The problem statement we have in our hand is to classify the species of the penguin, given the different predictor variables. We have the dataset available in hand which has the following predictors and targets.

Predictors:

* Sex
* Culmen Length (mm)
* Culmen Depth (mm)
* Flipper Length (mm)
* Body Mass (g)
* Island

Target - Species:

* Adelie
* Chinstrap
* Gentoo

Problem Type:

Supervised Learning:

Supervised learning is a type of training the system by providing labelled inputs. While we feed the system the input features, we also say the expected output. In this case, we are training the system with predictors (independant variables) along with the target (dependant variable).

Classification:

Classification is a subset of supervised learning where the output or dependant variable is discrete. We have the 'Species' feature as the target which is discrete.

Hence this is a classification problem. However this dataset can also be used to carry out clustering tasks as well. We are not covering clustering in this notebook.

Importing Libraries and Datasets

The first thing to do is to import the required libraries. I've listed down the libraies we are going to use in this notebook.

The libraries which are used in this Kernel are,

* Numpy - Matrices and Mathematical Functions
* Pandas - Data Manipulation and Analysis
* Matplotlib - Simple Visualization
* Seaborn - More Sophisticated Visualizations
* Scikit Learn - Machine Learning Algorithms and Evaluation Metrics

In [1]:

import numpy as np *# linear algebra*

import pandas as pd *# data processing, CSV file I/O (e.g. pd.read\_csv)*

import matplotlib.pyplot as plt *#simple data visualization*

%matplotlib inline

import seaborn as sns *#some advanced data visualizations*

import warnings

warnings.filterwarnings('ignore') *# to get rid of warnings*

plt.style.use('seaborn-white') *#defining desired style of viz*

import os

for dirname, \_, filenames **in** os.walk('/kaggle/input'):

for filename **in** filenames:

print(os.path.join(dirname, filename))

/kaggle/input/palmer-archipelago-antarctica-penguin-data/penguins\_size.csv

/kaggle/input/palmer-archipelago-antarctica-penguin-data/penguins\_lter.csv

Let's load the dataset and store it in a variable. We'll have a copy of the original dataset so that we can rollback to the original version of the dataset whenever required.

In [2]:

df = pd.read\_csv('../input/palmer-archipelago-antarctica-penguin-data/penguins\_size.csv')

original = df.copy()

Quick Inspection of the Data

In [3]:

print('Dataset has', df.shape[0] , 'rows and', df.shape[1], 'columns')

Dataset has 344 rows and 7 columns

In [4]:

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 culmen\_length\_mm 342 non-null float64

3 culmen\_depth\_mm 342 non-null float64

4 flipper\_length\_mm 342 non-null float64

5 body\_mass\_g 342 non-null float64

6 sex 334 non-null object

dtypes: float64(4), object(3)

memory usage: 18.9+ KB

In [5]:

df.describe()

Out[5]:

|  | culmen\_length\_mm | culmen\_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 |

In [6]:

df.isnull().sum()

Out[6]:

species 0

island 0

culmen\_length\_mm 2

culmen\_depth\_mm 2

flipper\_length\_mm 2

body\_mass\_g 2

sex 10

dtype: int64

This data seems to have some missing values. Let's leave this for now, we'll impute missing values later.

In [7]:

df.head(10)

Out[7]:

|  | species | island | culmen\_length\_mm | culmen\_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 |
| 5 | Adelie | Torgersen | 39.3 | 20.6 | 190.0 | 3650.0 | MALE |
| 6 | Adelie | Torgersen | 38.9 | 17.8 | 181.0 | 3625.0 | FEMALE |
| 7 | Adelie | Torgersen | 39.2 | 19.6 | 195.0 | 4675.0 | MALE |
| 8 | Adelie | Torgersen | 34.1 | 18.1 | 193.0 | 3475.0 | NaN |
| 9 | Adelie | Torgersen | 42.0 | 20.2 | 190.0 | 4250.0 | NaN |

Exploratory Data Analysis

Univariate Analysis

Let's try to understand how the categorical variables are distributed. I'll use the value\_counts() method with an argument 'normalize' set to True to see the result i terms of percentage.

In [8]:

plt.rcParams['figure.figsize'] = (10,7)

In [9]:

df['species'].value\_counts(normalize = True).plot(kind = 'bar', color = 'seagreen', linewidth = 1, edgecolor = 'k')

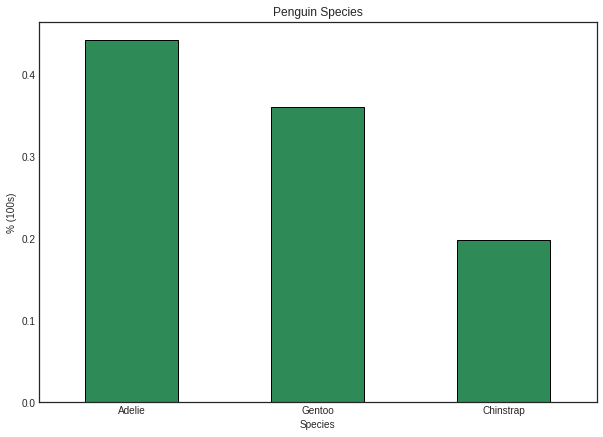
plt.title('Penguin Species')

plt.xlabel('Species')

plt.ylabel('% (100s)')

plt.xticks(rotation = 360)

plt.show()



In [10]:

df['island'].value\_counts(normalize = True).plot(kind = 'bar', color = 'seagreen', linewidth = 1, edgecolor = 'k')

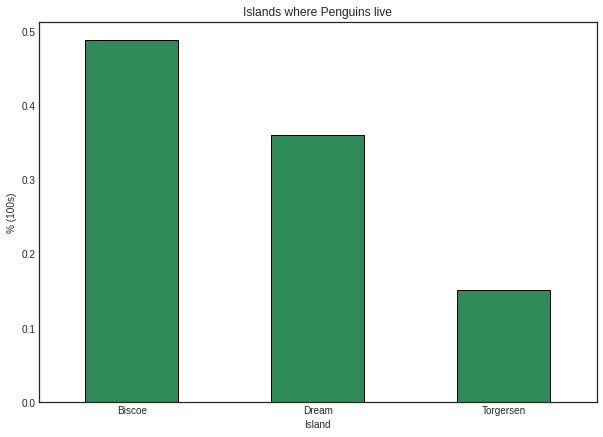
plt.title('Islands where Penguins live')

plt.xlabel('Island')

plt.ylabel('% (100s)')

plt.xticks(rotation = 360)

plt.show()



In [11]:

df['sex'].value\_counts(normalize = True).plot(kind = 'bar', color = 'seagreen', linewidth = 1, edgecolor = 'k')

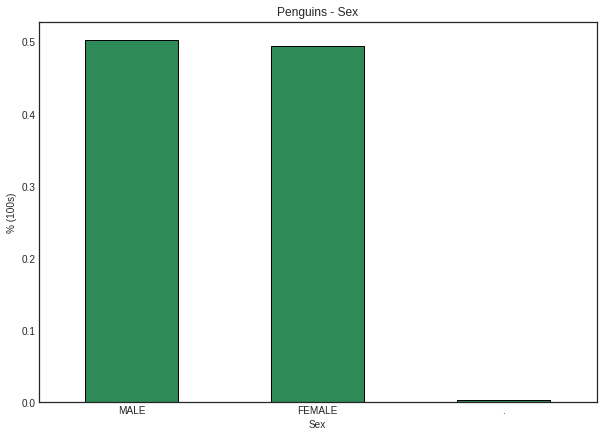
plt.title('Penguins - Sex')

plt.xlabel('Sex')

plt.ylabel('% (100s)')

plt.xticks(rotation = 360)

plt.show()



The third bar in this graph shows the inconsistency in this feature. This would be treated in the upcoming sections.

Okay! We explored the categorical features. What about the numerical features?

Shall we use histograms for this?

We can use histograms, but it suffers from binning bias. I would go with the Probability Density Function which says the probability of a random variable x picked at a time. Since the variable is continuous, we have chosen PDF.

We also have something called Empirical Cumulative Distribution Function, which says the probability of getting a value less than or equal to a random value picked at a time. Simple! This is a cumulative distribution function basically, except the fact that the CDF works on samples whereas the ECDF works on the real data.

Let me write a simple function which can plot both ECDF and PDF.

In [12]:

def ecdf(x):

n = len(x)

a = np.sort(x)

b = np.arange(1, 1 + n) / n

plt.subplot(211)

plt.plot(a, b, marker = '.', linestyle = 'None', c = 'seagreen')

mean\_x = np.mean(x)

plt.axvline(mean\_x, c = 'k', label = 'Mean')

plt.title('ECDF')

plt.legend()

plt.show()

plt.subplot(212)

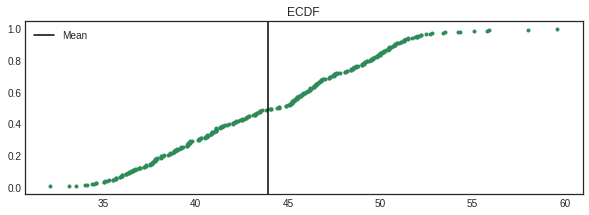
sns.distplot(x, color = 'r')

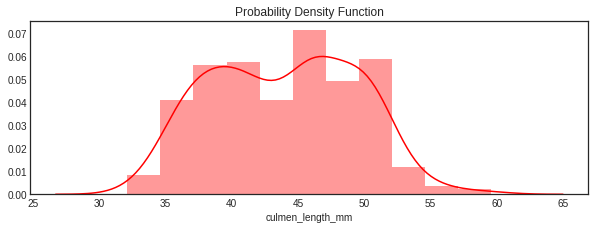
plt.title('Probability Density Function')

plt.show()

In [13]:

ecdf(df['culmen\_length\_mm'])





What does the ECDF shows?

Well, do you notice a black line there? It is the mean value of this feature, which is at 44. Look at the value in y-axis corresponding to the mean. It is somewhere around 0.5.

This infers that the probability of getting a value less than the mean culmen length is 0.5!

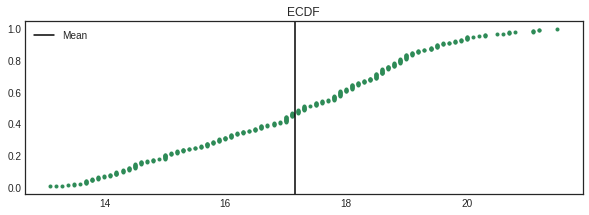
Quite interesting right?

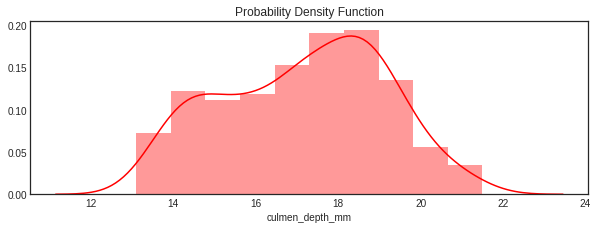
Let's now look for the probability of getting a value less than the culmen length 40mm. It is around 0.2, which means there is only a 20% chance of the culmen length to be less than 40mm if you randomly pick a value.

You got the logic?

In [14]:

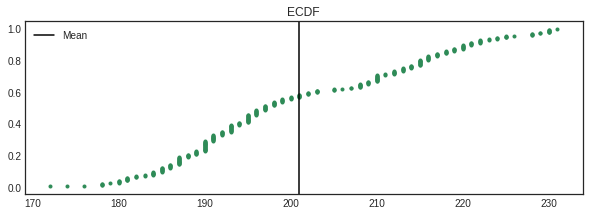
ecdf(df['culmen\_depth\_mm'])

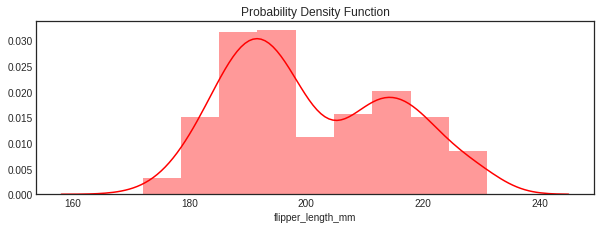




In [15]:

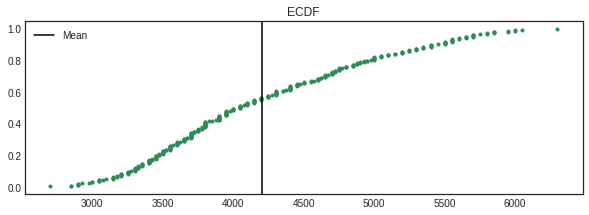
ecdf(df['flipper\_length\_mm'])

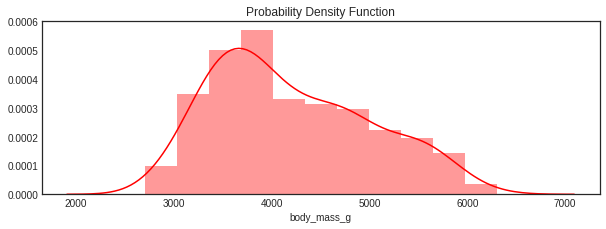




In [16]:

ecdf(df['body\_mass\_g'])





Multivariate Analysis

As we have analyzed the distribution of every features, let's try to analyze the relationship between them. Let me write a simple function which plots the boxplot of features which is classified by the species and their sex.

This is a great way to check how the features vary for different sex and species.

In [17]:

def box(f):

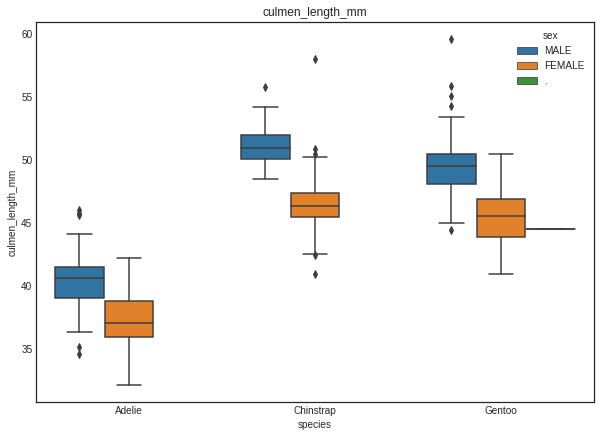
sns.boxplot(y = f, x = 'species', hue = 'sex',data = df)

plt.title(f)

plt.show()

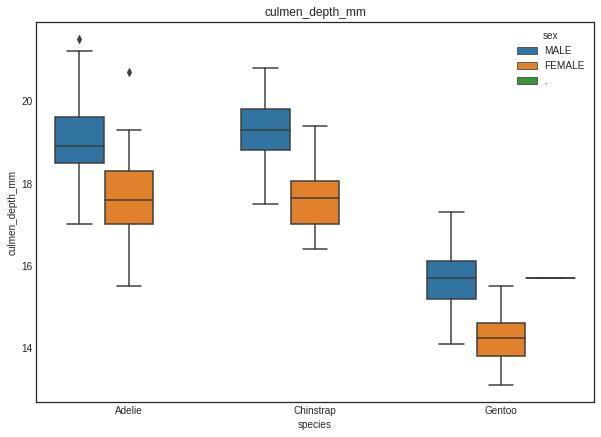
In [18]:

box('culmen\_length\_mm')



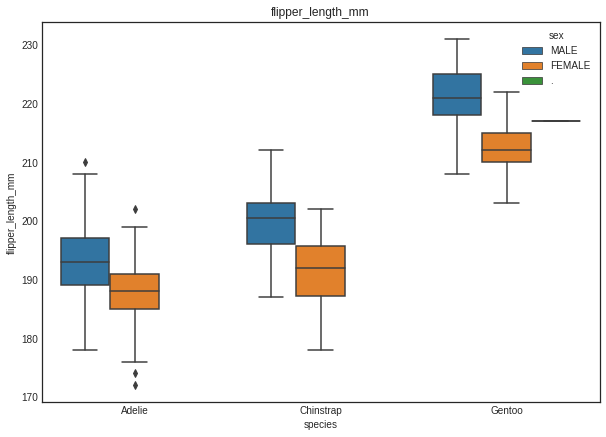
In [19]:

box('culmen\_depth\_mm')



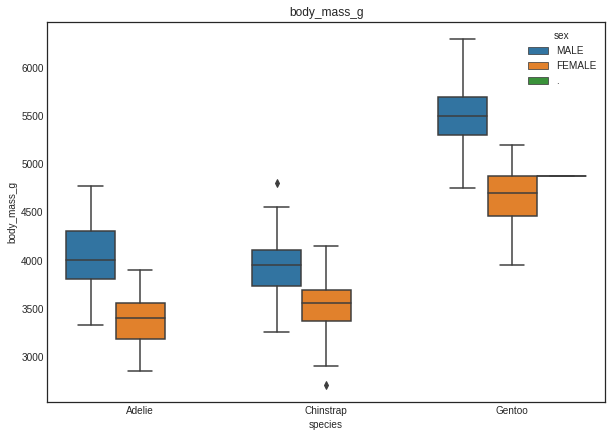
In [20]:

box('flipper\_length\_mm')



In [21]:

box('body\_mass\_g')



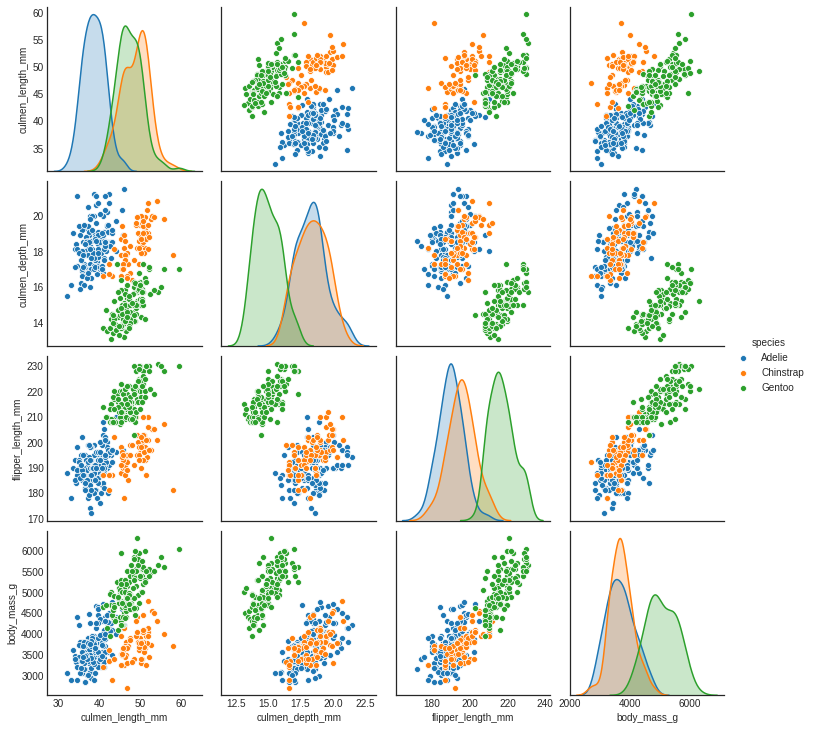
A common thing which I noticed from all the above graphs is that the male penguins have more culmen length, depth, flipper length and body mass irrespective of their species. This would help us immensely during our modelling.

Now let's plot a pairplot to see the multivariate trends all at the same time.

In [22]:

sns.pairplot(df, hue = 'species')

plt.show()



Missing Values Treatment

As you have seen earlier, we were having some missing values in the original dataset. Let's treat them.

Since the missing values are negligible in number, let's use the most common imputation strategies - mean and mode. For numeric variables, I would use the mean technique and for cateogorical variables mode is used.

In [23]:

new\_df = original.copy()

new\_df['culmen\_length\_mm'].fillna(np.mean(original['culmen\_length\_mm']), inplace = True)

new\_df['culmen\_depth\_mm'].fillna(np.mean(original['culmen\_depth\_mm']), inplace = True)

new\_df['flipper\_length\_mm'].fillna(np.mean(original['flipper\_length\_mm']), inplace = True)

new\_df['body\_mass\_g'].fillna(np.mean(original['body\_mass\_g']), inplace = True)

new\_df['sex'].fillna(original['sex'].mode()[0], inplace = True)

In [24]:

new\_df.head()

Out[24]:

|  | species | island | culmen\_length\_mm | culmen\_depth\_mm | flipper\_length\_mm | body\_mass\_g | sex |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | Adelie | Torgersen | 39.10000 | 18.70000 | 181.000000 | 3750.000000 | MALE |
| 1 | Adelie | Torgersen | 39.50000 | 17.40000 | 186.000000 | 3800.000000 | FEMALE |
| 2 | Adelie | Torgersen | 40.30000 | 18.00000 | 195.000000 | 3250.000000 | FEMALE |
| 3 | Adelie | Torgersen | 43.92193 | 17.15117 | 200.915205 | 4201.754386 | MALE |
| 4 | Adelie | Torgersen | 36.70000 | 19.30000 | 193.000000 | 3450.000000 | FEMALE |

In [25]:

new\_df.isnull().sum()

Out[25]:

species 0

island 0

culmen\_length\_mm 0

culmen\_depth\_mm 0

flipper\_length\_mm 0

body\_mass\_g 0

sex 0

dtype: int64

Cool, now we have got rid of all the missing values. Let's move ahead to the feature transformation.

Feature Transformation

Let's check whether the dataset is skewed. As we have noticed from the density plots of the numeric variables, there was not seen any normal distribution. But let's check the skewnesss of the features once. If the skewness is more, we can transform the variables using np.sqrt, np.log etc.

In [26]:

print('Skewness of numeric variables')

print('-' \* 35)

for i **in** new\_df.select\_dtypes(['int64', 'float64']).columns.tolist():

print(i, ' : ',new\_df[i].skew())

Skewness of numeric variables

-----------------------------------

culmen\_length\_mm : 0.053271788831634054

culmen\_depth\_mm : -0.1438798068350749

flipper\_length\_mm : 0.34668222408256033

body\_mass\_g : 0.47169044722118986

I do not see that the data is highly skewed. Let's quickly move to the normalization section.

Why do we need to normalize our data?

The reason being, the scale of every feature in the dataset is different. We noticed this during our inspection of the dataset at an initial stage. This is something to be treated.

I've chosen MinMaxScaler for this exercise. This scales the values in the particular feature such that they lie within 0 and 1. This makes the dataset to have the same range.

In [27]:

from sklearn.preprocessing import MinMaxScaler

mms = MinMaxScaler()

In [28]:

new\_df['culmen\_length\_mm'] = mms.fit\_transform(new\_df['culmen\_length\_mm'].values.reshape(-1, 1))

new\_df['culmen\_depth\_mm'] = mms.fit\_transform(new\_df['culmen\_depth\_mm'].values.reshape(-1, 1))

new\_df['flipper\_length\_mm'] = mms.fit\_transform(new\_df['flipper\_length\_mm'].values.reshape(-1, 1))

new\_df['body\_mass\_g'] = mms.fit\_transform(new\_df['body\_mass\_g'].values.reshape(-1, 1))

In [29]:

new\_df.head()

Out[29]:

|  | species | island | culmen\_length\_mm | culmen\_depth\_mm | flipper\_length\_mm | body\_mass\_g | sex |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | Adelie | Torgersen | 0.254545 | 0.666667 | 0.152542 | 0.291667 | MALE |
| 1 | Adelie | Torgersen | 0.269091 | 0.511905 | 0.237288 | 0.305556 | FEMALE |
| 2 | Adelie | Torgersen | 0.298182 | 0.583333 | 0.389831 | 0.152778 | FEMALE |
| 3 | Adelie | Torgersen | 0.429888 | 0.482282 | 0.490088 | 0.417154 | MALE |
| 4 | Adelie | Torgersen | 0.167273 | 0.738095 | 0.355932 | 0.208333 | FEMALE |

Now the dataset seems to have normalized, let's check this by seeing the summary stats of the data.

In [30]:

new\_df.describe()

Out[30]:

|  | culmen\_length\_mm | culmen\_depth\_mm | flipper\_length\_mm | body\_mass\_g |
| --- | --- | --- | --- | --- |
| count | 344.000000 | 344.000000 | 344.000000 | 344.000000 |
| mean | 0.429888 | 0.482282 | 0.490088 | 0.417154 |
| std | 0.197951 | 0.234408 | 0.237638 | 0.222115 |
| min | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 25% | 0.260909 | 0.297619 | 0.305085 | 0.236111 |
| 50% | 0.441818 | 0.500000 | 0.423729 | 0.375000 |
| 75% | 0.596364 | 0.666667 | 0.694915 | 0.569444 |
| max | 1.000000 | 1.000000 | 1.000000 | 1.000000 |

Did you notice the mean is now in the same range? Also the min and max of every variable are 0 and 1. So the dataset is now normalized.

We have categorical variables in our dataset. What are we going to do for that? Fine, let's use the pd.get\_dummies function to create dummy variables, as these variables can't be randomly assigned any values.

In [31]:

new\_df\_dummy = pd.get\_dummies(new\_df, columns = ['sex', 'island'], drop\_first = True)

In [32]:

new\_df\_dummy['species'].unique()

Out[32]:

array(['Adelie', 'Chinstrap', 'Gentoo'], dtype=object)

In [33]:

new\_df\_dummy['species'].replace({'Adelie' : 0,

'Chinstrap' : 1,

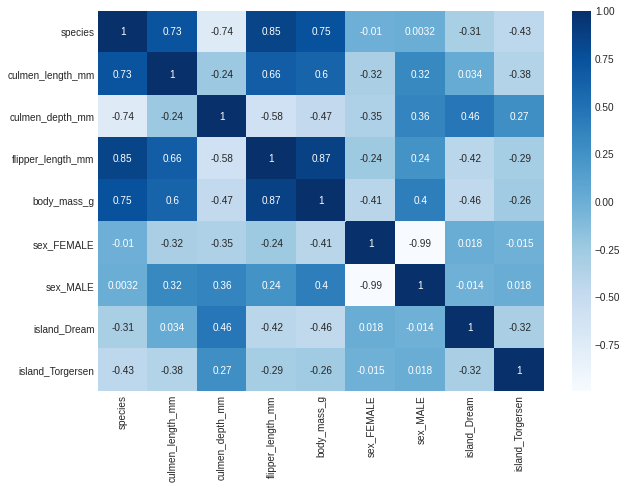
'Gentoo': 2}, inplace = True)

In [34]:

sns.heatmap(new\_df\_dummy.corr(), annot = True, cmap = 'Blues')

Out[34]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fc2342e69d0>



As you see in the correlation map, there is a significant correlation seen between the predictors and the target. This would help us during the modelling stage.

Model Building

Since we are all set, let's start the modelling. Let's import the required machine learning libraries and evaluation metrics from sklearn.

Then we'll separate the independant and dependant variables before splitting them into train and test sets using train\_test\_split.

In [35]:

from sklearn.model\_selection import train\_test\_split, KFold, cross\_val\_score

from sklearn.metrics import accuracy\_score, f1\_score, recall\_score, precision\_score, confusion\_matrix

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import SVC

In [36]:

X = new\_df\_dummy.drop(columns = ['species', 'sex\_FEMALE', 'sex\_MALE'])

Y = new\_df\_dummy['species']

In [37]:

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size = 0.25, random\_state = 123)

Let's first try with a simple Logistic Regression model.

In [38]:

LR = LogisticRegression()

LR.fit(X\_train, Y\_train)

pred = LR.predict(X\_test)

In [39]:

print('Accuracy : ', accuracy\_score(Y\_test, pred))

print('F1 Score : ', f1\_score(Y\_test, pred, average = 'weighted'))

Accuracy : 1.0

F1 Score : 1.0

This turned out to be a cool task! Let's try cross validation with different models and then pick up one.

In [40]:

models = []

models.append(('LR', LogisticRegression()))

models.append(('DT', DecisionTreeClassifier()))

models.append(('RF', RandomForestClassifier()))

models.append(('kNN', KNeighborsClassifier()))

models.append(('SVC', SVC()))

In [41]:

for name, model **in** models:

kfold = KFold(n\_splits = 5, random\_state = 42)

cv\_res = cross\_val\_score(model, X\_train, Y\_train, scoring = 'accuracy', cv = kfold)

print(name, ' : ', cv\_res.mean())

LR : 0.9846153846153847

DT : 0.9496229260935143

RF : 0.9612368024132729

kNN : 0.9846153846153847

SVC : 0.9961538461538462

In [42]:

svc = SVC()

svc.fit(X\_train, Y\_train)

pred = LR.predict(X\_test)

Model Evaluation

In [43]:

print('Accuracy : ', accuracy\_score(Y\_test, pred))

print('F1 Score : ', f1\_score(Y\_test, pred, average = 'weighted'))

print('Precision : ', precision\_score(Y\_test, pred , average = 'weighted'))

print('Recall : ', recall\_score(Y\_test, pred, average = 'weighted'))

Accuracy : 1.0

F1 Score : 1.0

Precision : 1.0

Recall : 1.0

In [44]:

confusion\_matrix(Y\_test, pred)

Out[44]:

array([[39, 0, 0],

[ 0, 19, 0],

[ 0, 0, 28]])

We tried modelling using the SVC model and it resulted in a good model. In this kernel, we tried out the basic stuff in all aspects. We can improve this by applying feature engineering (where we create more features which could result in a better model) and hyperparamater tuning.

We can also use this dataset to apply clustering algorithm to cluster the penguins to 3 clusters based on the species.

That's it from me!

V