# Abalone Data Analysis

The first step in knowing the data is to know what it contains. This means understanding the type (continuous numeric, discrete numeric or categorical) and meaning of each feature and noting down the number of instances and features in the dataset. (For readers familiar to Excel, features correspond to columns and instances correspond to rows).

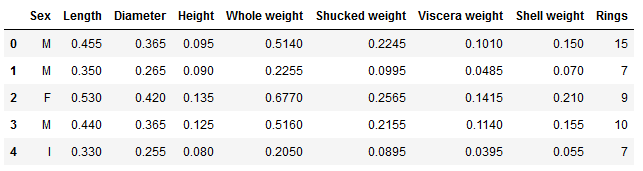
*The aim is to predict the age of abalone from physical measurements. The age of abalone is traditionally determined by cutting the shell through the cone, staining it, and counting the number of rings through a microscope — a boring and time-consuming task. Other measurements, which are easier to obtain, are used to predict the age.*

Let’s now see the type and name of the features:

1. *Sex :*This is the gender of the abalone and has *categorical*value (M, F or I).
2. *Length :*The longest measurement of the abalone shell in mm. *Continuous numeric* value.
3. *Diameter :*The measurement of the abalone shell perpendicular to length in mm. *Continuous numeric* value.
4. *Height :*Height of the shell in mm. *Continuous numeric*value.
5. *Whole Weight :*Weight of the abalone in grams. *Continuous numeric*value.
6. *Shucked Weight :*Weight of just the meat in the abalone in grams. *Continuous numeric* value.
7. *Viscera Weight :*Weight of the abalone after bleeding in grams. *Continuous numeric* value.
8. *Shell Weight :*Weight of the abalone after being dried in grams. *Continuous numeric* value.
9. *Rings :*This is the target, that is the feature that we will train the model to predict. As mentioned earlier, we are interested in the age of the abalone and it has been established that number of rings + 1.5 gives the age. *Discrete numeric* value.

Let’s take a look at the actual data with the code for it.

target\_url = ("http://archive.ics.uci.edu/ml/machine-learning-databases/abalone/abalone.data")# we use pandas to read the dataset and specify its feature names  
abalone = pd.read\_csv(target\_url,header=None, prefix="V")  
abalone.columns = ['Sex', 'Length', 'Diameter', 'Height',  
 'Whole weight', 'Shucked weight',  
 'Viscera weight', 'Shell weight', 'Rings']# this line displays the first five rows of the dataset  
abalone.head()

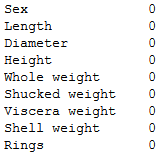


First 5 Rows of the Abalone Dataset

Clean and Analyze the Data

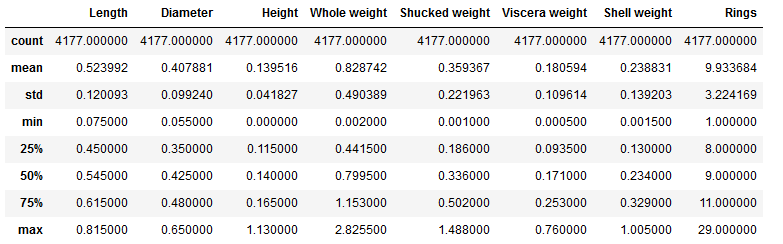
Now that we can make sense of the features the next step is to clean and analyze the data. Both of these go hand-in-hand and we may need to repeatedly do one after the other. But first, cleaning. Why do we clean the data? There are many reasons why data cleaning is important. Sometimes due to wrong entries, the data may contain some weird characters which can confuse the model as to what they mean. Sometimes, some fields have been unintentionally left blank. The model then needs to know how to treat these blank values: whether to ignore them or fill in a default value.

Next, let’s make sure that there are no missing values in the dataset. We can do this with: abalone.isnull().sum(axis = 0) where (axis = 0) specifies that we are summing null values over each feature. We can see from the output there are no null values in the dataset.



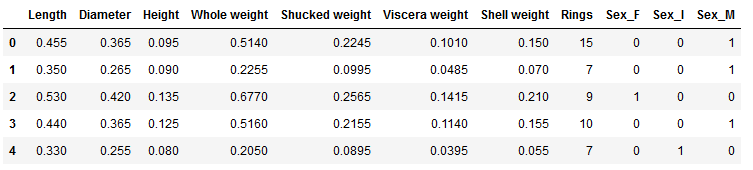
Checking for Null Values in the Dataset

Now, let’s look at some statistics of the dataset possible by: abalone.describe()



Statistical Description of the Dataset

As we can see, the feature *Sex* is missing. This is because the values of *Sex* are categorical and categorical values do not have means and percentiles. A point to note here: ML models find it difficult to work with values of different types (such as both categorical and numeric, as is the case here) at the same time. This is why we will convert *Sex*by doing something called One-hot encoding. which is basically converting a categorical feature into binary numeric feature(s) indicating the presence or absence of the values that were originally there in the categorical feature. This is done by: abalone = pd.get\_dummies(abalone)



First five rows of the converted dataset

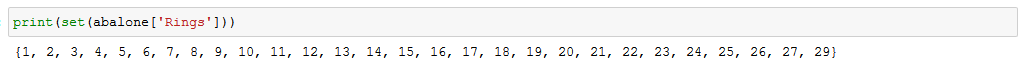
Now, let’s move on to analyzing the dataset. Why do we analyze the dataset? Quite simply, to decide on the best algorithm and the best features to use to train a model on the dataset.

One way to approximate what might be the best features for prediction is to use a correlation heatmap. For those who do not know what [correlation](https://www.mathsisfun.com/data/correlation.html) is, it is simply a measure of the degree to which two variables move in relation to one another. For instance, a positive correlation implies that variable **A** increases or decreases as **B**increases or decreases. Now, what does this have to do with feature selection? Well, we can find out which features are strongly correlated with the one we are trying to predict (**target**) as these are the ones that have the most effect on the target. We do this in code:

#calculate and round off correlation matrix  
corMat = DataFrame(abalone.iloc[:,:8].corr()).values  
corMat = np.around(corMat, decimals = 3)#print correlation with 'Rings' feature  
feature\_importance = DataFrame(abalone.iloc[:,:8].corr()).iloc[:-1, -1].sort\_values(ascending=False)print('Features in Descending Order of Importance', list(feature\_importance.index))

Features in descending order of importance

Here too, *Sex* is missing. And this is again because correlation is a statistical measure for continuous numeric values. Not to say that it isn’t useful but it doesn’t cover all the features in this case. Now, if we look at the distribution (or even the values) of the target in the data, we will find that even that is not continuous but discrete valued.



The set of discrete target values

You may ask, is that a problem? No, its not a problem but a good place to mention that supervised learning problems can be solved in two ways: **regression**and **classification**. Regression is when we train the model to output a value belonging to the real number set. Because both input and output are continuous-valued, correlation is better used here. On the other hand, classification is when we train the model to categorize an input into one of the two(**binary classification)** or more (**multi-class classification**) of the categorical or discrete target values. For instance, in our case any input can be said to belong to an abalone of age in the range of [1,29]. Hence, ours is a multi-class classification problem. Now that we don’t have a method for finding out the best features, that leaves us in a bit of a quandary, doesn’t it? Not quite. There are many other methods but I do not cover them here because they are highly math-based and related to correlation which gives a good approximation anyhow.

Finally, because we are training the model to predict *Rings*, we will remove it from our dataset and pass it separately. We will do this by:

y = abalone["Rings"]  
X = abalone.drop(columns=”Rings”)

Since we are finally done with the cleaning and analyzing, lets dive right into the algorithms! Take note here that training, evaluation and inference are done iteratively, together (even in practice).

In the previous code snippet, we saw the features and the target being separated. By convention, we assign **X** to the input features and **y** to the input target for training. Now, if you remember, after training we also evaluate the model. On what data do we evaluate the model if we give as input the whole dataset? The answer is that we split the dataset into two: One for training known as the **training data** and one for evaluation known as the **test data.**The typical split is in the 80:20 ratio. We do this by: X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

Now, let’s cover the intuition and working for some algorithms.

Algorithm: K-nearest Neighbours (KNN)

**Intuition:**The intuition behind KNN is really simple. Given a point and asked to predict which category/class it belongs to, KNN simply checks the **k**nearest points to the given point and outputs the category that is represented in the majority of the **k**nearest points. How are the nearest points found? The most common way is by using the Euclidean distance, the distance formula that we used to use in school.

K Nearest Neighbours

For instance, in the above image, we have to find what class the point represented by question mark belongs to. If **k=3,**the 3 nearest neighbours are found (given by inner circle); the majority class represented by the neighbouring points is green so the given point belongs to the green class. Similarly, if **k=7**, the point belongs to the red class. And so we can see on changing **k**, the output changes. **k** is called a **hyperparameter**, a parameter supplied by us to train different configs of the models to try and obtain the best possible performance (stage 6 of the model building pipeline).

Performance. How is that measured? The most common method of measuring performance for classifiers is calculating the ratio of instances correctly classified and total instances in the dataset (here just the test dataset).

Let’s see the code:

# Initializing classifier and giving hyperparameter k=3  
knn = KNeighborsClassifier(n\_neighbors=3)# training classifier  
knn.fit(X\_train, y\_train)# Evaluate the classifier  
print(knn.score(X\_test, y\_test))# Try changing hyperparameter  
knn = KNeighborsClassifier(n\_neighbors=5)  
knn.fit(X\_train, y\_train)  
print(knn.score(X\_test, y\_test))

We get accuracy of 0.2021 and 0.2212 for **k=3**and **k=5** respectively.