20VV1A1263 DATA SCIENCE LABORATORY IV-I B.Tech IT

Exp1 : python program to find sum of series 1+1/2+1/3 + ..+1/N

```
def sum_of_series(n):
    result = 0.0
    for i in range(1, n + 1):
        result += 1 / i
    return result

# Get the value of N from the user
N = int(input("Enter the value of N: "))

# Calculate the sum of the series
series_sum = sum_of_series(N)

# Display the result
print(f"The sum of the series 1 + 1/2 + 1/3 + ... + 1/{N} is: {series_sum}")
```

```
Enter the value of N: 10
The sum of the series 1 + 1/2 + 1/3 + ... + 1/10 is: 2.9289682539682538
```

Exp2: write a python program to split the array and add the first part to the end

```
def split_and_add(arr, k):
    # Check if the array length is divisible by k
    if len(arr) % k != 0:
        print("Array length is not divisible by k. Cannot perform the operation.")
        return

    # Split the array into two parts
    first_part = arr[:k]
    second_part = arr[k:]

    # Add the first part to the end of the array
    result_array = second_part + first_part

    return result_array

# Example usage
input_array = list(map(int,input('Enter Array').split(',')))
k_value = int(input("Enter split index:"))

result = split_and_add(input_array, k_value)

print(f"Original Array: {input_array}")
print(f"Array after splitting and adding the first part to the end: {result}")
```

```
Enter Array1,2,3,4,5,6,7,8,9,0
Enter split index:5
Original Array: [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
Array after splitting and adding the first part to the end: [6, 7, 8, 9, 0, 1, 2, 3, 4, 5]
```

Exp3: write a python program to create a list of tuples with the first element as the number and the second one as square of it

```
def create_tuples(n):
    # Use a list comprehension to create the list of tuples
    result_list = [(i, i**2) for i in range(1, n+1)]
    return result_list

# Example usage
n_value = 5
tuples_list = create_tuples(n_value)

print(f"List of Tuples with Numbers and Their Squares:")
print(tuples_list)
```

```
List of Tuples with Numbers and Their Squares:
[(1, 1), (2, 4), (3, 9), (4, 16), (5, 25)]
```

Exp4: write a python program to count number of vowels using sets in a given string

```
def count_vowels(input_string):
    # Define a set of vowels
    vowels = set("aeiouAEIOU")

    # Use a set intersection to find common elements (vowels) between the string and the vowel set
    vowel_count = len(set(input_string) & vowels)

    return vowel_count

# Example usage
input_str = input("Enter a string: ")
result = count_vowels(input_str)

print(f"Number of vowels in the given string: {result}")
```

```
Enter a string: hello how are you!
Number of vowels in the given string: 4
```

Exp5: Write a program to implement permutation of a given string using innbuilt function

```
from itertools import permutations

def generate_permutations(input_string):
    # Use the permutations function to generate all permutations
    permuted_strings = permutations(input_string)

    # Convert each permutation to a string and store in a list
    result_list = [''.join(permutation) for permutation in permuted_strings]

    return result_list

# Example usage
input_str = input("Enter a string: ")
permutations_list = generate_permutations(input_str)

print(f"Permutations of the given string '{input_str}':")
for permuted_str in permutations_list:
    print(permuted_str)
```

```
Enter a string: hello
Permutations of the given string 'hello':
hello
helol
hello
helol
heoll
heoll
hlelo
hleol
hlleo
hlloe
hloel
hlole
hlelo
hleol
hlleo
hlloe
hloel
hlole
hoell
hoell
holel
holle
holel
holle
ehllo
ehlol
ehllo
ehlol
eholl
eholl
elhlo
elhol
ellho
elloh
elohl
elolh
elhlo
elhol
ellho
elloh
elohl
elolh
```

```
eohll
eohll
eolhl
eollh
eolhl
eollh
lhelo
lheol
lhleo
lhloe
lhoel
lhole
lehlo
lehol
```

Exp6: write a python program to sort list of dictionaries by values in python using lambda function

```
# List of dictionaries
data = [
    {'name': 'Alice', 'age': 30, 'salary': 50000},
    {'name': 'Bob', 'age': 25, 'salary': 60000},
    {'name': 'Charlie', 'age': 35, 'salary': 45000}
]

# Sort the list of dictionaries by the 'age' key using a lambda function
sorted_data_by_age = sorted(data, key=lambda x: x['age'])

# Display the sorted list
print("Sorted by age:")
print(sorted_data_by_age)

# Sort the list of dictionaries by the 'salary' key using a lambda function
sorted_data_by_salary = sorted(data, key=lambda x: x['salary'])

# Display the sorted list
print("\nSorted by salary:")
print(sorted_data_by_salary)
```

```
Sorted by age:
[{'name': 'Bob', 'age': 25, 'salary': 60000}, {'name': 'Alice', 'age': 30, 'salary': 50000}, {'name': 'Charlie', 'age':

Sorted by salary:
[{'name': 'Charlie', 'age': 35, 'salary': 45000}, {'name': 'Alice', 'age': 30, 'salary': 50000}, {'name': 'Bob', 'age':
```

Exp7: write a python program for following sorting: 1. Quick Sort 2. Heap Sort

```
# Quick Sort
def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2]
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    return quick_sort(left) + middle + quick_sort(right)

# Heap Sort
def heapify(arr, n, i):
    largest = i
    left = 2 * i + 1
    right = 2 * i + 2

    if left < n and arr[i] < arr[left]:
        largest = left

    if right < n and arr[largest] < arr[right]:
        largest = right

    if largest != i:
        arr[i], arr[largest] = arr[largest], arr[i]
        heapify(arr, n, largest)

def heap_sort(arr):
    n = len(arr)

    for i in range(n // 2 - 1, -1, -1):
        heapify(arr, n, i)

    for i in range(n - 1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i]
        heapify(arr, i, 0)

# Example usage
input_array = [64, 34, 25, 12, 22, 11, 90]

# Quick Sort
sorted_array_quick_sort = quick_sort(input_array.copy())
print("Sorted array using Quick Sort:", sorted_array_quick_sort)

# Heap Sort
input_array_heap_sort = input_array.copy()
heap_sort(input_array_heap_sort)
print("Sorted array using Heap Sort:", input_array_heap_sort)
```

```
Sorted array using Quick Sort: [11, 12, 22, 25, 34, 64, 90]
Sorted array using Heap Sort: [11, 12, 22, 25, 34, 64, 90]
```

Exp8: write a python program to reverse a string using recursion ChatGPT

```
def reverse_string_recursive(input_str):
    # Base case: if the string is empty or has only one character, it is already reversed
    if len(input_str) <= 1:
        return input_str
    # Recursive case: reverse the substring excluding the first character, and append the first character at the end
    else:
        return reverse_string_recursive(input_str[1:]) + input_str[0]

# Example usage
input_string = "Hello, World!"

# Reverse the string using recursion
result_recursive = reverse_string_recursive(input_string)

print("Original string:", input_string)
print("Reversed string using recursion:", result_recursive)
```

```
Original string: Hello, World!
Reversed string using recursion: !dlroW ,olleH
```

Exp 9: write a python program to count no of number in a text file

```
def count_words_in_file(file_path):
    try:
        with open(file_path, 'r') as file:
            content = file.read()
            # Count the number of words in the file content
            word_count = len(content.split())
            return word_count
    except FileNotFoundError:
        print(f"File not found: {file_path}")
    except Exception as e:
        print(f"An error occurred: {e}")

# Example usage
file_path = 'sample_text_file.txt'  # Replace with the path to your text file
word_count = count_words_in_file(file_path)

if word_count is not None:
    print(f"Number of words in the file: {word_count}")
```

```
Number of words in the file: 2
```

Exp 10: write a python program to read contents of a file in reverse order

```
def read_file_reverse(file_path):
    try:
        with open(file_path, 'r') as file:
            # Read the contents of the file
            content = file.read()
            # Print the contents in reverse order
            print("Contents of the file in reverse order:")
            print(content[::-1])
    except FileNotFoundError:
        print(f"File not found: {file_path}")
    except Exception as e:
        print(f"An error occurred: {e}")

# Example usage
file_path = 'sample_text_file.txt'  # Replace with the path to your text file
read_file_reverse(file_path)
```

```
Contents of the file in reverse order:
!dlrow olleh
```

Exp 11: write a python program to merge and join dataframes using pandas

```
import pandas as pd

# Create two sample DataFrames
df1 = pd.DataFrame({
    'ID': [1, 2, 3],
    'Name': ['John', 'Alice', 'Bob'],
    'Age': [25, 30, 22]
})

df2 = pd.DataFrame({
    'ID': [2, 3, 4],
    'City': ['New York', 'Paris', 'Tokyo'],
    'Salary': [60000, 70000, 80000]
})

# Merge DataFrames based on a common column (ID in this case)
merged_df = pd.merge(df1, df2, on='ID', how='inner')  # You can use 'left', 'right', or 'outer' as well

print("Merged DataFrame:")
print(merged_df)

# Join DataFrames based on index
df1.set_index('ID', inplace=True)
df2.set_index('ID', inplace=True)
joined_df = df1.join(df2, how='inner', lsuffix='_left', rsuffix='_right')

print("\nJoined DataFrame:")
print(joined_df)
```

```
Merged DataFrame:
   ID   Name  Age      City  Salary
0   2  Alice   30  New York   60000
1   3    Bob   22     Paris   70000

Joined DataFrame:
     Name  Age      City  Salary
ID
2   Alice   30  New York   60000
3     Bob   22     Paris   70000
```

Exp 12: write a python program to merge and join dataframes using pandas

```
import pandas as pd

# Create three sample DataFrames
df1 = pd.DataFrame({
    'ID': [1, 2, 3],
    'Name': ['John', 'Alice', 'Bob'],
    'Age': [25, 30, 22]
})

df2 = pd.DataFrame({
    'ID': [2, 3, 4],
    'City': ['New York', 'Paris', 'Tokyo'],
    'Salary': [60000, 70000, 80000]
})

df3 = pd.DataFrame({
    'ID': [1, 2, 4],
    'Department': ['IT', 'HR', 'Marketing'],
    'Experience': [2, 5, 3]
})

# Merge DataFrames based on a common column (ID)
merged_df = pd.merge(df1, df2, on='ID', how='inner')  # You can use 'left', 'right', or 'outer' as well
```

```
print("Merged DataFrame:")
print(merged_df)

# Join DataFrames based on index
df1.set_index('ID', inplace=True)
df2.set_index('ID', inplace=True)
joined_df = df1.join([df2, df3], how='inner')

print("\nJoined DataFrame:")
print(joined_df)
```

```
Merged DataFrame:
   ID   Name  Age      City  Salary
0   2  Alice   30  New York   60000
1   3    Bob   22     Paris   70000

Joined DataFrame:
    Name  Age      City  Salary  ID Department  Experience
2  Alice   30  New York   60000   4  Marketing           3
```

Exp 13: write a python program to append contents of a file to another file

```
def append_file(source_path, destination_path):
    try:
        with open(source_path, 'r') as source_file:
            source_content = source_file.read()

        with open(destination_path, 'a') as destination_file:
            destination_file.write(source_content)

        print(f"Contents from '{source_path}' appended to '{destination_path}' successfully.")
    except FileNotFoundError:
        print(f"File not found: {source_path}")
    except Exception as e:
        print(f"An error occurred: {e}")

# Example usage
source_file_path = 'source.txt'  # Replace with the path to your source file
destination_file_path = 'destination.txt'  # Replace with the path to your destination file

append_file(source_file_path, destination_file_path)
```

```
Contents from 'source.txt' appended to 'destination.txt' successfully.
```

Exp 14: How to install and laod csv files in Python Pandas

```
!pip install pandas
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (1.5.3)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2023.3.post1)
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.10/dist-packages (from pandas) (1.23.5)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.1->pandas)
```

```
import pandas as pd

# Specify the path to your CSV file
csv_file_path = '/content/sample_data/california_housing_train.csv'

# Load the CSV file into a Pandas DataFrame
df = pd.read_csv(csv_file_path,)

# Display the DataFrame
print(df)
```

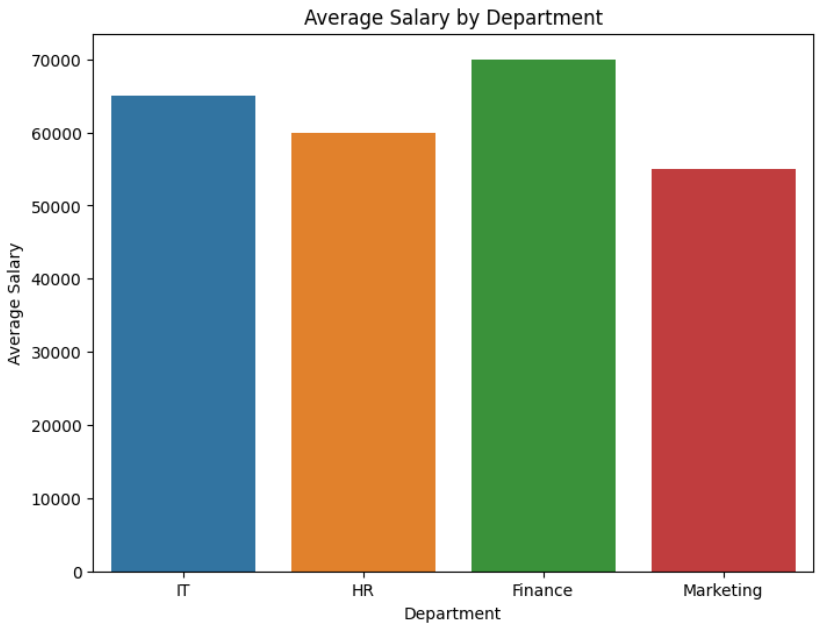
```
       longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
0        -114.31     34.19                15.0       5612.0          1283.0
1        -114.47     34.40                19.0       7650.0          1901.0
2        -114.56     33.69                17.0        720.0           174.0
3        -114.57     33.64                14.0       1501.0           337.0
4        -114.57     33.57                20.0       1454.0           326.0
...          ...       ...                 ...          ...             ...
16995    -124.26     40.58                52.0       2217.0           394.0
16996    -124.27     40.69                36.0       2349.0           528.0
16997    -124.30     41.84                17.0       2677.0           531.0
```

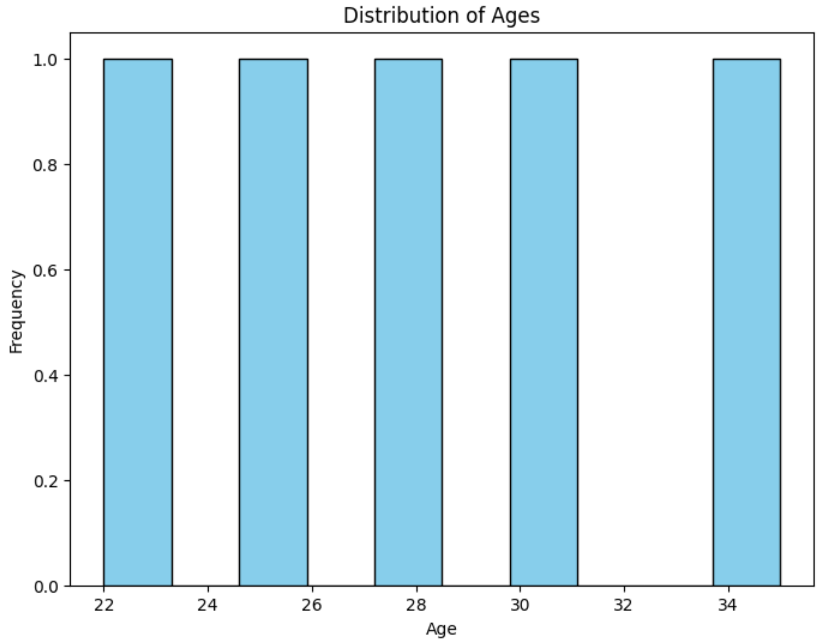
```
16998    -124.30     41.80                19.0       2672.0           552.0
16999    -124.35     40.54                52.0       1820.0           300.0

       population  households  median_income  median_house_value
0          1015.0       472.0         1.4936             66900.0
1          1129.0       463.0         1.8200             80100.0
2           333.0       117.0         1.6509             85700.0
3           515.0       226.0         3.1917             73400.0
4           624.0       262.0         1.9250             65500.0
...           ...         ...            ...                 ...
16995       907.0       369.0         2.3571            111400.0
16996      1194.0       465.0         2.5179             79000.0
16997      1244.0       456.0         3.0313            103600.0
16998      1298.0       478.0         1.9797             85800.0
16999       806.0       270.0         3.0147             94600.0

[17000 rows x 9 columns]
```

Exp 15: write a program to implement data analysis and visualization with python using pandas

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Create a sample DataFrame
data = {
    'Name': ['John', 'Alice', 'Bob', 'Charlie', 'David'],
    'Age': [25, 30, 22, 35, 28],
    'Salary': [50000, 60000, 70000, 80000, 55000],
    'Department': ['IT', 'HR', 'Finance', 'IT', 'Marketing']
}

df = pd.DataFrame(data)

# Display basic information about the DataFrame
print("Basic Info:")
print(df.info())

# Display summary statistics
print("\nSummary Statistics:")
print(df.describe())

# Plot a bar chart of average salary by department using Seaborn
plt.figure(figsize=(8, 6))
sns.barplot(x='Department', y='Salary', data=df, ci=None)
plt.title('Average Salary by Department')
plt.xlabel('Department')
plt.ylabel('Average Salary')
plt.show()

# Plot a histogram of ages using Matplotlib
plt.figure(figsize=(8, 6))
plt.hist(df['Age'], bins=10, color='skyblue', edgecolor='black')
plt.title('Distribution of Ages')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```

```
sns.barplot(x='Department', y='Salary', data=df, ci=None)
```

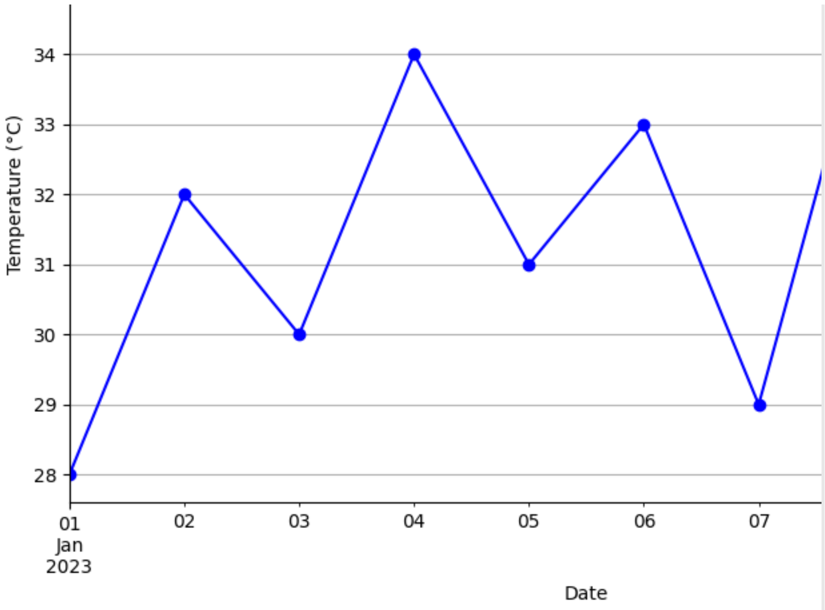

Average Salary by Department

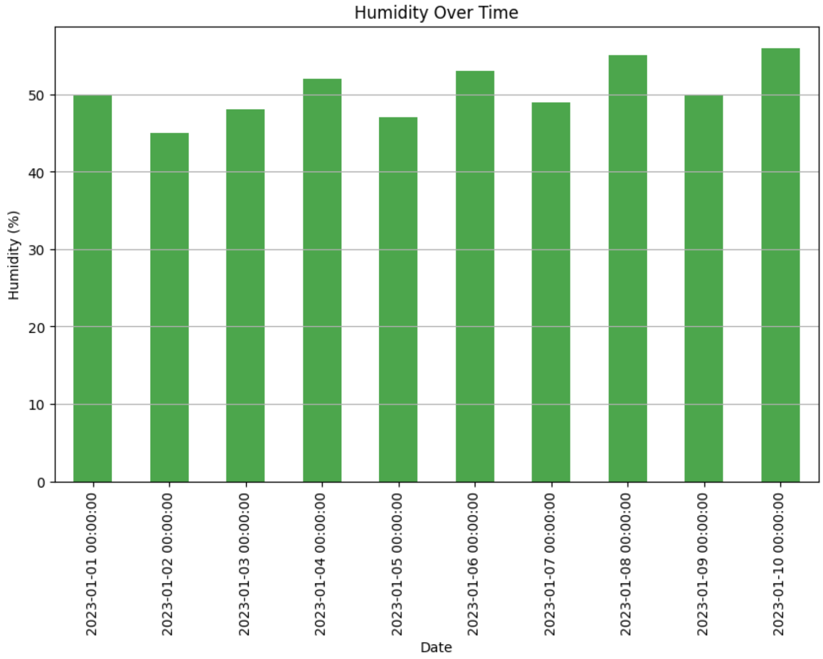

Distribution of Ages

Exp 16: write a program to implement plotting functgions in pandas

```
import pandas as pd
import matplotlib.pyplot as plt

# Create a sample DataFrame
data = {
    'Date': pd.date_range(start='2023-01-01', periods=10, freq='D'),
    'Temperature': [28, 32, 30, 34, 31, 33, 29, 35, 30, 36],
    'Humidity': [50, 45, 48, 52, 47, 53, 49, 55, 50, 56]
}

df = pd.DataFrame(data)
df.set_index('Date', inplace=True)

# Plot a line chart of temperature over time
df['Temperature'].plot(figsize=(10, 6), linestyle='-', marker='o', color='blue')
plt.title('Temperature Over Time')
plt.xlabel('Date')
```

```
    plt.ylabel('Temperature (°C)')
    plt.grid(True)
    plt.show()

    # Plot a bar chart of humidity over time
    df['Humidity'].plot(kind='bar', figsize=(10, 6), color='green', alpha=0.7)
    plt.title('Humidity Over Time')
    plt.xlabel('Date')
    plt.ylabel('Humidity (%)')
    plt.grid(axis='y')
    plt.show()
```





Project: Amazon User Segmentation

Problem Statement

Identify users with simliar purchase bahviour using Amazon Customer Purchase Rating data.

- The data contains the following features/ attributes:
    - Customer ID
    - Sex / Gender
    - Age
    - Income
    - Rating

Solution:

- The problem requires Segmentation of Users, which can be done using Clustering Algorithms.
- Some of the Clustering Algorithms available are:
    - K - Means Clustering
    - Hierarchical Clustering

Importing libraries

```
# data handling libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler,LabelEncoder


# machine learning libraries
from sklearn.cluster import KMeans
from scipy.cluster import hierarchy
from sklearn.cluster import AgglomerativeClustering
```

Importing dataset

```
dataset = pd.read_excel('Amazon User Segmentation.xls')
```

```
dataset
```

	Cus_ID	Sex	Age	Income	Rating
0	301219	M	23	306555	44
1	301220	F	26	306555	91
2	301221	F	24	326992	7
3	301222	M	28	326992	87
4	301223	F	38	347429	45
...	...	...	...	...	...
195	301414	F	42	2452440	89
196	301415	F	54	2575062	32
197	301416	M	39	2575062	83
198	301417	M	39	2799869	21
199	301418	M	36	2799869	93

200 rows × 5 columns

```
X = dataset.iloc[:,[2,4]].values
```

```
X
```

```
       [ 23,   6],
       [ 38, 105],
       [ 60,  30],
       [ 44,  84],
       [ 51,  23],
       [ 40, 107],
       [ 44,  31],
       [ 39,  71],
       [ 48,  15],
       [ 34,  84],
       [ 44,  12],
       [ 44, 104],
       [ 63,  15],
       [ 36,  97],
       [ 70,  17],
       [ 33,  78],
       [ 71,  16],
       [ 42, 101],
       [ 45,  36],
       [ 39,  97],
       [ 56,  17],
       [ 35,  99],
       [ 50,  44],
       [ 36, 109],
       [ 65,  27],
       [ 34,  77],
       [ 50,  20],
       [ 44,  96],
       [ 41,  26],
       [ 39,  78],
       [ 40,   9],
       [ 46, 102],
       [ 57,  18],
       [ 42,  89],
       [ 54,  32],
       [ 39,  83],
       [ 39,  21],
       [ 36,  93]])
```

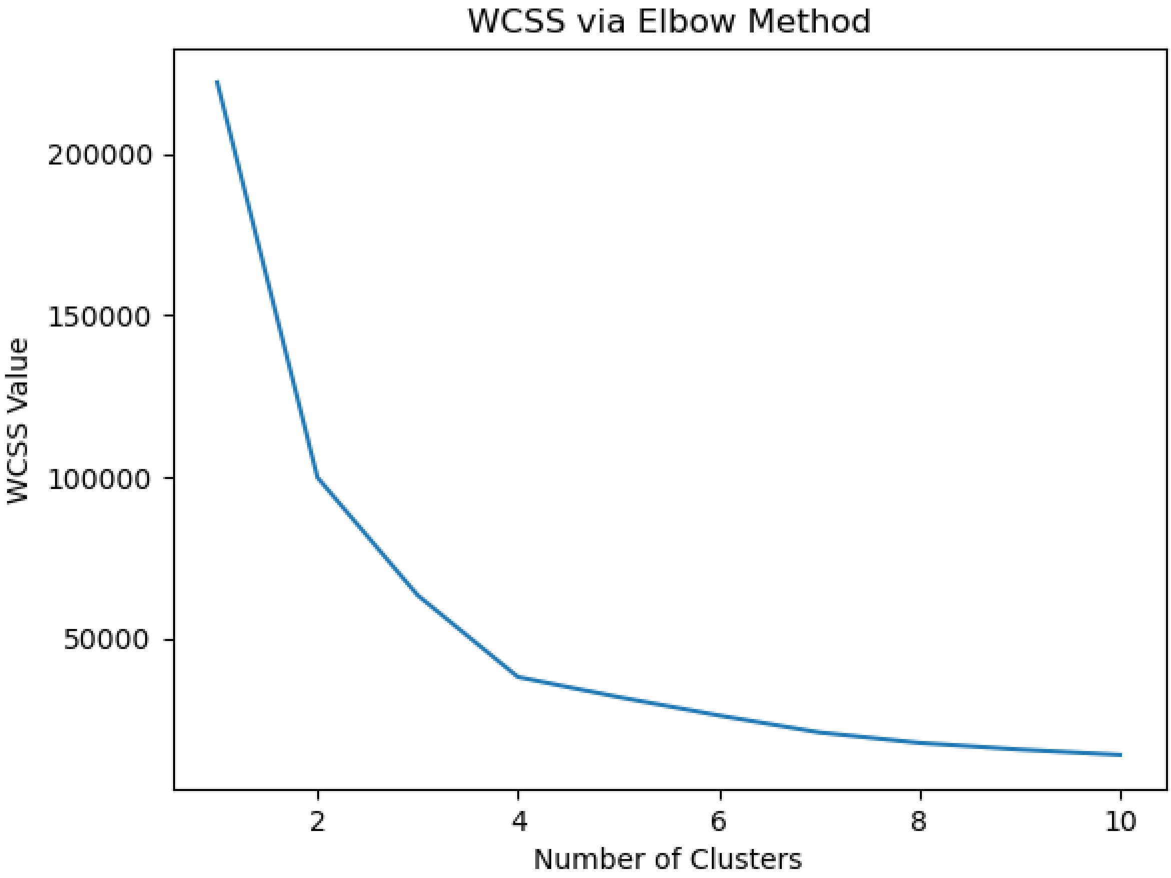
K-Means Clustering -- Age vs Rating

Feature Scaling

Optimal number of clusters via Elbow Method

```
# With-in CLuster Sum of Squares -WCSS is the sum of squared distance between each
wcss = []

for i in range(1,11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=0)
    kmeans.fit(X)
```

```
        wcss.append(kmeans.inertia_)

    plt.plot(range(1,11),wcss)
    plt.title('WCSS via Elbow Method')
    plt.xlabel('Number of Clusters')
    plt.ylabel('WCSS Value')
    plt.show()
```



- No much deviation after number of cluster chosen are >=6

K Means Model Training on Training set

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

Predicting

```
y_means = kmeans.fit_predict(X)
```

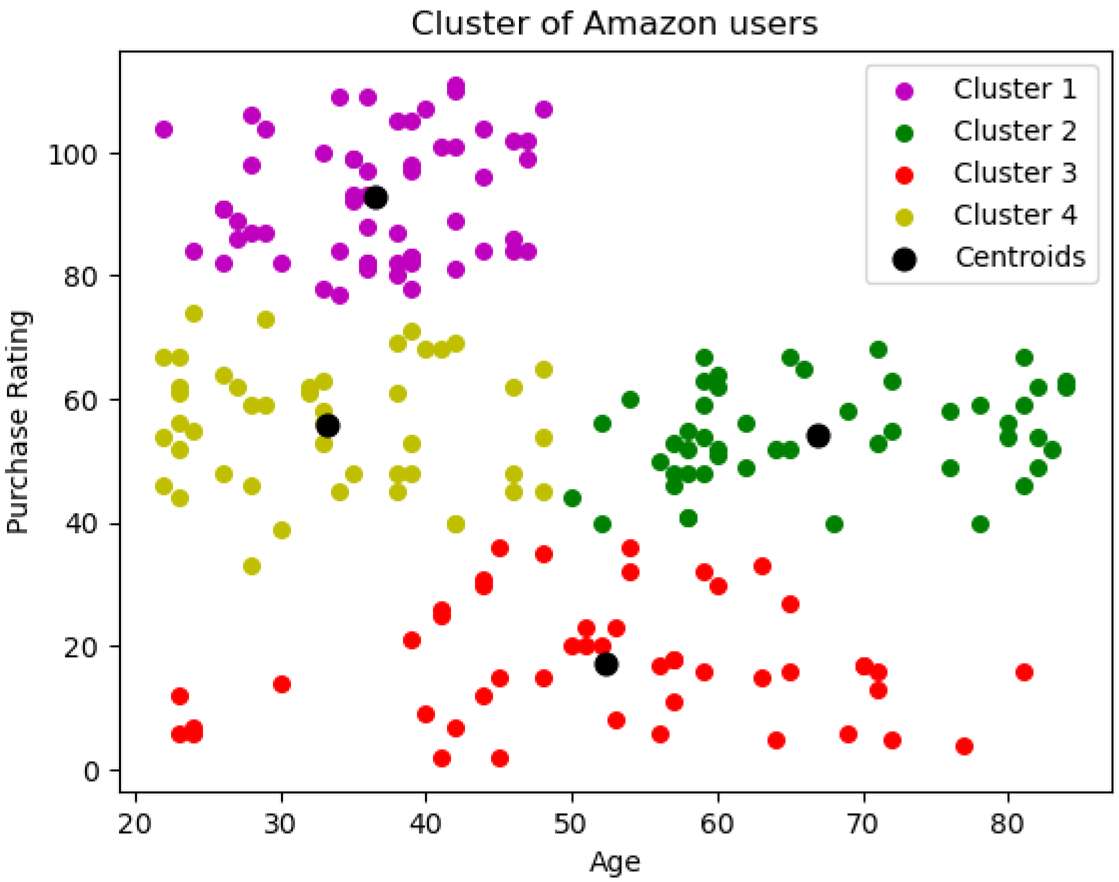
Predicting results

```
y_means
```

```
    array([3, 0, 2, 0, 3, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 3, 3, 2, 0, 3, 0,
           2, 0, 2, 0, 2, 3, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 1, 0, 1, 3,
           2, 3, 1, 3, 3, 3, 1, 3, 3, 1, 1, 1, 1, 1, 3, 1, 1, 3, 1, 1, 1, 3,
           1, 1, 3, 3, 1, 1, 1, 1, 1, 3, 1, 3, 3, 1, 1, 3, 1, 1, 3, 1, 1, 3,
           3, 1, 1, 3, 1, 3, 3, 3, 1, 3, 1, 3, 3, 1, 1, 3, 1, 3, 1, 1, 1, 1,
           1, 3, 3, 3, 3, 3, 1, 1, 1, 1, 3, 3, 3, 0, 3, 0, 1, 0, 2, 0, 2, 0,
           3, 0, 2, 0, 2, 0, 2, 0, 2, 0, 3, 0, 2, 0, 1, 0, 2, 0, 2, 0, 2, 0,
           2, 0, 2, 0, 2, 0, 1, 0, 2, 0, 2, 0, 2, 0, 2, 3, 2, 0, 2, 0, 2, 0,
           2, 0, 2, 0, 2, 0, 2, 0, 1, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0,
           2, 0], dtype=int32)
```

Visualising Clusters

```
plt.scatter(X[y_means==0, 0], X[y_means==0, 1],s=30, c='m', label='Cluster 1')
plt.scatter(X[y_means==1, 0],X[y_means==1,1],s=30,c='g',label='Cluster 2')
plt.scatter(X[y_means==2,0],X[y_means==2,1],s=30,c='r',label='Cluster 3')
plt.scatter(X[y_means==3,0],X[y_means==3,1],s=30,c='y',label='Cluster 4')
plt.scatter(kmeans.cluster_centers_[:,0],kmeans.cluster_centers_[:,1],s=60,c='black
plt.title('Cluster of Amazon users')
plt.xlabel('Age')
plt.ylabel('Purchase Rating')
plt.legend()
plt.show()
```



Double-click (or enter) to edit

Double-click (or enter) to edit

Hierarchical Clustering -- Income vs Rating

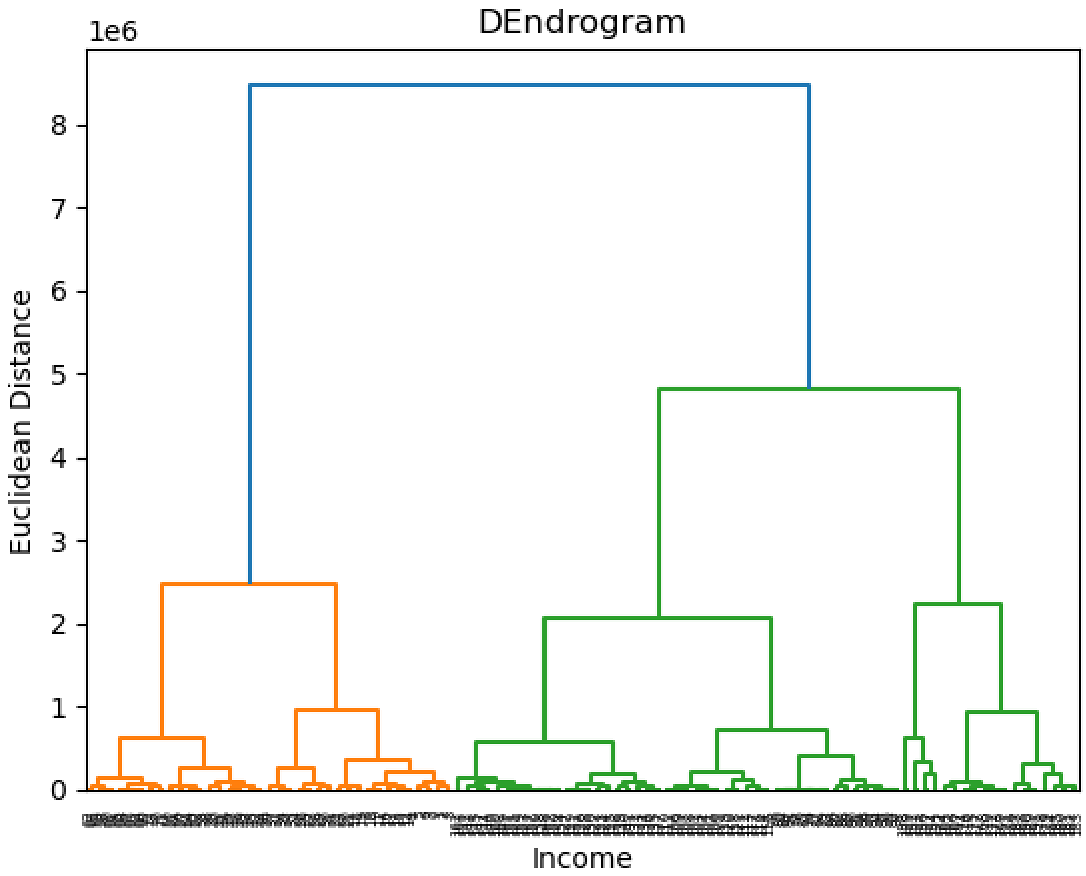
```
X = dataset.iloc[:,[3,4]].values
```

```
X
```

```
array([[ 306555,      44],
       [ 306555,      91],
       [ 326992,       7],
       [ 326992,      87],
       [ 347429,      45],
       [ 347429,      86],
       [ 367866,       7],
       [ 367866,     106],
       [ 388303,       4],
       [ 388303,      81],
       [ 388303,      16],
       [ 388303,     111],
       [ 408740,      17],
       [ 408740,      87],
       [ 408740,      15],
       [ 408740,      89],
       [ 429177,      40],
       [ 429177,      74],
       [ 470051,      33],
       [ 470051,     110],
       [ 490488,      40],
       [ 490488,      82],
       [ 510925,       6],
       [ 510925,      82],
       [ 572236,      16],
       [ 572236,      92],
       [ 572236,      36],
       [ 572236,      69],
       [ 592673,      35],
       [ 592673,      98],
       [ 613110,       5],
       [ 613110,      82],
       [ 674421,       5],
       [ 674421,     104],
       [ 674421,      16],
       [ 674421,      91],
       [ 694858,      20],
       [ 694858,      82],
       [ 756169,      30],
       [ 756169,      84],
       [ 776606,      40],
       [ 776606,     104],
       [ 797043,      41],
       [ 797043,      69],
       [ 797043,      32],
       [ 797043,      73],
       [ 817480,      62],
       [ 817480,      53],
       [ 817480,      48],
       [ 817480,      48],
       [ 858354,      59],
       [ 858354,      68],
       [ 878791,      61],
       [ 878791,      68],
       [ 878791,      51],
       [ 878791,      46],
       [ 899228,      56],
       [ 899228,      52],
```

Optimal number of clusters via DendoGrams

```
dendrogram = hierarchy.dendrogram(hierarchy.linkage(X,method='ward'))
plt.title('DEndrogram')
plt.xlabel('Income')
plt.ylabel('Euclidean Distance')
plt.show()
```



Hierarchical Clustering Model Training on Training set

```
hc = AgglomerativeClustering(n_clusters=2,affinity='euclidean',linkage='ward')
y_hc = hc.fit_predict(X)
```

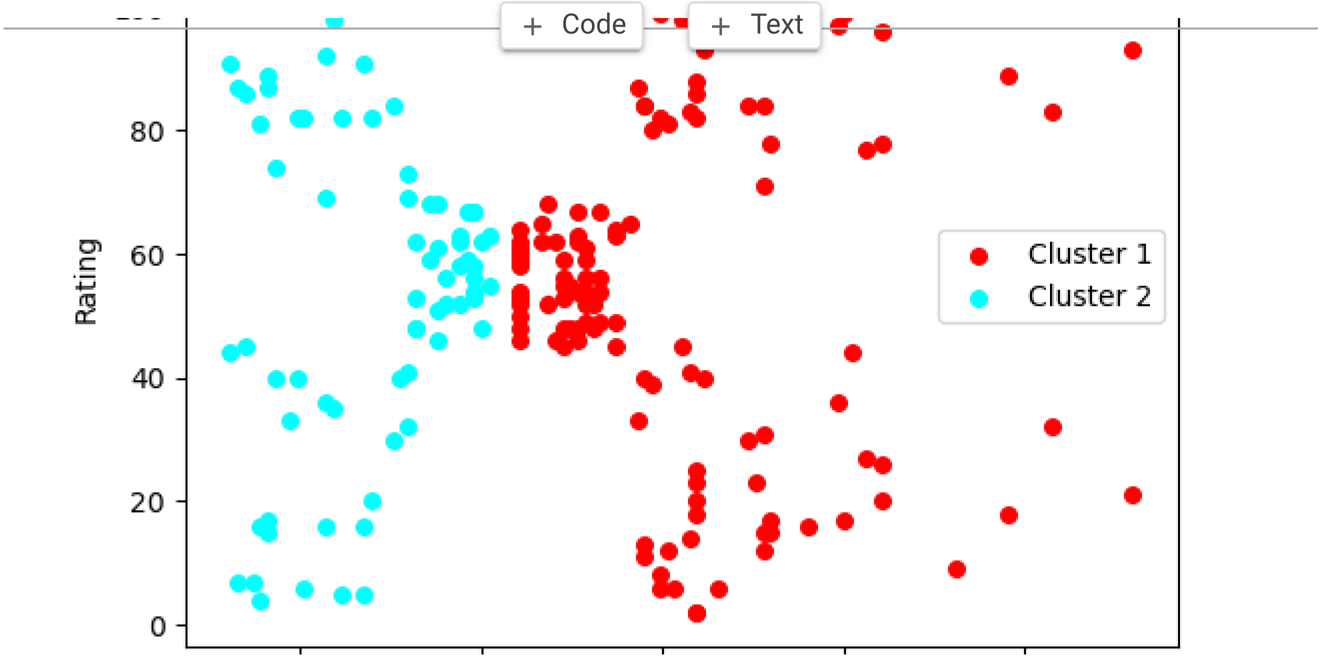
```
y_hc
```

```
    array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
           1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
           1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
           1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
           0, 0])
```

Visualizing Clusters

```
plt.scatter(X[y_hc == 0, 0], X[y_hc == 0, 1], s=30, c='red', label='Cluster 1')
plt.scatter(X[y_hc == 1, 0], X[y_hc == 1, 1], s=30, c='cyan', label='Cluster 2')

plt.title('Amazon Users')
plt.xlabel('Income')
plt.ylabel('Rating')

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

- As we can see K-Menas suggests 4 Clusters of users whereas Hierarchical suggests 2 Clusters


+ Code
+ Text
Rating
● Cluster 1
● Cluster 2