**Customer Segmentation**   
 **using Data Science**

**Team members:**

1) F . ANTONY RAJAN (961721104303)

2) A . FRENI (961721104008)

3) A . EBISTON (961721104307)

4) P . FRANKLIN GINO (961721104308)

5) M.S. JEBINLAL (961721104309)

**Project Title:** Customer segmentation using

Data Science

**Phase 4: Development Part 2**

**Topic:** In this section continue building the project by performing different activities like feature engineering, model training, evaluation etc as per the instructions in the project.



**Customer segmentation using Data Science**

**Introduction:**

* Customer segmentation involves implementing data science methods to divide the customer base into smaller groups based on certain characteristics.
* It assists marketing managers in better understanding their customer’s preferences and presenting them with better targeted advertisements.
* A method of analysing a client base and grouping customers into categories or segments which share particular attributes.

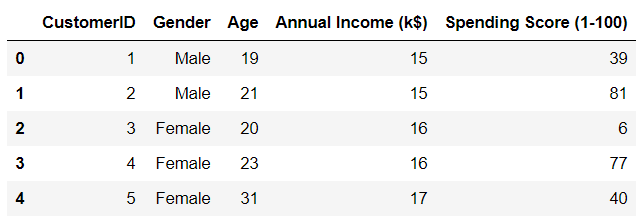
**Data Collection :**

* Data sources
* Data collection methods
* Data cleaning and preprocessing steps
* Mention any challenges encountered

**Data set:**

import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
import sea  
from kneed import KneeLocator  
from sklearn.datasets import make\_blobs  
from sklearn.cluster import KMeans  
from sklearn.metrics import silhouette\_score  
from sklearn.preprocessing import StandardScaler  
from sklearn.decomposition import PCA  
from mpl\_toolkits.mplot3d import Axes3D  
df =pd.read\_csv('**E:\Customer Segmentation.csv’**)

df.head()



**Necessary step to follow:**

**1)Import Libraries:**

**Start by importing the necessary lidraries;**

**Program:**

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_test\_split

from sklearn.preprocessing import StandardScaler

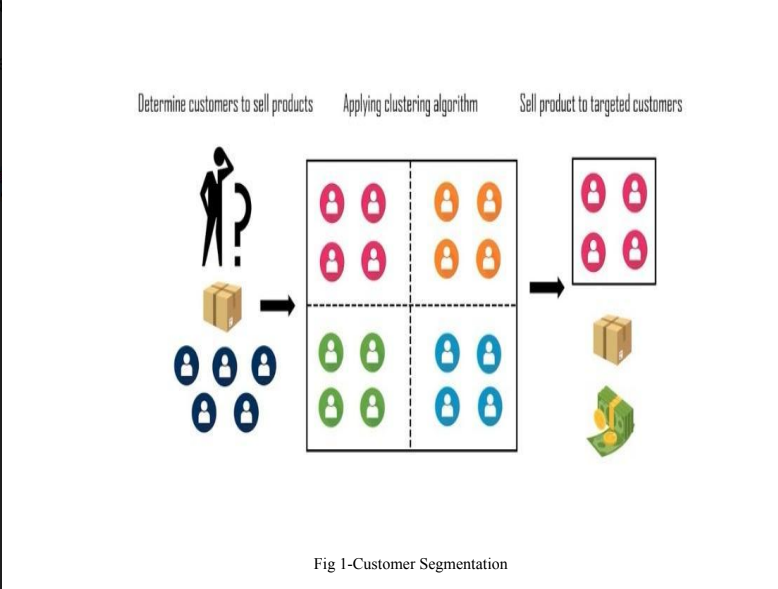
**2)Load the Dataset:**

**Load your dataset into a Pandas DataFrame. You can typically find house price datasets in CSV format,but you can adapt this code to other formats as needed.**

**Program:**

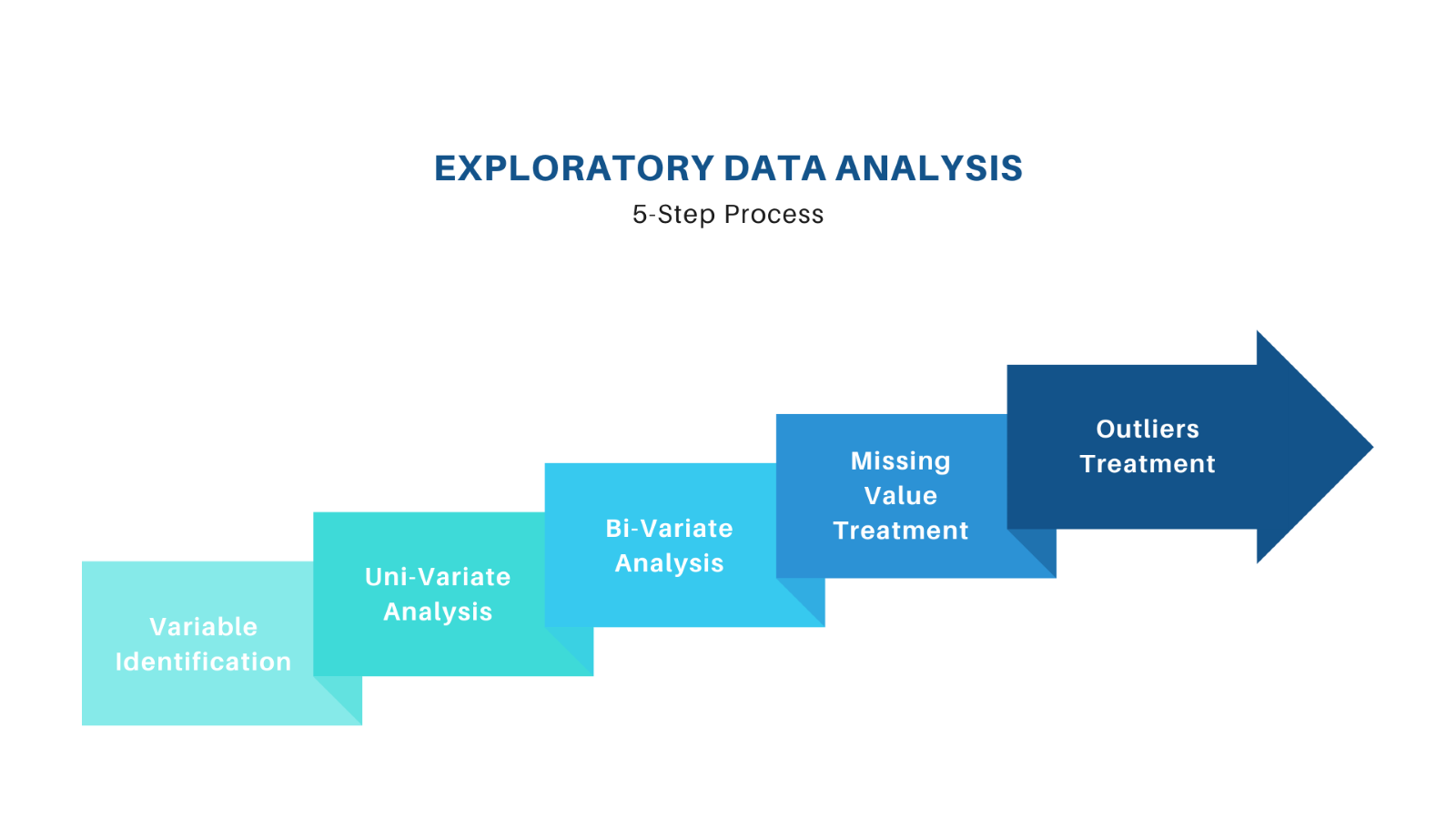
**df =pd.read\_csv(‘E:\Customer Segmentation.csv’)**

**pd.read()**



**3)Exploratory Data Analysis:**

* Summary statistics
* Data virualization (Histograms,scatter plots,etc.)
* Key insights from EDA.

  
**Program:**

import pandas as pd  
import seaborn as sns  
import matplotlib.pyplot as plt  
import plotly.express as px  
import numpy as np  
from scipy.stats import iqr  
from sklearn.preprocessing import StandardScaler  
from sklearn.cluster import KMeans  
df = pd.read\_csv("data/marketing\_campaign.csv", sep="\t")  
df.head()

df["TotalAmountSpent"] = df["MntFishProducts"] + df["MntFruits"] + df["MntGoldProds"] + df["MntSweetProducts"] + df["MntMeatProducts"] + df["MntWines"]

**4)Pre processing:**

* Data Segmentation is the process of taking the data you hold and dividing it up and grouping similar data together based on the chosen parameters so that you can use it more efficiently within marketing and operations.
* Determination of the Need of the Segment.
* Identification of the Segment.



**5) Split the Data:**

* Split your dataset into training and testing sets.
* This helps you evaluateyour model's performance later.

Program:

X = df.drop('price', axis=1)

# Features y = df['price'] # Target variable

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

**Importance of loading and processing dataset:**

* Loading and preprocessing the dataset is an important first step in building any applied data science model.
* However, it is especially important for Customer segmentation using data science models, as Customer segmentation datasets are often complex and noisy.
* By loading and preprocessing the dataset, we can ensure that the machine learning algorithm is able to learn from the data effectively and accurately

**Loading the dataset:**

* Loading the dataset using machine learning is the process of bringing the data into the machine learning environment so that it can be used to train and evaluate a model.
* The specific steps involved in loading the dataset will vary depending on the machine learning library or framework that is being used.
* However, there are some general steps that are common to most machine learning frameworks:

**a.Identify the dataset:**

* The first step is to identify the dataset that you want to load. This dataset may be stored in a local file, in a database, or in a cloud storage service.

**b.Load the dataset:**

* Once you have identified the dataset, you need to load it into the machine learning environment.
* This may involve using a built-in function in the machine learning library, or it may involve writing your own code.

**c.Preprocess the dataset:**

* Once the dataset is loaded into the machine learning environment, you may need to preprocess it before you can start training and evaluating your model.
* This may involve cleaning the data, transforming the data into a suitable format, and splitting the data into training and test sets.
* Preprocess the dataset Load the dataset Identify the dataset Loading the dataset Here, how to load a dataset using machine learning in Python

Program:

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import r2\_score, mean\_absolute\_error,mean\_squared\_error

from sklearn.linear\_model import LinearRegression

from sklearn.linear\_model import Lasso

from sklearn.ensemble import RandomForestRegressor

from sklearn.svm import SVR

import xgboost as xg

%matplotlib inline

import warnings

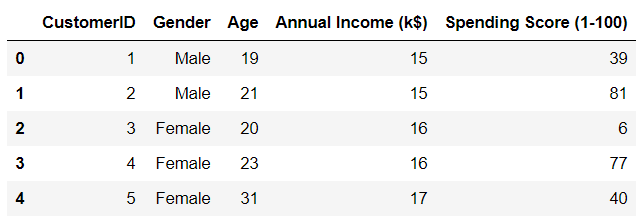
warnings.filterwarnings("ignore") /opt/conda/lib/python3.10/site-packages/scipy/\_\_init\_\_.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required forthis version of SciPy (detected version 1.23.5

warnings.warn(f"A NumPy version >={np\_minversion} and<{np\_maxversion}"

**Loading Dataset:**

dataset = pd.read\_csv('E: **\Customer Segmentation**.csv')

**Output:**



**Customer Segmentation Methodology:**

* Customer segmentation is a crucial start for business to better understand and target their diverse customer base.
* Customer segmentation is a crucial strategy for businesses to better understand and target their diverse customer base

## **Data preparation:**

import pandas as pd  
 import numpy as np  
 import matplotlib as mpl  
 import matplotlib.pyplot as plt  
 import seaborn as sns  
 import datetime, nltk, warnings  
 import matplotlib.cm as cm  
 import itertools  
 from pathlib import Path

from sklearn.preprocessing import StandardScaler  
 from sklearn.cluster import KMeans  
 from sklearn.metrics import silhouette\_samples, silhouette\_score  
 from sklearn import preprocessing, model\_selection, metrics, feature\_selection

from sklearn.model\_selection import GridSearchCV, learning\_curve  
 from sklearn.svm import SVC  
 from sklearn.metrics import confusion\_matrix  
 from sklearn import neighbors, linear\_model, svm, tree, ensemble  
 from wordcloud import WordCloud, STOPWORDS  
 from sklearn.ensemble import AdaBoostClassifier  
 from sklearn.decomposition import PCA  
 from IPython.display import display, HTML  
 import plotly.graph\_objs as go  
 from plotly.offline import init\_notebook\_mode,iplot  
 init\_notebook\_mode(connected=True)  
 warnings.filterwarnings("ignore")  
 plt.rcParams["patch.force\_edgecolor"] = True  
 plt.style.use('fivethirtyeight')  
 mpl.rc('patch', edgecolor = 'dimgray', linewidth=1)  
 %matplotlib inline

df\_initial = pd.read\_csv('../input/data.csv',encoding="ISO-8859-1",  
dtype={'CustomerID': str,'InvoiceID':str})  
 print('Dataframe dimensions:', df\_initial.shape)  
 #\_\_\_\_\_\_  
 df\_initial['InvoiceDate'] = pd.to\_datetime(df\_initial['InvoiceDate'])

tab\_info=pd.DataFrame(df\_initial.dtypes).T.rename(index={0:'column type'})  
 tab\_info=tab\_info.append(pd.DataFrame(df\_initial.isnull().sum()).T.rename(index={0:'null values (nb)'}))  
 tab\_info=tab\_info.append(pd.DataFrame(df\_initial.isnull().sum()/df\_initial.shape[0]\*100).T.  
 rename(index={0:'null values (%)'}))  
 display(tab\_info)

display(df\_initial[:5])

Dataframe dimensions: (541909, 8)

**Out:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| InvoiceNo | StockCode | Description | Quantity | InvoiceDate | UnitPrice | CustomerID | Country |  |
| column type | object | object | object | int64 | datetime64[ns] | float64 | object | object |
| null values (nb) | 0 | 0 | 1454 | 0 | 0 | 0 | 135080 | 0 |
| null values (%) | 0 | 0 | 0.268311 | 0 | 0 | 0 | 24.9267 | 0 |

**Exploring the content of variables:**

data = dict(type='choropleth',  
 locations = countries.index,  
 locationmode = 'country names', z = countries,  
 text = countries.index, colorbar = {'title':'Order nb.'},  
 colorscale=[[0, 'rgb(224,255,255)'],  
 [0.01, 'rgb(166,206,227)'], [0.02, 'rgb(31,120,180)'],  
 [0.03, 'rgb(178,223,138)'], [0.05, 'rgb(51,160,44)'],  
 [0.10, 'rgb(251,154,153)'], [0.20, 'rgb(255,255,0)'],  
 [1, 'rgb(227,26,28)']],   
reversescale = False)

layout = dict(title='Number of orders per country',  
 geo = dict(showframe = True, projection={'type':'mercator'}))

choromap = go.Figure(data = [data], layout = layout)  
 iplot(choromap, validate=False)

**Out:**

**products, transactions, customers,**

**quantity, 3684, 22190, 4372**

### **Customers and products:**

temp = df\_initial.groupby(by=['CustomerID', 'InvoiceNo'], as\_index=False)['InvoiceDate'].count()  
 nb\_products\_per\_basket = temp.rename(columns = {'InvoiceDate':'Number of products'})  
 nb\_products\_per\_basket[:10].sort\_values('CustomerID')

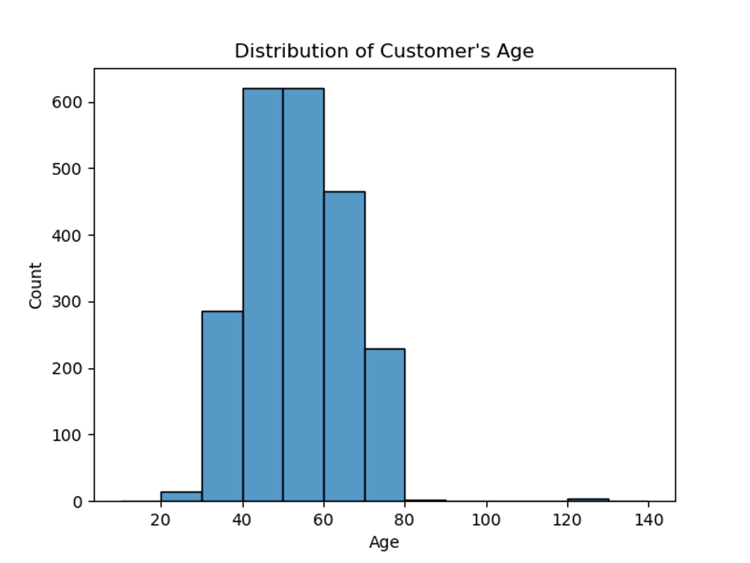
**Out:**

|  |  |  |  |
| --- | --- | --- | --- |
| CustomerID | InvoiceNo | Number of products |  |
| 0 | 12346 | 541431 | 1 |
| 1 | 12346 | C541433 | 1 |
| 2 | 12347 | 537626 | 31 |
| 3 | 12347 | 542237 | 29 |
| 4 | 12347 | 549222 | 24 |
| 5 | 12347 | 556201 | 18 |

**Univariate analysis**

* Univariate analysis entails evaluating a single feature in order to get insights about it. So, the initial step in performing EDA is to undertake univariate analysis, which includes evaluating descriptive or summary statistics about the feature.

sns.histplot(data=df, x="Age", bins = list(range(10, 150, 10)))  
 plt.title("Distribution of Customer's Age")



**Bivariate Analysis**

* After you've performed univariate analysis on all your feature of interest, the next step is to perform bivariate analysis. This involves comparing two attributes at the same time.

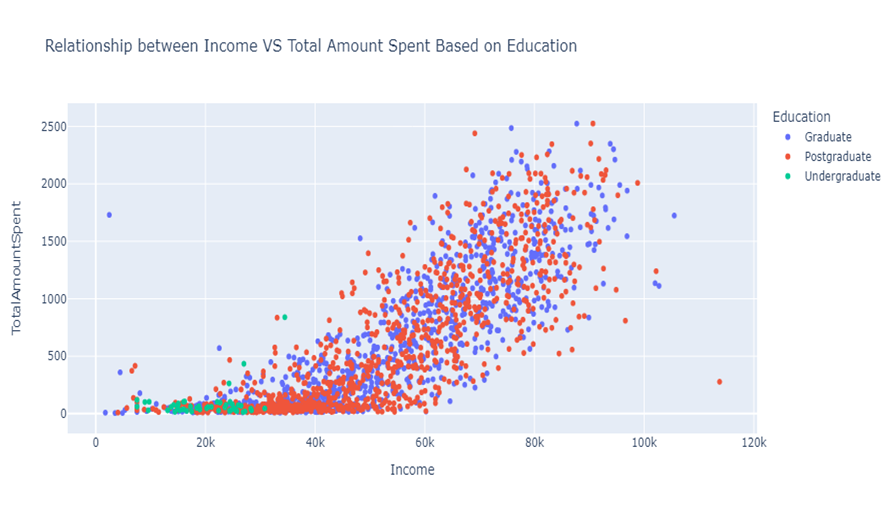
fig = px.scatter(data\_frame=df\_cut, x="Income",  
 y="TotalAmountSpent",title="Relationship Between Customer's Income and Total Amount Spent", height=500,  
 color\_discrete\_sequence = px.colors.qualitative.G10[1:])  
 fig.show()



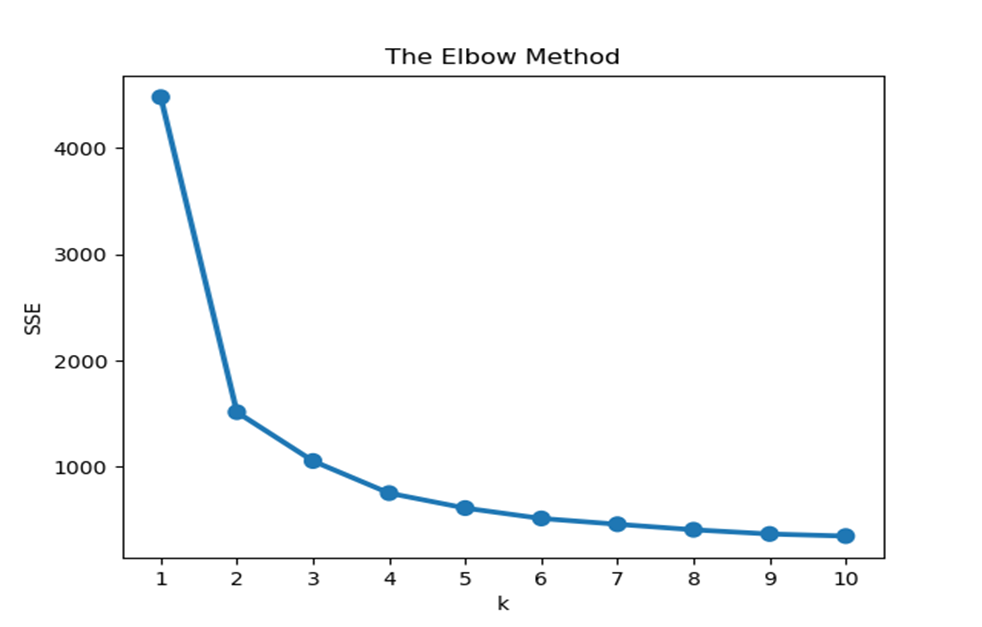
**Multivariate Analysis:**

* After you've completed univariate (analysis of single feature) and bivariate (analysis of two features) analysis, the last phase of EDA is to perform Multivariate Analysis

fig = px.scatter( data\_frame=df\_cut, x = "Income",y= "TotalAmountSpent",  
 title = "Relationship between Income VS Total Amount Spent Based on Education",color = "Education",height=500)  
 fig.show()



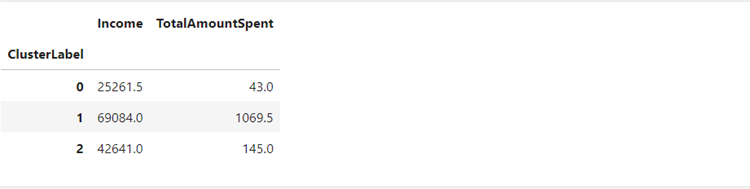
errors = []  
 for k in range(1, 11):  
 model = KMeans(n\_clusters=k, random\_state=42)  
 model.fit(df\_scaled)  
 error.append(model.inertia\_)  
   
   
 plt.title('The Elbow Method')  
 plt.xlabel('k'); plt.ylabel('Error of Cluster')  
 sns.pointplot(x=list(range(1, 11), y=errors)  
 plt.show()



**How to Interpret the Cluster Result:**

* Now that we've built the model, the next thing will be to interpret the result from each cluster.
* There are numerous way you can summarize the results of your cluster depending on what you want to achieve.

data.groupby("ClusterLabel")[["Income", "TotalAmountSpent"]].median()



We can see that there is a trend within the clusters:

* Cluster 0 translates to customers who earn less and spend less.
* Cluster 1 represent customers that earn more and spend more.
* Cluster 2 represents customers that earn moderate and spend moderate.

fig = px.scatter(  
 data\_frame=data,  
 x = "Income",  
 y= "TotalAmountSpent",  
 title = "Relationship between Income VS Total Amount Spent",  
 color = "ClusterLabel",  
 height=500  
 )  
 fig.show()



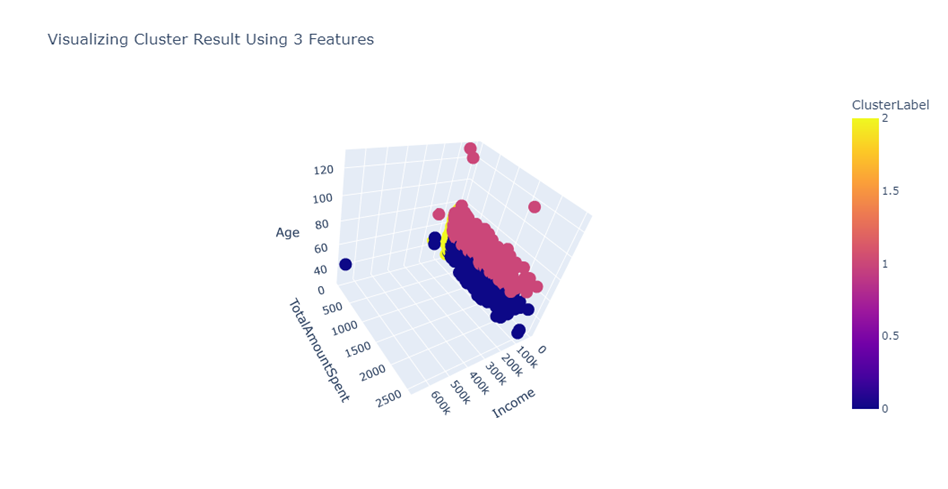
data = df[["Age", "Income", "TotalAmountSpent"]]  
 df\_log = np.log(data)  
 std\_scaler = StandardScaler()  
 df\_scaled = std\_scaler.fit\_transform(df\_log)model = KMeans(n\_clusters=3, random\_state=42)  
 model.fit(df\_scaled)  
   
data = data.assign(ClusterLabel= model.labels\_)  
   
result = df\_result.groupby("ClusterLabel").agg({"Age":"mean", "Income":"median", "TotalAmountSpent":"median"}).round()



We can see from the above summary that:

* Cluster 0 depicts young customers that earn a lot and also spend a lot.
* Cluster 1 translates to older customers that earn a lot and also spend a lot.
* Cluster 2 depicts young customers that earn less and also spend less.

fig = px.scatter\_3d(data\_frame=data, x="Income",   
 y="TotalAmountSpent", z="Age", color="ClusterLabel", height=550,  
 title = "Visualizing Cluster Result Using 3 Features")  
 fig.show()



## **Classification of customers:**

* In this part, the objective will be to adjust a classifier that will classify consumers in the different client categories that were established in the previous section.
* The objective is to make this classification possible at the first visit.
* To fulfill this objective, I will test several classifiers implemented in scikit-learn

**Program:**

class Class\_Fit(object):  
 def \_\_init\_\_(self, clf, params=None):  
 if params:   
 self.clf = clf(\*\*params)  
 else:  
 self.clf = clf()  
  
 def train(self, x\_train, y\_train):  
 self.clf.fit(x\_train, y\_train)  
  
 def predict(self, x):  
 return self.clf.predict(x)  
   
 def grid\_search(self, parameters, Kfold):  
 self.grid = GridSearchCV(estimator = self.clf, param\_grid = parameters, cv = Kfold)  
   
 def grid\_fit(self, X, Y):  
 self.grid.fit(X, Y)  
   
 def grid\_predict(self, X, Y):  
 self.predictions = self.grid.predict(X)  
 print("Precision: {:.2f} % ".format(100\*metrics.accuracy\_score(Y, self.predictions)))

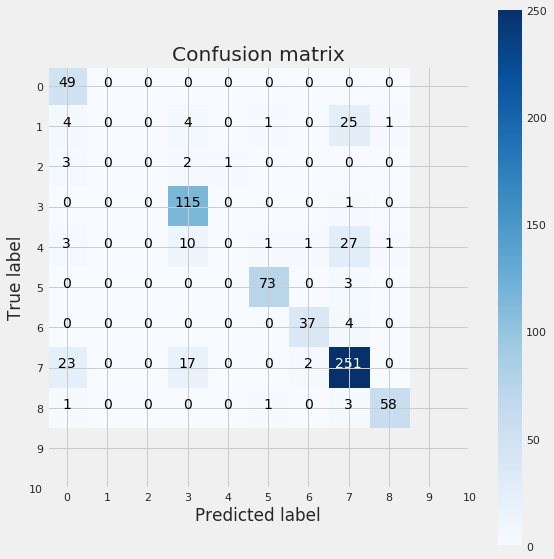
olumns = ['mean', 'categ\_0', 'categ\_1', 'categ\_2', 'categ\_3', 'categ\_4' ]  
X = selected\_customers[columns]  
Y = selected\_customers['cluster']

X\_train, X\_test, Y\_train, Y\_test = model\_selection.train\_test\_split(X, Y, train\_size = 0.8)

**Confusion matrix:**

The accuracy of the results seems to be correct.

def plot\_confusion\_matrix(cm, classes, normalize=False, title='Confusion matrix', cmap=plt.cm.Blues):  
 if normalize:  
 cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]  
 print("Normalized confusion matrix")  
 else:  
 print('Confusion matrix, without normalization')  
 plt.imshow(cm, interpolation='nearest', cmap=cmap)  
 plt.title(title)  
 plt.colorbar()  
 tick\_marks = np.arange(len(classes))  
 plt.xticks(tick\_marks, classes, rotation=0)  
 plt.yticks(tick\_marks, classes)  
 fmt = '.2f' if normalize else 'd'  
 thresh = cm.max() / 2.  
 for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):  
 plt.text(j, i, format(cm[i, j], fmt),  
 horizontalalignment="center",  
 color="white" if cm[i, j] > thresh else "black")  
 plt.tight\_layout()  
 plt.ylabel('True label')  
 plt.xlabel('Predicted label')

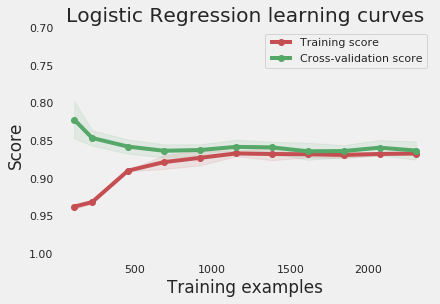
class\_names = [i for i in range(11)]  
cnf\_matrix = confusion\_matrix(Y\_test, svc.predictions)   
np.set\_printoptions(precision=2)  
plt.figure(figsize = (8,8))  
plot\_confusion\_matrix(cnf\_matrix, classes=class\_names, normalize = False, title='Confusion matrix')

### **Logistic Regression:**

* I now consider the logistic regression classifier.
* As before, I create an instance of the Class\_Fit class, adjust the model on the training data and see how the predictions compare to the real values:

**Program:**

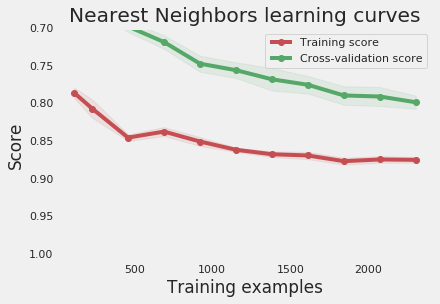
lr = Class\_Fit(clf = linear\_model.LogisticRegression)  
lr.grid\_search(parameters = [{'C':np.logspace(-2,2,20)}], Kfold = 5)  
lr.grid\_fit(X = X\_train, Y = Y\_train)  
lr.grid\_predict(X\_test, Y\_test)

g = plot\_learning\_curve(lr.grid.best\_estimator\_, "Logistic Regression learning curves", X\_train, Y\_train,ylim = [1.01, 0.7], cv = 5,train\_sizes = [0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8,0.9, 1])

### **k-Nearest Neighbors:**

knn = Class\_Fit(clf = neighbors.KNeighborsClassifier)  
knn.grid\_search(parameters = [{'n\_neighbors': np.arange(1,50,1)}], Kfold = 5)  
knn.grid\_fit(X = X\_train, Y = Y\_train)  
knn.grid\_predict(X\_test, Y\_test)

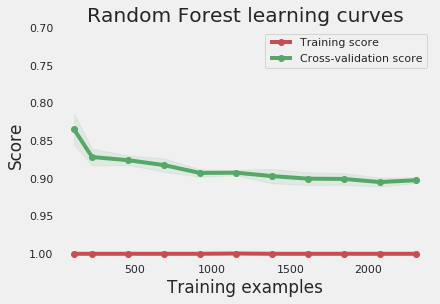
g = plot\_learning\_curve(knn.grid.best\_estimator\_, "Nearest Neighbors learning curves", X\_train, Y\_train,ylim = [1.01, 0.7], cv = 5, train\_sizes = [0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1])



### **Random Forest:**

rf = Class\_Fit(clf = ensemble.RandomForestClassifier)  
param\_grid = {'criterion' : ['entropy', 'gini'], 'n\_estimators' : [20, 40, 60, 80, 100],'max\_features' :['sqrt', 'log2']}  
rf.grid\_search(parameters = param\_grid, Kfold = 5)  
rf.grid\_fit(X = X\_train, Y = Y\_train)  
rf.grid\_predict(X\_test, Y\_test)  
Precision: 89.61 %

g = plot\_learning\_curve(rf.grid.best\_estimator\_, "Random Forest learning curves", X\_train, Y\_train,ylim = [1.01, 0.7], cv = 5,train\_sizes = [0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1])



# Conclusion:

* customer segmentation is a powerful strategy that enables businesses to gain a deeper understanding of their diverse customer base and tailor their marketing, product development, and customer service efforts to meet the specific needs of different customer groups.
* Understanding the data's structure, characteristics, and anypotential issues through exploratory data analysis (EDA) isessential for informed decision-making.
* With these foundational steps completed, our dataset is now primed for the subsequent stages of building and training a Customer segmentation using data science model.