Norah Khan nmk180004 Aleena Syed azs180003

Text Classification

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score,classification_report
from sklearn.datasets import load_digits
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, models
%matplotlib inline
```

In [2]:
```
df = pd.read_csv("penguins_size.csv")
df.head()
```

Out[2]:

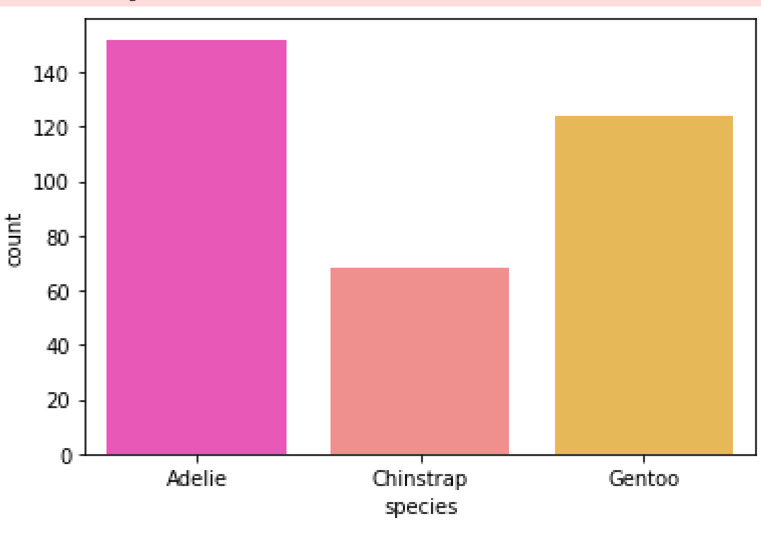
	species	island	culmen_length_mm	culmen_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	MALE
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	FEMALE
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	FEMALE
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	FEMALE

In [3]:
```
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
```

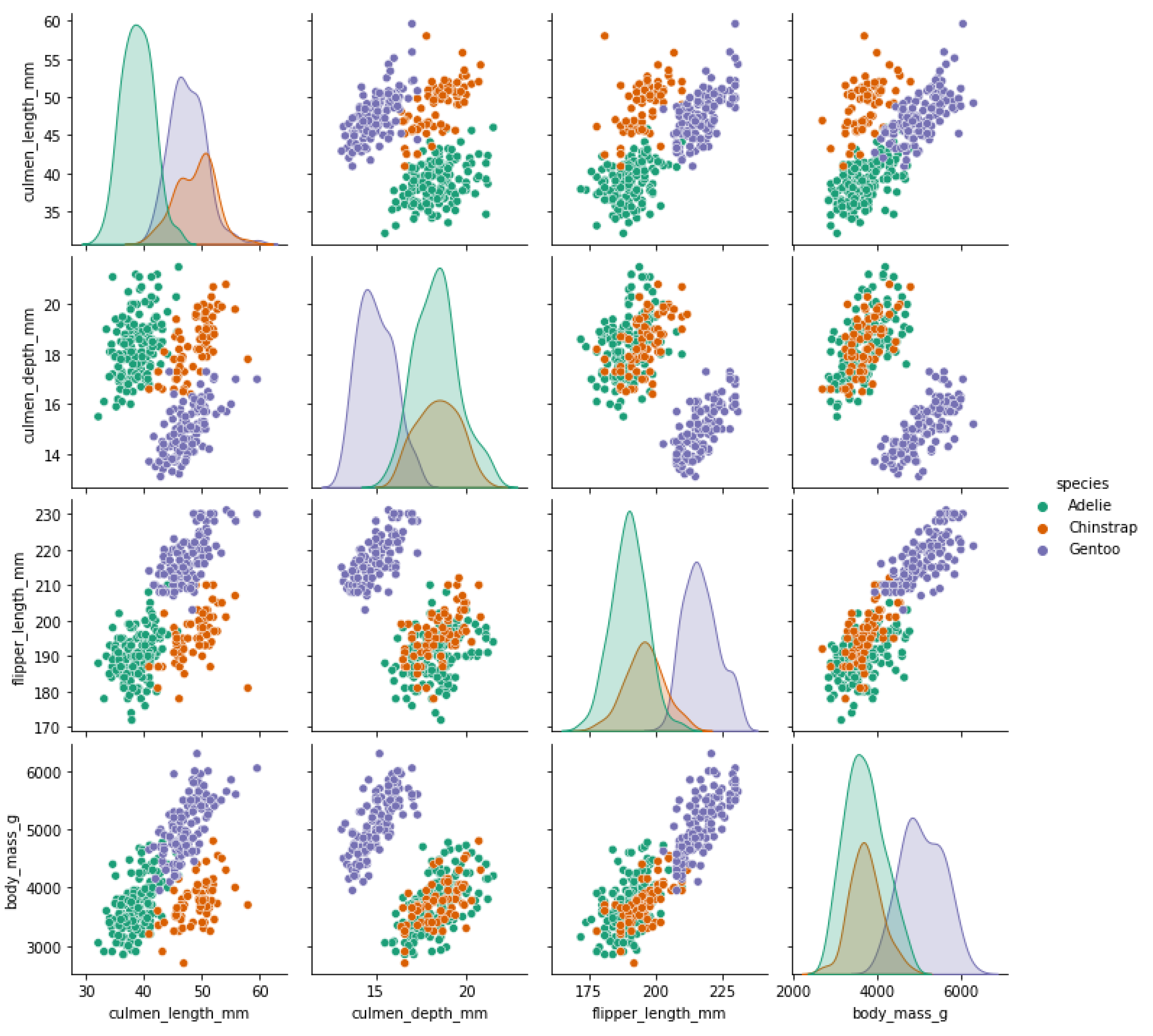
In [4]:
```
# distribution of classes
sns.countplot(df['species'],palette='spring');
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other argum ents without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



In [5]:
```
sns.pairplot(df, hue='species', palette='Dark2')
```

Out[5]: <seaborn.axisgrid.PairGrid at 0x7f54aaebe850>



In [6]:
```
# fill in missing data
df['sex'].fillna(df['sex'].mode()[0],inplace=True)
col_to_be_imputed = ['culmen_length_mm', 'culmen_depth_mm','flipper_length_mm', 'body_mass_g']
for item in col_to_be_imputed:
    df[item].fillna(df[item].mean(),inplace=True)
```

In [7]:
```
# replace '.' with proper entry
df[df['sex']=='.']
df.loc[336,'sex'] = 'FEMALE'
```

In [8]:
```
# encode target variables and create dummy variables
df['species']=df['species'].map({'Adelie':0,'Gentoo':1,'Chinstrap':2})
dummies = pd.get_dummies(df[['island','sex']],drop_first=True)
```

In [9]:
```
# standardize feature variables
df_to_be_scaled = df.drop(['island','sex'],axis=1)
target = df_to_be_scaled.species
df_feat= df_to_be_scaled.drop('species',axis=1)
```

In [11]:
```
# standardize feature variables
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(df_feat)
df_scaled = scaler.transform(df_feat)
df_scaled = pd.DataFrame(df_scaled,columns=df_feat.columns[:4])
df_preprocessed = pd.concat([df_scaled,dummies,target],axis=1)
df_preprocessed.head()
```

Out[11]:

	culmen_length_mm	culmen_depth_mm	flipper_length_mm	body_mass_g	island_Dream	island_Torgersen	sex_MALE	species
0	-8.870812e-01	7.877425e-01	-1.422488	-0.565789	0	1	1	0
1	-8.134940e-01	1.265563e-01	-1.065352	-0.503168	0	1	0	0
2	-6.663195e-01	4.317192e-01	-0.422507	-1.192003	0	1	0	0
3	-1.307172e-15	1.806927e-15	0.000000	0.000000	0	1	1	0
4	-1.328605e+00	1.092905e+00	-0.565361	-0.941517	0	1	0	0

In [12]:
```
# split data intro training and testing
X_train, X_test, y_train, y_test = train_test_split(df_preprocessed.drop('species',axis=1),target,test_size=0.5
```

In [13]:
```
# get dimension of X_train
input_units = X_train.shape[1]
```

In [14]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
```

In [37]:
```
# KNN model
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df_preprocessed.drop('species',axis=1),target,test_size=0.5
knn.fit(X_train, y_train)
```

Out[37]: KNeighborsClassifier(n_neighbors=1)

In [38]:
```
# predict class with KNN
pred = knn.predict(X_test)
```

In [39]:
```
# knn classification
model  =
knn = KNeighborsClassifier(n_neighbors=12)
knn.fit(X_train, y_train)
pred = knn.predict(X_test)
cm = confusion_matrix(y_test, pred)
print(cm)
print(classification_report(y_test, pred))
```

```
[[65  0  0]
 [ 0 64  0]
 [ 0  0 43]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        65
           1       1.00      1.00      1.00        64
           2       1.00      1.00      1.00        43

    accuracy                           1.00       172
   macro avg       1.00      1.00      1.00       172
weighted avg       1.00      1.00      1.00       172
```

In [ ]:
```
# sequential model
model = keras.Sequential()
model.add(keras.layers.Dense(input_units, input_dim=input_units, activation='relu'))
model.add(keras.layers.Dense(input_units, activation='relu'))
model.add(keras.layers.Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy',
              optimizer='adam', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=250, batch_size=1000)
scores = model.evaluate(x=X_test, y=y_test)
model.summary()
```

In [16]:
```
# evaluate
score = model.evaluate(X_test, y_test, batch_size=1000, verbose=1)
print('Accuracy: ', score[1])
```

```
1/1 [==============================] - 0s 150ms/step - loss: -0.3722 - accuracy: 0.7791
Accuracy:  0.7790697813034058
```

In [17]:
```
# split data intro training and test
X_train, X_test, y_train, y_test = train_test_split(df_preprocessed.drop('species',axis=1),target,test_size=0.5
```

In [18]:
```
model = keras.Sequential()
model.add(layers.Embedding(input_dim=input_units, output_dim=64))

# The output of GRU will be a 3D tensor of shape (batch_size, timesteps, 256)
model.add(layers.GRU(256, return_sequences=True))

# The output of SimpleRNN will be a 2D tensor of shape (batch_size, 128)
model.add(layers.SimpleRNN(128))

model.add(layers.Dense(10))

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding (Embedding)       (None, None, 64)          448       
                                                                 
 gru (GRU)                   (None, None, 256)         247296    
                                                                 
 simple_rnn (SimpleRNN)      (None, 128)               49280     
                                                                 
 dense_3 (Dense)             (None, 10)                1290      
                                                                 
=================================================================
Total params: 298,314
Trainable params: 298,314
Non-trainable params: 0
_________________________________________________________________
```

In [44]:
```
score = model.evaluate(X_test, y_test, batch_size=1000, verbose=1)
print('Accuracy: ', score[1])
```

```
1/1 [==============================] - 0s 38ms/step - loss: -0.5191 - accuracy: 0.7326
Accuracy:  0.7325581312179565
```

Knn is used to categorize data using n closest neighbors. After training the data it can be used to make predictions. The same can be done with the RNN model. The RNN model does an efficient job of categorizing and predicting this type of dataset. The evaluation of the RNN model shows to be about 73% whereas the Sequential model is 77%. Both models have similar accuracy and prediction but for this dataset the sequential model is more precise. The CNN model is not used in this dataset because that is focused on predicting/identifying images which does not apply in this case.