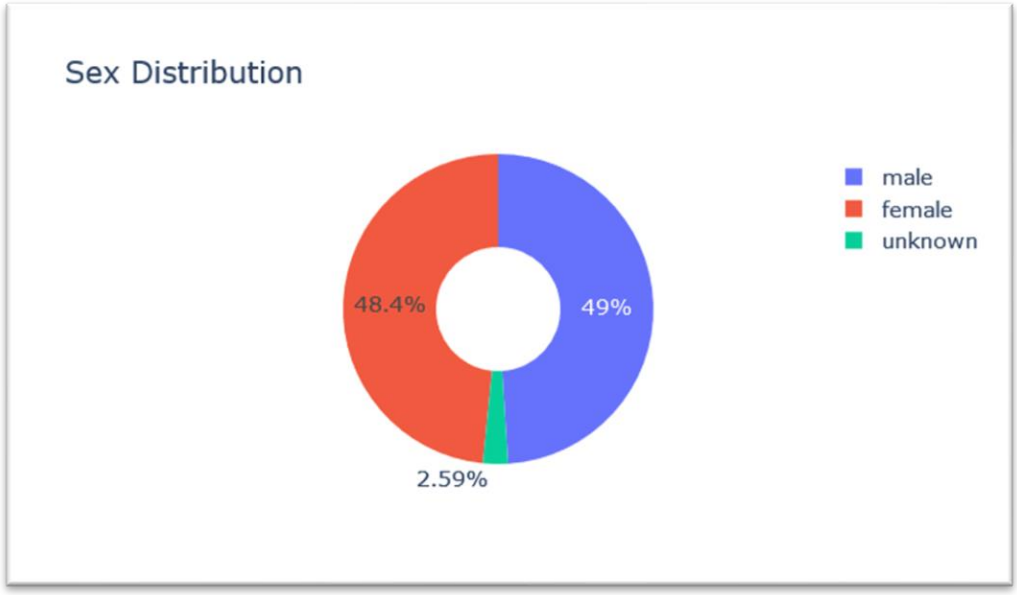

Figure 9 Sex distribution

In order to gain deeper insight, we added more detail in the sex distribution graphs.

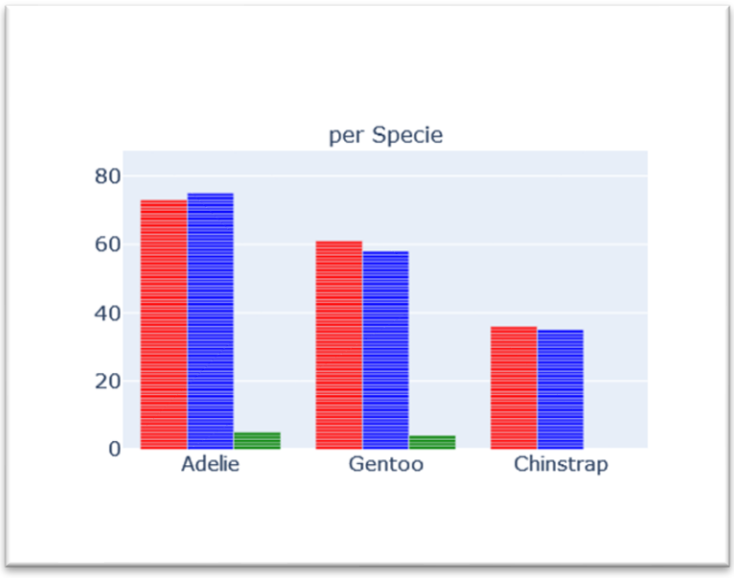

Figure 10 Sex distribution per specie

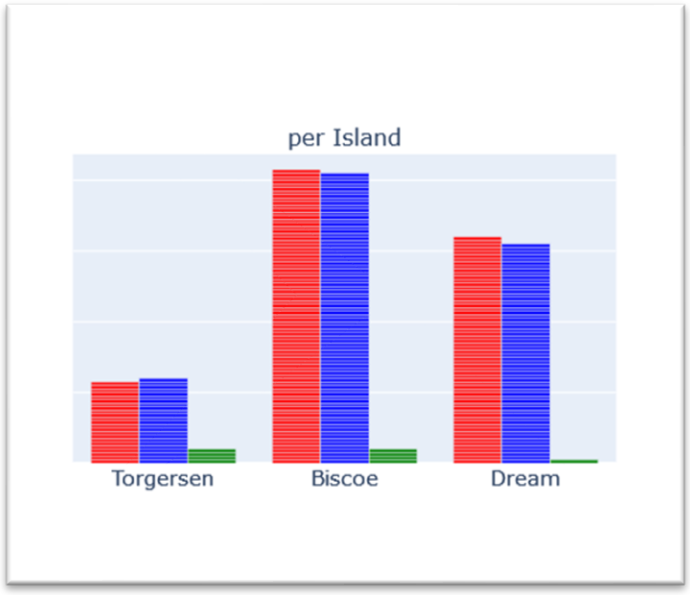

Figure 11 Sex distribution per island

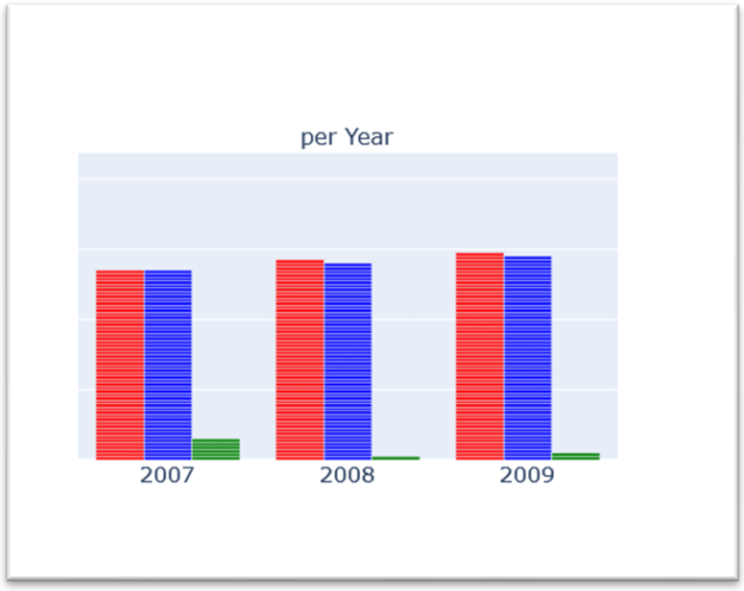

Figure 12 Sex distribution per year

All the graphs above show that sex variable is balanced throughout the dataset as a whole, per species, per island, as well as per year. This observation was another argument for the future deletion of the column year.

Numerical variables

After we visualized all the categorical variables, we prepared histograms and violin plots for the four numerical variables – *bill_length_mm*, *bill_depth_mm*, *flipper_length_mm*, *body_mass_g*.

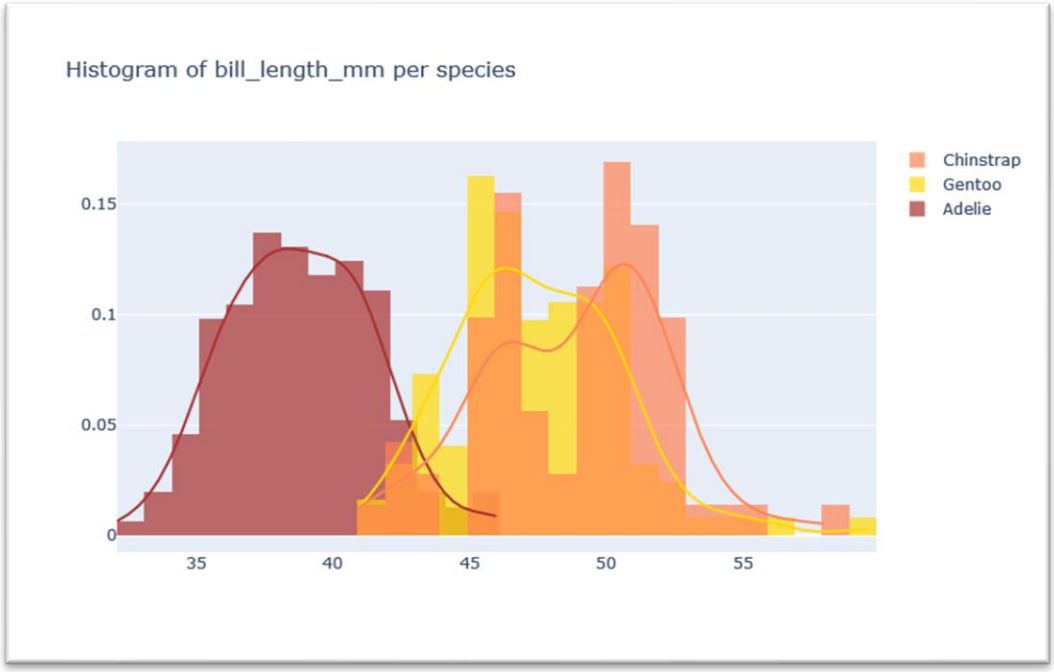

Figure 13 Histogram of bill_length_mm per species

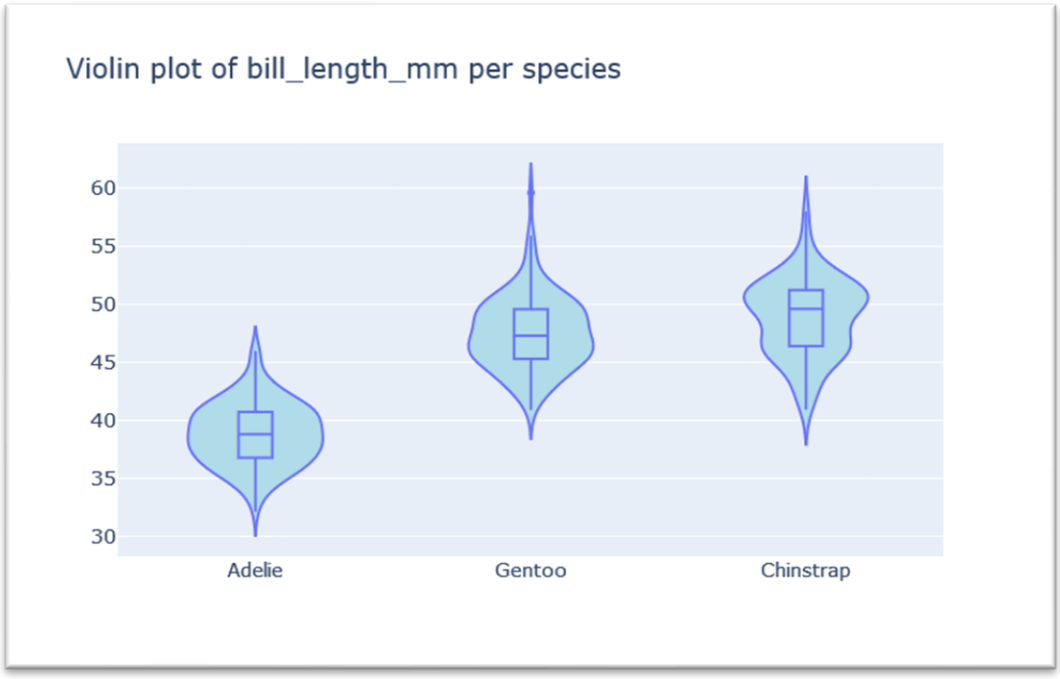

Figure 14 Violin plot of bill_length_mm per species

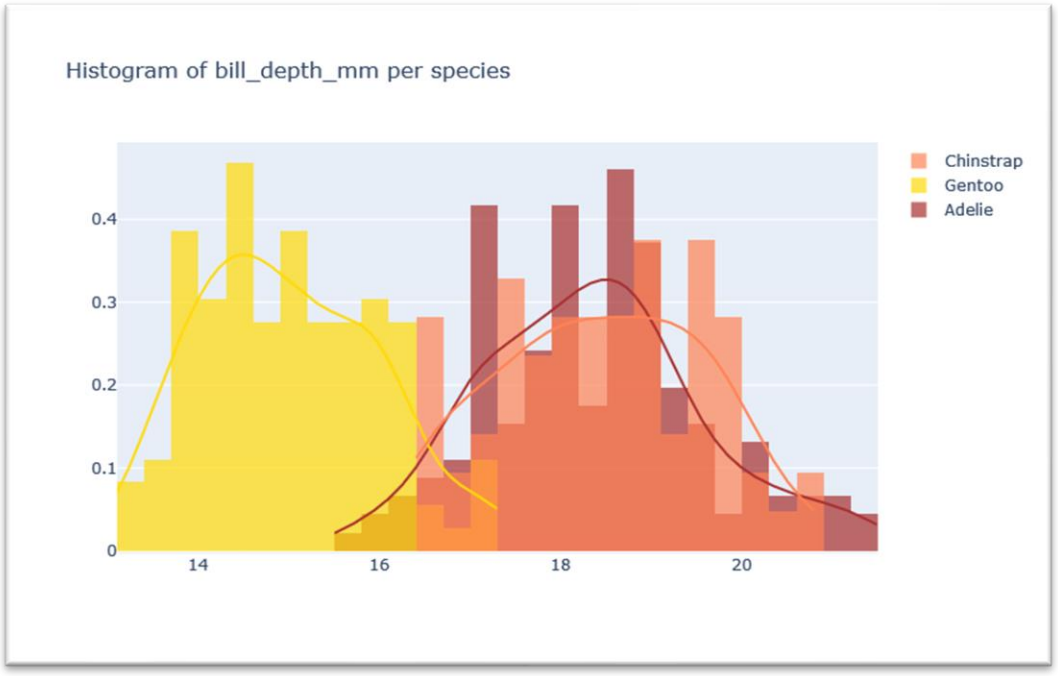

Figure 15 Histogram of bill_depth_mm per species

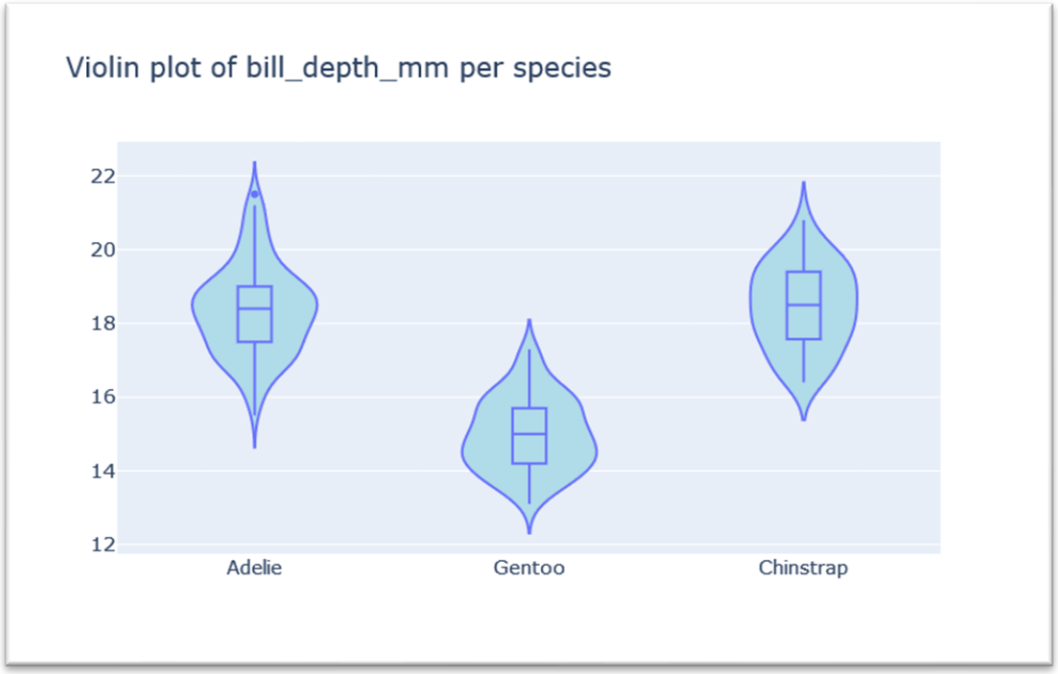

Figure 16 Violin plot of bill_depth_mm per species

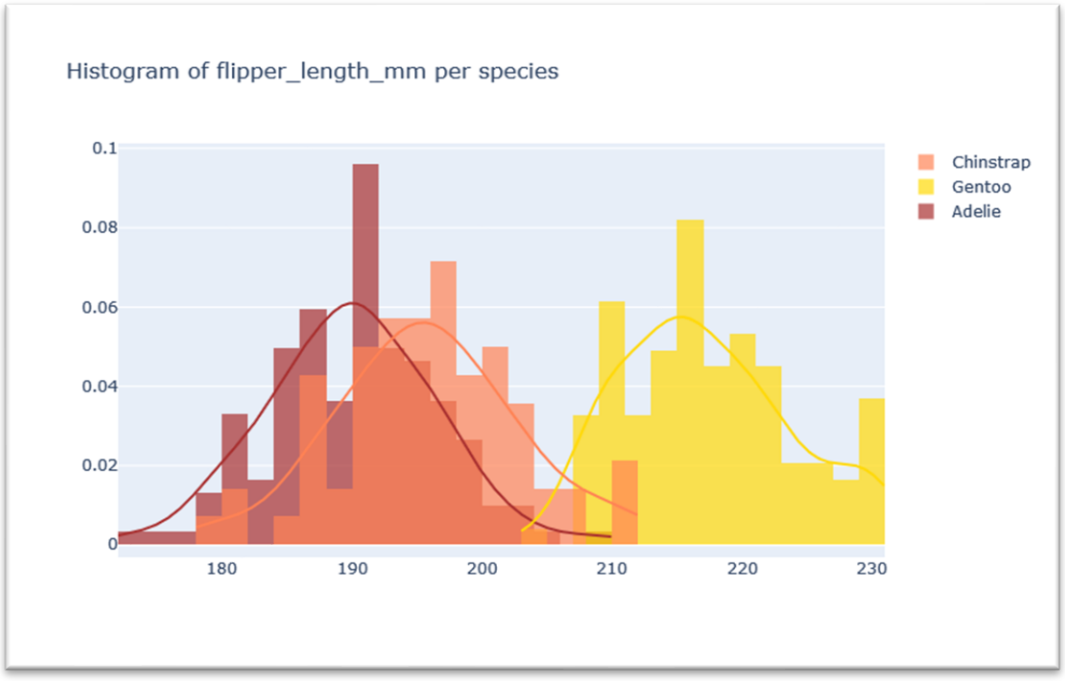

Figure 17 Histogram of flipper_length_mm per species

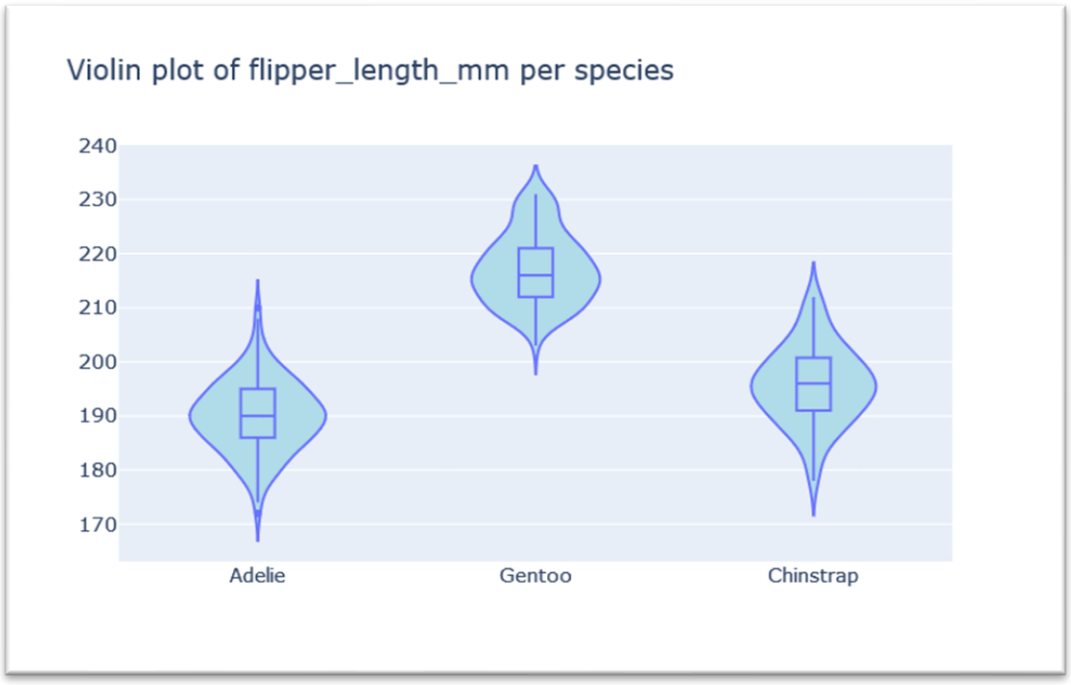

Figure 18 Violin plot of flipper_length_mm per species

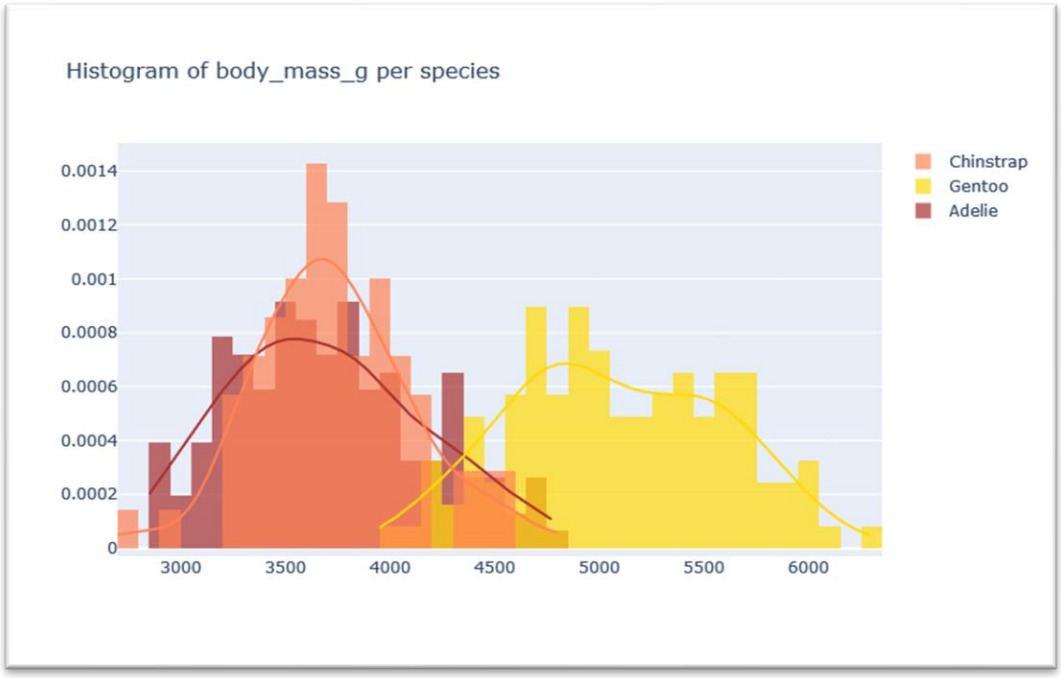

Figure 19 Histogram of body_mass_g per species

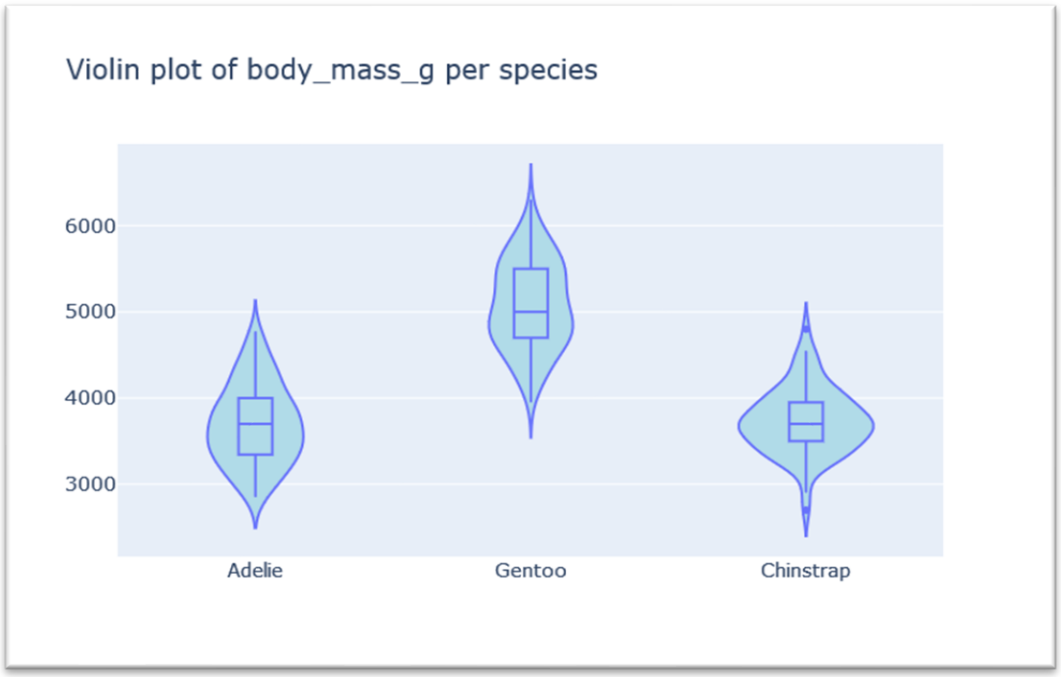

Figure 20 Violin plot of body_mass_g per species

Analysis of the above graphs brought us the following information:

1. The Adelie specie seems to have shorter bills on average compared to other species.
2. The Gentoo specie seems to have less deep bills on average compared to other species.

3. The Gentoo specie seems to have longer flippers on average compared to other species.
4. The Gentoo specie seems to have bigger body mass on average compared to other species.

We also plotted correlograms for all the pairs of numerical variables.

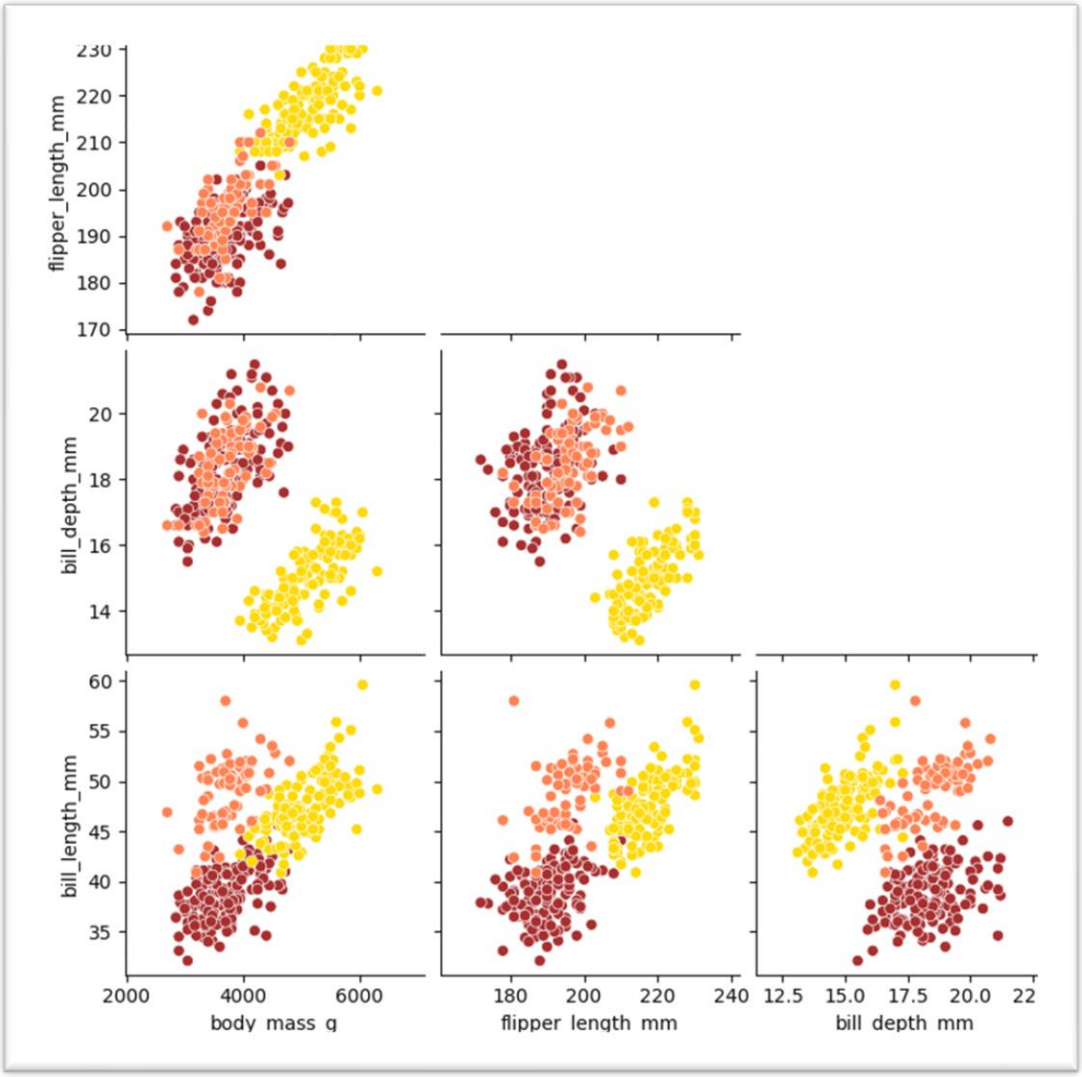

Figure 21 Pairplot

Correlation

We checked for correlation between independent numerical variables and also for correlation between all variables. The correlation coefficients are calculated using the Spearman correlation method, which "measures the strength and direction of association between two ranked variables. It basically gives the measure of monotonicity of the relation between two variables" (simplilearn.com) "rather than the

strength and direction of the linear relationship between your two variables, which is what Pearson's correlation determines" (statistics.laerd.com).

After checking the correlation, it was clear that we would have to remove some variables.

In *Figure 22 Correlation between independent numerical variables* and *Figure 21 Pairplot*, we can see that the columns *flipper_length_mm* and *body_mass_g* are strongly positively correlated with the meaning: the higher the body mass, the longer the flipper.

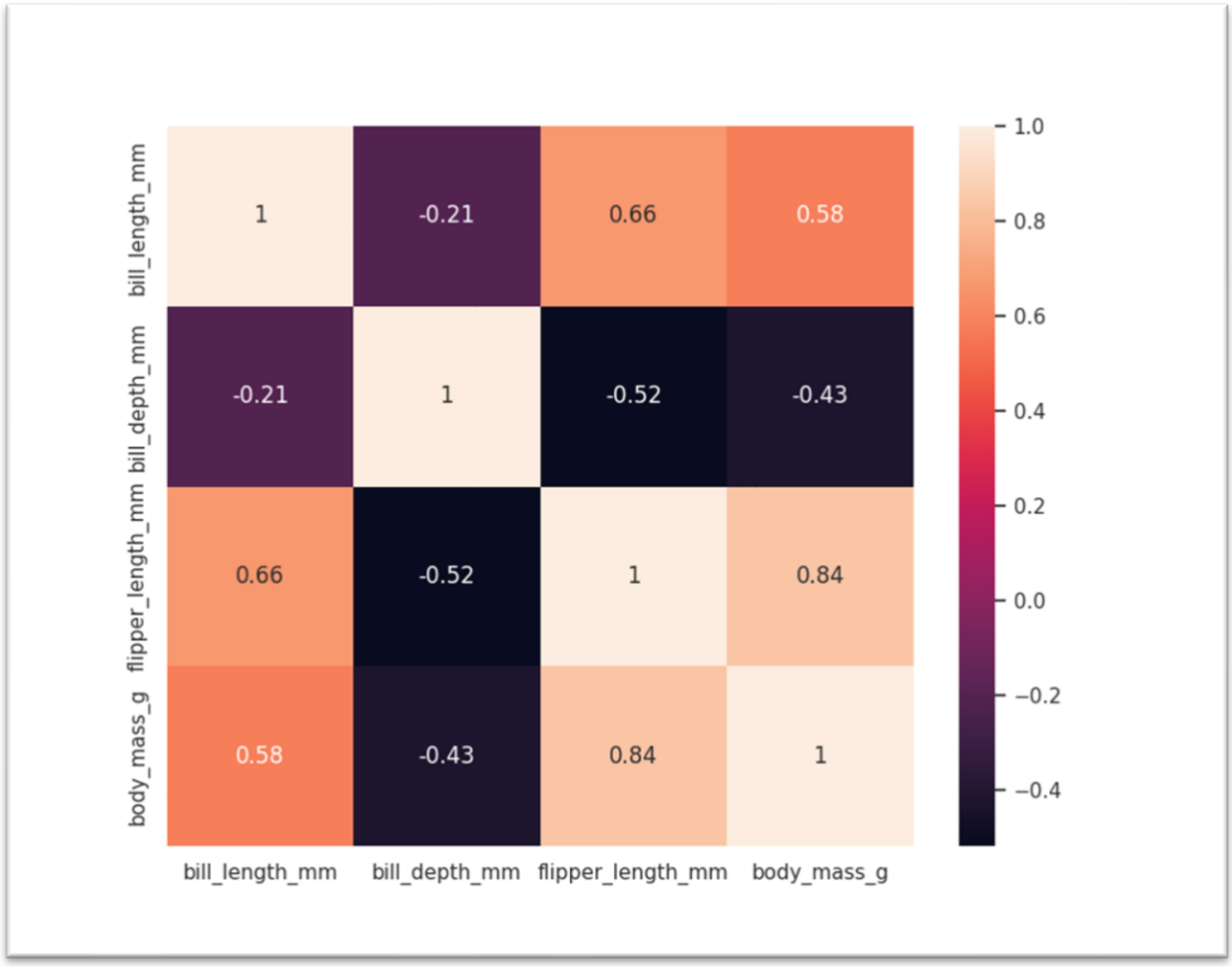

Figure 22 Correlation between independent numerical variables

We decided to delete the column *flipper_length_mm* because of the high correlation with two other features *(body_mass_g* and *bill_lenght_m*) and an occurance of a missing value.

Modeling

Data for machine learning

Preparing the dataset that we will use to train the machine learning models is key. Quality pre-processing is reflected in the accuracy of the models used.

The process of preparing the dataset for this work was as follows:

1. *Encoding the categorical variables*

   Creating columns with binary values for seamless work with machine learning models. For gender, the drop_first argument was used, which removes the first unique value of the column (e.g. the female value is removed and there is only a male column).

2. *Dealing with sex NaN*

   Added NaN (aka sex_unknown) to the dataset to avoid data leakage and correctly handle missing data in the following steps.

3. *Splitting the data into train and test data*

   Division of data into training and testing part in a ratio of 80:20. No validation set was used because we use cross-validation.

4. *Data stratification*

   Due to the uneven representation of the categories, a stratification in the ratio of 3:3:3 was used. The function takes the input data, the names of the columns we want to stratify by a list of values for those columns, the ratio of sample sizes for each group, and a random number to initialize the pseudorandom number generator and returns a new dataframe.

   Before:

```
Count of each species in the training data: Counter({'Adelie': 125, 'Gentoo': 96, 'Chinstrap': 56})
```

   After:

```
Count of each species in the stratified training data: Counter({'Gentoo': 93, 'Adelie': 92, 'Chinstrap': 92})
```

Note: In the end it was not used in the model, the results were excellent even without this step.

5. *Scaling data*

In this step, columns containing numeric values were normalized to reduce the variance of the values to a range of 0 to 1. Normalization is often used when preparing data for machine learning, as it can improve model performance by helping to minimize the influence of extreme values and ensure that each column of data has approximately the same effect on the predicted value.

6. *Missing values in sex_column*

We use KNN Imputer to solve missing values. We assume that NaNs are missing at random (x_train_imputed). Using the nearest neighbors method (n_neighbors = 5), missing values are filled.

7. *Ravel() function*

Converting a two-dimensional array to one-dimensional using the ravel() function. This was done to meet the requirements of machine learning models that often require this dimension to function properly.

Functions

def evaluate_model

This code is a function that evaluates the machine learning model and calculates the accuracy and F1-score. The function takes as input a model, model name, X_test and y_test test data, as well as a tuning parameter that specifies whether hyperparameter optimization has been performed.

After the model is trained on the training data and the results on the test data are predicted, the accuracy, precision, recall and F1-score values are calculated. The function prints these results to the screen and returns the predicted results y_pred. If hyperparameter optimization was performed, the function also lists the best-found model parameters.

def get_features_importance

This code contains a function to calculate the importance of features for a given machine learning model and return the top 5 most important features. The function takes as input a dataframe (df) containing the flags and the machine learning model.

The function first creates a copy of the dataframe (df_copy) and then calculates the feature importance using the feature_importances_ method from the model. If hyperparameter optimization has been done, then the function will use best_estimator_ to find the importance of the flags. The results are stored in a dictionary and subsequently in a dataframe (df_importances). The function returns a dataframe sorted by the importance of the symptoms and creates an interactive bar chart using the Plotly library that displays the most important top 5 symptoms and their importance.

def plot_learning_curve

The plot_learning_curve() function is used to plot the learning curve for a classifier. It takes as input the classifier to be evaluated, the training input samples, the target values (class labels), and optional parameters such as the evaluation metric to use, the sizes of the training set to use, and the number of cross-validation folds to use.

The function uses the learning_curve() function from scikit-learn to calculate the learning curve with cross-validation. It then calculates the mean and standard deviation of the training and testing scores and plots the learning curve using Matplotlib. The training and validation scores are plotted against the number of training examples used, with shaded regions indicating the standard deviation of the scores.

The resulting plot helps to visualize how well the classifier is learning from the training data, and whether it is overfitting or underfitting.

Models

Random Forest

Random Forest (RF) is a machine learning algorithm that is used for supervised tasks such as classification and regression. It operates using an ensemble learning method that involves building multiple decision trees during the training phase, which are later combined to make predictions. The algorithm creates a forest of decision trees, where each tree is trained on a randomly selected subset of the training data and a randomly selected subset of the features. This approach introduces randomness that helps to minimize overfitting and enhance the overall performance of the model.

Hyper-parameters tuning

Hyperparameters for RF were tuned using Grid Search Cross Validation. Parameters tested include bootstrap, max_depth, max_features, min_samples_leaf, min_samples_split, and n_estimators.

Best parameters: bootstrap = True, max_depth = 80, max_features = 2, min_samples_leaf = 3, min_samples_split = 8 and n_estimators = 200.

The predictions on the test data were evaluated through the 'evaluate_model' function. The model achieved very high accuracy (99%) on the test data, which means that it can classify new unknown samples with high confidence. Similarly, precision, recall and F1-score were also very high, indicating that the model has the ability to accurately identify positive and negative classes.

```
Fitting 3 folds for each of 1 candidates, totalling 3 fits

********** Random Forest Classifier **********

Best params found: {'bootstrap': True, 'max_depth': 80, 'max_features': 2, 'min_samples_leaf': 3, 'min_samples_split': 8, 'n_estimators': 200}

Accuracy: 99 %
Precision: 98 %
Recall: 99 %
F1-score: 98 %
```

Figure 23 Evaluation of Random Forest Classifier

High accuracy is graphically represented in the confusion matrix. There was only one misclassified specimen (Chinstrap mistaken for an Adelie).

A learning curve was used to determine overlearning/underlearning. In this case, the curves of the training and validation set do not differ and are close to each other, while the success rate of the model is high. The model can generalize to the new data and is well trained.

Figure 24 Learning curve – Random Forest

Main features for Random Forest are bill_length_mm and bill_depth_mm.

XGB Classifier

eXtreme Gradient Boosting Classifier is a classification algorithm using the gradient boosting framework. Gradient boosting is a machine learning technique that combines weak learning models such as decision trees to create a stronger predictive model. It uses techniques such as regularization, weighting, and thresholding to help minimize overfitting and improve model generalization. XGBoost is a popular choice for machine learning competitions and is used in industry for various tasks such as image recognition, fraud detection or recommendation personalization.

The model only works with numerical values, y_train and y_test were therefore converted via LabelEncoder to numerical values corresponding to the three target categories.

Hyper-parameters tuning

Hyperparameters for XGB were tuned using Randomized Search Cross Validation. Parameters tested include colsample_bytree, learning_rate, min_child_weight, gamma, subsample, colsample_bytree, max_depth.

For this model, we used Randomized Search, as it is more efficient than Grid Search, which saves time in the case of computationally demanding XBG.

Best parameters: colsample_bytree: 0.8, gamma: 0.5, learning_rate: 0.13083364655022833, max_depth: 9, min_child_weight: 5, n_estimators: 876, subsample: 1.0.

Like the previous model, this one also achieved 99% in all monitored metrics.

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits

********** XGB Classifier **********

Best params found: {'colsample_bytree': 0.8, 'gamma': 0.5, 'learning_rate': 0.13083364655022833, 'max_depth': 9, 'min_child_weight': 5, 'n_estimators': 876, 'subsample': 1.0}

Accuracy: 99 %
Precision: 98 %
Recall: 99 %
F1-score: 98 %
```

Figure 25 Evaluation of XGB Classifier

Again, high accuracy is graphically represented in the confusion matrix, where the only misclassified value was Chinstrap mistaken for an Adelie.

The learning curve does not look as smooth as the other models, but the increasing number of observations again leads to the conclusion that the model is neither overtrained nor undertrained and explains its high accuracy.

Figure 26 Learning curve – XGB Classifier

Main features for XGB Classifier are island_Dream and bill_depth_mm.

Decision Tree

Decision Trees is a machine learning technique that uses a tree-like model to make predictions. It works by splitting the data into smaller and smaller subsets based on different features, eventually leading to a decision or prediction at the end of each branch. Decision trees can handle both categorical and numerical data, as well as missing values and feature interactions. They are interpretable, flexible, and can capture nonlinear relationships between the features and the target variable.

Hyper-parameters tuning

Hyperparameters for Decision Tree were tuned using Grid Search Cross Validation. Parameters tested include criterion, max_depth, min_samples_split, min_samples_leaf, max_features.

Best parameters: criterion: entropy, max_depth: 10, max_features: sqrt, min_samples_leaf: 1, min_samples_split: 2.

After achieving a 100% score in all monitored metrics by adjusting the hyperparameters of the model, we plotted a learning curve to ensure that the model was neither overfitted nor underfitted, which confirmed that the model's performance was optimal without any overfitting or underfitting issues.

Figure 27 Learning curve – Decision Tree

Main features for Decision Tree are bill_length_mm and bill_depth_mm.

Logistic regression

Logistic regression is a machine learning algorithm, where the goal is to predict a binary or categorical target variable based on one or more predictor variables. It works by fitting a logistic function to the data, which transforms the output of a linear regression model into a probability distribution over the possible outcomes. Logistic regression models can handle both categorical and numerical data, as well as missing values and feature interactions. Unlike decision trees, logistic regression models are generally not as flexible in capturing nonlinear relationships between the features and the target variable.

Hyper-parameters tuning

Hyperparameters for Logistic Regression were tuned using Grid Search Cross Validation. Parameters tested include Penalty, C, Solver. The **penalty** parameter is used to control overfitting by adding a regularization term to the loss function. It specifies the norm used in the penalization, and can take values of "l1", "l2", "elasticnet", or "none".

C is a hyperparameter that controls the strength of regularization, and is used to balance the trade-off between overfitting and underfitting. It determines the strength of the regularization term, with smaller values of C corresponding to stronger regularization and larger values of C corresponding to weaker regularization. A higher value of C means that the model will focus on minimizing the training error, whereas a lower value of C means that the model will focus on minimizing the regularization term.

The **solver** parameter specifies the algorithm used for optimization in logistic regression. It can take values of "newton-cg", "lbfgs", "liblinear", "sag", or "saga". "liblinear" is a good choice for small datasets, while "sag" and "saga" are optimized for large datasets. "newton-cg" and "lbfgs" are suitable

for small to medium-sized datasets with multi-class classification problems. Therefore, only newton-cg", "lbfgs", "liblinear" were used for our hyper-parametr tuning.

When performing hyperparameter tuning, we encounter warning messages indicating that not all hyperparameters were fitted. To avoid being distracted by these warnings during the tuning process, we ignored all these warnings, using the "warnings" module.

Best parameters: C: 1.0, penalty: l1, solver: liblinear.

The logistic regression model also achieved 100% accuracy in all evaluation metrics, like the previous model. However, we needed to assess whether the model was overfitted or underfitted. After examining the plot, it became evident that the model was neither overfitted nor underfitted.

Figure 28 Learning curve – Logistic Regression

Features and beta- coefficients

Beta coefficients describe how much each feature contributes to predicting the target variable (penguin species). A positive beta coefficient suggests that an increase in the feature's value is linked to an increase in the probability of that particular penguin species.

The plot of the beta coefficients shows that **bill length** and **bill depth** are the most important features for predicting the species of penguins. For identifying Gentoo penguins, body mass and bill depth is a critical feature, while the island location is also significant in predicting the species.

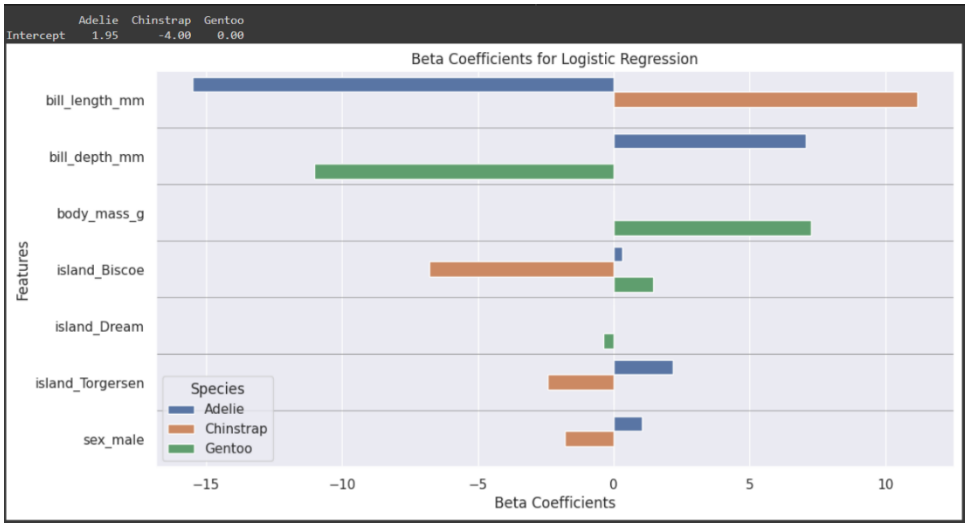

Figure 29 Features and beta-coefficients

Evaluation

To evaluate the effectiveness of each model, we created a Confusion Matrix for every model and below each model we displayed four evaluation metrics:

- Accuracy
- Precision
- Recall (sensitivity)
- F1-score

These metrics are calculated for each class separately and then averaged, using macro-averaging method.

The Logistic Regression and Decision Tree both performed exceptionally well, achieving accuracy scores of 100%. Their precision, recall, and F1-score metrics were also excellent, indicating that these models did not make any false predictions and were highly effective at identifying penguin species. While the Random Forest and XGBoost models had slightly lower metrics, they still performed reasonably well.

	model	accuracy	precision	recall	f1
0	Decision Tree Classifier	100.00 %	100.00 %	100.00 %	100.00 %
1	Random Forest Classifier	99.00 %	98.00 %	99.00 %	98.00 %
2	XGB Classifier	99.00 %	98.00 %	99.00 %	98.00 %
3	Logistic Regression	100.00 %	100.00 %	100.00 %	100.00 %

Figure 30 Evaluation metrics – Results

Conclusion

Based on the evaluation results, it appears that there are two models that score 100% on all metrics. In order to choose the most suitable model for our real-time case, we need to consider the advantages and disadvantages of each model. Therefore, we will carefully evaluate each model's performance and characteristics, taking into account factors such as interpretability, model complexity, scalability and assuptions.

Interpretability: Decision trees are often easier to interpret than logistic regression models. They provide a visual representation of the decision-making process. However, logistic regression models can be more interpretable when you only have a few predictors. There are only a few predictors in our analysis. Therefore, the interpretability of both models is very similar.

Model complexity: Decision trees can easily become complex and overfit to the training data, which can result in poor performance on new data. Logistic regression, on the other hand, has a simpler structure that is less prone to overfitting.

Scalability: Logistic regression is generally faster and more efficient than decision trees, particularly when dealing with large datasets.

Assumptions: Logistic regression requires a linear connection between input features and the outcome's probability, while decision trees do not have any specific expectations about the relationship between inputs and outputs.

	0	1	2	3	4	5
Factors	Accuracy	Interpretability	Model complexity	Scalability	Assumptions	Total Count
Logistic Regression	+	+	++	++	+	7
Decision Trees	+	+	+	+	++	6

Figure 31 Pros and cons of Logistic Regression and Decision Tree

Both Logistic Regression and Decision Tree models are suitable for our case study. Because of these reasons we have decided to use **Logistic Regression** before Decision Tree. Mainly for its good interpretability, speed, efficiency and the fact that the Logistic Regression is less prone to overfitting.

Development Environment Characteristics

The most important packages utilized during the development of the classification models are:

- NumPy and Pandas for operations on data frames,
- Seaborn, Matplotlib and Plotly for data visualisation,
- Scikit-learn and XGBoost for modeling,
- Joblib, h5py and Tensorflow for saving and loading models.

This project uses python 3.9.13 and consists of the following parts:

- *README.md* file gives a brief description of the project and also contains instructions on how to run the project,
- *data* folder contains the dataset we worked with,
- N*otebooks* contains python code, which is the output of this project and contains all dataset editing and modeling,
- *Requirements.txt* contains detailed information about all the packages and their respective versions.

List of Figures