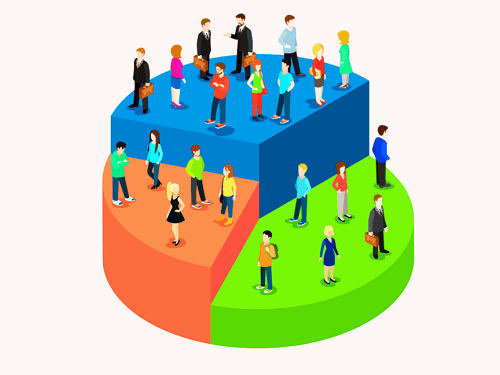
**Customer Segmentation using Data Science**

**TEAM MEMBER**

**732521104030 : NAVEEN S**

**Phase 3 Submission Document**

**Project : Customer Segmentation using Data Science**



**Introduction**

The main purpose behind this study was to analyze the problems faced by big retail banks during business expansion. Typically banks tend to acquire new customers at huge costs rather than leveraging their existing customer base. A bank’s customers leave behind a large footprint in terms of the transactions they perform, which can be analyzed to understand their behavior pattern which may be leveraged for selling new products.

This paper analyzes the customers’ transaction patterns, product holdings, demographics, past trends, and other attributes to devise an effective strategy for engaging them further. In a retail bank with various product offerings, the focus should be on customer segmentation and profiling to ensure ease of targeting, marketing, and offering personalized products to retain profitable customers and capturing market share across geographies.

**Content For Project Phase 3** :

Phase 3: Development Part 1

Dataset Link : (https://www.kaggle.com/datasets/luiscorter/netflix-original-films-imdb-scores

**Data Loading**

The Load method provides a technique for filling a single DataTable with data, retrieved from an IDataReader instance. This method provides the same functionality, but allows you to load multiple result sets from an IDataReader into multiple tables within a DataSet.

**5 Different Ways to Load Data in Python**

As a beginner, you might only know a single way to load data (normally in CSV) which is to read it using pandas.read\_csv function. It is one of the most mature and strong functions, but other ways are a lot helpful and will definitely come in handy sometimes.

The ways that I am going to discuss are:

* Manual function
* loadtxt function
* genfromtxt function
* read\_csv function
* Pickle

The dataset that we are going to use to load data can be found [here](http://eforexcel.com/wp/downloads-18-sample-csv-files-data-sets-for-testing-sales/). It is named as 100-Sales-Records.

**Imports**  
We will use Numpy, Pandas, and Pickle packages so import them.

import numpy as np

import pandas as pd

import pickle

**1. Manual Function**

This is the most difficult, as you have to design a custom function, which can load data for you. You have to deal with Python’s normal filing concepts and using that you have to read a .csv file.

Let’s do that on 100 Sales Records file.

def load\_csv(filepath):

data = []

col = []

checkcol = False

with open(filepath) as f:

for val in f.readlines():

val = val.replace("\n","")

val = val.split(',')

if checkcol is False:

col = val

checkcol = True

else:

data.append(val)

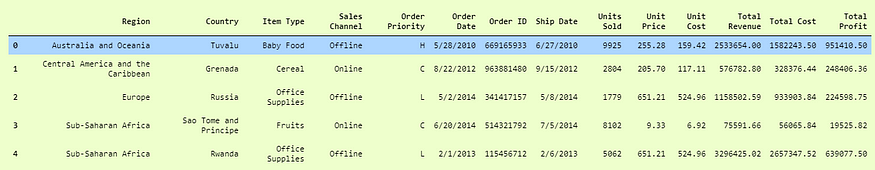
df = pd.DataFrame(data=data, columns=col)

return df

**Output**

myData = load\_csv('100 Sales Record.csv')

print(myData.head())



**2. Numpy.loadtxt function**

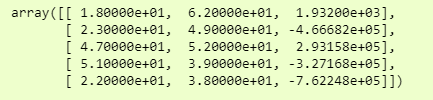
This is a built-in function in Numpy, a famous numerical library in Python. It is a really simple function to load the data. It is very useful for reading data which is of the same datatype.

When data is more complex, it is hard to read using this function, but when files are easy and simple, this function is really powerful.

df = np.loadtxt('convertcsv.csv', delimeter = ',')

Here we simply used the loadtxt function as passed in delimeter as ',' because this is a CSV file.

Now if we print df, we will see our data in pretty decent numpy arrays that are ready to use.

print(df[:5,:])

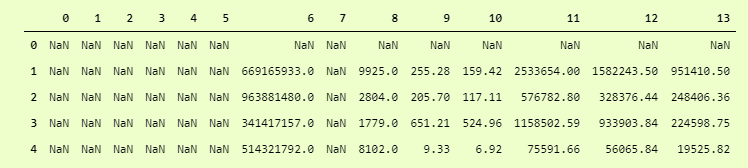
**3. Numpy.genfromtxt()**

We will use the dataset, which is ‘100 Sales Records.csv’ which we used in our first example to demonstrate that we can have multiple data types in it.

Let’s jump to code.

data = np.genfromtxt('100 Sales Records.csv', delimiter=',')

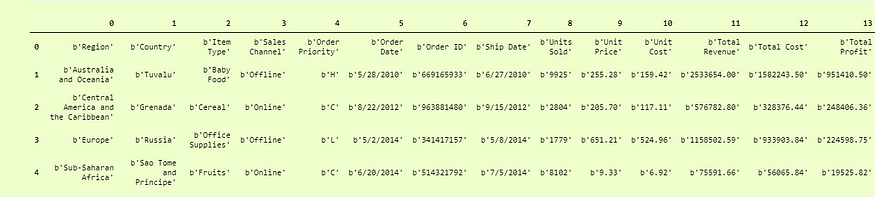
and to see it more clearly, we can just see it in a dataframe format, i.e.,

>>> pd.DataFrame(data)

Just add another dtype parameter and set dtype to None which means that it has to take care of datatypes of each column itself. Not to convert whole data to single dtype.

data = np.genfromtxt('100 Sales Records.csv', delimiter=',', dtype=None)

And then for output:

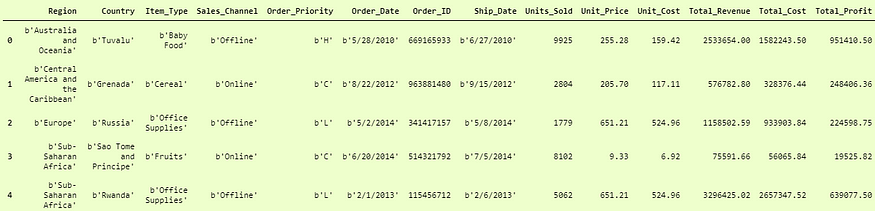
>>> pd.DataFrame(data).head()

Quite better than the first one, but here our Columns titles are Rows, to make them column titles, we have to add another parameter which is names and set it to True so it will take the first row as the Column Titles.

i.e.

data = np.genfromtxt('100 Sales Records.csv', delimiter=',', dtype=None, names = True)

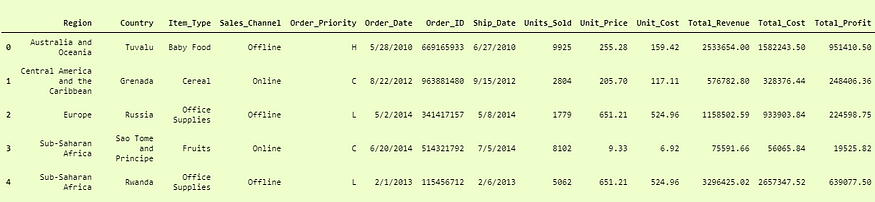
and we can print it as:

>>> pd.DataFrame(df3).head()

Now the last problem is that the columns which are of string data types are not the actual strings, but they are in bytes format. You can see that before every string, we have a b' so to encounter them, we have to decode them in utf-8 format.

df3 = np.genfromtxt('100 Sales Records.csv', delimiter=',', dtype=None, names=True, encoding='utf-8')

This will return our dataframe in the desired form.

>>> pd.DataFrame(df3)

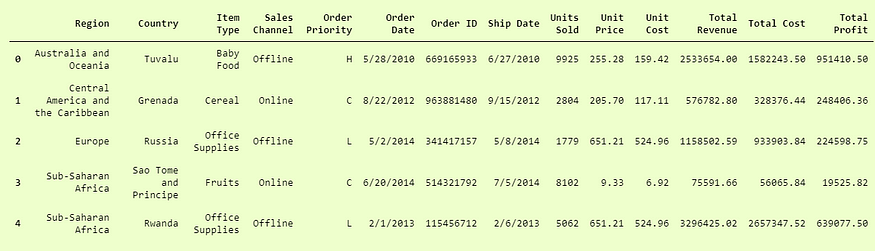
**4. Pandas.read\_csv()**

Pandas is a very popular data manipulation library, and it is very commonly used. One of it’s very important and mature functions is read\_csv() which can read any .csv file very easily and help us manipulate it. Let’s do it on our 100-Sales-Record dataset.

This function is very popular due to its ease of use. You can compare it with our previous codes, and you can check it.

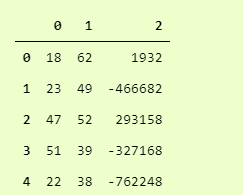
>>> pdDf = pd.read\_csv('100 Sales Record.csv')

>>> pdDf.head()

This was actually so simple and easy to use. Pandas.read\_csv definitely offers a lot of other parameters to tune in our data set, for example in our convertcsv.csv file, we had no column names so we can read it as:

>>> newdf = pd.read\_csv('convertcsv.csv', header=None)

>>> newdf.head()



**5. Pickle**

When your data is not in a good, human-readable format, you can use pickle to save it in a binary format. Then you can easily reload it using the pickle library.

We will take our 100-Sales-Record CSV file and first save it in a pickle format so we can read it.

with open('test.pkl','wb') as f:

pickle.dump(pdDf, f)

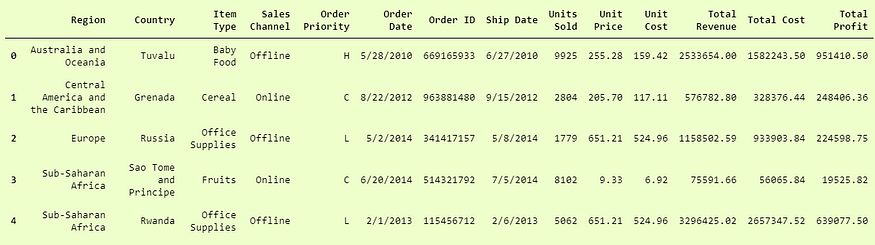
This will create a new file test.pkl which has inside it our pdDf from Pandas heading.

Now to open it using pickle, we just have to use pickle.load function.

with open("test.pkl", "rb") as f:

d4 = pickle.load(f)

>>> d4.head()



**Data preprocessing**

Data preprocessing is converting raw data into legible and defined sets that allow businesses to conduct data mining, analyze the data, and process it for business activities. It's important for businesses to preprocess their data correctly, as they use various forms of input to collect raw data, which can affect its quality. Preprocessing data is an important step, as raw data can be inconsistent or incomplete in its formatting. Effectively preprocessing raw data can increase its accuracy, which can increase the quality of projects and improve its reliability.

**Importance of data preprocessing**

Preprocessing data is an important step for data analysis. The following are some benefits of preprocessing data:

It improves accuracy and reliability. Preprocessing data removes missing or inconsistent data values resulting from human or computer error, which can improve the accuracy and quality of a dataset, making it more reliable.

It makes data consistent. When collecting data, it's possible to have data duplicates, and discarding them during preprocessing can ensure the data values for analysis are consistent, which helps produce accurate results.

It increases the data's algorithm readability. Preprocessing enhances the data's quality and makes it easier for machine learning algorithms to read, use, and interpret it.

**Conclusion**

The data preprocessing phase is crucial for determining the correct input data for the machine learning algorithms. As we saw previously, without applying the proper techniques, you can have a worse model result. For example, the k-nearest neighbors algorithm is affected by noisy and redundant data, is sensitive to different scales, and doesn’t handle a high number of attributes well. If you use this algorithm, you must clean the data, avoid high dimensionality and normalize the attributes to the same scale.

One last important thing to remember, which is usually a common mistake in this field, is that you need to split your dataset into training and test sets before applying some of these techniques, using only the training set to learn and apply it in the test part. For those already familiar with Python and sklearn, you apply the fit and transform method in the training data, and only the transform method in the test data.