

# ABSTRACT

In this competitive commercial world, an organization has to satisfy the needs and wants of the customers, and has to attract new customers, and hence enhance their business. Customer value is considered as a control element for all business strategies. Therefore, every organization has to emphasize on customer satisfaction. So, the best approach to satisfy majority of the customers is to classify them into groups or clusters which saves resources and find best possible suggestions to that particular group. With Unsupervised learning approach this segmentation can be achieved. There are various unsupervised learning methods we choose K-Means clustering algorithm as it is cost efficient, fast and easy to implement technique. Clustering based on their age, Average salary their spending's and gender into optimum number of clusters based on the data helped us to analyze the customers interests. After modeling the customer base we visualize them, which helps us to identify the trends based on their previous shopping history we get to know properly about the whole segment which helps in improving the market. With the help of Python language computation became lot easier. Python libraries are useful and fast working. Working with K Means require the data to be pre-processed. Various pre-processing techniques are used based on type of data. Missing data, Categorical variables need to be handled before modeling, as their presence makes computation resourceful. After modeling a efficient model we analyze the market segments.

# Table of Contents

1. Abstract	1
2. Table of Contents	2
3. Introduction	3
4. Existing Method	4-6
5. Proposed Method	7
6. Methodology	8-14
7. Implementation	15-25
8. Analysis	26-28
9. Conclusion	29

# Introduction

Customer segmentation is important for businesses to understand their target audience. Different advertisements can be curated and sent to different audience segments based on their demographic profile, interests, and affluence level.

There are many unsupervised machine learning algorithms that can help companies identify their user base and create consumer segments. In this project, we will be looking at a popular unsupervised learning technique called K-Means clustering. This algorithm can take in un-labelled customer data and assign each data point to clusters. The goal of K-Means is to group all the data available into non-overlapping sub-groups that are distinct from each other.

That means each sub-group/cluster will consist of features that distinguish them from other clusters. K-Means clustering is a commonly used technique by data scientists to help companies with customer segmentation. After segmenting the customers we need to analyze the clusters and find patterns which help us to add a new audience based on their interests into particular segment

In this Project, we will learn the following:

- Data pre-processing for K-Means clustering
- Building a K-Means clustering algorithm from scratch
- Visualizing clusters built
- Interpretation and analysis of clusters built

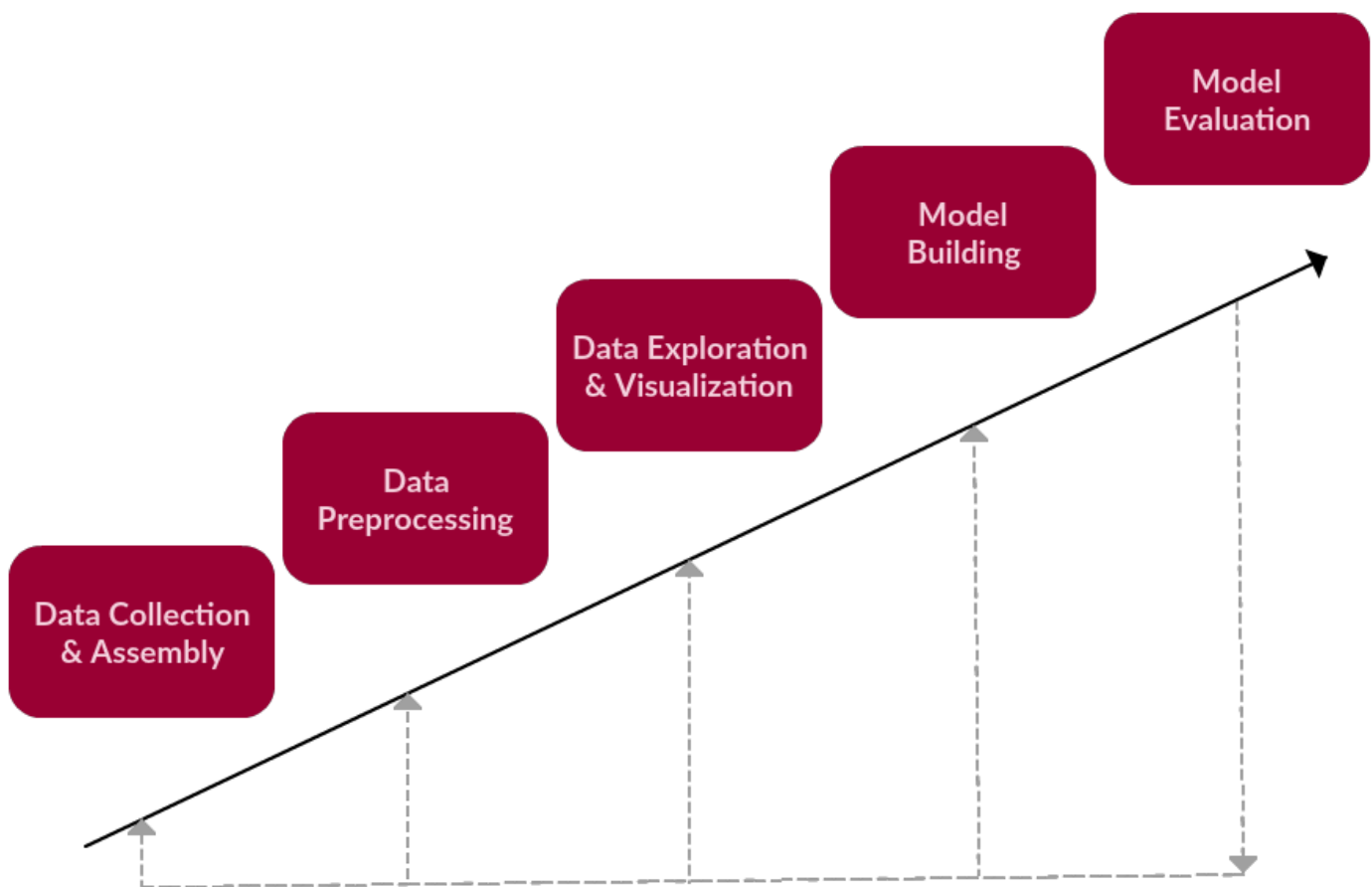
## Existing Method

This is a general existing way for creating a model.

### **Data Collection& Assembly:**

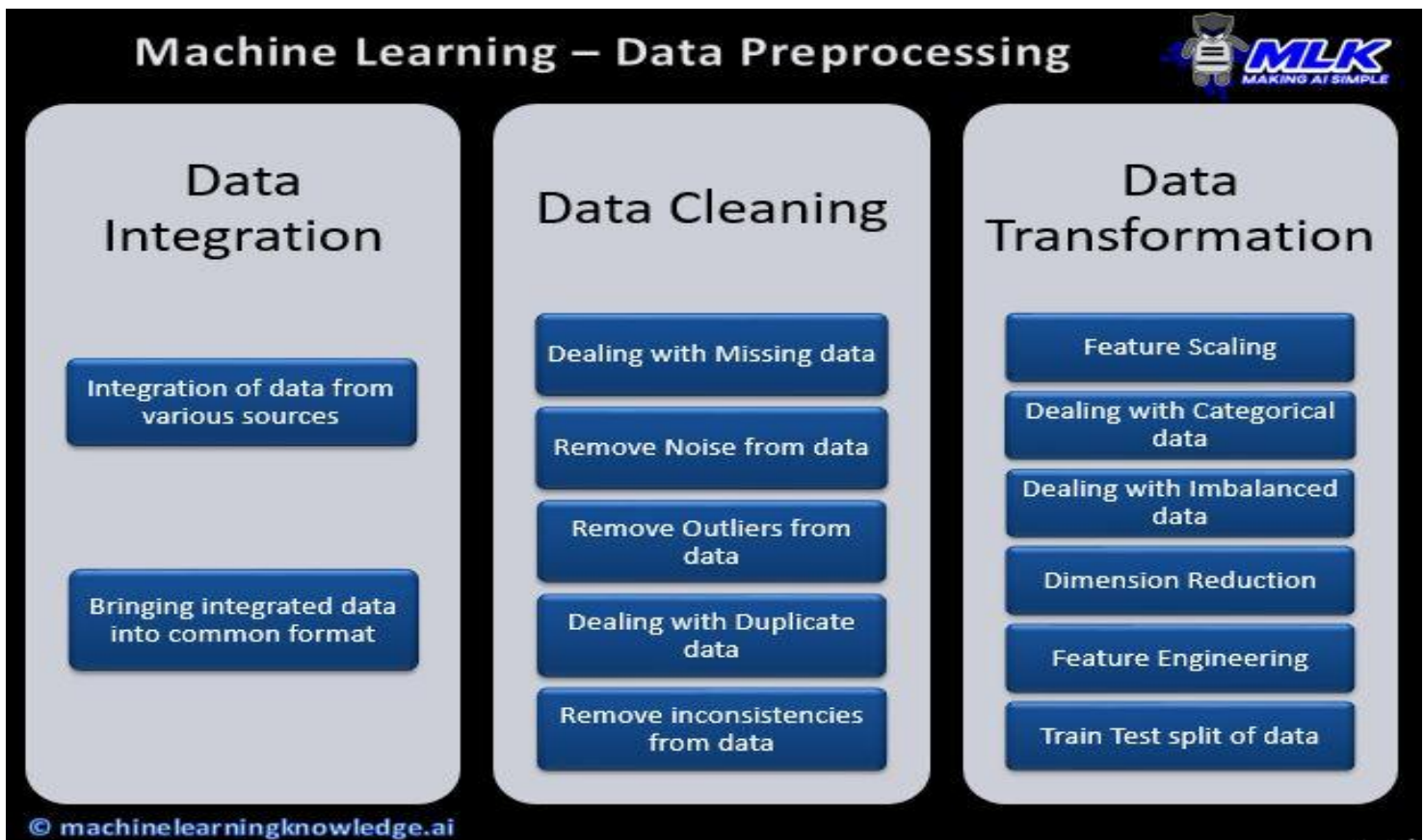
As we are provided directly with a dataset, This step is complete.

For creating a Model we need to first have glimpse of our data. Based on our dataset we come to conclusion for any preprocessing steps needed to be done before we use the data fitting the model.



## Data Processing:

After having a glimpse of our data we get to know about our data. As per our requirement we preprocess our data. Before preprocessing we check for any missing values, any inconsistent data entry and act per the situation.



## Data Visualization:

Now our data is good for modeling. Before we model our data we explore and visualize our data to have a look our data and to note trends, relationship between various entries. It's a crucial step as it provides all the insights we miss while looking at tabular data.



## Model Building:

Using Various libraries of Python we can build a model. Most frequently used Scikit Learn has many useful and highly efficient algorithms. We select a model based on our requirement. We train the model with our dataset and tune the parameters based on requirement.

## Model Evaluation:

We evaluate the performance of our model, there are various scores which explain the accuracy of our model. This is an important factor for modifying our model. Depending on the accuracy we may change the model.

## Proposed Method

### **Data Pre-processing:**

As we already have a dataset we can skip the Data Collection & Assembly step. So our next concern is to pre-process the raw data so that our model works accurately. Here we are using K-Means we need to make necessary process that updates our dataset.

### **Model Building:**

As the raw data is processed now, we can go ahead to our model building step. In, this step we decide the model based on our requirement and create it from scratch using Scikit Learn library. There may already pre tuned models with good accuracy but we don't know it's accuracy on our model. Here, we are using K-Means, it requires number of clusters parameter. So, we find the optimum number of clusters using Elbow Method. As we get the number of clusters we start creating our model. After model is create we fit the model with our data and transform the dataset.

### **Data Exploration & Visualization:**

Now we explore our data come to know about the relationship between the columns and their trends. After having a brief image of our model's output we Visualize our data so we get a good relationship between the columns and their interests, way to approach to new product etc many factors.

### **Cluster Analysis:**

Our K-Means Clustering model outputs us the clusters. Our aim for this project is to segment the customers. Now we need to analyze our clusters. As provides the company with insights of segments of customers.

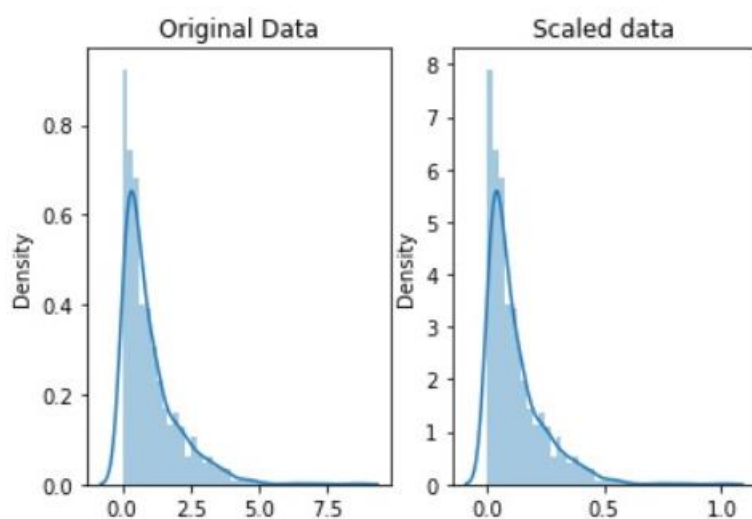
# Methodology

## Data Pre-Processing:

### 1. Scaling Data:

This means that you're transforming your data so that it fits within a specific scale, like 0-100 or 0-1. You want to scale data when you're using methods based on measures of how far apart data points are, like K-Means Clustering Algorithm, SVM or KNN. With these algorithms, a change of "1" in any numeric feature is given the same importance.

For example, you might be looking at the prices of some products in both Yen and US Dollars. One US Dollar is worth about 100 Yen, but if you don't scale your prices, methods like SVM or KNN will consider a difference in price of 1 Yen as important as a difference of 1 US Dollar! This clearly doesn't fit with our intuitions of the world. With currency, you can convert between currencies. But what about if you're looking at something like height and weight? It's not entirely clear how many pounds should equal one inch. By scaling your variables, you can help compare different variables on equal footing. Notice that the *shape* of the data doesn't change, but that instead of ranging from 0 to 8ish, it now ranges from 0 to 1.





## 2. Categorical Variables:

A **categorical variable** takes only a limited number of values.

- Consider a survey that asks how often you eat breakfast and provides four options: "Never", "Rarely", "Most days", or "Every day". In this case, the data is categorical, because responses fall into a fixed set of categories.
- If people responded to a survey about which what brand of car they owned, the responses would fall into categories like "Honda", "Toyota", and "Ford". In this case, the data is also categorical.

You will get an error if you try to plug these variables into most machine learning models in Python without preprocessing them first. In this tutorial, we'll compare three approaches that you can use to prepare your categorical data.

### Three Approaches

#### 1) Drop Categorical Variables

The easiest approach to dealing with categorical variables is to simply remove them from the dataset. This approach will only work well if the columns did not contain useful information.

#### 2) Ordinal Encoding

**Ordinal encoding** assigns each unique value to a different integer.



Breakfast
Every day
Never
Rarely
Most days
Never


Breakfast
3
0
1
2
0

This approach assumes an ordering of the categories: "Never" (0) < "Rarely" (1) < "Most days" (2) < "Every day" (3).

This assumption makes sense in this example, because there is an indisputable ranking to the categories. Not all categorical variables have a clear ordering in the values, but we refer to those that do as **ordinal variables**. For tree-based models (like decision trees and random forests), you can expect ordinal encoding to work well with ordinal variables.

### 3) One-Hot Encoding

**One-hot encoding** creates new columns indicating the presence (or absence) of each possible value in the original data. To understand this, we'll work through an example



Color	Red	Yellow	Green
Red	1	0	0
Red	1	0	0
Yellow	0	1	0
Green	0	0	1
Yellow	0	1	0

In the original dataset, "Color" is a categorical variable with three categories: "Red", "Yellow", and "Green". The corresponding one-hot encoding contains one column for each possible value, and one row for each row in the original dataset. Wherever the original value was "Red", we put a 1 in the "Red" column; if the original value was "Yellow", we put a 1 in the "Yellow" column, and so on.

In contrast to ordinal encoding, one-hot encoding *does not* assume an ordering of the categories. Thus, you can expect this approach to work particularly well if there is no clear ordering in the categorical data (e.g., "Red" is neither *more* nor *less* than "Yellow"). We refer to categorical variables without an intrinsic ranking as **nominal variables**. One-hot encoding generally does not perform well if the categorical variable takes on a large number of values (i.e., you generally won't use it for variables taking more than 15 different values).

## **Model Building:**

### **K-Means Clustering:**

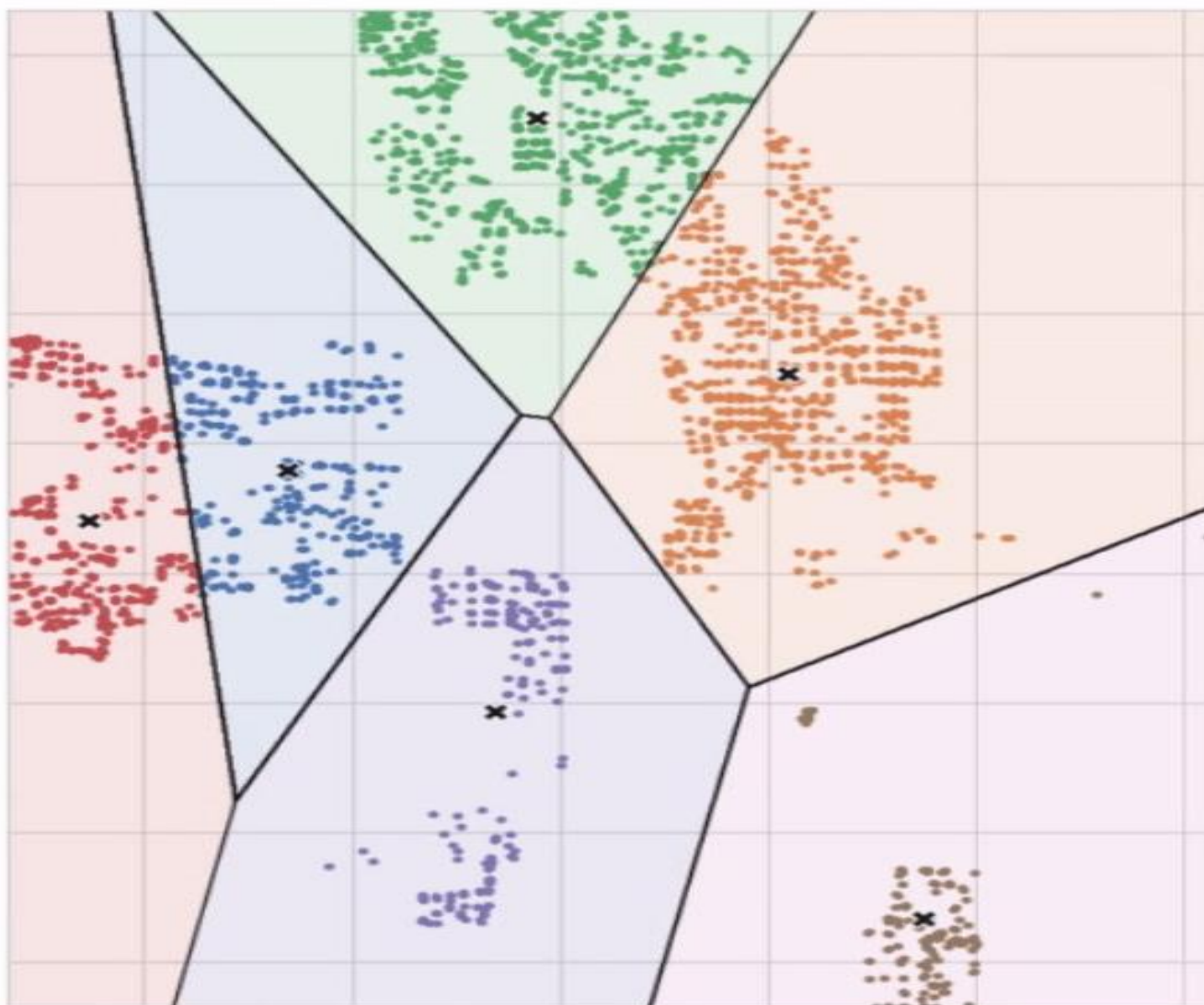
Unsupervised algorithms don't make use of a target; instead, their purpose is to learn some property of the data, to represent the structure of the features in a certain way. In the context of feature engineering for prediction, you could think of an unsupervised algorithm as a "feature discovery" technique.

**Clustering** simply means the assigning of data points to groups based upon how similar the points are to each other. A clustering algorithm makes "birds of a feather flock together," so to speak.

When used for feature engineering, we could attempt discover groups of customers representing a market segment, for instance, or geographic areas that share similar weather patterns. Adding a feature of cluster labels can help machine learning models untangle complicated relationships of space or proximity.

There are a great many clustering algorithms. They differ primarily in how they measure "similarity" or "proximity" and in what kinds of features they work with. The algorithm we'll use, k-means, is intuitive and easy to apply in a feature engineering context. Depending on your application another algorithm might be more appropriate.

**K-means clustering** measures similarity using ordinary straight-line distance (Euclidean distance, in other words). It creates clusters by placing a number of points, called **centroids**, inside the feature-space. Each point in the dataset is assigned to the cluster of whichever centroid it's closest to. The "k" in "k-means" is how many centroids (that is, clusters) it creates. You define the k yourself.



*K-means clustering creates a Voronoi tessellation of the feature space.*

Let's review how the k-means algorithm learns the clusters and what that means for feature engineering. We'll focus on three parameters from scikit-learn's implementation: `n_clusters`, `max_iter`, and `n_init`.

It's a simple two-step process. The algorithm starts by randomly initializing some predefined number (`n_clusters`) of centroids. It then iterates over these two operations:

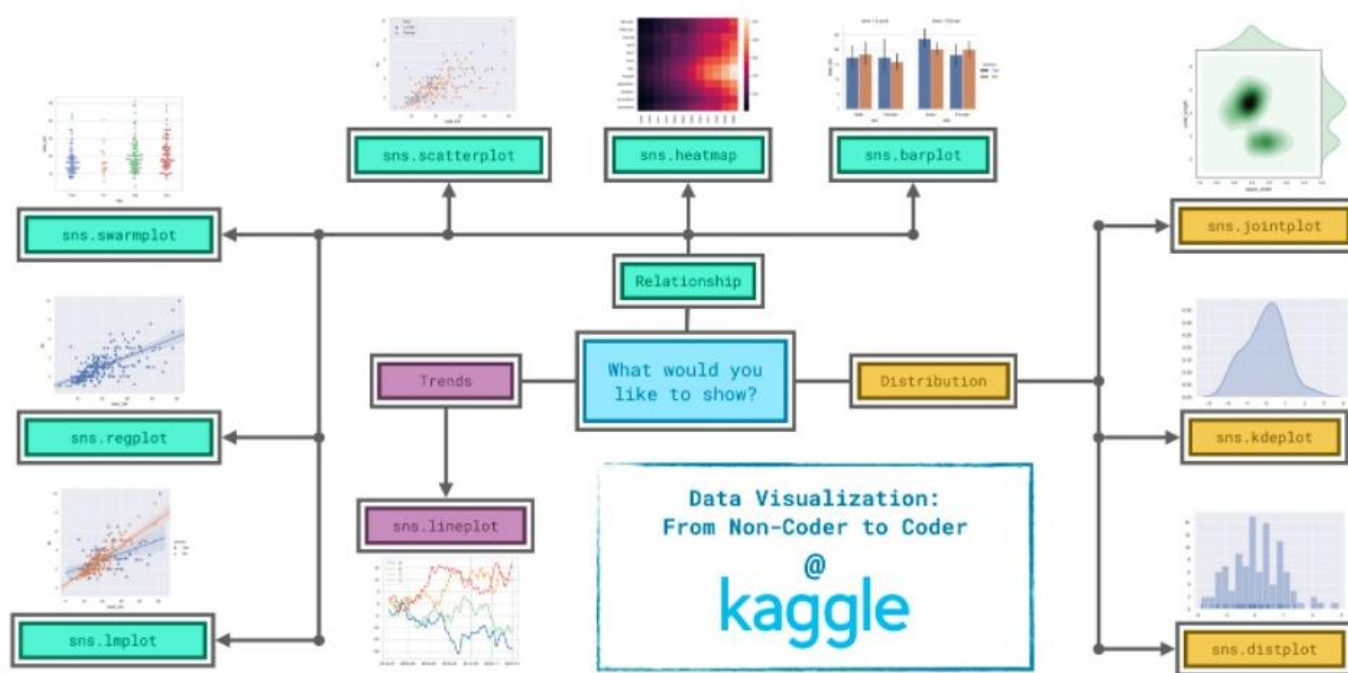
1. assign points to the nearest cluster centroid
2. move each centroid to minimize the distance to its points

It iterates over these two steps until the centroids aren't moving anymore, or until some maximum number of iterations has passed (`max_iter`).

It often happens that the initial random position of the centroids ends in a poor clustering. For this reason the algorithm repeats a number of times (`n_init`) and returns the clustering that has the least total distance between each point and its centroid, the optimal clustering.

## Data Visualization:

Using Seaborn Data Visualization is easier, faster and efficient. There are various Plots which help in data visualization we choose based on our requirement.



Since it's not always easy to decide how to best tell the story behind your data, we've broken the chart types into three broad categories to help with this.

- **Trends** - A trend is defined as a pattern of change.
  - `sns.lineplot` - **Line charts** are best to show trends over a period of time, and multiple lines can be used to show trends in more than one group.

- **Relationship** - There are many different chart types that you can use to understand relationships between variables in your data.
  - `sns.barplot` - **Bar charts** are useful for comparing quantities corresponding to different groups.
  - `sns.heatmap` - **Heatmaps** can be used to find color-coded patterns in tables of numbers.
  - `sns.scatterplot` - **Scatter plots** show the relationship between two continuous variables; if color-coded, we can also show the relationship with a third categorical variable.
  - `sns.regplot` - Including a **regression line** in the scatter plot makes it easier to see any linear relationship between two variables.
  - `sns.lmplot` - This command is useful for drawing multiple regression lines, if the scatter plot contains multiple, color-coded groups.
  - `sns.swarmplot` - **Categorical scatter plots** show the relationship between a continuous variable and a categorical variable.
- **Distribution** - We visualize distributions to show the possible values that we can expect to see in a variable, along with how likely they are.
  - `sns.distplot` - **Histograms** show the distribution of a single numerical variable.
  - `sns.kdeplot` - **KDE plots** (or **2D KDE plots**) show an estimated, smooth distribution of a single numerical variable (or two numerical variables).
  - `sns.jointplot` - This command is useful for simultaneously displaying a 2D KDE plot with the corresponding KDE plots for each individual variable.

# Implementation

Make sure to have the following libraries installed before getting started:

- 1) Pandas
- 2) Numpy
- 3) Matplotlib
- 4) scikit-learn

Once we're done, we can start building the model!

Run the following lines of code to import the necessary libraries.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScale
# if we are using PCA we need to import the following library
from sklearn.decomposition import PCA
# we may need to import OneHot Encoder based on the dataset
```

Now let's import our data.

```
# Importing the data
data_path = "Mall_Customers.csv"
dataset = pd.read_csv(data_path, index_col = 'CustomerID')
dataset.head()
```

```
:      Gender  Age  Annual Income (k$)  Spending Score (1-100)
CustomerID
1      Male   19             15             39
2      Male   21             15             81
3      Female  20             16              6
4      Female  23             16             77
5      Female  31             17             40
```

As the Data is imported, Now let's take a look at our dataset so we could preprocess it before modeling it.



As the data is imported we have a look at our data, it's a crucial step as we get to know what are the steps we need to implement.

```
dataset.tail()
```

	Gender	Age	Annual Income (k\$)	Spending Score (1-100)
<b>CustomerID</b>				
<b>196</b>	Female	35	120	79
<b>197</b>	Female	45	126	28
<b>198</b>	Male	32	126	74
<b>199</b>	Male	32	137	18
<b>200</b>	Male	30	137	83

```
dataset.shape
```

```
(200, 4)
```

```
dataset.describe()
```

	Age	Annual Income (k\$)	Spending Score (1-100)
<b>count</b>	200.000000	200.000000	200.000000
<b>mean</b>	38.850000	60.560000	50.200000
<b>std</b>	13.969007	26.264721	25.823522
<b>min</b>	18.000000	15.000000	1.000000
<b>25%</b>	28.750000	41.500000	34.750000
<b>50%</b>	36.000000	61.500000	50.000000
<b>75%</b>	49.000000	78.000000	73.000000
<b>max</b>	70.000000	137.000000	99.000000



```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 200 entries, 1 to 200
Data columns (total 4 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Gender                200 non-null   object
1   Age                   200 non-null   int64
2   Annual Income (k$)    200 non-null   int64
3   Spending Score (1-100) 200 non-null   int64
dtypes: int64(3), object(1)
memory usage: 7.8+ KB
```

```
dataset.value_counts()
```

```
Gender  Age  Annual Income (k$)  Spending Score (1-100)
Female  18   65                  48                        1
Male    29   28                  82                        1
        24   60                  52                        1
        25   24                  73                        1
        77                  12                        1
..
Female  41   99                  39                        1
        103                 17                        1
        42   34                  17                        1
        43   48                  50                        1
Male    70   49                  55                        1
Length: 200, dtype: int64
```

Let's check for any missing or null values in our dataset.

```
dataset.isnull().sum()
```

```
Gender                0
Age                   0
Annual Income (k$)    0
Spending Score (1-100) 0
dtype: int64
```

There are 4 columns. Index is already set to Customer ID. Three of the columns are of integer data type and Gender Column is of Object Datatype indicating Categorical Data. The dataset has no missing data.

## Pre-Processing Data:

As we are using KMeans Clustering Algorithm, It's best to Scale the data. We are using StandardScaler().

```
from sklearn.preprocessing import StandardScaler
#let's filter out the object datatype before using the scaler

feature_cols = ['Age', 'Annual Income (k$)', 'Spending Score (1-100)']
features = dataset[feature_cols]
scaler = StandardScaler().fit(features.values)
features = scaler.transform(features.values)
scaled_features = pd.DataFrame(features, columns = feature_cols)
scaled_features.head()
```

	Age	Annual Income (k\$)	Spending Score (1-100)
0	-1.424569	-1.738999	-0.434801
1	-1.281035	-1.738999	1.195704
2	-1.352802	-1.700830	-1.715913
3	-1.137502	-1.700830	1.040418
4	-0.563369	-1.662660	-0.395980

As we have a Object datatype column Gender, It's a categorical variable, there are many ways to deal with categorical variable we go ahead with Ordinal Encoding without using built-in Ordinal Encoder function we go ahead with simple pandas replace function.

```
gender = dataset['Gender']
gender.replace('Male', '1', inplace = True)
gender.replace('Female', '0', inplace = True)

gender.head()
```

```
CustomerID
1      1
2      1
3      0
4      0
5      0
Name: Gender, dtype: object
```

As we can see the Data Type is we use pandas inbuilt astype function to convert it to integer datatype

```
gender.astype('int64')
```

```
CustomerID
```

```
1      1
2      1
3      0
4      0
5      0
```

```
..
```

```
196    0
197    0
198    1
199    1
200    1
```

```
Name: Gender, Length: 200, dtype: int64
```

Now we use Standard Scaler for Scaling. Scaling is always useful before using the data for building the model.

```
scaled_features.index = dataset.index
processed_data = scaled_features.join(gender)
processed_data.head()
```

	Age	Annual Income (k\$)	Spending Score (1-100)	Gender
<b>CustomerID</b>				
<b>1</b>	-1.424569	-1.738999	-0.434801	1
<b>2</b>	-1.281035	-1.738999	1.195704	1
<b>3</b>	-1.352802	-1.700830	-1.715913	0
<b>4</b>	-1.137502	-1.700830	1.040418	0
<b>5</b>	-0.563369	-1.662660	-0.395980	0

As the pre-processing is over we shall now proceed with the Model Building Step.

## Model Building:

For applying K-Means we need to first find the optimum number of clusters. So, we use Elbow Method to find the Optimum number of Clusters .

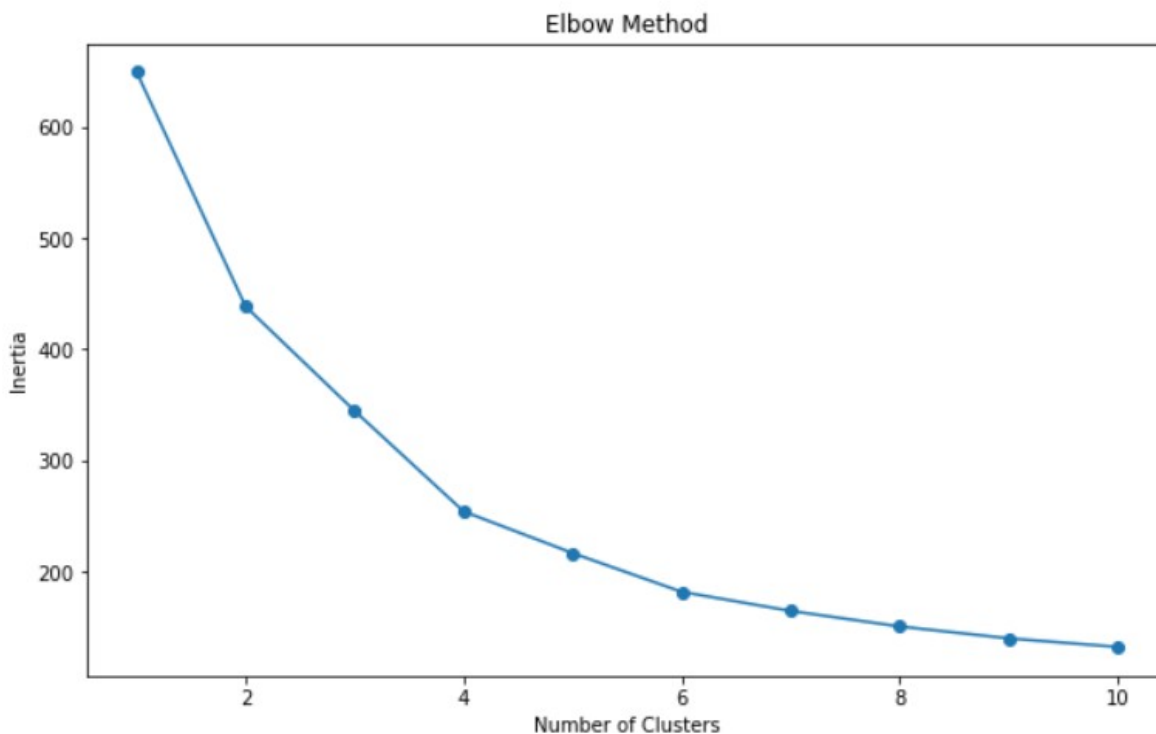
```
from sklearn.cluster import KMeans

SSE = []

for cluster in range(1,11):
    kmeans = KMeans( n_clusters = cluster , init = 'k-means++')
    kmeans.fit(processed_data)
    SSE.append(kmeans.inertia_)

#now let's plot the data for different no of clusters

plt.figure(figsize = (10,6))
plt.plot(np.arange(1,11), SSE , marker = 'o')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.title('Elbow Method')
```



We can see that the Optimum number of Clusters are 4. So, Now let's build the Model.

```
kmeans_model = KMeans(init = 'k-means++', n_clusters = 4, random_state = 0)

y_kmeans = kmeans_model.fit_predict(processed_data)
print(y_kmeans)
```

```
[3 3 3 3 3 3 2 3 2 3 2 3 2 3 2 3 3 2 3 3 3 2 3 2 3 2 3 2 3 2 3 2 3 2
 3 2 3 2 3 2 3 2 3 2 3 3 3 2 3 3 2 2 2 2 2 3 2 2 3 2 2 3 3 2 2 2 2
 2 3 2 2 3 2 2 3 2 2 3 2 2 3 3 2 2 3 2 2 3 3 2 3 2 3 3 2 2 3 2 3 2 2 2 2
 3 1 3 3 3 2 2 2 2 3 1 0 0 1 0 1 0 2 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0
 1 0 1 0 1 0 1 0 1 0 1 0 2 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0]
```

Building the model is over now let's Visualize the Clusters

## Data Visualization:

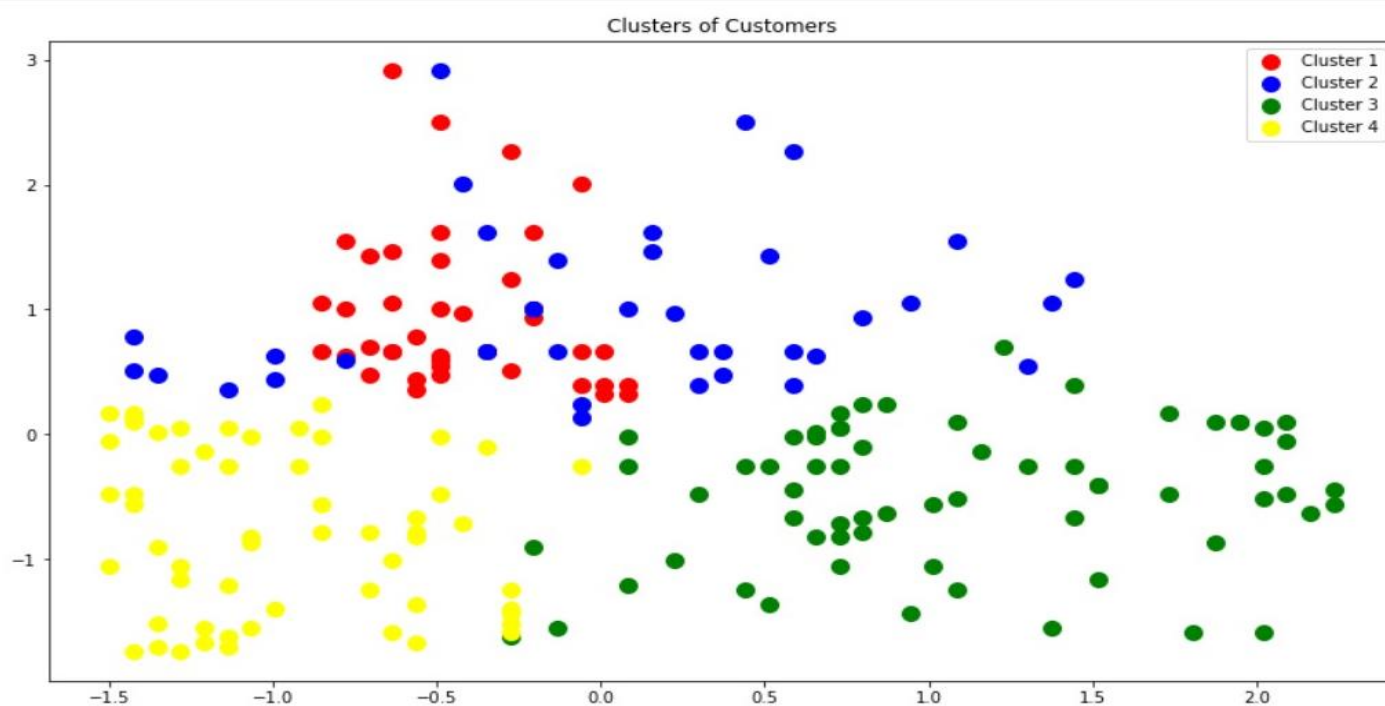
```
plt.figure(figsize = (14,8))

X = processed_data.values

plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c = 'red', label = 'Cluster 1')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c = 'green', label = 'Cluster 3')
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s = 100, c = 'yellow', label = 'Cluster 4')

plt.title('Clusters of Customers')

plt.legend()
plt.show()
```



Let's Visualize the relationship between various parameters and have a proper picture at our data.

Now let's add a Cluster column into our dataset

```
dataset['Cluster'] = y_kmeans
dataset.head()
```

	Gender	Age	Annual Income (k\$)	Spending Score (1-100)	Cluster
CustomerID					
1	1	19	15	39	3
2	1	21	15	81	3
3	0	20	16	6	3
4	0	23	16	77	3
5	0	31	17	40	3

```
grupdf = dataset.groupby('Cluster').mean()
grupdf.head()
```

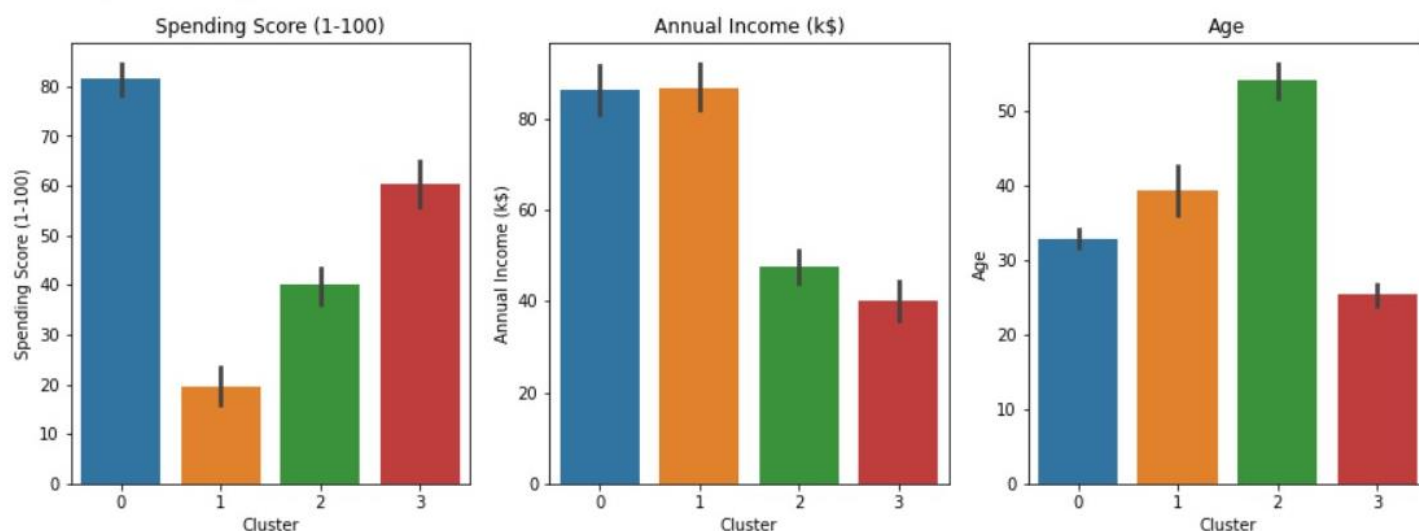
	Age	Annual Income (k\$)	Spending Score (1-100)
Cluster			
0	32.875000	86.100000	81.525000
1	39.368421	86.500000	19.578947
2	53.984615	47.707692	39.969231
3	25.438596	40.000000	60.298246



## Spending Score Vs Annual Income Vs Age

```
fig, ax = plt.subplots(1,3, figsize=(15,5))
sns.barplot(x = dataset.Cluster, y = dataset['Spending Score (1-100)'], label = 'Spending Score (1-100)', ax = ax[0])
ax[0].set_title("Spending Score (1-100)")
sns.barplot(x = dataset.Cluster, y = dataset['Annual Income (k$)'], label = 'Annual Income in (k$)', ax = ax[1])
ax[1].set_title("Annual Income (k$)")
sns.barplot(x = dataset.Cluster, y = dataset['Age'], ax = ax[2])
ax[2].set_title("Age")
```

```
Text(0.5, 1.0, 'Age')
```

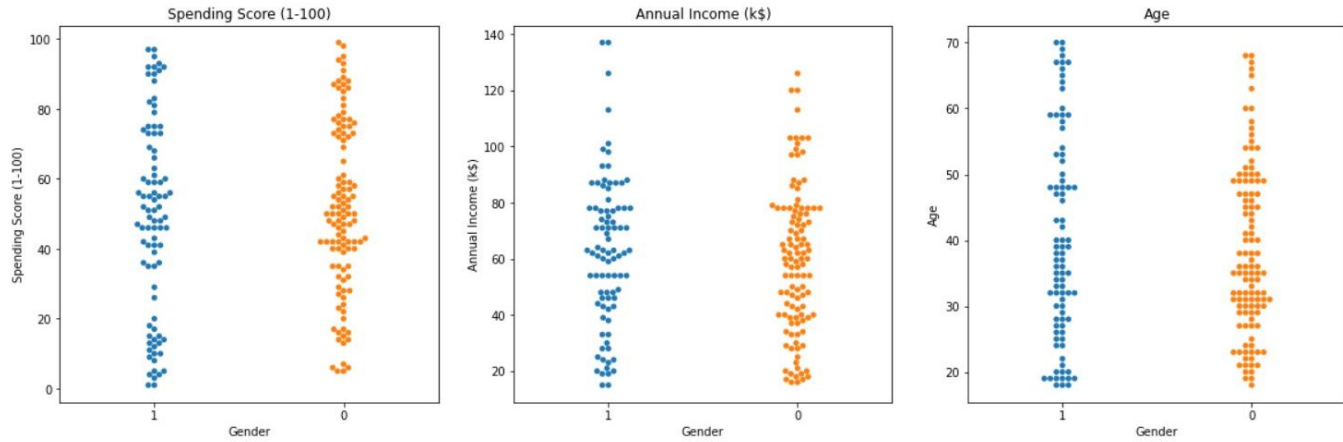


```
dfgender = pd.DataFrame(dataset.groupby(['Cluster', 'Gender']).Gender.count())
dfgender
```

Gender		
Cluster	Gender	
0	0	22
	1	18
1	0	19
	1	19
2	0	37
	1	28
3	0	34
	1	23

```
fig, ax = plt.subplots(1,3, figsize=(20,6))
sns.swarmplot(x=dataset['Gender'], y = dataset['Spending Score (1-100)'], label = 'Spending Score (1-100)', ax = ax[0])
ax[0].set_title("Spending Score (1-100)")
sns.swarmplot(x=dataset['Gender'], y = dataset['Annual Income (k$)'], label = 'Annual Income in (k$)', ax = ax[1])
ax[1].set_title("Annual Income (k$)")
sns.swarmplot(x=dataset['Gender'], y = dataset['Age'], ax = ax[2])
ax[2].set_title("Age")
```

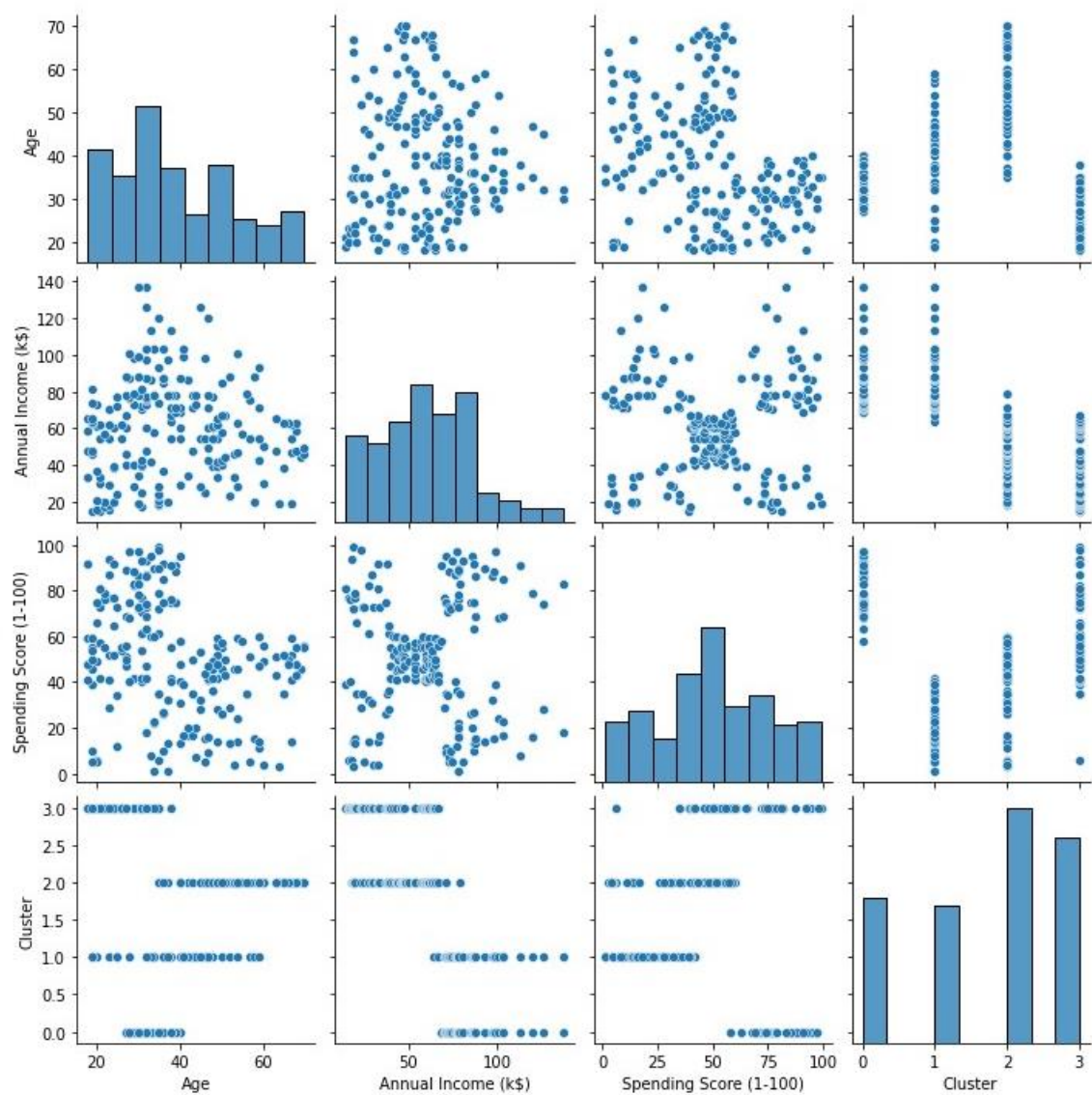
Text(0.5, 1.0, 'Age')





```
sns.pairplot(data = dataset)
```

```
<seaborn.axisgrid.PairGrid at 0x249c8b2dfd0>
```



# Cluster Analysis

## **Main attributes of each segment:**

### Cluster 0:

- High average annual income, High spending.
- Mean age is around 35 and gender is predominantly Female.

### Cluster 1:

- High average annual income, Low spending.
- Mean age is around 40 and gender is equally male and female.

### Cluster 2:

- Low average income, Average spending score.
- Mean age is around 55 and gender is predominantly female.

### Cluster 3:

- Low average income, Average spending score.
- Mean age is around 25 and gender is predominantly female.

## Building personas around each cluster:

Now that we know the attributes of each cluster, we can build personas around them.

Being able to tell a story around your analysis is an important skill to have as a data scientist.

This will help our clients or stakeholders understand your findings more easily.

Here is an example of building consumer personas based on the clusters created:

### **Cluster 0: Highly affluent individuals**

This persona comprises of middle aged individuals spend more money. They are the profit segment.

Having the highest average income compared to individuals in all other clusters, they spend the most. These are individuals who have worked hard to build up a significant amount of wealth.

They also spend large amounts of money to live a good lifestyle.

These individuals have likely just started a family, and are leading baby or family-focused lifestyles. It is a good idea to promote baby or child related products to these individuals.

Recommendation: Due to their large spending capacity and their demographic, these individuals are likely to be looking for properties to buy or invest in. They are also more likely than all other segments to take out housing loans and make serious financial commitments.

### **Cluster 1: Careful buyer**

This segment comprises of middle group of people.

Despite having the highest average income compared to individuals in all other clusters, they spend the least.

Recommendation: Promos, coupons, and discount codes will attract individuals in this segment due to their tendency to spend less.

### **Cluster 2: Almost retired**

This segment comprises of an older group of people.

They earn less and spend less, and are probably saving up for retirement.

Recommendation: Marketing to these individuals can be done through Facebook, which appeals to an older demographic. Promote healthcare related products to people in this segment.

### **Cluster 3: The careless buyer**

This segment is made up of a younger age group.

Individuals in this segment are most likely first jobbers. They make the least amount of money compared to all other segments.

However, they are very high spenders.

These are enthusiastic young individuals who enjoy living a good lifestyle, and tend to spend above their means.

Recommendation: Since these are young individuals who spend a lot, providing them with travel coupons or hotel discounts might be a good idea. Providing them with discounts off top clothing and makeup brands would also work well for this segment.

## Conclusion

In this work, we approached the problem of customer market segmentation, on demand model for many cooperate companies to analyze their customer market and cluster them. Working on clusters of customers is lot easier than working on whole market, it boosts the company's growth. We proposed the solution based on the Unsupervised Learning Methods, with the help of K-Means Clustering Algorithm.

We have successfully built a K-Means clustering model for customer segmentation. We also explored cluster interpretation, and analyzed the behavior of individuals in each cluster.

Finally, we took a look at some business recommendations that could be provided based on the attributes of each individual in the cluster.