**Project title:**   customer segmentation using Data Science

**Phase 3:** Development

**Part 1**

In this part we will begin building your project by loading and preprocessing the dataset.

Begin the customer segmentation project by loading and preprocessing the customer data.

Collect and preprocess the customer data for analysis.

**Data Preprocessing:**

* Data preprocessing is a crucial step within the statistics analysis and gadget gaining knowledge of pipeline.
* It includes a sequence of strategies and operations finished on uncooked statistics to clean, organize, and transform it right into a layout that is suitable for analysis or device mastering version schooling.
* Data preprocessing goals to enhance the first-class of the records, making it greater reliable and conducive to generating accurate consequences.

Here are some common tasks and techniques involved in data preprocessing:

**Data Cleaning:**

* Handling missing values: Deciding how to deal with missing data, whether by imputing values or removing incomplete records.
* Outlier detection and treatment: Identifying and handling data points that significantly deviate from the norm.

**Noise reduction:**

* Smoothing noisy data through techniques like filtering.

**Data Transformation:**

* **Data normalization:** Scaling numerical features to a standard range (e.g., between 0 and 1) to ensure that they have similar influence in the analysis.
* **Encoding categorical variables:** Converting categorical data into numerical format, such as one-hot encoding or label encoding.
* **Feature engineering:** Creating new features or modifying existing ones to capture more meaningful information from the data.
* **Dimensionality reduction:** Reducing the number of features while retaining essential information, using methods like Principal Component Analysis (PCA).

**Data Integration:**

* **Merging or joining datasets:** Combining data from multiple sources into a single dataset for analysis.

**Aggregation:** Summarizing data at a higher level of granularity, such as aggregating daily sales into monthly totals.

**Data Reduction:**

* **Sampling:** Reducing the size of a large dataset by randomly selecting a representative subset.
* **Binning:** Grouping continuous data into discrete bins to simplify analysis.
* **Filtering:** Selecting a subset of data based on specific criteria.

**Data Standardization:**

* Ensuring that data follows a consistent format and structure.
* Date and time format conversion: Converting date and time data into a uniform format.
* Currency conversion: Converting monetary values into a common currency.

**Data Scaling:**

* Scaling numerical data to a common range to prevent some features from dominating the analysis.

Data preprocessing is an iterative process that may involve several of these steps in various orders, depending on the specific dataset and the analysis goals. Proper data preprocessing is essential for improving the accuracy and effectiveness of machine learning models, as well as for making data more accessible for traditional statistical analysis.

Here is the data preprocessing codes along with the output of the given dataset:

**Importing the libraries:**

Import three basic libraries which are very common in machine learning and will be used every time you train a model

* **NumPy:** it is a library that allows us to work with arrays and as most machine learning models work on arrays NumPy makes it easier
* **matplotlib:** this library helps in plotting graphs and charts, which are very useful while showing the result of your model
* **Pandas:** pandas allows us to import our dataset and also creates a matrix of features containing the dependent and independent variable.

**Code:**

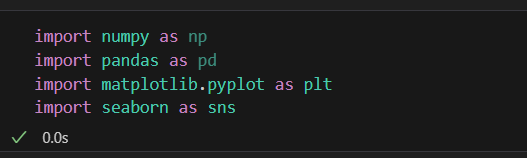
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

**Output:**



**Load the dataset:**

* Data sets are available in .csv format. A CSV file stores tabular data in plain text.
* Each line of the file is a data record. We use the read\_csv method of the pandas library to read a local CSV file as a dataframe.
* Load our customer data from the CSV file

**Code:**

# Try reading the file with different encodings

encodings = ['utf-8', 'latin1', 'ISO-8859-1']

for encoding in encodings:

try:

dataset = pd.read\_csv(r'C:\\Users\\ADMIN\Downloads\\archive\\Mall\_Customers.csv', encoding=encoding)

print(f"Successfully read with encoding: {encoding}")

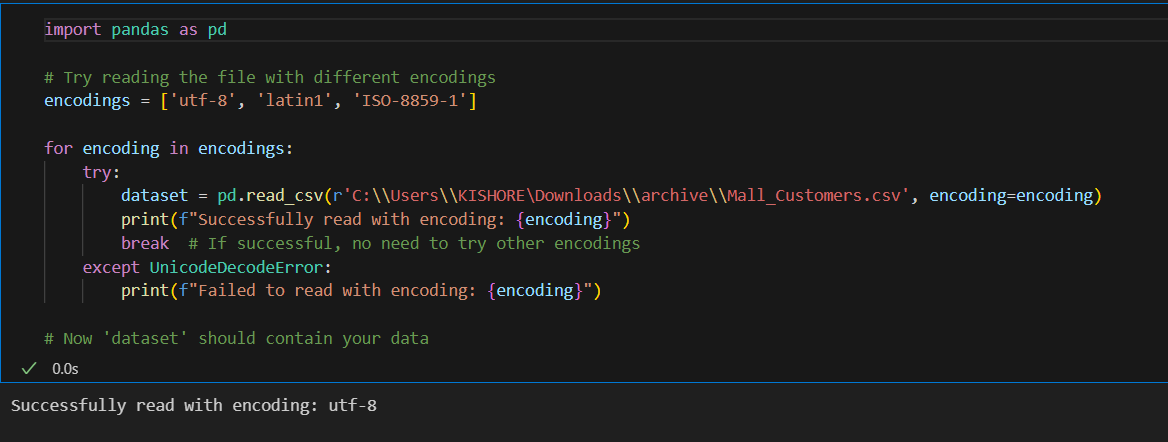
break  # If successful, no need to try other encodings

except UnicodeDecodeError:

print(f"Failed to read with encoding: {encoding}")

# Now 'dataset' should contain your data

**Output:**

****

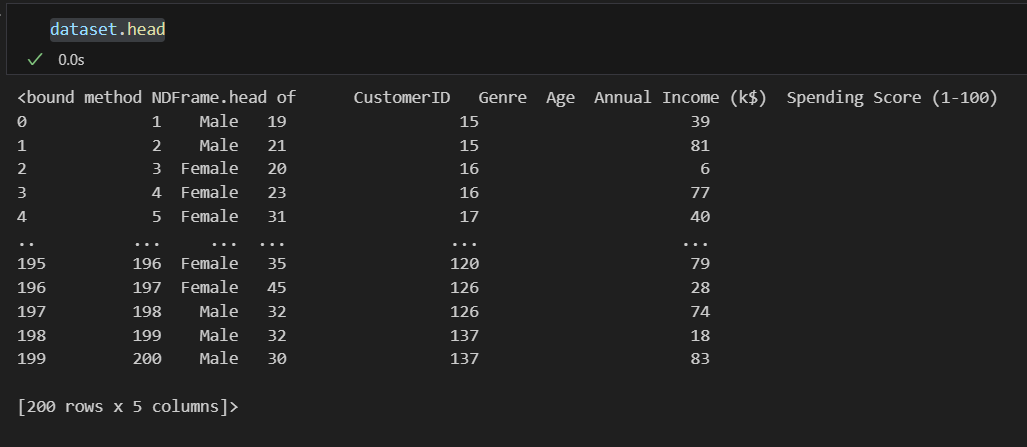
**Head() Function:**

* The head() function is used to get the first n rows.
* This function returns the first n rows for the object based on position.
* It is useful for quickly testing if your object has the right type of data in it.
* If the value of the n is not assigned it returns a default value of first 5 rows

**Code:**

dataset.head()

**Output:**



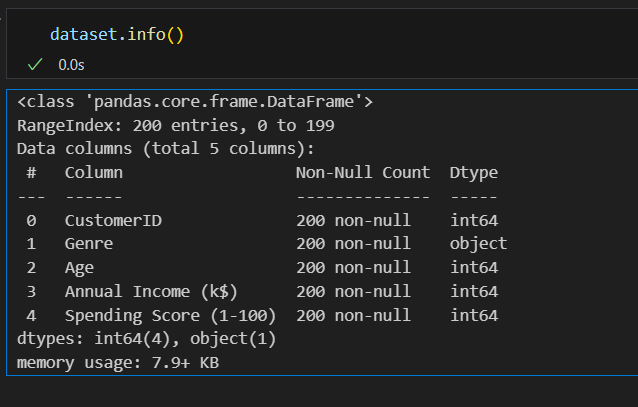
**Info() Function:**

* The info() method prints information about the DataFrame.
* The information contains the number of columns, column labels, column data types, memory usage, range index, and the number of cells in each column (non-nullvalues).

**Code:**

dataset.info()

**Output:**



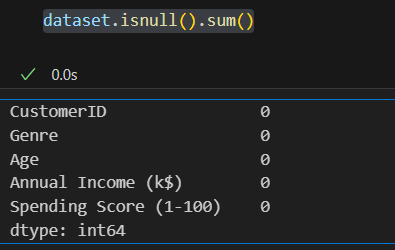
**Df.isnull().sum() Function:**

* This code is used to count the number of missing (null) values in each column of a DataFrame, denoted as df.
* It returns a summary of the missing data for each column, showing how many missing values are there in each column.
* This information is essential in data preprocessing and analysis to identify and handle missing data appropriately.Top of Form

**Code:**

dataset.isnull().sum()

**Output:**



**Describe Function:**

* The describe() function in pandas, a popular Python data analysis library, is used to generate summary statistics of a DataFrame or Series.

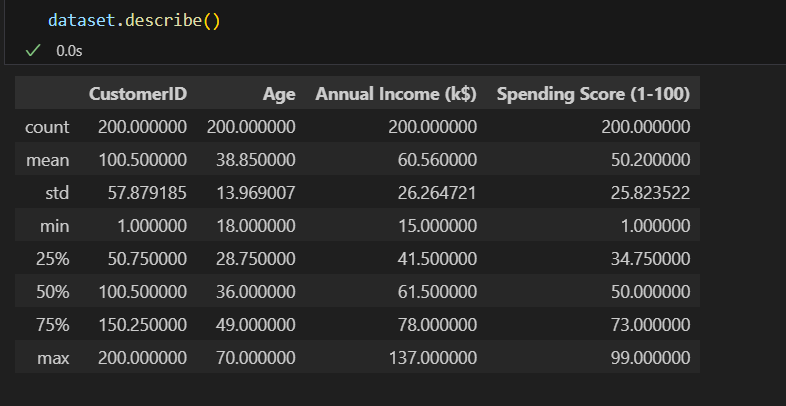
It provides a quick overview of the key statistics for numerical data in the dataset, including:

* **Count:** The number of non-null values.
* **Mean:** The average of the values.
* **Standard Deviation (std):** A measure of the spread or dispersion of the data.
* **Minimum:** The minimum value in the dataset.
* **25th Percentile (25%):** The value below which 25% of the data falls (the first quartile).
* **Median (50% or the 2nd quartile):** The middle value when the data is sorted.
* **75th Percentile (75%):** The value below which 75% of the data falls (the third quartile).
* **Maximum:** The maximum value in the dataset.

**Code:**

dataset.describe()

**Output:**

****

**Outliers:**

* Outliers are data points that significantly deviate from the rest of the data in a dataset.
* They can be exceptionally high or low values compared to the majority of the data.

**Code:**

import matplotlib.pyplot as plt

# Ensure your dataset contains only numerical data for box plotting

numerical\_data = dataset.select\_dtypes(include='number')

# Transpose the data to prepare for box plotting

data\_to\_plot = numerical\_data.values.T

# Create subplots

fig, axs = plt.subplots(9, 1, dpi=95, figsize=(7, 17))

# Iterate through columns and create boxplots

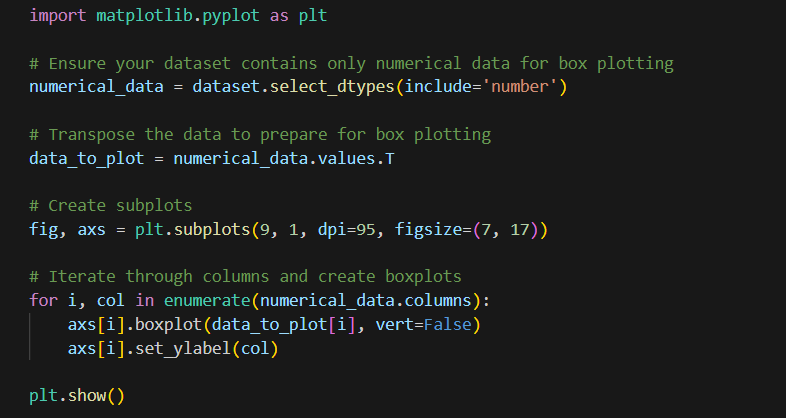
for i, col in enumerate(numerical\_data.columns):

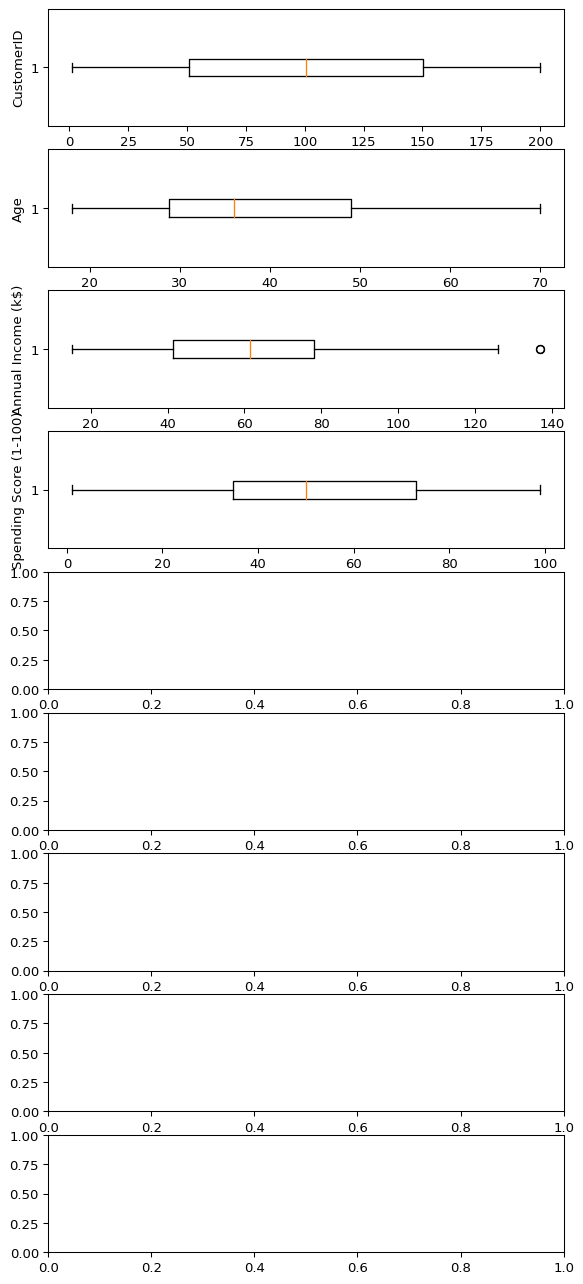
    axs[i].boxplot(data\_to\_plot[i], vert=False)

    axs[i].set\_ylabel(col)

plt.show()

**Output:**

****

****

**Corelation:**

* Correlation is a statistical measure that indicates the extent to which two or more variables fluctuate in relation to each other.
* Correlation describes the relationship between variables. It can be described as either strong or weak, and as either positive or negative.

**Code:**

numeric\_dataset = dataset.select\_dtypes(include=['number'])

corr = numeric\_dataset.corr()

import matplotlib.pyplot as plt

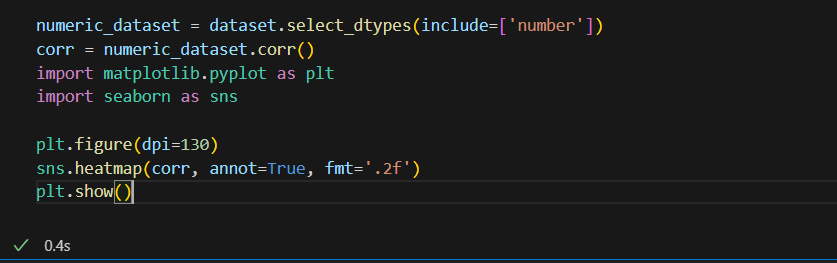
import seaborn as sns

plt.figure(dpi=130)

sns.heatmap(corr, annot=True, fmt='.2f')

plt.show()

**Output:**

****



**Normalization**

* MinMaxScaler scales the data so that each feature is in the range [0, 1].
* It works well when the features have different scales and the algorithm being used is sensitive to the scale of the features, such as k-nearest neighbors or neural networks.
* Rescale your data using scikit-learn using the MinMaxScalar.

**Code:**

from sklearn.preprocessing import MinMaxScaler

from sklearn.preprocessing import OneHotEncoder

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

numeric\_cols = dataset.select\_dtypes(include=['number']).columns

categorical\_cols = dataset.select\_dtypes(exclude=['number']).columns

numeric\_transformer = Pipeline(steps=[

('scaler', MinMaxScaler(feature\_range=(0, 1)))])

categorical\_transformer = Pipeline(steps=[

('encoder', OneHotEncoder())])

preprocessor = ColumnTransformer(

transformers=[

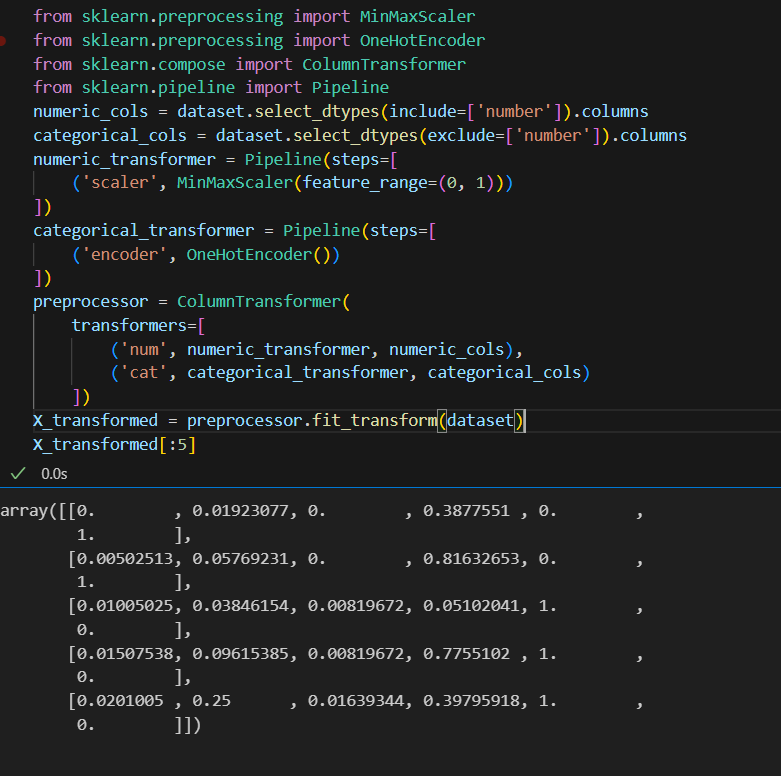
('num', numeric\_transformer, numeric\_cols),

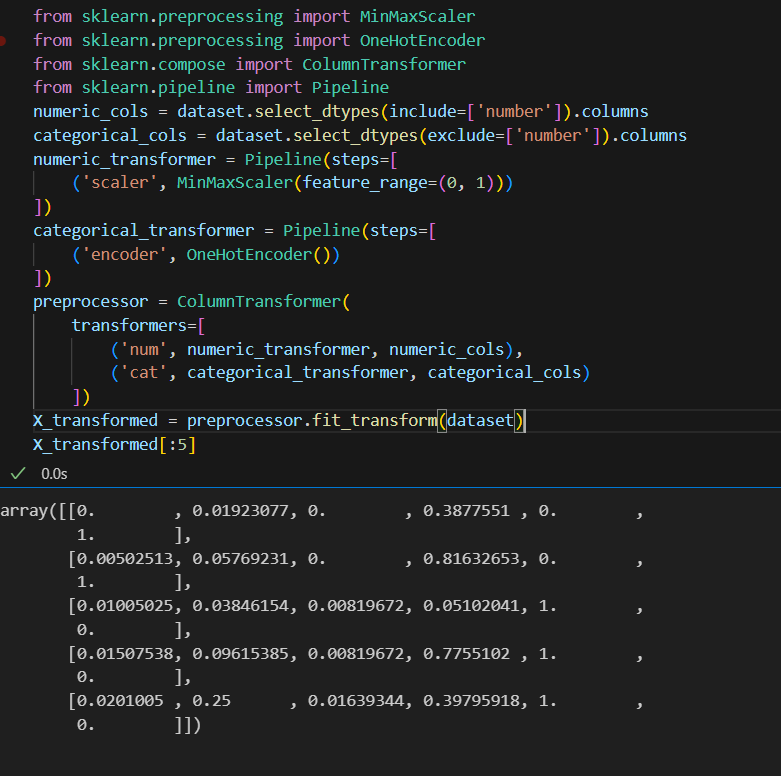
('cat', categorical\_transformer, categorical\_cols)])

X\_transformed = preprocessor.fit\_transform(dataset)

X\_transformed[:5]

**Output:**



****

**Standardization**

* Standardization is a useful technique to transform attributes with a Gaussian distribution and differing means and standard deviations to a standard Gaussian distribution with a mean of 0 and a standard deviation of 1.
* We can standardize data using scikit-learn with the StandardScalar class.
* It works well when the features have a normal distribution or when the algorithm being used is not sensitive to the scale of the features

**Code:**

from sklearn.preprocessing import StandardScaler

from sklearn.preprocessing import OneHotEncoder

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

numeric\_cols = dataset.select\_dtypes(include=['number']).columns

categorical\_cols = dataset.select\_dtypes(exclude=['number']).columns

numeric\_transformer = Pipeline(steps=[('scaler', StandardScaler())])

categorical\_transformer = Pipeline(steps=[('encoder', OneHotEncoder())])

preprocessor = ColumnTransformer(

transformers=[

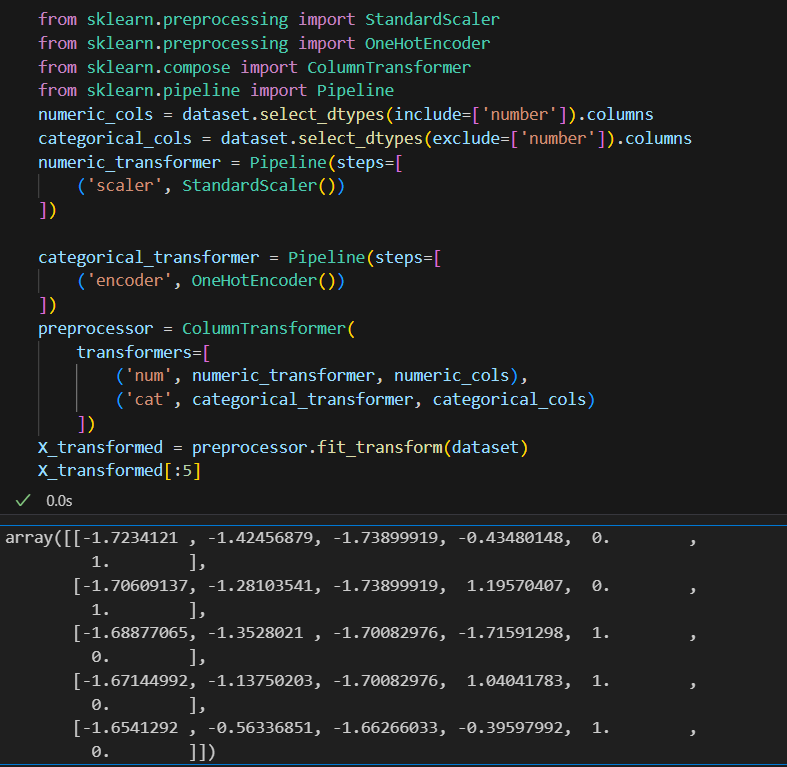
('num', numeric\_transformer, numeric\_cols),

('cat', categorical\_transformer, categorical\_cols])

X\_transformed = preprocessor.fit\_transform(dataset)

X\_transformed[:5]

**Output:**

****

**K-means Clustering Function:**

K-means clustering is a machine learning and data analysis technique used for grouping data points into clusters based on their similarity. It's primarily used for:

* **Unsupervised Learning:** K-means helps identify patterns or structure in data without labelled categories.
* **Segmentation:** It can segment data into distinct groups, making it useful for customer segmentation, image compression, and more.
* **Pattern Recognition:** It's used in pattern recognition tasks, such as image analysis and natural language processing.
* **Anomaly Detection:** It can identify outliers by placing data points that don't fit well into any cluster.
* **Data Compression:** K-means can reduce the dimensionality of data while preserving important information.
* **Recommendation Systems:** It can be applied to recommend items or services based on user preferences.

**Code:**

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="cyan",label = "cluster 4")

plt.scatter(x[y\_kmeans==4,0],x[y\_kmeans==4,1],s=100,c="magenta",label = "cluster 5")

plt.scatter(kmeans.cluster\_centers\_[:,0],kmeans.cluster\_centers\_[:,1],s=300,c="yellow",label="centroids")

plt.title("clusters of customers")

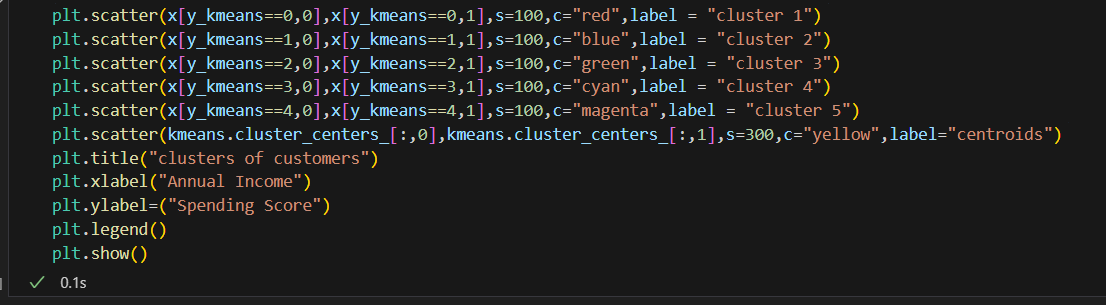
plt.xlabel("Annual Income")

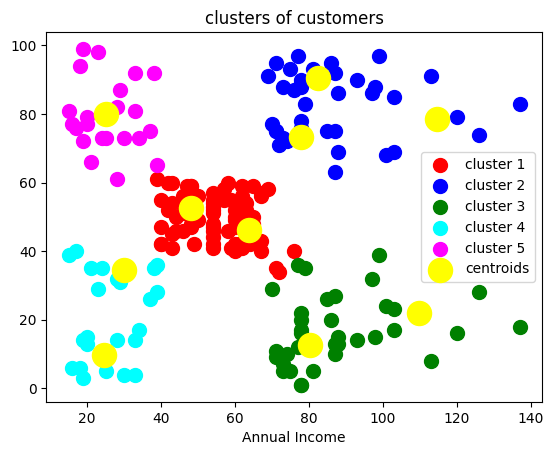
plt.ylabel=("Spending Score")

plt.legend()

plt.show()

**Output:**





**WCSS Function:**

* WCSS is the sum of the squared distance between each point and the centroid in a cluster.
* When we plot the WCSS with the K value, the plot looks like an Elbow.
* As the number of clusters increases, the WCSS value will start to decrease.

**Code:**

plt.plot(range(1,11),wcss)

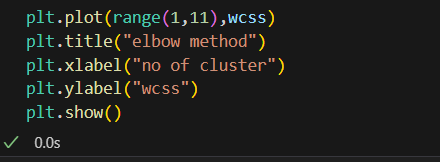
plt.title("elbow method")

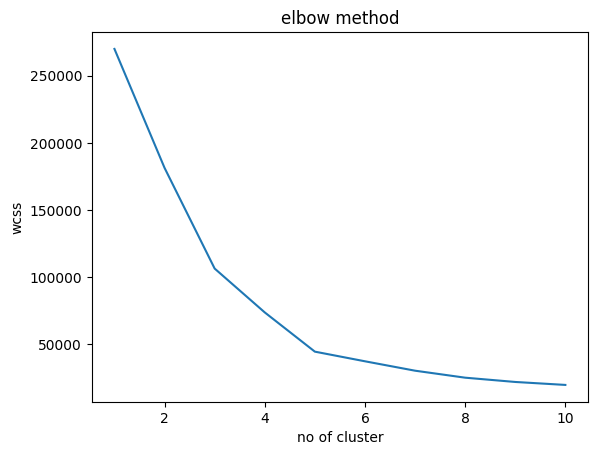
plt.xlabel("no of cluster")

plt.ylabel("wcss")

plt.show()

**Output:**

****



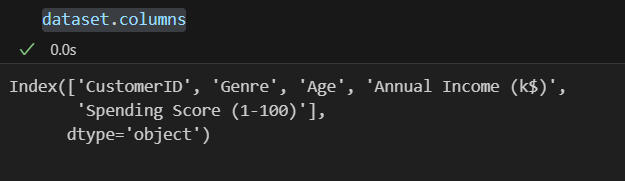
**Dataset.columns :**

* We can use the loc and iloc functions to access columns in a Pandas DataFrame.
* for example the Grades column, we could simply use the loc function and specify the name of the column in order to retrieve it.

**Code:**

dataset.columns

**Output:**

****

**Memory Function:**

* Pandas**dataframe.memory\_usage()** function return the memory usage of each column in bytes.
* The memory usage can optionally include the contribution of the index and elements of object dtype. This value is displayed in DataFrame.info by default.

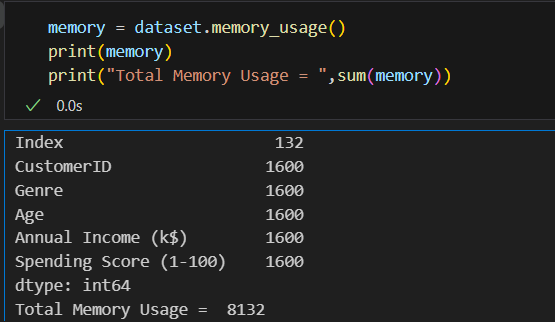
**Code:**

memory = dataset.memory\_usage()

print(memory)

print("Total Memory Usage = ",sum(memory))

**Output:**

****

**Dropna() Function:**

* dropna() is a function used in data preprocessing, often in the context of data analysis and cleaning, to remove or drop rows or columns with missing (NaN or null) values from a dataset.
* It's a method to eliminate incomplete or unreliable data from your dataset, which can be important to ensure the quality of your analysis or machine learning models

**Code:**

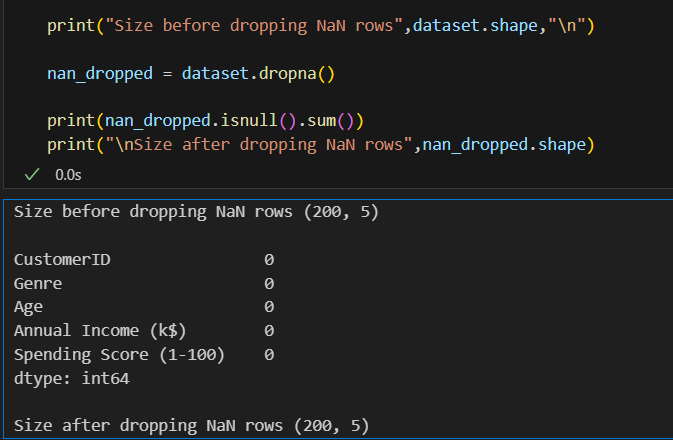
print("Size before dropping NaN rows",dataset.shape,"\n")

nan\_dropped = dataset.dropna()

print(nan\_dropped.isnull().sum())

print("\nSize after dropping NaN rows",nan\_dropped.shape)

**Output:**



**Iloc() Function:**

The iloc() function is a method in pandas, a popular Python library for data manipulation and analysis. It is primarily used to select and access data in a DataFrame by integer-based indexing.

* Select specific rows and columns from a DataFrame using integer-based indexing.
* Provide a way to slice and filter data by row and column positions.

**Code:**

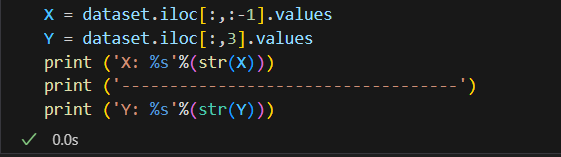
X = dataset.iloc[:,:-1].values

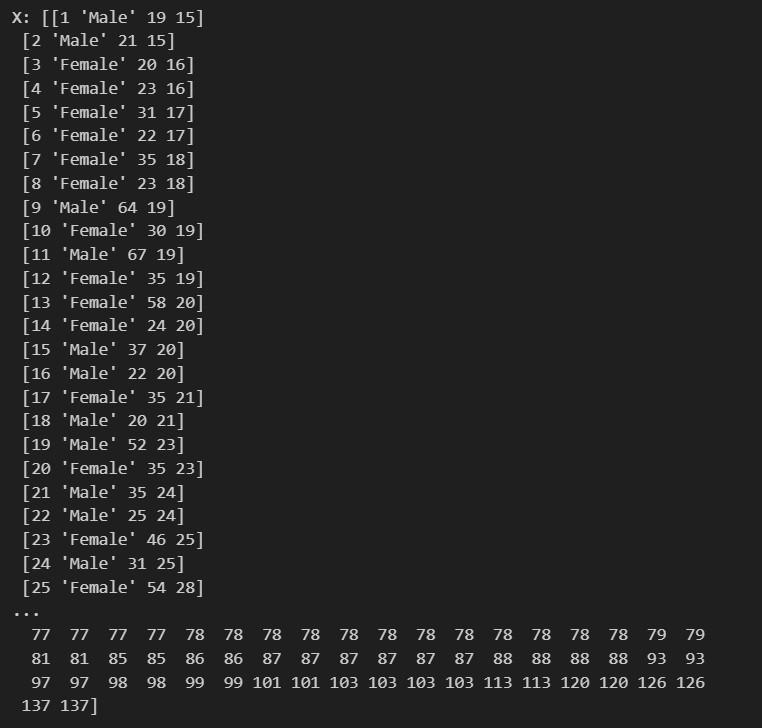
Y = dataset.iloc[:,3].values

print ('X: %s'%(str(X)))

print ('-----------------------------------')

print ('Y: %s'%(str(Y)))

**Output:**



**Subplots Function:**

* Subplots are a feature in data visualization that allow you to create multiple smaller plots within a larger figure.
* They are useful for displaying multiple related visualizations side by side, making it easier to compare and analyze data.
* subplots help you arrange and present multiple charts, graphs, or plots in a single figure, improving the overall clarity and readability of your data visualizations.

**Code:**

fig, axs = plt.subplots()

sns.boxplot(data=dataset,orient='h',palette="Set2")

plt.show()

**Output:**



**Missingno Function:**

* "missingno" is a Python library used for visualizing and analyzing missing data in a dataset.
* It provides various visualization tools to quickly understand and identify missing values in your data, allowing you to make informed decisions on how to handle or impute missing data.

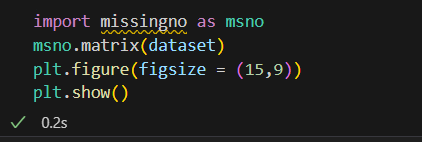
Code:

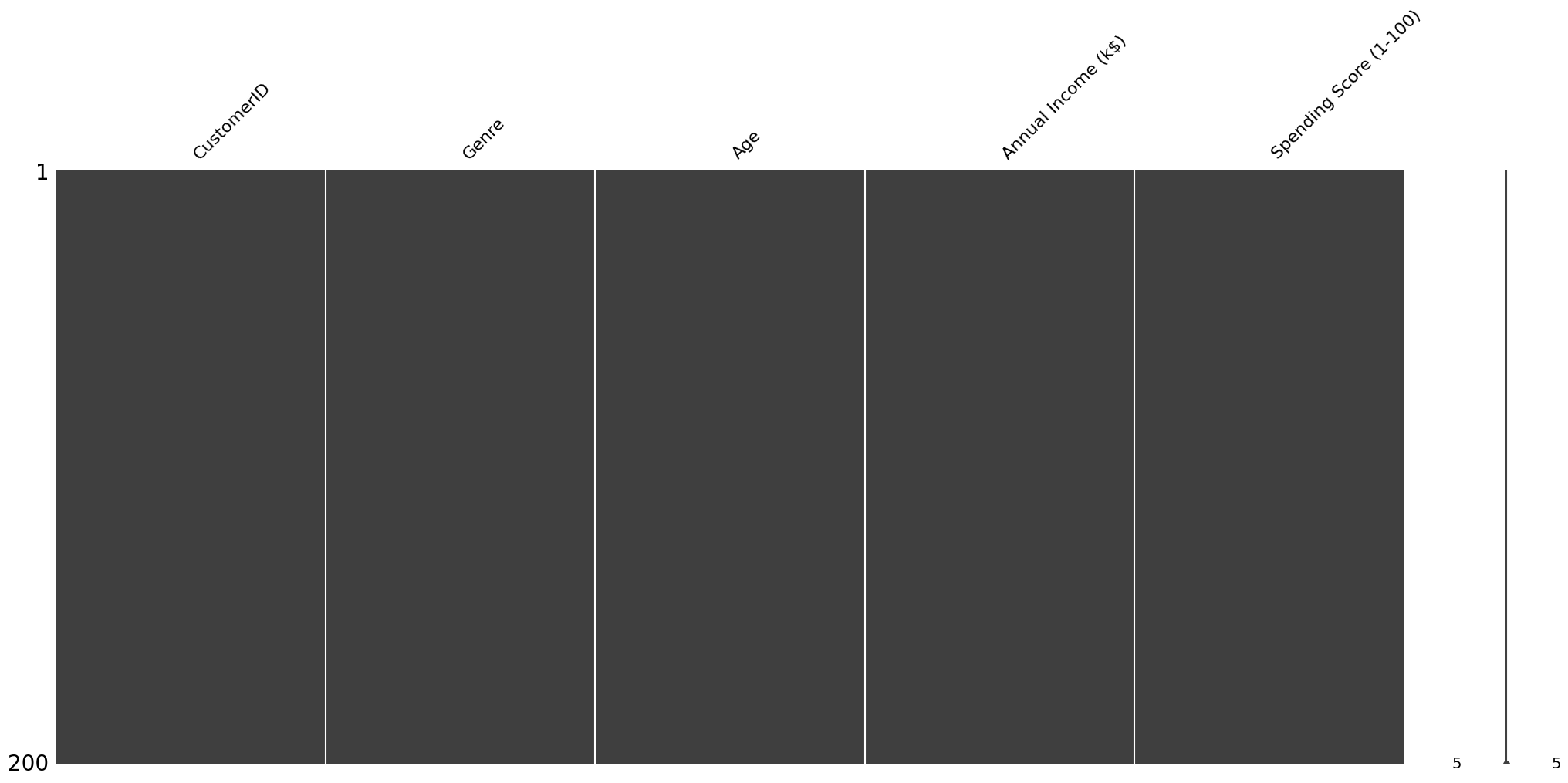
import missingno as msno

msno.matrix(dataset)

plt.figure(figsize = (15,9))

plt.show()

Output:

****

**Code:**

msno.heatmap(dataset, labels = True)

Output:

