Project Report

ZOO ANIMAL CLASSIFICATION

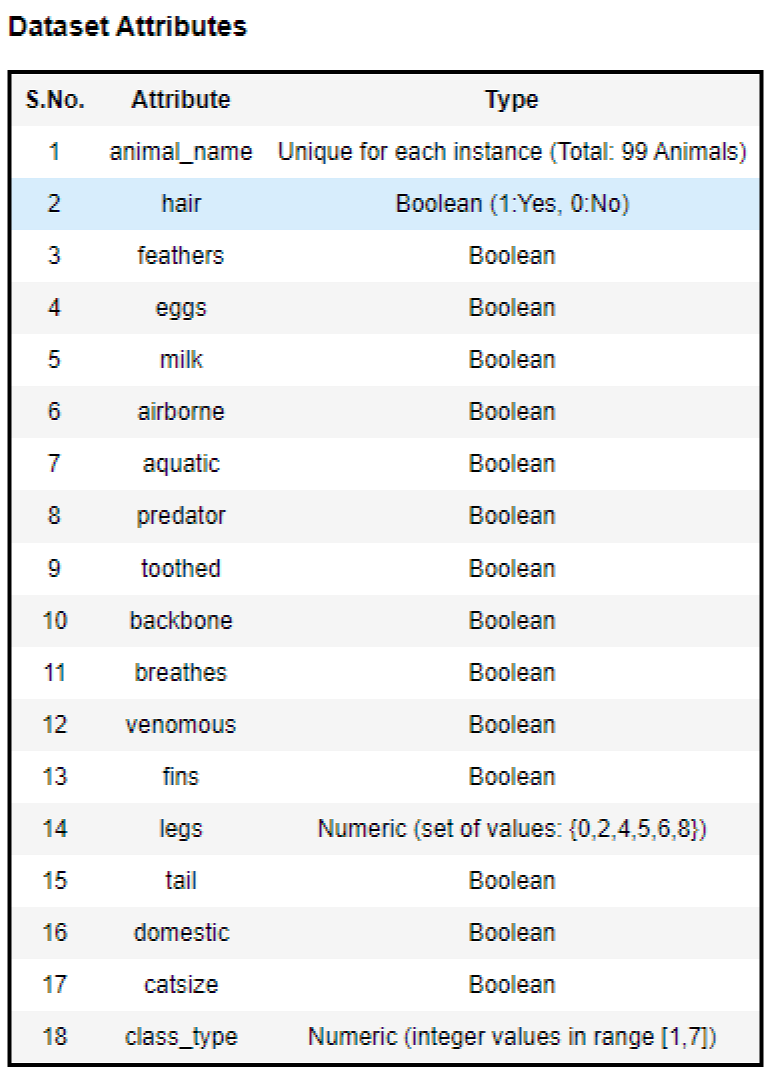
(Machine Learning)

INDEX

INTRODUCTION:

A zoo (short for zoological garden; also called an animal park) is a facility in which animals are housed within enclosures, cared for, displayed to the public, and in some cases bred for conservation purposes.

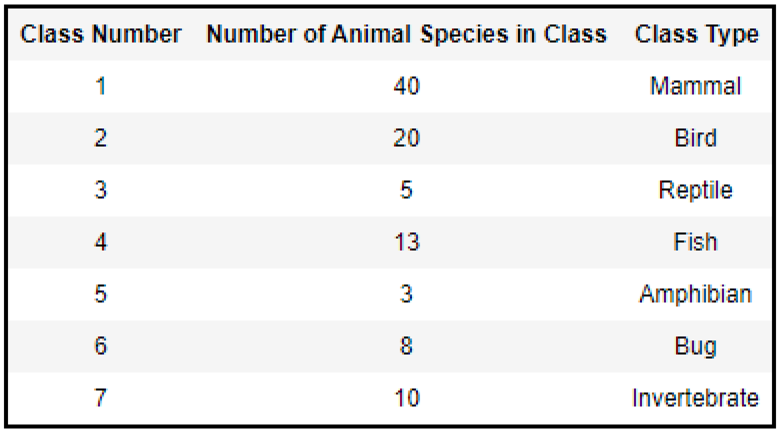
In the Zoo Dataset, there are 16 Variables that describe the features of the Animals.



Dataset Attributes

S.No.	Attribute	Type
1	animal_name	Unique for each instance (Total: 99 Animals)
2	hair	Boolean (1:Yes, 0:No)
3	feathers	Boolean
4	eggs	Boolean
5	milk	Boolean
6	airborne	Boolean
7	aquatic	Boolean
8	predator	Boolean
9	toothed	Boolean
10	backbone	Boolean
11	breathes	Boolean
12	venomous	Boolean
13	fins	Boolean
14	legs	Numeric (set of values: {0,2,4,5,6,8})
15	tail	Boolean
16	domestic	Boolean
17	catsize	Boolean
18	class_type	Numeric (integer values in range [1,7])

Target (class_type)



Class Number	Number of Animal Species in Class	Class Type
1	40	Mammal
2	20	Bird
3	5	Reptile
4	13	Fish
5	3	Amphibian
6	8	Bug
7	10	Invertebrate

The target is class_type. There are 7 class types.
They are Mammal, Bird, Reptile, Fish, Amphibian, Bug and Invertebrate.

Machine Learning: Machine learning (ML) is a type of artificial intelligence (AI) that allows software applications to become more accurate at predicting outcomes without being explicitly programmed to do so. Machine learning algorithms use historical data as input to predict new output values.

Why is machine learning important?
Machine learning is important because it gives enterprises a view of trends in customer behavior and business operational patterns, as well as supports the development of new products. Many of today's leading companies, such as Facebook, Google and Uber, make machine learning a central part of their operations. Machine learning has become a significant competitive differentiator for many companies.

What are the different types of machine learning?
Classical machine learning is often categorized by how an algorithm learns to become more accurate in its predictions. There are four basic approaches: supervised learning, unsupervised learning, semi-supervised learning and reinforcement learning. The type of algorithm data scientists choose to use depends on what type of data they want to predict.

- **Supervised learning:** In this type of machine learning, data scientists supply algorithms with labeled training data and define the variables they want the algorithm to assess for correlations. Both the input and the output of the algorithm is specified.
- **Unsupervised learning:** This type of machine learning involves algorithms that train on unlabeled data. The algorithm scans through datasets looking for any meaningful

connection. The data that algorithms train on as well as the predictions or recommendations they output are predetermined.

- **Semi-supervised learning:** This approach to machine learning involves a mix of the two preceding types. Data scientists may feed an algorithm mostly labeled training data, but the model is free to explore the data on its own and develop its own understanding of the data set.
- **Reinforcement learning:** Data scientists typically use reinforcement learning to teach a machine to complete a multi-step process for which there are clearly defined rules. Data scientists program an algorithm to complete a task and give it positive or negative cues as it works out how to complete a task. But for the most part, the algorithm decides on its own what steps to take along the way.

How does supervised machine learning work?
Supervised machine learning requires the data scientist to train the algorithm with both labeled inputs and desired outputs. Supervised learning algorithms are good for the following tasks:

- **Binary classification:** Dividing data into two categories.
- **Multi-class classification:** Choosing between more than two types of answers
- **Regression modeling:** Predicting continuous values.
- **Ensembling:** Combining the predictions of multiple machine learning models to produce an accurate prediction.

How does unsupervised machine learning work?
Unsupervised machine learning algorithms do not require data to be labeled. They shift through unlabeled data to look for patterns that can be used to group data points into subsets. Most types of deep learning, including neural networks, are unsupervised algorithms. Unsupervised learning algorithms are good for the following tasks:

- **Clustering:** Splitting the dataset into groups based on similarity.
- **Anomaly detection:** Identifying unusual data points in a data set.
- **Association mining:** Identifying sets of items in a data set that frequently occur together.
- **Dimensionality reduction:** Reducing the number of variables in a data set.

How does semi-supervised learning work?
Semi-supervised learning works by data scientists feeding a small amount of labeled training data to an algorithm. From this, the algorithm learns the dimensions of the data set, which it can then apply to new, unlabeled data. The performance of algorithms typically improves when they train on labeled data sets. But labeling data can be time consuming and expensive. Semi-supervised learning strikes a middle ground between the performance of supervised learning

and the efficiency of unsupervised learning. Some areas where semi-supervised learning is used include:

- **Machine translation:** Teaching algorithms to translate language based on less than a full dictionary of words.
- **Fraud detection:** Identifying cases of fraud when you only have a few positive examples.
- **Labelling data:** Algorithms trained on small data sets can learn to apply data labels to larger sets automatically.

How does reinforcement learning work?

Reinforcement learning works by programming an algorithm with a distinct goal and a prescribed set of rules for accomplishing that goal. Data scientists also program the algorithm to seek positive rewards -- which it receives when it performs an action that is beneficial toward the ultimate goal -- and avoid punishments -- which it receives when it performs an action that gets it farther away from its ultimate goal. Reinforcement learning is often used in areas such as:

- Robotics: Robots can learn to perform tasks the physical world using this technique
- Video gameplay: Reinforcement learning has been used to teach bots to play a number of video games.
- Resource management: Given finite resources and a defined goal, reinforcement learning can help enterprises plan out how to allocate resources.

Who's using machine learning and what's it used for?

Today, machine learning is used in a wide range of applications. Perhaps one of the most well-known examples of machine learning in action is the recommendation engine that powers Facebook's news feed.

Facebook uses machine learning to personalize how each member's feed is delivered. If a member frequently stops to read a particular group's posts, the recommendation engine will start to show more of that group's activity earlier in the feed.

Behind the scenes, the engine is attempting to reinforce known patterns in the member's online behavior. Should the member change patterns and fail to read posts from that group in the coming weeks, the news feed will adjust accordingly.

In addition to recommendation engines, other uses for machine learning include the following:

- Customer relationship management. CRM software can use machine learning models to analyze email and prompt sales team members to respond to the most important messages first. More advanced systems can even recommend potentially effective responses.

- Business intelligence. BI and analytics vendors use machine learning in their software to identify potentially important data points, patterns of data points and anomalies.

- Human resource information systems. HRIS systems can use machine learning models to filter through applications and identify the best candidates for an open position.

- Self-driving cars. Machine learning algorithms can even make it possible for a semi-autonomous car to recognize a partially visible object and alert the driver.

- Virtual assistants. Smart assistants typically combine supervised and unsupervised machine learning models to interpret natural speech and supply context.

What are the advantages and disadvantages of machine learning?
Machine learning has seen use cases ranging from predicting customer behavior to forming the operating system for self-driving cars.

When it comes to advantages, machine learning can help enterprises understand their customers at a deeper level. By collecting customer data and correlating it with behaviors over time, machine learning algorithms can learn associations and help teams tailor product development and marketing initiatives to customer demand.

Some companies use machine learning as a primary driver in their business models. Uber, for example, uses algorithms to match drivers with riders. Google uses machine learning to surface the ride advertisements in searches.

But machine learning comes with disadvantages. First and foremost, it can be expensive. Machine learning projects are typically driven by data scientists, who command high salaries. These projects also require software infrastructure that can be expensive.

There is also the problem of machine learning bias. Algorithms trained on data sets that exclude certain populations or contain errors can lead to inaccurate models of the world that, at best, fail and, at worst, are discriminatory. When an enterprise bases core business processes on biased models it can run into regulatory and reputational harm.

How to choose the right machine learning model

The process of choosing the right machine learning model to solve a problem can be time consuming if not approached strategically.

Step 1: Align the problem with potential data inputs that should be considered for the solution. This step requires help from data scientists and experts who have a deep understanding of the problem.

Step 2: Collect data, format it and label the data if necessary. This step is typically led by data scientists, with help from data wranglers.

Step 3: Choose which algorithm(s) to use and test to see how well they perform. This step is usually carried out by data scientists.

Step 4: Continue to fine tune outputs until they reach an acceptable level of accuracy. This step is usually carried out by data scientists with feedback from experts who have a deep understanding of the problem.

What is the future of machine learning?

While machine learning algorithms have been around for decades, they've attained new popularity as artificial intelligence has grown in prominence. Deep learning models, in particular, power today's most advanced AI applications.

Machine learning platforms are among enterprise technology's most competitive realms, with most major vendors, including Amazon, Google, Microsoft, IBM and others, racing to sign customers up for platform services that cover the spectrum of machine learning activities, including data collection, data preparation, data classification, model building, training and application deployment.

Continued research into deep learning and AI is increasingly focused on developing more general applications. Today's AI models require extensive training in order to produce an algorithm that is highly optimized to perform one task. But some researchers are exploring ways to make models more flexible and are seeking techniques that allow a machine to apply context learned from one task to future, different tasks.

CODE:

Zoo Animal Classification

PROBLEM STATEMENT : Predict the Classification of the Animals, Based upon the Variables.

DATASET REFERENCE : https://www.kaggle.com/datasets/uciml/zoo-animal-classification

DESCRIPTION : There are 16 Variables that describes the features of the Animals.

Dataset Attributes

Target (class_type)

DATA EXPLORATION

Import the Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

Extract the Data

```
ZooDataFrame=pd.read_csv('zoo.csv')
ZooDataFrame
```

```
   animal_name  hair  feathers  eggs  milk  airborne  aquatic  predator  \
0     aardvark     1         0     0     1         0        0         1   
1     antelope     1         0     0     1         0        0         0   
2         bass     0         0     1     0         0        1         1   
3         bear     1         0     0     1         0        0         1   
4         boar     1         0     0     1         0        0         1   
..         ...   ...       ...   ...   ...       ...      ...       ...   
95     wallaby     1         0     0     1         0        0         0   
96        wasp     1         0     1     0         1        0         0   
97        wolf     1         0     0     1         0        0         1   
98        worm     0         0     1     0         0        0         0   
99        wren     0         1     1     0         1        0         0   

    toothed  backbone  breathes  venomous  fins  legs  tail  domestic  \
0         1         1         1         0     0     4     0         0   
```

```
1         1         1         1         0     0     4     1         0
2         1         1         0         0     1     0     1         0
3         1         1         1         0     0     4     0         0
4         1         1         1         0     0     4     1         0
..      ...       ...       ...       ...   ...   ...   ...       ...
95        1         1         1         0     0     2     1         0
96        0         0         1         1     0     6     0         0
97        1         1         1         0     0     4     1         0
98        0         0         1         0     0     0     0         0
99        0         1         1         0     0     2     1         0

    catsize  class_type
0         1           1
1         1           1
2         0           4
3         1           1
4         1           1
..      ...         ...
95        1           1
96        0           6
97        1           1
98        0           7
99        0           2

[100 rows x 18 columns]
```

DataFrame Information

```
ZooDataFrame.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 18 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   animal_name  100 non-null    object
 1   hair         100 non-null    int64
 2   feathers     100 non-null    int64
 3   eggs         100 non-null    int64
 4   milk         100 non-null    int64
 5   airborne     100 non-null    int64
 6   aquatic      100 non-null    int64
 7   predator     100 non-null    int64
 8   toothed      100 non-null    int64
 9   backbone     100 non-null    int64
 10  breathes     100 non-null    int64
 11  venomous     100 non-null    int64
 12  fins         100 non-null    int64
 13  legs         100 non-null    int64
 14  tail         100 non-null    int64
 15  domestic     100 non-null    int64
 16  catsize      100 non-null    int64
 17  class_type   100 non-null    int64
dtypes: int64(17), object(1)
memory usage: 14.2+ KB
```

DataFrame Shape, Columns, DataTypes

```
print(ZooDataFrame.shape)
display(ZooDataFrame.columns)
print(ZooDataFrame.dtypes)
```

```
(100, 18)
```

```
Index(['animal_name', 'hair', 'feathers', 'eggs', 'milk', 'airborne',
       'aquatic', 'predator', 'toothed', 'backbone', 'breathes', 'venomous',
       'fins', 'legs', 'tail', 'domestic', 'catsize', 'class_type'],
      dtype='object')
```

```
animal_name    object
hair            int64
feathers        int64
eggs            int64
milk            int64
airborne        int64
aquatic         int64
predator        int64
toothed         int64
backbone        int64
breathes        int64
venomous        int64
fins            int64
legs            int64
tail            int64
domestic        int64
catsize         int64
class_type      int64
dtype: object
```

Data Cleansing

Check for Null values

```
ZooDataFrame.isnull().sum()
```

```
animal_name    0
hair           0
feathers       0
eggs           0
milk           0
airborne       0
aquatic        0
predator       0
toothed        0
```

```
backbone       0
breathes       0
venomous       0
fins           0
legs           0
tail           0
domestic       0
catsize        0
class_type     0
dtype: int64
```

Check for Duplicates

```
ZooDataFrame['animal_name'].duplicated().sum()

1
```

Remove the Duplicates

```
ZooDataFrame.drop_duplicates(subset='animal_name',inplace=True)
ZooDataFrame['animal_name'].duplicated().sum()

0
```

Data Wrangling

Number of Animals that belong to different Class types

```
Class_types= pd.DataFrame(ZooDataFrame.groupby(['class_type'])
['animal_name'].nunique())
Class_types.columns=['Number of Animals ']
Class_types

            Number of Animals
class_type
1                           40
2                           20
3                            5
4                           13
5                            3
6                            8
7                           10
```

Zoo Domestic Animals

```
Domestic_Animals=ZooDataFrame[ZooDataFrame['domestic']==1]
Domestic_Animals['animal_name'].unique()

array(['calf', 'carp', 'cavy', 'chicken', 'dove', 'goat', 'hamster',
       'honeybee', 'parakeet', 'pony', 'pussycat', 'reindeer'],
      dtype=object)
```

Total Animals in the Dataset

```
print('Total Animals in the Zoo Dataset: 
',ZooDataFrame['animal_name'].nunique())
display(ZooDataFrame['animal_name'].unique())
```

```
Total Animals in the Zoo Dataset:  99

array(['aardvark', 'antelope', 'bass', 'bear', 'boar', 'buffalo', 'calf',
       'carp', 'catfish', 'cavy', 'cheetah', 'chicken', 'chub', 'clam',
       'crab', 'crayfish', 'crow', 'deer', 'dogfish', 'dolphin', 'dove',
       'duck', 'elephant', 'flamingo', 'flea', 'frog', 'fruitbat',
       'giraffe', 'gnat', 'goat', 'gorilla', 'gull', 'haddock', 'hamster',
       'hare', 'hawk', 'herring', 'honeybee', 'housefly', 'kiwi',
       'ladybird', 'lark', 'leopard', 'lion', 'lobster', 'lynx', 'mink',
       'mole', 'mongoose', 'moth', 'newt', 'octopus', 'opossum', 'oryx',
       'ostrich', 'parakeet', 'penguin', 'pheasant', 'pike', 'piranha',
       'pitviper', 'platypus', 'polecat', 'pony', 'porpoise', 'puma',
       'pussycat', 'raccoon', 'reindeer', 'rhea', 'scorpion', 'seahorse',
       'seal', 'sealion', 'seasnake', 'seawasp', 'skimmer', 'skua',
       'slowworm', 'slug', 'sole', 'sparrow', 'squirrel', 'starfish',
       'stingray', 'swan', 'termite', 'toad', 'tortoise', 'tuatara',
       'tuna', 'vampire', 'vole', 'vulture', 'wallaby', 'wasp', 'wolf',
       'worm', 'wren'], dtype=object)
```

Aquatic Animals

```
Aquatic_Animals=ZooDataFrame[ZooDataFrame['aquatic']==1]
print('Aquatic Animals: ',Aquatic_Animals['animal_name'].nunique())
display(Aquatic_Animals['animal_name'].unique())
```

```
Aquatic Animals:  35

array(['bass', 'carp', 'catfish', 'chub', 'crab', 'crayfish', 'dogfish',
       'dolphin', 'duck', 'frog', 'gull', 'haddock', 'herring', 'lobster',
       'mink', 'newt', 'octopus', 'penguin', 'pike', 'piranha',
       'platypus', 'porpoise', 'seahorse', 'seal', 'sealion', 'seasnake',
       'seawasp', 'skimmer', 'skua', 'sole', 'starfish', 'stingray',
       'swan', 'toad', 'tuna'], dtype=object)
```

Predators

```
Predators=ZooDataFrame[ZooDataFrame['predator']==1]
print('Predators: ', Predators['animal_name'].nunique())
display(Predators['animal_name'].unique())
```

```
Predators:  54

array(['aardvark', 'bass', 'bear', 'boar', 'catfish', 'cheetah', 'chub',
       'clam', 'crab', 'crayfish', 'crow', 'dogfish', 'dolphin', 'frog',
       'gull', 'hawk', 'herring', 'kiwi', 'ladybird', 'leopard', 'lion',
       'lobster', 'lynx', 'mink', 'mole', 'mongoose', 'newt', 'octopus',
       'opossum', 'penguin', 'pike', 'piranha', 'pitviper', 'platypus',
       'polecat', 'porpoise', 'puma', 'pussycat', 'raccoon', 'rhea',
       'scorpion', 'seal', 'sealion', 'seasnake', 'seawasp', 'skimmer',
       'skua', 'slowworm', 'starfish', 'stingray', 'tuatara', 'tuna',
       'vulture', 'wolf'], dtype=object)
```

Descriptive Statistics

Descriptive Statistics Information of the Numerical columns

```
ZooDataFrame.describe()
```

```
            hair   feathers       eggs       milk   airborne    aquatic  \
count  99.000000  99.000000  99.000000  99.000000  99.000000  99.000000   
mean    0.424242   0.202020   0.585859   0.404040   0.242424   0.353535   
std     0.496743   0.403551   0.495080   0.493203   0.430730   0.480500   
min     0.000000   0.000000   0.000000   0.000000   0.000000   0.000000   
25%     0.000000   0.000000   0.000000   0.000000   0.000000   0.000000   
50%     0.000000   0.000000   1.000000   0.000000   0.000000   0.000000   
75%     1.000000   0.000000   1.000000   1.000000   0.000000   1.000000   
max     1.000000   1.000000   1.000000   1.000000   1.000000   1.000000   

        predator    toothed   backbone   breathes   venomous       fins  \
count  99.000000  99.000000  99.000000  99.000000  99.000000  99.000000   
mean    0.545455   0.595960   0.818182   0.787879   0.080808   0.171717   
std     0.500464   0.493203   0.387657   0.410891   0.273927   0.379054   
min     0.000000   0.000000   0.000000   0.000000   0.000000   0.000000   
25%     0.000000   0.000000   1.000000   1.000000   0.000000   0.000000   
50%     1.000000   1.000000   1.000000   1.000000   0.000000   0.000000   
75%     1.000000   1.000000   1.000000   1.000000   0.000000   0.000000   
max     1.000000   1.000000   1.000000   1.000000   1.000000   1.000000   

            legs       tail   domestic    catsize  class_type  
count  99.000000  99.000000  99.000000  99.000000   99.000000  
mean    2.838384   0.757576   0.121212   0.434343    2.828283  
std     2.048927   0.430730   0.328035   0.498193    2.104613  
min     0.000000   0.000000   0.000000   0.000000    1.000000  
25%     2.000000   1.000000   0.000000   0.000000    1.000000  
50%     4.000000   1.000000   0.000000   0.000000    2.000000  
75%     4.000000   1.000000   0.000000   1.000000    4.000000  
max     8.000000   1.000000   1.000000   1.000000    7.000000  
```

Correlation

```
ZooDataFrame[['hair','domestic']].corr()
```

```
              hair  domestic
hair      1.000000  0.182171
domestic  0.182171  1.000000
```

Covariance

```
ZooDataFrame[['class_type','domestic']].cov()
```

```
            class_type  domestic
class_type    4.429396 -0.111626
domestic     -0.111626  0.107607
```

Mean

```
ZooDataFrame.mean()
```

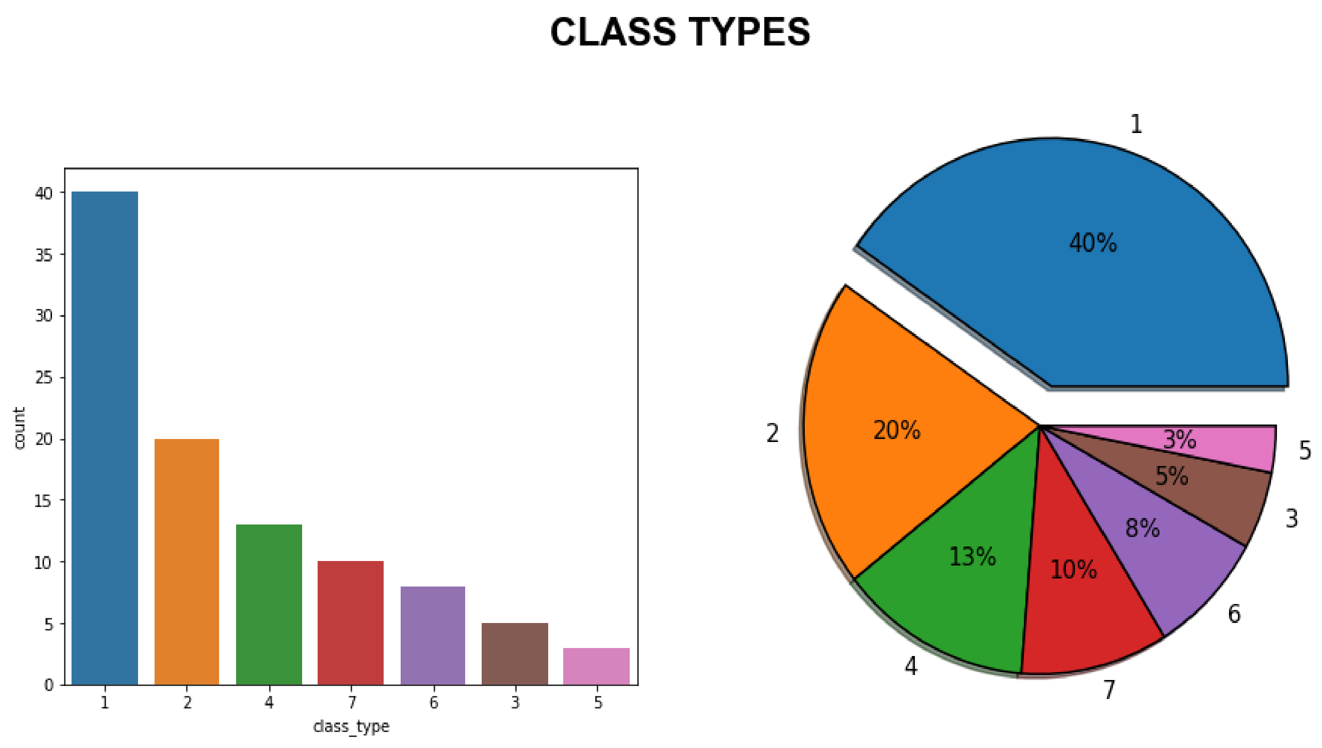
```
hair          0.424242
feathers      0.202020
eggs          0.585859
milk          0.404040
airborne      0.242424
aquatic       0.353535
predator      0.545455
toothed       0.595960
backbone      0.818182
breathes      0.787879
venomous      0.080808
fins          0.171717
legs          2.838384
tail          0.757576
domestic      0.121212
catsize       0.434343
class_type    2.828283
dtype: float64
```

Data Visualization

Number of Animals in Each Class

```
#Creating Seaborn Countplot
fig, ax=plt.subplots(1, 2, figsize=(15,6))
Countplot= sns.countplot(x='class_type', data=ZooDataFrame, ax=ax[0],
                         order =
ZooDataFrame['class_type'].value_counts().index)

#Creating Matplotlib PieChart
explode = [0.2,0,0,0,0,0,0]
Piechart=ZooDataFrame['class_type'].value_counts(sort=True).plot.pie(autopct=
"%.0f%%",ax=ax[1],

shadow=True,explode=explode,radius=1.2,
                                                            textprops =
{'color': 'black','fontsize':15},
                                                            wedgeprops =
{'linewidth':1.5,'edgecolor' : "black"})
plt.ylabel(' ')
```

```
# Title
plt.suptitle(" CLASS TYPES", y=1.1, fontsize=25,
fontweight='bold',family='Arial')

# Show plot
plt.show()
```

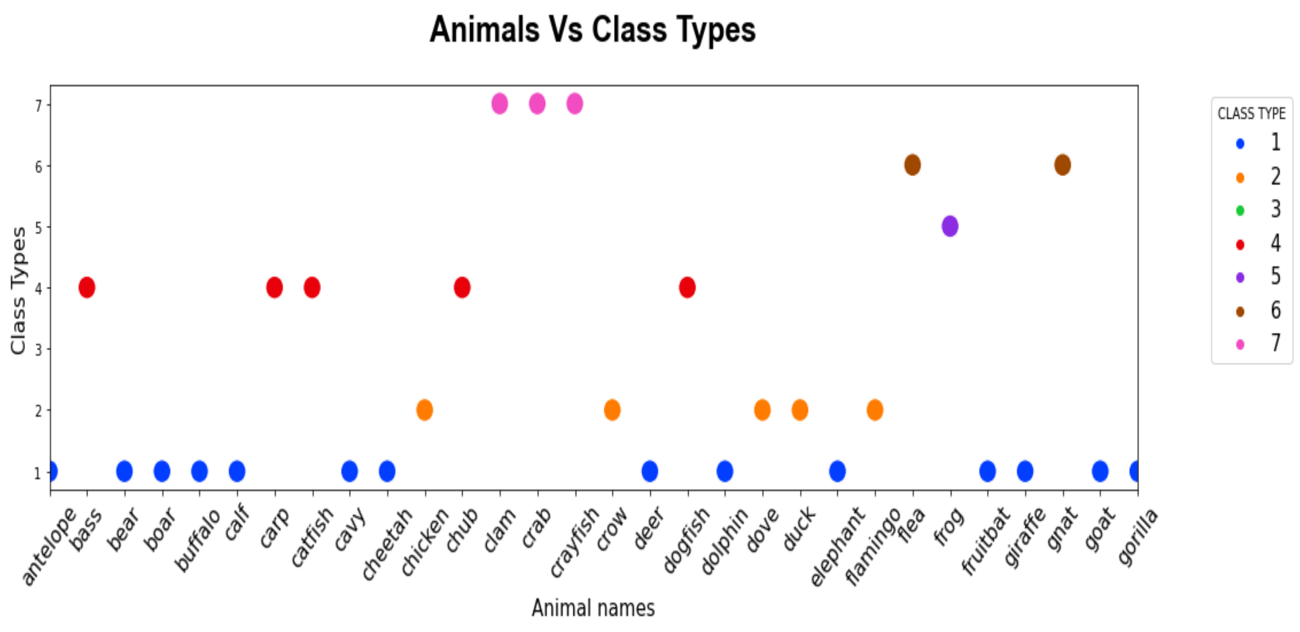


Observation: Class Type 1- Mammals is in Majority while Class Type 5- Amphibians is the Minority

Animals Vs Class Types

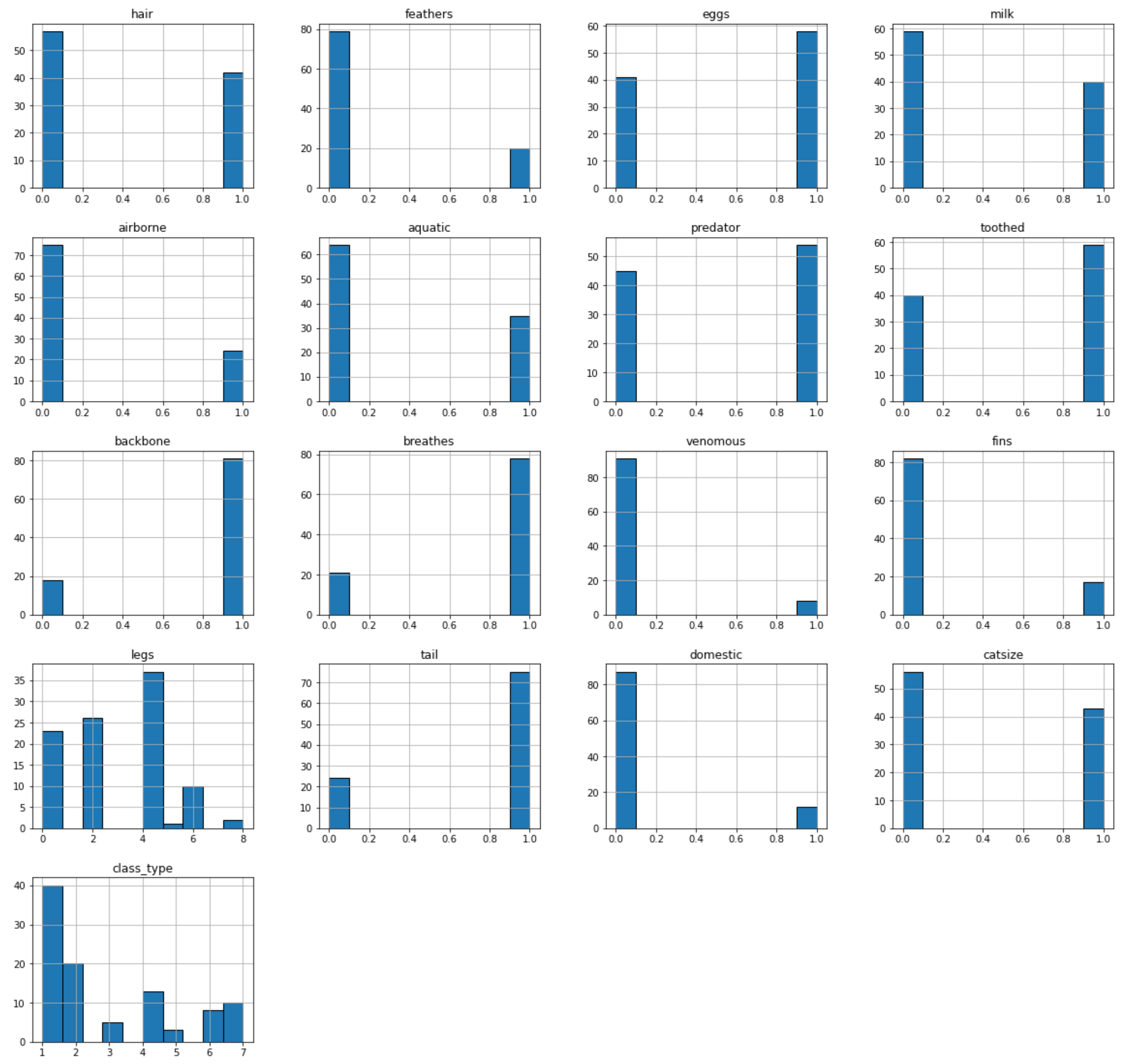
```
# Creating Figure size
plt.figure(figsize=(17,5))

# Creating Scatterplot
sns.scatterplot(x=ZooDataFrame['animal_name'],y=ZooDataFrame['class_type'],
                hue=ZooDataFrame['class_type'], palette='bright',s=250)

# Title
plt.title("Animals Vs Class Types",fontsize=25,
fontweight='bold',family='Arial',pad=30)

# Legend
plt.legend(loc='upper right', title='CLASS TYPE', bbox_to_anchor =(1.15,
1),fontsize=15)
```

```
# X-Axis and Y-Axis Labels
plt.xlabel("Animal names",fontsize=15)
plt.ylabel("Class Types",fontsize=15)

# X-Axis ticks
plt.xticks(rotation=50,fontsize=15)

# X-Axis Limits
plt.xlim(1,30)

# show plot
plt.show()
```



Visualizing the distibution of the data for every feature

```
#Creating Histogram
ZooDataFrame.hist(edgecolor='black', figsize=(20,20))

# Show plot
plt.show()
```



BUILD THE MODEL

Let us consider x as Features array and y as Target values.

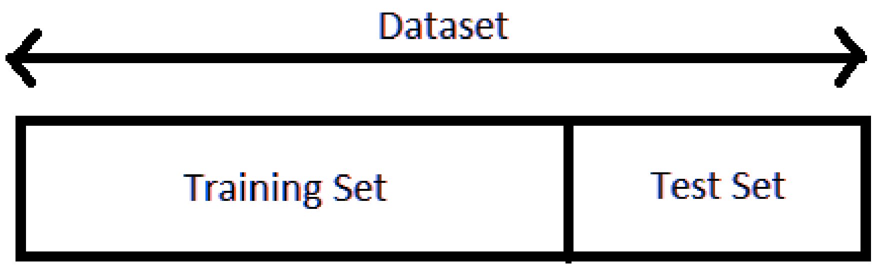
```
x=ZooDataFrame.drop(['animal_name','class_type'], axis=1).values
y=ZooDataFrame['class_type'].values
print(x,y)
```

```
[[1 0 0 ... 0 0 1]
 [1 0 0 ... 1 0 1]
 [0 0 1 ... 1 0 0]
 ...
 [1 0 0 ... 1 0 1]
 [0 0 1 ... 0 0 0]
 [0 1 1 ... 1 0 0]] [1 1 4 1 1 1 1 4 4 1 1 2 4 7 7 7 2 1 4 1 2 2 1 2 6 5 1 1
6 1 1 2 4 1 1 2 4
 6 6 2 6 2 1 1 7 1 1 1 1 6 5 7 1 1 2 2 2 2 4 4 3 1 1 1 1 1 1 1 1 2 7 4 1 1
 3 7 2 2 3 7 4 2 1 7 4 2 6 5 3 3 4 1 1 2 1 6 1 7 2]
```

Train Test Split Method :

1. A technique for evaluating the performance of a machine learning algorithm.
2. The procedure involves taking a dataset and dividing it into two subsets.
3. The first subset is used to fit the model and is referred to as the training dataset. The second subset is not used to train the model; instead, the input element of the dataset is provided to the model, then predictions are made and compared to the expected values. This second dataset is referred to as the test dataset.

Train Dataset: Used to fit the machine learning model. Test Dataset: Used to evaluate the fit machine learning model.



Apply the train test split method

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_stat
e=2)
```

Classification Models

Let us consider the following classification models for our Dataset

1. DecisionTreeClassifier(): The decision tree classifier creates the classification model by building a decision tree. Each node in the tree specifies a test on an attribute, each branch descending from that node corresponds to one of the possible values for that attribute.
2. RandomForestClassifier(): A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

3. SVC(kernel='linear'): The objective of a Linear SVC (Support Vector Classifier) is to fit to the data that we provide, returning a "best fit" hyperplane that divides, or categorizes our data.

4. GaussianNB(): Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable. Naive Bayes learners and classifiers can be extremely fast compared to more sophisticated methods.

5. GradientBoostingClassifier(): Gradient boosting classifiers are a group of machine learning algorithms that combine many weak learning models together to create a strong predictive model. Decision trees are usually used when doing gradient boosting. Boosting is a general ensemble technique that involves sequentially adding models to the ensemble where subsequent models correct the performance of prior models.

Import the Libraries and create a model list

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import GradientBoostingClassifier

MODEL_LIST=[DecisionTreeClassifier(random_state=2),
            RandomForestClassifier(random_state=2,n_estimators=25),
            SVC(kernel='linear'),
            GaussianNB(),
            GradientBoostingClassifier(n_estimators=30,random_state=2)]
```

Calculate Score and Time for each Model

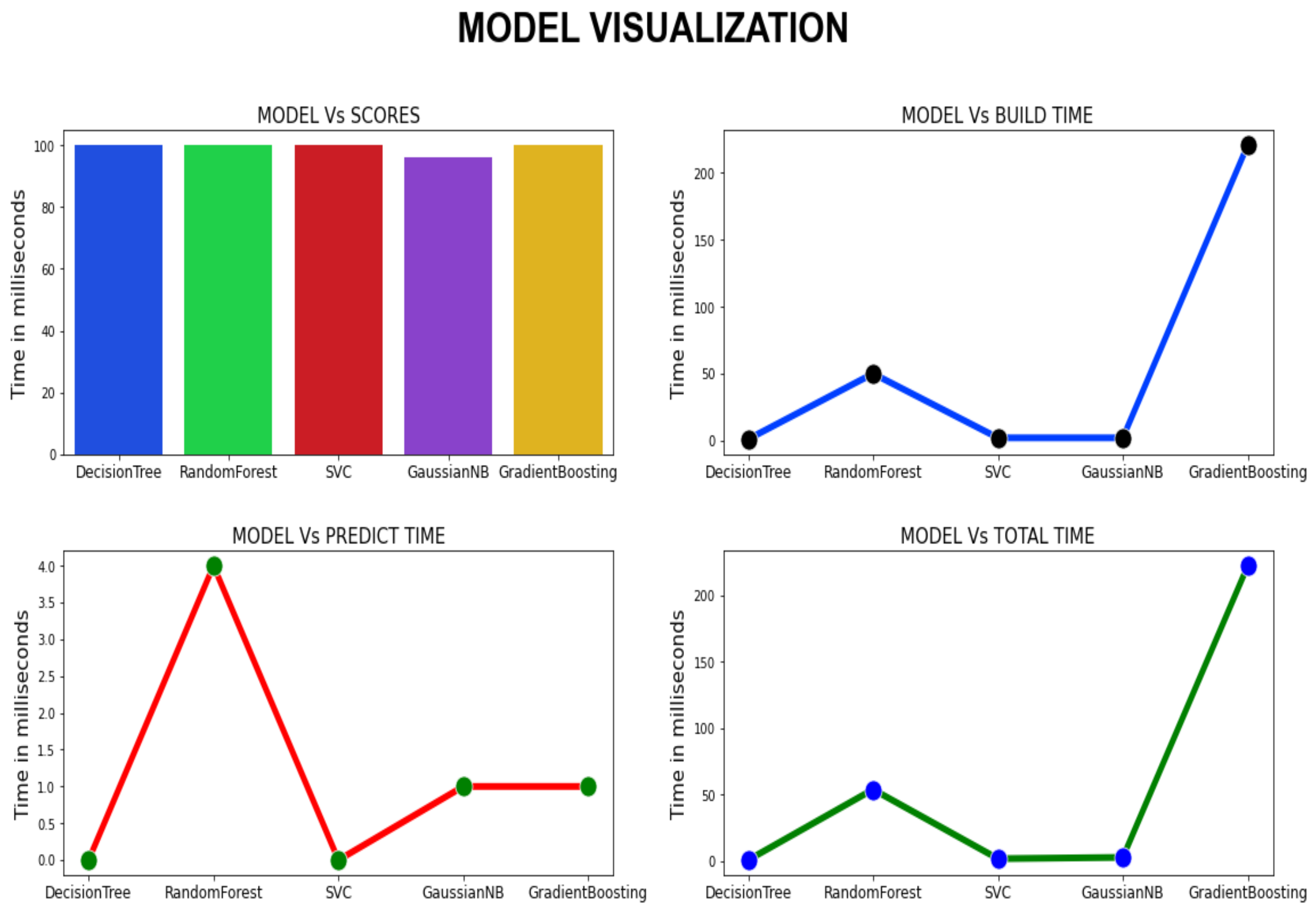
```python
import time

#Created a Dictionary to store the score for each model
Score_List={}

#Created a List to store the time taken to build each model
Time_Build=[]

# Created a List to store the time taken to predict each model
Time_Predict=[]

# Calculate score and time
for model in MODEL_LIST:
    #Calculate Time taken to build the model
    start_build = time.time()
    model.fit(x_train,y_train)
    end_build=time.time()
    Time_Build.append(round((end_build-start_build)*1000,2))
    Score_List[model]=model.score(x_test,y_test)*100

    # Now Calculate Time taken to predict
    start_predict = time.time()
    y_predict=model.predict(x_test)
    end_predict=time.time()
    Time_Predict.append(round((end_predict-start_predict)*1000,2))

#Created a Dictionary for Model names, scores and Time Taken
ScoreTimeDict={'Model_Name': list(map(str, MODEL_LIST)),
'Score':Score_List.values(),

'Build_Time_in_Milliseconds':Time_Build,'Predict_Time_in_Milliseconds':Time_P
redict}

#Created a Data Frame
ScoreTimeDataFrame=pd.DataFrame(ScoreTimeDict, index=range(1,6))

#Converting the Column Datatypes
ScoreTimeDataFrame=ScoreTimeDataFrame.convert_dtypes()

#Calculate the Total Time taken
ScoreTimeDataFrame['Total_Time_Milliseconds']=round(ScoreTimeDataFrame['Build
_Time_in_Milliseconds']+ScoreTimeDataFrame['Predict_Time_in_Milliseconds'],2)

#Print the DataFrame
ScoreTimeDataFrame
```

```
                                          Model_Name  Score  \
1             DecisionTreeClassifier(random_state=2)    100
2  RandomForestClassifier(n_estimators=25, random...    100
3                               SVC(kernel='linear')    100
4                                       GaussianNB()     96
5  GradientBoostingClassifier(n_estimators=30, ra...    100

   Build_Time_in_Milliseconds  Predict_Time_in_Milliseconds  \
1                           1                             0
2                          50                             4
3                           2                             0
4                           2                             1
5                         221                             1

   Total_Time_Milliseconds
1                        1
2                       54
3                        2
4                        3
5                      222
```

Plot a Graph for the scores and time taken by each model

```
#Created a Bargraph that represents the scores and lineplot that represents
Total Time taken by the models

plt.style.use('seaborn-bright')

#Create Subplots with 2 rows and 2 columns
fig, ax= plt.subplots(2,2, figsize=(19,10))

# set the spacing between subplots
plt.subplots_adjust(wspace=0.2, hspace=0.3)

#Title
fig.suptitle(" MODEL VISUALIZATION", y=1, fontsize=30,
fontweight='bold',family='Arial')

#Create Subplots

#Create barplot for MODEL Vs Score
Graph1=sns.barplot(y=ScoreTimeDataFrame['Score'],x=ScoreTimeDataFrame['Model_
Name'],ax=ax[0,0])
ax[0,0].set_title('MODEL Vs SCORES',fontsize=15)

#Create lineplot for MODEL Vs BUILD TIME
Graph2=sns.lineplot(ScoreTimeDataFrame['Model_Name'],ScoreTimeDataFrame['Buil
d_Time_in_Milliseconds'],
                    linewidth=5, ax=ax[0,1],marker="o",mfc="black",ms=15)
ax[0,1].set_title("MODEL Vs BUILD TIME",fontsize=15)

#Create lineplot for MODEL Vs PREDICT TIME
Graph3=sns.lineplot(ScoreTimeDataFrame['Model_Name'],ScoreTimeDataFrame['Pred
ict_Time_in_Milliseconds'],

linewidth=5,ax=ax[1,0],color="red",marker="o",mfc="green",ms=15)
ax[1,0].set_title("MODEL Vs PREDICT TIME",fontsize=15)

#Create lineplot for MODEL Vs TOTAL TIME
Graph4=sns.lineplot(ScoreTimeDataFrame['Model_Name'],ScoreTimeDataFrame['Tota
l_Time_Milliseconds'],

linewidth=5,ax=ax[1,1],color="green",marker="o",mfc="blue",ms=15)
ax[1,1].set_title("MODEL Vs TOTAL TIME",fontsize=15)

# Create a list to edit all the graphs at a time
Graphs=[Graph1,Graph2,Graph3,Graph4]

# For loop- to customize all the graphs at a time
for grph in Graphs:
```

```
grph.set_xticklabels(['DecisionTree','RandomForest','SVC','GaussianNB','Gradi
entBoosting'])
    grph.set(xlabel=None)
    grph.set_ylabel("Time in milliseconds", fontsize=15)
    grph.tick_params(axis='x', labelsize=12)

# Show the plots
plt.show()
```



MODEL SELECTION

From the above table and graphs, it is clear that we have four models with 100% accuracy.

Let us select 'DecisionTreeClassifier()' (Accuracy:100%, Less Time)

```
# Building the model
MODEL=DecisionTreeClassifier(random_state=2)
MODEL.fit(x_train,y_train)

# Evaluating the model
MODEL.score(x_test,y_test)*100

100.0
```

Classification Report

1. Build a text report showing the main classification metrics.
2. It is a performance evaluation metric in machine learning which is used to show the precision, recall, F1 Score, and support score of the trained classification model

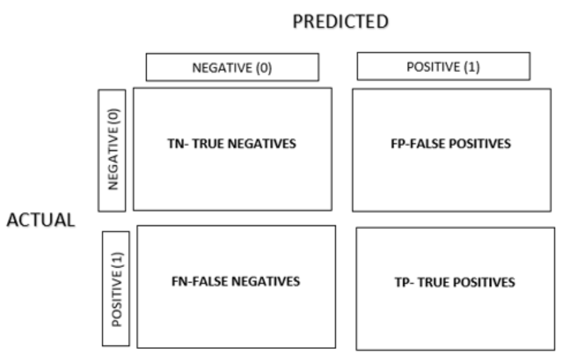
Print the Classification Report

```
from sklearn.metrics import classification_report
y_predict=MODEL.predict(x_test)
performance_report=classification_report(y_test, y_predict)
print(performance_report)
```

```
              precision    recall  f1-score   support

           1       1.00      1.00      1.00         8
           2       1.00      1.00      1.00         8
           3       1.00      1.00      1.00         2
           4       1.00      1.00      1.00         1
           6       1.00      1.00      1.00         2
           7       1.00      1.00      1.00         4

    accuracy                           1.00        25
   macro avg       1.00      1.00      1.00        25
weighted avg       1.00      1.00      1.00        25
```

Confusion Matrix

1. A tabular summary of the number of correct and incorrect predictions made by a classifier.
2. It can be used to evaluate the performance of a classification model through the calculation of performance metrics like accuracy, precision, recall, and F1-score.



$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

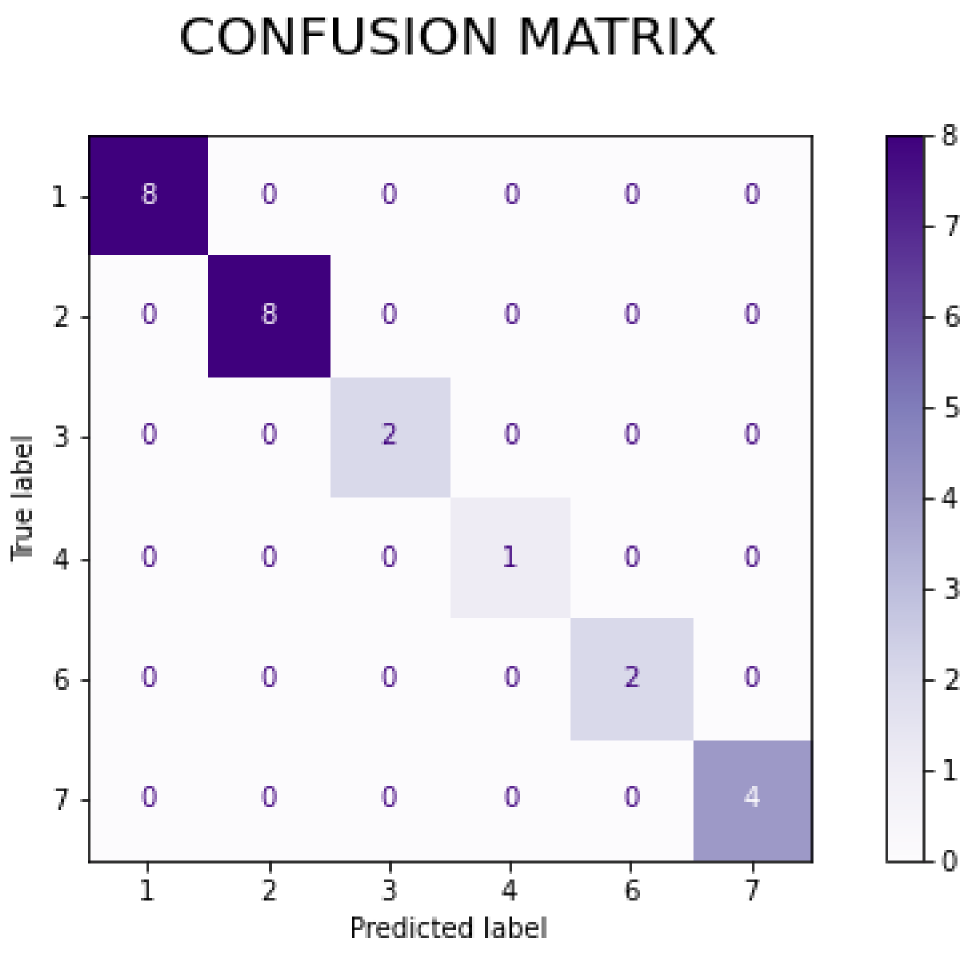
$$F1\text{-}score = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Plot the Confusion Matrix

```
from sklearn.metrics import confusion_matrix, plot_confusion_matrix
print(confusion_matrix(y_test, y_predict),"\n")
fig,ax=plt.subplots(figsize=(10, 5))
plot_confusion_matrix(MODEL,x_test,y_test,cmap="Purples", ax=ax)
```

```
plt.title("CONFUSION MATRIX",fontsize='20', pad=30)
plt.grid(False)
plt.show()
```

```
[[8 0 0 0 0 0]
 [0 8 0 0 0 0]
 [0 0 2 0 0 0]
 [0 0 0 1 0 0]
 [0 0 0 0 2 0]
 [0 0 0 0 0 4]]
```



Actual Vs Predicted Values

```
print("Actual: ",len(y_test))
print("Predicted: ",(y_predict==y_test).sum())
```

```
Actual:  25
Predicted:  25
```

Predicted Values DataFrame

```
predicted=pd.DataFrame({'Actual':y_test,'Predicted':y_predict})
predicted
```

```
    Actual  Predicted
0        2          2
1        1          1
2        2          2
3        6          6
4        2          2
5        2          2
6        4          4
7        1          1
8        6          6
9        7          7
10       2          2
11       1          1
12       3          3
13       7          7
14       1          1
15       2          2
16       1          1
17       1          1
18       1          1
19       3          3
20       2          2
21       2          2
22       7          7
23       7          7
24       1          1
```

Actual and Predicted Values Table

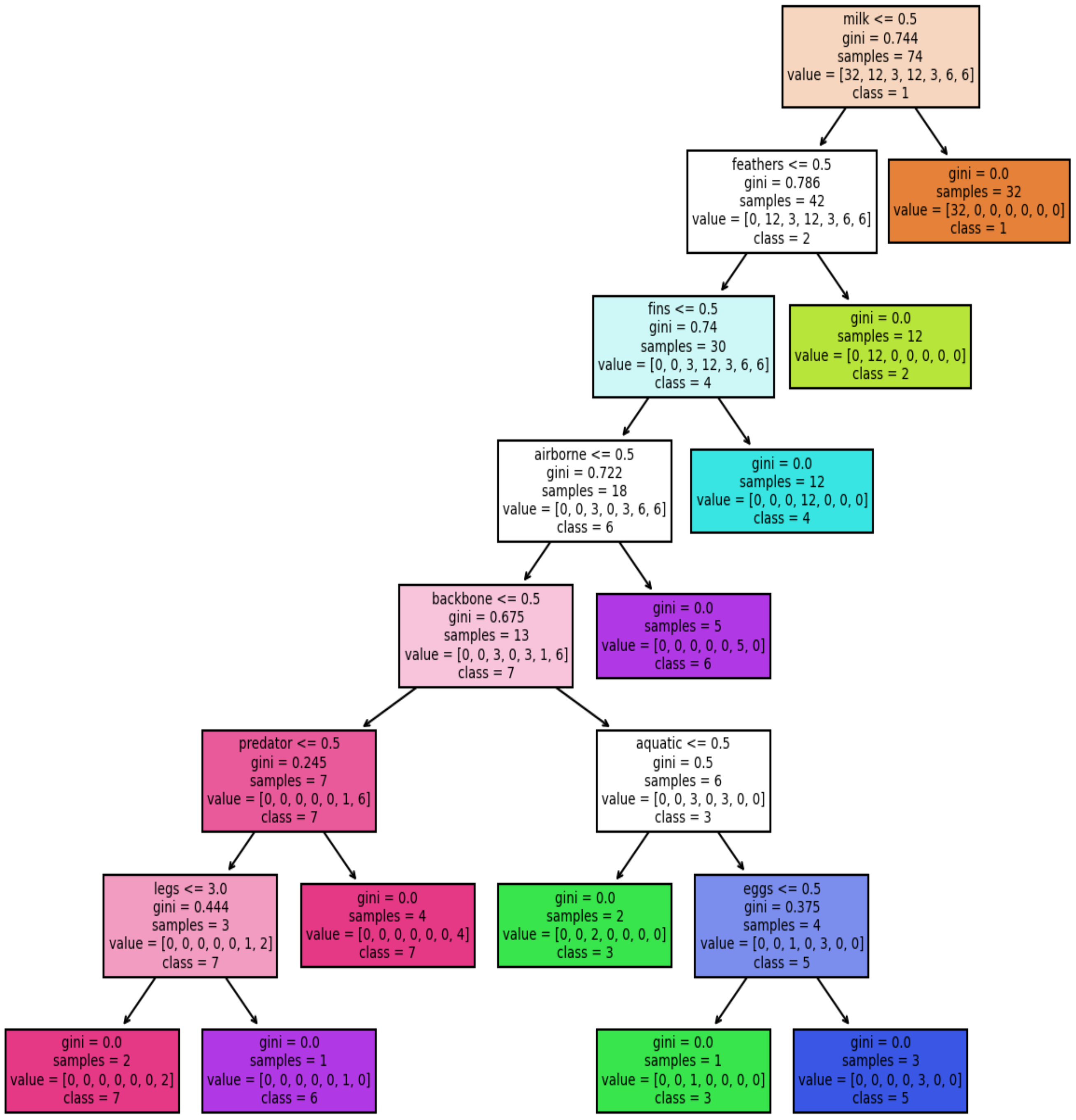
```
Actual_Predicted=pd.crosstab(predicted['Actual'],predicted['Predicted'])
Actual_Predicted
```

```
Predicted  1  2  3  4  6  7
Actual
1          8  0  0  0  0  0
2          0  8  0  0  0  0
3          0  0  2  0  0  0
4          0  0  0  1  0  0
6          0  0  0  0  2  0
7          0  0  0  0  0  4
```

Plot Decision Tree

```
Features=['hair', 'feathers', 'eggs', 'milk', 'airborne',
          'aquatic', 'predator', 'toothed', 'backbone',
          'breathes', 'venomous','fins', 'legs', 'tail',
          'domestic', 'catsize']
Class=['1','2','3','4','5','6','7']
```

```
from sklearn import tree
fig,ax=plt.subplots(nrows=1,ncols=1,figsize=(10,9),dpi=150)
tree.plot_tree(MODEL,feature_names=Features,class_names=Class,filled=True)
plt.show()
```



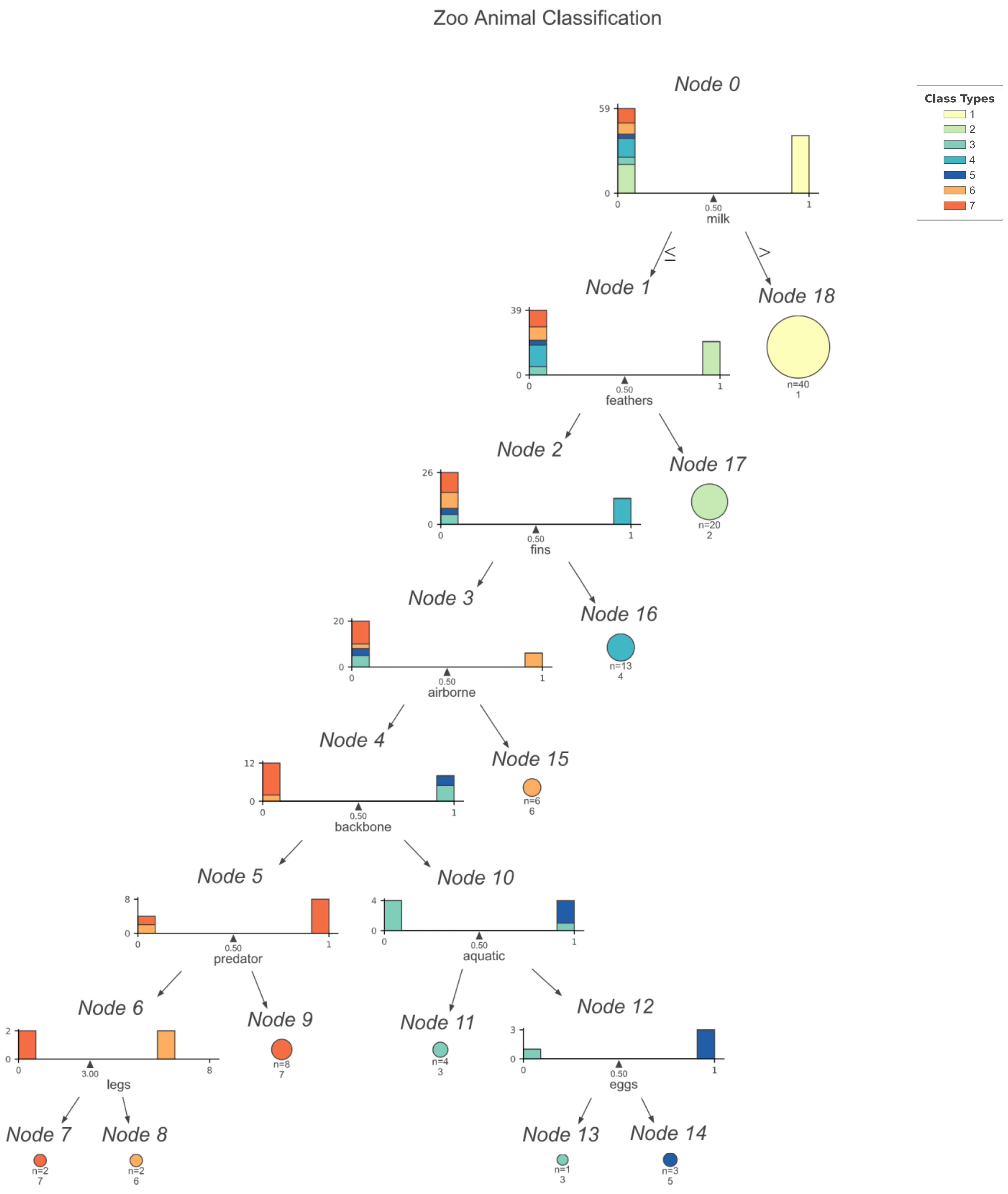
Install dtreeviz

1. Install the latest Windows/Ubuntu Graphviz   https://graphviz.org/download/
2. Type the below command in Anaconda Powershell Prompt   conda install graphviz python-graphviz   pip install dtreeviz
3. Set the Windows Graphiz Path from Jupyter Notebook   import os os.environ["PATH"] += os.pathsep + 'C:/Program Files/Graphviz/bin'

Plot Decision Tree using dtreeviz Package

```
import os
os.environ["PATH"] += os.pathsep + 'C:/Program Files/Graphviz/bin'

from dtreeviz.trees import dtreeviz
DecisionTreeVisualization = dtreeviz(MODEL, x, y,
                                        target_name="Class Types",
                                        feature_names=Features,
                                        class_names=Class,
                                        title="Zoo Animal Classification",
                                        scale=1.22, show_node_labels = True)

DecisionTreeVisualization.view()  # To view in the browser
DecisionTreeVisualization
```

```
<dtreeviz.trees.DTreeViz at 0x1ee2d258190>
```


Zoo Animal Classification
Class Types
1
2
3
4
5
6
7
Node 0
milk
≤
>
Node 1
feathers
Node 18
n=40
1
Node 2
fins
Node 17
n=20
2
Node 3
airborne
Node 16
n=13
4
Node 4
backbone
Node 15
n=6
6
Node 5
predator
Node 10
aquatic
Node 6
legs
Node 9
n=8
7
Node 11
n=4
3
Node 12
eggs
Node 7
n=2
7
Node 8
n=2
6
Node 13
n=1
3
Node 14
n=3
5

SUMMARY:

For the Zoo Animal Classification, we have selected the following classification models.

- DecisionTreeClassifier
- RandomForestClassifier
- SVC(kernel='linear')
- Naive Bayes- GaussianNB()
- GradientBoostingClassifier()

Accuracy Scores for each Model:

MODEL NAME	ACCURACY SCORE
DecisionTreeClassifier	100
RandomForestClassifier	100
SVC	100
Naive Bayes- GaussianNB	96
GradientBoostingClassifier	100

Model Selected: DecisionTreeClassifier(random_state=2)

Advantages and Disadvantages of the DecisionTreeClassifier model:

Advantages:

1. No considerable impact of missing values.
2. Easy to explain to non-technical team members.
3. Easy visualization
4. Automatic Feature selection: Irrelevant features won’t affect decision trees.

Disadvantages:

1. Prone to overfitting. Good performance on the training data, poor generalization to new data.
2. Sensitive to data. If data changes slightly, the outcomes can change to a very large extent.
3. Higher time required to train decision trees of large datasets.

Applications:

Identifying buyers for products, finding strategy that can maximize the profit, finding strategy for cost minimization, which features are most important to attract and retain customers (is it the frequency of shopping, is it the frequent discounts, is it the product mix), fault diagnosis in machines (keep measuring pressure, vibrations and other measures and predict before a fault occurs) and many other applications.

The Best Machine learning models for the Zoo dataset are **DecisionTreeClassifier, RandomForestClassifier, SVC, GradientBoostingClassifier.**

REFERENCES:

https://www.kaggle.com/datasets/uciml/zoo-animal-classification

https://matplotlib.org/

https://seaborn.pydata.org/

https://scikit-learn.org/stable/