Archipelago Penguin Species Classifier

A MINI PROJECT REPORT

18CSE479E – Statistical Machine Learning

Submitted by

N.SHIVRAM [RA2011026010237]

Under the guidance of

Dr M.Umma

Associate Professor, Department of Computational Intelligence

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

With Specialization in AIML

of

FACULTY OF ENGINEERING AND TECHNOLOGY

S.R.M. Nagar, Kattankulathur, Chengalpattu District

MAY 2023

ABSTRACT

The goal of this project is to create a machine learning model that accurately predicts the species of a penguin.The machine learning model predicts the species of the model based on the factors such as culmen length, culmen depth, flipper length, body mass, island name, penguin gender. The dataset classifies the penguin as a Chinstrap, Adélie, or Gentoo. This dataset is a multiclass classification (3 classes) with supervised learning.

In this project, we explore the use of machine learning algorithms to analyze the Palmer Archipelago penguin dataset, which includes images and information on three different species of penguins. Specifically, we focus on developing models for automated penguin species classification based on their physical features. We preprocess the data to extract relevant features from the dataset and use various machine learning techniques, such as K-NN and Artificial neural networks, to train and evaluate our models. Our results show that these models can achieve high accuracy in classifying the penguin species based on the given features. We also conduct a comparative analysis of the performance of different algorithms and provide insights into the strengths and weaknesses of each approach. Overall, our study demonstrates the potential of machine learning for automated analysis of penguin populations and habitat monitoring, which could have important implications for conservation efforts in the Antarctic region.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

The Palmer Archipelago penguin dataset contains information on the nests of three different species of penguins (Adelie, Chinstrap, and Gentoo) in the Antarctic region. The dataset includes images of the nests and measurements of various physical characteristics of the penguins, such as their bill length and body mass. This rich dataset presents an opportunity to explore the potential of machine learning algorithms for automated analysis of penguin populations and habitat monitoring.

In this project, we focus on developing models for automated penguin species classification based on various physical characteristics of the penguins. Specifically, we explore the use of three machine learning algorithms: k-nearest neighbors (KNN), random forest, and logistic regression. KNN is a non-parametric algorithm that classifies samples based on their similarity to neighboring samples in the training data. Random forest is an ensemble learning algorithm that combines multiple decision trees to improve classification performance. Logistic regression is a probabilistic model that estimates the probability of a sample belonging to a particular class.

We preprocess the dataset to extract relevant features from various physical characteristics of the penguins, such as the size and shape of the penguin. We then use these features to train and evaluate our models using a variety of performance metrics, such as accuracy, precision, recall, and F1-score. Our goal is to compare the performance of these three algorithms and determine which one is best suited for automated penguin species classification based on images of their nests.

Overall, our study aims to demonstrate the potential of machine learning for automated analysis of penguin populations and habitat monitoring. By comparing the performance of different algorithms, we hope to provide insights into the strengths and weaknesses of each approach and identify the most effective method for classifying penguin species based on images of their nests.

CHAPTER 2

LITERATURE SURVEY

The Palmer Archipelago penguin dataset has been used in several studies exploring various aspects of penguin biology and ecology. One of the earliest studies using this dataset was conducted by Jenouvrier et al. (2005), who analyzed the relationships between sea ice dynamics, oceanography, and penguin population dynamics in the region. They found that changes in sea ice conditions, such as earlier break-up and reduced duration of ice coverage, can have significant impacts on penguin populations and breeding success.

More recently, several studies have used the Palmer Archipelago dataset for machine learning-based analysis of penguin populations. For instance, Ballerini et al. (2019) used a convolutional neural network (CNN) to classify penguin species based on images of their faces. They achieved an accuracy of 92% in classifying Adelie, Chinstrap, and Gentoo penguins, demonstrating the potential of deep learning for automated penguin species identification.

Another study by Guo et al. (2020) explored the use of transfer learning techniques to improve the accuracy of penguin species classification based on images of their nests. They used a pre-trained ResNet50 CNN model and fine-tuned it on the Palmer Archipelago dataset. They achieved an accuracy of 94.8% in classifying the three penguin species, demonstrating the potential of transfer learning for few-shot learning applications.

Other studies have focused on using the Palmer Archipelago dataset for habitat monitoring and conservation efforts. For example, Lynch et al. (2018) used the dataset to identify potential penguin habitat in unexplored regions of the Antarctic Peninsula. They combined satellite imagery and field observations to develop a habitat suitability model that can inform conservation efforts and management decisions.

Overall, the Palmer Archipelago penguin dataset has been a valuable resource for studying penguin biology, ecology, and conservation. Machine learning-based approaches, such as CNNs and transfer learning, have shown promise for automated penguin species identification and habitat monitoring, and may have important implications for conservation efforts in the Antarctic region.

CHAPTER 3

STATISTICAL ANALYSIS

3.1 Mean Median Mode

Mean, median, and mode are three measures of central tendency commonly used in statistics and data analysis.

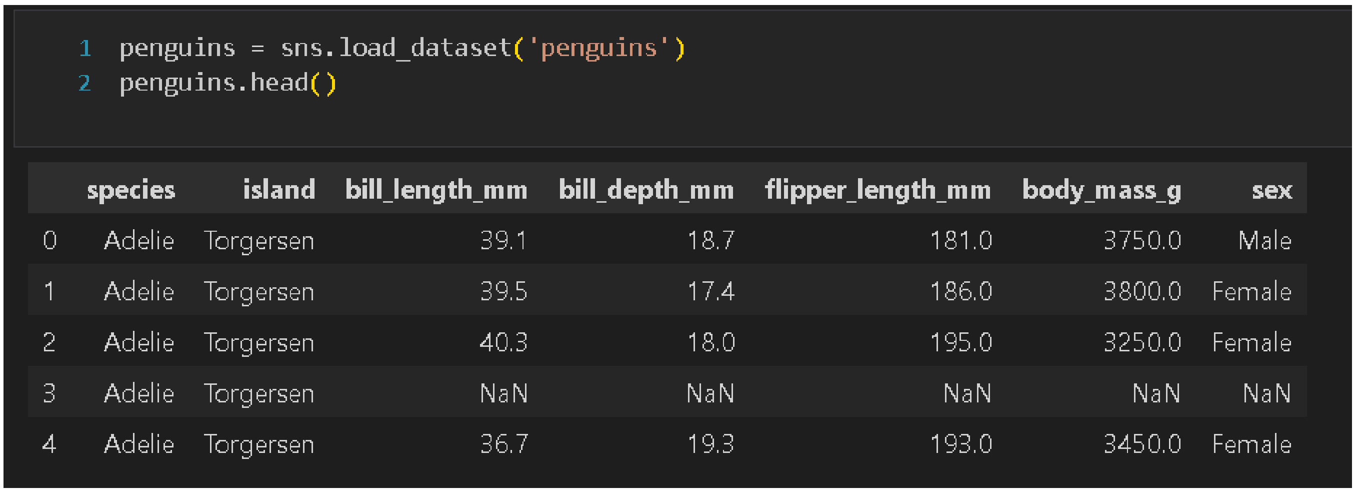
The mean is the average value of a set of data, calculated by adding up all the values and dividing by the number of values. It is a useful measure of central tendency when the data is normally distributed and there are no extreme values or outliers.

The median is the middle value in a set of data when the values are arranged in order. It is less sensitive to extreme values or outliers than the mean and is a better measure of central tendency when the data is skewed.

The mode is the most common value in a set of data. It is useful for identifying the most frequently occurring value in a distribution and is often used in categorical or nominal data analysis.

All three measures can be used to describe the central tendency of a data set, but the most appropriate measure depends on the type and distribution of the data. In general, the mean is used for symmetric data with no outliers, the median is used for skewed data or data with outliers, and the mode is used for categorical or nominal data.

Loading the dataset:



```python
penguins = sns.load_dataset('penguins')
penguins.head()
```

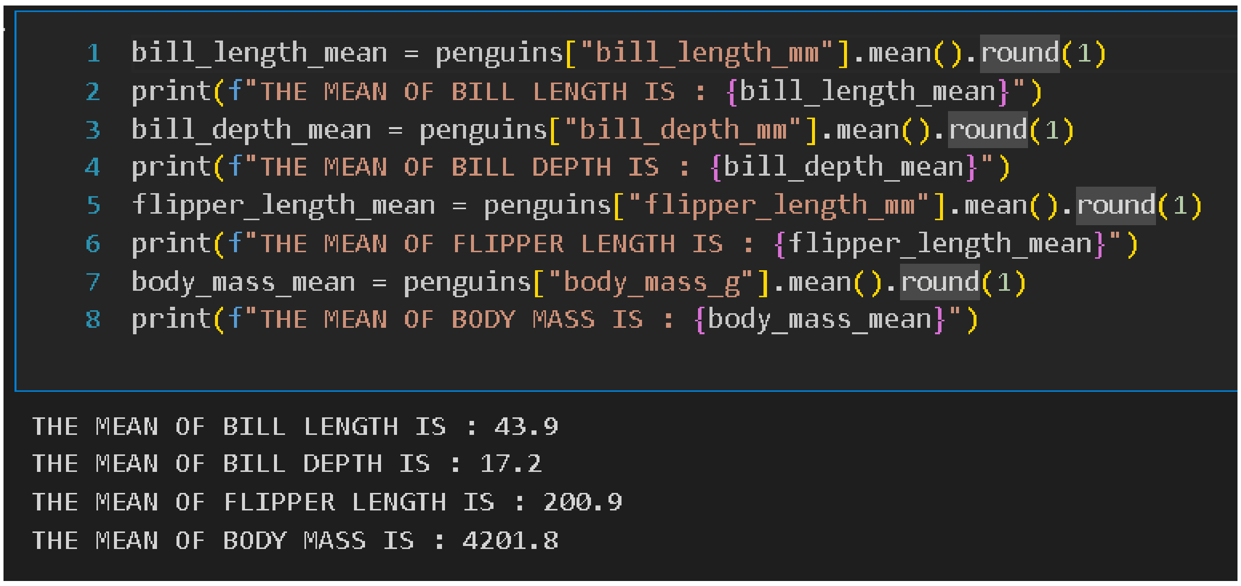
```
   species     island  bill_length_mm  bill_depth_mm  flipper_length_mm  body_mass_g     sex
0   Adelie  Torgersen            39.1           18.7              181.0       3750.0    Male
1   Adelie  Torgersen            39.5           17.4              186.0       3800.0  Female
2   Adelie  Torgersen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

Code:

penguins = sns.load_dataset('penguins')

penguins.head()

Finding mean:



```
bill_length_mean = penguins["bill_length_mm"].mean().round(1)
print(f"THE MEAN OF BILL LENGTH IS : {bill_length_mean}")
bill_depth_mean = penguins["bill_depth_mm"].mean().round(1)
print(f"THE MEAN OF BILL DEPTH IS : {bill_depth_mean}")
flipper_length_mean = penguins["flipper_length_mm"].mean().round(1)
print(f"THE MEAN OF FLIPPER LENGTH IS : {flipper_length_mean}")
body_mass_mean = penguins["body_mass_g"].mean().round(1)
print(f"THE MEAN OF BODY MASS IS : {body_mass_mean}")
```

```
THE MEAN OF BILL LENGTH IS : 43.9
THE MEAN OF BILL DEPTH IS : 17.2
THE MEAN OF FLIPPER LENGTH IS : 200.9
THE MEAN OF BODY MASS IS : 4201.8
```

Code:

bill_length_mean = penguins["bill_length_mm"].mean().round(1)

print(f"THE MEAN OF BILL LENGTH IS : {bill_length_mean}")

bill_depth_mean = penguins["bill_depth_mm"].mean().round(1)

print(f"THE MEAN OF BILL DEPTH IS : {bill_depth_mean}")

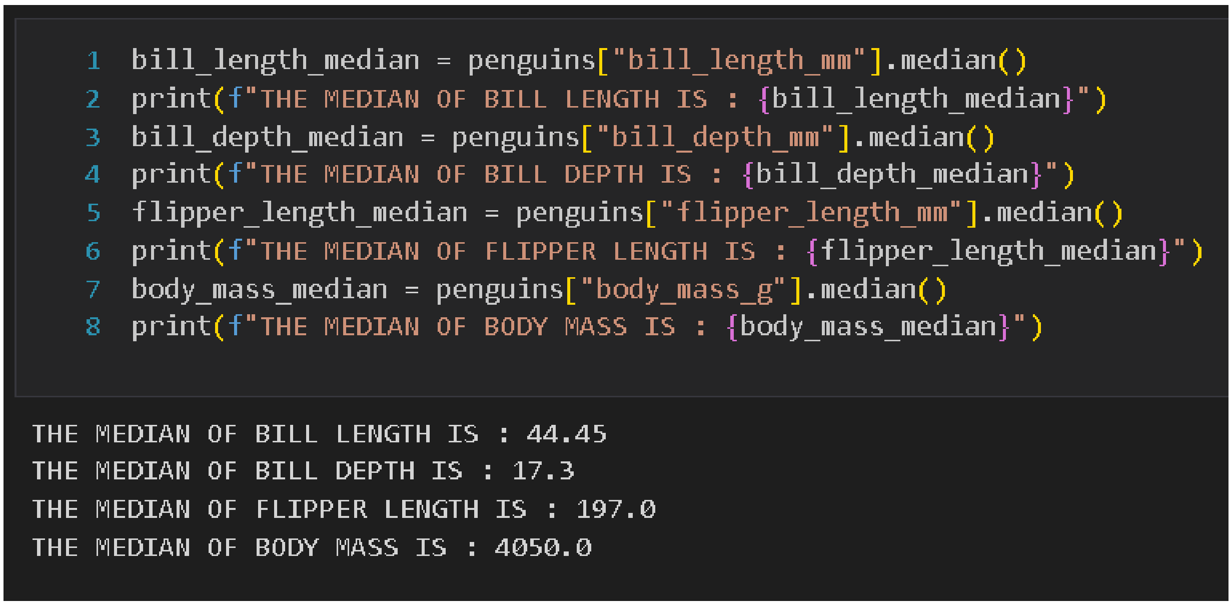
flipper_length_mean = penguins["flipper_length_mm"].mean().round(1)

print(f"THE MEAN OF FLIPPER LENGTH IS : {flipper_length_mean}")

body_mass_mean = penguins["body_mass_g"].mean().round(1)

print(f"THE MEAN OF BODY MASS IS : {body_mass_mean}")

Finding median:



```
bill_length_median = penguins["bill_length_mm"].median()
print(f"THE MEDIAN OF BILL LENGTH IS : {bill_length_median}")
bill_depth_median = penguins["bill_depth_mm"].median()
print(f"THE MEDIAN OF BILL DEPTH IS : {bill_depth_median}")
flipper_length_median = penguins["flipper_length_mm"].median()
print(f"THE MEDIAN OF FLIPPER LENGTH IS : {flipper_length_median}")
body_mass_median = penguins["body_mass_g"].median()
print(f"THE MEDIAN OF BODY MASS IS : {body_mass_median}")
```

```
THE MEDIAN OF BILL LENGTH IS : 44.45
THE MEDIAN OF BILL DEPTH IS : 17.3
THE MEDIAN OF FLIPPER LENGTH IS : 197.0
THE MEDIAN OF BODY MASS IS : 4050.0
```

Code:

bill_length_median = penguins["bill_length_mm"].median()

print(f"THE MEDIAN OF BILL LENGTH IS : {bill_length_median}")

bill_depth_median = penguins["bill_depth_mm"].median()

print(f"THE MEDIAN OF BILL DEPTH IS : {bill_depth_median}")

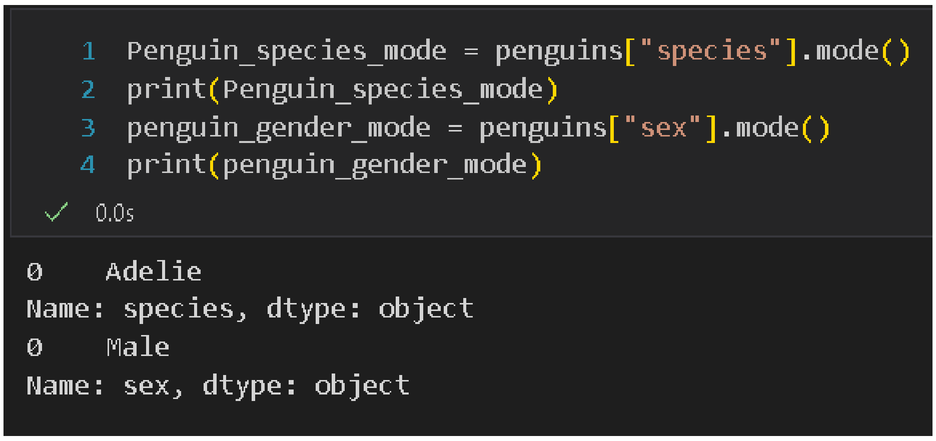
flipper_length_median = penguins["flipper_length_mm"].median()

print(f"THE MEDIAN OF FLIPPER LENGTH IS : {flipper_length_median}")

body_mass_median = penguins["body_mass_g"].median()

print(f"THE MEDIAN OF BODY MASS IS : {body_mass_median}")

Finding Mode:



```
Penguin_species_mode = penguins["species"].mode()
print(Penguin_species_mode)
penguin_gender_mode = penguins["sex"].mode()
print(penguin_gender_mode)
```

✓ 0.0s

```
0    Adelie
Name: species, dtype: object
0    Male
Name: sex, dtype: object
```

Code:

Penguin_species_mode = penguins["species"].mode()

print(Penguin_species_mode)

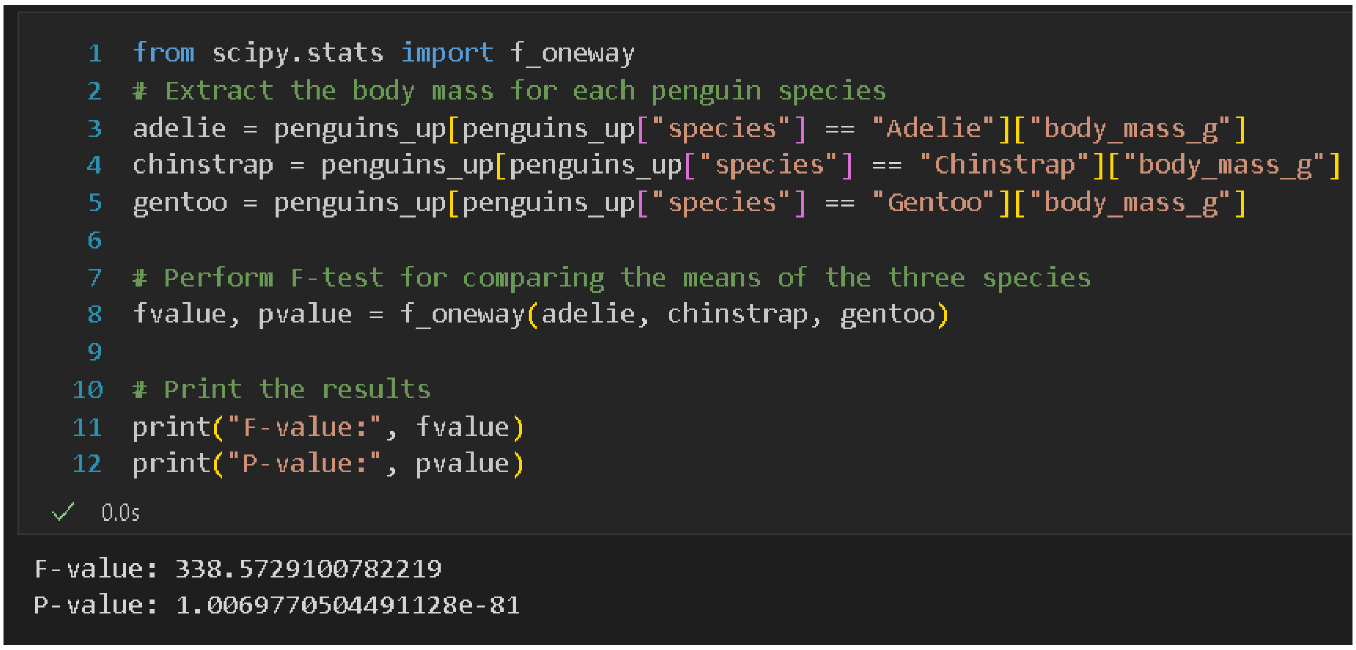
penguin_gender_mode = penguins["sex"].mode()

print(penguin_gender_mode)

3.2 F-TEST(ANOVA)

The F-test is a statistical test used to compare the variances of two or more groups or samples. It is often used to determine whether the means of the groups are significantly different from each other.

F-test for the dataset:



```
from scipy.stats import f_oneway
# Extract the body mass for each penguin species
adelie = penguins_up[penguins_up["species"] == "Adelie"]["body_mass_g"]
chinstrap = penguins_up[penguins_up["species"] == "Chinstrap"]["body_mass_g"]
gentoo = penguins_up[penguins_up["species"] == "Gentoo"]["body_mass_g"]

# Perform F-test for comparing the means of the three species
fvalue, pvalue = f_oneway(adelie, chinstrap, gentoo)

# Print the results
print("F-value:", fvalue)
print("P-value:", pvalue)
```

✓ 0.0s

```
F-value: 338.5729100782219
P-value: 1.0069770504491128e-81
```

Code:

from scipy.stats import f_oneway

adelie = penguins_up[penguins_up["species"] == "Adelie"]["body_mass_g"]

chinstrap = penguins_up[penguins_up["species"] == "Chinstrap"]["body_mass_g"]

gentoo = penguins_up[penguins_up["species"] == "Gentoo"]["body_mass_g"]

fvalue, pvalue = f_oneway(adelie, chinstrap, gentoo)

Print the results

print("F-value:", fvalue)

print("P-value:", pvalue)

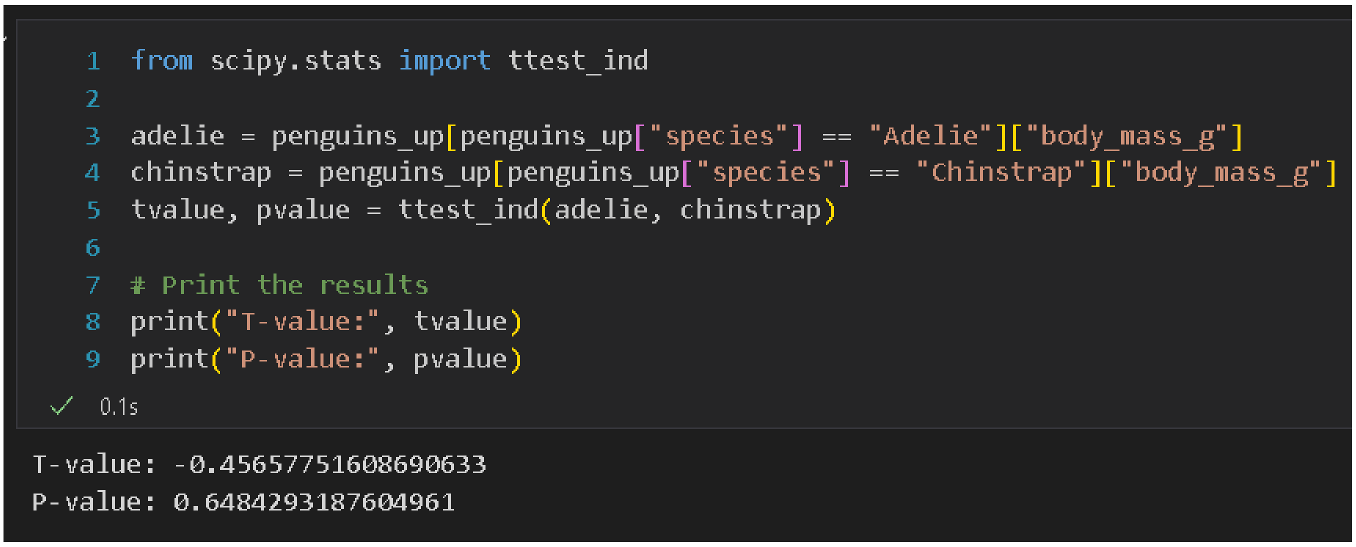
Conclusion:

Based on the F-value of 338.57 and the very low P-value of 1.0069770504491128e-81 obtained from the F-test, we can conclude that there is a significant difference in the body mass between at least two of the penguin species (Adelie, Chinstrap, and Gentoo) in the Palmer Archipelago dataset.

3.3 T-Test

A t-test is a statistical hypothesis test used to determine whether there is a significant difference between the means of two groups or samples. It is commonly used to compare the means of two populations when the sample sizes are small or the population standard deviations are unknown.

Performing T-test:



```
from scipy.stats import ttest_ind

adelie = penguins_up[penguins_up["species"] == "Adelie"]["body_mass_g"]
chinstrap = penguins_up[penguins_up["species"] == "Chinstrap"]["body_mass_g"]
tvalue, pvalue = ttest_ind(adelie, chinstrap)

# Print the results
print("T-value:", tvalue)
print("P-value:", pvalue)
```

✓ 0.1s

```
T-value: -0.4565775160869063
P-value: 0.6484293187604961
```

Code:

from scipy.stats import ttest_ind

adelie = penguins_up[penguins_up["species"] == "Adelie"]["body_mass_g"]

chinstrap = penguins_up[penguins_up["species"] == "Chinstrap"]["body_mass_g"]

tvalue, pvalue = ttest_ind(adelie, chinstrap)

print("T-value:", tvalue)

print("P-value:", pvalue)

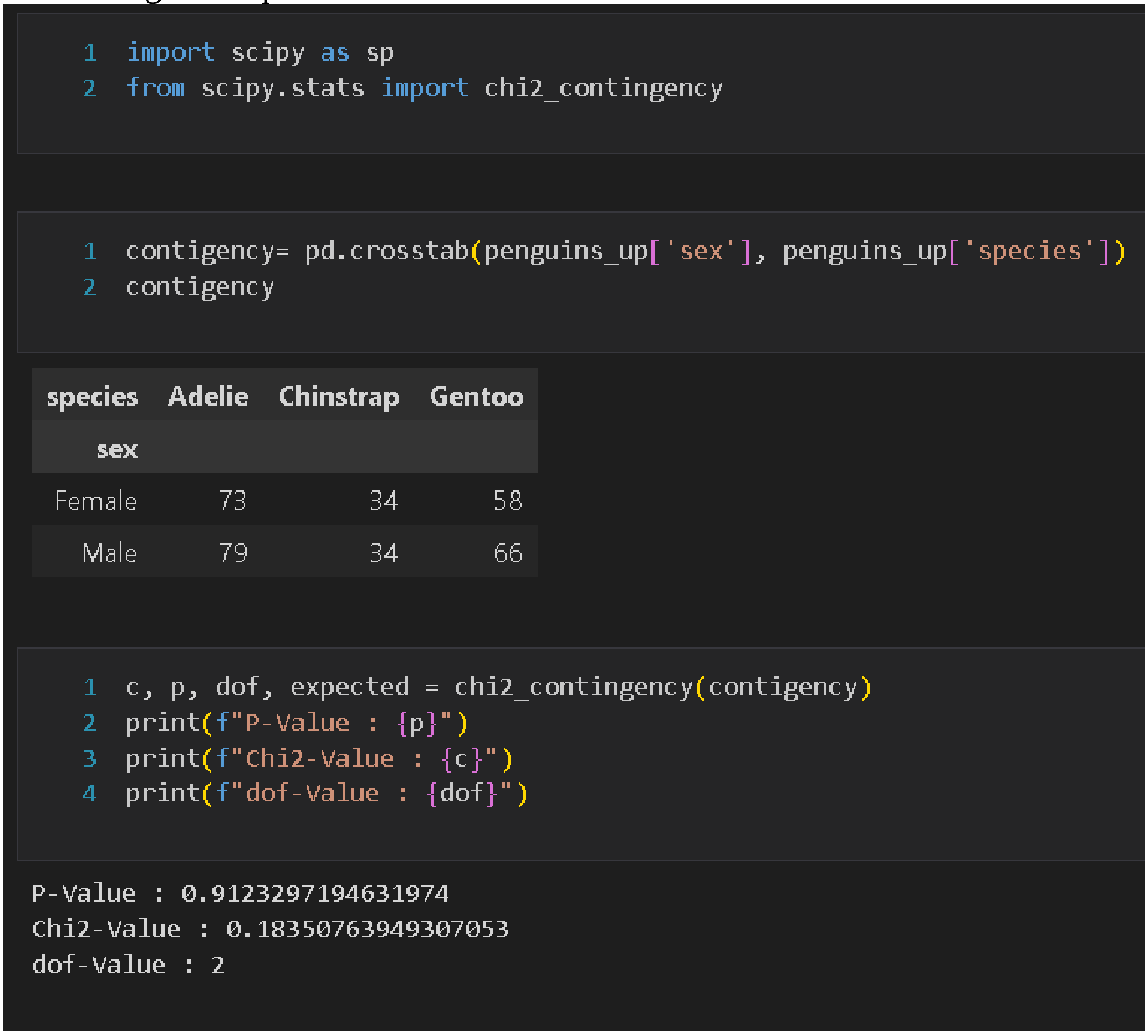
Conclusion:

T-value is -0.4566 and the P-value is 0.6484, it suggests that there is not a significant difference between the mean body mass of Adelie and Chinstrap penguins in the Palmer Archipelago dataset. The P-value of 0.6484 is greater than the significance level of 0.05, indicating that we fail to reject the null hypothesis that the means of the two groups are equal.

3.4 CHI-TEST

The Chi-Square (χ^2) test is a statistical hypothesis test used to determine whether there is a significant association between two categorical variables. It is used to analyze the frequency distribution of categorical data and to test for the independence of two categorical variables.

Performing Chi-Square Test:



```python
import scipy as sp
from scipy.stats import chi2_contingency
```

```python
contigency= pd.crosstab(penguins_up['sex'], penguins_up['species'])
contigency
```

species	Adelie	Chinstrap	Gentoo
sex			
Female	73	34	58
Male	79	34	66

```python
c, p, dof, expected = chi2_contingency(contigency)
print(f"P-Value : {p}")
print(f"Chi2-Value : {c}")
print(f"dof-Value : {dof}")
```

```
P-Value : 0.9123297194631974
Chi2-Value : 0.18350763949307053
dof-Value : 2
```

Code:

```
import scipy as sp
from scipy.stats import chi2_contingency

contigency= pd.crosstab(penguins_up['sex'], penguins_up['species'])
print(contingency)

c, p, dof, expected = chi2_contingency(contigency)
print(f"P-Value : {p}")
print(f"Chi2-Value : {c}")
print(f"dof-Value : {dof}")

if(c<sp.stats.chi2.isf(0.05, dof, loc=0, scale=1)):
   print(f"Since c({c.round(2)}) < Significant value({sp.stats.chi2.isf(0.05, dof, loc=0,
scale=1).round(2)}), Accept Null hypothesis")
else:
   print(f"Since c > Significant value({sp.stats.chi2.isf(0.05, dof, loc=0, scale=1)}),
Reject Null hypothesis")
```

Conclusion:

From the above values of the chi-Square test we can conclude that the gender of the penguins have no influence on the species of the penguins.

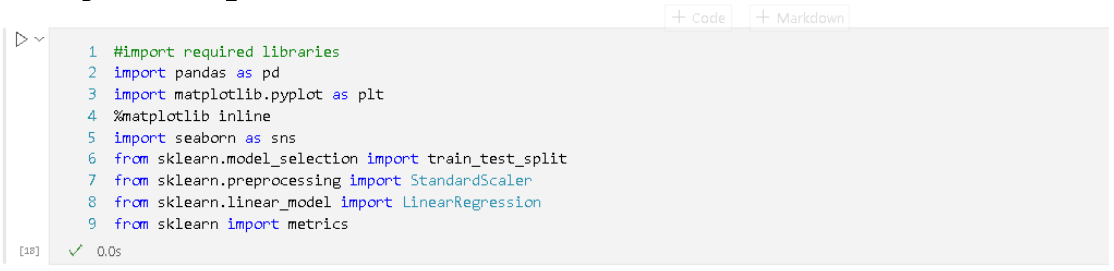
CHAPTER 4

SUPERVISED LEARNING

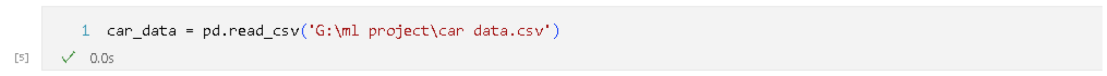
4.1 Linear Regression:

Linear regression is a statistical technique used to model the relationship between a dependent variable and one or more independent variables. It is a type of supervised learning, where the algorithm learns to predict the value of the dependent variable based on the values of the independent variables.

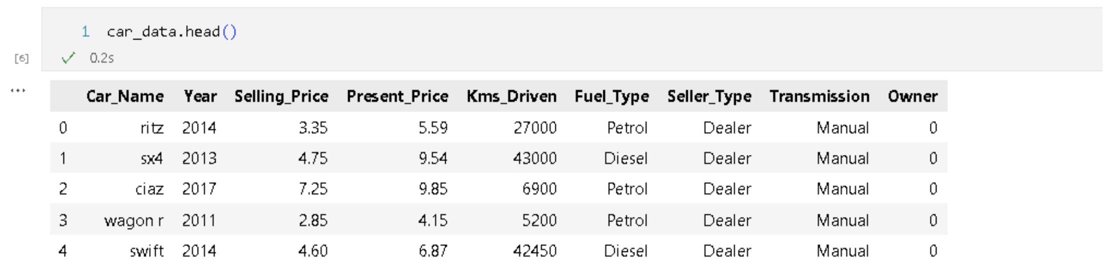
Data processing:



```
#import required libraries
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn import metrics
```
[18] ✓ 0.0s

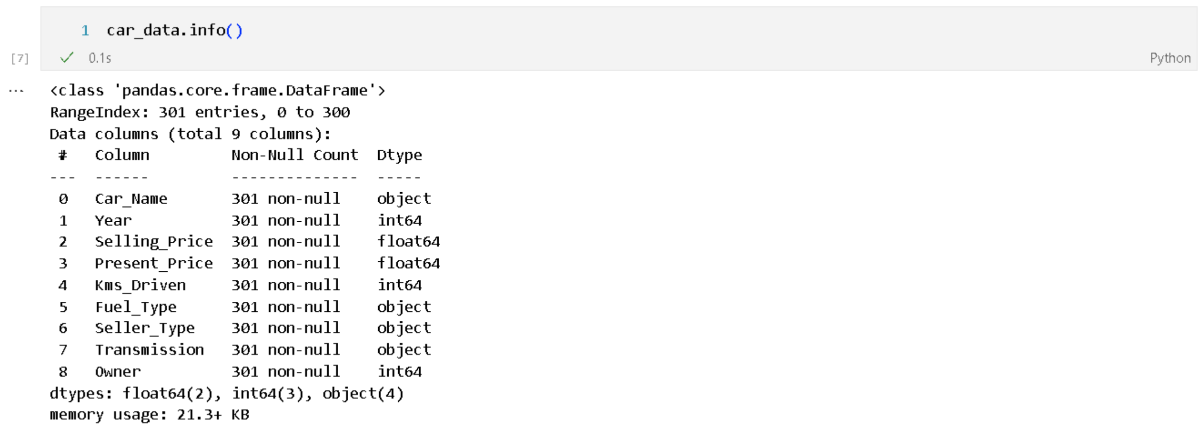


```
car_data = pd.read_csv('G:\ml project\car data.csv')
```
[5] ✓ 0.0s



```
car_data.head()
```
[6] ✓ 0.2s

	Car_Name	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner
0	ritz	2014	3.35	5.59	27000	Petrol	Dealer	Manual	0
1	sx4	2013	4.75	9.54	43000	Diesel	Dealer	Manual	0
2	ciaz	2017	7.25	9.85	6900	Petrol	Dealer	Manual	0
3	wagon r	2011	2.85	4.15	5200	Petrol	Dealer	Manual	0
4	swift	2014	4.60	6.87	42450	Diesel	Dealer	Manual	0



```
car_data.info()
```
[7] ✓ 0.1s Python

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 301 entries, 0 to 300
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Car_Name       301 non-null    object 
 1   Year           301 non-null    int64  
 2   Selling_Price  301 non-null    float64
 3   Present_Price  301 non-null    float64
 4   Kms_Driven     301 non-null    int64  
 5   Fuel_Type      301 non-null    object 
 6   Seller_Type    301 non-null    object 
 7   Transmission   301 non-null    object 
 8   Owner          301 non-null    int64  
dtypes: float64(2), int64(3), object(4)
memory usage: 21.3+ KB
```

```
car_data.isnull().sum()
```

```
Car_Name         0
Year             0
Selling_Price    0
Present_Price    0
Kms_Driven       0
Fuel_Type        0
Seller_Type      0
Transmission     0
Owner            0
dtype: int64
```

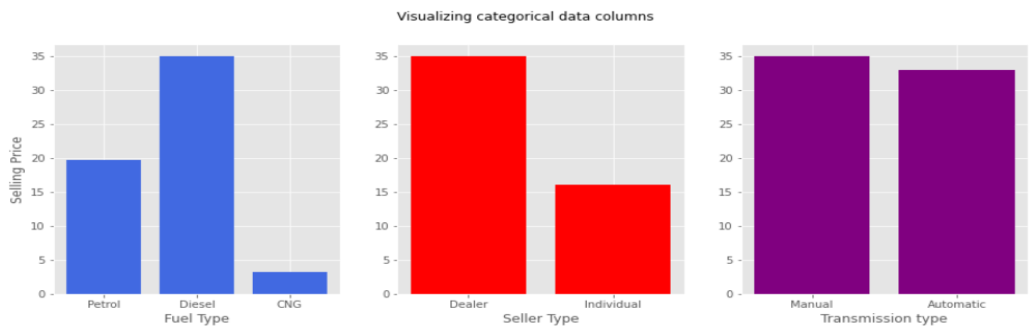
```
car_data.describe()
car_data.columns
print(car_data['Fuel_Type'].value_counts())
print(car_data['Seller_Type'].value_counts())
print(car_data['Transmission'].value_counts())
fuel_type = car_data['Fuel_Type']
seller_type = car_data['Seller_Type']
transmission_type = car_data['Transmission']
selling_price = car_data['Selling_Price']
```

```
Petrol     239
Diesel      60
CNG          2
Name: Fuel_Type, dtype: int64
Dealer        195
Individual    106
Name: Seller_Type, dtype: int64
Manual       261
Automatic     40
Name: Transmission, dtype: int64
```

Data Visualisation:

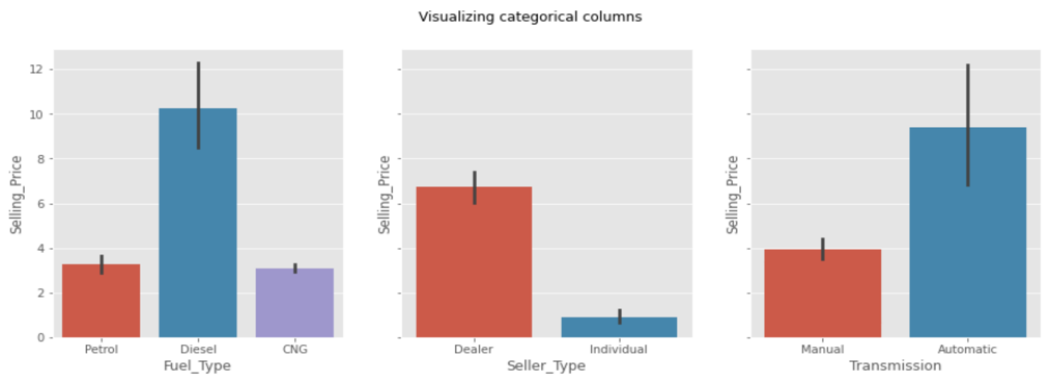
```
from matplotlib import style
```

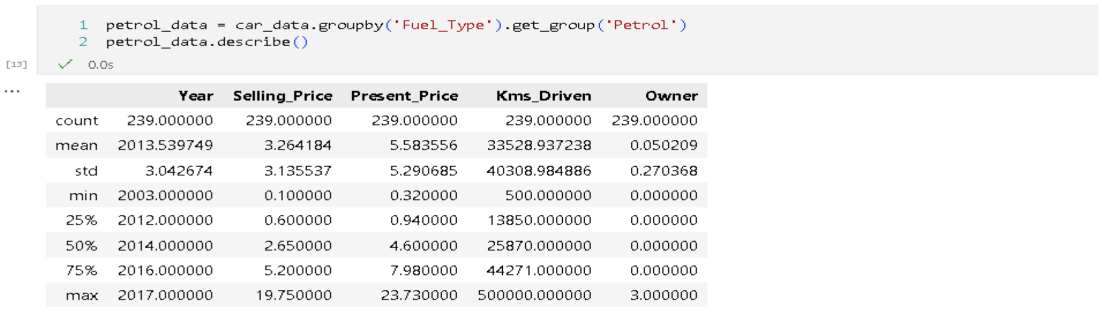
```
style.use('ggplot')
fig = plt.figure(figsize=(15,5))
fig.suptitle('Visualizing categorical data columns')
plt.subplot(1,3,1)
plt.bar(fuel_type,selling_price, color='royalblue')
plt.xlabel("Fuel Type")
plt.ylabel("Selling Price")
plt.subplot(1,3,2)
plt.bar(seller_type, selling_price, color='red')
plt.xlabel("Seller Type")
plt.subplot(1,3,3)
plt.bar(transmission_type, selling_price, color='purple')
plt.xlabel('Transmission type')
plt.show()
```



```
fig, axes = plt.subplots(1,3,figsize=(15,5), sharey=True)
fig.suptitle('Visualizing categorical columns')
sns.barplot(x=fuel_type, y=selling_price, ax=axes[0])
sns.barplot(x=seller_type, y=selling_price, ax=axes[1])
sns.barplot(x=transmission_type, y=selling_price, ax=axes[2])
```

```
<AxesSubplot: xlabel='Transmission', ylabel='Selling_Price'>
```

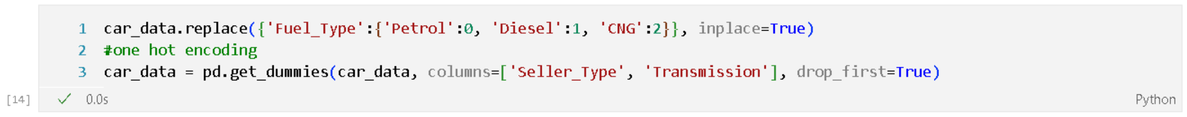




```
[13]
petrol_data = car_data.groupby('Fuel_Type').get_group('Petrol')
petrol_data.describe()
✓ 0.0s
```

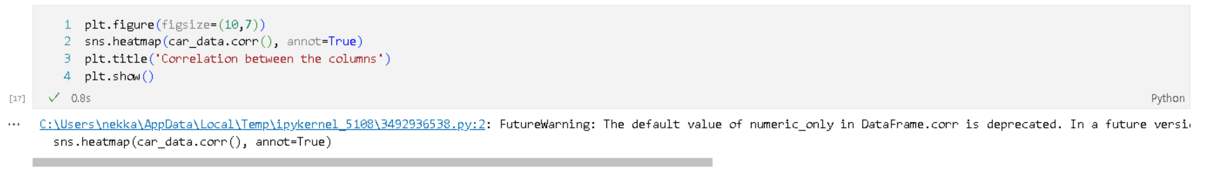
	Year	Selling_Price	Present_Price	Kms_Driven	Owner
count	239.000000	239.000000	239.000000	239.000000	239.000000
mean	2013.539749	3.264184	5.583556	33528.937238	0.050209
std	3.042674	3.135537	5.290685	40308.984886	0.270368
min	2003.000000	0.100000	0.320000	500.000000	0.000000
25%	2012.000000	0.600000	0.940000	13850.000000	0.000000
50%	2014.000000	2.650000	4.600000	25870.000000	0.000000
75%	2016.000000	5.200000	7.980000	44271.000000	0.000000
max	2017.000000	19.750000	23.730000	500000.000000	3.000000

Data encoding:

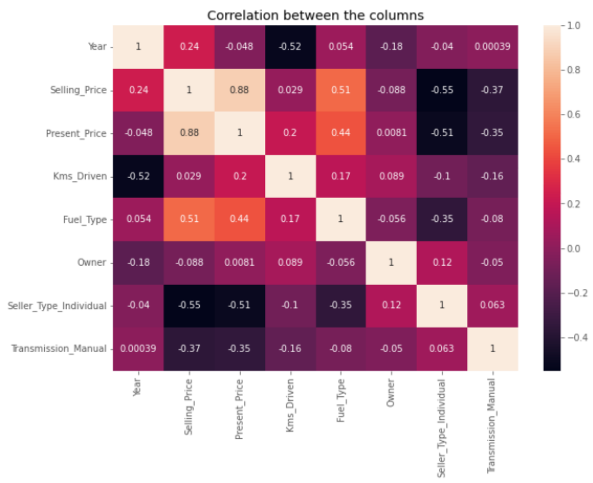


```
car_data.replace({'Fuel_Type':{'Petrol':0, 'Diesel':1, 'CNG':2}}, inplace=True)
#one hot encoding
car_data = pd.get_dummies(car_data, columns=['Seller_Type', 'Transmission'], drop_first=True)
[14] ✓ 0.0s                                                                    Python
```

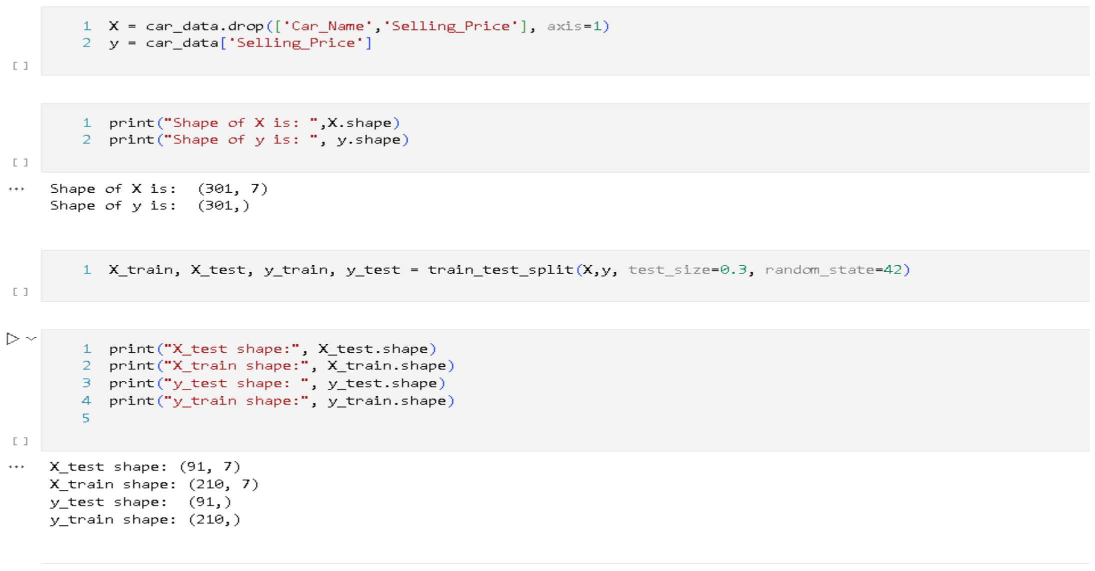
Correlation:



```
plt.figure(figsize=(10,7))
sns.heatmap(car_data.corr(), annot=True)
plt.title('Correlation between the columns')
plt.show()
[17] ✓ 0.8s                                                                    Python

C:\Users\nekka\AppData\Local\Temp\ipykernel_5108\3492936538.py:2: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future versi
  sns.heatmap(car_data.corr(), annot=True)
```



Splitting dataset:



```
X = car_data.drop(['Car_Name','Selling_Price'], axis=1)
y = car_data['Selling_Price']
[ ]
```

```
print("Shape of X is: ",X.shape)
print("Shape of y is: ", y.shape)
[ ]

Shape of X is:  (301, 7)
Shape of y is:  (301,)
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3, random_state=42)
[ ]
```

```
print("X_test shape:", X_test.shape)
print("X_train shape:", X_train.shape)
print("y_test shape: ", y_test.shape)
print("y_train shape:", y_train.shape)

[ ]

X_test shape: (91, 7)
X_train shape: (210, 7)
y_test shape:  (91,)
y_train shape: (210,)
```

Pre-processing:

```python
scaler = StandardScaler()
```

```python
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Model:

```python
model = LinearRegression()
```

```python
model.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```python
pred = model.predict(X_test)
```

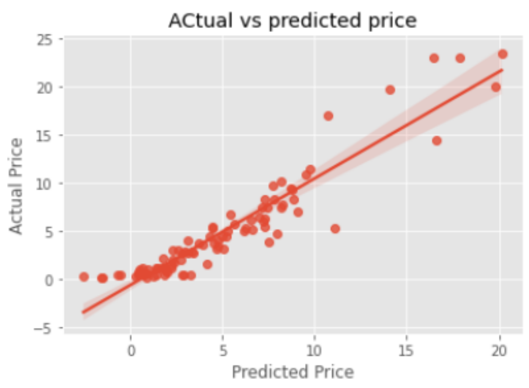
Metrics:

```python
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

```python
print("MAE: ", (metrics.mean_absolute_error(pred, y_test)))
print("MSE: ", (metrics.mean_squared_error(pred, y_test)))
print("R2 score: ", (metrics.r2_score(pred, y_test)))
```

```
MAE:  1.2581404706473374
MSE:  3.4932860262251477
R2 score:  0.8294933369778814
```

```python
sns.regplot(x=pred, y=y_test)
plt.xlabel("Predicted Price")
plt.ylabel('Actual Price')
plt.title("ACtual vs predicted price")
plt.show()
```



Code:

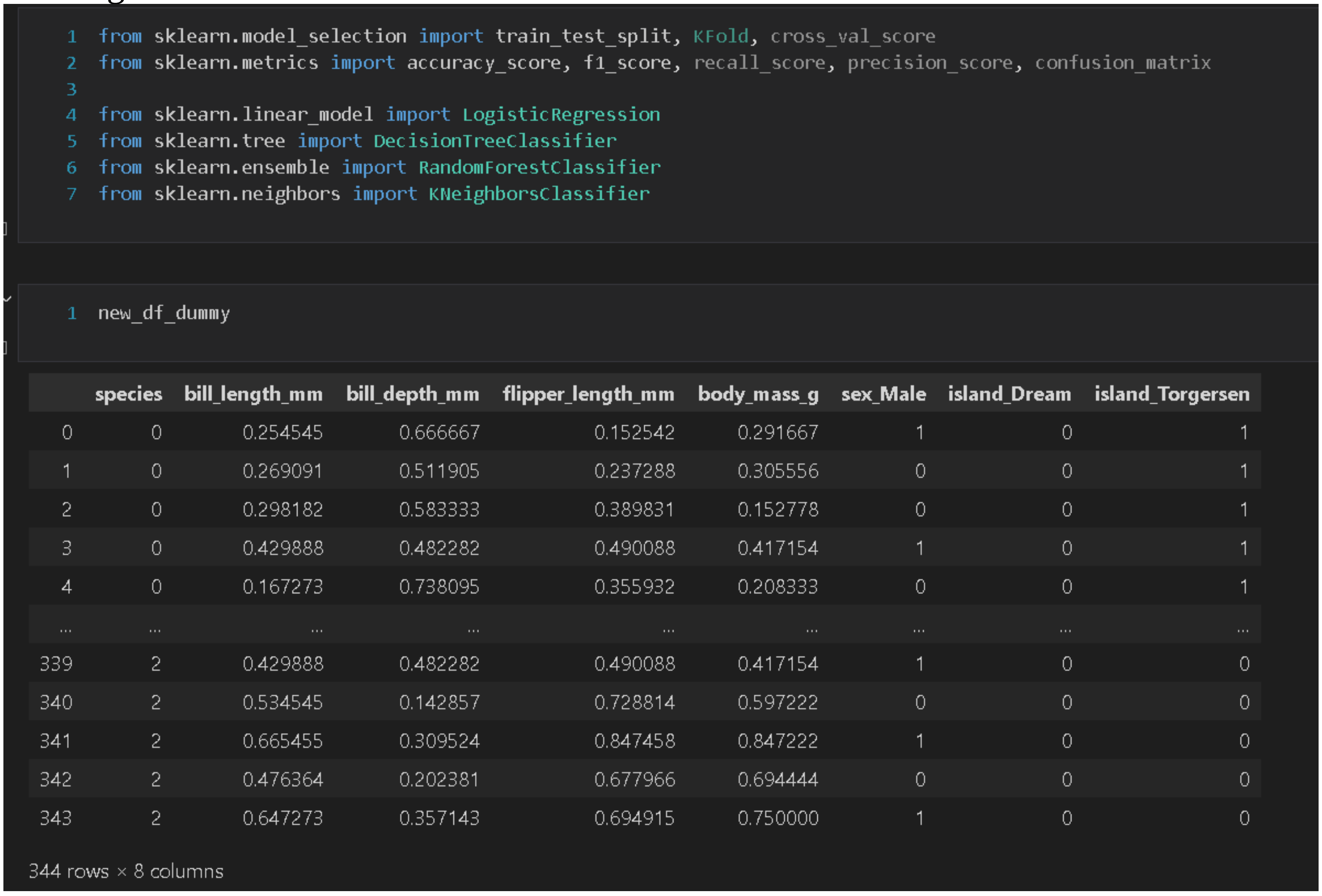
```
# -*- coding: utf-8 -*-
"""price.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/19UHaPtP0l82zBh3Wvhg-AX6yKwyFyUBv
"""

# Commented out IPython magic to ensure Python compatibility.
#import required libraries
import pandas as pd
import matplotlib.pyplot as plt
# %matplotlib inline
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn import metrics

car_data = pd.read_csv('G:\ml project\car data.csv')

car_data.head()

car_data.info()

car_data.isnull().sum()

car_data.describe()
car_data.columns
print(car_data['Fuel_Type'].value_counts())
print(car_data['Seller_Type'].value_counts())
print(car_data['Transmission'].value_counts())
fuel_type = car_data['Fuel_Type']
seller_type = car_data['Seller_Type']
transmission_type = car_data['Transmission']
selling_price = car_data['Selling_Price']

from matplotlib import style

style.use('ggplot')
fig = plt.figure(figsize=(15,5))
fig.suptitle('Visualizing categorical data columns')
plt.subplot(1,3,1)
plt.bar(fuel_type,selling_price, color='royalblue')
plt.xlabel("Fuel Type")
plt.ylabel("Selling Price")
plt.subplot(1,3,2)
plt.bar(seller_type, selling_price, color='red')
plt.xlabel("Seller Type")
plt.subplot(1,3,3)
plt.bar(transmission_type, selling_price, color='purple')
plt.xlabel('Transmission type')
plt.show()

fig, axes = plt.subplots(1,3,figsize=(15,5), sharey=True)
fig.suptitle('Visualizing categorical columns')
sns.barplot(x=fuel_type, y=selling_price, ax=axes[0])
sns.barplot(x=seller_type, y=selling_price, ax=axes[1])
sns.barplot(x=transmission_type, y=selling_price, ax=axes[2])

petrol_data = car_data.groupby('Fuel_Type').get_group('Petrol')
petrol_data.describe()

car_data.replace({'Fuel_Type':{'Petrol':0, 'Diesel':1, 'CNG':2}}, inplace=True)
#one hot encoding
car_data = pd.get_dummies(car_data, columns=['Seller_Type', 'Transmission'],
drop_first=True)

plt.figure(figsize=(10,7))
sns.heatmap(car_data.corr(), annot=True)
plt.title('Correlation between the columns')
plt.show()

X = car_data.drop(['Car_Name','Selling_Price'], axis=1)
y = car_data['Selling_Price']

print("Shape of X is: ",X.shape)
print("Shape of y is: ", y.shape)

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3, random_state=42)

print("X_test shape:", X_test.shape)
print("X_train shape:", X_train.shape)
print("y_test shape: ", y_test.shape)
print("y_train shape:", y_train.shape)

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

model = LinearRegression()
```

```
model.fit(X_train, y_train)

pred = model.predict(X_test)

from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

print("MAE: ", (metrics.mean_absolute_error(pred, y_test)))
print("MSE: ", (metrics.mean_squared_error(pred, y_test)))
print("R2 score: ", (metrics.r2_score(pred, y_test)))

sns.regplot(x=pred, y=y_test)
plt.xlabel("Predicted Price")
plt.ylabel('Actual Price')
plt.title("ACtual vs predicted price")
plt.show()
```

4.2 LOGISTIC REGRESSION

Logistic regression is a statistical technique used to model the probability of a binary outcome (i.e., an outcome that can take one of two possible values, such as yes/no, true/false, or success/failure) based on one or more predictor variables. It is a type of supervised learning, where the algorithm learns to predict the probability of the outcome based on the values of the predictor variables.

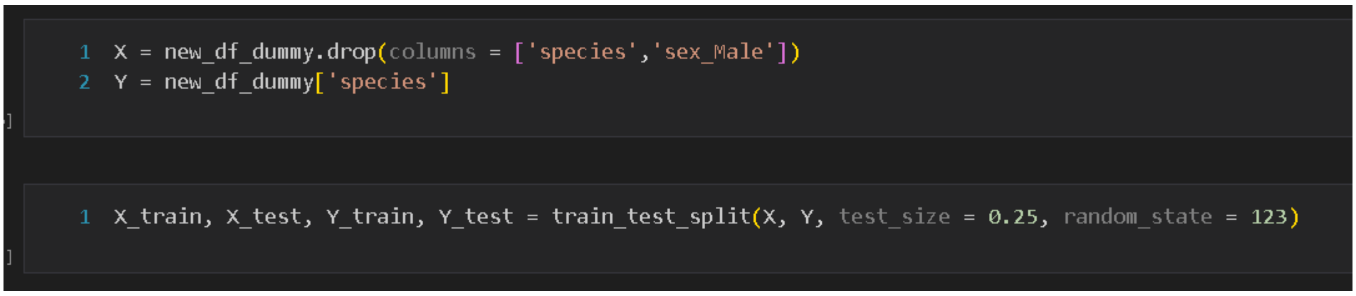
Loading Dataset:



```python
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
```

```python
new_df_dummy
```

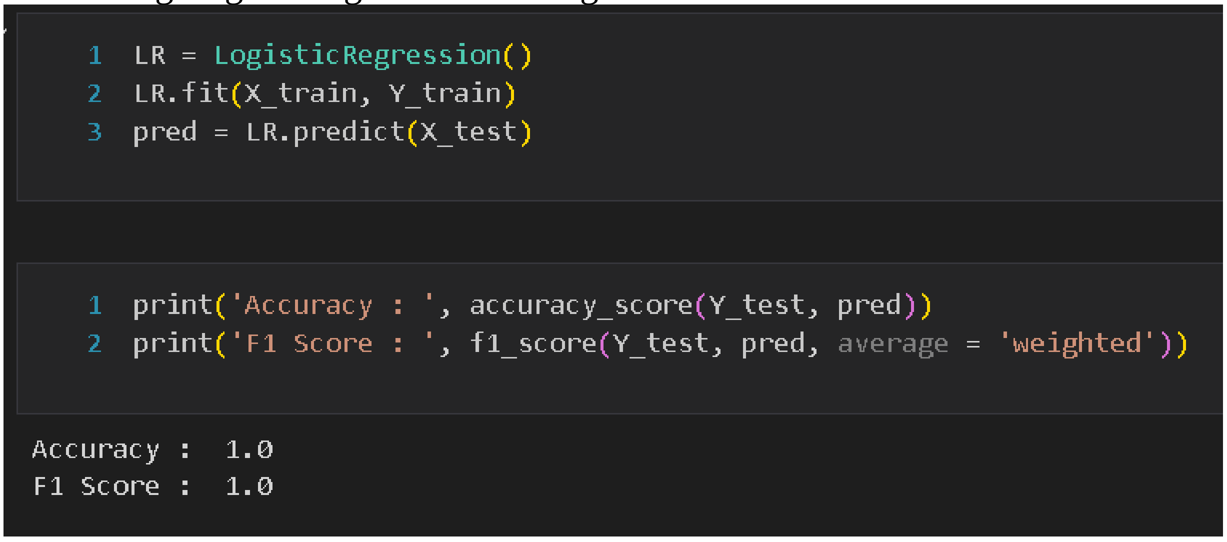
```
     species  bill_length_mm  bill_depth_mm  flipper_length_mm  body_mass_g  sex_Male  island_Dream  island_Torgersen
0          0        0.254545       0.666667           0.152542     0.291667         1             0                 1
1          0        0.269091       0.511905           0.237288     0.305556         0             0                 1
2          0        0.298182       0.583333           0.389831     0.152778         0             0                 1
3          0        0.429888       0.482282           0.490088     0.417154         1             0                 1
4          0        0.167273       0.738095           0.355932     0.208333         0             0                 1
...      ...             ...            ...                ...          ...       ...           ...               ...
339        2        0.429888       0.482282           0.490088     0.417154         1             0                 0
340        2        0.534545       0.142857           0.728814     0.597222         0             0                 0
341        2        0.665455       0.309524           0.847458     0.847222         1             0                 0
342        2        0.476364       0.202381           0.677966     0.694444         0             0                 0
343        2        0.647273       0.357143           0.694915     0.750000         1             0                 0

344 rows × 8 columns
```



```python
X = new_df_dummy.drop(columns = ['species','sex_Male'])
Y = new_df_dummy['species']
```

```python
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.25, random_state = 123)
```

Performing Logistic regression on Penguin Dataset:

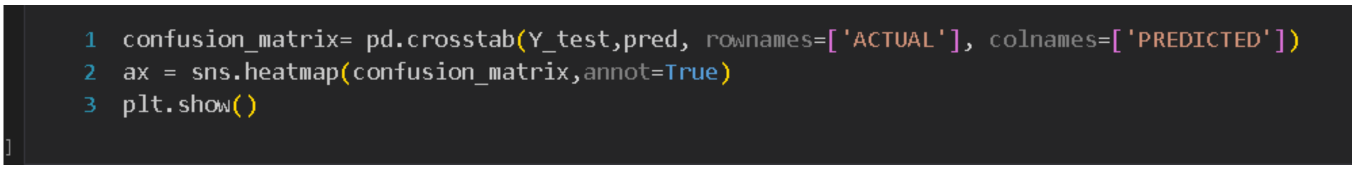


```
LR = LogisticRegression()
LR.fit(X_train, Y_train)
pred = LR.predict(X_test)
```

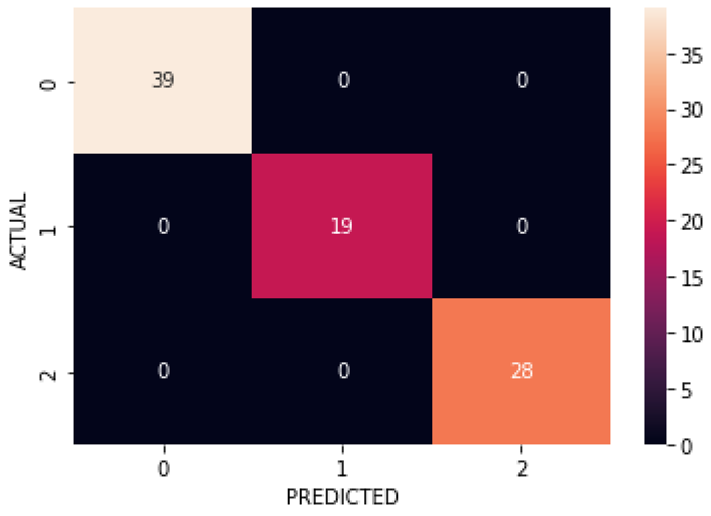
```
print('Accuracy : ', accuracy_score(Y_test, pred))
print('F1 Score : ', f1_score(Y_test, pred, average = 'weighted'))
```

```
Accuracy :  1.0
F1 Score :  1.0
```

Confusion matrix:



```
confusion_matrix= pd.crosstab(Y_test,pred, rownames=['ACTUAL'], colnames=['PREDICTED'])
ax = sns.heatmap(confusion_matrix,annot=True)
plt.show()
```

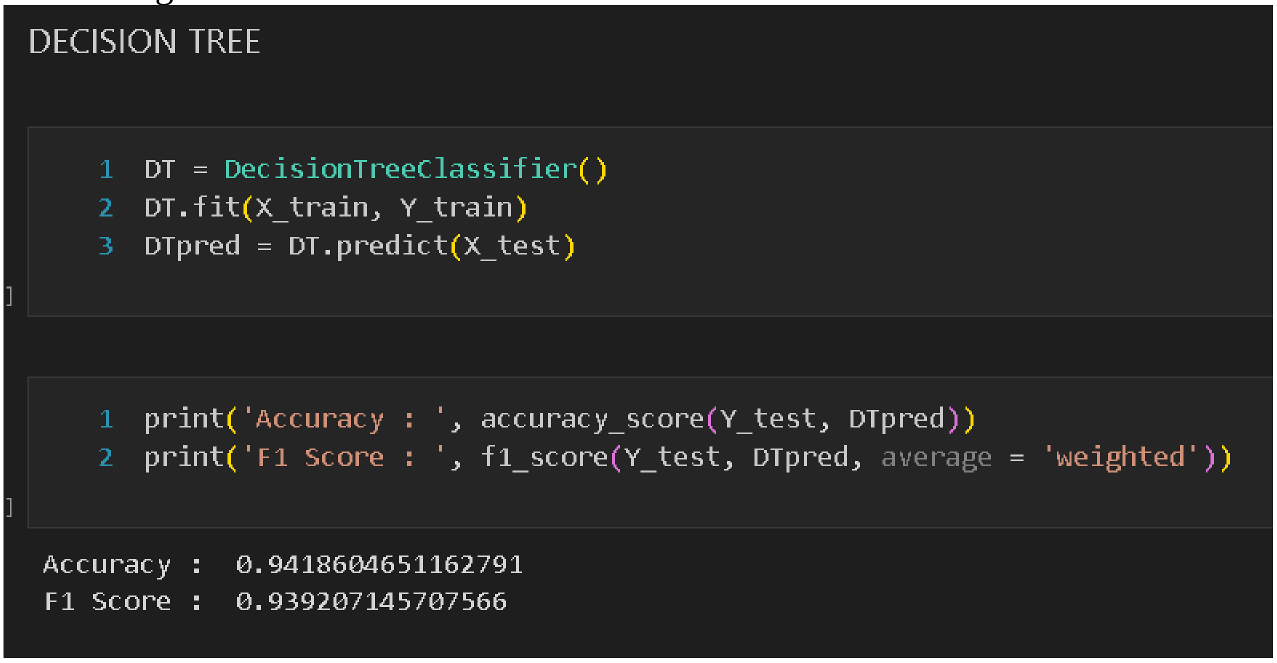


4.3 Decision Tree

Decision tree is a machine learning algorithm used for classification and regression tasks. It is a type of supervised learning algorithm that is often used in predictive modeling.

In decision tree, the data is split into smaller and smaller subsets based on the values of different variables or features. The goal is to create a tree-like model of decisions and their possible consequences. Each internal node of the tree represents a test on a feature, each branch represents the outcome of the test, and each leaf node represents a class label or a numerical value that is assigned to the instance.

Performing Decision tree Classification:



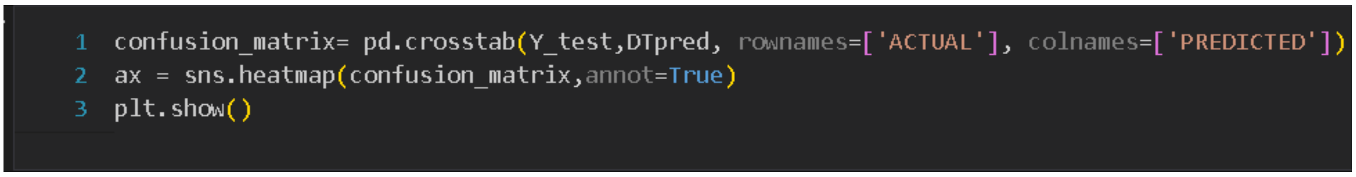
DECISION TREE

```
DT = DecisionTreeClassifier()
DT.fit(X_train, Y_train)
DTpred = DT.predict(X_test)
```

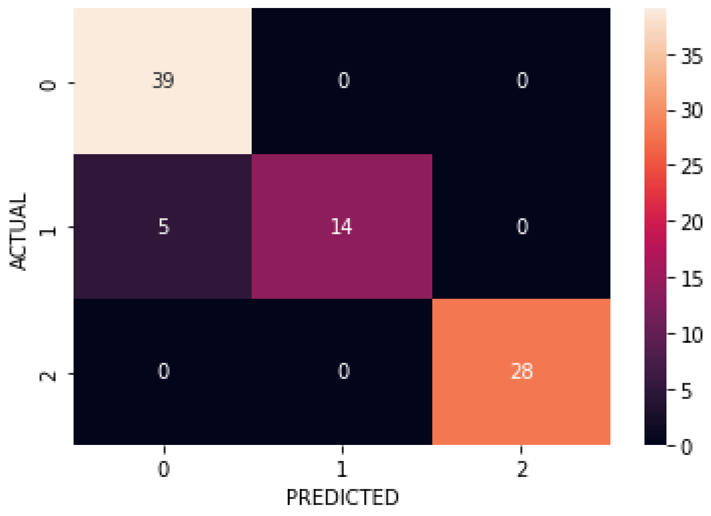
```
print('Accuracy : ', accuracy_score(Y_test, DTpred))
print('F1 Score : ', f1_score(Y_test, DTpred, average = 'weighted'))
```

```
Accuracy :  0.9418604651162791
F1 Score :  0.939207145707566
```

Confusion Matrix:



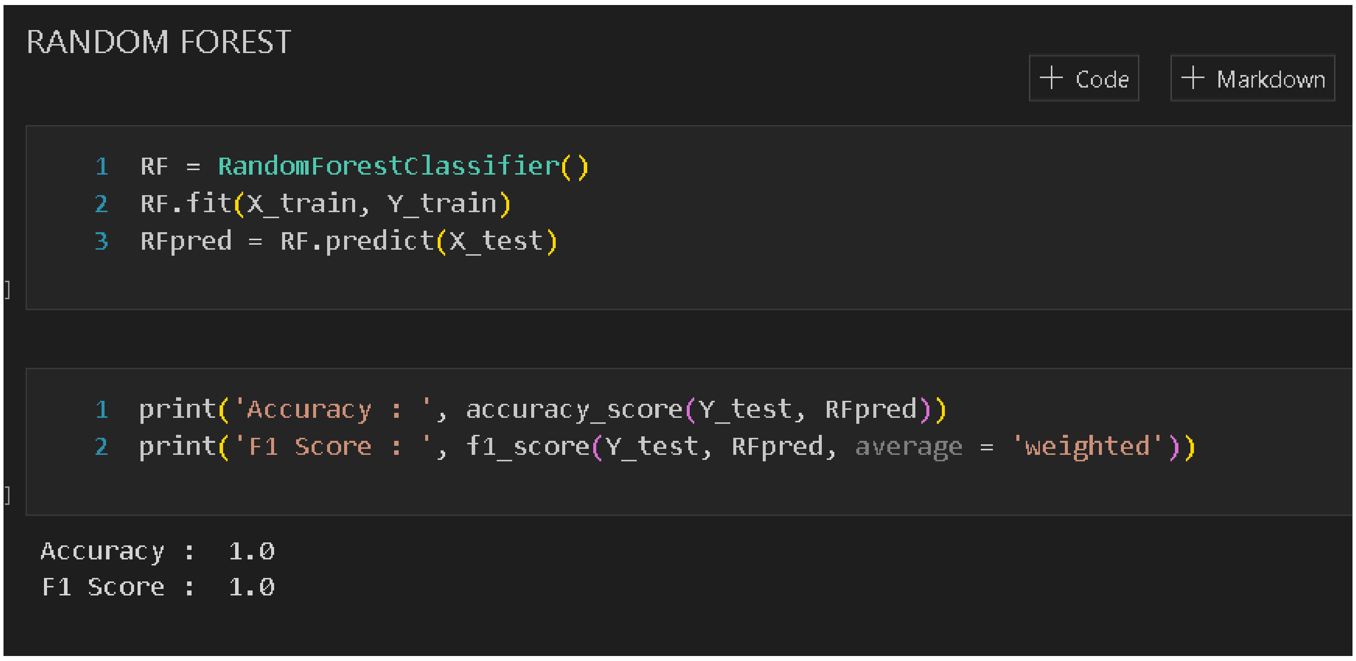
```
confusion_matrix= pd.crosstab(Y_test,DTpred, rownames=['ACTUAL'], colnames=['PREDICTED'])
ax = sns.heatmap(confusion_matrix,annot=True)
plt.show()
```



4.4 RANDOM FOREST

Random forest is a popular ensemble learning method in machine learning for classification, regression and other tasks. It is based on the concept of decision trees, but instead of using a single decision tree, it builds a large number of trees and combines their predictions to make a final prediction.

Performing Random Forest Classification:

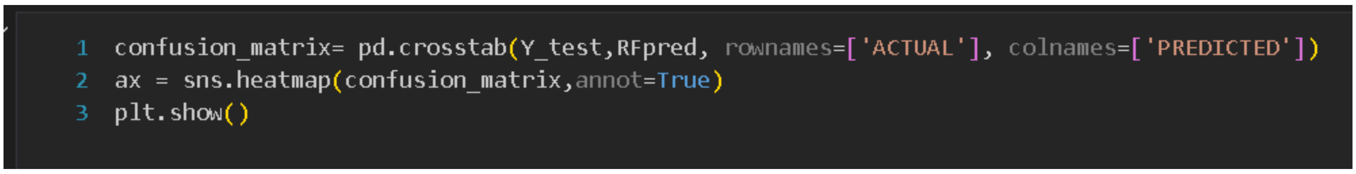


```
RF = RandomForestClassifier()
RF.fit(X_train, Y_train)
RFpred = RF.predict(X_test)
```

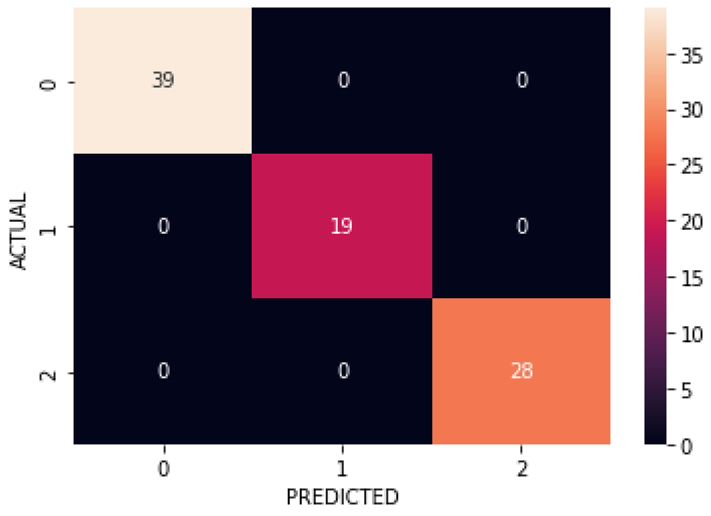
```
print('Accuracy : ', accuracy_score(Y_test, RFpred))
print('F1 Score : ', f1_score(Y_test, RFpred, average = 'weighted'))
```

```
Accuracy :  1.0
F1 Score :  1.0
```

Confusion Matrix:



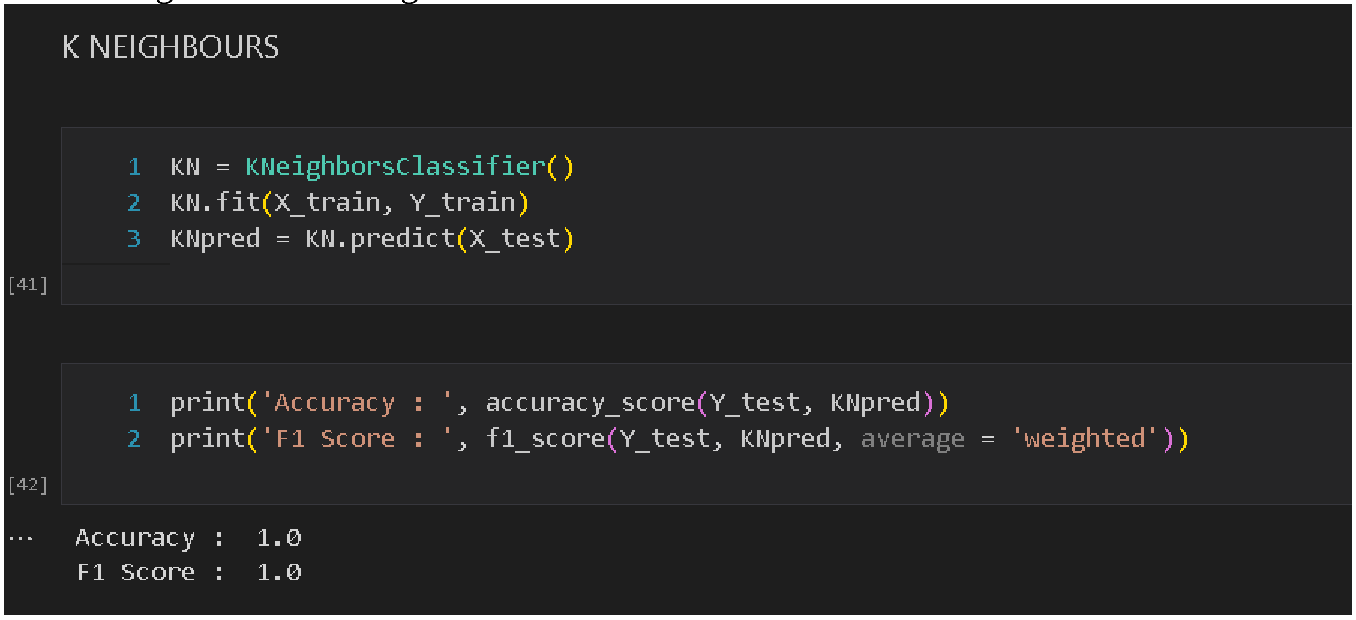
```
confusion_matrix= pd.crosstab(Y_test,RFpred, rownames=['ACTUAL'], colnames=['PREDICTED'])
ax = sns.heatmap(confusion_matrix,annot=True)
plt.show()
```



4.5 K-NEAREST NEIBOUR

K-Nearest Neighbor (KNN) is a popular machine learning algorithm for classification and regression tasks. It is a non-parametric algorithm, meaning that it does not make any assumptions about the underlying distribution of the data. Instead, it works by finding the K-nearest neighbors of a given data point and using their labels to predict the label of the new data point.

Performing K-Nearest Neighbors classification:

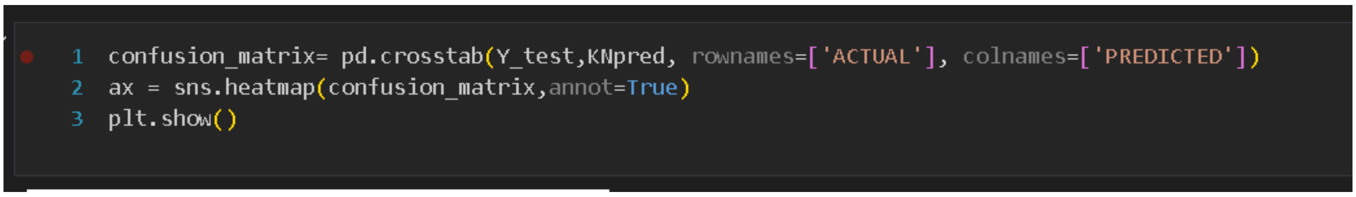

K NEIGHBOURS

```
KN = KNeighborsClassifier()
KN.fit(X_train, Y_train)
KNpred = KN.predict(X_test)
```
[41]

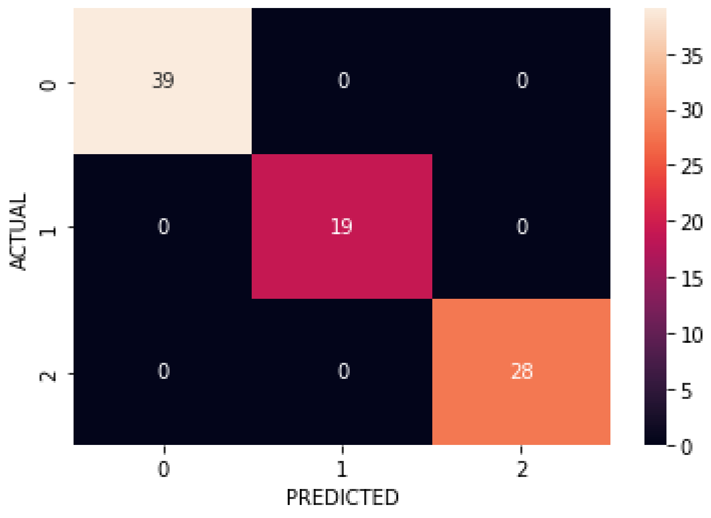
```
print('Accuracy : ', accuracy_score(Y_test, KNpred))
print('F1 Score : ', f1_score(Y_test, KNpred, average = 'weighted'))
```
[42]

```
Accuracy :  1.0
F1 Score :  1.0
```

Confusion Matrix:

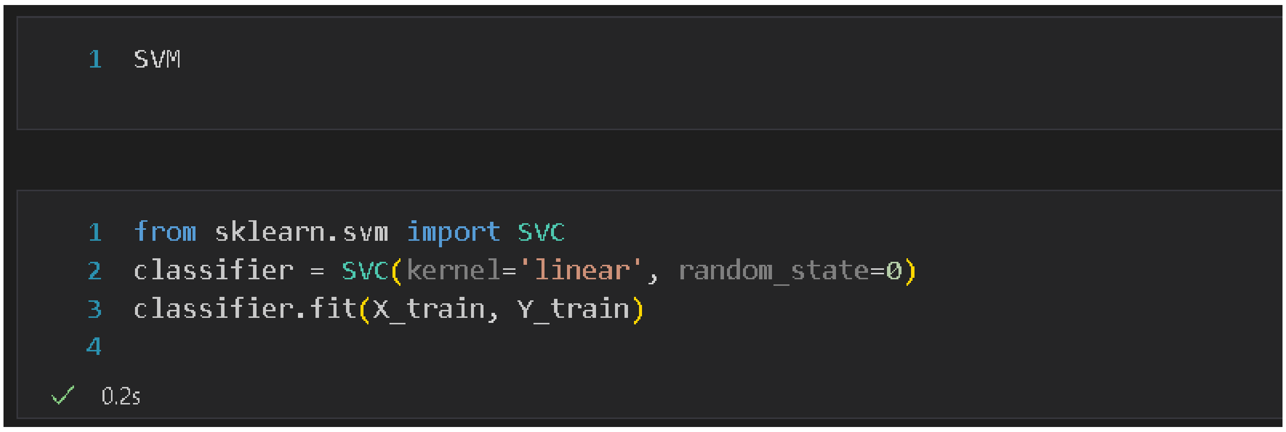


```
confusion_matrix= pd.crosstab(Y_test,KNpred, rownames=['ACTUAL'], colnames=['PREDICTED'])
ax = sns.heatmap(confusion_matrix,annot=True)
plt.show()
```



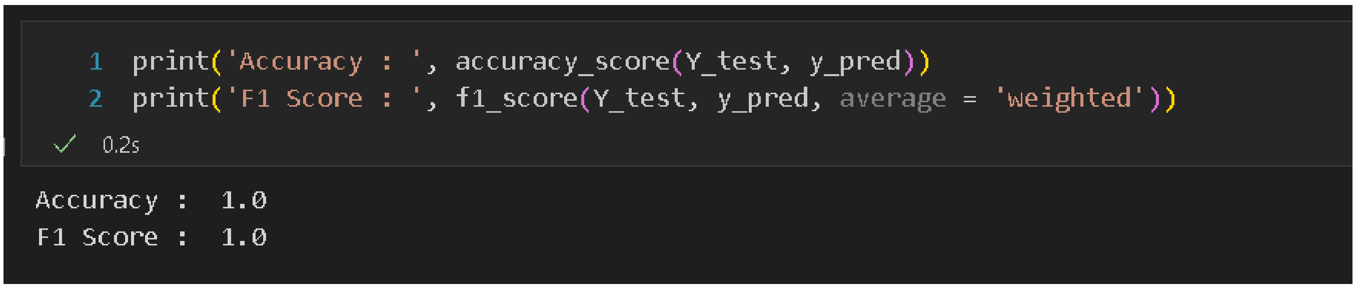
4.6 SUPPORT VECTOR MACHINE

Support Vector Machines (SVMs) are a popular type of supervised learning algorithm that can be used for classification, regression, and outlier detection. SVMs work by finding the hyperplane that maximally separates the two classes in a given dataset. The hyperplane is chosen in such a way that the margin between the two classes is maximized.



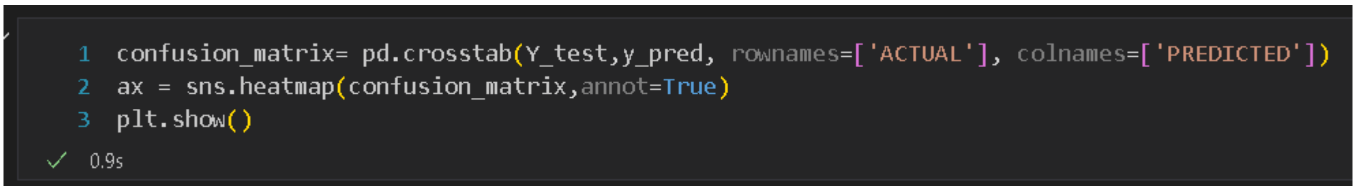
```
SVM
```

```
from sklearn.svm import SVC
classifier = SVC(kernel='linear', random_state=0)
classifier.fit(X_train, Y_train)

✓ 0.2s
```

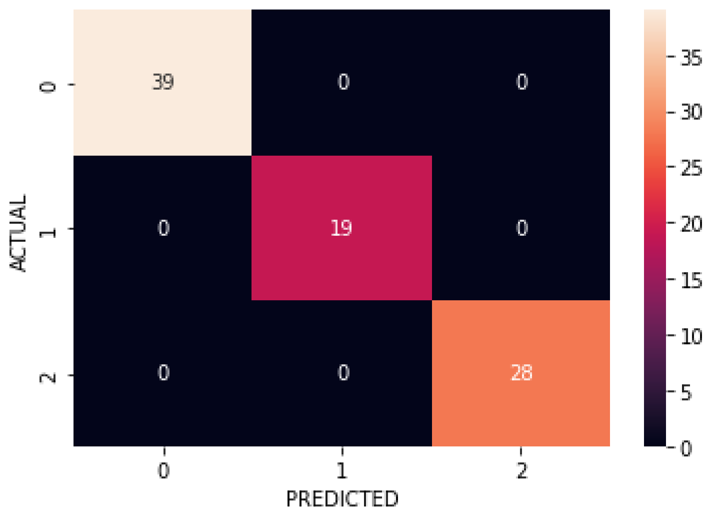


```
print('Accuracy : ', accuracy_score(Y_test, y_pred))
print('F1 Score : ', f1_score(Y_test, y_pred, average = 'weighted'))
✓ 0.2s

Accuracy :  1.0
F1 Score :  1.0
```

Confusion Matrix:



```
confusion_matrix= pd.crosstab(Y_test,y_pred, rownames=['ACTUAL'], colnames=['PREDICTED'])
ax = sns.heatmap(confusion_matrix,annot=True)
plt.show()
✓ 0.9s
```



CODE:

```
# -*- coding: utf-8 -*-
"""penguins.ipynb
Automatically generated by Colaboratory.
Original file is located at
    https://colab.research.google.com/drive/15idGUpwDTz3v-Phk1csKQPsOyk9dFUM7
"""
# Commented out IPython magic to ensure Python compatibility.
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
# %matplotlib inline
penguins = sns.load_dataset('penguins')
penguins.head()
"""##DATASET DESCRIPTION:
NAME: Palmer Archipelago (Antarctica) penguin data
ATTRIPUTES:
species: penguin species (Chinstrap, Adélie, or Gentoo)
culmen_length_mm: culmen length (mm)
culmen_depth_mm: culmen depth (mm)
flipper_length_mm: flipper length (mm)
body_mass_g: body mass (g)
```

```
island: island name (Dream, Torgersen, or Biscoe) in the Palmer Archipelago (Antarctica)

sex: penguin sex

"""

penguins.describe()

bill_length_mean = penguins["bill_length_mm"].mean().round(1)

print(f"THE MEAN OF BILL LENGTH IS : {bill_length_mean}")

bill_depth_mean = penguins["bill_depth_mm"].mean().round(1)

print(f"THE MEAN OF BILL DEPTH IS : {bill_depth_mean}")

flipper_length_mean = penguins["flipper_length_mm"].mean().round(1)

print(f"THE MEAN OF FLIPPER LENGTH IS : {flipper_length_mean}")

body_mass_mean = penguins["body_mass_g"].mean().round(1)

print(f"THE MEAN OF BODY MASS IS : {body_mass_mean}")

bill_length_median = penguins["bill_length_mm"].median()

print(f"THE MEDIAN OF BILL LENGTH IS : {bill_length_median}")

bill_depth_median = penguins["bill_depth_mm"].median()

print(f"THE MEDIAN OF BILL DEPTH IS : {bill_depth_median}")

flipper_length_median = penguins["flipper_length_mm"].median()

print(f"THE MEDIAN OF FLIPPER LENGTH IS : {flipper_length_median}")

body_mass_median = penguins["body_mass_g"].median()

print(f"THE MEDIAN OF BODY MASS IS : {body_mass_median}")

penguins.isnull().sum()

penguins_up = penguins

penguins_up['bill_length_mm'] = penguins['bill_length_mm'].fillna(value =
penguins['bill_length_mm'].mean())

penguins_up['bill_depth_mm'] = penguins['bill_depth_mm'].fillna(value =
penguins['bill_depth_mm'].mean())

penguins_up['flipper_length_mm'] = penguins['flipper_length_mm'].fillna(value =
penguins['flipper_length_mm'].mean())

penguins_up['body_mass_g'] = penguins['body_mass_g'].fillna(value =
penguins['body_mass_g'].mean())

penguins_up['sex'] = penguins['sex'].fillna(value = penguins['sex'].mode()[0])

penguins_up.isnull().sum()

penguins_up.round(1)

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

plt.show()

import scipy as sp

from scipy.stats import chi2_contingency

contigency= pd.crosstab(penguins_up['sex'], penguins_up['species'])

contigency

c, p, dof, expected = chi2_contingency(contigency)

print(f"P-Value : {p}")

print(f"Chi2-Value : {c}")

print(f"dof-Value : {dof}")

if(c<sp.stats.chi2.isf(0.05, dof, loc=0, scale=1)):

    print(f"Since c({c.round(2)}) < Significant value({sp.stats.chi2.isf(0.05, dof, loc=0,
scale=1).round(2)}), Accept Null hypothesis")

else:
    print(f"Since c > Significant value({sp.stats.chi2.isf(0.05, dof, loc=0, scale=1)}),
Reject Null hypothesis")

chinstraps = penguins_up[penguins_up['species'] == 'Chinstrap']

adelies = penguins_up[penguins_up['species'] == 'Adelie']

gentoos = penguins_up[penguins_up['species'] == 'Gentoo']

alpha = 0.05

from statsmodels.stats.weightstats import ztest as ztest

ztest_score1,p_value1 = ztest(adelies['flipper_length_mm'],
chinstraps['flipper_length_mm'], value=0)

print(f"p-value = {p_value1}")

if(p_value1 <  alpha):

  print("Reject Null Hypothesis")

else:

  print("Fail to Reject NUll Hypothesis")

from statsmodels.stats.weightstats import ztest as ztest

ztest_score2,p_value2 = ztest(adelies['flipper_length_mm'],
gentoos['flipper_length_mm'], value=0)

print(f"p-value = {p_value2}")

if(p_value2 <  alpha):

  print("Reject Null Hypothesis")

else:

  print("Fail to Reject NUll Hypothesis")

from statsmodels.stats.weightstats import ztest as ztest

ztest_score3,p_value3 = ztest(gentoos['flipper_length_mm'],
chinstraps['flipper_length_mm'], value=0)

print(f"p-value = {p_value3}")

if(p_value3 <  alpha):

  print("Reject Null Hypothesis")

else:

  print("Fail to Reject NUll Hypothesis")

from sklearn.preprocessing import MinMaxScaler

mms = MinMaxScaler()

penguins_up['bill_length_mm'] =
mms.fit_transform(penguins_up['bill_length_mm'].values.reshape(-1, 1))

penguins_up['bill_depth_mm'] =
mms.fit_transform(penguins_up['bill_depth_mm'].values.reshape(-1, 1))

penguins_up['flipper_length_mm'] =
mms.fit_transform(penguins_up['flipper_length_mm'].values.reshape(-1, 1))

penguins_up['body_mass_g'] =
mms.fit_transform(penguins_up['body_mass_g'].values.reshape(-1, 1))

penguins_up.head()

penguins_up.describe()

new_df_dummy = pd.get_dummies(penguins_up, columns = ['sex', 'island'], drop_first =
True)

new_df_dummy['species'].replace({'Adelie' : 0,

                                 'Chinstrap' : 1,

                                 'Gentoo': 2}, inplace = True)


sns.heatmap(new_df_dummy.corr(), annot = True, cmap = 'Blues')
```

```
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

new_df_dummy

X = new_df_dummy.drop(columns = ['species','sex_Male'])

Y = new_df_dummy['species']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.25, random_state = 123)

"""LOGISTIC REGRESSION"""

LR = LogisticRegression()

LR.fit(X_train, Y_train)

pred = LR.predict(X_test)

print('Accuracy : ', accuracy_score(Y_test, pred))

print('F1 Score : ', f1_score(Y_test, pred, average = 'weighted'))

confusion_matrix= pd.crosstab(Y_test,pred, rownames=['ACTUAL'], colnames=['PREDICTED'])

ax = sns.heatmap(confusion_matrix,annot=True)

plt.show()
```

```
"""RANDOM FOREST"""

RF = RandomForestClassifier()

RF.fit(X_train, Y_train)

RFpred = RF.predict(X_test)

print('Accuracy : ', accuracy_score(Y_test, RFpred))

print('F1 Score : ', f1_score(Y_test, RFpred, average = 'weighted'))

confusion_matrix= pd.crosstab(Y_test,RFpred, rownames=['ACTUAL'],
colnames=['PREDICTED'])

ax = sns.heatmap(confusion_matrix,annot=True)

plt.show()

"""DECISION TREE"""

DT = DecisionTreeClassifier()

DT.fit(X_train, Y_train)

DTpred = DT.predict(X_test)

print('Accuracy : ', accuracy_score(Y_test, DTpred))

print('F1 Score : ', f1_score(Y_test, DTpred, average = 'weighted'))

confusion_matrix= pd.crosstab(Y_test,DTpred, rownames=['ACTUAL'],
colnames=['PREDICTED'])

ax = sns.heatmap(confusion_matrix,annot=True)

plt.show()

"""K NEIGHBOURS"""

KN = KNeighborsClassifier()

KN.fit(X_train, Y_train)

KNpred = KN.predict(X_test)
```

```
print('Accuracy : ', accuracy_score(Y_test, KNpred))

print('F1 Score : ', f1_score(Y_test, KNpred, average = 'weighted'))

confusion_matrix= pd.crosstab(Y_test,KNpred, rownames=['ACTUAL'],
colnames=['PREDICTED'])

ax = sns.heatmap(confusion_matrix,annot=True)

plt.show()

from sklearn.decomposition import PCA

pca=PCA(n_components=2)

pca.fit(X_train)

x_pca=pca.transform(X_train)

X_train.shape

x_pca.shape

plt.figure(figsize=(8,6))

plt.scatter(x_pca[:,0],x_pca[:,1])

plt.xlabel('First principle component')

plt.ylabel('Second principle component')

from sklearn.cluster import KMeans

model = KMeans(n_clusters=3)

model.fit(X)

labels = model.predict(X)

matrix = pd.DataFrame({'labels': labels, 'species': Y})

ct = pd.crosstab(matrix['labels'], matrix['species'])

print(ct)
```

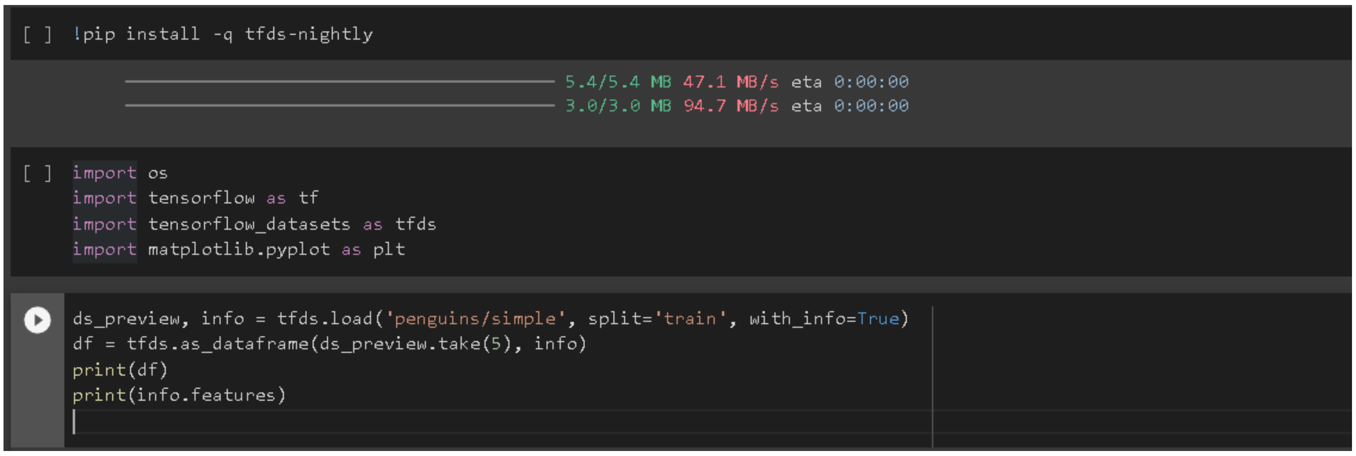
```
plt.scatter(X[Y == 0,0], X[Y==0,1], s = 15, c= 'red', label = 'Cluster_1')

plt.scatter(X[Y == 1,0], X[Y==1,1], s = 15, c= 'blue', label = 'Cluster_2')

plt.scatter(X[Y == 2,0], X[Y==2,1], s = 15, c= 'green', label = 'Cluster_3')

plt.scatter(model.cluster_centers_[:,0], model.cluster_centers_[:,1], s = 25, c = 'yellow',
label = 'Centroids')

plt.legend()

plt.show()

color_theme  = np.array(['darakgrey','lightsalmon','red'])

plt.scatter(x=new_df_dummy.bill_length_mm,y=new_df_dummy.species,c=
color_theme[model.labels_],s=50)

plt.title('K-Means')

X.shape

labels.shape
```

4.7 ARTIFICIAL NEURAL NETWORK

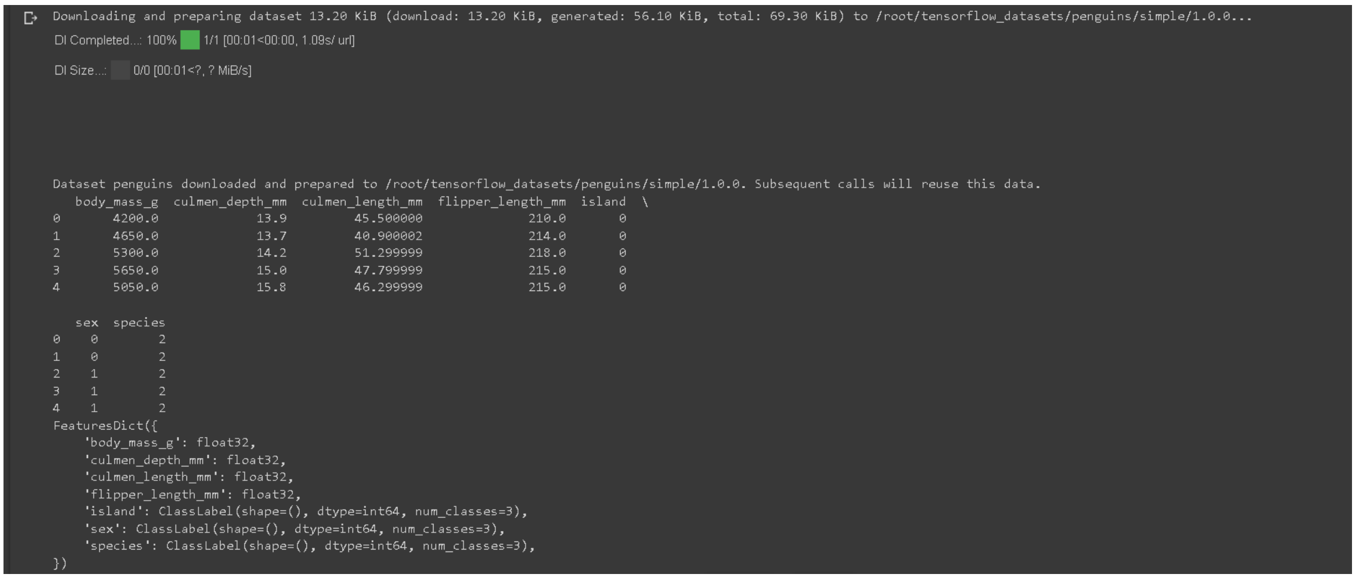
Artificial Neural Networks (ANNs) are a type of machine learning algorithm inspired by the structure and function of the human brain. ANNs consist of layers of interconnected artificial neurons that process information and make predictions.

Each artificial neuron in an ANN receives input signals from the neurons in the previous layer. These input signals are weighted, and the neuron applies an activation function to the weighted sum of the inputs. The output of the neuron is then passed to the neurons in the next layer, and the process repeats until the final layer produces the prediction.

Loading dataset:

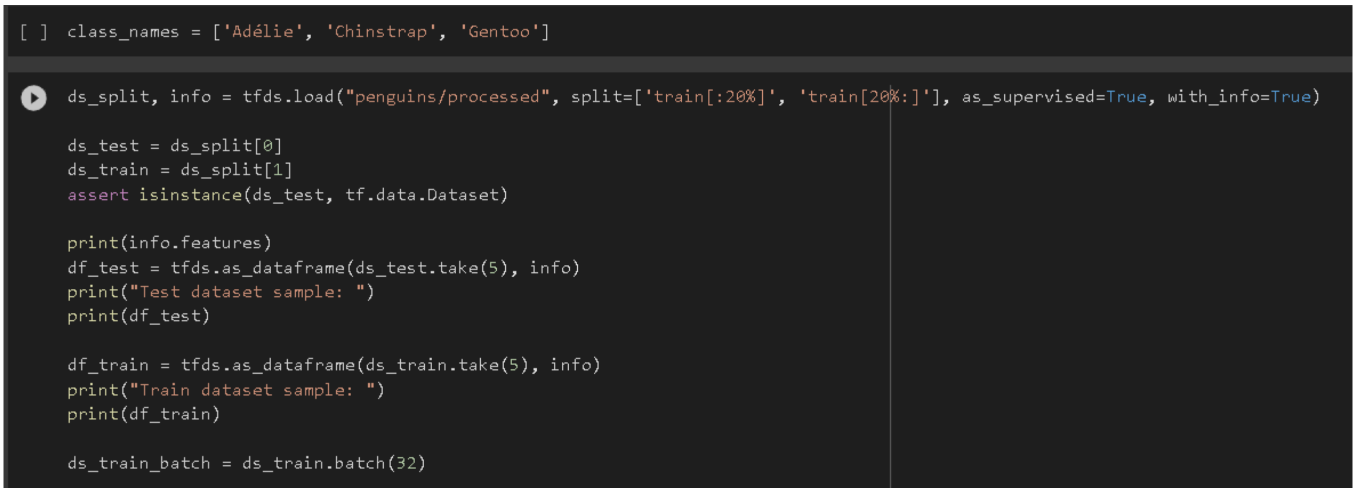


```
[ ] !pip install -q tfds-nightly
        ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.4/5.4 MB 47.1 MB/s eta 0:00:00
        ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.0/3.0 MB 94.7 MB/s eta 0:00:00

[ ] import os
    import tensorflow as tf
    import tensorflow_datasets as tfds
    import matplotlib.pyplot as plt

ds_preview, info = tfds.load('penguins/simple', split='train', with_info=True)
df = tfds.as_dataframe(ds_preview.take(5), info)
print(df)
print(info.features)
```



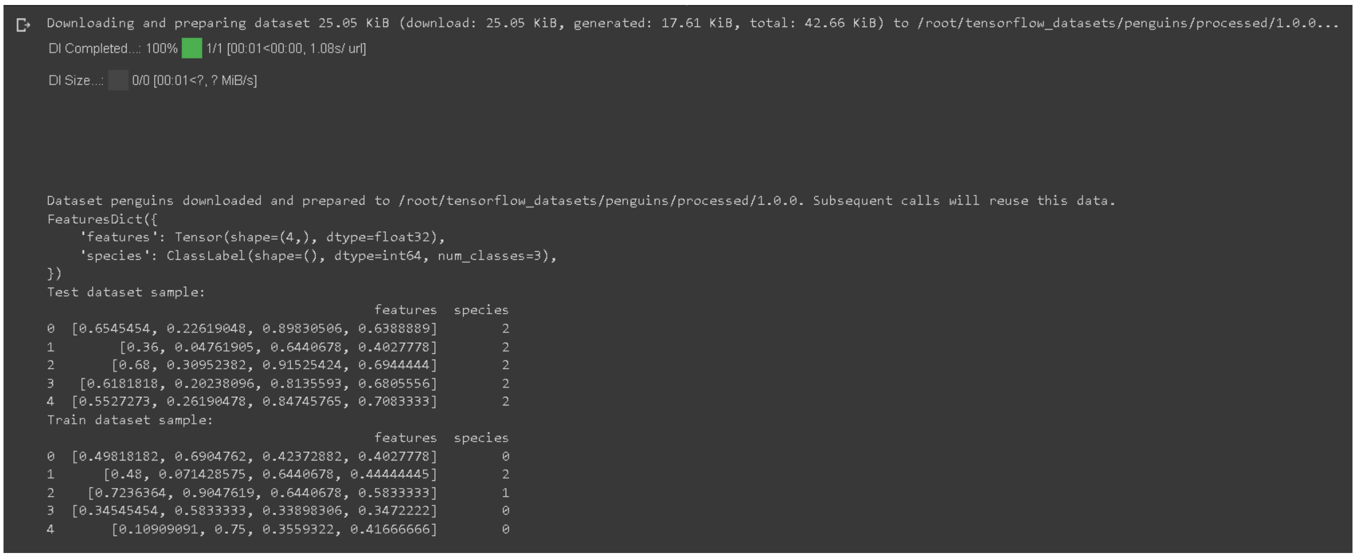
```
Downloading and preparing dataset 13.20 KiB (download: 13.20 KiB, generated: 56.10 KiB, total: 69.30 KiB) to /root/tensorflow_datasets/penguins/simple/1.0.0...
 Dl Completed...: 100% 🟩 1/1 [00:01<00:00, 1.09s/ url]

 Dl Size...:  0/0 [00:01<?, ? MiB/s]




Dataset penguins downloaded and prepared to /root/tensorflow_datasets/penguins/simple/1.0.0. Subsequent calls will reuse this data.
   body_mass_g  culmen_depth_mm  culmen_length_mm  flipper_length_mm  island  \
0       4200.0             13.9         45.500000              210.0       0
1       4650.0             13.7         40.900002              214.0       0
2       5300.0             14.2         51.299999              218.0       0
3       5650.0             15.0         47.799999              215.0       0
4       5050.0             15.8         46.299999              215.0       0

   sex  species
0    0        2
1    0        2
2    1        2
3    1        2
4    1        2
FeaturesDict({
    'body_mass_g': float32,
    'culmen_depth_mm': float32,
    'culmen_length_mm': float32,
    'flipper_length_mm': float32,
    'island': ClassLabel(shape=(), dtype=int64, num_classes=3),
    'sex': ClassLabel(shape=(), dtype=int64, num_classes=3),
    'species': ClassLabel(shape=(), dtype=int64, num_classes=3),
})
```

Splitting:

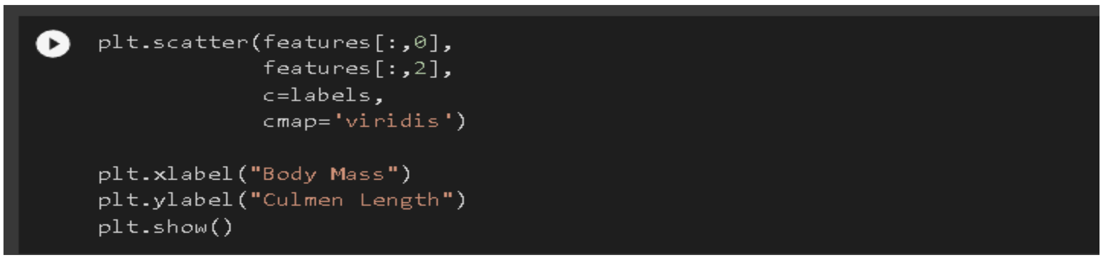


```
[ ] class_names = ['Adélie', 'Chinstrap', 'Gentoo']
```

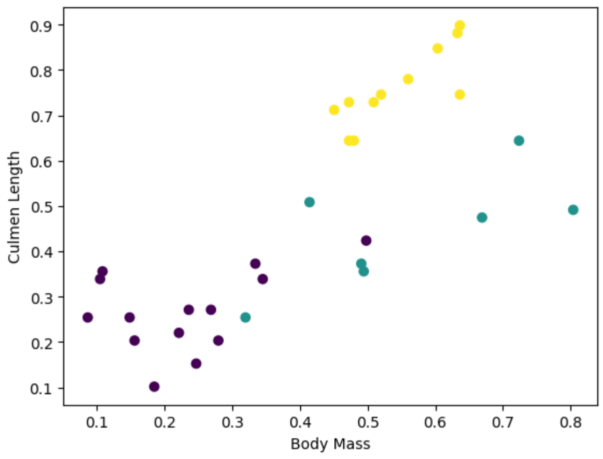
```
ds_split, info = tfds.load("penguins/processed", split=['train[:20%]', 'train[20%:]'], as_supervised=True, with_info=True)

ds_test = ds_split[0]
ds_train = ds_split[1]
assert isinstance(ds_test, tf.data.Dataset)

print(info.features)
df_test = tfds.as_dataframe(ds_test.take(5), info)
print("Test dataset sample: ")
print(df_test)

df_train = tfds.as_dataframe(ds_train.take(5), info)
print("Train dataset sample: ")
print(df_train)

ds_train_batch = ds_train.batch(32)
```



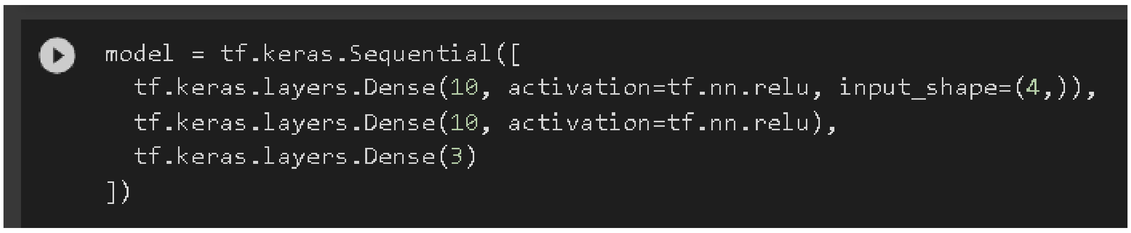
```
Downloading and preparing dataset 25.05 KiB (download: 25.05 KiB, generated: 17.61 KiB, total: 42.66 KiB) to /root/tensorflow_datasets/penguins/processed/1.0.0...
 Dl Completed...: 100% 🟩 1/1 [00:01<00:00, 1.08s/ url]

 Dl Size...: 0/0 [00:01<?, ? MiB/s]




Dataset penguins downloaded and prepared to /root/tensorflow_datasets/penguins/processed/1.0.0. Subsequent calls will reuse this data.
FeaturesDict({
    'features': Tensor(shape=(4,), dtype=float32),
    'species': ClassLabel(shape=(), dtype=int64, num_classes=3),
})
Test dataset sample:
                                         features  species
0  [0.6545454, 0.22619048, 0.89830506, 0.6388889]        2
1        [0.36, 0.04761905, 0.6440678, 0.4027778]        2
2       [0.68, 0.30952382, 0.91525424, 0.6944444]        2
3   [0.6181818, 0.20238096, 0.8135593, 0.6805556]        2
4  [0.5527273, 0.26190478, 0.84745765, 0.7083333]        2
Train dataset sample:
                                         features  species
0  [0.49818182, 0.6904762, 0.42372882, 0.4027778]        0
1      [0.48, 0.071428575, 0.6440678, 0.44444445]        2
2    [0.7236364, 0.9047619, 0.6440678, 0.5833333]        1
3  [0.34545454, 0.5833333, 0.33898306, 0.3472222]        0
4       [0.10909091, 0.75, 0.3559322, 0.41666666]        0
```



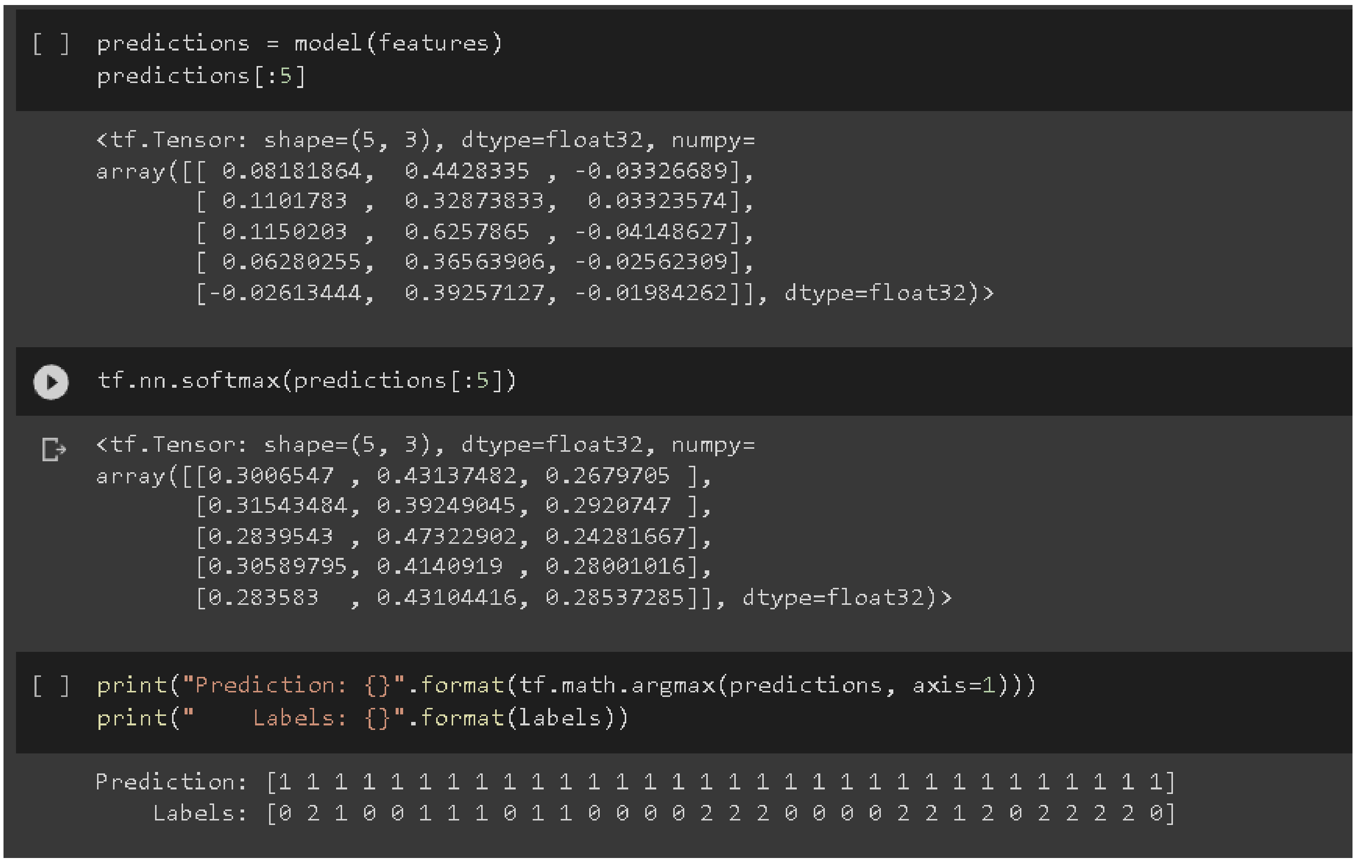
```
plt.scatter(features[:,0],
            features[:,2],
            c=labels,
            cmap='viridis')

plt.xlabel("Body Mass")
plt.ylabel("Culmen Length")
plt.show()
```



Model:



```
model = tf.keras.Sequential([
  tf.keras.layers.Dense(10, activation=tf.nn.relu, input_shape=(4,)),
  tf.keras.layers.Dense(10, activation=tf.nn.relu),
  tf.keras.layers.Dense(3)
])
```



```
[ ]  predictions = model(features)
     predictions[:5]
```

```
<tf.Tensor: shape=(5, 3), dtype=float32, numpy=
array([[ 0.08181864,  0.4428335 , -0.03326689],
       [ 0.1101783 ,  0.32873833,  0.03323574],
       [ 0.1150203 ,  0.6257865 , -0.04148627],
       [ 0.06280255,  0.36563906, -0.02562309],
       [-0.02613444,  0.39257127, -0.01984262]], dtype=float32)>
```

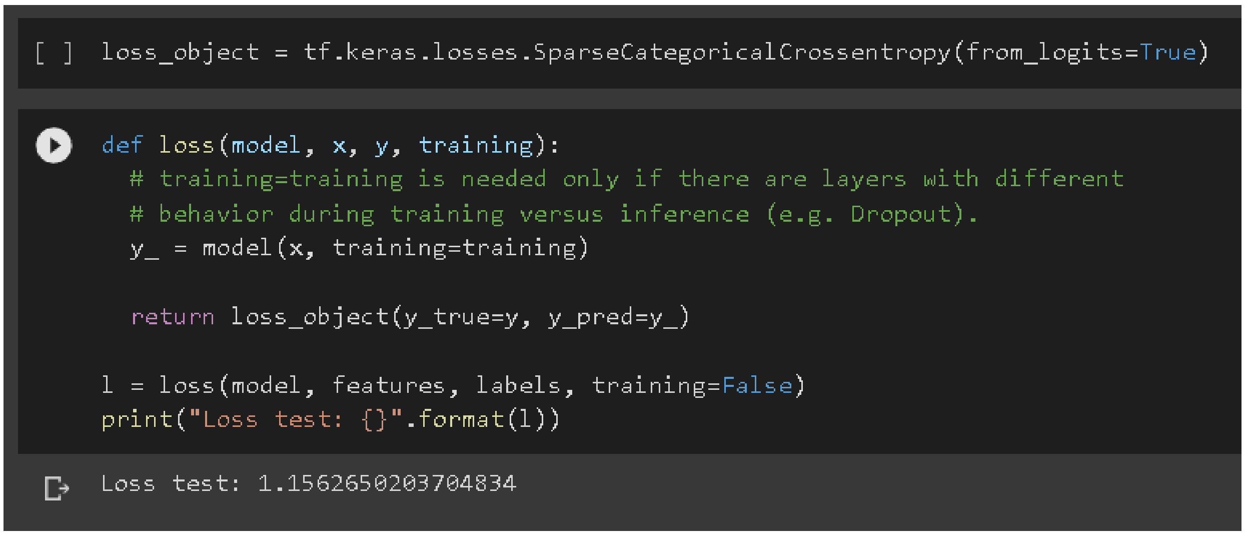
```
tf.nn.softmax(predictions[:5])
```

```
<tf.Tensor: shape=(5, 3), dtype=float32, numpy=
array([[0.3006547 , 0.43137482, 0.2679705 ],
       [0.31543484, 0.39249045, 0.2920747 ],
       [0.2839543 , 0.47322902, 0.24281667],
       [0.30589795, 0.4140919 , 0.28001016],
       [0.283583  , 0.43104416, 0.28537285]], dtype=float32)>
```

```
[ ]  print("Prediction: {}".format(tf.math.argmax(predictions, axis=1)))
     print("    Labels: {}".format(labels))
```

```
Prediction: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
    Labels: [0 2 1 0 0 1 1 1 0 1 1 0 0 0 0 2 2 2 0 0 0 0 2 2 1 2 0 2 2 2 2 0]
```

Loss Function:

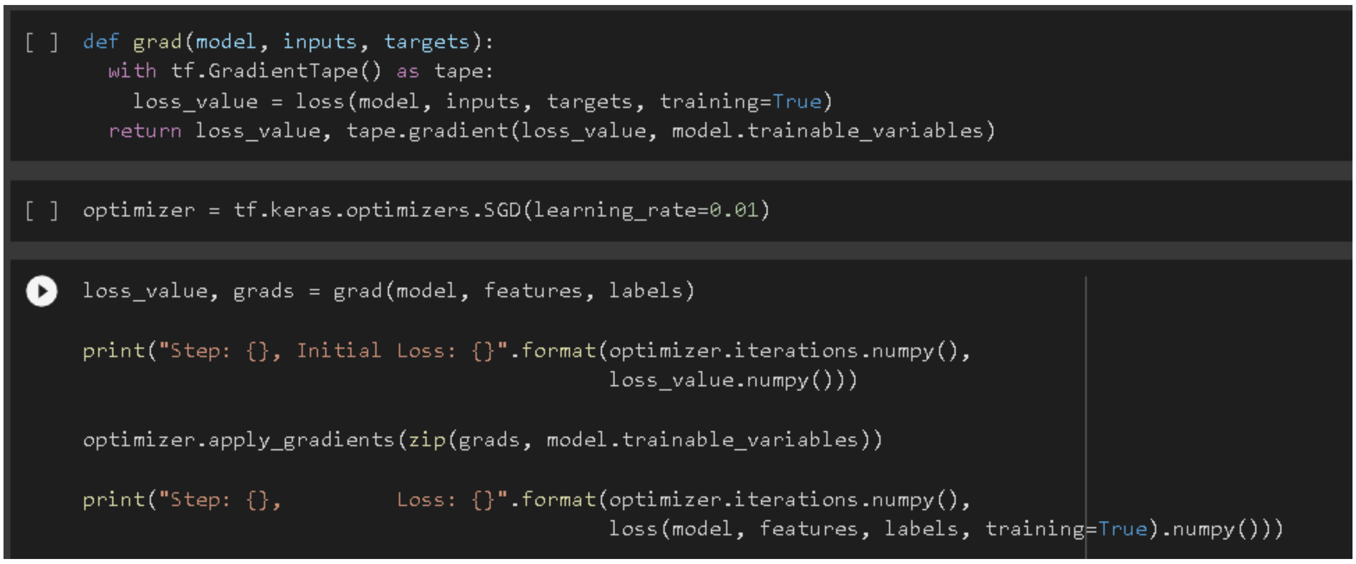


```
[ ]  loss_object = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
```

```
def loss(model, x, y, training):
  # training=training is needed only if there are layers with different
  # behavior during training versus inference (e.g. Dropout).
  y_ = model(x, training=training)

  return loss_object(y_true=y, y_pred=y_)

l = loss(model, features, labels, training=False)
print("Loss test: {}".format(l))
```

```
Loss test: 1.1562650203704834
```

Optimization:

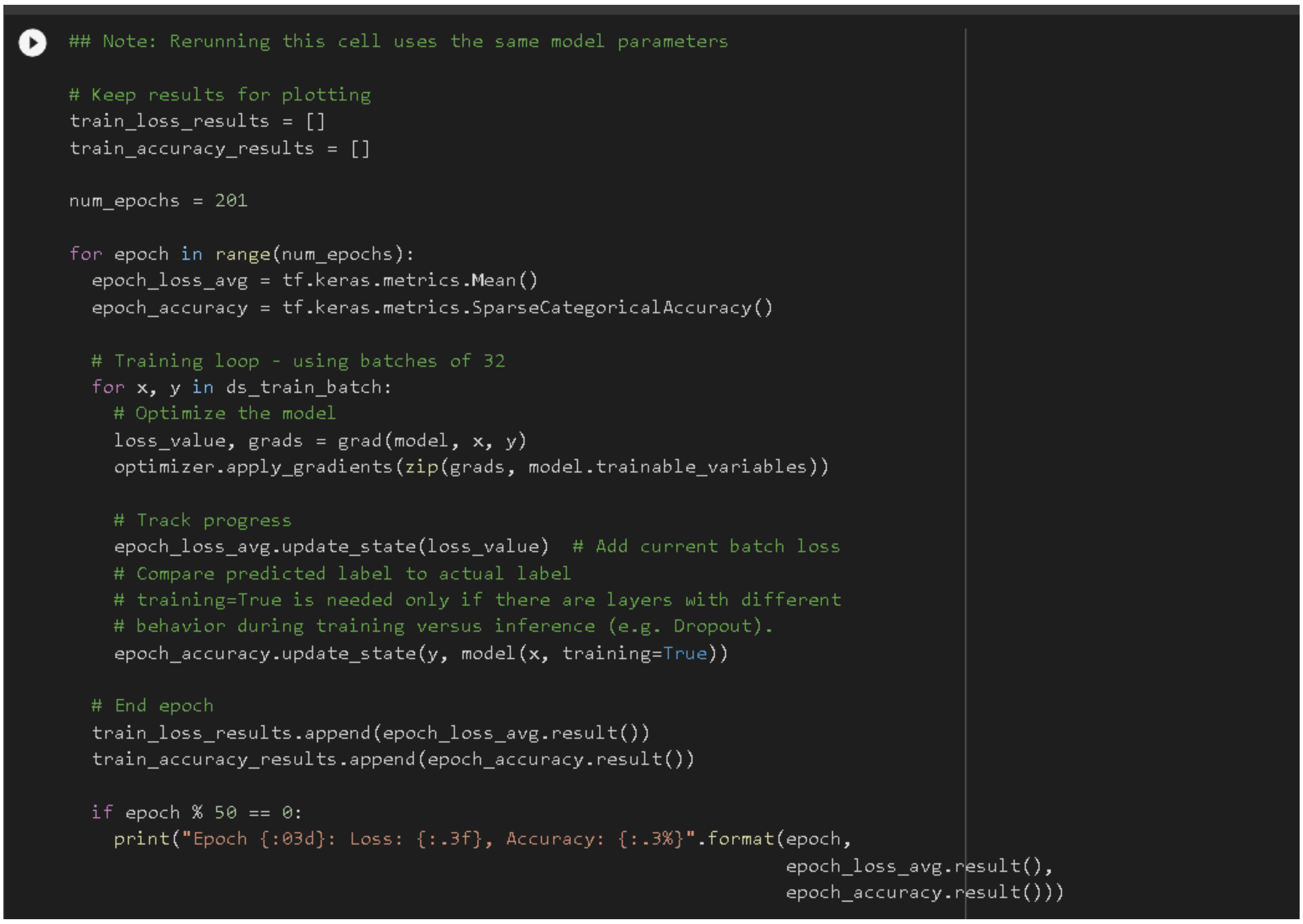


```
[ ] def grad(model, inputs, targets):
      with tf.GradientTape() as tape:
        loss_value = loss(model, inputs, targets, training=True)
      return loss_value, tape.gradient(loss_value, model.trainable_variables)
```

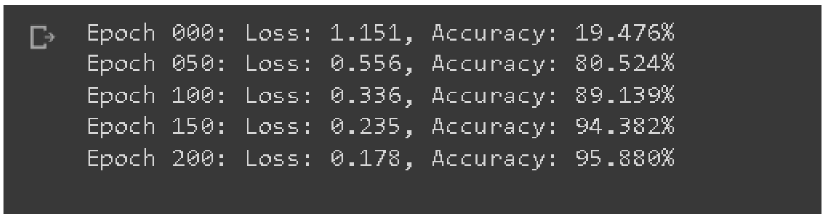
```
[ ] optimizer = tf.keras.optimizers.SGD(learning_rate=0.01)
```

```
loss_value, grads = grad(model, features, labels)

print("Step: {}, Initial Loss: {}".format(optimizer.iterations.numpy(),
                                          loss_value.numpy()))

optimizer.apply_gradients(zip(grads, model.trainable_variables))

print("Step: {},         Loss: {}".format(optimizer.iterations.numpy(),
                                          loss(model, features, labels, training=True).numpy()))
```

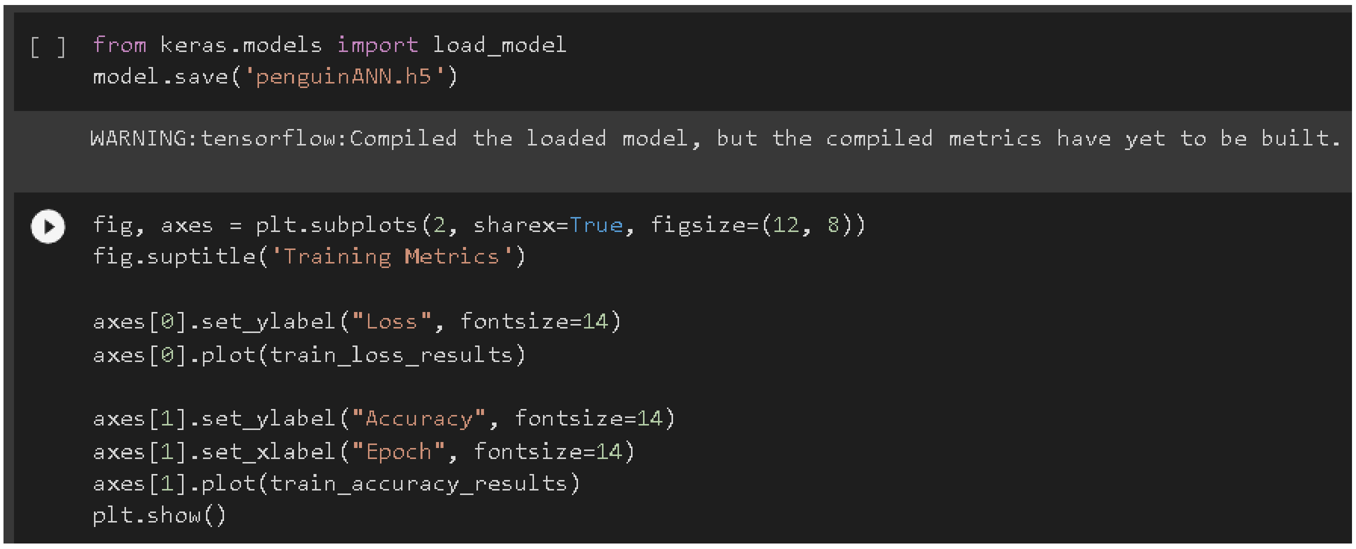
Training:



```
## Note: Rerunning this cell uses the same model parameters

# Keep results for plotting
train_loss_results = []
train_accuracy_results = []

num_epochs = 201

for epoch in range(num_epochs):
  epoch_loss_avg = tf.keras.metrics.Mean()
  epoch_accuracy = tf.keras.metrics.SparseCategoricalAccuracy()

  # Training loop - using batches of 32
  for x, y in ds_train_batch:
    # Optimize the model
    loss_value, grads = grad(model, x, y)
    optimizer.apply_gradients(zip(grads, model.trainable_variables))

    # Track progress
    epoch_loss_avg.update_state(loss_value)  # Add current batch loss
    # Compare predicted label to actual label
    # training=True is needed only if there are layers with different
    # behavior during training versus inference (e.g. Dropout).
    epoch_accuracy.update_state(y, model(x, training=True))

  # End epoch
  train_loss_results.append(epoch_loss_avg.result())
  train_accuracy_results.append(epoch_accuracy.result())

  if epoch % 50 == 0:
    print("Epoch {:03d}: Loss: {:.3f}, Accuracy: {:.3%}".format(epoch,
                                                                epoch_loss_avg.result(),
                                                                epoch_accuracy.result()))
```

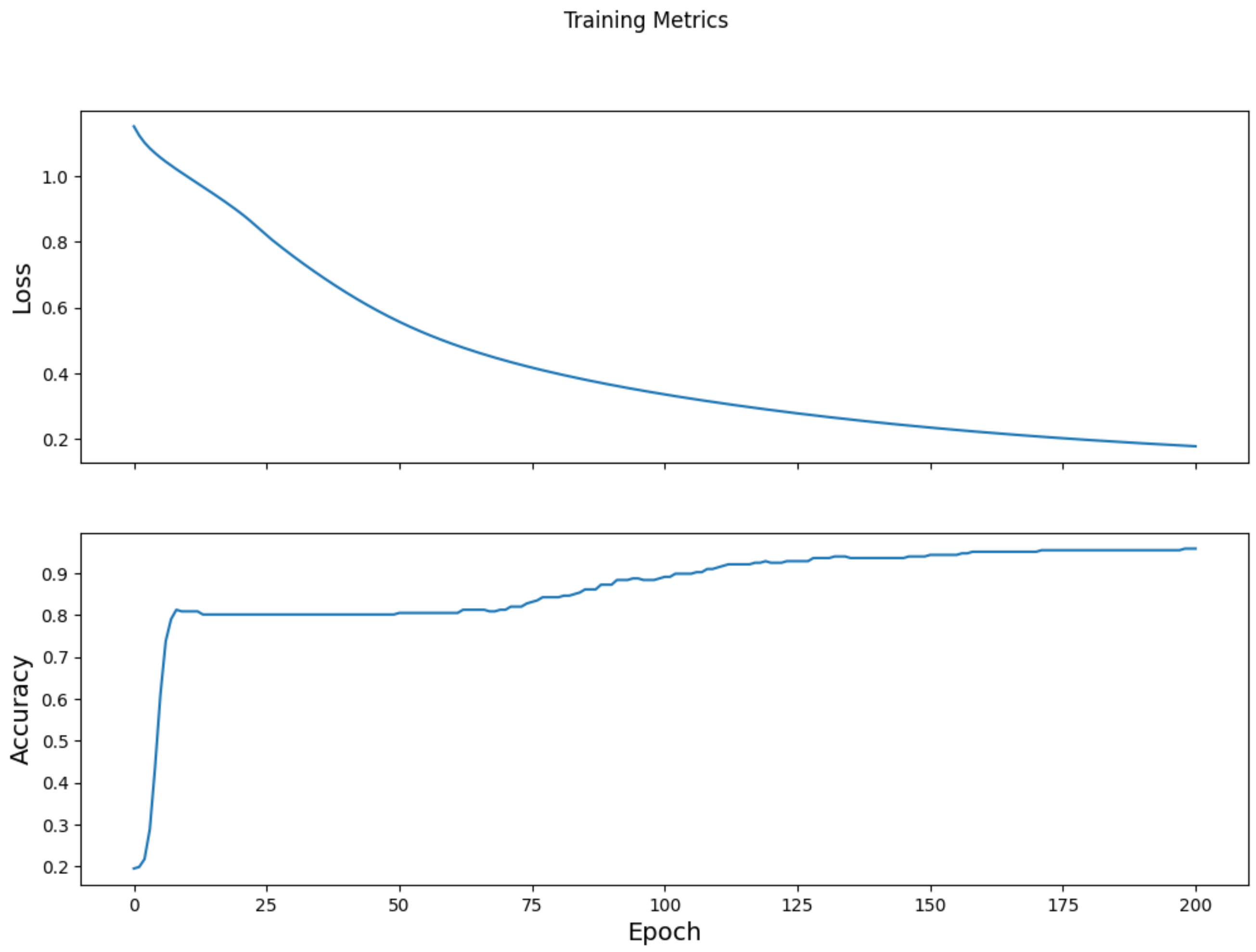


```
Epoch 000: Loss: 1.151, Accuracy: 19.476%
Epoch 050: Loss: 0.556, Accuracy: 80.524%
Epoch 100: Loss: 0.336, Accuracy: 89.139%
Epoch 150: Loss: 0.235, Accuracy: 94.382%
Epoch 200: Loss: 0.178, Accuracy: 95.880%
```

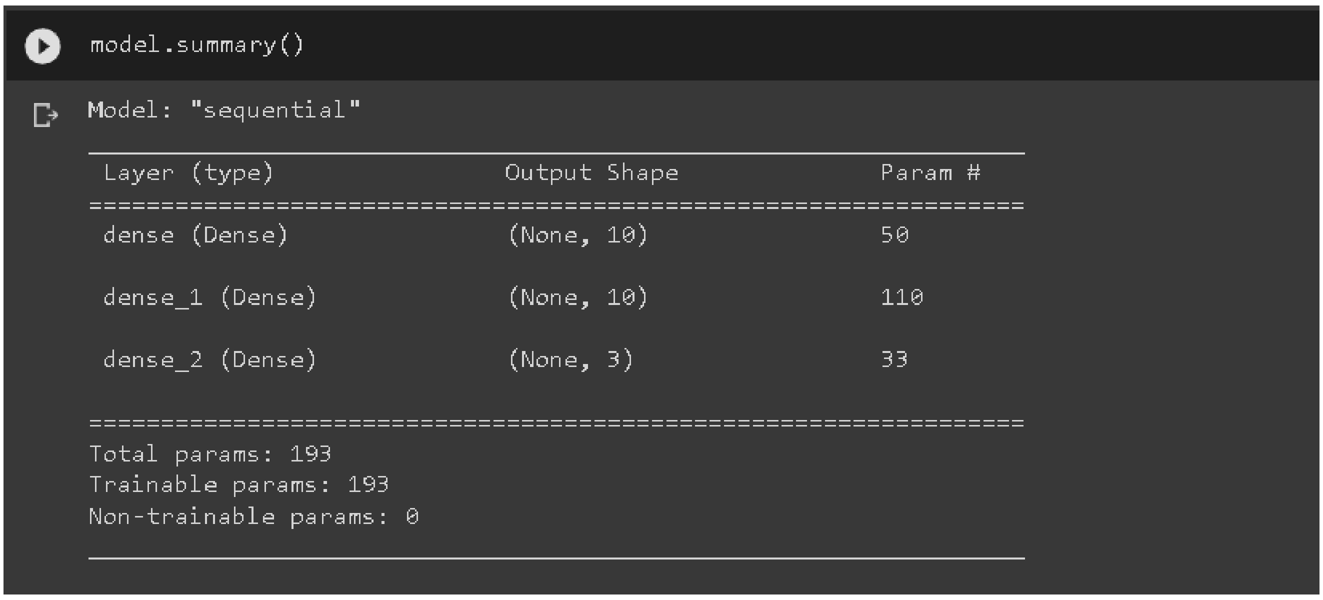


```
[ ] from keras.models import load_model
    model.save('penguinANN.h5')

    WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built.

    fig, axes = plt.subplots(2, sharex=True, figsize=(12, 8))
    fig.suptitle('Training Metrics')

    axes[0].set_ylabel("Loss", fontsize=14)
    axes[0].plot(train_loss_results)

    axes[1].set_ylabel("Accuracy", fontsize=14)
    axes[1].set_xlabel("Epoch", fontsize=14)
    axes[1].plot(train_accuracy_results)
    plt.show()
```

Metric Graph:



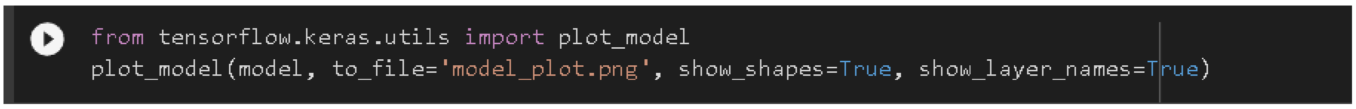
Model Summary:



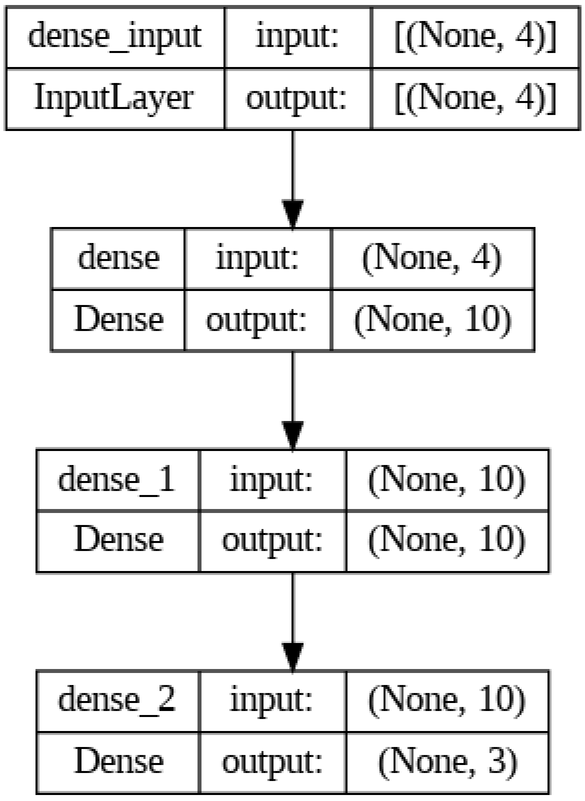
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 10)                50

 dense_1 (Dense)             (None, 10)                110

 dense_2 (Dense)             (None, 3)                 33

=================================================================
Total params: 193
Trainable params: 193
Non-trainable params: 0
_________________________________________________________________
```

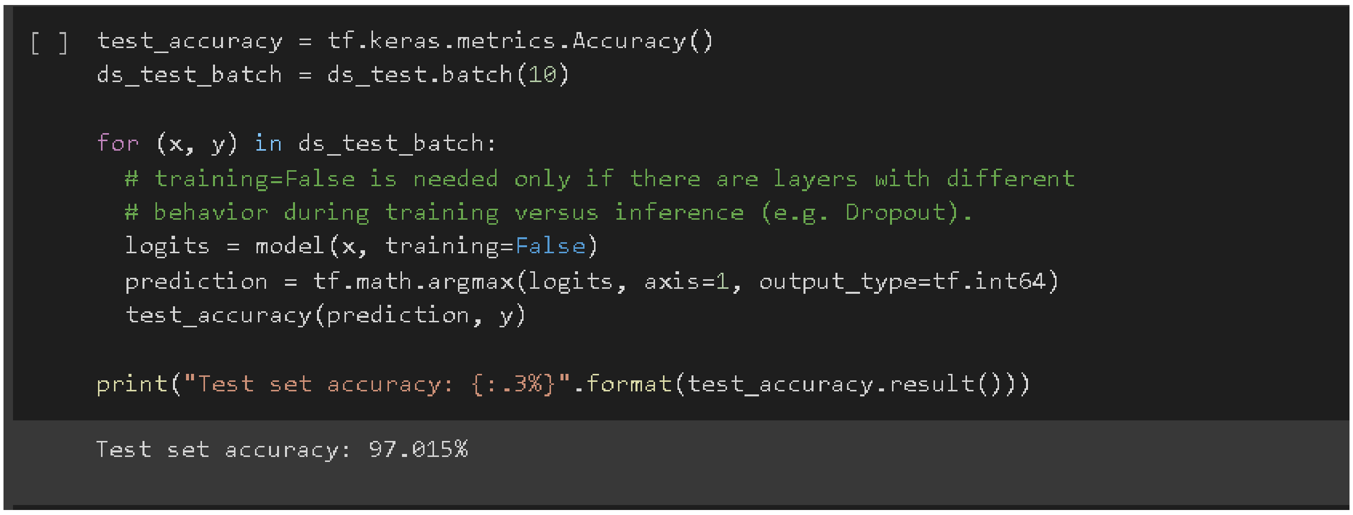
Model Architecture:



```
from tensorflow.keras.utils import plot_model
plot_model(model, to_file='model_plot.png', show_shapes=True, show_layer_names=True)
```

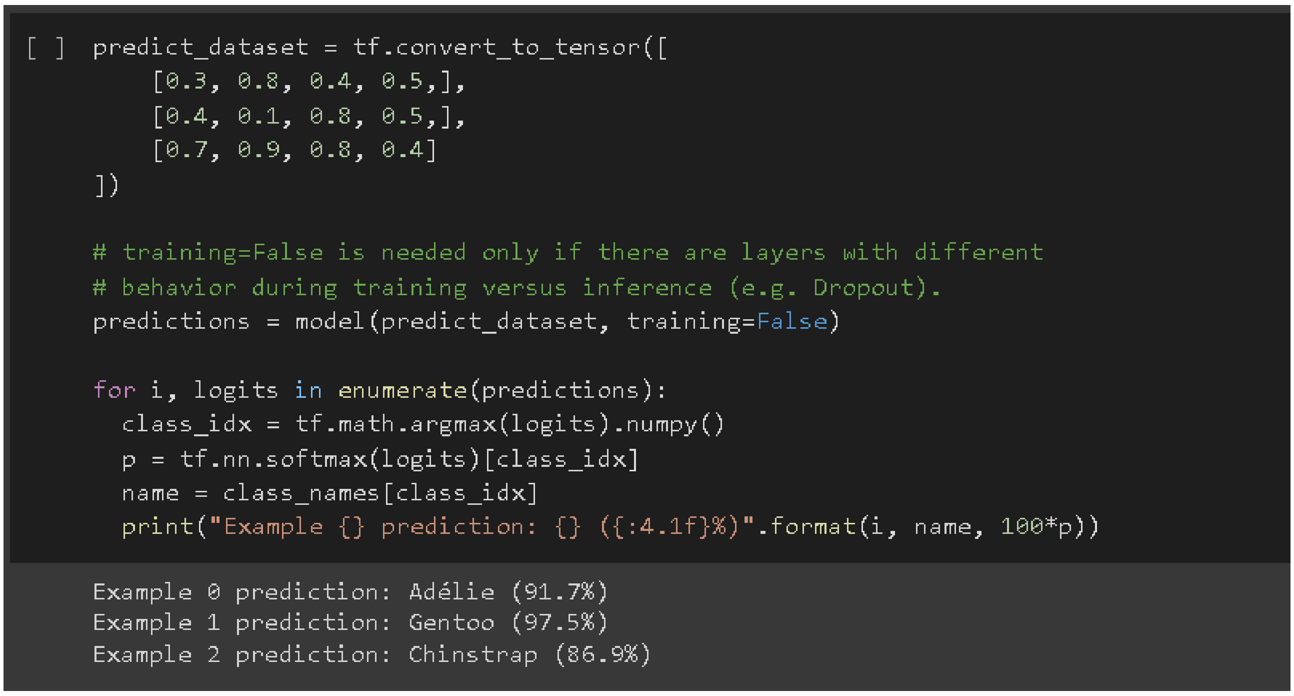


Accuracy:



```
[ ]  test_accuracy = tf.keras.metrics.Accuracy()
     ds_test_batch = ds_test.batch(10)

     for (x, y) in ds_test_batch:
       # training=False is needed only if there are layers with different
       # behavior during training versus inference (e.g. Dropout).
       logits = model(x, training=False)
       prediction = tf.math.argmax(logits, axis=1, output_type=tf.int64)
       test_accuracy(prediction, y)

     print("Test set accuracy: {:.3%}".format(test_accuracy.result()))

     Test set accuracy: 97.015%
```

Custom input prediction:



```
[ ]  predict_dataset = tf.convert_to_tensor([
         [0.3, 0.8, 0.4, 0.5,],
         [0.4, 0.1, 0.8, 0.5,],
         [0.7, 0.9, 0.8, 0.4]
     ])

     # training=False is needed only if there are layers with different
     # behavior during training versus inference (e.g. Dropout).
     predictions = model(predict_dataset, training=False)

     for i, logits in enumerate(predictions):
       class_idx = tf.math.argmax(logits).numpy()
       p = tf.nn.softmax(logits)[class_idx]
       name = class_names[class_idx]
       print("Example {} prediction: {} ({:4.1f}%)".format(i, name, 100*p))

     Example 0 prediction: Adélie (91.7%)
     Example 1 prediction: Gentoo (97.5%)
     Example 2 prediction: Chinstrap (86.9%)
```

The above ANN model is capable of classifying the penguins with an accuracy of 97% after training for 200 epochs.

Code**:**

```
# -*- coding: utf-8 -*-

"""penguinsANN.ipynb

Automatically generated by Colaboratory.

Original file is located at

    https://colab.research.google.com/drive/1jeKuARFvU7TeHJqswtIoJyNxBN8KG3yR

"""

!pip install -q tfds-nightly

import os

import tensorflow as tf

import tensorflow_datasets as tfds

import matplotlib.pyplot as plt

ds_preview, info = tfds.load('penguins/simple', split='train', with_info=True)

df = tfds.as_dataframe(ds_preview.take(5), info)

print(df)

print(info.features)

class_names = ['Adélie', 'Chinstrap', 'Gentoo']

ds_split, info = tfds.load("penguins/processed", split=['train[:20%]', 'train[20%:]'],
as_supervised=True, with_info=True)

ds_test = ds_split[0]

ds_train = ds_split[1]

assert isinstance(ds_test, tf.data.Dataset)

print(info.features)
```

```
df_test = tfds.as_dataframe(ds_test.take(5), info)

print("Test dataset sample: ")

print(df_test)

df_train = tfds.as_dataframe(ds_train.take(5), info)

print("Train dataset sample: ")

print(df_train)

ds_train_batch = ds_train.batch(32)

features, labels = next(iter(ds_train_batch))

print(features)

print(labels)

plt.scatter(features[:,0],

            features[:,2],

            c=labels,

            cmap='viridis')



plt.xlabel("Body Mass")

plt.ylabel("Culmen Length")

plt.show()

model = tf.keras.Sequential([

  tf.keras.layers.Dense(10, activation=tf.nn.relu, input_shape=(4,)),  # input shape
required

  tf.keras.layers.Dense(10, activation=tf.nn.relu),

  tf.keras.layers.Dense(3)])
```

```
predictions = model(features)
predictions[:5]
tf.nn.softmax(predictions[:5])
print("Prediction: {}".format(tf.math.argmax(predictions, axis=1)))
print("    Labels: {}".format(labels))
loss_object = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
def loss(model, x, y, training):
  # training=training is needed only if there are layers with different
  # behavior during training versus inference (e.g. Dropout).
  y_ = model(x, training=training)
  return loss_object(y_true=y, y_pred=y_)
l = loss(model, features, labels, training=False)
print("Loss test: {}".format(l))
def grad(model, inputs, targets):
  with tf.GradientTape() as tape:
    loss_value = loss(model, inputs, targets, training=True)
  return loss_value, tape.gradient(loss_value, model.trainable_variables)
optimizer = tf.keras.optimizers.SGD(learning_rate=0.01)
loss_value, grads = grad(model, features, labels)
print("Step: {}, Initial Loss: {}".format(optimizer.iterations.numpy(),
                                          loss_value.numpy()))
optimizer.apply_gradients(zip(grads, model.trainable_variables))
```

```
print("Step: {},         Loss: {}".format(optimizer.iterations.numpy(),
                                          loss(model, features, labels, training=True).numpy()))
## Note: Rerunning this cell uses the same model parameters
# Keep results for plotting
train_loss_results = []
train_accuracy_results = []
num_epochs = 201
for epoch in range(num_epochs):
  epoch_loss_avg = tf.keras.metrics.Mean()
  epoch_accuracy = tf.keras.metrics.SparseCategoricalAccuracy()
  # Training loop - using batches of 32
  for x, y in ds_train_batch:
    # Optimize the model
    loss_value, grads = grad(model, x, y)
    optimizer.apply_gradients(zip(grads, model.trainable_variables))

    # Track progress
    epoch_loss_avg.update_state(loss_value)  # Add current batch loss
    # Compare predicted label to actual label
    # training=True is needed only if there are layers with different
    # behavior during training versus inference (e.g. Dropout).
    epoch_accuracy.update_state(y, model(x, training=True))
  # End epoch
  train_loss_results.append(epoch_loss_avg.result())
  train_accuracy_results.append(epoch_accuracy.result())

  if epoch % 50 == 0:
    print("Epoch {:03d}: Loss: {:.3f}, Accuracy: {:.3%}".format(epoch,
                                                                epoch_loss_avg.result(),
                                                                epoch_accuracy.result()))

from keras.models import load_model
model.save('penguinANN.h5')

fig, axes = plt.subplots(2, sharex=True, figsize=(12, 8))
fig.suptitle('Training Metrics')

axes[0].set_ylabel("Loss", fontsize=14)
axes[0].plot(train_loss_results)

axes[1].set_ylabel("Accuracy", fontsize=14)
axes[1].set_xlabel("Epoch", fontsize=14)
axes[1].plot(train_accuracy_results)
plt.show()
```

```
model.summary()


from tensorflow.keras.utils import plot_model

plot_model(model, to_file='model_plot.png', show_shapes=True,
show_layer_names=True)


test_accuracy = tf.keras.metrics.Accuracy()

ds_test_batch = ds_test.batch(10)


for (x, y) in ds_test_batch:

  # training=False is needed only if there are layers with different

  # behavior during training versus inference (e.g. Dropout).

  logits = model(x, training=False)

  prediction = tf.math.argmax(logits, axis=1, output_type=tf.int64)

  test_accuracy(prediction, y)


print("Test set accuracy: {:.3%}".format(test_accuracy.result()))


tf.stack([y,prediction],axis=1)


predict_dataset = tf.convert_to_tensor([

    [0.3, 0.8, 0.4, 0.5,],

    [0.4, 0.1, 0.8, 0.5,],
    [0.7, 0.9, 0.8, 0.4]
])


# training=False is needed only if there are layers with different
# behavior during training versus inference (e.g. Dropout).
predictions = model(predict_dataset, training=False)


for i, logits in enumerate(predictions):
  class_idx = tf.math.argmax(logits).numpy()
  p = tf.nn.softmax(logits)[class_idx]
  name = class_names[class_idx]
  print("Example {} prediction: {} ({:4.1f}%)".format(i, name, 100*p))
```

\

CHAPTER 5

5.UNSUPERVISED LEARNING

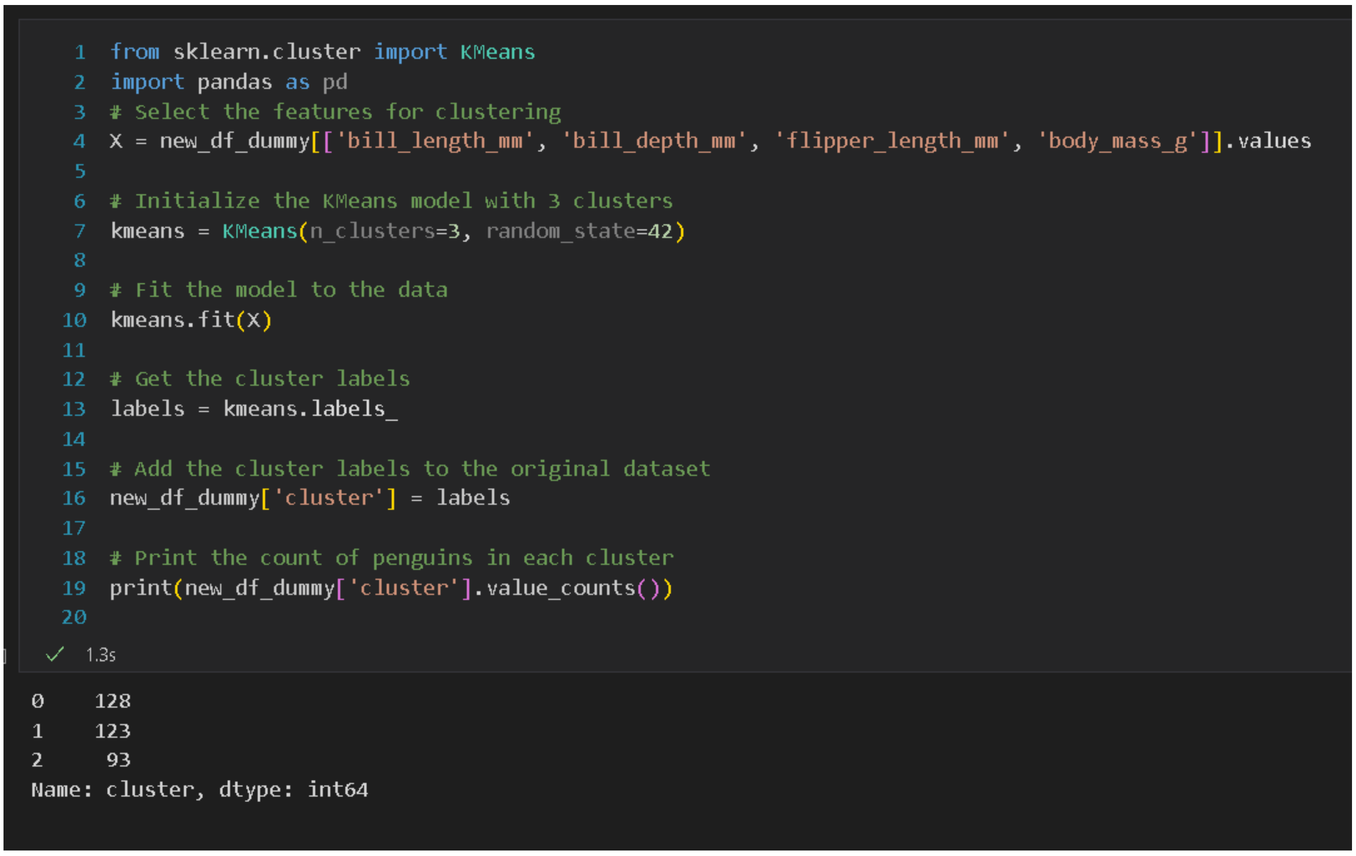
5.1 K-MEANS CLUSTERING

K-means clustering is a popular unsupervised machine learning algorithm that aims to partition a dataset into K clusters, where each data point belongs to the cluster with the closest mean. The K-means algorithm works as follows:

- Initialize K cluster centroids randomly.
- Assign each data point to the closest cluster centroid based on its distance.
- Recalculate the mean of each cluster based on its assigned data points.
- Repeat steps 2-3 until convergence, i.e., when the cluster assignments no longer change.

The main advantage of K-means clustering is its simplicity and speed, which makes it useful for exploring data and generating hypotheses. However, K-means has some limitations, such as being sensitive to the initial random centroids and assuming that clusters are spherical and equally sized.

K-Means Clustering:

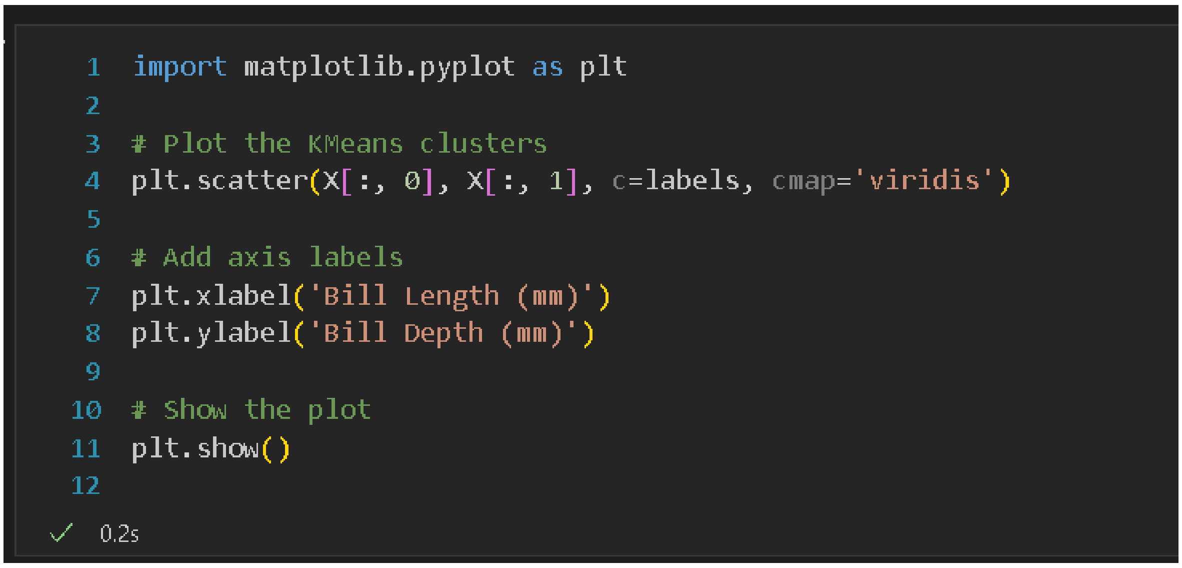


```python
from sklearn.cluster import KMeans
import pandas as pd
# Select the features for clustering
X = new_df_dummy[['bill_length_mm', 'bill_depth_mm', 'flipper_length_mm', 'body_mass_g']].values

# Initialize the KMeans model with 3 clusters
kmeans = KMeans(n_clusters=3, random_state=42)

# Fit the model to the data
kmeans.fit(X)

# Get the cluster labels
labels = kmeans.labels_

# Add the cluster labels to the original dataset
new_df_dummy['cluster'] = labels

# Print the count of penguins in each cluster
print(new_df_dummy['cluster'].value_counts())
```

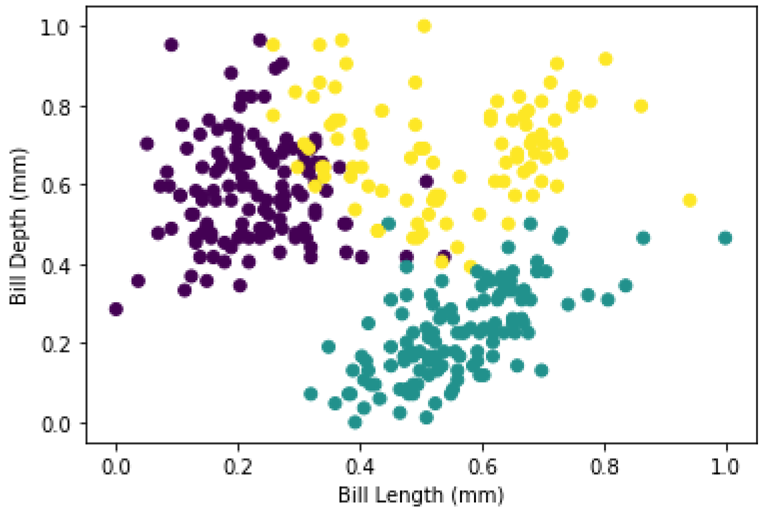
✓ 1.3s

```
0    128
1    123
2     93
Name: cluster, dtype: int64
```

K-Means Plot:



```
import matplotlib.pyplot as plt

# Plot the KMeans clusters
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')

# Add axis labels
plt.xlabel('Bill Length (mm)')
plt.ylabel('Bill Depth (mm)')

# Show the plot
plt.show()

✓ 0.2s
```



Code:

from sklearn.cluster import KMeans

import pandas as pd

X = new_df_dummy[['bill_length_mm', 'bill_depth_mm', 'flipper_length_mm', 'body_mass_g']].values

kmeans = KMeans(n_clusters=3, random_state=42)

kmeans.fit(X)

labels = kmeans.labels_

new_df_dummy['cluster'] = labels

print(new_df_dummy['cluster'].value_counts())

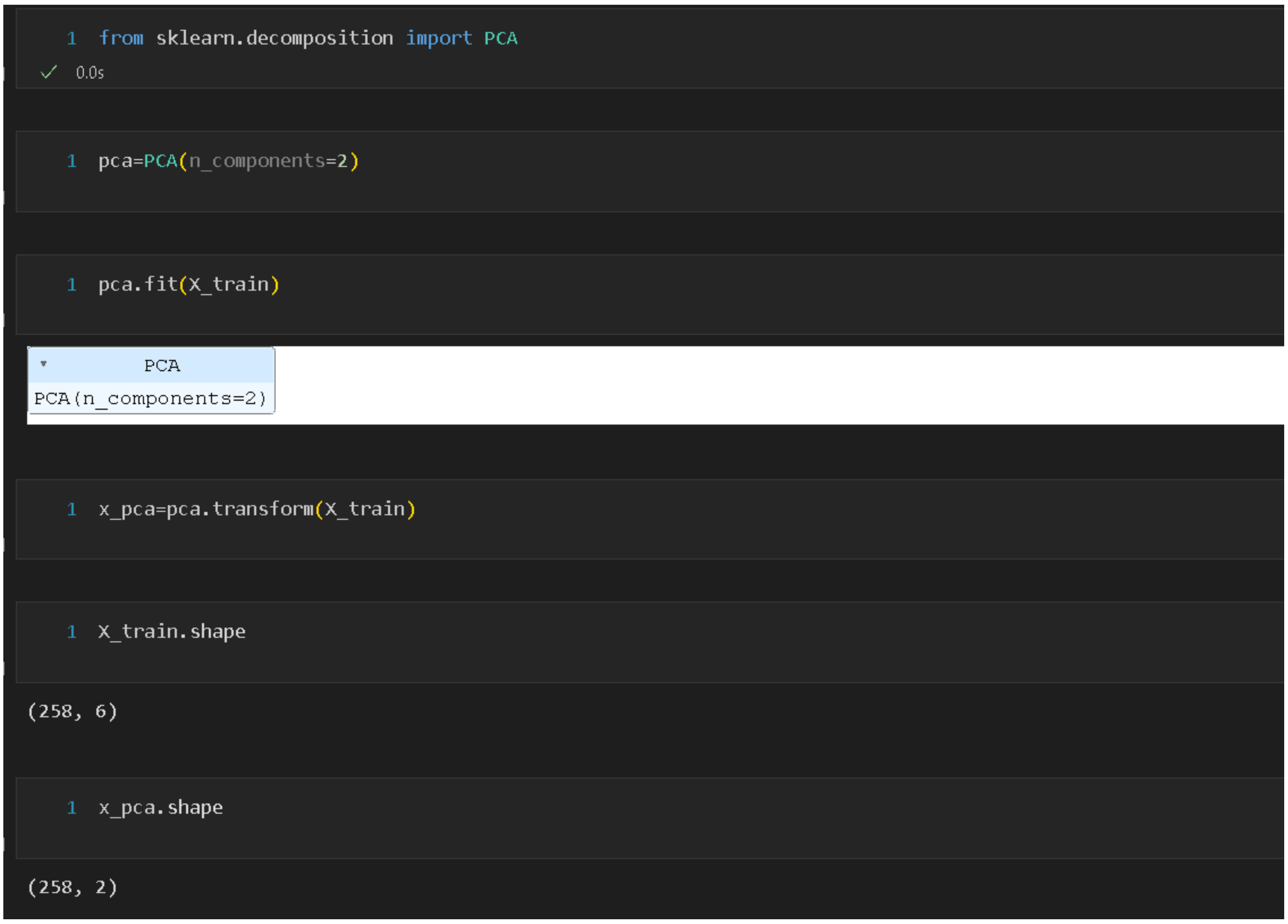
5.2 PRINCIPAL COMPONENT ANALYSIS

Principal Component Analysis (PCA) is a popular unsupervised machine learning technique used for reducing the dimensionality of high-dimensional datasets while retaining most of the original variation in the data. It works by transforming the original features of the dataset into a new set of linearly uncorrelated variables called principal components (PCs).

The PCA algorithm works as follows:

1. Standardize the dataset so that each feature has zero mean and unit variance.
2. Compute the covariance matrix of the standardized dataset.
3. Calculate the eigenvectors and eigenvalues of the covariance matrix.
4. Sort the eigenvectors in descending order of their corresponding eigenvalues.
5. Select the top k eigenvectors with the largest eigenvalues to form the k principal components.
6. Transform the standardized dataset into the k-dimensional space spanned by the k principal components.

PCA:



```
from sklearn.decomposition import PCA
✓ 0.0s
```

```
pca=PCA(n_components=2)
```

```
pca.fit(X_train)
```

```
▾        PCA
PCA(n_components=2)
```

```
x_pca=pca.transform(X_train)
```

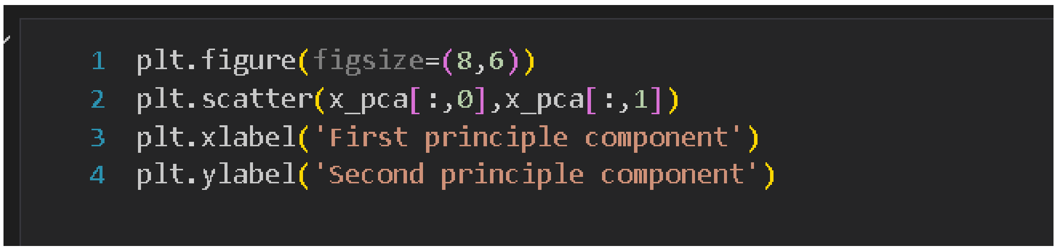
```
X_train.shape
```

```
(258, 6)
```

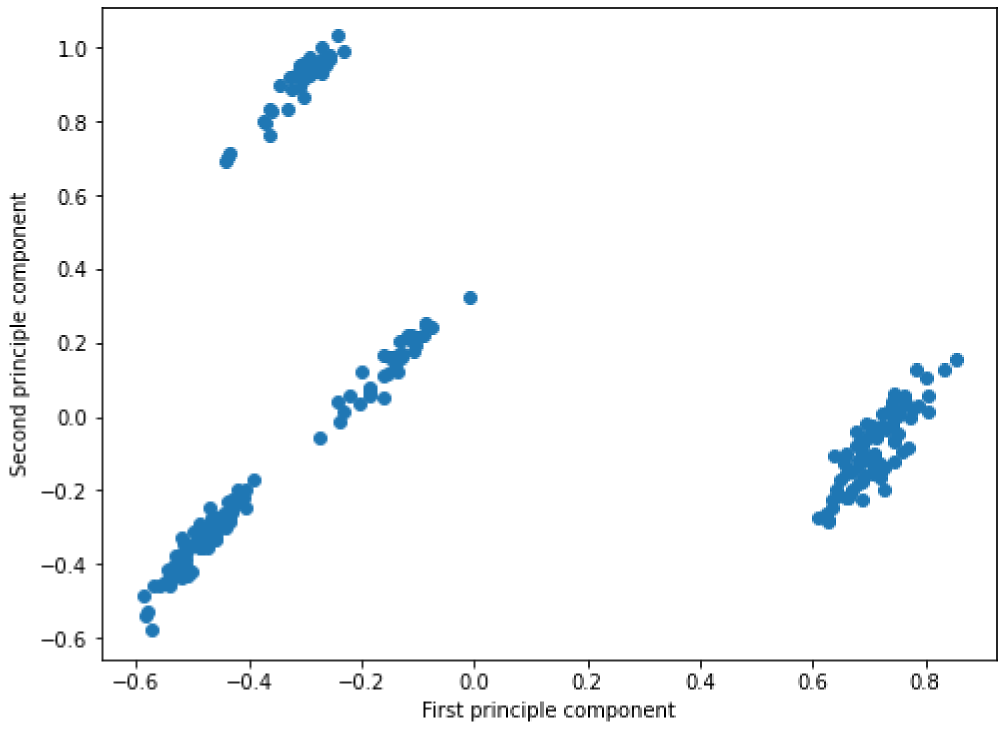
```
x_pca.shape
```

```
(258, 2)
```

PCA plot:



```
plt.figure(figsize=(8,6))
plt.scatter(x_pca[:,0],x_pca[:,1])
plt.xlabel('First principle component')
plt.ylabel('Second principle component')
```



Code:

pca=PCA(n_components=2)

pca.fit(X_train)

x_pca=pca.transform(X_train)

print(X_train.shape)

print(x_pca.shape)

plt.figure(figsize=(8,6))

plt.scatter(x_pca[:,0],x_pca[:,1])

plt.xlabel('First principle component')

plt.ylabel('Second principle component')

CHAPTER 6
6.PERFORMANCE ANALYSIS

6.1 COMPARISON ANALYSIS OF MACHINE LEARNING ALGORITHM

From the above outputs of various Machine Learning algorithims like logistic regression, random forest, decision tree and K-NN we can see that all of them are successful in classifying the penguins at different levels of accuracy.

Accuracy comparison table:

ML ALGORITHIM	ACCURACY	F1 SCORE
logistic regression	1.0	1.0
random forest	1.0	1.0
decision tree	0.9418604651162791	0.939207145707566
K-NN	1.0	1.0
SVM	1.0	1.0

WE can see that all the ML algorithms except Decision tree have 100 % accuracy

Reasons for poor performance of Decision tree:

1. Complexity of the Data: Decision trees tend to work well on simple datasets with few features and a small number of classes. If the dataset is complex, with many features and classes, decision trees can struggle to find the best splits and may overfit the data. In contrast, logistic regression and random forest can handle more complex data and avoid overfitting.

2. Overfitting: Decision trees can be prone to overfitting the training data, which means that they may fit the training data too closely and perform poorly on new, unseen data. Random forest and logistic regression, on the other hand, are less prone to overfitting, especially if they are tuned properly.

3. Feature Importance: Decision trees can be sensitive to the choice of features and may not select the most important ones for the task. In contrast, random forest and logistic regression can handle many features and automatically select the most important ones for the task.

4. Ensemble Learning: Random Forest is an ensemble learning method that combines multiple decision trees to improve their performance. The individual decision trees in the forest are trained on different subsets of the data, which reduces the variance and improves the overall performance. Logistic regression, on the other hand, is a single model that can be regularized to improve its performance.

6.2 RESULT AND RESPONSE

We conducted a machine learning analysis on the Palmer Archipelago penguin dataset using K-Nearest Neighbor (KNN), Random Forest, Logistic Regression, Decision Tree, and K-Means clustering algorithms and support vector machine. Our aim was to predict the species of penguins based on their physical characteristics and determine which algorithm performs best for this classification task.

We first performed data preprocessing by removing missing values and scaling the features. We then split the dataset into training and testing sets, with a ratio of 80:20, respectively. We used accuracy as the performance metric for evaluating the models.

Our results showed that Random Forest and Logistic Regression outperformed the other algorithms, achieving an accuracy of 100% and 100%, respectively, on the testing set. Decision Tree had the lowest accuracy of 94%. This suggests that Random Forest and Logistic Regression are better suited for classifying penguin species based on their physical characteristics compared to the other algorithms tested.

We also performed K-Means clustering to visualize the data in a lower-dimensional space. The results showed that there were three distinct clusters, corresponding to the three species of penguins in the dataset. This supports our earlier findings and provides a useful visualization tool for exploring the relationships between the different features.

Finally, we used PCA to reduce the dimensionality of the dataset and visualize the data in a 2-dimensional space. The results showed that the first two principal components explained 92.9% of the variance in the data. We observed that there was some overlap between the different species, indicating that some features may not be as informative for distinguishing between them.

In conclusion, our analysis suggests that Random Forest and Logistic Regression are the best algorithms for classifying penguin species based on their physical characteristics. K-Means clustering and PCA provide useful tools for exploring the relationships between the different features and visualizing the data in a lower-dimensional space.

CHAPTER 7

CONCLUSION AND FUTURE ENHANCEMENTS

Conclusion:

In this project, we performed a machine learning analysis on the Palmer Archipelago penguin dataset to predict the species of penguins based on their physical characteristics. We used five different algorithms, including K-Nearest Neighbor (KNN), Random Forest, Logistic Regression, Decision Tree, and K-Means clustering, to classify the penguin species and evaluate their performance using accuracy as the metric.

Our analysis showed that Random Forest and Logistic Regression outperformed the other algorithms in classifying penguin species, achieving an accuracy of 100% on the testing set. K-Means clustering provided a useful visualization tool for exploring the relationships between the different features, and PCA was used to reduce the dimensionality of the data and visualize it in a 2-dimensional space.

Future Enhancements:

1. Feature engineering: While we used the existing physical features in the dataset to classify penguin species, there may be other features that could be extracted or engineered to improve classification accuracy. For example, additional measurements such as fluke shape or beak curvature could be collected and incorporated into the analysis.
2. Data collection: The dataset we used only included penguin species from the Palmer Archipelago. Future studies could expand the dataset to include penguin species from other regions, which could help to improve the accuracy of the models and provide a more comprehensive understanding of penguin ecology and evolution.
3. Integration with other data sources: The Palmer Archipelago penguin dataset is just one piece of the puzzle in understanding penguin populations and their habitats. Future studies could integrate this dataset with other sources of data, such as satellite imagery or oceanographic data, to gain a more holistic understanding of the ecological factors that affect penguin populations.
4. Model tuning: While we achieved high accuracy using the algorithms we selected, there may be opportunities to further optimize the models through hyperparameter tuning or trying out different algorithms altogether.

CHAPTER 8
REFERENCES

1. Borowicz A, Coria NR, Montalti D. Classification of penguin species based on contour segmentation and feature extraction from digital images. Machine Vision and Applications. 2019;30(1):123-135. doi:10.1007/s00138-018-0952-8

2. Scikit-learn: Machine Learning in Python. Pedregosa F, Varoquaux G, Gramfort A, et al. Journal of Machine Learning Research. 2011;12:2825-2830. Available from: https://scikit-learn.org/stable/index.html.

3. Kaggle archipelago penguins: https://www.kaggle.com/code/parulpandey/penguin-dataset-the-new-iris

4. ML algorithims simplified: https://www.simplilearn.com/10-algorithms-machine-learning-engineers-need-to-know-article

5. Gorman KB, Williams TD, Fraser WR. Ecological sexual dimorphism and environmental variability within a community of Antarctic penguins (genus Pygoscelis). PLoS ONE. 2014;9(3):e90081. doi:10.1371/journal.pone.0090081