**A simple case study of K-Means Clustering in Python**

K-means is a clustering algorithm which can be used to find and classify groups of similar points in a dataset. It has been listed as one of the top 10 most important algorithms in data mining.

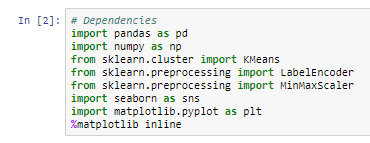
For the implementation part, you will be using the Titanic dataset (available on SOL). Before proceeding with it, I would like to discuss some facts about the data itself. The sinking of the RMS Titanic is one of the most infamous shipwrecks in history. On April 15, 1912, during her maiden voyage, the Titanic sank after colliding with an iceberg, killing 1502 out of 2224 passengers and crew. This sensational tragedy shocked the international community and led to better safety regulations for ships.

One of the reasons that the shipwreck led to such loss of life was that there were not enough lifeboats for the passengers and crew. Although there was some element of luck involved in surviving the sinking, some groups of people were more likely to survive than others, such as women, children, and the upper-class.

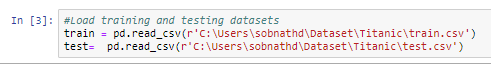
Now, talking about the dataset, the training set contains several records about the passengers of Titanic (hence the name of the dataset). It has 12 features capturing information about passenger\_class, port\_of\_Embarkation, passenger\_fare etc. The dataset's label is **survival** which denotes the survivial status of a particular passenger. Your task is to cluster the records into two i.e. the ones who survived and the ones who did not.

You might be thinking that since it is a labeled dataset, how could it be used for a clustering task? You just have to drop the 'survival' column from the dataset and make it unlabeled. It's the task of K-Means to cluster the records of the datasets if they survived or not.

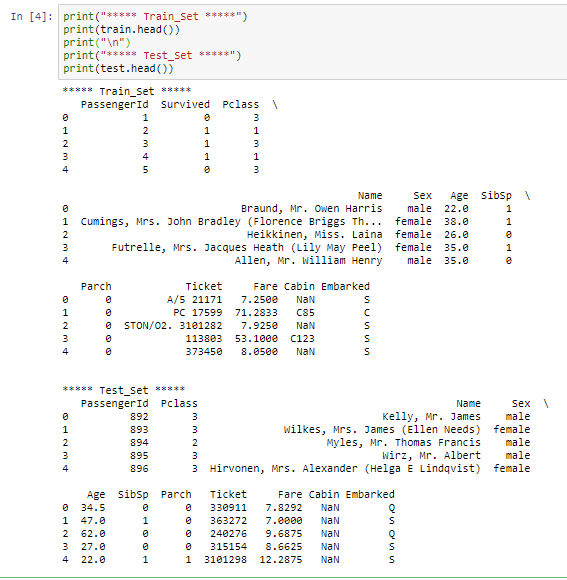
For this tutorial, you will need the following Python packages: pandas, NumPy, scikit-learn, Seaborn and Matplotlib. You also need to download the folder Titanic from SOL which contains different datasets.



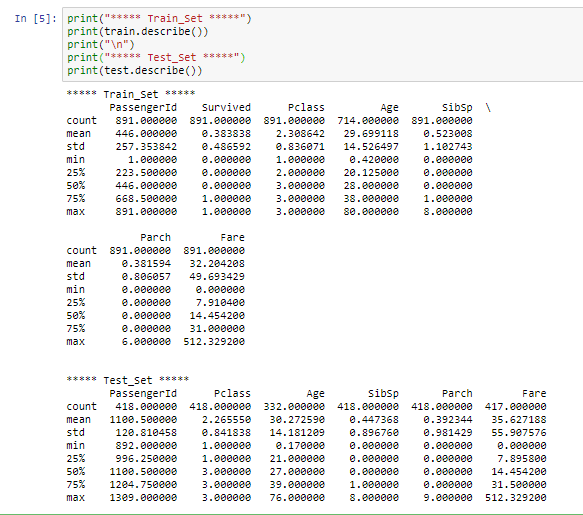
Click on Run, then read the csv files from the Titanic folder you downloaded from SOL as follows:



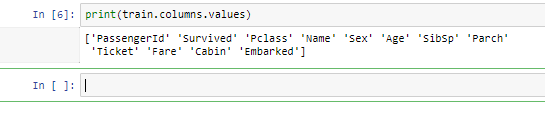
Let's preview the kind of data you will be working with by printing some samples from both the train and test DataFrames.



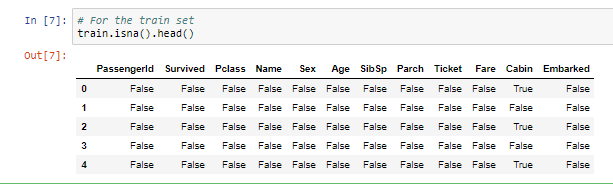
You can get some initial statistics of both the train and test DataFrames using pandas' describe() method.

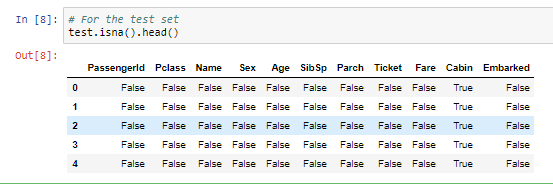


So, from the above outputs you definitely got to know about the features of the dataset and some basic statistics of it. Use the print function as follows and you should be able to see the different columns as printed below.

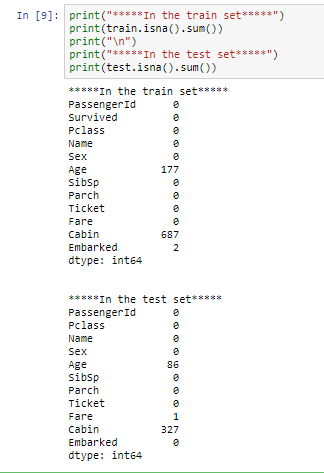


It is very important to note that not all machine learning algorithms support missing values in the data that you are feeding to them. K-Means being one of them. So we need to handle the missing values present in the data. Let's first see where the values are missing:





Let's get the total number of missing values in both datasets.



So, you can see in the training set, in the columns Age, Cabin and Embarked, there are missing values and in the test set, the Age and Cabin columns contain missing values.

There are a couple of ways to handle missing values:

* Remove rows with missing values
* Impute missing values

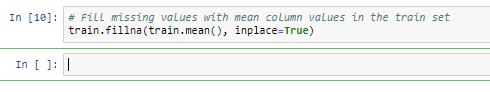
Sometimes, if you remove the rows with missing values it can cause insufficiency in the data which in turn results in inefficient training of the machine learning model.

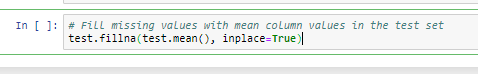
Now, there are several ways you can perform the imputation:

* A constant value that has meaning within the domain, such as 0, distinct from all other values.
* A value from another randomly selected record.
* A mean, median or mode value for the column.
* A value estimated by another machine learning model.

Any imputation performed on the train set will have to be performed on test data in the future when predictions are needed from the final machine learning model. This needs to be taken into consideration when choosing how to impute the missing values.

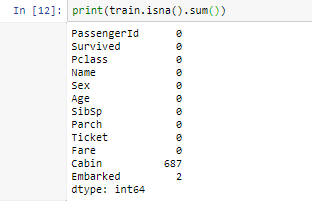
Pandas provides the fillna() function for replacing missing values with a specific value. Let's apply that with **Mean Imputation**. Write the codes and hit Run. Do this for both the train and test datasets as shown below.



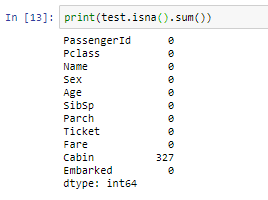


Now that you have imputed the missing values in the dataset, it's time to see if the dataset still has any missing values.

For the training dataset:



Let's see if you have any missing values in the test set.

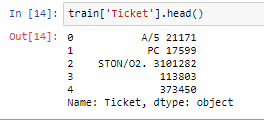


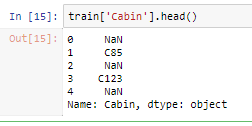
Yes, you can see there are still some missing values in the Cabin and Embarked columns. This is because these values are non-numeric. In order to perform the imputation the values need to be in numeric form. There are ways to convert a non-numeric value to a numeric one, but we will leave this for now.

Let's do some more analytics in order to understand the data better. Understanding is really required in order to perform any Machine Learning task. Let's start with finding out which features are categorical and which are numerical.

* Categorical: Survived, Sex, and Embarked. Ordinal: Pclass.
* Continuous: Age, Fare. Discrete: SibSp, Parch.

Two features are left out which are not listed above in any of the categories. Yes, you guessed it right, **Ticket** and **Cabin**. Ticket is a mix of numeric and alphanumeric data types. Cabin is alphanumeric. Let see some sample values.



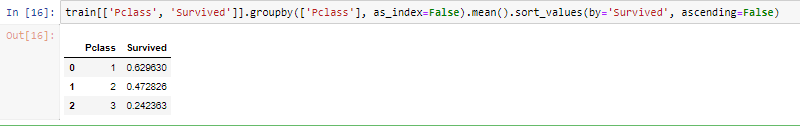


Let's see the survival count of passengers with respect to the following features:

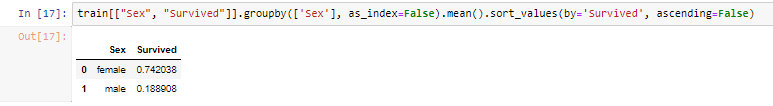
* Pclass
* Sex
* SibSp
* Parch

Let's do that one by one:

**Survival count with respect to Pclass:**

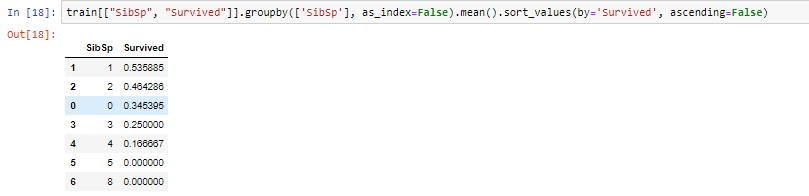


**Survival count with respect to Sex:**

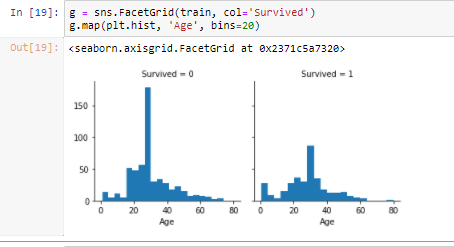


You can see the survival rate of female passengers is significantly higher for males.

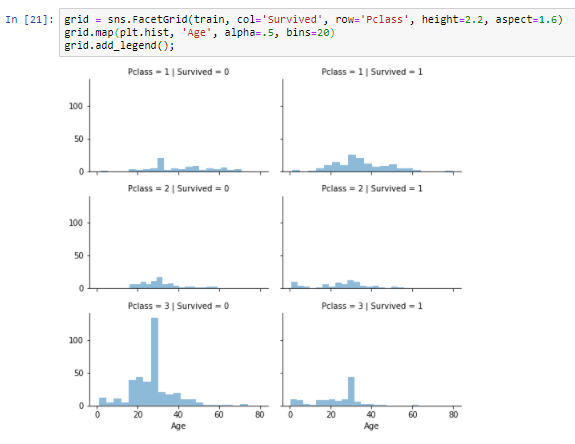
**Survival count with respect to SibSp:**



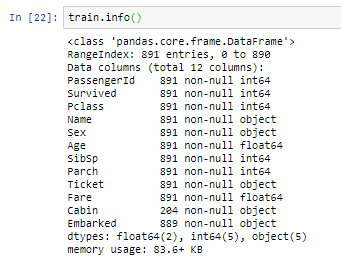
Now it's time for some quick plotting. Let's first plot the graph of "Age vs. Survived":



It is time to see how the Pclass and Survived features are related to each other with a graph:



Enough of visualization and analytics for now! Let's actually build a K-Means model with the training set. But before that you will need some data pre-processing as well. You can see that not all the feature values are of same type. Some of them are numerical and some of them are not. In order to ease the computation, you will feed all numerical data to the model. Let's see the data types of different features that you have:



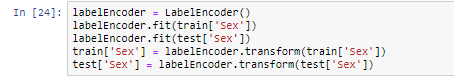
So, you can see that the following features are non-numeric as denoted by the ‘object’:

* Name
* Sex
* Ticket
* Cabin
* Embarked

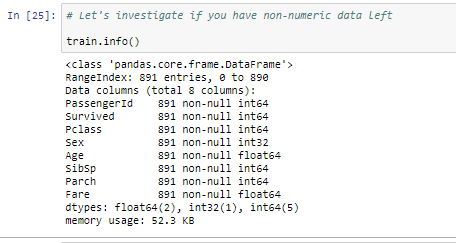
Before converting them into numeric ones, you might want to do some feature engineering, i.e. features like Name, Ticket, Cabin and Embarked do not have any impact on the survival status of the passengers. Often, it is better to train your model with only significant features than to train it with all the features, including unnecessary ones. It not only helps in efficient modelling, but also the training of the model can happen in much lesser time. Although, feature engineering is a whole field of study itself, I will encourage you to dig it further. But for this tutorial, know that the features Name, Ticket, Cabin and Embarked can be dropped and they will not have significant impact on the training of the K-Means model.



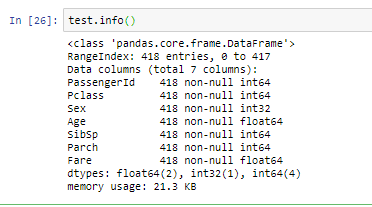
Now that the dropping part is done let's convert the 'Sex' feature to a numerical one (only 'Sex' is remaining now which is a non-numeric feature). You will do this using a technique called [Label Encoding](http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html).



Let's investigate if you have non-numeric data left



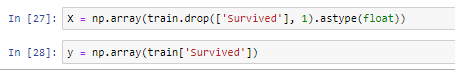
**Note** that the test set does not have the Survived feature.



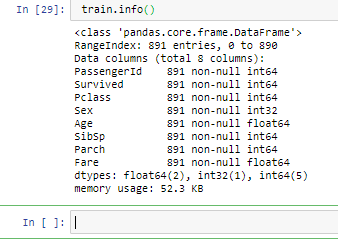
### Brilliant!

Looks like you are good to go to train your K-Means model now.

You can first drop the Survival column from the data with the drop() function and make Y your target/dependent variable.

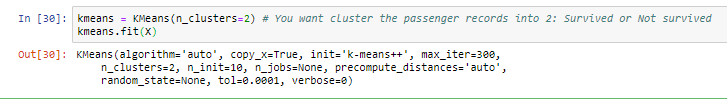


You can review all the features you are going to feed to the algorithm with train.info().

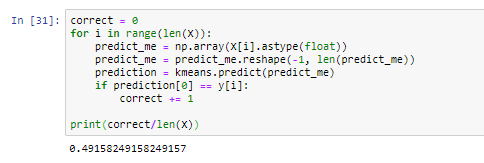


Let's now build the K-Means model.

You want cluster the passenger records into 2: **Survived** or **Not survived**



You can see all the other parameters of the model other than n\_clusters. Let's see how well the model is doing by looking at the percentage of passenger records that were clustered correctly.



That is nice for the first go. Your model was able to cluster correctly with a 49.1% (accuracy of your model). But in order to enhance the performance of the model you could tweak some parameters of the model itself.

Now, let's discuss K-Means's limitations.

### Disadvantages of K-Means

Now that you have a fairly good idea on how K-Means algorithm works let's discuss some its disadvantages.

The biggest disadvantage is that K-Means requires you to pre-specify the number of clusters (k). However, for the Titanic dataset, you had some domain knowledge available that told you the number of people who survived in the shipwreck. This might not always be the case with real world datasets. Hierarchical clustering is an alternative approach that does not require a choice of clusters. An additional disadvantage of k-means is that it is sensitive to outliers and different results can occur if you change the ordering of the data.

K-Means is a lazy learner where generalization of the training data is delayed until a query is made to the system. This means K-Means starts working only when you trigger it to, thus lazy learning methods can construct a different approximation or result to the target function for each encountered query. It is a good method for online learning, but it requires a possibly large amount of memory to store the data, and each request involves starting the identification of a local model from scratch.

### Conclusion

So, in this tutorial you scratched the surface of one of the most popular clustering techniques; K-Means. You learned about its inner mechanics, implemented it using the Titanic Dataset in Python, and you also got a fair idea of its disadvantages.