Project 5: Machine Learning

John Wesley Mathis

Dr. Anthony Choi

June 19, 2024

ECE/SSE 591, Summer 2024

Table of Contents:

[Deliverable Table 5](#__RefHeading___Toc2569_331549091)

[1. Introduction 6](#__RefHeading___Toc2573_331549091)

[2. Predicting Ice Cream Sales Linear Regression 7](#__RefHeading___Toc2577_331549091)

[3. Fruit Classification 11](#__RefHeading___Toc7733_1546921247)

[4. Pokémon Classification 16](#__RefHeading___Toc848_3052841676)

[5. Predicting Animal Species 20](#__RefHeading___Toc697_3052841676)

[6. Conclusion 41](#__RefHeading___Toc2609_331549091)

[References 42](#__RefHeading___Toc3692_331549091)

Table of Figures

[Figure 1: Libraries 7](#Figure!2|sequence)

[Figure 2: Generating Ice Cream Data 8](#Figure!0|sequence)

[Figure 3: Defining features matrix and target vector and spliting the data 9](#Figure!1|sequence)

[Figure 4: Training and Evaluating LinearRegression Model 9](#Figure!3|sequence)

[Figure 5: Plotting the LinearRegression Model 10](#Figure!4|sequence)

[Figure 6: Plotting predicitons 11](#Figure!5|sequence)

[Figure 7: Generating data for Fruit Classification 11](#Figure!6|sequence)

[Figure 8: Encoding fruit\_df features 13](#Figure!7|sequence)

[Figure 9: Decoding numerical categories 14](#Figure!8|sequence)

[Figure 10: Training and evaluating a DecisionTreeClassifier model 15](#Figure!9|sequence)

[Figure 11: Pairplot to visualize fruit features with the fruit category 16](#Figure!10|sequence)

[Figure 12: Importing Pokemon dataset 17](#Figure!11|sequence)

[Figure 13: Training and evaluating RandomForestClassifier model 17](#Figure!12|sequence)

[Figure 14: Code to visualize a single tree 18](#Figure!13|sequence)

[Figure 15: Visualizing the decision tree 18](#Figure!14|sequence)

[Figure 16: Evaluating RandomForestClassifier model using accuracy score and classification report 19](#Figure!15|sequence)

[Figure 17: Visualizing confusion matrix 20](#Figure!16|sequence)

[Figure 18: Importing libraries and extracting species names from folder names 21](#Figure!17|sequence)

[Figure 19: Converting images and defining features matrix and target vector 22](#Figure!18|sequence)

[Figure 20: Flattening features matrix 22](#Figure!19|sequence)

[Figure 21: Encoding target vector 23](#Figure!20|sequence)

[Figure 22: Testing and evaluating RandomForestClassifier for Animal Prediction 24](#Figure!21|sequence)

[Figure 23: Code to visualize results of the RandomForestClassifier 25](#Figure!22|sequence)

[Figure 24: Animal Species Predictions of the RandomForestClassifier model 26](#Figure!23|sequence)

[Figure 25: Code to construct a validation curve for RandomForestClassifier model 27](#Figure!24|sequence)

[Figure 26: Validation Curve plotting the relationship between the accuracy and the number of estimators 28](#Figure!25|sequence)

[Figure 27: Code to plot learning curve for RandomForestClassifier model 29](#Figure!26|sequence)

[Figure 28: Learning Curve for RandomForestClassifier model 30](#Figure!27|sequence)

[Figure 29: Hyperparameter tuning for RandomForestClassifier using GridSearchCV 31](#Figure!28|sequence)

[Figure 30: Hyperparameter tuniong for DecisionTreeClassifier using GridSearchCV 31](#Figure!29|sequence)

[Figure 31: Re-training and evaluating update models with best parameters 32](#Figure!30|sequence)

[Figure 32: Loading libraries and importing dataset 33](#Figure!31|sequence)

[Figure 33: Converting images to grayscale and defining images and labels arrays 34](#Figure!32|sequence)

[Figure 34: Extracting HOG features and defining features matrix and target vector and splitting data 35](#Figure!33|sequence)

[Figure 35: Code for training, evaluating, and plotting the model 36](#Figure!34|sequence)

[Figure 36: Model Evaluation output and visualization 37](#Figure!35|sequence)

[Figure 37: Loading and converting data 39](#Figure!36|sequence)

[Figure 38: Extracting HOG features and defining features matrix 40](#Figure!37|sequence)

[Figure 39: Training a K-means cluster model 40](#Figure!38|sequence)

[Figure 40: Training a PCA model and visualizing the models 40](#Figure!39|sequence)

[Figure 41: K-means clusters with PCA components 41](#Figure!40|sequence)

# Deliverable Table

The purpose of this table is to provide a complete view of the concepts covered in chapter 5 of *"Python Data Science Handbook"* (VanderPlas, 2016) and provide a general project/page location for where the topic was demonstrated. Some of the projects cover more than the specified deliverable.

|  |  |
| --- | --- |
| Deliverables | Location |
| What is Machine Learning? | Predicting Animal Species (p.19) |
| Introducing Scikit-Learn | Predicting Ice Cream Sales Linear Regression (p.6) |
| Hyperparameters and Model Validation | Predicting Animal Species (p.19) |
| Feature Engineering | Fruit Classification (p.10) |
| In-Depth: Decision Trees and Random Forests | Pokémon Classification (p.15) |

Additionally, here is a link to my GitHub were the datasets and the Jupyter Notebook for the project can be downloaded: https://github.com/jwmathis/SSE591\_Project5. In order to run the file, Python and other dependencies must be installed.

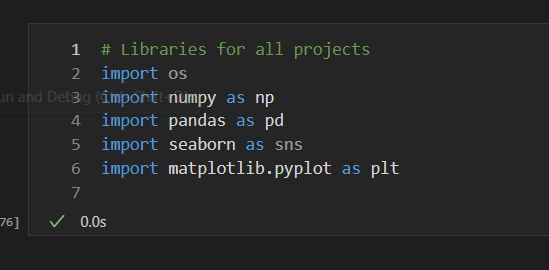
# 1. Introduction

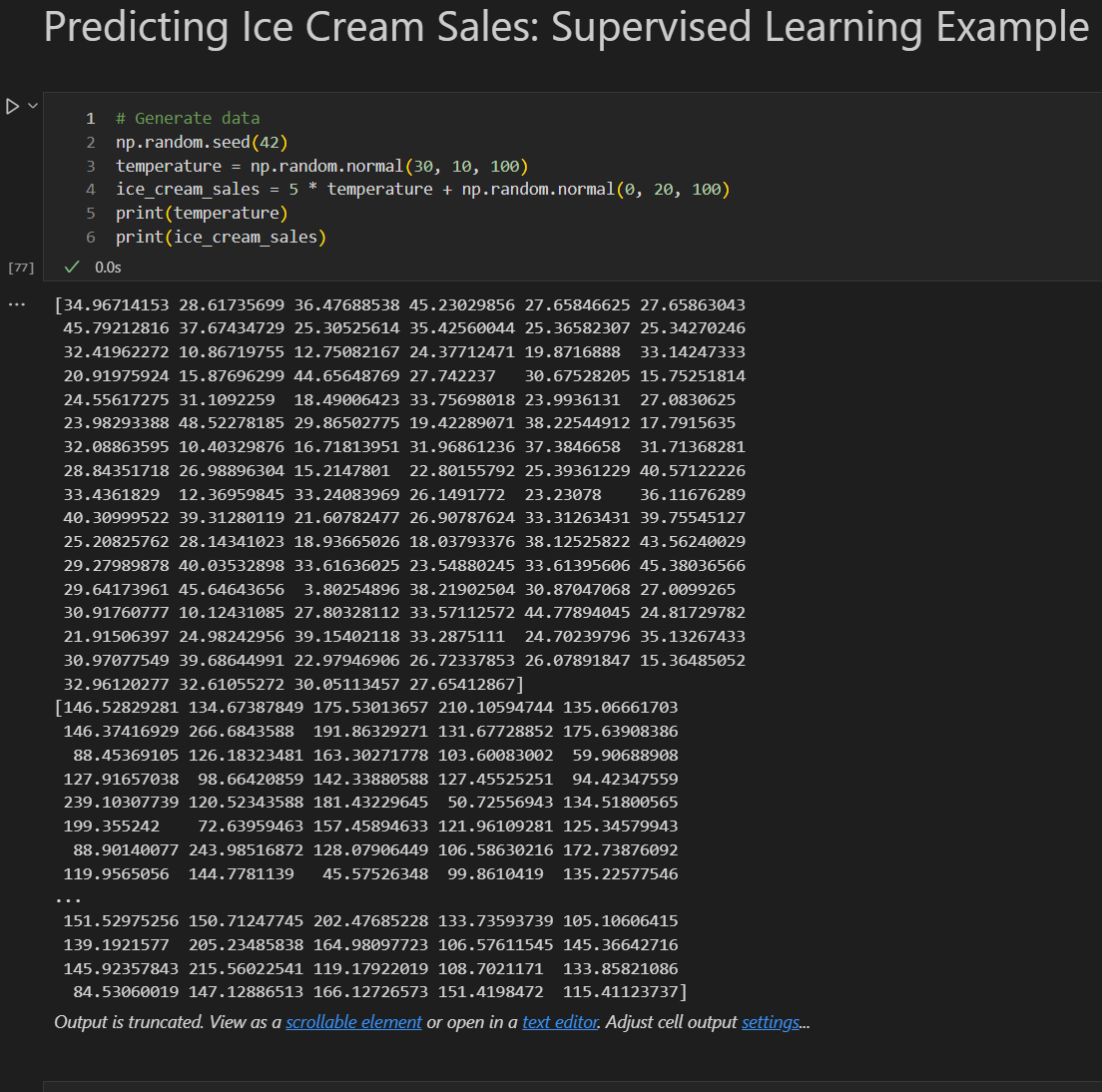
Machine learning focuses on developing mathematical models that enable computers to learn and improve without being explicitly programmed but by having the program be fit to data (Vanderplas, 2016). In general, the more data that is available, the better the model and the performance. Machine learning has become an invaluable tool across various fields that enable the extraction of valuable insights from vast amounts of data. This report delves into several projects that serve as practical examples of machine learning applications. By using the foundational concepts and methodologies described in Jake Vanderplas’ book Python Data Science Handbook (2016), this report presents my hands-on exploration of machine learning.

Jake Vanderplas focuses on many practical implementations of machine learning algorithms and attempts to provide a comprehensive guide for understanding and applying machine learning techniques by using various examples. He broadly categorizes machine learning into two main branches: supervised and unsupervised learning. This report follows his presented structure closely and illustrates supervised and unsupervised learning algorithms, feature engineering, model validation and many more concepts. The code presented in this report was developed using Visual Studio Code with Jupyter Notebook extensions. I will provide detailed explanations, highlighting key features and operations that make machine learning an essential tool for data analysis.

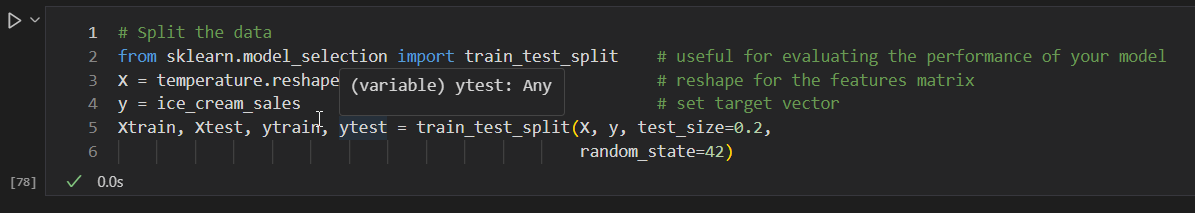
# 2. Predicting Ice Cream Sales Linear Regression

My first example I demonstrate how to perform a simple linear regression using SciKit-Learn from data I generated to model evaluation and visualization. For this I decided to model ice cream sales and correlate it to the temperature. This example represents a supervised regression learning model where the output labels are continuous values. To generate the data I use *np.random.seed(42)* to generate random numbers and ensure that the same numbers are generated every time. I use *np.random.normal(30, 10, 100)* to generate temperature values with a mean of 30C and standard deviation of 10C. To better simulate ice cream sales, I use a linear function of temperature with added random noise. Figure 2 shows the code and results below.

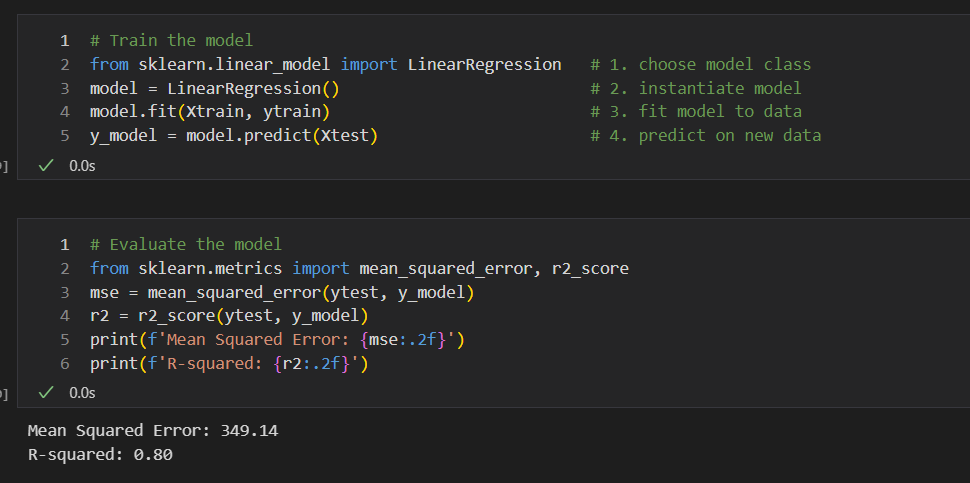
Figure 1: Libraries

Figure 2: Generating Ice Cream Data

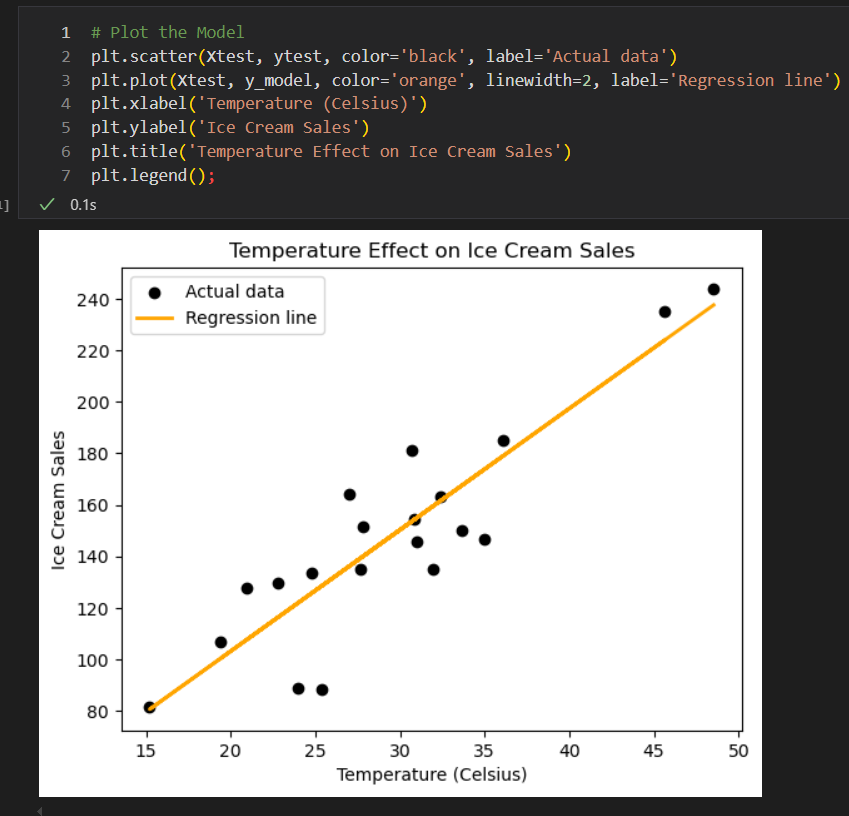
To begin the machine learning, I first define my features matrix and target vector by reshaping the temperature array to a 2D array. Then I import *train\_test\_split*  to split the data into training and testing sets. Figure 3 shows the code defining the variables.

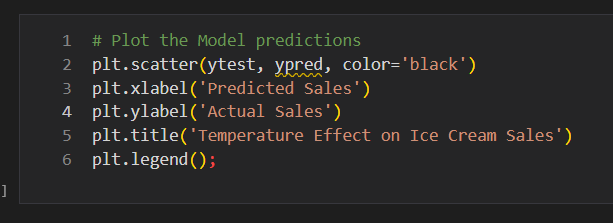
Figure 3: Defining features matrix and target vector and spliting the data

Afterwards I follow the procedure of training the model as outlined in chapter 5. I choose my model, in this case *LinearRegression,* I instantiate the model, then I fit the model to the training data, *Xtrain and ytrain.* Lastly, I use the trained model to predict ice cream sales on the test data. Next I evaluate the model by importing the metrics *mean\_squared\_error t*o computethe mean squared error between actual values and predicted sales and  *r2\_score* to compute the R-squared value to have some indication of the proportion of the variance in the dependent variable that is predictable form the independent variable. Figure 4 shows the code and the output.

Figure 4: Training and Evaluating LinearRegression Model

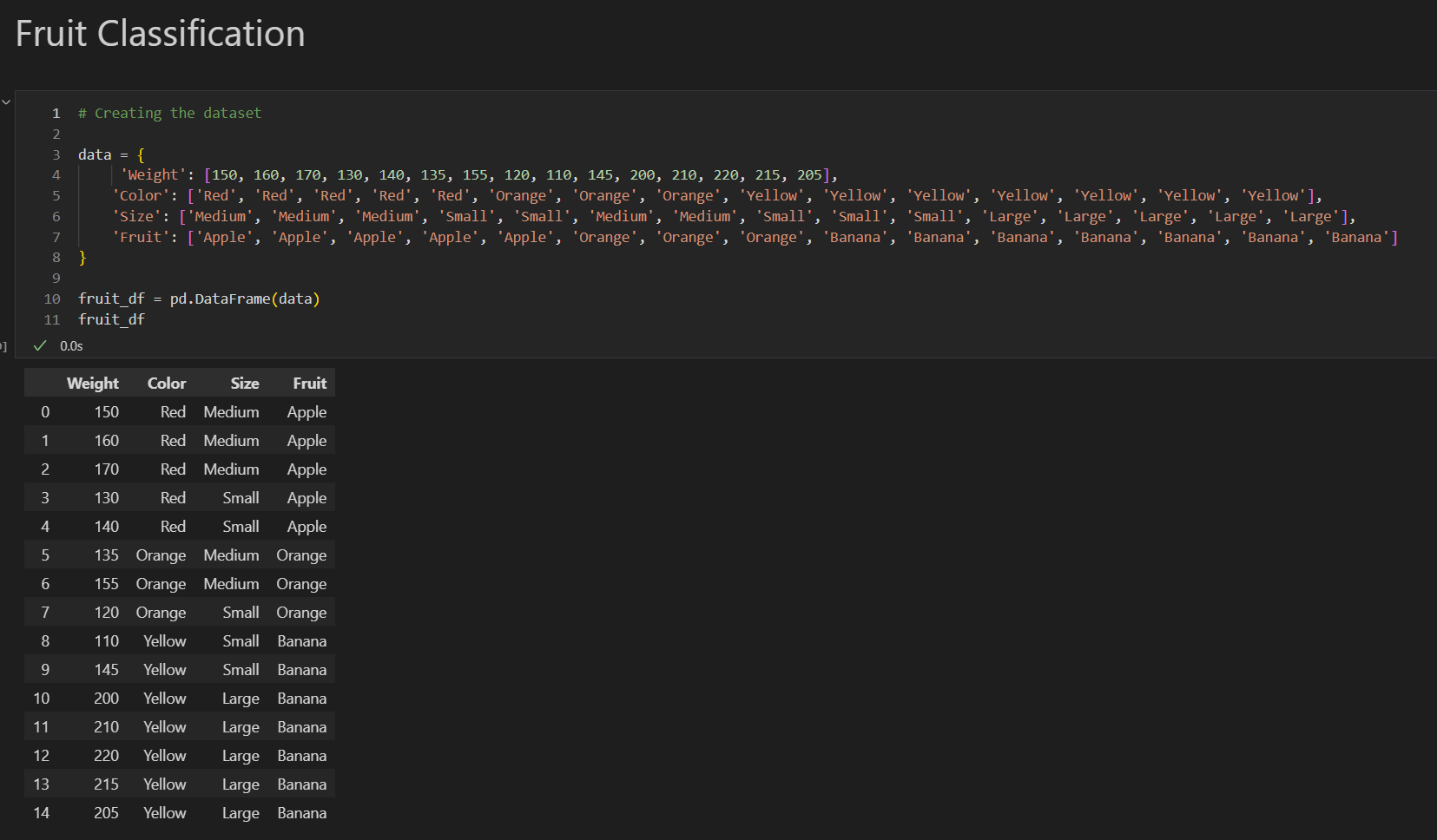
Lastly, I visualize the results using Matplotlib’s *plt.scatter* and *plt.plot* to plot the actual test data points and the regression line based on the model’s predictions. From this example I was able to walk through the process of generating synthetic data, splitting the data into training and testing sets, training a linear regression model, and evaluating and visualizing the performance and results. Figure 5 shows the code and the output.

Figure 5: Plotting the LinearRegression Model

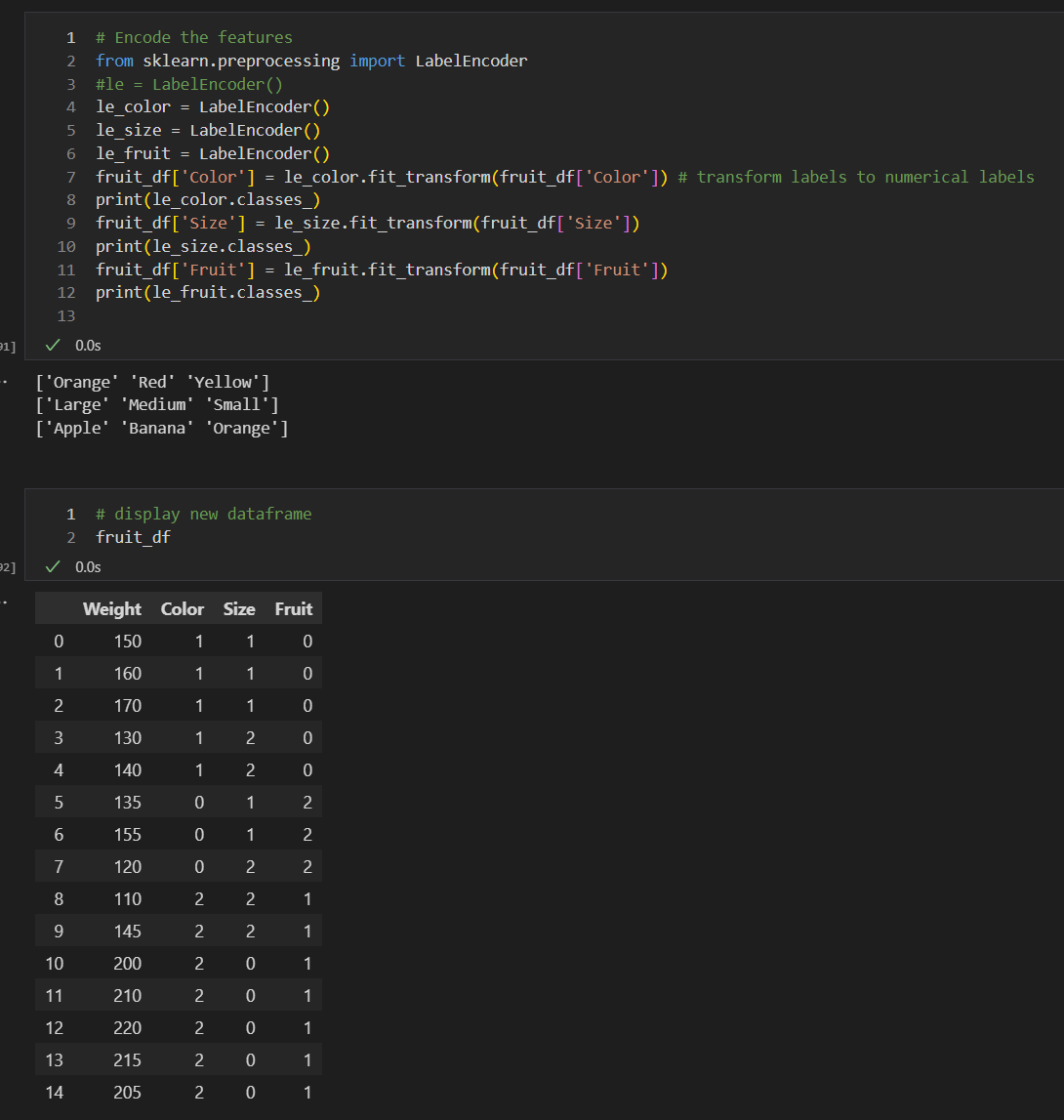
Figure 6: Plotting predicitons

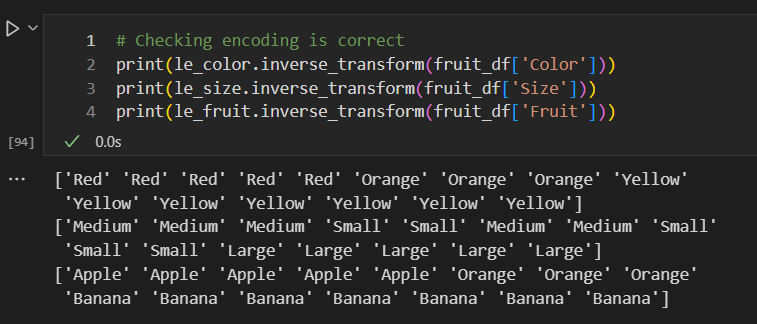
# 3. Fruit Classification

For my next example, I demonstrate how to process categorical data by making a simple fruit supervised classification learning program. For this type of example, the ouptut labels will be categorical as the model learns to assign inputs to one of the predefined categories. To begin, I constructed a DataFrame called *fruit\_df* containing columns called *Weight, Color, Size, and Fruit* containing the data of information that I defined in a dictionary. This example served as my means of better understanding how to convert and process data to be used for machine learning models. Figure 7 shows the code and output.

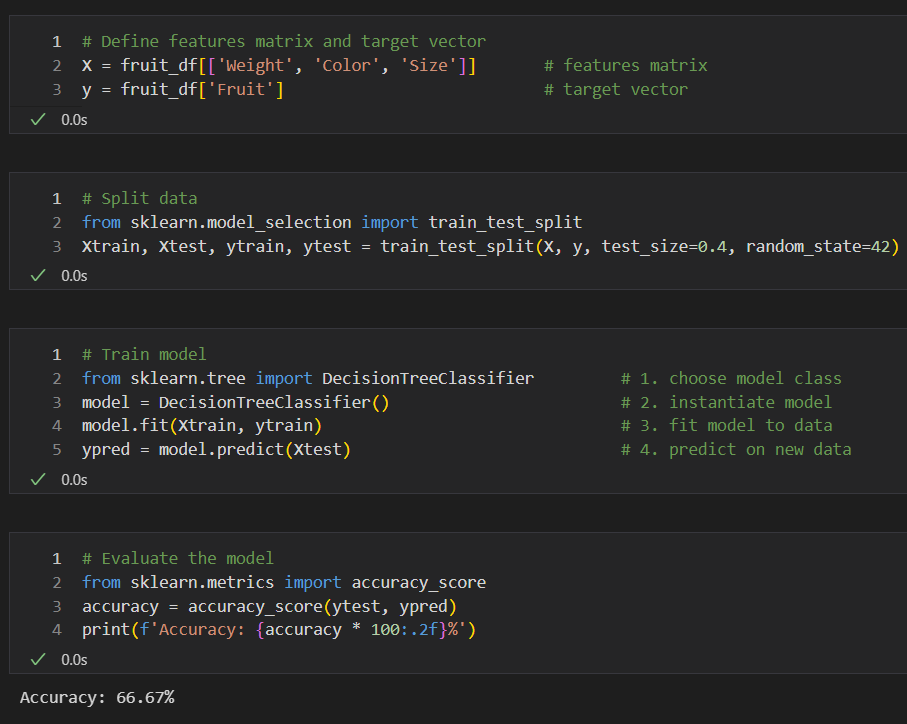
Figure 7: Generating data for Fruit Classification

I used *Labelencoder* to encode the features of the fruits from categorical variables into numerical labels. Initially, I used *le = LabelEncoder()*  for all the features. Unfortunately, this resulted in errors as I quickly learn that LabelEncoder() stores only the most recent information. So, when I would try to decode the information, I would get incorrect results. To fix this, I created three separate variables to encode the features. Using the *fit\_transform* attribute I transformed the columns of Color, Size, and Fruit. To ensure the DataFrame was properly encoded, I printed the decoded versions to the screen using the attribute *inverse\_transform.* This allowed me to verify the correctness of the encoding by inversely transforming the numerical labels back to their original values. Figure 8 and 9 shows the code and the output.

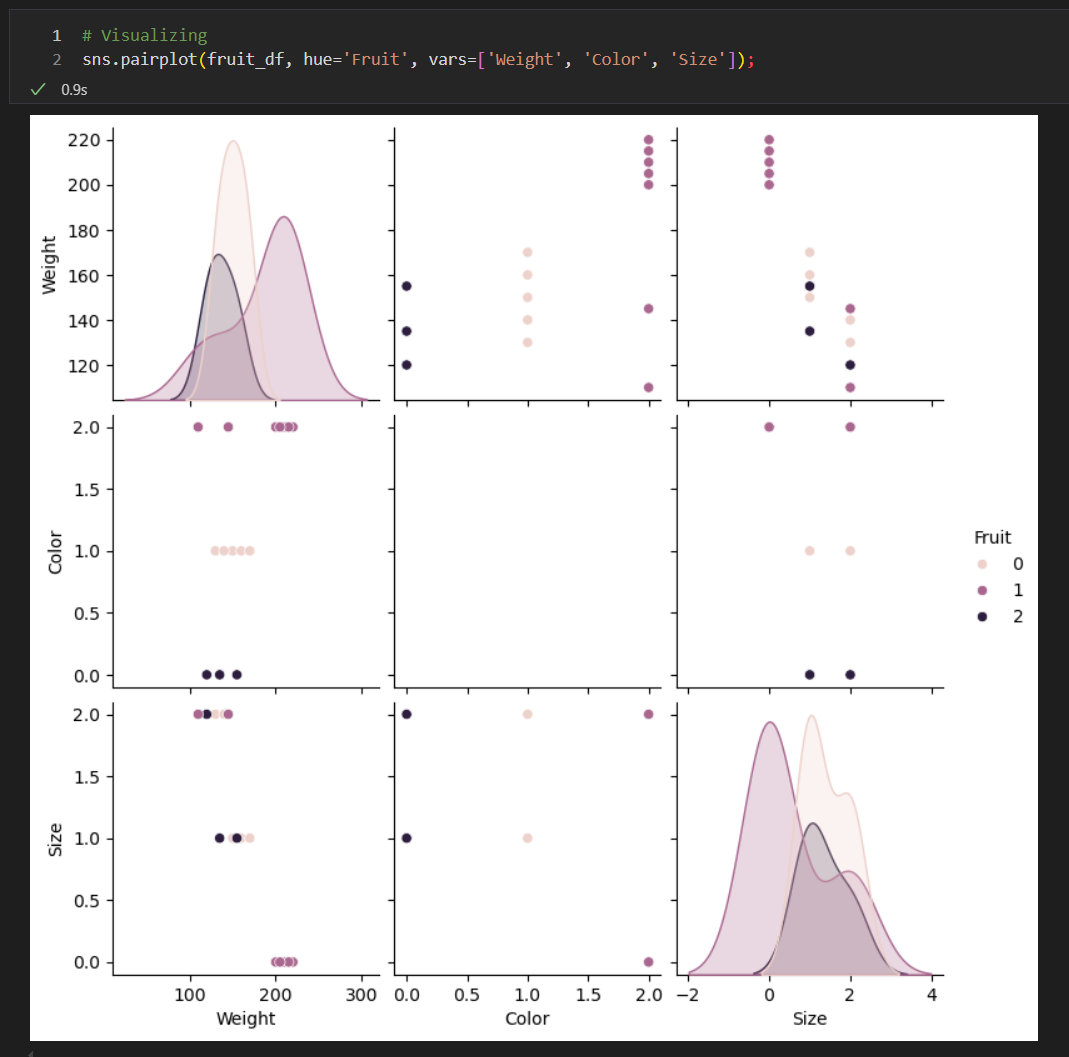
Figure 8: Encoding fruit\_df features

Figure 9: Decoding numerical categories

Now that the data is processed and in a format that can be used for machine learning, I begin the process of training a model as outline in chapter 5. I define my features matrix and target vector; split the data into training and testing sets. For this example, I instantiate a decision tree classifier and fit the training data to this model and predict on the test data. For this example, I use *accuracy\_score*  to evaluate the performance of the model. Figure 10 shows the code and output.

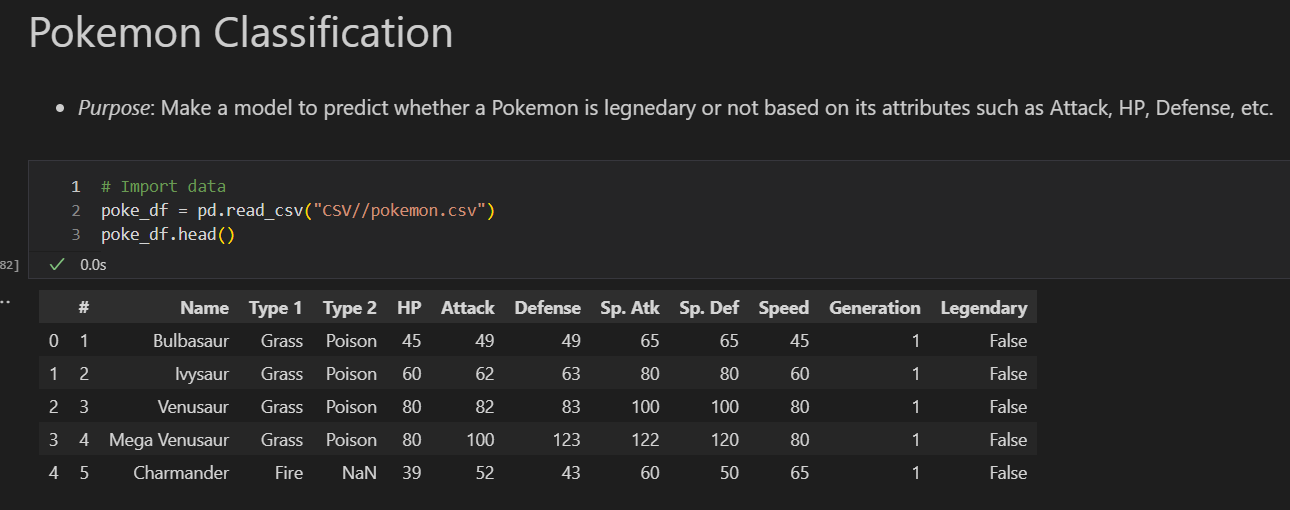
Figure 10: Training and evaluating a DecisionTreeClassifier model

I use Seaborn’s *pairplot* to visualize the relationships in *fruit\_df*. Each feature (*Weight, Color, Size)* is plotted, with points colored by *Fruit* category. This example was simple but it allowed me begin understanding the process how to handle categorical data and build a simple machine learning model. Figure 11 shows the code and output.

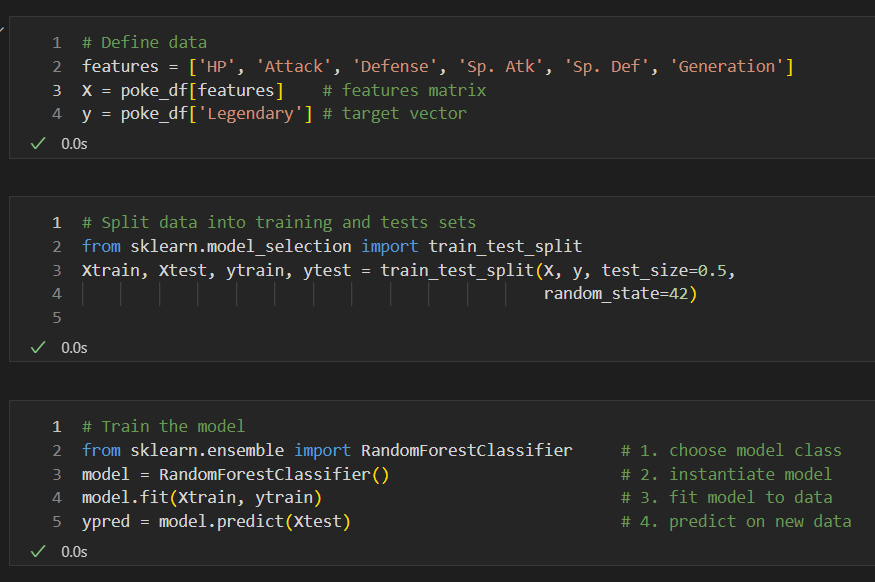
Figure 11: Pairplot to visualize fruit features with the fruit category

# 4. Pokémon Classification

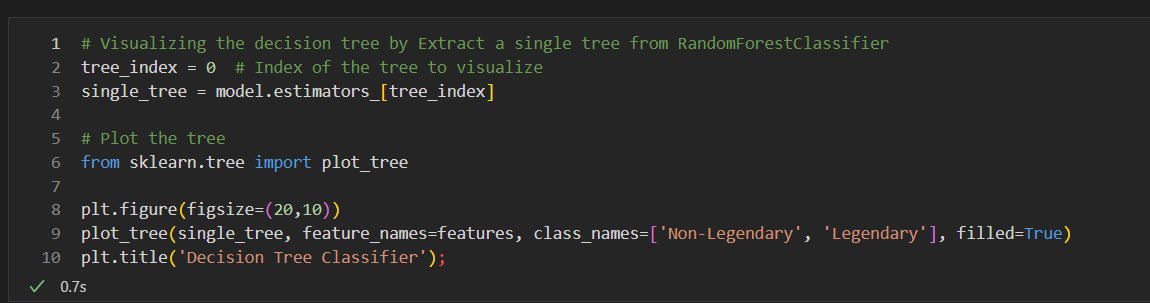
For the next example, I attempt to showcase another supervised classification learning algorithm by classifying Pokémon based off their legendary status. First I load the Pokémon data obtained from Kaggle and display the first few rows as shown in figure 12.

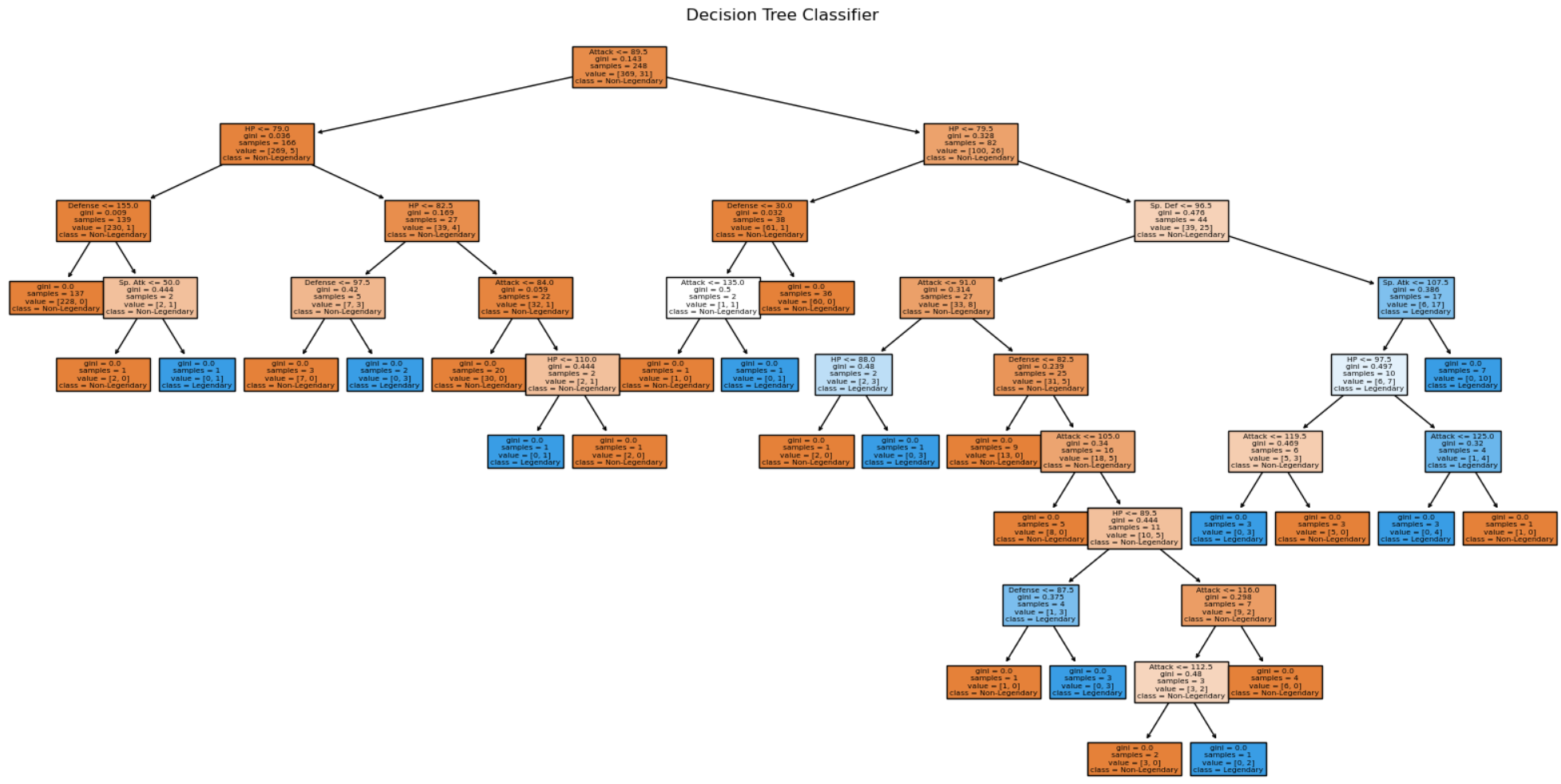
Figure 12: Importing Pokemon dataset

Then I begin to define my features matrix using the columns *HP, Attack, Defense, Sp. Atk, Sp. Def, and Generation* and the target vector as the *Legendary* column and split the data using *train\_test\_split.* Then I train a *RandomForestClassifier* model and make predictions on the test data as shown in figure 13.

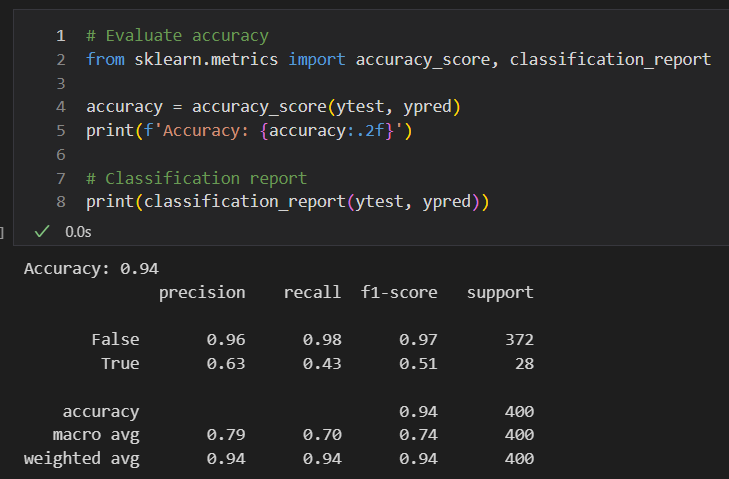
Figure 13: Training and evaluating RandomForestClassifier model

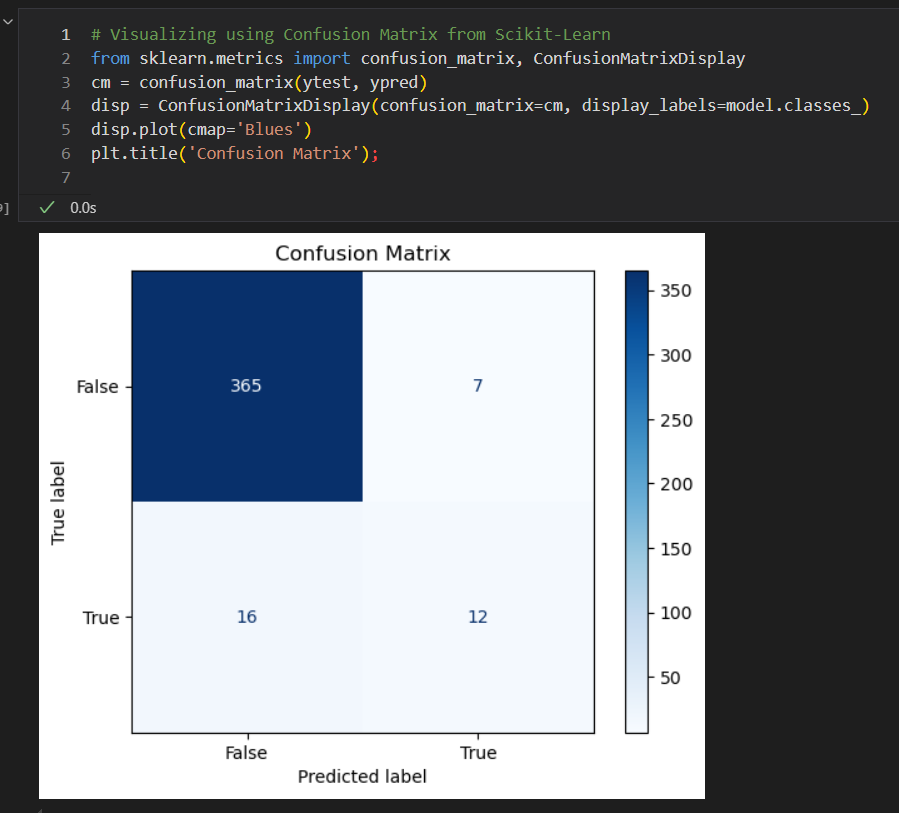
For visualization, I extract a single decision tree from the *RandomForestClassifier* model using the *estimators\_[]* attribute. Then I used *plot\_tree* to plot the decision tree with features and class names as shown in figure 14 and 15.

Figure 14: Code to visualize a single tree

Figure 15: Visualizing the decision tree

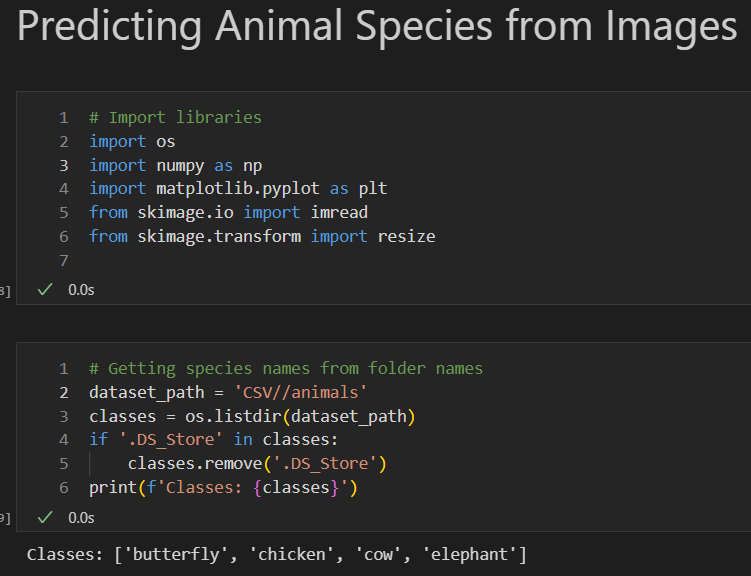
In order to properly evaluate the model, I import the metrics *accuracy­\_score* to calculate the proportion of correct predictions *and classification\_report* to provide a more detailed analysis of the performance of the model. Then I visualized the model using a *confusion\_matrix and ConfusionMatrixDisplay.* Figure 16 and 17 show the code and output.

Figure 16: Evaluating RandomForestClassifier model using accuracy score and classification report

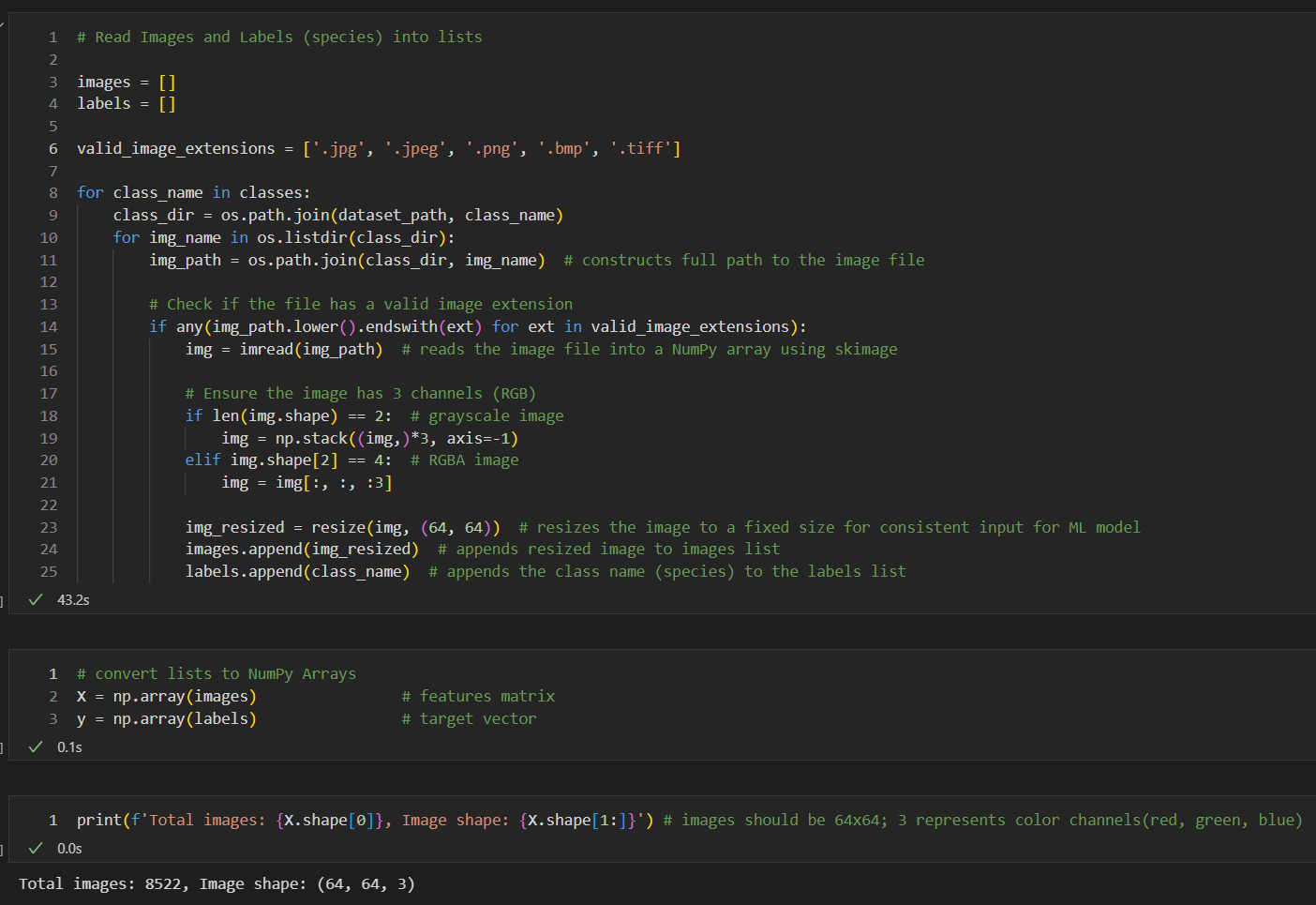
Figure 17: Visualizing confusion matrix

# 5. Predicting Animal Species

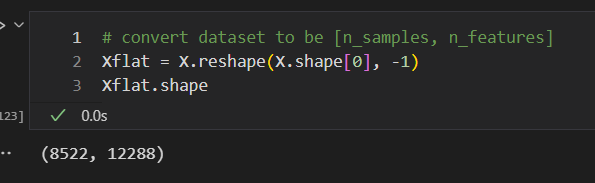
For my last project I tried to make a comprehensive example using image classification and clustering tasks. First I import the necessary libraries such as *numpy, matplotlib.pyplot,* *imread and resize.* With proper libraries imported, I then load my images folder, *animals-10* and get the class names from the folder names. Because I was using a MacBook when coding this section, a macOS system file was present so in order to deal with this being included as a class name, I use a conditional statement to remove it from the list if it is present. Figure 18 shows the code and output.

Figure 18: Importing libraries and extracting species names from folder names

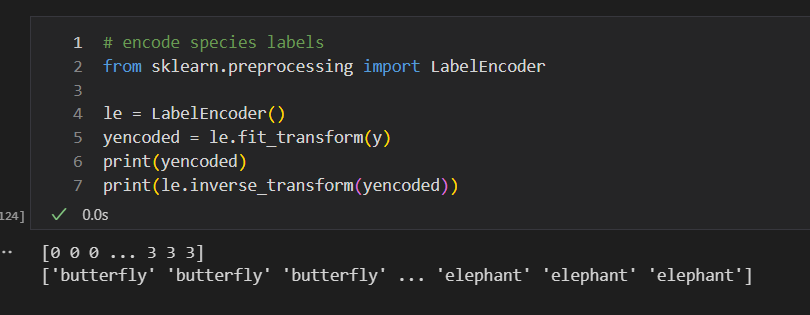
Next I begin pre-processing the images to ensure that are in a cohesive format for machine learning. I use a nested for loop with conditional statements to read images and their corresponding labels into two separate lists. Additionally, the images are converted to 3-channel images. Once the images and labels lists are completed, the lists are then converted into NumPy arrays. The images array is defined as the features matrix and the labels array is defined as the target vector. Figure 19 shows the code and the output.

Figure 19: Converting images and defining features matrix and target vector

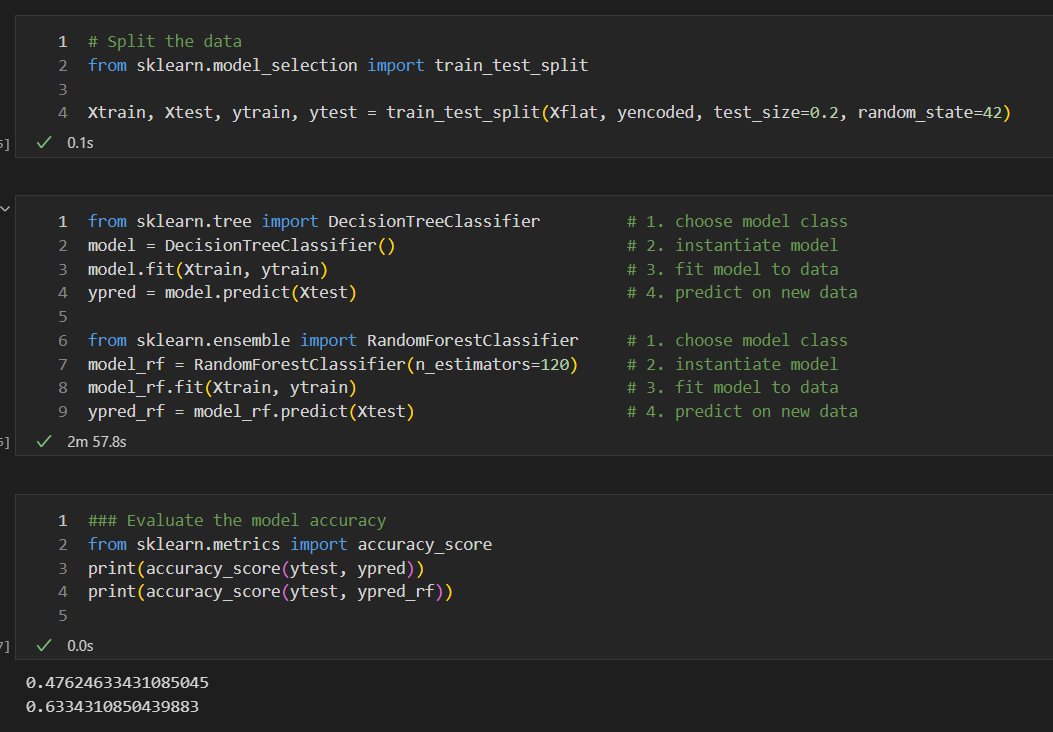
For machine learning, the features array is typically in the format of [n\_samples, n\_features]. By using images, the NumPy array is in the format of (64, 64, 3) as seen above in figure 19. The first two numbers represent the width and height of the picture, and the last number represents the color channels (red, green, blue) of the image. In order to ensure the array is in the correct format, I reshape it using *reshape(),* to flatten the array into a one dimensional array as shown in figure 20.

Figure 20: Flattening features matrix

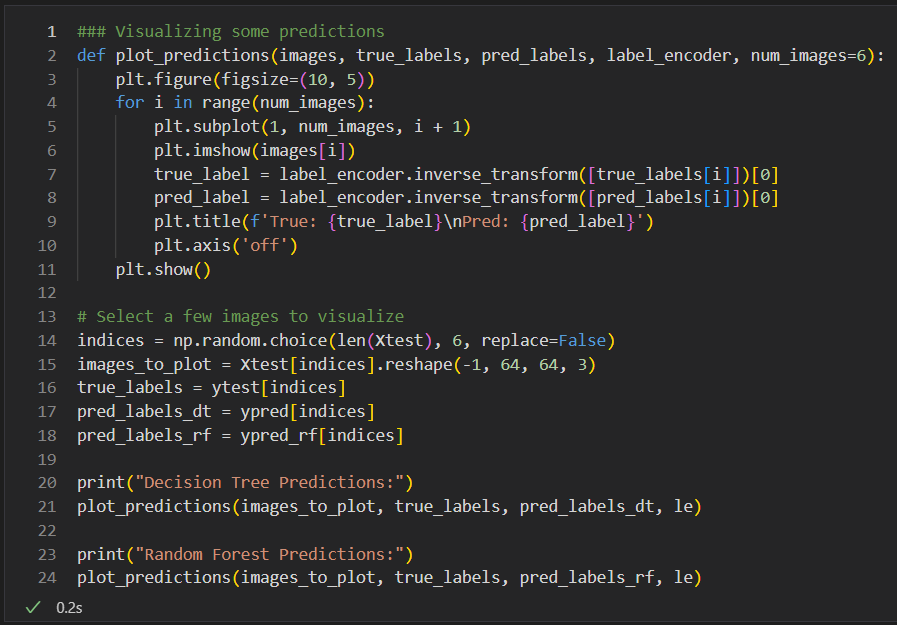
Next, I use *Labelencoder()* to encode the species labels into integers. To be sure this was done correctly, I print the encoded labels and their original forms. Figure 21 shows the code and the output.

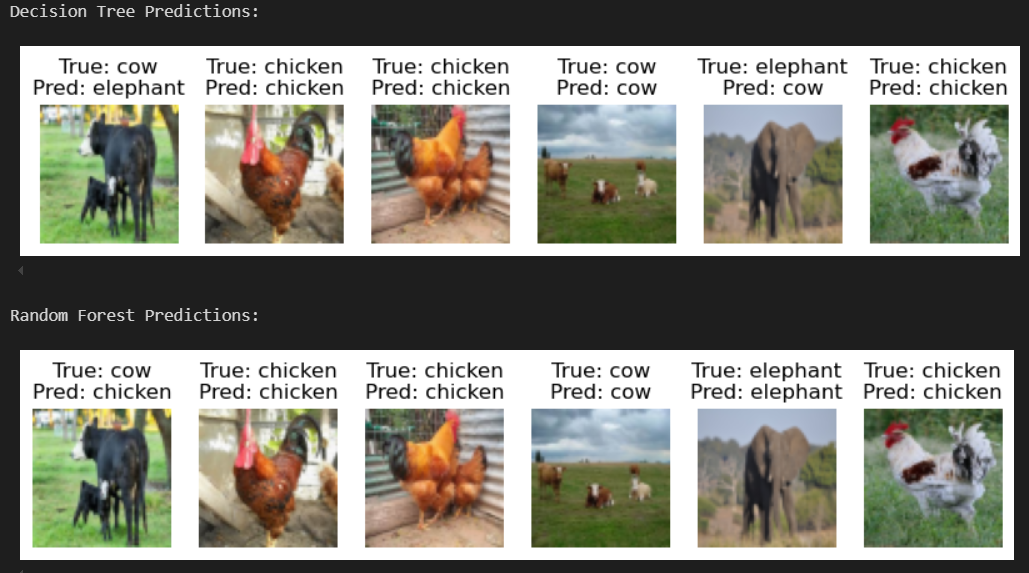
Figure 21: Encoding target vector

Now that the images have been pre-processed, I use *train\_test\_split*  to split the data with 20% of the data for testing, and 80% of the data for training. With the data split, I then choose my model, instantiate the model, fit the data to the model, and then make predictions on the test data. For this example, I chose both *DecisionTreeClassifier()* and *RandomForestClassifier().* With the model trained, I evaluate the accuracy of each model. The *DecisionTreeClassifier* had an accuracy of 48% and the *RandomForestClassifier* had an accuracy of 63%. Figure 22 shows the code and the output.

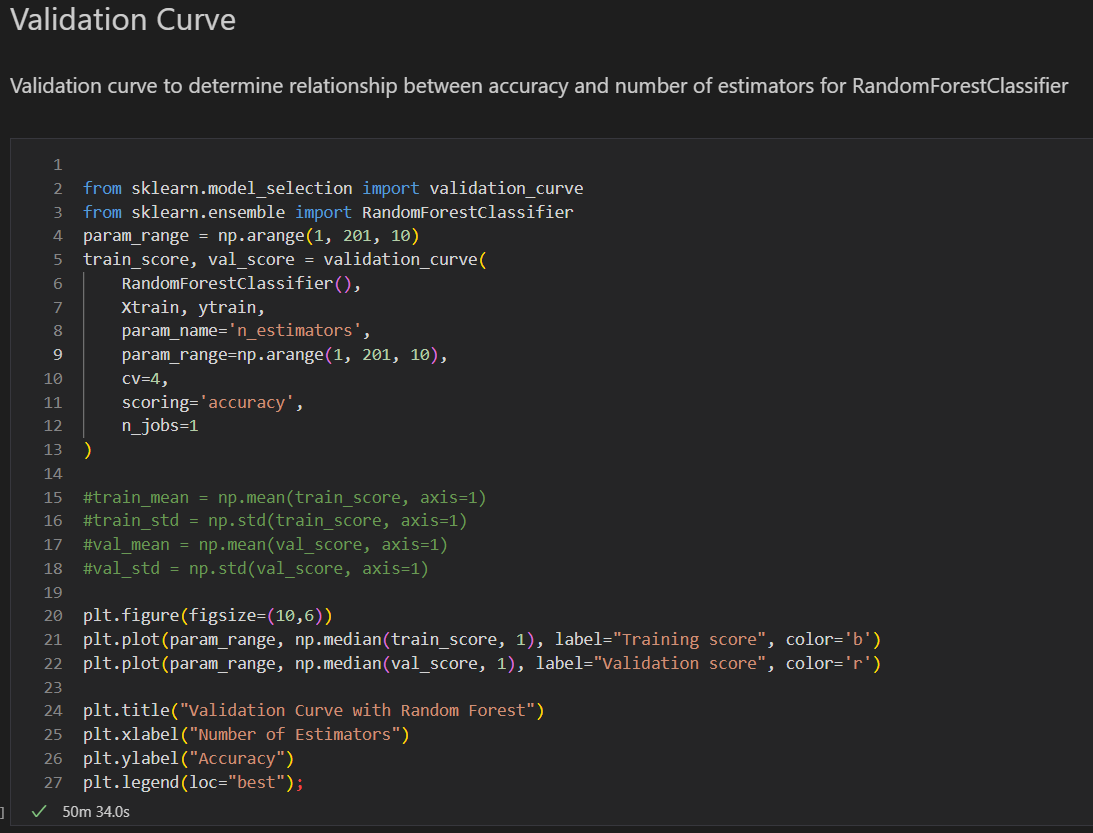
Figure 22: Testing and evaluating RandomForestClassifier for Animal Prediction

To visualize the results, I visualize some of the predictions for each model. From this visualization, you are able to see how accurate the model is able to determine what species a particular picture is showing. Figure 23 and 24 show the code and the output.

Figure 23: Code to visualize results of the RandomForestClassifier

Figure 24: Animal Species Predictions of the RandomForestClassifier model

From the accuracy score and the visualization, we are able to conclude that the RandomForestClassfier is a little more accurate than a single decisiontreeclassifier. In an attempt to improve the accuracy, a validation curve was plotted of the *RandomForestClassifier* model to view the relationship between the accuracy and the number of estimators. The training score is higher than the validation score. Increasing the model complexity does improve the training score, and around 20 estimators, the training score is at its best and any higher complexity does not show any additional benefits. The validation score reaches a maximum around 120 estimators and though it shows some fluctuations with increasing complexity, the validation score essentially levels off and does not show any improvement. There is a consistent space between the two lines showing that it is a high-variance model and is therefore over-fitting the data. Figure 24 and 25 shows the code and the validation curve. I also plotted a learning curve to analyze the performance of the R*andomForestClassifier* as the training size varies. By increasing the training size the better the model becomes as the validation score begins to converge with the training score. However, from this model, it is not evident at what point it truly begins to converge. Figures 27 and 28 show the code and the output.

Figure 25: Code to construct a validation curve for RandomForestClassifier model

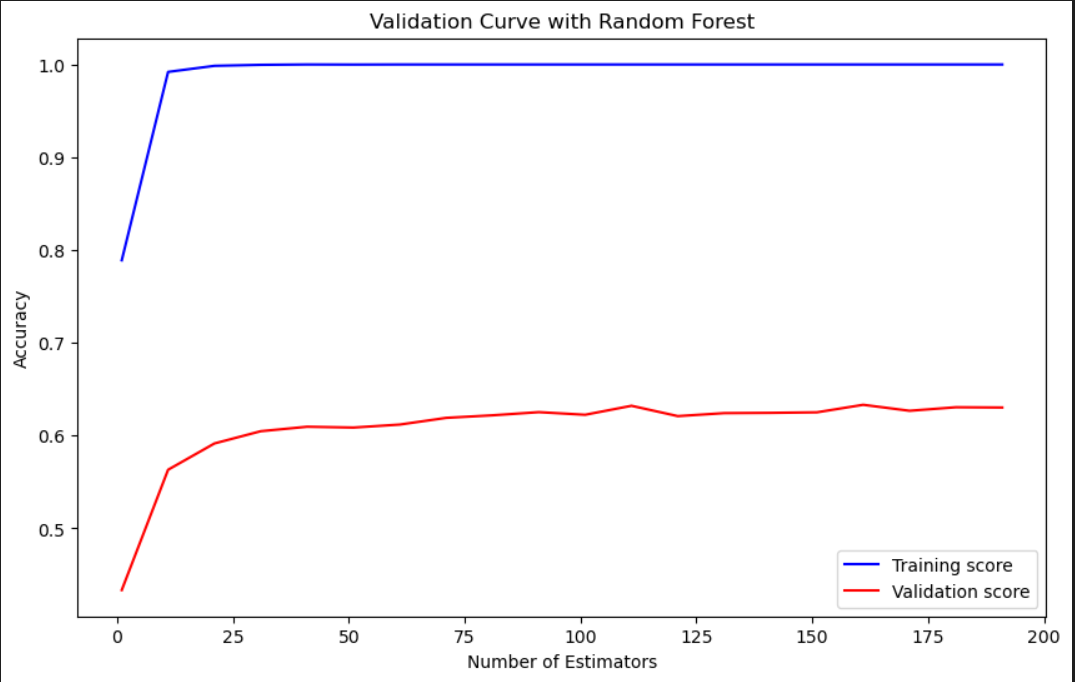
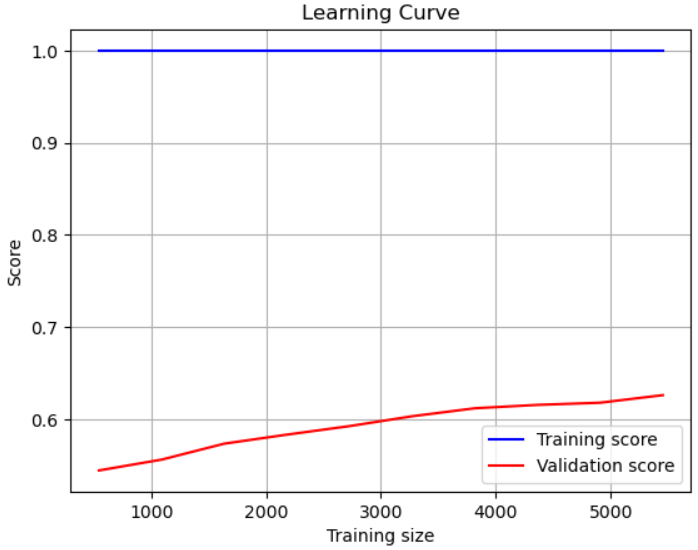
Figure 26: Validation Curve plotting the relationship between the accuracy and the number of estimators

Figure 27: Code to plot learning curve for RandomForestClassifier model

Figure 28: Learning Curve for RandomForestClassifier model

To try to improve the model, I performed hyperparameter tuning using GridSeachCV to find the best parameters for RandomForestClassifier. I use *param\_grid* to define the hyperparameters to search and called *GridSearchCV* to perform grid search with cross-validation. The best hyperparamters found are then printed to the screen. Figure 29 and 30 show the results. With the new information, I train and evaluate an updated *DecisionTreeClassifier* and *RandomForestClassifier* model. Figure 31 shows the code and output. The accuracy score for both was printed to the screen and there was a slight improvement. The score for the decision tree model increased to 50% and the the score for the random forest model increased to 64%.

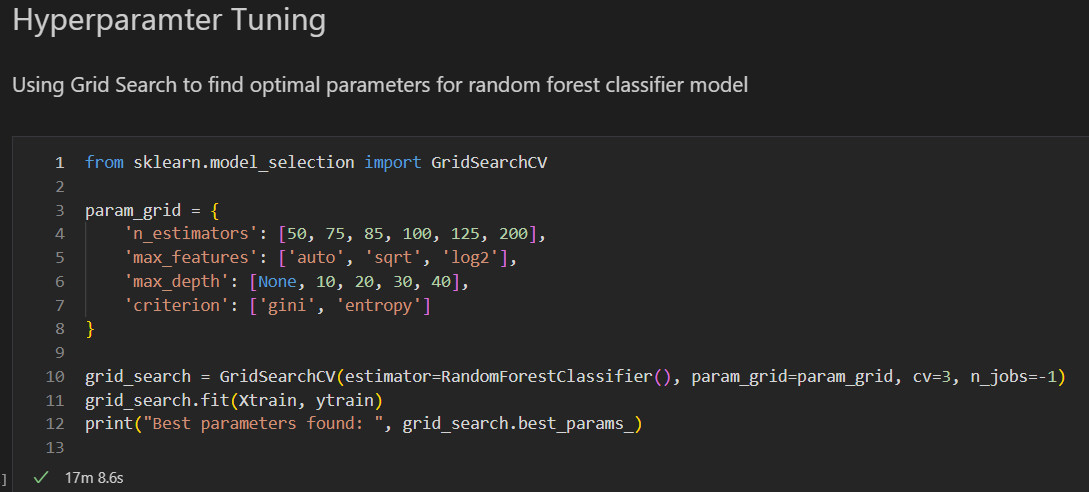
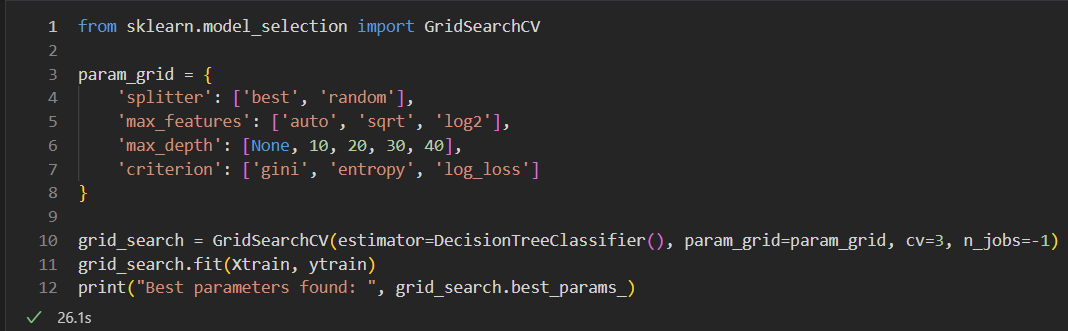
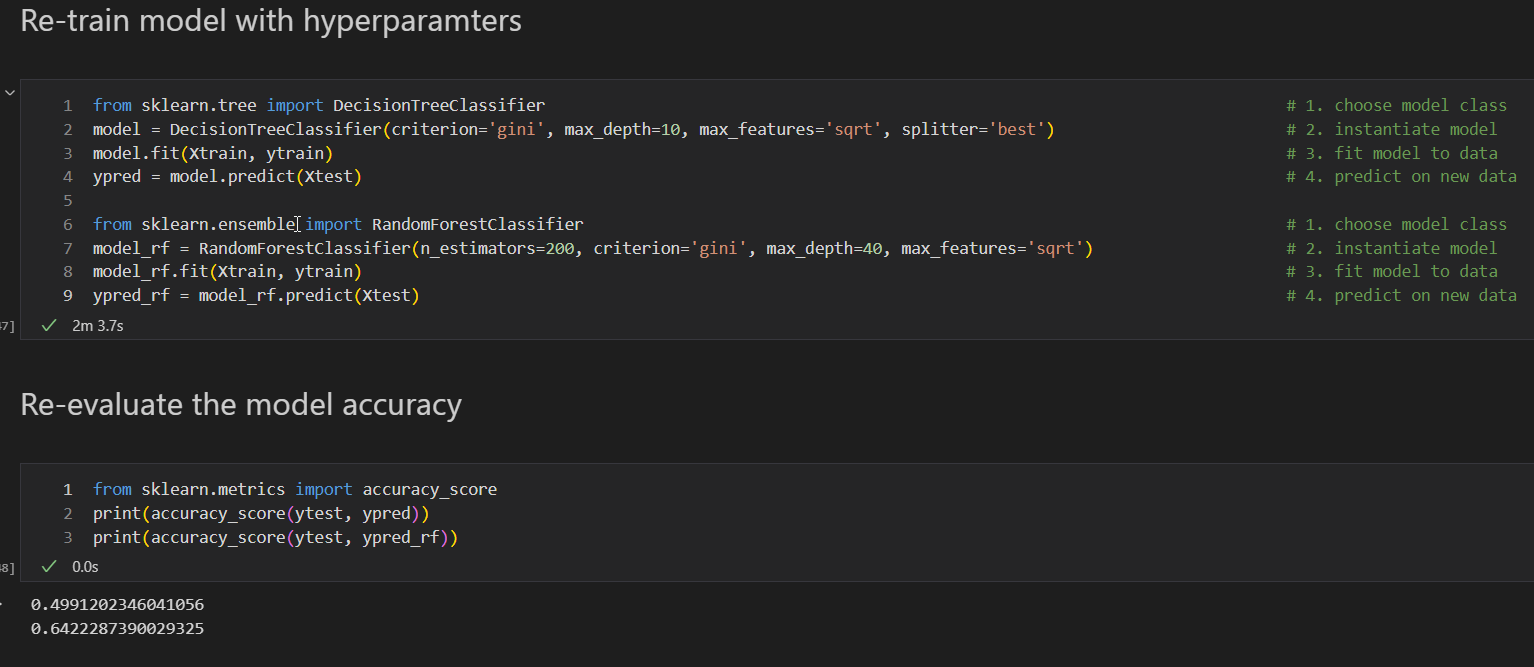


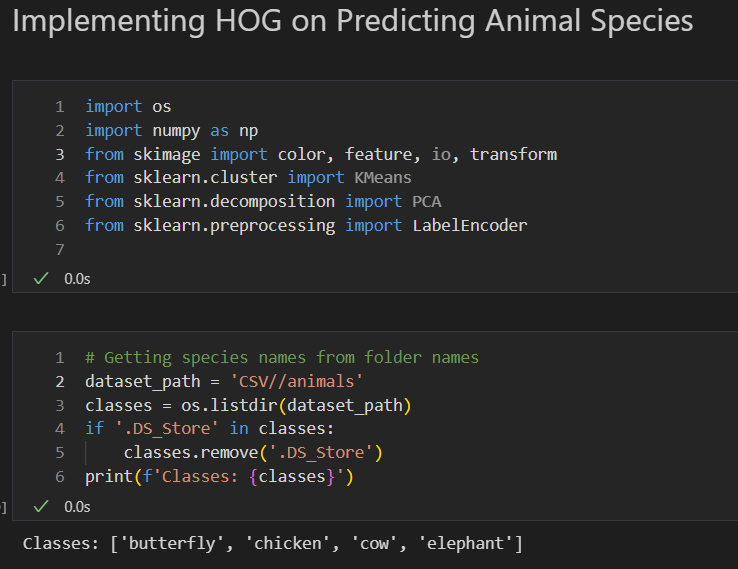
Figure 29: Hyperparameter tuning for RandomForestClassifier using GridSearchCV

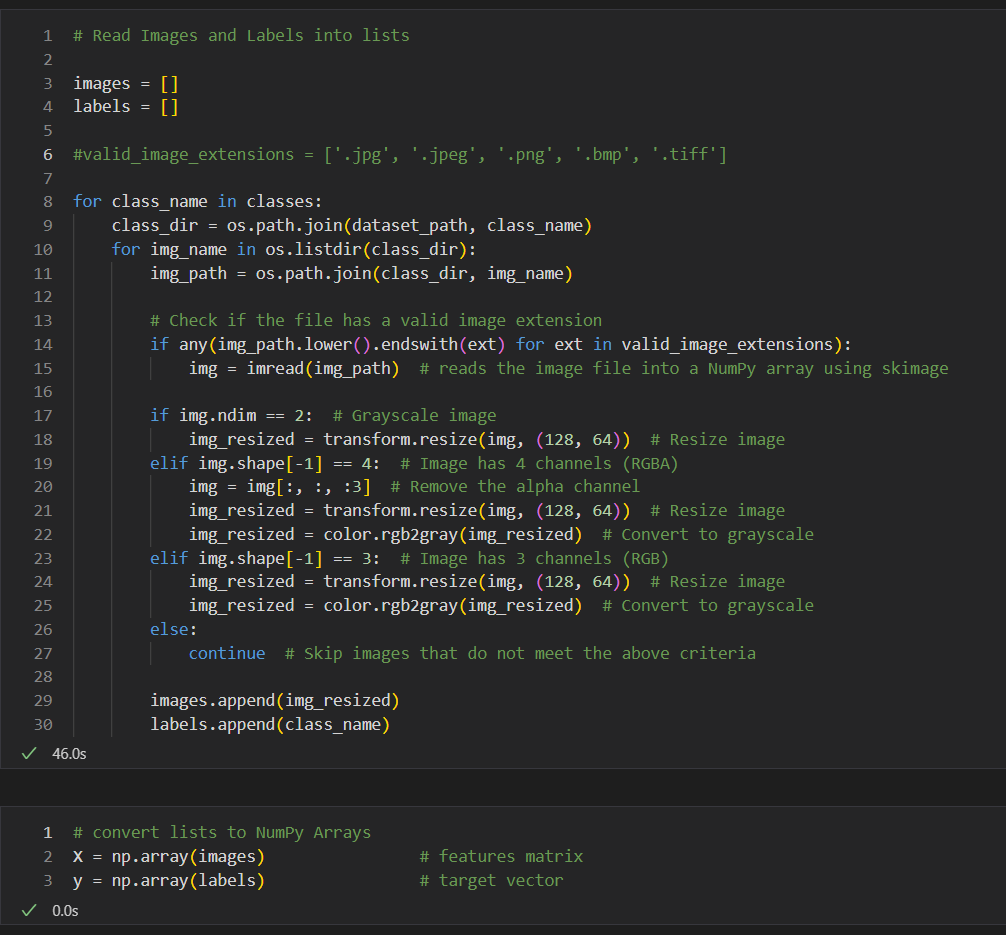


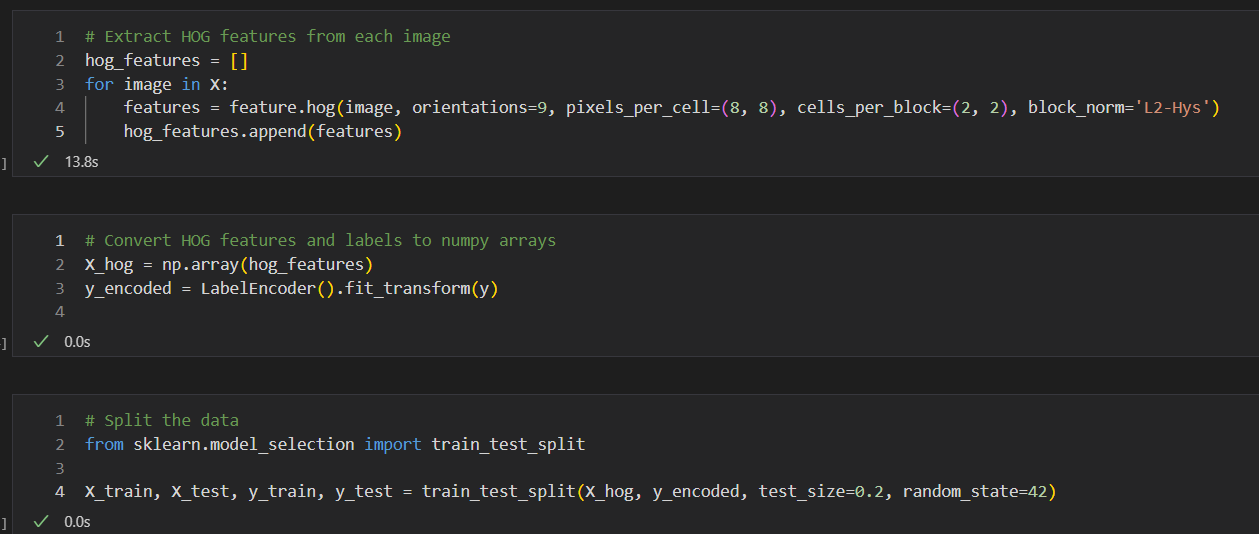
Figure 30: Hyperparameter tuniong for DecisionTreeClassifier using GridSearchCV

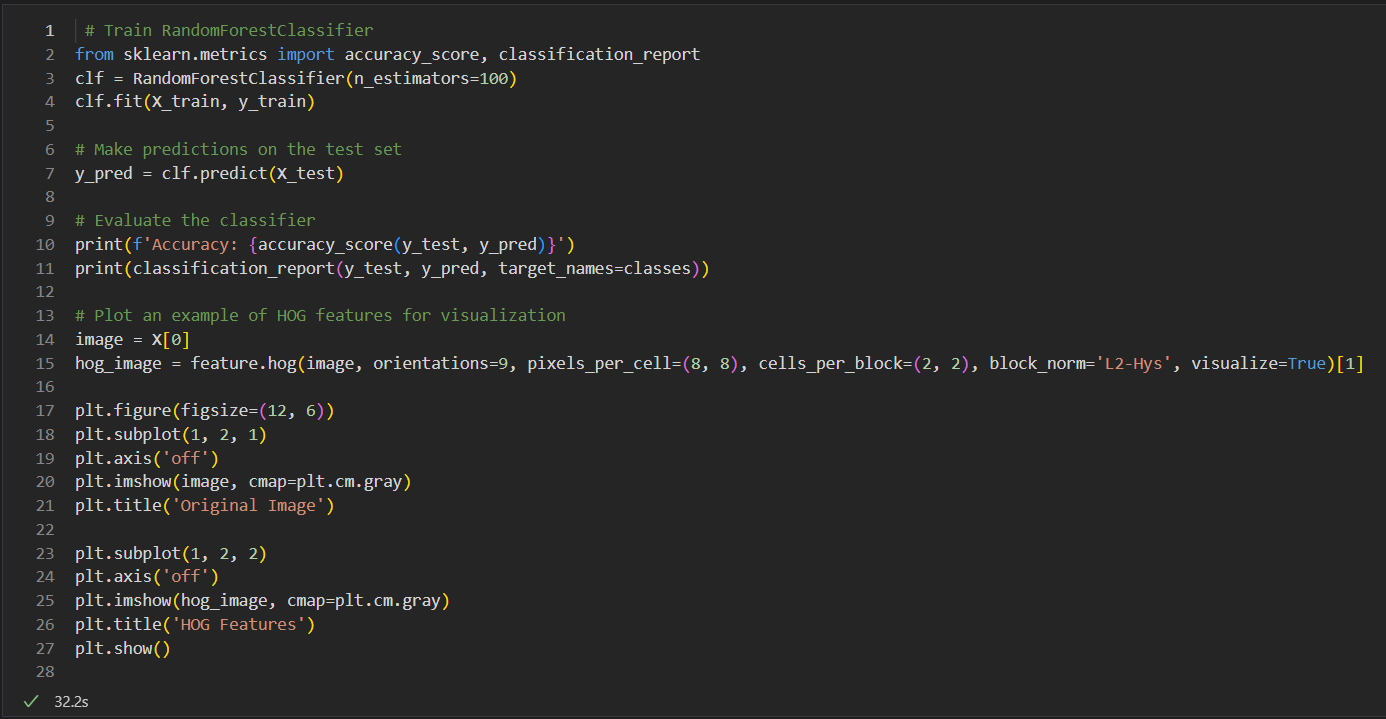
Figure 31: Re-training and evaluating update models with best parameters

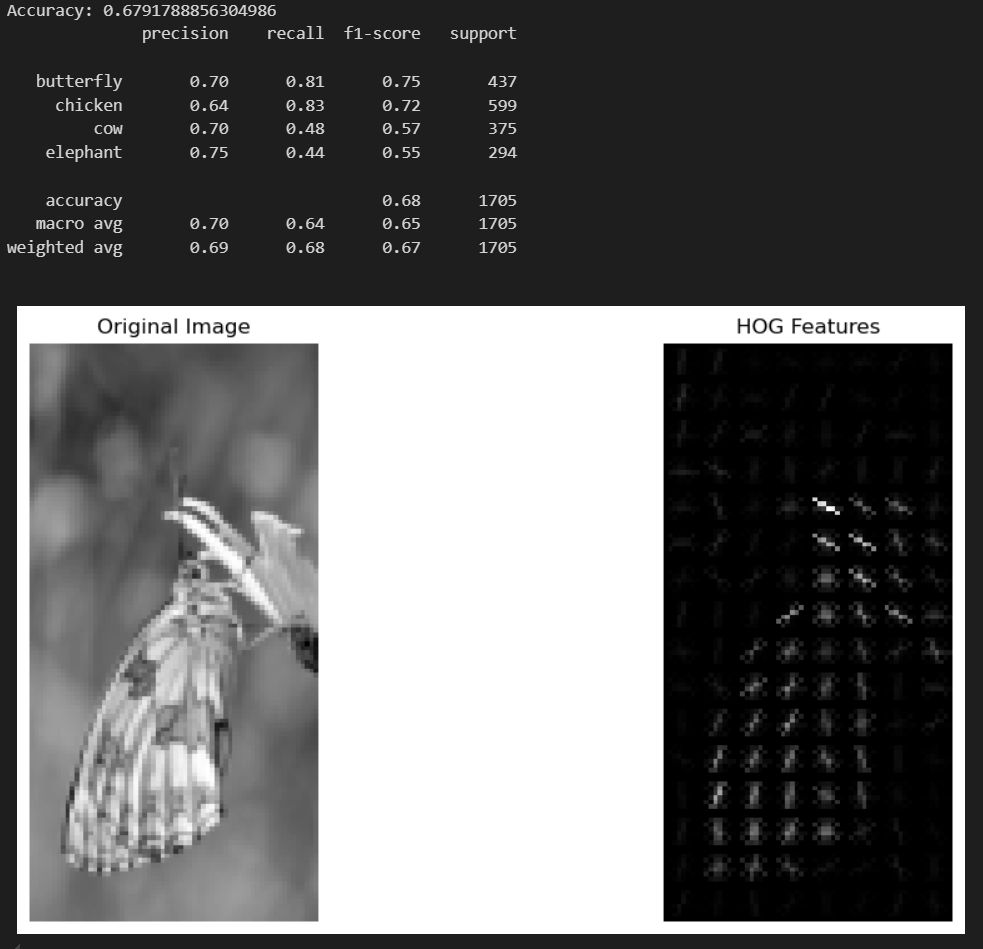
I then attempt to implement histogram of oriented gradients (HOG) to potentially improve the accuracy of the model since it can make the model a more robust representation of images and reduce the dimensionality of the model. For my implementation, I essentially perform the same functions as outline above. In the preprocess of the images, I convert every image to a grayscale (2 channel) image. I then extract the HOG features from each image and convert the HOG features and labels to NumPy arrays. I then split the data, and train the model and make predictions on the test set. Using HOG did show improvement in the model accuracy for the random forest model with an increase to 68% accuracy. Figures 32-36 show the code and outputs.

Figure 32: Loading libraries and importing dataset

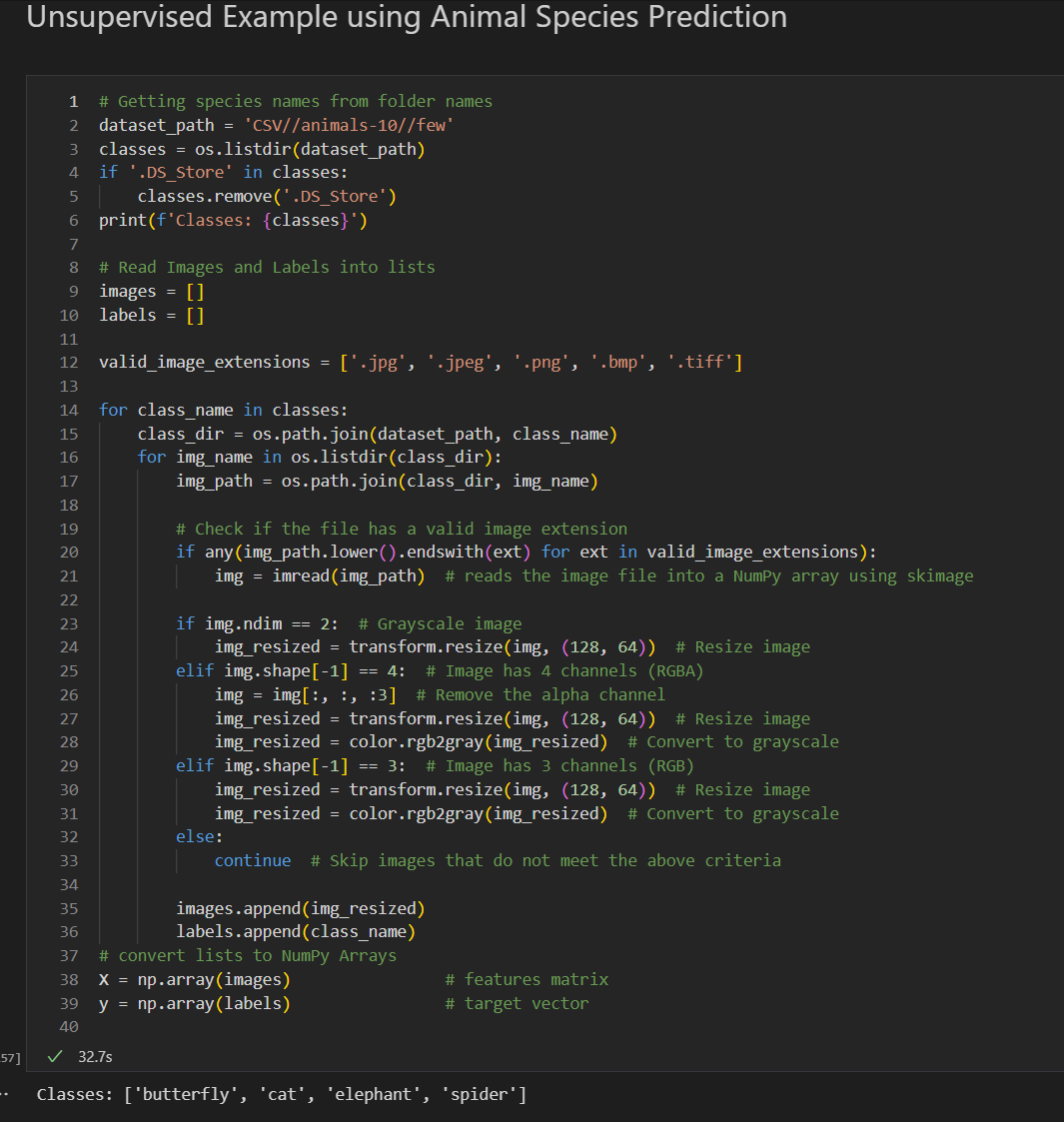
Figure 33: Converting images to grayscale and defining images and labels arrays

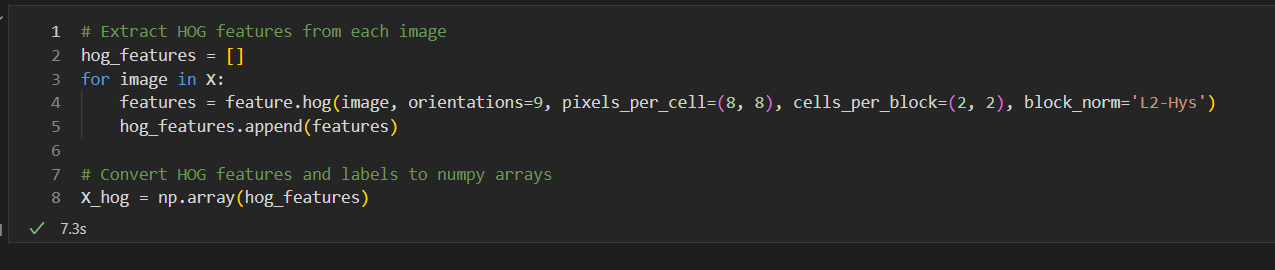
Figure 34: Extracting HOG features and defining features matrix and target vector and splitting data

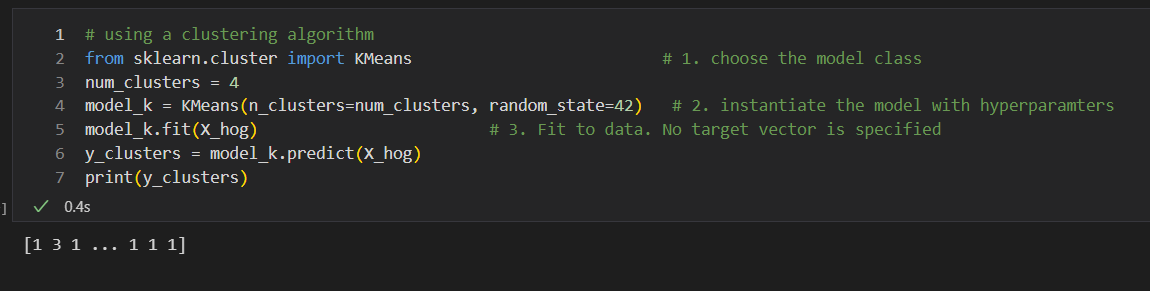
Figure 35: Code for training, evaluating, and plotting the model

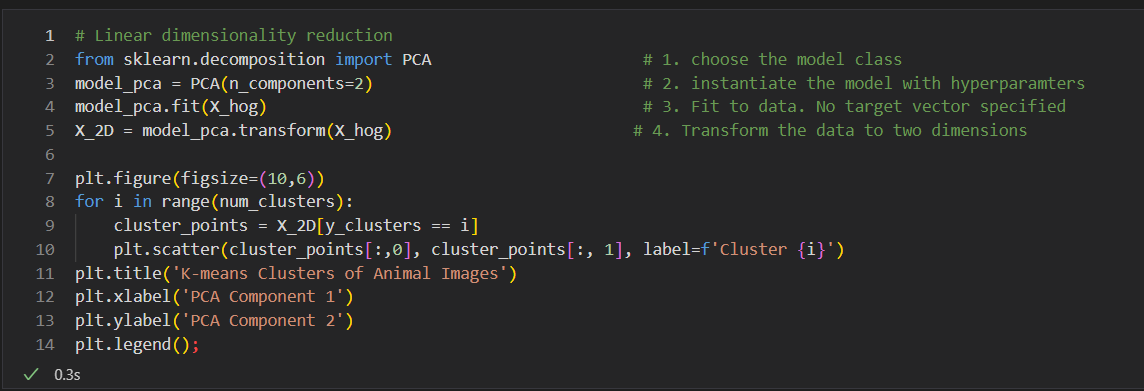
Figure 36: Model Evaluation output and visualization

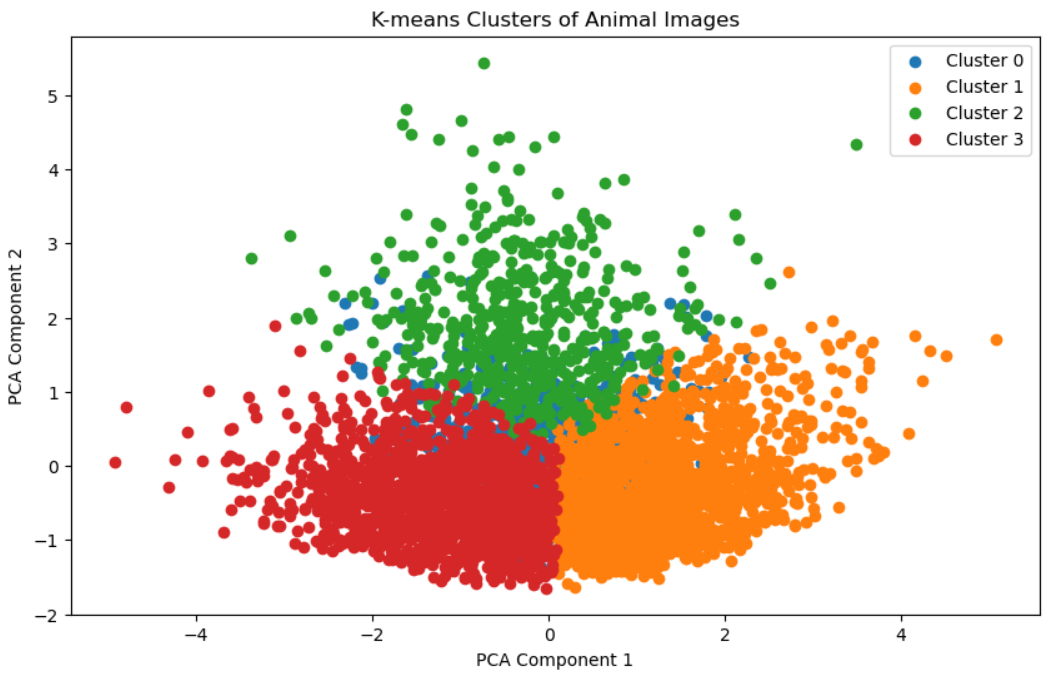
So far, all the examples have been either supervised regression or classification examples. To showcase an unsupervised example, I reconstruct the Animal Predictions code and use clustering and dimensionality reduction to implement an unsupervised learning model. An unsupervised model differs from a supervised learning model in that it is not guided bu a known outcome or target vector. The algorithm tries to learn the underlying structure of the data. I begin coding the unsupervised model example in the same way as the supervised learning models. I load the dataset, extract the animal names from folders, and process the images by converting them all to 2-channel images. For this example, I extract HOG features from each image and then use this for the features matrix for clustering. I then train and fit the data using K-means clustering and predict method assigns each image to a cluster. PCA is then used to reduce the dimensionality of the HOG features to 2D for visualization purposes. Using the Matplotlib library, I used a scatter plot to visualize the clusters in a 2D space using the PCA components. This example combines key techniques: HOG, PCA, and K-means, to extract meanignful features from the images, reduce the dimensionality for visualization, and cluster the images into groups. By using these three techniques it provided an effective way to handle and analyze the data. Figures 37-41 show the code and the output of the data.

Figure 37: Loading and converting data

Figure 38: Extracting HOG features and defining features matrix

Figure 39: Training a K-means cluster model

Figure 40: Training a PCA model and visualizing the models

Figure 41: K-means clusters with PCA components

# 6. Conclusion

This report explored the practical applications of machine learning focusing specifically on decision tree classifier and random forest classifier algorithms. Guided by Python Data Science Handbook (Vanderplas, 2016), I provided examples that demonstrated foundational machine learning techniques such as model validation, feature engineering, visualizing results and more. This report covered supervised and unsupervised learning algorithms, and attempted to highlight the versatility and power of machine learning by showcasing practical examples of classification, regression, clustering, and dimensionality reduction.

# References

1. Corrado, A. (n.d.). Animals10. Kaggle. Retrieved June 20, 2024, from https://www.kaggle.com/datasets/alessiocorrado99/animals10
2. Terminus7. (2024). Pokemon Challenge dataset. Kaggle. Retrieved June 14, 2024, from https://www.kaggle.com/datasets/terminus7/pokemon-challenge?select=pokemon.csv
3. VanderPlas, J. (*2016*).  *Python Data Science Handbook*. O’Reilly Media. Retrieved from https://jakevdp.github.io/PythonDataScienceHandbook/index.html