Deliverable 2

Feng(Shelley) Xia

1. Problem Statement: Our goal is to classify a set of zoo animals based on their traits and develop a webapp analogous to Akinator through predicting the class the animal belongs to given its traits by the user.

2. Data Preprocessing:

We would use the datasets from the following sources:
https://www.kaggle.com/uciml/zoo-animal-classification?select=zoo.csv
https://www.kaggle.com/agajorte/zoo-animals-extended-dataset?select=zoo2.csv

There are four datasets: zoo1.csv, zoo2.csv, zoo3.csv and class.csv

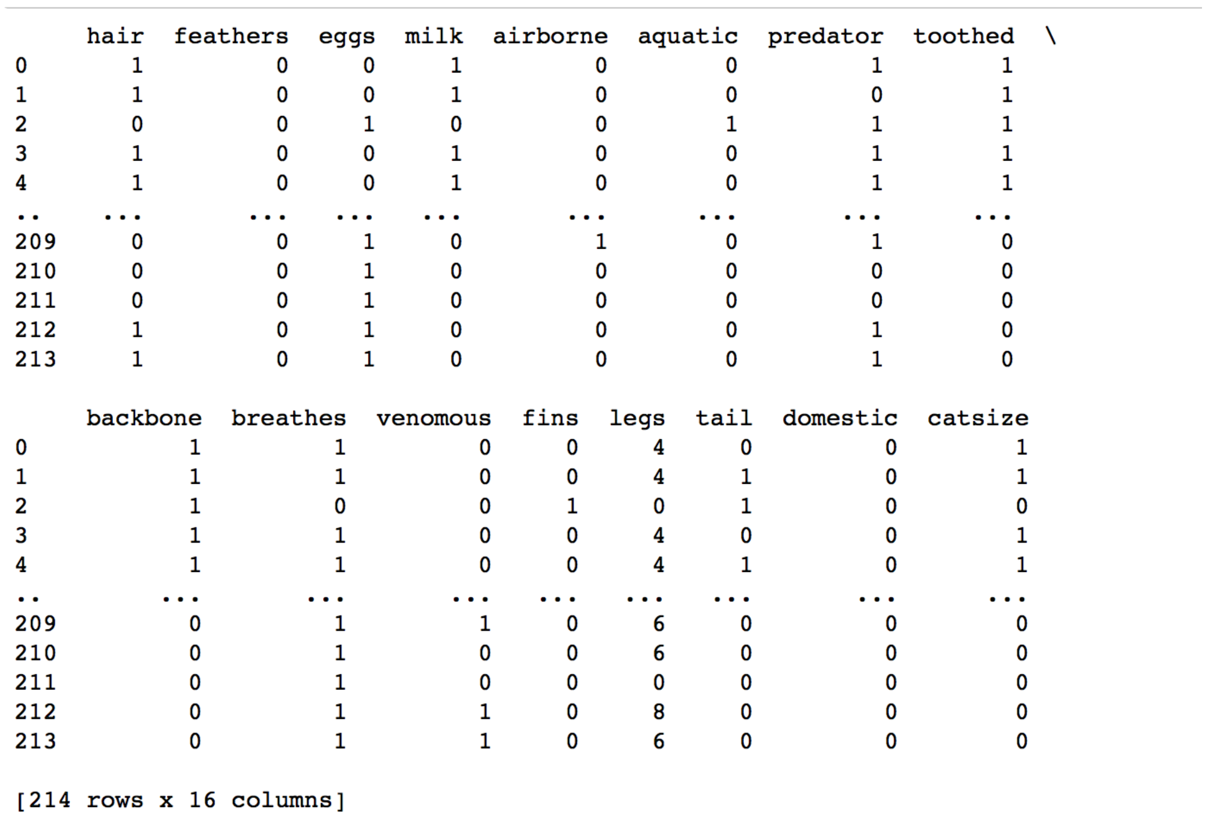
We first manually combine zoo.1csv, zoo2.csv, zoo3.csv together into one csv file zoo.csv.

Zoo.csv is made up of 214 samples of zoo animals, where each sample consists of 17 features and 1 label. There are 7 choices for the label, representing the 7 classes to which the 214 animals belong. The 7 Class types are: Mammal, Bird, Reptile, Fish, Amphibian, Bug and Invertebrate.

Class.csv originally contains only information about classes in zoo1.csv. We would update it using zoo2.csv and zoo3.csv. The updated clss.csv contains guidance about zoo.csv, namely the class number, number of animal species in the class, class type and animals in the class. The class.csv file does not carry imperative information for our interest in classification, so we decide to leave it untouched for now.

We would like to preprocess zoo.csv as follows:

- The original dataset downloaded from the source is highly mature. Apart from the first feature, 'animal_name', which is unique for each animal, the selection of the rest of 16 features is sufficiently concise and representative for our purpose. Hence, we would strip the first column and plan on no further feature selection.
- We would then perform a check on each sample to make sure all the fields of features are complete by filling up missing values, if any.
- In terms of feature scaling, all of the features in the dataset, except for 'legs', which has values in the set {0,2,4,5,6,8}, already take on values either 1 or 0.



```
     hair  feathers  eggs  milk  airborne  aquatic  predator  toothed  \
0       1         0     0     1         0        0         1        1
1       1         0     0     1         0        0         0        1
2       0         0     1     0         0        1         1        1
3       1         0     0     1         0        0         1        1
4       1         0     0     1         0        0         1        1
..    ...       ...   ...   ...       ...      ...       ...      ...
209     0         0     1     0         1        0         1        0
210     0         0     1     0         0        0         0        0
211     0         0     1     0         0        0         0        0
212     1         0     1     0         0        0         1        0
213     1         0     1     0         0        0         1        0

     backbone  breathes  venomous  fins  legs  tail  domestic  catsize
0           1         1         0     0     4     0         0        1
1           1         1         0     0     4     1         0        1
2           1         0         0     1     0     1         0        0
3           1         1         0     0     4     0         0        1
4           1         1         0     0     4     1         0        1
..        ...       ...       ...   ...   ...   ...       ...      ...
209         0         1         1     0     6     0         0        0
210         0         1         0     0     6     0         0        0
211         0         1         0     0     0     0         0        0
212         0         1         1     0     8     0         0        0
213         0         1         1     0     6     0         0        0

[214 rows x 16 columns]
```

Without further do, we are ready to train our model.

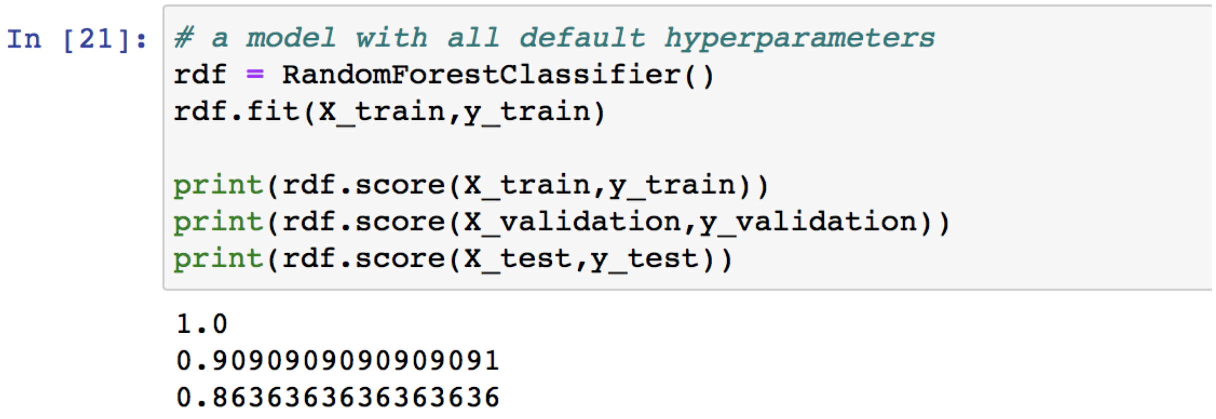
3. Machine Learning Model:

   a) Framework and Tools and b) Validation Methods:

   We proposed KNN as our desired model in Deliverable 1. Taking at a closer look at class.csv, 'number_of_animal_in_species' for each class varies to a large extent. For example, there are 60 mammals but only 16 amphibians. Hence, choosing k in a KNN model for our dataset is a challenging task, as a small k may not capture all mammals while a large k may mistakenly classify animals in general. Thus, we may want to look for other alternatives. The other popular classification models include Naïve Bayes, Support Vector Machine and Random Forest Classifier. Since the features are closely related to each other, we resort to the latter two. While Support Vector Machine tends to separate the different classes in maximal, in reality, some animals of different species have highly similar features. Random Forest Classifier seems intuitively a great option, as our ultimate goal is to go through a series of inputs and obtain our final result, which in this case, could be a leaf.

To build a Random Forest Classifier, we leverage the popular ML library sci-kit learn. To separate our data into training, validation and test sets, we choose a ratio of 80%:10%:10%. We rely on a large training portion because the dataset size is small relative to the size of labels, specifically 214 samples to 7 distinct classes. If the training set is too small, it may not incapsulate enough samples for each class.

We will first create the model with all hyperparameters set to default and see how it goes.

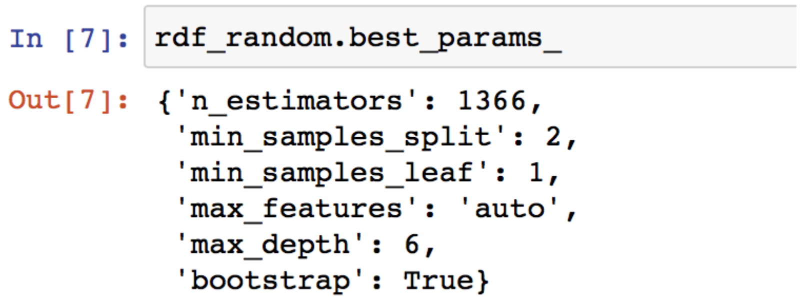


```
In [21]: # a model with all default hyperparameters
         rdf = RandomForestClassifier()
         rdf.fit(X_train,y_train)

         print(rdf.score(X_train,y_train))
         print(rdf.score(X_validation,y_validation))
         print(rdf.score(X_test,y_test))

         1.0
         0.9090909090909091
         0.8636363636363636
```

The accuracies look good as this is our 'raw' model before tuning. The accuracy on the training set is perfect, 1.0, whereas the accuracies on validation and test sets are relatively lower, with the accuracy on test set lower than that of validation test, which warns us the possibility of overfitting. Our dataset is small, thus susceptible to overfitting.

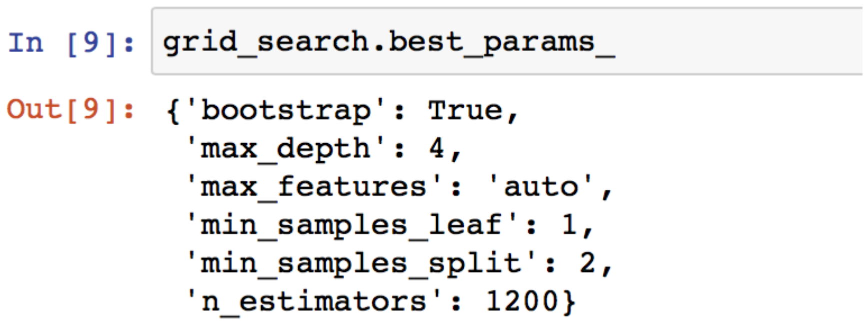
Manually testing out all possible cases on hyperparameters is tedious. We would utilize Randomized Search Cross Validation and Grid Search Cross Validation provided in sci-kit learn. Using these two methods, we wish to find the relatively optimal hyperparameter combination to our model. Our code in this part learnt from the following source:
https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74

After randomized search cross validation, our hyperparameter ranges are narrowed down.



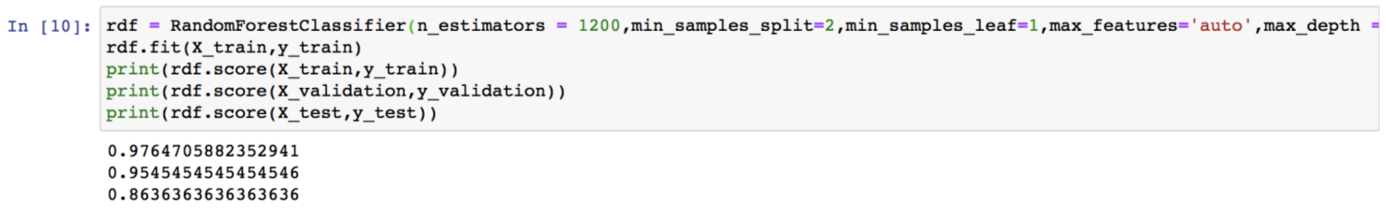
```
In [7]: rdf_random.best_params_

Out[7]: {'n_estimators': 1366,
         'min_samples_split': 2,
         'min_samples_leaf': 1,
         'max_features': 'auto',
         'max_depth': 6,
         'bootstrap': True}
```

We can now perform grid search building on the result from the random search. We will test a range of hyperparameters around the best values returned by random search. The 'best_params_' output is:



```
In [9]: grid_search.best_params_

Out[9]: {'bootstrap': True,
         'max_depth': 4,
         'max_features': 'auto',
         'min_samples_leaf': 1,
         'min_samples_split': 2,
         'n_estimators': 1200}
```

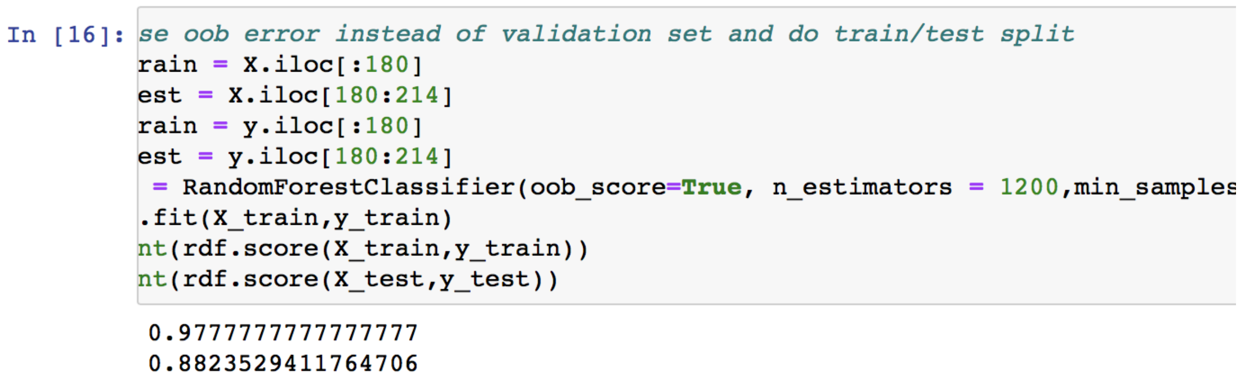
We will use this combination to build our Random Forest Classifier model and evaluate whether accuracies have improved.



```
In [10]: rdf = RandomForestClassifier(n_estimators = 1200,min_samples_split=2,min_samples_leaf=1,max_features='auto',max_depth =
         rdf.fit(X_train,y_train)
         print(rdf.score(X_train,y_train))
         print(rdf.score(X_validation,y_validation))
         print(rdf.score(X_test,y_test))

         0.9764705882352941
         0.9545454545454546
         0.8636363636363636
```

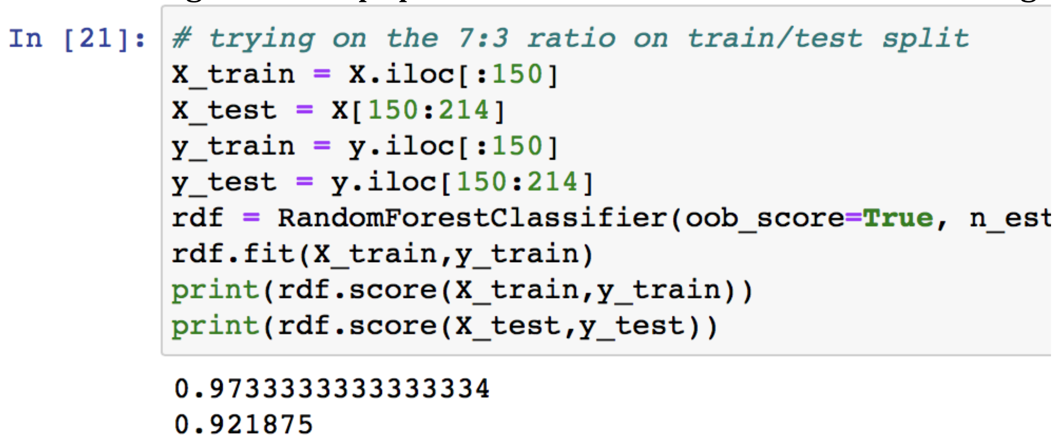
The result indicates that the overfitting issue on our model is alleviated. We would mark this combination as our favorable hyperparameter set for now. On the other hand, the accuracy on the test set could be enhanced, and tuning hyperparameters seemingly did not solve this problem.

The short of data hints us to consider using Out-of-bag error instead of having validation set. Our next approach is to split our data into train and test sets with a ratio of 85%:15%, and test the performance of the model using oob_error. In this experiment, we would employ our favorable hyperparameters tuned previously.



```
In [16]: se oob error instead of validation set and do train/test split
         rain = X.iloc[:180]
         est = X.iloc[180:214]
         rain = y.iloc[:180]
         est = y.iloc[180:214]
          = RandomForestClassifier(oob_score=True, n_estimators = 1200,min_samples
         .fit(X_train,y_train)
         nt(rdf.score(X_train,y_train))
         nt(rdf.score(X_test,y_test))

          0.9777777777777777
          0.8823529411764706
```

The accuracies look better. An accuracy close to 0.9 seems adequate for a ML classification model. Just to make sure this model is not subject to such selection of training set by 'iloc'-ing the first 85% of rows, we would try it on the other ratio of train/test split, such as the 7:3, deriving from the popular ratio 7:1.5:1.5 in machine learning society.



```
In [21]: # trying on the 7:3 ratio on train/test split
         X_train = X.iloc[:150]
         X_test = X[150:214]
         y_train = y.iloc[:150]
         y_test = y.iloc[150:214]
         rdf = RandomForestClassifier(oob_score=True, n_est
         rdf.fit(X_train,y_train)
         print(rdf.score(X_train,y_train))
         print(rdf.score(X_test,y_test))

         0.9733333333333334
         0.921875
```

The outcome is satisfactory. Even it looks nicer than our former 8.5:1.5 split. We may replace that with 7:3 split.

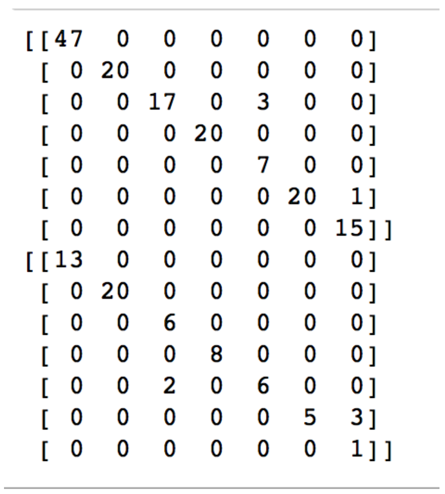
c) Challenges:

- The original zoo.csv dataset only embodies 101 samples, and when building a model upon it, the accuracies were very low and hard to improve. By searching online, I found the other two extended datasets zoo2.csv and zoo3.csv.
- Acquiring these two additional files, the problem became how to merge them and update class.csv. I am not familiar with excel, and hence took a long time researching on manually updating the two csv files in excel.

4. <u>Preliminary Results</u>:

As discussed in the previous section, the performance of our model looks satisfactory. To quantitatively evaluate our model, we want to utilize confusion matrix, with the help of again, sci-kit learn.

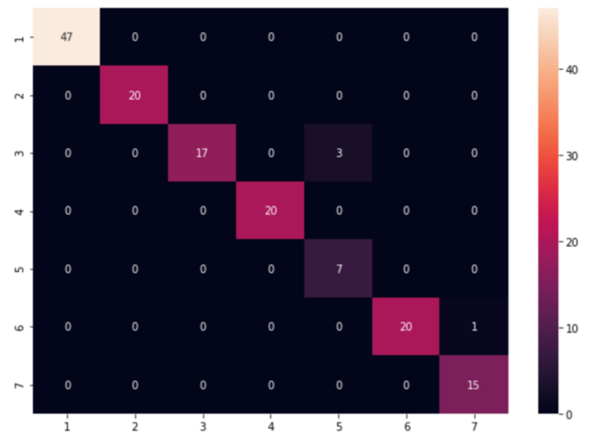
Here is what the two confusion matrices look like, one for training set and the other for test set.



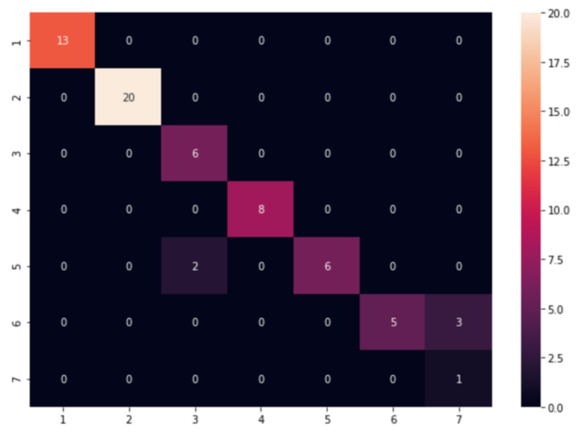
```
[[47  0  0  0  0  0  0]
 [ 0 20  0  0  0  0  0]
 [ 0  0 17  0  3  0  0]
 [ 0  0  0 20  0  0  0]
 [ 0  0  0  0  7  0  0]
 [ 0  0  0  0  0 20  1]
 [ 0  0  0  0  0  0 15]]
[[13  0  0  0  0  0  0]
 [ 0 20  0  0  0  0  0]
 [ 0  0  6  0  0  0  0]
 [ 0  0  0  8  0  0  0]
 [ 0  0  2  0  6  0  0]
 [ 0  0  0  0  0  5  3]
 [ 0  0  0  0  0  0  1]]
```

With most of the numbers are on the diagonal, it illustrates the possibility of misclassification is low in our datasets, indicating a fairly high accuracy of our classification model.

The colored version of these matrices using seaborn is as follows:



Train



Test

The overall performance of our model is satisfactory, with a testing accuracy as high as > 0.9 and a comparable balance between training accuracy and testing accuracy indicates that the model is not overly overfitting. Yet since our final aim is to create a game that guesses the animal the user is thinking, we wish it to have good performances on unknown data as well and we would leave that to our future work to explore the other sets of animals.

5. Next Steps:

   For the future work, we are hoping to find some other sets of zoo animals to test on the accuracy of our model. Furthermore, we will take into consideration the actual mechanism of a guessing game, for example to determine what input we shall expect from the users and how to serialize the input and convert them into features of our model. Finally, we will embark on creating a webpage for our app.