PART-A

WEEK-1

**AIM:**

Extract data from different file formats and display the summary statistics.

**CSV**

Extraction of data from a csv file A comma-separated values file is a delimited text file that uses a comma to separate values. Each line of the file is a data record. Each record consists of one or more fields, separated by commas.

**Code:**

import pandas as pd

from google.colab import drive

drive.mount('/content/drive')

**Output :**Mounted at /content/drive

csv=pd.read\_csv("/content/drive/MyDrive/Academics/5th sem/ml lab/India\_Menu.csv")

print(csv.head())

**Output:**

Menu Category Menu Items Per Serve Size Energy (kCal) \

0 Regular Menu McVeggie™ Burger 168 g 402.05

1 Regular Menu McAloo Tikki Burger® 146 g 339.52

2 Regular Menu McSpicy™ Paneer Burger 199 g 652.76

3 Regular Menu Spicy Paneer Wrap 250 g 674.68

4 Regular Menu American Veg Burger 177 g 512.17

Protein (g) Total fat (g) Sat Fat (g) Trans fat (g) Cholesterols (mg) \

0 10.24 13.83 5.34 0.16 2.49

1 8.50 11.31 4.27 0.20 1.47

2 20.29 39.45 17.12 0.18 21.85

3 20.96 39.10 19.73 0.26 40.93

4 15.30 23.45 10.51 0.17 25.24

Total carbohydrate (g) Total Sugars (g) Added Sugars (g) Sodium (mg)

0 56.54 7.90 4.49 706.13

1 50.27 7.05 4.07 545.34

2 52.33 8.35 5.27 1074.58

3 59.27 3.50 1.08 1087.46

4 56.96 7.85 4.76 1051.24

**CSV READER**

import csv

with open('/content/drive/MyDrive/Academics/5th sem/ml lab/India\_Menu.csv', 'r') as file:

    reader = csv.reader(file)

    for row in reader:

        print(row)

**Output:**

['Menu Category', 'Menu Items', 'Per Serve Size', 'Energy (kCal)', 'Protein (g)', 'Total fat (g)', 'Sat Fat (g)', 'Trans fat (g)', 'Cholesterols (mg)', 'Total carbohydrate (g)', 'Total Sugars (g)', 'Added Sugars (g)', 'Sodium (mg)']

['Regular Menu', 'McVeggie™ Burger', '168 g', '402.05', '10.24', '13.83', '5.34', '0.16', '2.49', '56.54', '7.9', '4.49', '706.13']

['Regular Menu', 'McAloo Tikki Burger®', '146 g', '339.52', '8.5', '11.31', '4.27', '0.2', '1.47', '50.27', '7.05', '4.07', '545.34']

['Regular Menu', 'McSpicy™ Paneer Burger', '199 g', '652.76', '20.29', '39.45', '17.12', '0.18', '21.85', '52.33', '8.35', '5.27', '1074.58']

['Regular Menu', 'Spicy Paneer Wrap', '250 g', '674.68', '20.96', '39.1', '19.73', '0.26', '40.93', '59.27', '3.5', '1.08', '1087.46']

['Regular Menu', 'American Veg Burger', '177 g', '512.17', '15.3', '23.45', '10.51', '0.17', '25.24', '56.96', '7.85', '4.76', '1051.24']

['Regular Menu', 'Veg Maharaja Mac', '306 g', '832.67', '24.17', '37.94', '16.83', '0.28', '36.19', '93.84', '11.52', '6.92', '1529.22']

['Regular Menu', 'Green Chilli Aloo Naan', '132 g', '356.09', '7.91', '15.08', '6.11', '0.24', '9.45', '46.36', '4.53', '1.15', '579.6']

['Regular Menu', 'Pizza Puff', '87 g', '228.21', '5.45', '11.44', '5.72', '0.09', '5.17', '24.79', '2.73', '0.35', '390.74']

['Regular Menu', 'Mc chicken Burger', '173 g', '400.8', '15.66', '15.7', '5.47', '0.16', '31.17', '47.98', '5.53', '4.49', '766.33']

['Regular Menu', 'FILLET-O-FISH Burger', '136 g', '348.11', '15.44', '14.16', '5.79', '0.21', '32.83', '38.85', '5.58', '3.54', '530.54']

['Regular Menu', 'Mc Spicy Chicken Burger', '186 g', '451.92', '21.46', '19.36', '7.63', '0.18', '66.04', '46.08', '5.88', '4.49', '928.52']

['Regular Menu', 'Spicy Chicken Wrap', '257 g', '567.19', '23.74', '26.89', '12.54', '0.27', '87.63', '57.06', '2.52', '1.08', '1152.38']

['Regular Menu', 'Chicken Maharaja Mac', '296 g', '689.12', '34.0', '36.69', '10.33', '0.25', '81.49', '55.39', '8.92', '6.14', '1854.71']

['Regular Menu', 'American Chicken Burger', '165 g', '446.95', '20.29', '22.94', '7.28', '0.15', '47.63', '38.54', '7.48', '4.76', '1132.3']

['Regular Menu', 'Chicken Kebab Burger', '138 g', '357.05', '8.64', '14.02', '4.84', '0.13', '1.51', '47.9', '5.08', '3.49', '548.79']

['Regular Menu', 'Green Chilli Kebab naan', '138 g', '230.95', '5.67', '9.32', '3.27', '0.19', '8.74', '31.06', '3.64', '1.15', '410.78']

['Regular Menu', 'Mc Egg Masala Burger', '126.2 g', '290.42', '12.45', '12.27', '3.64', '0.11', '213.09', '32.89', '4.89', '3.64', '757.91']

['Regular Menu', 'Mc Egg Burger for Happy Meal', '123 g', '282.98', '12.29', '12.21', '3.63', '0.11', '213.09', '31.32', '4.66', '3.64', '399.41']

['Regular Menu', 'Ghee Rice with Mc Spicy Fried Chicken 1 pc', '325 g', '720.3', '26.91', '29.2', '5.08', '0.3', '31.32', '77.47', '3.28', '0.35', '2399.49']

['Regular Menu', 'McSpicy Fried Chicken 1 pc', '115 g', '248.76', '17.33', '14.29', '2.82', '0.06', '31.11', '12.7', '0.58', '0.0', '873.89']

['Regular Menu', '4 piece Chicken McNuggets', '64 g', '169.68', '10.03', '9.54', '4.45', '0.06', '24.66', '10.5', '0.32', '0.0', '313.25']

['Regular Menu', '6 piece Chicken McNuggets', '96 g', '254.52', '15.04', '14.3', '6.68', '0.1', '36.99', '15.74', '0.48', '0.0', '469.87']

['Regular Menu', '9 piece Chicken McNuggets', '144 g', '381.77', '22.56', '21.46', '10.02', '0.14', '55.48', '23.62', '0.72', '0.0', '704.81']

['Regular Menu', '2 piece Chicken Strips', '58 g', '164.44', '10.17', '12.38', '11.41', '0.06', '30.1', '2.68', '0.29', '0.0', '477.22']

['Regular Menu', '3 piece Chicken Strips', '87 g', '246.65', '15.26', '18.57', '17.12', '0.09', '45.15', '4.02', '0.44', '0.0', '715.83']

['Regular Menu', '5 piece Chicken Strips', '145 g', '411.09', '25.43', '28.54', '0.15', '75.26', '6.7', '0.73', '0.72', '0.0', '1193.05']

['Regular Menu', 'Regular Fries', '77 g', '224.59', '3.38', '10.39', '4.97', '0.08', '0.77', '27.08', '0.39', '0.0', '153.15']

['Regular Menu', 'Medium Fries', '109 g', '317.92', '4.79', '14.7', '7.04', '0.11', '1.09', '38.34', '0.55', '0.0', '216.79']

['Regular Menu', 'Large Fries', '154 g', '449.17', '6.76', '20.77', '9.95', '0.15', '1.54', '54.16', '0.77', '0.0', '306.29']

['Regular Menu', 'Regular Wedges', '114 g', '204.65', '3.97', '7.15', '3.39', '0.1', '0.97', '28.74', '0.48', '0.0', '356.44']

['Regular Menu', 'Medium Wedges', '156 g', '280.05', '5.44', '9.79', '4.64', '0.13', '1.33', '39.33', '0.66', '0.0', '487.76']

['Regular Menu', 'Large Wedges', '216 g', '387.76', '7.53', '13.55', '6.43', '0.18', '1.84', '54.46', '0.92', '0.0', '675.35']

['Regular Menu', 'L1 Coffee', '200 ml', '6.8', '0.0', '0.0', '0.0', '0.0', '0.0', '1.7', '0.0', '0.0', '0.0']

['Regular Menu', 'L1 Coffee with milk', '205 ml', '35.8', '1.0', '2.0', '1.2', '0.08', '6.0', '1.6', '3.45', '0.0', '14.0']

['Regular Menu', 'Double Chocochips Muffin', '80 g', '341.68', '5.13', '17.28', '7.14', '0.08', '15.96', '40.13', '29.44', '0.0', '313.21']

['Regular Menu', 'Vanilla Chocochips Muffin', '80 g', '329.29', '4.48', '15.46', '7.14', '0.08', '78.52', '40.13', '29.6', '0.0', '254.92']

['Breakfast Menu', 'Veg McMuffin', '119 g', '309.35', '10.22', '11.78', '7.29', '0.18', '25.31', '38.86', '3.02', '0.75', '804.63']

['Breakfast Menu', 'Double Cheese McMuffin', '100 g', '273.78', '9.58', '12.82', '8.84', '0.21', '37.75', '29.0', '2.59', '0.75', '622.95']

['Breakfast Menu', 'Spicy Egg McMuffin', '123.2 g', '278.27', '11.49', '11.81', '3.43', '0.11', '212.61', '31.37', '2.63', '1.05', '773.6']

['Breakfast Menu', 'Sausage Mc Muffin', '112 g', '281.44', '16.25', '10.81', '6.04', '0.17', '53.02', '28.62', '2.38', '0.75', '742.6']

['Breakfast Menu', 'Sausage Mc Muffin with egg', '157 g', '290.42', '22.46', '15.94', '8.08', '0.22', '264.8', '28.87', '2.61', '0.75', '804.04']

['Breakfast Menu', 'Egg McMuffin', '112 g', '283.46', '14.05', '12.31', '6.92', '0.17', '233.3', '28.12', '2.38', '0.75', '519.31']

['Breakfast Menu', 'Hot Cake with maple syrup', '142 g', '432.98', '8.6', '14.02', '7.11', '0.32', '28.14', '68.01', '25.72', '13.5', '615.74']

['Breakfast Menu', 'Hash Brown', '64 g', '140.29', '1.93', '7.32', '3.42', '0.06', '0.64', '15.63', '0.32', '0.0', '275.26']

['Breakfast Menu', 'Espresso', '26.5 ml', '12.87', '0.52', '0.03', '0.03', '0.03', '0.27', '2.55', '0.13', '0.0', '0.32']

['Breakfast Menu', 'Espresso Machiato', '76.5 ml', '44.98', '2.09', '2.02', '1.49', '0.08', '6.27', '4.97', '2.5', '0.0', '26.05']

['Breakfast Menu', 'Americano (S)', '276.5 ml', '12.87', '0.52', '0.03', '0.03', '0.03', '0.27', '2.55', '0.13', '0.0', '0.32']

['Breakfast Menu', 'Americano (R)', '347.5 ml', '23.07', '0.94', '0.05', '0.05', '0.05', '0.48', '4.57', '0.24', '0.0', '0.57']

['Breakfast Menu', 'Americano (L)', '455 ml', '26.71', '1.09', '0.06', '0.06', '0.06', '0.55', '5.3', '0.28', '0.0', '0.65']

['Breakfast Menu', 'Cappuccino (S)', '201.5 ml', '125.25', '6.02', '7.01', '5.15', '0.2', '21.27', '11.02', '8.4', '0.0', '90.39']

['Breakfast Menu', 'Cappuccino (R)', '297.5 ml', '183.61', '8.79', '10.02', '7.37', '0.3', '30.48', '16.67', '12.05', '0.0', '129.24']

['McCafe Menu', 'Cappuccino (L)', '355 ml', '219.36', '10.51', '12.03', '8.85', '0.36', '36.55', '19.81', '14.45', '0.0', '155.06']

['McCafe Menu', 'Latte (S)', '236.5 ml', '147.72', '7.12', '8.41', '6.18', '0.24', '25.47', '12.71', '10.06', '0.0', '108.4']

['McCafe Menu', 'Latte (R)', '307.5 ml', '190.03', '9.1', '10.42', '7.67', '0.31', '31.68', '17.15', '12.52', '0.0', '134.39']

['McCafe Menu', 'Latte (L)', '375 ml', '232.2', '11.14', '12.82', '9.43', '0.38', '38.95', '20.77', '15.4', '0.0', '165.36']

['McCafe Menu', 'Flat White (S)', '266.5 ml', '166.99', '8.06', '9.6', '7.06', '0.27', '29.07', '14.16', '11.47', '0.0', '123.84']

['McCafe Menu', 'Flat White (R)', '347.5 ml', '215.72', '10.36', '12.02', '8.84', '0.35', '36.48', '19.08', '14.41', '0.0', '154.98']

['McCafe Menu', 'Flat White (L)', '405 ml', '251.47', '12.08', '14.02', '10.31', '0.41', '42.55', '22.22', '16.81', '0.0', '180.8']

['McCafe Menu', 'Mocha (S)', '242.5 ml', '185.85', '7.15', '8.2', '5.94', '0.26', '24.43', '22.59', '17.57', '6.08', '132.84']

['McCafe Menu', 'Mocha (R)', '311.5 ml', '244.0', '8.99', '9.91', '7.17', '0.34', '29.52', '31.72', '23.56', '9.12', '168.47']

['McCafe Menu', 'Mocha (L)', '377 ml', '302.02', '10.88', '12.01', '8.67', '0.41', '35.67', '40.04', '29.96', '12.16', '209.09']

['McCafe Menu', 'Babycino', '127 ml', '143.5', '3.87', '4.38', '3.08', '0.15', '12.27', '22.85', '18.53', '9.21', '96.44']

['McCafe Menu', 'Hot Chocolate (S)', '223 ml', '239.42', '6.73', '7.77', '5.49', '0.26', '22.03', '37.08', '30.31', '15.29', '167.21']

['McCafe Menu', 'Hot Chocolate (R)', '259 ml', '296.81', '7.7', '8.76', '6.13', '0.32', '24.59', '48.41', '39.24', '21.37', '207.09']

['McCafe Menu', 'Hot Chocolate (L)', '367 ml', '383.29', '11.01', '12.84', '9.09', '0.43', '36.67', '58.43', '47.96', '24.41', '273.35']

['McCafe Menu', 'Premium Dark Hot Chocolate', '153 ml', '214.21', '6.15', '5.96', '4.27', '0.15', '14.73', '33.04', '25.73', '1.04', '68.28']

['McCafe Menu', 'Double Dark Hot Chocolate', '163 ml', '255.78', '6.87', '6.32', '4.49', '0.16', '14.83', '41.29', '31.81', '1.35', '70.26']

['McCafe Menu', 'English Breakfast (S)', '279 ml', '9.93', '0.56', '0.28', '0.28', '0.28', '2.79', '0.28', '1.4', '0.0', '13.84']

['McCafe Menu', 'English Breakfast (R)', '330 ml', '11.75', '0.66', '0.33', '0.33', '0.33', '3.3', '0.33', '1.65', '0.0', '16.37']

['McCafe Menu', 'English Breakfast (L)', '456 ml', '16.23', '0.91', '0.46', '0.46', '0.46', '4.56', '0.46', '2.28', '0.0', '22.62']

['McCafe Menu', 'Moroccon Mint Green Tea (S)', '279 ml', '6.25', '0.33', '0.28', '0.28', '0.28', '2.79', '2.79', '1.4', '0.0', '14.95']

['McCafe Menu', 'Moroccon Mint Green Tea (R)', '330 ml', '7.39', '0.4', '0.33', '0.33', '0.33', '3.3', '3.3', '1.65', '0.0', '17.69']

['McCafe Menu', 'Moroccon Mint Green Tea (L)', '456 ml', '10.21', '0.55', '0.46', '0.46', '0.46', '4.56', '4.56', '2.28', '0.0', '24.44']

['McCafe Menu', 'Strawberry Green Tea (S)', '279 ml', '7.03', '0.47', '0.28', '0.28', '0.28', '2.79', '2.79', '1.4', '0.0', '14.54']

['McCafe Menu', 'Strawberry Green Tea (R)', '330 ml', '8.32', '0.56', '0.33', '0.33', '0.33', '3.3', '3.3', '1.65', '0.0', '17.19']

['McCafe Menu', 'Strawberry Green Tea (L)', '456 ml', '11.49', '0.78', '0.46', '0.46', '0.46', '4.56', '4.56', '2.28', '0.0', '23.76']

['McCafe Menu', 'Lemon Ice Tea', '245 ml', '121.86', '0.27', '0.17', '0.17', '0.17', '1.65', '30.59', '26.53', '25.6', '10.26']

['McCafe Menu', 'Strawberry Ice Tea', '236.5 ml', '94.95', '0.24', '0.16', '0.16', '0.16', '1.57', '24.17', '21.1', '20.27', '9.72']

['McCafe Menu', 'Green Apple Ice Tea', '236.5 ml', '94.94', '0.24', '0.16', '0.16', '0.16', '1.57', '24.17', '20.75', '19.94', '9.61']

['McCafe Menu', 'Iced Coffee', '291.5 ml', '185.34', '4.36', '4.45', '3.26', '0.15', '12.13', '31.88', '26.95', '17.5', '78.35']

['McCafe Menu', 'Cold Coffee Frappe', '296.5 ml', '331.17', '4.98', '14.73', '13.91', '0.16', '9.18', '45.39', '35.57', '27.51', '188.93']

['McCafe Menu', 'Mocha Frappe', '320.5 ml', '397.98', '5.49', '15.01', '14.0', '0.2', '9.42', '60.93', '47.55', '36.63', '233.32']

['McCafe Menu', 'Chocolate Oreo Frappe', '334 ml', '481.11', '6.03', '18.89', '15.91', '0.22', '9.36', '72.51', '55.14', '44.35', '332.6']

['McCafe Menu', 'Strawberry Shake', '259 ml', '255.51', '3.67', '7.44', '6.68', '0.12', '8.39', '44.07', '37.42', '29.8', '139.97']

['McCafe Menu', 'Chocolate Shake', '259 ml', '270.9', '4.16', '7.7', '6.74', '0.14', '8.39', '46.76', '37.78', '27.88', '178.46']

['McCafe Menu', 'Mango Smoothie', '280 ml', '231.44', '3.21', '3.63', '2.65', '0.14', '9.89', '46.25', '38.87', '29.72', '85.45']

['McCafe Menu', 'Mixed Berry Smoothie', '290 ml', '235.43', '3.33', '3.59', '2.64', '0.15', '9.99', '47.16', '43.0', '33.65', '92.07']

['McCafe Menu', 'Raw Mango Cooler', '310 ml', '102.38', '0.14', '0.04', '0.04', '0.04', '0.4', '25.18', '21.06', '19.28', '102.68']

['McCafe Menu', 'Mix Berry Cooler', '310 ml', '103.85', '0.16', '0.04', '0.04', '0.04', '0.4', '25.56', '21.25', '20.52', '23.82']

['McCafe Menu', 'Sweet Lime Beverage', '310 ml', '128.21', '0.07', '0.08', '0.0', '0.0', '0.0', '31.72', '28.72', '28.08', '66.2']

['McCafe Menu', 'Iced Americano', '266.5 ml', '150.85', '3.59', '3.57', '2.62', '0.13', '9.76', '26.01', '21.58', '14.0', '62.75']

['McCafe Menu', 'American Mud Pie Shake', '317 ml', '398.19', '5.67', '12.77', '11.38', '0.2', '10.89', '64.75', '53.4', '34.35', '185.73']

['McCafe Menu', 'Soft serve cone', '81.29 g', '85.73', '1.99', '1.82', '1.31', '0.05', '4.75', '15.23', '10.68', '6.99', '40.78']

['McCafe Menu', 'McSwirl ChocoDip', '93.29 g', '160.14', '2.71', '7.14', '5.25', '0.07', '5.71', '20.92', '15.39', '11.31', '51.31']

['McCafe Menu', 'Regular Soft Serve: Hot Fudge', '91.79 g', '121.64', '2.25', '4.02', '3.01', '0.08', '5.85', '19.11', '17.07', '10.78', '65.56']

['McCafe Menu', 'Medium Soft Serve: Hot Fudge', '132.08 g', '197.45', '3.49', '6.87', '5.16', '0.13', '8.55', '30.42', '27.01', '16.9', '110.39']

['McCafe Menu', 'Regular Soft Serve: Strawberry', '91.79 g', '100.99', '1.54', '1.77', '1.3', '0.06', '4.85', '19.78', '17.66', '12.49', '34.51']

['McCafe Menu', 'Medium Soft Serve: Strawberry', '132.08 g', '156.14', '2.05', '2.36', '1.74', '0.1', '6.55', '31.77', '28.2', '20.32', '48.28']

['McCafe Menu', 'Regular Soft Serve: Brownie with Hot Fudge', '110.79 g', '205.26', '3.2', '5.45', '3.65', '0.1', '6.04', '35.26', '20.75', '14.39', '100.89']

['McCafe Menu', 'Medium Soft Serve: Brownie with Hot Fudge', '155.08 g', '311.39', '4.65', '7.46', '4.71', '0.13', '7.78', '55.24', '27.94', '20.28', '146.4']

['McCafe Menu', 'Regular Blackforest', '125.79 g', '237.89', '3.22', '5.47', '3.66', '0.11', '6.19', '43.42', '27.79', '19.94', '104.47']

['McCafe Menu', 'Medium Blackforest', '200.08 g', '429.95', '5.42', '9.76', '6.47', '0.19', '9.23', '79.04', '48.45', '35.22', '188.2']

['Desserts Menu', 'Small McFlurry - Oreo', '86.79 g', '116.36', '2.05', '3.7', '2.25', '0.07', '4.8', '18.69', '14.49', '10.8', '80.73']

['Desserts Menu', 'Regular McFlurry - Oreo', '147.38 g', '209.39', '3.58', '6.81', '4.07', '0.12', '8.0', '33.42', '25.35', '19.23', '150.9']

['Gourmet Menu', 'American Triple Cheese Chicken', '195 g', '457.94', '24.43', '22.65', '11.56', '0.17', '71.23', '37.45', '7.64', '3.84', '1396.17']

['Gourmet Menu', 'American Triple Cheese Veg', '207 g', '524.69', '19.54', '23.16', '14.78', '0.19', '48.74', '56.24', '7.9', '3.84', '1174.27']

['Gourmet Menu', 'Cheese Lava Burger', '240 g', '671.06', '14.99', '33.48', '14.12', '0.21', '33.21', '74.25', '16.27', '10.01', '1153.99']

['Gourmet Menu', 'Chicken Cheese Lava Burger', '307 g', '834.36', '27.37', '45.18', '17.0', '0.27', '73.11', '76.03', '16.75', '10.01', '1745.04']

['Gourmet Menu', 'Chunky Chipotle American Burger Chicken', '301 g', '641.36', '39.47', '31.51', '9.54', '0.26', '110.37', '46.24', '9.16', '6.32', '1906.27']

['Gourmet Menu', 'McSpicy Premium Chicken Burger', '264.5 g', '622.25', '31.49', '34.65', '15.55', '0.24', '302.61', '43.6', '6.07', '2.64', '1614.88']

['Gourmet Menu', 'McSpicy Premium Veg Burger', '212.5 g', '634.71', '22.44', '39.21', '20.46', '0.2', '43.68', '46.0', '7.57', '3.28', '1446.87']

['Gourmet Menu', 'Piri piri Mc Spicy Chicken Burger', '228 g', '443.4', '25.63', '17.3', '4.01', '0.19', '64.19', '43.29', '9.29', '6.32', '']

['Gourmet Menu', 'Piri piri Mc Spicy Veg Burger', '211 g', '517.98', '11.97', '24.53', '6.01', '0.18', '8.1', '58.87', '12.87', '6.67', '1170.89']

['Gourmet Menu', 'Cheesy Veg Nuggets (6pc)', '90 g', '252.29', '8.48', '13.09', '7.53', '0.09', '20.03', '23.6', '1.31', '0.0', '428.17']

['Gourmet Menu', 'Cheesy Veg Nuggets (9pc)', '135 g', '378.43', '12.72', '19.63', '11.3', '0.14', '30.05', '35.4', '1.96', '0.0', '642.25']

['Beverages Menu', 'Small Coca-Cola', '299 ml', '109.56', '0.0', '0.0', '0.0', '0.0', '0.0', '27.39', '27.39', '27.39', '21.17']

['Beverages Menu', 'Medium Coca-Cola', '394 ml', '151.36', '0.0', '0.0', '0.0', '0.0', '0.0', '37.84', '37.84', '37.84', '29.24']

['Beverages Menu', 'Large Coca-Cola', '544 ml', '217.36', '0.0', '0.0', '0.0', '0.0', '0.0', '54.34', '54.34', '54.34', '41.99']

['Beverages Menu', 'Small Fanta Oragne', '299 ml', '129.48', '0.0', '0.0', '0.0', '0.0', '0.0', '32.37', '32.37', '32.37', '55.53']

['Beverages Menu', 'Medium Fanta Orange', '394 ml', '178.88', '0.0', '0.0', '0.0', '0.0', '0.0', '44.72', '44.72', '44.72', '76.71']

['Beverages Menu', 'Large Fanta Oragne', '544 ml', '256.88', '0.0', '0.0', '0.0', '0.0', '0.0', '64.22', '64.22', '64.22', '110.16']

['Beverages Menu', 'Small Thums-up', '299 ml', '99.6', '0.0', '0.0', '0.0', '0.0', '0.0', '24.9', '24.9', '24.9', '25.15']

['Beverages Menu', 'Medium Thums-up', '394 ml', '137.6', '0.0', '0.0', '0.0', '0.0', '0.0', '34.4', '34.4', '34.4', '34.74']

['Beverages Menu', 'Large Thums-up', '544 ml', '197.6', '0.0', '0.0', '0.0', '0.0', '0.0', '49.4', '49.4', '49.4', '49.89']

['Beverages Menu', 'Small Sprite', '299 ml', '119.52', '0.0', '0.0', '0.0', '0.0', '0.0', '29.88', '29.88', '29.88', '2.02']

['Beverages Menu', 'Medium Sprite', '394 ml', '165.12', '0.0', '0.0', '0.0', '0.0', '0.0', '41.28', '41.28', '41.28', '2.79']

['Beverages Menu', 'Large Sprite', '544 ml', '237.12', '0.0', '0.0', '0.0', '0.0', '0.0', '59.28', '59.28', '59.28', '4.0']

['Beverages Menu', 'Coke Float', '286.79 ml', '138.76', '1.52', '1.75', '1.28', '0.05', '4.7', '29.22', '28.23', '24.54', '44.53']

['Beverages Menu', 'Fanta Float', '286.79 ml', '151.56', '1.52', '1.75', '1.28', '0.05', '4.7', '32.42', '31.43', '27.74', '66.61']

['Beverages Menu', 'Sprite Float', '286.79 ml', '145.16', '1.52', '1.75', '1.28', '0.05', '4.7', '30.82', '29.83', '26.14', '47.09']

['Beverages Menu', 'Coke Zero Can', '330 ml', '0.99', '0.0', '0.0', '0.0', '0.0', '0.0', '0.0', '0.0', '0.0', '24.75']

['Beverages Menu', 'Vedica Natural Mineral Water', '500 ml', '0.0', '0.0', '0.0', '0.0', '0.0', '0.0', '0.0', '0.0', '0.0', '2.9']

['Condiments Menu', 'Mustard diping sauce', '25 g', '81.18', '0.52', '5.57', '1.78', '0.47', '0.29', '7.24', '6.66', '4.0', '221.32']

['Condiments Menu', 'BBQ diping sauce', '25 g', '54.89', '0.26', '0.49', '0.15', '0.04', '0.25', '12.36', '7.65', '2.5', '113.23']

['Condiments Menu', 'Chilli Sauce', '10 g', '8.07', '0.03', '0.01', '0.01', '0.01', '0.1', '1.99', '1.53', '1.34', '65.24']

['Condiments Menu', 'Piri Piri Mix', '5 g', '17.13', '0.51', '0.36', '0.08', '0.01', '0.05', '2.5', '0.66', '0.4', '414.71']

['Condiments Menu', 'Tomato Ketchup Sachets', '8 g', '11.23', '0.08', '23.45', '0.0', '0.01', '0.08', '2.63', '2.33', '1.64', '71.05']

['Condiments Menu', 'Maple Syrup', '30 g', '86.4', '0.0', '0.0', '0.0', '0.0', '0.3', '21.6', '16.2', '5.34', '15.0']

['Condiments Menu', 'Cheese Slice', '14 g', '51.03', '3.06', '3.99', '2.89', '0.01', '13.43', '0.72', '0.54', '0.0', '178.95']

['Condiments Menu', 'Sweet Corn', '40 g', '45.08', '1.47', '1.0', '0.22', '0.04', '2.0', '7.55', '2.54', '0.0', '0.04']

['Condiments Menu', 'Mixed Fruit Beverage', '180 ml', '72.25', '0.65', '0.02', '0.02', '0.02', '0.01', '18.0', '16.83', '0.0', '10.8']

csv.info()

csv.describe()

**Output:**

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 141 entries, 0 to 140

Data columns (total 13 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Menu Category 141 non-null object

1 Menu Items 141 non-null object

2 Per Serve Size 141 non-null object

3 Energy (kCal) 141 non-null float64

4 Protein (g) 141 non-null float64

5 Total fat (g) 141 non-null float64

6 Sat Fat (g) 141 non-null float64

7 Trans fat (g) 141 non-null float64

8 Cholesterols (mg) 141 non-null float64

9 Total carbohydrate (g) 141 non-null float64

10 Total Sugars (g) 141 non-null float64

11 Added Sugars (g) 141 non-null float64

12 Sodium (mg) 140 non-null float64

dtypes: float64(10), object(3)

memory usage: 14.4+ KB

|  | **Energy (kCal)** | **Protein (g)** | **Total fat (g)** | **Sat Fat (g)** | **Trans fat (g)** | **Cholesterols (mg)** | **Total carbohydrate (g)** | **Total Sugars (g)** | **Added Sugars (g)** | **Sodium (mg)** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **count** | 141.000000 | 141.000000 | 141.000000 | 141.000000 | 141.000000 | 141.000000 | 141.000000 | 141.000000 | 141.000000 | 140.000000 |
| **mean** | 244.635461 | 7.493546 | 9.991702 | 4.997589 | 0.687163 | 26.350071 | 31.190284 | 15.464894 | 10.336950 | 362.064143 |
| **std** | 185.554837 | 8.336863 | 10.339511 | 4.900451 | 6.326136 | 50.334200 | 20.602044 | 15.690202 | 14.283388 | 473.160490 |
| **min** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| **25%** | 116.360000 | 0.650000 | 0.460000 | 0.280000 | 0.060000 | 1.510000 | 15.740000 | 2.330000 | 0.000000 | 43.895000 |
| **50%** | 219.360000 | 4.790000 | 7.770000 | 4.270000 | 0.150000 | 8.390000 | 30.820000 | 9.160000 | 3.640000 | 152.025000 |
| **75%** | 339.520000 | 10.880000 | 14.160000 | 7.280000 | 0.220000 | 31.110000 | 46.000000 | 26.950000 | 19.230000 | 534.240000 |
| **max** | 834.360000 | 39.470000 | 45.180000 | 20.460000 | 75.260000 | 302.610000 | 93.840000 | 64.220000 | 64.220000 | 2399.490000 |

**JSON**

**Description**

JSON is a human-readable format that is easy/simple to parse in most programming/scripting

languages. A JSON file/object is simply a collection of name(key)-value pairs. Such key-value

pair structures have corresponding data structures available in programming languages in the form of dictionaries

**Code:**

import json

f=open('/content/drive/MyDrive/Academics/5th sem/ml lab/iris.json')

data=json.load(f)

for i in data:

  print(i)

f.close()

**Output:**

{'sepalLength': 5.1, 'sepalWidth': 3.5, 'petalLength': 1.4, 'petalWidth': 0.2, 'species': 'setosa'}

{'sepalLength': 4.9, 'sepalWidth': 3.0, 'petalLength': 1.4, 'petalWidth': 0.2, 'species': 'setosa'}

{'sepalLength': 4.7, 'sepalWidth': 3.2, 'petalLength': 1.3, 'petalWidth': 0.2, 'species': 'setosa'}

{'sepalLength': 4.6, 'sepalWidth': 3.1, 'petalLength': 1.5, 'petalWidth': 0.2, 'species': 'setosa'}

{'sepalLength': 5.0, 'sepalWidth': 3.6, 'petalLength': 1.4, 'petalWidth': 0.2, 'species': 'setosa'}

{'sepalLength': 5.4, 'sepalWidth': 3.9, 'petalLength': 1.7, 'petalWidth': 0.4, 'species': 'setosa'}

{'sepalLength': 4.6, 'sepalWidth': 3.4, 'petalLength': 1.4, 'petalWidth': 0.3, 'species': 'setosa'}

{'sepalLength': 5.0, 'sepalWidth': 3.4, 'petalLength': 1.5, 'petalWidth': 0.2, 'species': 'setosa'}

{'sepalLength': 4.4, 'sepalWidth': 2.9, 'petalLength': 1.4, 'petalWidth': 0.2, 'species': 'setosa'}

{'sepalLength': 4.9, 'sepalWidth': 3.1, 'petalLength': 1.5, 'petalWidth': 0.1, 'species': 'setosa'}

{'sepalLength': 5.4, 'sepalWidth': 3.7, 'petalLength': 1.5, 'petalWidth': 0.2, 'species': 'setosa'}

{'sepalLength': 4.8, 'sepalWidth': 3.4, 'petalLength': 1.6, 'petalWidth': 0.2, 'species': 'setosa'}

{'sepalLength': 4.8, 'sepalWidth': 3.0, 'petalLength': 1.4, 'petalWidth': 0.1, 'species': 'setosa'}

{'sepalLength': 4.3, 'sepalWidth': 3.0, 'petalLength': 1.1, 'petalWidth': 0.1, 'species': 'setosa'}

{'sepalLength': 5.8, 'sepalWidth': 4.0, 'petalLength': 1.2, 'petalWidth': 0.2, 'species': 'setosa'}

{'sepalLength': 5.7, 'sepalWidth': 4.4, 'petalLength': 1.5, 'petalWidth': 0.4, 'species': 'setosa'}

{'sepalLength': 5.4, 'sepalWidth': 3.9, 'petalLength': 1.3, 'petalWidth': 0.4, 'species': 'setosa'}

{'sepalLength': 5.1, 'sepalWidth': 3.5, 'petalLength': 1.4, 'petalWidth': 0.3, 'species': 'setosa'}

{'sepalLength': 5.7, 'sepalWidth': 3.8, 'petalLength': 1.7, 'petalWidth': 0.3, 'species': 'setosa'}

{'sepalLength': 5.1, 'sepalWidth': 3.8, 'petalLength': 1.5, 'petalWidth': 0.3, 'species': 'setosa'}

{'sepalLength': 5.4, 'sepalWidth': 3.4, 'petalLength': 1.7, 'petalWidth': 0.2, 'species': 'setosa'}

{'sepalLength': 5.1, 'sepalWidth': 3.7, 'petalLength': 1.5, 'petalWidth': 0.4, 'species': 'setosa'}

{'sepalLength': 4.6, 'sepalWidth': 3.6, 'petalLength': 1.0, 'petalWidth': 0.2, 'species': 'setosa'}

{'sepalLength': 5.1, 'sepalWidth': 3.3, 'petalLength': 1.7, 'petalWidth': 0.5, 'species': 'setosa'}

{'sepalLength': 4.8, 'sepalWidth': 3.4, 'petalLength': 1.9, 'petalWidth': 0.2, 'species': 'setosa'}

{'sepalLength': 5.0, 'sepalWidth': 3.0, 'petalLength': 1.6, 'petalWidth': 0.2, 'species': 'setosa'}

{'sepalLength': 5.0, 'sepalWidth': 3.4, 'petalLength': 1.6, 'petalWidth': 0.4, 'species': 'setosa'}

{'sepalLength': 5.2, 'sepalWidth': 3.5, 'petalLength': 1.5, 'petalWidth': 0.2, 'species': 'setosa'}

{'sepalLength': 5.2, 'sepalWidth': 3.4, 'petalLength': 1.4, 'petalWidth': 0.2, 'species': 'setosa'}

{'sepalLength': 4.7, 'sepalWidth': 3.2, 'petalLength': 1.6, 'petalWidth': 0.2, 'species': 'setosa'}

{'sepalLength': 4.8, 'sepalWidth': 3.1, 'petalLength': 1.6, 'petalWidth': 0.2, 'species': 'setosa'}

{'sepalLength': 5.4, 'sepalWidth': 3.4, 'petalLength': 1.5, 'petalWidth': 0.4, 'species': 'setosa'}

{'sepalLength': 5.2, 'sepalWidth': 4.1, 'petalLength': 1.5, 'petalWidth': 0.1, 'species': 'setosa'}

{'sepalLength': 5.5, 'sepalWidth': 4.2, 'petalLength': 1.4, 'petalWidth': 0.2, 'species': 'setosa'}

{'sepalLength': 4.9, 'sepalWidth': 3.1, 'petalLength': 1.5, 'petalWidth': 0.2, 'species': 'setosa'}

{'sepalLength': 5.0, 'sepalWidth': 3.2, 'petalLength': 1.2, 'petalWidth': 0.2, 'species': 'setosa'}

{'sepalLength': 5.5, 'sepalWidth': 3.5, 'petalLength': 1.3, 'petalWidth': 0.2, 'species': 'setosa'}

{'sepalLength': 4.9, 'sepalWidth': 3.6, 'petalLength': 1.4, 'petalWidth': 0.1, 'species': 'setosa'}

{'sepalLength': 4.4, 'sepalWidth': 3.0, 'petalLength': 1.3, 'petalWidth': 0.2, 'species': 'setosa'}

{'sepalLength': 5.1, 'sepalWidth': 3.4, 'petalLength': 1.5, 'petalWidth': 0.2, 'species': 'setosa'}

{'sepalLength': 5.0, 'sepalWidth': 3.5, 'petalLength': 1.3, 'petalWidth': 0.3, 'species': 'setosa'}

{'sepalLength': 4.5, 'sepalWidth': 2.3, 'petalLength': 1.3, 'petalWidth': 0.3, 'species': 'setosa'}

{'sepalLength': 4.4, 'sepalWidth': 3.2, 'petalLength': 1.3, 'petalWidth': 0.2, 'species': 'setosa'}

{'sepalLength': 5.0, 'sepalWidth': 3.5, 'petalLength': 1.6, 'petalWidth': 0.6, 'species': 'setosa'}

{'sepalLength': 5.1, 'sepalWidth': 3.8, 'petalLength': 1.9, 'petalWidth': 0.4, 'species': 'setosa'}

{'sepalLength': 4.8, 'sepalWidth': 3.0, 'petalLength': 1.4, 'petalWidth': 0.3, 'species': 'setosa'}

{'sepalLength': 5.1, 'sepalWidth': 3.8, 'petalLength': 1.6, 'petalWidth': 0.2, 'species': 'setosa'}

{'sepalLength': 4.6, 'sepalWidth': 3.2, 'petalLength': 1.4, 'petalWidth': 0.2, 'species': 'setosa'}

{'sepalLength': 5.3, 'sepalWidth': 3.7, 'petalLength': 1.5, 'petalWidth': 0.2, 'species': 'setosa'}

{'sepalLength': 5.0, 'sepalWidth': 3.3, 'petalLength': 1.4, 'petalWidth': 0.2, 'species': 'setosa'}

{'sepalLength': 6.9, 'sepalWidth': 3.1, 'petalLength': 4.9, 'petalWidth': 1.5, 'species': 'versicolor'}

{'sepalLength': 5.5, 'sepalWidth': 2.3, 'petalLength': 4.0, 'petalWidth': 1.3, 'species': 'versicolor'}

{'sepalLength': 6.5, 'sepalWidth': 2.8, 'petalLength': 4.6, 'petalWidth': 1.5, 'species': 'versicolor'}

{'sepalLength': 5.7, 'sepalWidth': 2.8, 'petalLength': 4.5, 'petalWidth': 1.3, 'species': 'versicolor'}

{'sepalLength': 6.3, 'sepalWidth': 3.3, 'petalLength': 4.7, 'petalWidth': 1.6, 'species': 'versicolor'}

{'sepalLength': 4.9, 'sepalWidth': 2.4, 'petalLength': 3.3, 'petalWidth': 1.0, 'species': 'versicolor'}

{'sepalLength': 6.6, 'sepalWidth': 2.9, 'petalLength': 4.6, 'petalWidth': 1.3, 'species': 'versicolor'}

{'sepalLength': 5.2, 'sepalWidth': 2.7, 'petalLength': 3.9, 'petalWidth': 1.4, 'species': 'versicolor'}

{'sepalLength': 5.0, 'sepalWidth': 2.0, 'petalLength': 3.5, 'petalWidth': 1.0, 'species': 'versicolor'}

{'sepalLength': 5.9, 'sepalWidth': 3.0, 'petalLength': 4.2, 'petalWidth': 1.5, 'species': 'versicolor'}

{'sepalLength': 6.0, 'sepalWidth': 2.2, 'petalLength': 4.0, 'petalWidth': 1.0, 'species': 'versicolor'}

{'sepalLength': 6.1, 'sepalWidth': 2.9, 'petalLength': 4.7, 'petalWidth': 1.4, 'species': 'versicolor'}

{'sepalLength': 5.6, 'sepalWidth': 2.9, 'petalLength': 3.6, 'petalWidth': 1.3, 'species': 'versicolor'}

{'sepalLength': 6.7, 'sepalWidth': 3.1, 'petalLength': 4.4, 'petalWidth': 1.4, 'species': 'versicolor'}

{'sepalLength': 5.6, 'sepalWidth': 3.0, 'petalLength': 4.5, 'petalWidth': 1.5, 'species': 'versicolor'}

{'sepalLength': 5.8, 'sepalWidth': 2.7, 'petalLength': 4.1, 'petalWidth': 1.0, 'species': 'versicolor'}

{'sepalLength': 6.2, 'sepalWidth': 2.2, 'petalLength': 4.5, 'petalWidth': 1.5, 'species': 'versicolor'}

{'sepalLength': 5.6, 'sepalWidth': 2.5, 'petalLength': 3.9, 'petalWidth': 1.1, 'species': 'versicolor'}

{'sepalLength': 5.9, 'sepalWidth': 3.2, 'petalLength': 4.8, 'petalWidth': 1.8, 'species': 'versicolor'}

{'sepalLength': 6.1, 'sepalWidth': 2.8, 'petalLength': 4.0, 'petalWidth': 1.3, 'species': 'versicolor'}

{'sepalLength': 6.3, 'sepalWidth': 2.5, 'petalLength': 4.9, 'petalWidth': 1.5, 'species': 'versicolor'}

{'sepalLength': 6.1, 'sepalWidth': 2.8, 'petalLength': 4.7, 'petalWidth': 1.2, 'species': 'versicolor'}

{'sepalLength': 6.4, 'sepalWidth': 2.9, 'petalLength': 4.3, 'petalWidth': 1.3, 'species': 'versicolor'}

{'sepalLength': 6.6, 'sepalWidth': 3.0, 'petalLength': 4.4, 'petalWidth': 1.4, 'species': 'versicolor'}

{'sepalLength': 6.8, 'sepalWidth': 2.8, 'petalLength': 4.8, 'petalWidth': 1.4, 'species': 'versicolor'}

{'sepalLength': 6.7, 'sepalWidth': 3.0, 'petalLength': 5.0, 'petalWidth': 1.7, 'species': 'versicolor'}

{'sepalLength': 6.0, 'sepalWidth': 2.9, 'petalLength': 4.5, 'petalWidth': 1.5, 'species': 'versicolor'}

{'sepalLength': 5.7, 'sepalWidth': 2.6, 'petalLength': 3.5, 'petalWidth': 1.0, 'species': 'versicolor'}

{'sepalLength': 5.5, 'sepalWidth': 2.4, 'petalLength': 3.8, 'petalWidth': 1.1, 'species': 'versicolor'}

{'sepalLength': 5.5, 'sepalWidth': 2.4, 'petalLength': 3.7, 'petalWidth': 1.0, 'species': 'versicolor'}

{'sepalLength': 5.8, 'sepalWidth': 2.7, 'petalLength': 3.9, 'petalWidth': 1.2, 'species': 'versicolor'}

{'sepalLength': 6.0, 'sepalWidth': 2.7, 'petalLength': 5.1, 'petalWidth': 1.6, 'species': 'versicolor'}

{'sepalLength': 5.4, 'sepalWidth': 3.0, 'petalLength': 4.5, 'petalWidth': 1.5, 'species': 'versicolor'}

{'sepalLength': 6.0, 'sepalWidth': 3.4, 'petalLength': 4.5, 'petalWidth': 1.6, 'species': 'versicolor'}

{'sepalLength': 6.7, 'sepalWidth': 3.1, 'petalLength': 4.7, 'petalWidth': 1.5, 'species': 'versicolor'}

{'sepalLength': 6.3, 'sepalWidth': 2.3, 'petalLength': 4.4, 'petalWidth': 1.3, 'species': 'versicolor'}

{'sepalLength': 5.6, 'sepalWidth': 3.0, 'petalLength': 4.1, 'petalWidth': 1.3, 'species': 'versicolor'}

{'sepalLength': 5.5, 'sepalWidth': 2.5, 'petalLength': 4.0, 'petalWidth': 1.3, 'species': 'versicolor'}

{'sepalLength': 5.5, 'sepalWidth': 2.6, 'petalLength': 4.4, 'petalWidth': 1.2, 'species': 'versicolor'}

{'sepalLength': 6.1, 'sepalWidth': 3.0, 'petalLength': 4.6, 'petalWidth': 1.4, 'species': 'versicolor'}

{'sepalLength': 5.8, 'sepalWidth': 2.6, 'petalLength': 4.0, 'petalWidth': 1.2, 'species': 'versicolor'}

{'sepalLength': 5.0, 'sepalWidth': 2.3, 'petalLength': 3.3, 'petalWidth': 1.0, 'species': 'versicolor'}

{'sepalLength': 5.6, 'sepalWidth': 2.7, 'petalLength': 4.2, 'petalWidth': 1.3, 'species': 'versicolor'}

{'sepalLength': 5.7, 'sepalWidth': 3.0, 'petalLength': 4.2, 'petalWidth': 1.2, 'species': 'versicolor'}

{'sepalLength': 5.7, 'sepalWidth': 2.9, 'petalLength': 4.2, 'petalWidth': 1.3, 'species': 'versicolor'}

{'sepalLength': 6.2, 'sepalWidth': 2.9, 'petalLength': 4.3, 'petalWidth': 1.3, 'species': 'versicolor'}

{'sepalLength': 5.1, 'sepalWidth': 2.5, 'petalLength': 3.0, 'petalWidth': 1.1, 'species': 'versicolor'}

{'sepalLength': 5.7, 'sepalWidth': 2.8, 'petalLength': 4.1, 'petalWidth': 1.3, 'species': 'versicolor'}

{'sepalLength': 6.3, 'sepalWidth': 3.3, 'petalLength': 6.0, 'petalWidth': 2.5, 'species': 'virginica'}

{'sepalLength': 5.8, 'sepalWidth': 2.7, 'petalLength': 5.1, 'petalWidth': 1.9, 'species': 'virginica'}

{'sepalLength': 7.1, 'sepalWidth': 3.0, 'petalLength': 5.9, 'petalWidth': 2.1, 'species': 'virginica'}

{'sepalLength': 6.3, 'sepalWidth': 2.9, 'petalLength': 5.6, 'petalWidth': 1.8, 'species': 'virginica'}

{'sepalLength': 6.5, 'sepalWidth': 3.0, 'petalLength': 5.8, 'petalWidth': 2.2, 'species': 'virginica'}

{'sepalLength': 7.6, 'sepalWidth': 3.0, 'petalLength': 6.6, 'petalWidth': 2.1, 'species': 'virginica'}

{'sepalLength': 4.9, 'sepalWidth': 2.5, 'petalLength': 4.5, 'petalWidth': 1.7, 'species': 'virginica'}

{'sepalLength': 7.3, 'sepalWidth': 2.9, 'petalLength': 6.3, 'petalWidth': 1.8, 'species': 'virginica'}

{'sepalLength': 6.7, 'sepalWidth': 2.5, 'petalLength': 5.8, 'petalWidth': 1.8, 'species': 'virginica'}

{'sepalLength': 7.2, 'sepalWidth': 3.6, 'petalLength': 6.1, 'petalWidth': 2.5, 'species': 'virginica'}

{'sepalLength': 6.5, 'sepalWidth': 3.2, 'petalLength': 5.1, 'petalWidth': 2.0, 'species': 'virginica'}

{'sepalLength': 6.4, 'sepalWidth': 2.7, 'petalLength': 5.3, 'petalWidth': 1.9, 'species': 'virginica'}

{'sepalLength': 6.8, 'sepalWidth': 3.0, 'petalLength': 5.5, 'petalWidth': 2.1, 'species': 'virginica'}

{'sepalLength': 5.7, 'sepalWidth': 2.5, 'petalLength': 5.0, 'petalWidth': 2.0, 'species': 'virginica'}

{'sepalLength': 5.8, 'sepalWidth': 2.8, 'petalLength': 5.1, 'petalWidth': 2.4, 'species': 'virginica'}

{'sepalLength': 6.4, 'sepalWidth': 3.2, 'petalLength': 5.3, 'petalWidth': 2.3, 'species': 'virginica'}

{'sepalLength': 6.5, 'sepalWidth': 3.0, 'petalLength': 5.5, 'petalWidth': 1.8, 'species': 'virginica'}

{'sepalLength': 7.7, 'sepalWidth': 3.8, 'petalLength': 6.7, 'petalWidth': 2.2, 'species': 'virginica'}

{'sepalLength': 7.7, 'sepalWidth': 2.6, 'petalLength': 6.9, 'petalWidth': 2.3, 'species': 'virginica'}

{'sepalLength': 6.0, 'sepalWidth': 2.2, 'petalLength': 5.0, 'petalWidth': 1.5, 'species': 'virginica'}

{'sepalLength': 6.9, 'sepalWidth': 3.2, 'petalLength': 5.7, 'petalWidth': 2.3, 'species': 'virginica'}

{'sepalLength': 5.6, 'sepalWidth': 2.8, 'petalLength': 4.9, 'petalWidth': 2.0, 'species': 'virginica'}

{'sepalLength': 7.7, 'sepalWidth': 2.8, 'petalLength': 6.7, 'petalWidth': 2.0, 'species': 'virginica'}

{'sepalLength': 6.3, 'sepalWidth': 2.7, 'petalLength': 4.9, 'petalWidth': 1.8, 'species': 'virginica'}

{'sepalLength': 6.7, 'sepalWidth': 3.3, 'petalLength': 5.7, 'petalWidth': 2.1, 'species': 'virginica'}

{'sepalLength': 7.2, 'sepalWidth': 3.2, 'petalLength': 6.0, 'petalWidth': 1.8, 'species': 'virginica'}

{'sepalLength': 6.2, 'sepalWidth': 2.8, 'petalLength': 4.8, 'petalWidth': 1.8, 'species': 'virginica'}

{'sepalLength': 6.1, 'sepalWidth': 3.0, 'petalLength': 4.9, 'petalWidth': 1.8, 'species': 'virginica'}

{'sepalLength': 6.4, 'sepalWidth': 2.8, 'petalLength': 5.6, 'petalWidth': 2.1, 'species': 'virginica'}

{'sepalLength': 7.2, 'sepalWidth': 3.0, 'petalLength': 5.8, 'petalWidth': 1.6, 'species': 'virginica'}

{'sepalLength': 7.4, 'sepalWidth': 2.8, 'petalLength': 6.1, 'petalWidth': 1.9, 'species': 'virginica'}

{'sepalLength': 7.9, 'sepalWidth': 3.8, 'petalLength': 6.4, 'petalWidth': 2.0, 'species': 'virginica'}

{'sepalLength': 6.4, 'sepalWidth': 2.8, 'petalLength': 5.6, 'petalWidth': 2.2, 'species': 'virginica'}

{'sepalLength': 6.3, 'sepalWidth': 2.8, 'petalLength': 5.1, 'petalWidth': 1.5, 'species': 'virginica'}

{'sepalLength': 6.1, 'sepalWidth': 2.6, 'petalLength': 5.6, 'petalWidth': 1.4, 'species': 'virginica'}

{'sepalLength': 7.7, 'sepalWidth': 3.0, 'petalLength': 6.1, 'petalWidth': 2.3, 'species': 'virginica'}

{'sepalLength': 6.3, 'sepalWidth': 3.4, 'petalLength': 5.6, 'petalWidth': 2.4, 'species': 'virginica'}

{'sepalLength': 6.4, 'sepalWidth': 3.1, 'petalLength': 5.5, 'petalWidth': 1.8, 'species': 'virginica'}

{'sepalLength': 6.0, 'sepalWidth': 3.0, 'petalLength': 4.8, 'petalWidth': 1.8, 'species': 'virginica'}

{'sepalLength': 6.9, 'sepalWidth': 3.1, 'petalLength': 5.4, 'petalWidth': 2.1, 'species': 'virginica'}

{'sepalLength': 6.7, 'sepalWidth': 3.1, 'petalLength': 5.6, 'petalWidth': 2.4, 'species': 'virginica'}

{'sepalLength': 6.9, 'sepalWidth': 3.1, 'petalLength': 5.1, 'petalWidth': 2.3, 'species': 'virginica'}

{'sepalLength': 5.8, 'sepalWidth': 2.7, 'petalLength': 5.1, 'petalWidth': 1.9, 'species': 'virginica'}

{'sepalLength': 6.8, 'sepalWidth': 3.2, 'petalLength': 5.9, 'petalWidth': 2.3, 'species': 'virginica'}

{'sepalLength': 6.7, 'sepalWidth': 3.3, 'petalLength': 5.7, 'petalWidth': 2.5, 'species': 'virginica'}

{'sepalLength': 6.7, 'sepalWidth': 3.0, 'petalLength': 5.2, 'petalWidth': 2.3, 'species': 'virginica'}

{'sepalLength': 6.3, 'sepalWidth': 2.5, 'petalLength': 5.0, 'petalWidth': 1.9, 'species': 'virginica'}

{'sepalLength': 6.5, 'sepalWidth': 3.0, 'petalLength': 5.2, 'petalWidth': 2.0, 'species': 'virginica'}

{'sepalLength': 6.2, 'sepalWidth': 3.4, 'petalLength': 5.4, 'petalWidth': 2.3, 'species': 'virginica'}

{'sepalLength': 5.9, 'sepalWidth': 3.0, 'petalLength': 5.1, 'petalWidth': 1.8, 'species': 'virginica'}

­­­­

**XML**

**Description**

XMLs are widely used as configuration formats by different systems, metadata, and data

representation format for services like RSS, SOAP, and many more.

XML is a language with syntactic rules and schemas defined and refined over the years. The

most

import components of an XML are as follows:

• Tag: A markup construct denoted by strings enclosed with angled braces (“<” and “>”).

• Content: Any data not marked within the tag syntax is the content of the XML file/object.

• Element: A logical construct of an XML. An element may be defined with a start and

an end tag with or without attributes, or it may be simply an empty tag.

• Attribute: Key-value pairs that represent the properties or attributes of the element

in consideration. These are enclosed within a start or an empty tag.

**Code:**

import xml.etree.ElementTree as et

mt=et.parse('/content/drive/MyDrive/Academics/5th sem/ml lab/food.xml')

mr=mt.getroot()

print(mr)

print(mr[1])

**Output:**

<Element 'food' at 0x7fc79bd20cb0>

<Element 'price' at 0x7fc79bd6b350>

**Code:**

import xml.etree.ElementTree as et

data='''<food>

<item name="breakfast">idly</item>

<price>$2.5</price>

</food>'''

mr=et.fromstring(data)

print("root :",mr)

**Output:**

root : <Element 'food' at 0x7fc799f2f950>

**HTML**

The Hyper Text Markup Language (HTML) is a markup language similar to XML. HTML is

mainly used by web browsers and similar applications to render web pages for consumption.

HTML defines rules and structure to describe web pages using markup. The following are

standard components of an HTML page:

• Element: Logical constructs that form the basic building blocks of an HTML page

• Tags: A markup construct defined by angled braces (< and >). Some of the important

tags are:

• <html></html>: This pair of tags contains the whole of HTML document.

It marks the start and end of the HTML page.

• <body></body>: This pair of tags contains the main content of the HTML page

rendered by the browser.

Web Scraping

Web scraping is a technique to scrape or extract data from the web, particularly from web

pages. Web scraping may involve manually copying the data or using automation to crawl,

parse, and extract information from web pages.

• Crawl: A bot or a web crawler is designed to query a web server using the required

set of URLs to fetch the web pages.

• Scrape: Once the raw web page has been fetched, the next task is to extract

information from it. The task of scraping involves utilizing techniques like regular

expressions, extraction based on XPath, or specific tags and so on to narrow down to

the required information on the page.

**Code:**

import requests

from bs4 import BeautifulSoup

url = 'https://www.worldometers.info/coronavirus/countries-where-coronavirus-has-spread/'

page = requests.get(url)

soup = BeautifulSoup(page.text, 'html.parser')

data = []

data\_iterator = iter(soup.find\_all('td'))

while True:

    try:

        country = next(data\_iterator).text

        confirmed = next(data\_iterator).text

        deaths = next(data\_iterator).text

        continent = next(data\_iterator).text

        data.append((

            country,

            int(confirmed.replace(",", "")),

            int(deaths.replace(",","")),

            continent

        ))

    except StopIteration:

        break

data.sort(key = lambda row: row[1], reverse = True)

data

**Output:**

[('United States', 97157866, 1076013, 'North America'), ('India', 44502694, 528165, 'Asia'), ('France', 34722711, 154529, 'Europe'), ('Brazil', 34580412, 684951, 'South America'), ('Germany', 32452250, 148299, 'Europe'), ('South Korea', 24041825, 27498, 'Asia'),

('United Kingdom', 23554519, 189026, 'Europe'), ('Italy', 22054443, 176242, 'Europe'), ('Japan (+Diamond Princess)', 20157704, 42650, 'Asia'), ('Russia', 20113098, 385429, 'Europe'), ('Turkey', 16829941, 100979, 'Asia'), ('Spain', 13367647, 113130, 'Europe'), ('Vietnam', 11441626, 43130, 'Asia'), ('Australia', 10119203, 14432, 'Australia/Oceania'), ('Argentina', 9697763, 129830, 'South America'), ('Netherlands', 8396979, 22613, 'Europe'), ('Iran', 7539698, 144199, 'Asia'), ('Mexico', 7059348, 329761, 'North America'), ('Indonesia', 6394340, 157787, 'Asia'), ('Colombia', 6304317, 141708, 'South America'), ('Poland', 6213262, 117252, 'Europe'), ('Taiwan', 5707688, 10312, 'Asia'), ('Portugal', 5444993, 24924, 'Europe'), ('Ukraine', 5072533, 108885, 'Europe'), ('Austria', 4955082, 19486, 'Europe'), ('Malaysia', 4806954, 36285, 'Asia'), ('Greece', 4804982, 32757, 'Europe'), ('North Korea', 4772813, 74, 'Asia'), ('Thailand', 4668244, 32557, 'Asia'), ('Israel', 4643996, 11657, 'Asia'), ('Chile', 4566548, 60812, 'South America'), ('Belgium', 4497199, 32575, 'Europe'), ('Canada', 4197701, 44347, 'North America'), ('Peru', 4126021, 216125, 'South America'), ('Czech Republic (Czechia)', 4059441, 40910, 'Europe'), ('Switzerland', 4040280, 14157, 'Europe'), ('South Africa', 4014485, 102129, 'Africa'), ('Philippines', 3908295, 62342, 'Asia'), ('Romania', 3241772, 66856, 'Europe'), ('Denmark', 3098447, 6982, 'Europe'), ('Sweden', 2573548, 19974, 'Europe'), ('Iraq', 2458509, 25348, 'Asia'), ('Serbia', 2318677, 16829, 'Europe'), ('Hungary', 2058847, 47367, 'Europe'), ('Bangladesh', 2015308, 29334, 'Asia'), ('Singapore', 1861390, 1602, 'Asia'), ('Slovakia', 1837272, 20411, 'Europe'), ('New Zealand', 1760113, 2836, 'Australia/Oceania'), ('Jordan', 1742256, 14114, 'Asia'), ('Georgia', 1735682, 16889, 'Asia'), ('Hong Kong', 1659912, 9810, 'Asia'), ('Ireland', 1659034, 7829, 'Europe'), ('Pakistan', 1571098, 30599, 'Asia'), ('Norway', 1461119, 4004, 'Europe'), ('Kazakhstan', 1391645, 13688, 'Asia'), ('Finland', 1271516, 5768, 'Europe'), ('Morocco', 1264664, 16276, 'Africa'), ('Bulgaria', 1248200, 37646, 'Europe'), ('Lithuania', 1226685, 9296, 'Europe'), ('Croatia', 1220490, 16806, 'Europe'), ('Lebanon', 1212815, 10647, 'Asia'), ('Tunisia', 1144824, 29238, 'Africa'), ('Slovenia', 1143235, 6794, 'Europe'), ('Guatemala', 1111191, 19679, 'North America'), ('Cuba', 1110918, 8530, 'North America'), ('Bolivia', 1106142, 22217, 'South America'), ('Costa Rica', 1066630, 8893, 'North America'), ('United Arab Emirates', 1020412, 2342, 'Asia'), ('Nepal', 998870, 12015, 'Asia'), ('Ecuador', 998202, 35876, 'South America'), ('Belarus', 994037, 7118, 'Europe'), ('Uruguay', 982846, 7462, 'South America'), ('Panama', 981822, 8480, 'North America'), ('Mongolia', 981200, 2179, 'Asia'), ('Latvia', 907831, 5957, 'Europe'), ('Azerbaijan', 817938, 9857, 'Asia'), ('Saudi Arabia', 814597, 9317, 'Asia'), ('Paraguay', 715806, 19530, 'South America'), ('Bahrain', 674303, 1518, 'Asia'), ('Sri Lanka', 670471, 16731, 'Asia'), ('Kuwait', 657745, 2563, 'Asia'), ('Dominican Republic', 641677, 4384, 'North America'), ('State of Palestine', 620371, 5402, 'Asia'), ('Myanmar', 617056, 19442, 'Asia'), ('Estonia', 598580, 2657, 'Europe'), ('Cyprus', 579899, 1173, 'Asia'), ('Moldova', 579110, 11783, 'Europe'), ('Venezuela', 543811, 5809, 'South America'), ('Egypt', 515645, 24613, 'Africa'), ('Libya', 506898, 6437, 'Africa'), ('Ethiopia', 493340, 7572, 'Africa'), ('Réunion', 467816, 879, 'Africa'), ('Honduras', 455011, 10989, 'North America'), ('Armenia', 439302, 8669, 'Asia'), ('Qatar', 436820, 682, 'Asia'), ('Oman', 397846, 4260, 'Asia'), ('Bosnia and Herzegovina', 397296, 16100, 'Europe'), ('North Macedonia', 341583, 9512, 'Europe'), ('Kenya', 338301, 5674, 'Africa'), ('Zambia', 333234, 4017, 'Africa'), ('Albania', 331053, 3585, 'Europe'), ('Botswana', 325931, 2786, 'Africa'), ('Luxembourg', 288658, 1123, 'Europe'), ('Montenegro', 277440, 2778, 'Europe'), ('Algeria', 270551, 6879, 'Africa'), ('Nigeria', 264450, 3154, 'Africa'), ('Zimbabwe', 256888, 5596, 'Africa'), ('China', 247078, 5226, 'Asia'), ('Uzbekistan', 243893, 1637, 'Asia'), ('Mozambique', 230174, 2221, 'Africa'), ('Brunei ', 223059, 225, 'Asia'), ('Martinique', 219529, 1036, 'North America'), ('Laos', 214982, 757, 'Asia'), ('Kyrgyzstan', 205920, 2991, 'Asia'), ('Iceland', 205009, 213, 'Europe'), ('El Salvador', 201785, 4228, 'North America'), ('Afghanistan', 196182, 7789, 'Asia'), ('Guadeloupe', 191997, 986, 'North America'), ('Maldives', 184966, 308, 'Asia'), ('Trinidad and Tobago', 181421, 4174, 'North America'), ('Uganda', 169396, 3628, 'Africa'), ('Namibia', 169253, 4065, 'Africa'), ('Ghana', 168616, 1459, 'Africa'), ('Jamaica', 150844, 3284, 'North America'), ('Cambodia', 137719, 3056, 'Asia'), ('Rwanda', 132474, 1466, 'Africa'), ('Cameroon', 121652, 1935, 'Africa'), ('Malta', 114283, 803, 'Europe'), ('Angola', 103131, 1917, 'Africa'), ('Barbados', 101899, 556, 'North America'), ('French Guiana', 93837, 409, 'South America'), ('DR Congo', 92751, 1422, 'Africa'), ('Channel Islands', 90153, 199, 'Europe'), ('Senegal', 88199, 1968, 'Africa'), ('Malawi', 87943, 2680, 'Africa'), ("Côte d'Ivoire", 86941, 822, 'Africa'), ('Suriname', 81057, 1384, 'South America'), ('French Polynesia', 76542, 649, 'Australia/Oceania'), ('New Caledonia', 73989, 314, 'Australia/Oceania'), ('Eswatini', 73374, 1422, 'Africa'), ('Guyana', 71192, 1279, 'South America'), ('Belize', 68473, 680, 'North America'), ('Fiji', 68207, 878, 'Australia/Oceania'), ('Madagascar', 66652, 1410, 'Africa'), ('Sudan', 63275, 4961, 'Africa'), ('Mauritania', 62777, 993, 'Africa'), ('Cabo Verde', 62344, 410, 'Africa'), ('Bhutan', 61419, 21, 'Asia'), ('Syria', 57172, 3163, 'Asia'), ('Burundi', 49370, 38, 'Africa'), ('Gabon', 48668, 306, 'Africa'), ('Seychelles', 46358, 169, 'Africa'), ('Andorra', 46113, 155, 'Europe'), ('Curaçao', 45127, 282, 'North America'), ('Papua New Guinea', 44915, 664, 'Australia/Oceania'), ('Aruba', 42914, 227, 'North America'), ('Mauritius', 40342, 1023, 'Africa'), ('Mayotte', 40161, 187, 'Africa'), ('Tanzania', 39168, 845, 'Africa'), ('Togo', 38649, 284, 'Africa'), ('Isle of Man', 38008, 116, 'Europe'), ('Guinea', 37652, 449, 'Africa'), ('Bahamas', 37146, 823, 'North America'), ('Faeroe Islands', 34658, 28, 'Europe'), ('Lesotho', 34287, 704, 'Africa'), ('Haiti', 33658, 851, 'North America'), ('Mali', 32263, 739, 'Africa'), ('Cayman Islands', 30380, 29, 'North America'), ('Saint Lucia', 28894, 391, 'North America'), ('Benin', 27490, 163, 'Africa'), ('Somalia', 27020, 1350, 'Africa'), ('Congo', 24837, 386, 'Africa'), ('Timor-Leste', 23217, 138, 'Asia'), ('Solomon Islands', 21544, 153, 'Australia/Oceania'), ('Burkina Faso', 21128, 387, 'Africa'), ('San Marino', 20500, 118, 'Europe'), ('Gibraltar', 20069, 108, 'Europe'), ('Grenada', 19403, 236, 'North America'), ('Liechtenstein', 19333, 86, 'Europe'), ('Nicaragua', 18491, 225, 'North America'), ('Bermuda', 18019, 148, 'North America'), ('South Sudan', 17823, 138, 'Africa'), ('Tajikistan', 17786, 125, 'Asia'), ('Equatorial Guinea', 16965, 183, 'Africa'), ('Tonga', 16182, 12, 'Australia/Oceania'), ('Samoa', 15889, 29, 'Australia/Oceania'), ('Djibouti', 15690, 189, 'Africa'), ('Marshall Islands', 15177, 17, 'Australia/Oceania'), ('Central African Republic', 14883, 113, 'Africa'), ('Dominica', 14852, 68, 'North America'), ('Monaco', 14436, 57, 'Europe'), ('Gambia', 12311, 371, 'Africa'), ('Greenland', 11971, 21, 'North America'), ('Saint Martin', 11941, 63, 'North America'), ('Yemen', 11932, 2155, 'Asia'), ('Vanuatu', 11908, 14, 'Australia/Oceania'), ('Caribbean Netherlands', 11176, 36, 'North America'), ('Sint Maarten', 10847, 87, 'North America'), ('Eritrea', 10163, 103, 'Africa'), ('Niger', 9931, 312, 'Africa'), ('Antigua and Barbuda', 8974, 145, 'North America'), ('Guinea-Bissau', 8796, 175, 'Africa'), ('Comoros', 8455, 161, 'Africa'), ('Liberia', 7929, 294, 'Africa'), ('Micronesia', 7856, 27, 'Australia/Oceania'), ('Sierra Leone', 7749, 126, 'Africa'), ('Chad', 7558, 193, 'Africa'), ('British Virgin Islands', 7305, 64, 'North America'), ('St. Vincent & Grenadines', 7112, 115, 'North America'), ('Saint Kitts & Nevis', 6532, 46, 'North America'), ('Cook Islands', 6386, 1, 'Australia/Oceania'), ('Turks and Caicos', 6372, 36, 'North America'), ('Sao Tome & Principe', 6193, 76, 'Africa'), ('Palau', 5430, 6, 'Australia/Oceania'), ('Saint Barthelemy', 5263, 6, 'North America'), ('Nauru', 4611, 1, 'Australia/Oceania'), ('Anguilla', 3851, 12, 'North America'), ('Kiribati', 3430, 13, 'Australia/Oceania'), ('Saint Pierre & Miquelon', 3131, 1, 'North America'), ('Falkland Islands', 1886, 0, 'South America'), ('Montserrat', 1276, 8, 'North America'), ('Macao', 793, 6, 'Asia'), ('Wallis & Futuna', 761, 7, 'Australia/Oceania'), ('Niue', 77, 0, 'Australia/Oceania'), ('Saint Helena', 52, 0, 'Africa'), ('Holy See', 29, 0, 'Europe'), ('Tuvalu', 20, 0, 'Australia/Oceania'), ('Western Sahara', 10, 1, 'Africa'), ('MS Zaandam', 9, 2, '')]

**WEEK-2**

**AIM:**

Write a program that extracts the words (features) used in a sentence

**Description:**

1.Dealing with structured data attributes like numeric or categorical variables are usually

not as challenging as unstructured attributes like text and images.

2. In case of unstructured data like text documents, the first

challenge is dealing with the unpredictable nature of the syntax, format, and content of

the documents,which make it a challenge to extract useful information for building

models.

3.The second challenge is transforming these textual representations into numeric

representations that can be understood by MachineLearning algorithms.

there are two aspects to execute feature engineering on text data.

• Pre-processing and normalizing text

• Feature extraction and engineering

Bag of Words Model

This is perhaps one of the simplest yet effective schemes of vectorizing features from

unstructured text. The core principle of this model is to convert text documents into

numeric vectors. The dimension or size of each vector is N where N indicates all

possible distinct words across the corpus of documents.

**Code:**

import pandas as pd

import numpy as np

import re

import nltk

corpus = ['The sky is blue and beautiful.',

'Love this blue and beautiful sky!',

'The quick brown fox jumps over the lazy dog.',

'The brown fox is quick and the blue dog is lazy!',

'The sky is very blue and the sky is very beautiful today',

'The dog is lazy but the brown fox is quick!'

]

labels = ['weather', 'weather', 'animals', 'animals', 'weather', 'animals']

corpus = np.array(corpus)

corpus\_df = pd.DataFrame({'Document': corpus,

'Category': labels})

corpus\_df = corpus\_df[['Document', 'Category']]

corpus\_df

**Output:**

Document Category

0 The sky is blue and beautiful. weather

1 Love this blue and beautiful sky! weather

2 The quick brown fox jumps over the lazy dog. animals

3 The brown fox is quick and the blue dog is lazy! animals

4 The sky is very blue and the sky is very beaut... weather

5 The dog is lazy but the brown fox is quick! animals

CodeText

**Normalization**

**Code:**

nltk.download('stopwords')

**Output:**

[nltk\_data] Downloading package stopwords to /root/nltk\_data...

[nltk\_data] Unzipping corpora/stopwords.zip.

True

**Code:**

from nltk.corpus import stopwords

wpt = nltk.WordPunctTokenizer()

stop\_words =stopwords.words('english')

def normalize\_document(doc):

  doc = re.sub(r'[^a-zA-Z0-9\s]', '', doc, re.I)

  doc = doc.lower()

  doc = doc.strip()

  tokens = wpt.tokenize(doc)

  filtered\_tokens = [token for token in tokens if token not in stop\_words]

  doc = ' '.join(filtered\_tokens)

  return doc

normalize\_corpus = np.vectorize(normalize\_document)

norm\_corpus = normalize\_corpus(corpus)

norm\_corpus

**Output:**

array(['sky blue beautiful', 'love blue beautiful sky', 'quick brown fox jumps lazy dog', 'brown fox quick blue dog lazy', 'sky blue sky beautiful today', 'dog lazy brown fox quick'], dtype='<U30')

**Bag of words model**

from sklearn.feature\_extraction.text import CountVectorizer

cv = CountVectorizer(min\_df=0., max\_df=1.)

cv\_matrix = cv.fit\_transform(norm\_corpus)

cv\_matrix = cv\_matrix.toarray()

cv\_matrix

vocab = cv.get\_feature\_names()

pd.DataFrame(cv\_matrix, columns=vocab)

**Output:**

/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function get\_feature\_names is deprecated; get\_feature\_names is deprecated in 1.0 and will be removed in 1.2. Please use get\_feature\_names\_out instead.

warnings.warn(msg, category=FutureWarning)

|  | **beautiful** | **blue** | **brown** | **dog** | **fox** | **jumps** | **lazy** | **love** | **quick** | **sky** | **today** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| **1** | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| **2** | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| **3** | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| **4** | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 1 |
| **5** | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |

**WEEK-3**

**AIM:**

Write a program for edge detection to extract edge based features from a sample

image.

**Description:**

Another very popular format of unstructured data is images. Sound and visual data in

the form of images,video, and audio are very popular sources of data which pose a lot

of challenge to data scientists in terms of processing, storage, feature extraction and

modeling.

Due to the unstructured

nature of data, it is not possible to directly use images for training models.

The scikit-image (skimage) library is an excellent framework consisting of several useful

interfaces and algorithms for image processing and feature extraction.

**Code:**

from google.colab import drive

drive.mount('/content/drive')

import skimage

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from skimage import io

%matplotlib inline

cat = io.imread('/content/drive/MyDrive/Academics/5th sem/ml lab/cat.jpg')

dog = io.imread('/content/drive/MyDrive/Academics/5th sem/ml lab/dog.jpg')

df = pd.DataFrame(['Cat', 'Dog'], columns=['Image'])

from skimage.color import rgb2gray

cgs = rgb2gray(cat)

dgs = rgb2gray(dog)

from skimage.feature import canny

cat\_edges = canny(cgs, sigma=3)

dog\_edges = canny(dgs, sigma=3)

fig = plt.figure(figsize = (8,4))

ax1 = fig.add\_subplot(1,2, 1)

ax1.imshow(cat\_edges, cmap='binary')

ax2 = fig.add\_subplot(1,2, 2)

ax2.imshow(dog\_edges, cmap='binary')

**Output:**

<matplotlib.image.AxesImage at 0x7f545ac50d10>



**WEEK-4**

**AIM:**

Write a program to extract SURF/SIFT feature descriptors from a sample image.

**Description:**

The SURF method (Speeded Up Robust Features) is a fast and robust algorithm for local, similarity invariant representation and comparison of images. The main interest of the SURF approach lies in its fast computation of operators using box filters, thus enabling real-time applications such as tracking and object recognition.

SIFT, which stands for Scale Invariant Feature Transform, is a method for extracting feature vectors that describe local patches of an image. Not only are these feature vectors scale-invariant, but they are also invariant to translation, rotation, and illumination. Pretty much the holy grail for a descriptor.

**Code:**

from google.colab import drive

drive.mount('/content/drive')

**Output:**

Mounted at /content/drive

**Code:** !pip install mahotas

**Output:**

Looking in indexes: <https://pypi.org/simple>, <https://us-python.pkg.dev/colab-wheels/public/simple/>

Collecting mahotas

Downloading mahotas-1.4.13-cp37-cp37m-manylinux\_2\_12\_x86\_64.manylinux2010\_x86\_64.whl (5.7 MB)

|████████████████████████████████| 5.7 MB 7.7 MB/s

Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from mahotas) (1.21.6)

Installing collected packages: mahotas

Successfully installed mahotas-1.4.13

**Code [SURF];**

import skimage

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from skimage import io

%matplotlib inline

cat = io.imread('/content/drive/MyDrive/Academics/5th sem/ml lab/cat.jpg')

dog = io.imread('/content/drive/MyDrive/Academics/5th sem/ml lab/dog.jpg')

df = pd.DataFrame(['Cat', 'Dog'], columns=['Image'])

from mahotas.features import surf

import mahotas as mh

dog\_mh = mh.colors.rgb2gray(dog)

dog\_surf = surf.surf(dog\_mh, nr\_octaves=8, nr\_scales=16, initial\_step\_size=1,

threshold=0.1, max\_points=54)

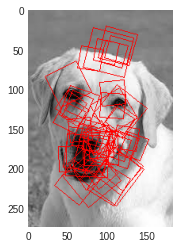
fig = plt.figure(figsize = (10,4))

ax2 = fig.add\_subplot(1,2, 1)

ax2.imshow(surf.show\_surf(dog\_mh, dog\_surf))

**Output:**

<matplotlib.image.AxesImage at 0x7f54573994d0>



**Code[SIFT]**

cat\_mh = mh.colors.rgb2gray(cat)

cat\_surf = surf.surf(cat\_mh, nr\_octaves=8, nr\_scales=16, initial\_step\_size=1,

threshold=0.1, max\_points=54)

fig = plt.figure(figsize = (10,4))

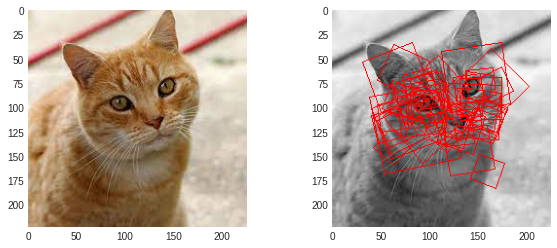
ax1=fig.add\_subplot(1,2,1)

ax1.imshow(cat)

ax2 = fig.add\_subplot(1,2,2)

ax2.imshow(surf.show\_surf(cat\_mh, cat\_surf))

matplotlib.image.AxesImage at 0x7f5455be10d0>



dog\_mh = mh.colors.rgb2gray(dog)

dog\_surf = surf.surf(dog\_mh, nr\_octaves=8, nr\_scales=16, initial\_step\_size=1,

threshold=0.1, max\_points=54)

fig = plt.figure(figsize = (10,4))

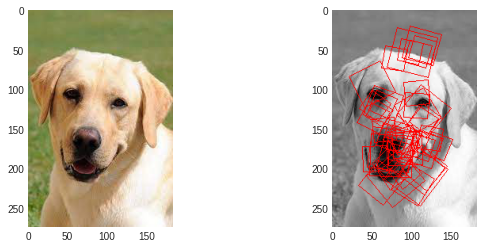
ax1=fig.add\_subplot(1,2,1)

ax1.imshow(dog)

ax2 = fig.add\_subplot(1,2,2)

ax2.imshow(surf.show\_surf(dog\_mh, dog\_surf))

matplotlib.image.AxesImage at 0x7f5454328410>



**WEEK-5**

**AIM:**

Write a program to perform Exploratory Data Analysis on real time datasets using the following approaches:

a) Univariate Analysis

b) Multivariate Analysis

c) Visualization using correlation matrix

**Description:**

Univariate analysis :Uni means one and variate means variable, so in univariate analysis, there is only one dependable variable. The objective of univariate analysis is to derive the data, define and summarize it, and analyze the pattern present in it. In a dataset, it explores each variable separately. It is possible for two kinds of variables- Categorical and Numerical.

Bivariate analysis: Bi means two and variate means variable, so here there are two variables. The analysis is related to cause and the relationship between the two variables.

Multivariate analysis: is required when more than two variables have to be analyzed simultaneously. It is a tremendously hard task for the human brain to visualize a relationship among 4 variables in a graph and thus multivariate analysis is used to study more complex sets of data. Types of Multivariate Analysis include Cluster Analysis, Factor Analysis, Multiple Regression Analysis, Principal Component Analysis, etc. More than 20 different ways to perform multivariate analysis exist and which one to choose depends upon the type of data and the end goal to achieve.

**Code**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

ds=pd.read\_csv('/content/drive/MyDrive/datasets/Iris.csv')

ds.head()

Id SepalLengthCm SepalWidthCm PetalLengthCm PetalWidthCm Species

0 1 5.1 3.5 1.4 0.2 Iris-setosa

1 2 4.9 3.0 1.4 0.2 Iris-setosa

2 3 4.7 3.2 1.3 0.2 Iris-setosa

3 4 4.6 3.1 1.5 0.2 Iris-setosa

4 5 5.0 3.6 1.4 0.2 Iris-setosa

print(ds.describe())

Id SepalLengthCm SepalWidthCm PetalLengthCm PetalWidthCm

count 150.000000 150.000000 150.000000 150.000000 150.000000

mean 75.500000 5.843333 3.054000 3.758667 1.198667

std 43.445368 0.828066 0.433594 1.764420 0.763161

min 1.000000 4.300000 2.000000 1.000000 0.100000

25% 38.250000 5.100000 2.800000 1.600000 0.300000

50% 75.500000 5.800000 3.000000 4.350000 1.300000

75% 112.750000 6.400000 3.300000 5.100000 1.800000

max 150.000000 7.900000 4.400000 6.900000 2.500000

ds.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 150 entries, 0 to 149

Data columns (total 6 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Id 150 non-null int64

1 SepalLengthCm 150 non-null float64

2 SepalWidthCm 150 non-null float64

3 PetalLengthCm 150 non-null float64

4 PetalWidthCm 150 non-null float64

5 Species 150 non-null object

dtypes: float64(4), int64(1), object(1)

memory usage: 7.2+ KB

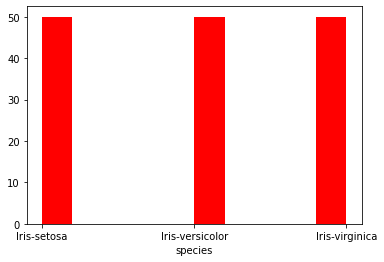
#Numerical Data

#Histogram

plt.hist(ds['Species'],color="red")

plt.xlabel('species')

ext(0.5, 0, 'species')

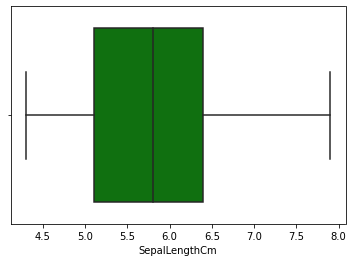


sns.boxplot(ds['SepalLengthCm'], color="green")

/usr/local/lib/python3.7/dist-packages/seaborn/\_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.

FutureWarning

<matplotlib.axes.\_subplots.AxesSubplot at 0x7f95128f4d90>

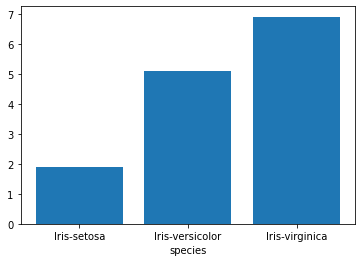


#barplot

plt.bar(ds['Species'],ds['PetalLengthCm'])

plt.xlabel('species')

Text(0.5, 0, 'species')



ds['Species'].value\_counts().plot.bar()

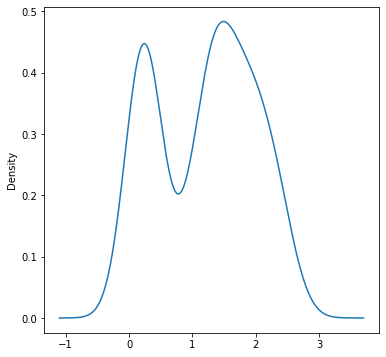
<matplotlib.axes.\_subplots.AxesSubplot at 0x7fb286d85110>



plt.figure(figsize=(6,6))

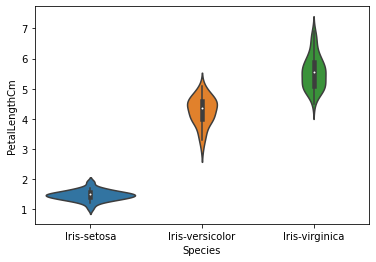
ds['PetalWidthCm'].plot(kind="density")

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fb286af0750>



sns.violinplot(x='Species',y='PetalLengthCm',size=7,data=ds)

<matplotlib.axes.\_subplots.AxesSubplot at 0x7f951268b110>



n=ds["Species"].unique()

print(n)

['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']

v=ds['Species'].value\_counts()

x=(np.array(v)).tolist()

print(x)

[50, 50, 50]

plt.pie(x,labels=n,autopct='%.0f%%')

([<matplotlib.patches.Wedge at 0x7f95120c3590>,

<matplotlib.patches.Wedge at 0x7f95120c3c90>,

<matplotlib.patches.Wedge at 0x7f951204e550>],

[Text(0.5499999702695115, 0.9526279613277875, 'Iris-setosa'),

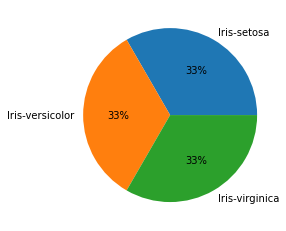
Text(-1.0999999999999954, -1.0298943258065002e-07, 'Iris-versicolor'),

Text(0.5500001486524352, -0.9526278583383436, 'Iris-virginica')],

[Text(0.2999999837833699, 0.5196152516333385, '33%'),

Text(-0.5999999999999974, -5.6176054134900006e-08, '33%'),

Text(0.30000008108314646, -0.5196151954572783, '33%')])



# **Bivariate analysis And Multivariate Analysis**

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

df=pd.read\_csv('/content/drive/MyDrive/datasets/Linear Regression/titanic.csv

df.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 891 entries, 0 to 890

Data columns (total 12 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 PassengerId 891 non-null int64

1 Survived 891 non-null int64

2 Pclass 891 non-null int64

3 Name 891 non-null object

4 Sex 891 non-null object

5 Age 714 non-null float64

6 SibSp 891 non-null int64

7 Parch 891 non-null int64

8 Ticket 891 non-null object

9 Fare 891 non-null float64

10 Cabin 204 non-null object

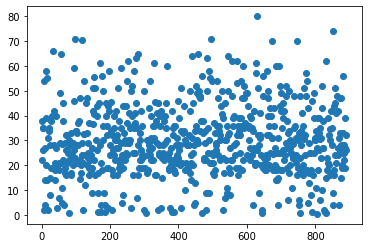
11 Embarked 889 non-null object

dtypes: float64(2), int64(5), object(5)

memory usage: 83.7+ KB

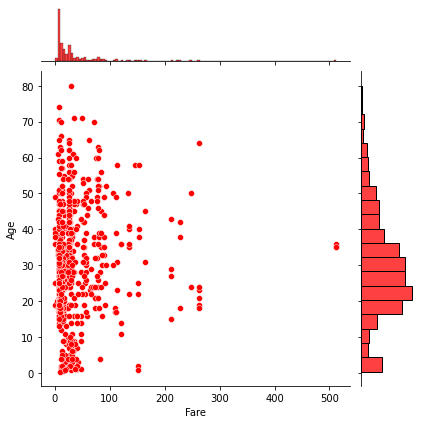
plt.scatter(df['PassengerId'],df['Age'])

<matplotlib.collections.PathCollection at 0x7f5d83108a90>

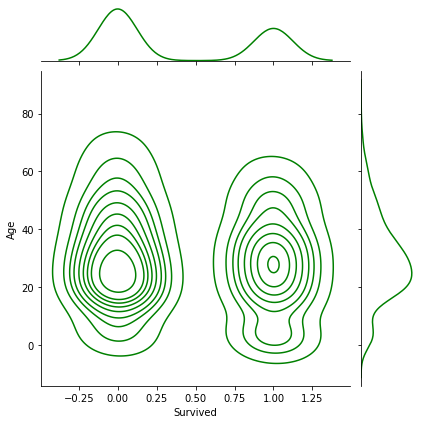


sns.jointplot(x='Fare',y='Age',data=df,color='r')

<seaborn.axisgrid.JointGrid at 0x7f5d8308d3d0>

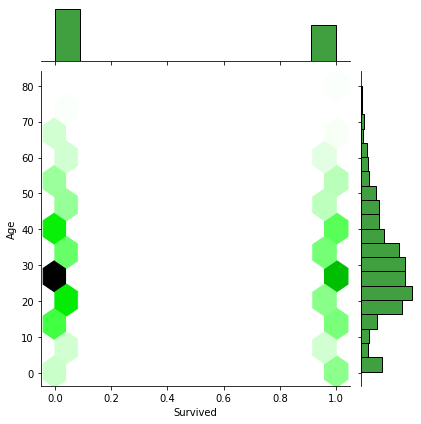


sns.jointplot(x='Survived',y='Age',data=df,color='g',kind="kde")

<seaborn.axisgrid.JointGrid at 0x7f5d80149450>

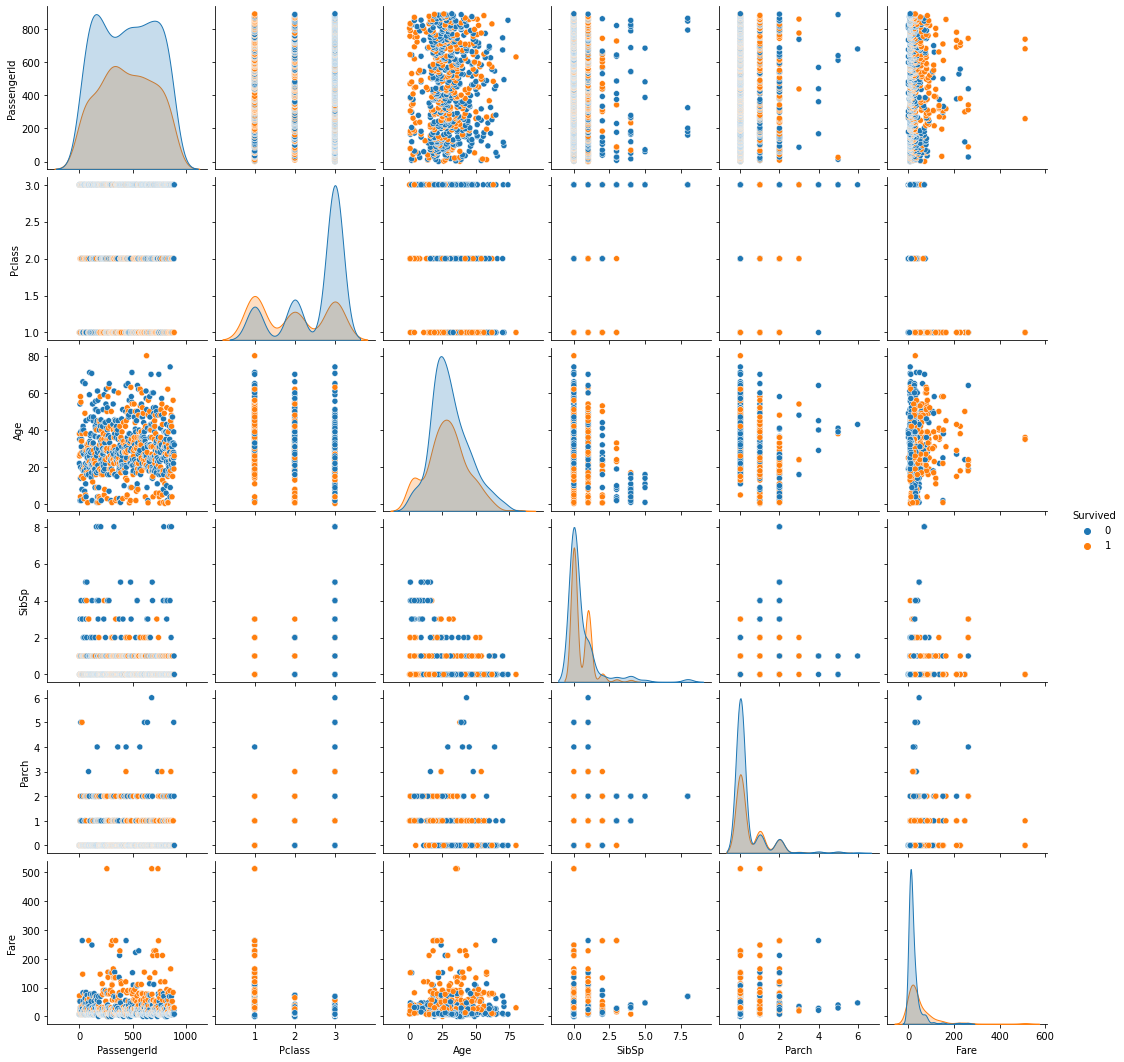
sns.jointplot(x='Survived',y='Age',data=df,color='g',kind="hex")

<seaborn.axisgrid.JointGrid at 0x7f5d8014b410>



sns.pairplot(df,hue='Survived')

<seaborn.axisgrid.PairGrid at 0x7f5d7d6a1750>



**WEEK-6**

**Write a program to perform any of the following Dimensionality Reduction techniques on real time datasets.**

**a) Principal Component Analysis**

**b) Single value Decomposition**

**c) Linear Discriminant Analysis**

**d) Factor Analysis**

1. **Principal Component analysis:**

Principal Component Analysis is an unsupervised learning algorithm that is used for the dimensionality reduction in [machine learning](https://www.javatpoint.com/machine-learning). It is a statistical process that converts the observations of correlated features into a set of linearly uncorrelated features with the help of orthogonal transformation. These new transformed features are called the **Principal Components**.

**Working of PCA:**

PCA works on a process called Eigenvalue Decomposition of a covariance matrix of a data set. The steps are as follows:

* First, calculate the covariance matrix of a data set.
* Then, calculate the eigenvectors of the covariance matrix.
* The eigenvector having the highest eigenvalue represents the direction in which there is the highest variance. So this will help in identifying the first principal component.
* The eigenvector having the next highest eigenvalue represents the direction in which data has the highest remaining variance and also orthogonal to the first direction. So, this helps in identifying the second principal component.
* Like this, identify the top ‘k’ eigenvectors having top ‘k’ eigenvalues to get the ‘k’ principal components.

**Uses of PCA:**

* It is used to find inter-relation between variables in the data.
* It is used to interpret and visualize data.
* The number of variables is decreasing it makes further analysis simpler.
* It is often used to visualize genetic distance and relatedness between populations.

**PROGRAM:**

**#import libraries**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

**#import dataset**

df=pd.read\_csv(r"C:\Users\Havilah Pragnam\Downloads\IRIS.csv")

df.head()

|  | **sepal\_length** | **sepal\_width** | **petal\_length** | **petal\_width** | **species** |
| --- | --- | --- | --- | --- | --- |
| **0** | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| **1** | 4.9 | 3.0 | 1.4 | 0.2 | Iris-setosa |
| **2** | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| **3** | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| **4** | 5.0 | 3.6 | 1.4 | 0.2 | Iris-setosa |

df1=df.species

df1

0 Iris-setosa

1 Iris-setosa

2 Iris-setosa

3 Iris-setosa

4 Iris-setosa

...

145 Iris-virginica

146 Iris-virginica

147 Iris-virginica

148 Iris-virginica

149 Iris-virginica

Name: species, Length: 150, dtype: object

df.drop('species',axis=1,inplace=True)

**#Scaling**

from sklearn.preprocessing import StandardScaler

sc=StandardScaler()

df=sc.fit\_transform(df)

**#Generating Principal components**

from sklearn.decomposition import PCA

pca=PCA(n\_components=2)

pcomponents=pca.fit\_transform(df)

pcomponents

pdf=pd.DataFrame(data=pcomponents,columns = ['principal component 1','principal component 2'])

pdf

| **principal component 1** | **principal component 2** |
| --- | --- |
| **0** | -2.264542 | 0.505704 |
| **1** | -2.086426 | -0.655405 |
| **2** | -2.367950 | -0.318477 |
| **3** | -2.304197 | -0.575368 |
| **4** | -2.388777 | 0.674767 |
| **...** | ... | ... |
| **145** | 1.870522 | 0.382822 |
| **146** | 1.558492 | -0.905314 |
| **147** | 1.520845 | 0.266795 |
| **148** | 1.376391 | 1.016362 |
| **149** | 0.959299 | -0.022284 |

150 rows × 2 columns

dim\_red=pd.concat([pdf,df1],axis=1)

dim\_red

| **principal component 1** | **principal component 2** | **species** |
| --- | --- | --- |
| **0** | -2.264542 | 0.505704 | Iris-setosa |
| **1** | -2.086426 | -0.655405 | Iris-setosa |
| **2** | -2.367950 | -0.318477 | Iris-setosa |
| **3** | -2.304197 | -0.575368 | Iris-setosa |
| **4** | -2.388777 | 0.674767 | Iris-setosa |
| **...** | ... | ... | ... |
| **145** | 1.870522 | 0.382822 | Iris-virginica |
| **146** | 1.558492 | -0.905314 | Iris-virginica |
| **147** | 1.520845 | 0.266795 | Iris-virginica |
| **148** | 1.376391 | 1.016362 | Iris-virginica |
| **149** | 0.959299 | -0.022284 | Iris-virginica |

1. ws × 3 columns

**b) Single value Decomposition**

The Singular Value Decomposition (SVD) of a matrix is a factorization of that matrix into three matrices. It has some interesting algebraic properties and conveys important geometrical and theoretical insights about linear transformations. It also has some important applications in data science.

**Working of SVD:**

* We need to find eigenvalues of matrix A and as A can be a rectangular matrix, we need to convert it to a square matrix by multiplying A with its transpose.
* Now, that we have a square matrix, we can calculate the eigenvalues of AT A and AAT.
* Once we have calculated the eigenvalues, it is time to calculate the two eigenvectors for each eigenvalue.
* The eigenvectors of *ATA* make up the columns of *V*, the eigenvectors of *AAT*make up the columns of *U*. Also, the singular values in **S** are square roots of eigenvalues from *AAT* or *ATA*. The singular values are the diagonal entries of the *S*matrix and are arranged in descending order.

**A*nxp*= U*nxn* S*nxp* VT*pxp***

**PROGRAM:**

**#import libraries**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

**#import dataset**

df=pd.read\_csv(r"C:\Users\Havilah Pragnam\Downloads\IRIS.csv")

df.head()

|  | **sepal\_length** | **sepal\_width** | **petal\_length** | **petal\_width** | **species** |
| --- | --- | --- | --- | --- | --- |
| **0** | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| **1** | 4.9 | 3.0 | 1.4 | 0.2 | Iris-setosa |
| **2** | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| **3** | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| **4** | 5.0 | 3.6 | 1.4 | 0.2 | Iris-setosa |

df1=df.species

df1

0 Iris-setosa

1 Iris-setosa

2 Iris-setosa

3 Iris-setosa

4 Iris-setosa

...

145 Iris-virginica

146 Iris-virginica

147 Iris-virginica

148 Iris-virginica

149 Iris-virginica

Name: species, Length: 150, dtype: object

df.drop('species',axis=1,inplace=True)

**#Scaling**

from sklearn.preprocessing import StandardScaler

sc=StandardScaler()

df=sc.fit\_transform(df)

#**Single value decomposition**

from sklearn.decomposition import TruncatedSVD

svd=TruncatedSVD(n\_components=2)

scomponents=svd.fit\_transform(df)

svd=pd.DataFrame(data=scomponents,columns = ['sd-1','sd-2'])

svd

| **sd-1** | **sd-2** |
| --- | --- |
| **0** | -2.264542 | 0.505704 |
| **1** | -2.086426 | -0.655405 |
| **2** | -2.367950 | -0.318477 |
| **3** | -2.304197 | -0.575368 |
| **4** | -2.388777 | 0.674767 |
| **...** | ... | ... |
| **145** | 1.870522 | 0.382822 |
| **146** | 1.558492 | -0.905314 |
| **147** | 1.520845 | 0.266795 |
| **148** | 1.376391 | 1.016362 |
| **149** | 0.959299 | -0.022284 |

150 rows × 2 columns

dim\_red2=pd.concat([svd,df1],axis=1)

dim\_red2

| **sd-1** | **sd-2** | **species** |
| --- | --- | --- |
| **0** | -2.264542 | 0.505704 | Iris-setosa |
| **1** | -2.086426 | -0.655405 | Iris-setosa |
| **2** | -2.367950 | -0.318477 | Iris-setosa |
| **3** | -2.304197 | -0.575368 | Iris-setosa |
| **4** | -2.388777 | 0.674767 | Iris-setosa |
| **...** | ... | ... | ... |
| **145** | 1.870522 | 0.382822 | Iris-virginica |
| **146** | 1.558492 | -0.905314 | Iris-virginica |
| **147** | 1.520845 | 0.266795 | Iris-virginica |
| **148** | 1.376391 | 1.016362 | Iris-virginica |
| **149** | 0.959299 | -0.022284 | Iris-virginica |

150 rows × 3 columns

**c)Linear Discriminant Analysis**

Linear Discriminant Analysis or Normal Discriminant Analysis or Discriminant Function Analysis is a dimensionality reduction technique that is commonly used for supervised classification problems. It is used for modelling differences in groups i.e., separating two or more classes. It is used to project the features in higher dimension space into a lower dimension space.

**Working of LDA:**

* Compute the dd-dimensional mean vectors for the different classes from the dataset.
* Compute the scatter matrices (in-between-class and within-class scatter matrix).
* Compute the eigenvectors (e1,e2,...,ed) and corresponding eigenvalues (λ1,λ2,...,λd) for the scatter matrices.
* Sort the eigenvectors by decreasing eigenvalues and choose kk eigenvectors with the largest eigenvalues to form a d×k dimensional matrix W (where every column represents an eigenvector).
* Use this d×k eigenvector matrix to transform the samples onto the new subspace. This can be summarized by the matrix multiplication: Y=X×W (where X is a n×d dimensional matrix representing the n samples, and y are the transformed n×k-dimensional samples in the new subspace).

**Program:**

**#import libraries**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

**#import dataset**

df=pd.read\_csv(r"C:\Users\Havilah Pragnam\Downloads\IRIS.csv")

df.head()

|  | **sepal\_length** | **sepal\_width** | **petal\_length** | **petal\_width** | **species** |
| --- | --- | --- | --- | --- | --- |
| **0** | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| **1** | 4.9 | 3.0 | 1.4 | 0.2 | Iris-setosa |
| **2** | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| **3** | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| **4** | 5.0 | 3.6 | 1.4 | 0.2 | Iris-setosa |

df1=df.species

df1

0 Iris-setosa

1 Iris-setosa

2 Iris-setosa

3 Iris-setosa

4 Iris-setosa

...

145 Iris-virginica

146 Iris-virginica

147 Iris-virginica

148 Iris-virginica

149 Iris-virginica

Name: species, Length: 150, dtype: object

df.drop('species',axis=1,inplace=True)

**#Scaling**

from sklearn.preprocessing import StandardScaler

sc=StandardScaler()

df=sc.fit\_transform(df)

**#linear discriminant analysis**

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis as LDA

lda = LDA(n\_components=2)

lcomponents= lda.fit\_transform(df,df1)

lda=pd.DataFrame(data=lcomponents,columns = ['ld-1','ld-2'])

lda

ld-1 ld-2

0 8.084953 0.328454

1 7.147163 -0.755473

2 7.511378 -0.238078

3 6.837676 -0.642885

4 8.157814 0.540639

... ... ...

145 -5.674013 1.661346

146 -5.197129 -0.365506

147 -4.981712 0.812973

148 -5.901486 2.320751

149 -4.684009 0.325081

dim\_red1=pd.concat([lda,df1],axis=1)

dim\_red1

| **ld-1** | **ld-2** | **species** |
| --- | --- | --- |
| **0** | 8.084953 | 0.328454 | Iris-setosa |
| **1** | 7.147163 | -0.755473 | Iris-setosa |
| **2** | 7.511378 | -0.238078 | Iris-setosa |
| **3** | 6.837676 | -0.642885 | Iris-setosa |
| **4** | 8.157814 | 0.540639 | Iris-setosa |
| **...** | ... | ... | ... |
| **145** | -5.674013 | 1.661346 | Iris-virginica |
| **146** | -5.197129 | -0.365506 | Iris-virginica |
| **147** | -4.981712 | 0.812973 | Iris-virginica |
| **148** | -5.901486 | 2.320751 | Iris-virginica |
| **149** | -4.684009 | 0.325081 | Iris-virginica |

150 rows × 3 columns

**PART-B**

**WEEK-1**

**AIM:**

Write a program to generate Association Rules using the Apriori algorithm

**Description:**

Apriori algorithm refers to the algorithm which is used to calculate the association rules between objects. It means how two or more objects are related to one another. In other words, we can say that the apriori algorithm is an association rule leaning that analyzes that people who bought product A also bought product B.

The primary objective of the apriori algorithm is to create the association rule between different objects. The association rule describes how two or more objects are related to one another. Apriori algorithm is also called frequent pattern mining. Generally, you operate the Apriori algorithm on a database that consists of a huge number of transactions. Let's understand the apriori algorithm with the help of an example; suppose you go to Big Bazar and buy different products. It helps the customers buy their products with ease and increases the sales performance of the Big Bazar. In this tutorial, we will discuss the apriori algorithm with examples. Apriori algorithm refers to an algorithm that is used in mining frequent products sets and relevant association rules. Generally, the apriori algorithm operates on a database containing a huge number of transactions. For example, the items customers but at a Big Bazar.

Apriori algorithm helps the customers to buy their products with ease and increases the sales performance of the particular store.

**Code:**

import pandas as pd

from mlxtend.preprocessing import TransactionEncoder

from mlxtend.frequent\_patterns import apriori, association\_rules

dataset = [['Pen','Bag','Pencil'],

           ['Pen','Book','Water Bottle','Pencil'],

           ['Book','Bag','Water Bottle'],

           ['Pen','Book','Bag','Water Bottle']]

te = TransactionEncoder()

te\_ary = te.fit(dataset).transform(dataset)

df = pd.DataFrame(te\_ary, columns=te.columns\_)

print(df)

# Building the model

frq\_items = apriori(df, min\_support = 0.05, use\_colnames = True)

print(frq\_items)

# Collecting the inferred rules in a dataframe

rules = association\_rules(frq\_items, metric ="lift", min\_threshold = 1)

rules = rules.sort\_values(['confidence', 'lift'], ascending =[False, False])

print(rules.head())

Bag Book Pen Pencil Water Bottle

0 True False True True False

1 False True True True True

2 True True False False True

3 True True True False True

support itemsets

0 0.75 (Bag)

1 0.75 (Book)

2 0.75 (Pen)

3 0.50 (Pencil)

4 0.75 (Water Bottle)

5 0.50 (Book, Bag)

6 0.50 (Bag, Pen)

7 0.25 (Pencil, Bag)

8 0.50 (Water Bottle, Bag)

9 0.50 (Book, Pen)

10 0.25 (Pencil, Book)

11 0.75 (Water Bottle, Book)

12 0.50 (Pencil, Pen)

13 0.50 (Water Bottle, Pen)

14 0.25 (Water Bottle, Pencil)

15 0.25 (Book, Bag, Pen)

16 0.50 (Water Bottle, Book, Bag)

17 0.25 (Pencil, Bag, Pen)

18 0.25 (Water Bottle, Bag, Pen)

19 0.25 (Pencil, Book, Pen)

20 0.50 (Water Bottle, Book, Pen)

21 0.25 (Pencil, Book, Water Bottle)

22 0.25 (Water Bottle, Pencil, Pen)

23 0.25 (Water Bottle, Book, Bag, Pen)

24 0.25 (Pencil, Water Bottle, Book, Pen)

antecedents consequents antecedent support \

40 (Water Bottle, Pencil) (Book, Pen) 0.25

41 (Book, Pencil) (Water Bottle, Pen) 0.25

0 (Water Bottle) (Book) 0.75

1 (Book) (Water Bottle) 0.75

2 (Pencil) (Pen) 0.50

consequent support support confidence lift leverage conviction

40 0.50 0.25 1.0 2.000000 0.1250 inf

41 0.50 0.25 1.0 2.000000 0.1250 inf

0 0.75 0.75 1.0 1.333333 0.1875 inf

1 0.75 0.75 1.0 1.333333 0.1875 inf

2 0.75 0.50 1.0 1.333333 0.1250 inf

**WEEK-2**

**AIM:**

Write a program to generate Association Rules using the FP-Growth algorithm.

**Description:**

In Data Mining, finding frequent patterns in large databases is very important and has been studied on a large scale in the past few years. Unfortunately, this task is computationally expensive, especially when many patterns exist.

The FP-Growth Algorithm proposed by **Han in**. This is an efficient and scalable method for mining the complete set of frequent patterns by pattern fragment growth, using an extended prefix-tree structure for storing compressed and crucial information about frequent patterns named frequent-pattern tree (FP-tree). In his study, Han proved that his method outperforms other popular methods for mining frequent patterns, e.g. the Apriori Algorithm and the TreeProjection. In some later works, it was proved that FP-Growth performs better than other methods, including **Eclat** and **Relim**. The popularity and efficiency of the FP-Growth Algorithm contribute to many studies that propose variations to improve its performance.

The FP-Growth Algorithm is an alternative way to find frequent item sets without using candidate generations, thus improving performance. For so much, it uses a divide-and-conquer strategy. The core of this method is the usage of a special data structure named frequent-pattern tree (FP-tree), which retains the item set association information.

**Code:**

import sys

!{sys.executable} -m pip install fpgrowth

Looking in indexes: <https://pypi.org/simple>, <https://us-python.pkg.dev/colab-wheels/public/simple/>

Collecting fpgrowth

Downloading fpGrowth-1.0.0.tar.gz (2.1 kB)

Building wheels for collected packages: fpgrowth

Building wheel for fpgrowth (setup.py) ... done

Created wheel for fpgrowth: filename=fpGrowth-1.0.0-py3-none-any.whl size=2866 sha256=458dd64cdcdca2a32dd39b0d375e68d6d04546eba93e66f5255e27362d23650f

Stored in directory: /root/.cache/pip/wheels/64/33/72/1991a9117d1813325c4ef85597ba8ece8c4780adc240bd0b0f

Successfully built fpgrowth

Installing collected packages: fpgrowth

Successfully installed fpgrowth-1.0.0

%pip install mlxtend –upgrade

Looking in indexes: <https://pypi.org/simple>, <https://us-python.pkg.dev/colab-wheels/public/simple/>

Requirement already satisfied: mlxtend in /usr/local/lib/python3.7/dist-packages (0.14.0)

Collecting mlxtend

Downloading mlxtend-0.21.0-py2.py3-none-any.whl (1.3 MB)

|████████████████████████████████| 1.3 MB 22.6 MB/s

Requirement already satisfied: numpy>=1.16.2 in /usr/local/lib/python3.7/dist-packages (from mlxtend) (1.21.6)

Requirement already satisfied: scikit-learn>=1.0.2 in /usr/local/lib/python3.7/dist-packages (from mlxtend) (1.0.2)

Requirement already satisfied: joblib>=0.13.2 in /usr/local/lib/python3.7/dist-packages (from mlxtend) (1.2.0)

Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (from mlxtend) (57.4.0)

Requirement already satisfied: matplotlib>=3.0.0 in /usr/local/lib/python3.7/dist-packages (from mlxtend) (3.2.2)

Requirement already satisfied: scipy>=1.2.1 in /usr/local/lib/python3.7/dist-packages (from mlxtend) (1.7.3)

Requirement already satisfied: pandas>=0.24.2 in /usr/local/lib/python3.7/dist-packages (from mlxtend) (1.3.5)

Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=3.0.0->mlxtend) (3.0.9)

Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=3.0.0->mlxtend) (1.4.4)

Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=3.0.0->mlxtend) (2.8.2)

Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=3.0.0->mlxtend) (0.11.0)

Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packages (from kiwisolver>=1.0.1->matplotlib>=3.0.0->mlxtend) (4.1.1)

Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.24.2->mlxtend) (2022.4)

Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.1->matplotlib>=3.0.0->mlxtend) (1.15.0)

Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=1.0.2->mlxtend) (3.1.0)

Installing collected packages: mlxtend

Attempting uninstall: mlxtend

Found existing installation: mlxtend 0.14.0

Uninstalling mlxtend-0.14.0:

Successfully uninstalled mlxtend-0.14.0

Successfully installed mlxtend-0.21.0

EXAMPLE2:

import pandas as pd

from mlxtend.preprocessing import TransactionEncoder

from mlxtend.frequent\_patterns import fpgrowth

dataset = [['k', 'O', 'N', 'M', 'E', 'Y'],

           ['N', 'O', 'D', 'K', 'E', 'Y'],

           ['M', 'A', 'K', 'E'],

           ['N', 'U', 'A', 'K', 'Y'],

           ['C', 'O', 'O', 'k']]

te = TransactionEncoder()

te\_ary = te.fit(dataset).transform(dataset)

df = pd.DataFrame(te\_ary, columns=te.columns\_)

print(df)

fpgrowth(df, min\_support=0.6)

fpgrowth(df, min\_support=0.6, use\_colnames=True)

A C D E K M N O U Y k

0 False False False True False True True True False True True

1 False False True True True False True True False True False

2 True False False True True True False False False False False

3 True False False False True False True False True True False

4 False True False False False False False True False False True

|  | **support** | **itemsets** |
| --- | --- | --- |
| **0** | 0.6 | (Y) |
| **1** | 0.6 | (O) |
| **2** | 0.6 | (N) |
| **3** | 0.6 | (E) |
| **4** | 0.6 | (K) |
| **5** | 0.6 | (N, Y) |

**WEEK-3**

**AIM:**

*Write a program to implement K-means clustering algorithm.*

**Description:**

K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science. In this topic, we will learn what is K-means clustering algorithm, how the algorithm works, along with the Python implementation of k-means clustering. K-Means Clustering is an [Unsupervised Learning algorithm](https://www.javatpoint.com/unsupervised-machine-learning), which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.

It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties. It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.

It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

The k-means [clustering](https://www.javatpoint.com/clustering-in-machine-learning) algorithm mainly performs two tasks:

* Determines the best value for K center points or centroids by an iterative process.
* Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

Hence each cluster has datapoints with some commonalities, and it is away from other clusters.

**Code:**import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

x = [1, 1.5, 3, 5, 3.5, 4.5, 3.5]

y = [1,2,4,7,5,5,4.5]

plt.scatter(x, y)

plt.show()

data = list(zip(x, y))

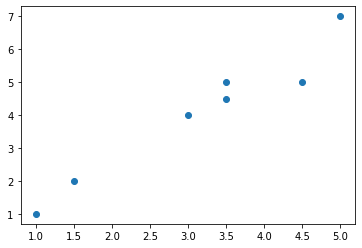
kmeans = KMeans(n\_clusters=2)

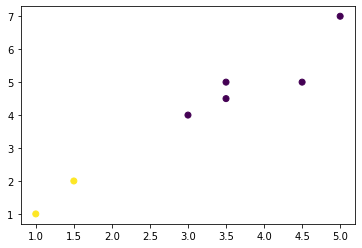
kmeans.fit(data)

plt.scatter(x, y, c=kmeans.labels\_)

plt.show()

plt.legend()







WARNING:matplotlib.legend:No handles with labels found to put in legend.

<matplotlib.legend.Legend at 0x7f8494666810>

**WEEK-4**

**Write a program to implement hierarchical clustering algorithms**.

**Description:**

Hierarchical clustering is another unsupervised machine learning algorithm, which is used to group the unlabelled datasets into a cluster and also known as hierarchical cluster analysis or HCA. In this algorithm, we develop the hierarchy of clusters in the form of a tree, and this tree-shaped structure is known as the dendrogram. Sometimes the results of K-means clustering and hierarchical clustering may look similar, but they both differ depending on how they work. As there is no requirement to predetermine the number of clusters as we did in the K-Means algorithm. The hierarchical clustering technique has two approaches:

1. Agglomerative: Agglomerative is a bottom-up approach, in which the algorithm starts with taking all data points as single clusters and merging them until one cluster is left.

2. Divisive: Divisive algorithm is the reverse of the agglomerative algorithm as it is a top-down approach.

**PROGRAM:**

**MIN:**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import AgglomerativeClustering

from scipy.cluster.hierarchy import dendrogram, linkage

x=[4,5,10,4,3,10,14,6,5,12]

y=[2,9,25,10,15,12,20,28,12,20]

data=list(zip(x,y))

hierarchical\_cluster = AgglomerativeClustering(n\_clusters=2, affinity='euclidean', linkage='single')

labels = hierarchical\_cluster.fit\_predict(data)

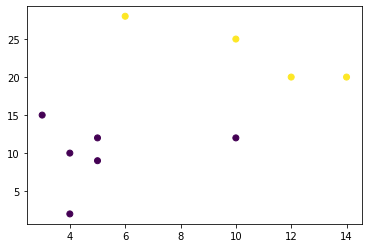
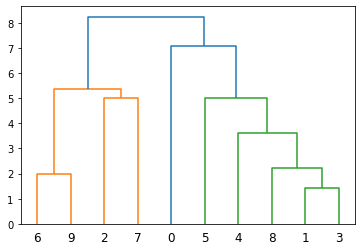
plt.scatter(x,y,c=labels)

plt.show()

dendrogram (linkage(data, method='single', metric='euclidean'))

plt.show()

**OUTPUT:**

** **

**MAX:**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import AgglomerativeClustering

from scipy.cluster.hierarchy import dendrogram, linkage

x=[4,5,10,4,3,10,14,6,5,12]

y=[2,9,25,10,15,12,20,28,12,20]

data=list(zip(x,y))

hierarchical\_cluster = AgglomerativeClustering(n\_clusters=2, affinity='euclidean', linkage='complete')

labels = hierarchical\_cluster.fit\_predict(data)

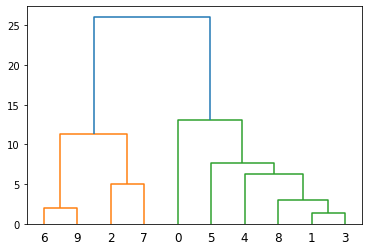
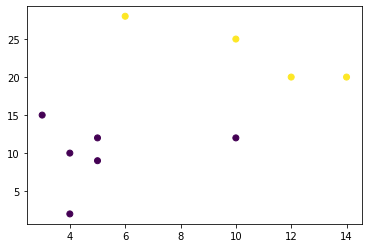
plt.scatter(x,y,c=labels)

plt.show()

dendrogram(linkage(data, method='complete', metric='euclidean'))

plt.show()

**OUTPUT:**

****

**WEEK-5**

**AIM:**

**Write a program to implement the DBSCAN clustering algorithm.**

**Description:**

Density-based spatial clustering of applications with noise (DBSCAN) clustering method. Clusters are dense regions in the data space, separated by regions of the lower density of points. The DBSCAN algorithm is based on this intuitive notion of “clusters” and “noise”. The key idea is that for each point of a cluster, the neighbourhood of a given radius must contain at least a minimum number of points.

**In this algorithm, we have 3 types of data points.**

**Core Point**: A point is a core point if it has more than MinPts points within eps.

**Border Point**: A point which has fewer than MinPts within eps but it is in the neighbourhood of a core point.

**Noise or outlier**: A point which is not a core point or border point.

**Code**

from sklearn.cluster import DBSCAN

import numpy as np

x=[1,2,2,7,8,25]

y=[2,5,3,8,10,80]

X=list(zip(x,y))

clustering = DBSCAN(eps=3, min\_samples=2).fit(X)

print(clustering.labels\_)

**OUTPUT:**

[ 0 0 0 1 1 -1]

**WEEK-6**

**Aim:**

**Write a program to implement the Web Scraping**

**Description:**

Web scraping is a technique to scrape or extract data from the web, particularly from web

pages. Web scraping may involve manually copying the data or using automation to crawl,

parse, and extract information from web pages.

• Crawl: A bot or a web crawler is designed to query a web server using the required

set of URLs to fetch the web pages.

• Scrape: Once the raw web page has been fetched, the next task is to extract

information from it. The task of scraping involves utilizing techniques like regular

expressions, extraction based on XPath, or specific tags and so on to narrow down to

the required information on the page.

**Code:**

import requests

from bs4 import BeautifulSoup

url = 'https://www.worldometers.info/coronavirus/countries-where-coronavirus-has-spread/'

page = requests.get(url)

soup = BeautifulSoup(page.text, 'html.parser')

data = []

data\_iterator = iter(soup.find\_all('td'))

while True:

    try:

        country = next(data\_iterator).text

        confirmed = next(data\_iterator).text

        deaths = next(data\_iterator).text

        continent = next(data\_iterator).text

        data.append((

            country,

            int(confirmed.replace(",", "")),

            int(deaths.replace(",","")),

            continent

        ))

    except StopIteration:

        break

data.sort(key = lambda row: row[1], reverse = True)

data

**Output:**

[('United States', 97157866, 1076013, 'North America'), ('India', 44502694, 528165, 'Asia'), ('France', 34722711, 154529, 'Europe'), ('Brazil', 34580412, 684951, 'South America'), ('Germany', 32452250, 148299, 'Europe'), ('South Korea', 24041825, 27498, 'Asia'),

('United Kingdom', 23554519, 189026, 'Europe'), ('Italy', 22054443, 176242, 'Europe'), ('Japan (+Diamond Princess)', 20157704, 42650, 'Asia'), ('Russia', 20113098, 385429, 'Europe'), ('Turkey', 16829941, 100979, 'Asia'), ('Spain', 13367647, 113130, 'Europe'), ('Vietnam', 11441626, 43130, 'Asia'), ('Australia', 10119203, 14432, 'Australia/Oceania'), ('Argentina', 9697763, 129830, 'South America'), ('Netherlands', 8396979, 22613, 'Europe'), ('Iran', 7539698, 144199, 'Asia'), ('Mexico', 7059348, 329761, 'North America'), ('Indonesia', 6394340, 157787, 'Asia'), ('Colombia', 6304317, 141708, 'South America'), ('Poland', 6213262, 117252, 'Europe'), ('Taiwan', 5707688, 10312, 'Asia'), ('Portugal', 5444993, 24924, 'Europe'), ('Ukraine', 5072533, 108885, 'Europe'), ('Austria', 4955082, 19486, 'Europe'), ('Malaysia', 4806954, 36285, 'Asia'), ('Greece', 4804982, 32757, 'Europe'), ('North Korea', 4772813, 74, 'Asia'), ('Thailand', 4668244, 32557, 'Asia'), ('Israel', 4643996, 11657, 'Asia'), ('Chile', 4566548, 60812, 'South America'), ('Belgium', 4497199, 32575, 'Europe'), ('Canada', 4197701, 44347, 'North America'), ('Peru', 4126021, 216125, 'South America'), ('Czech Republic (Czechia)', 4059441, 40910, 'Europe'), ('Switzerland', 4040280, 14157, 'Europe'), ('South Africa', 4014485, 102129, 'Africa'), ('Philippines', 3908295, 62342, 'Asia'), ('Romania', 3241772, 66856, 'Europe'), ('Denmark', 3098447, 6982, 'Europe'), ('Sweden', 2573548, 19974, 'Europe'), ('Iraq', 2458509, 25348, 'Asia'), ('Serbia', 2318677, 16829, 'Europe'), ('Hungary', 2058847, 47367, 'Europe'), ('Bangladesh', 2015308, 29334, 'Asia'), ('Singapore', 1861390, 1602, 'Asia'), ('Slovakia', 1837272, 20411, 'Europe'), ('New Zealand', 1760113, 2836, 'Australia/Oceania'), ('Jordan', 1742256, 14114, 'Asia'), ('Georgia', 1735682, 16889, 'Asia'), ('Hong Kong', 1659912, 9810, 'Asia'), ('Ireland', 1659034, 7829, 'Europe'), ('Pakistan', 1571098, 30599, 'Asia'), ('Norway', 1461119, 4004, 'Europe'), ('Kazakhstan', 1391645, 13688, 'Asia'), ('Finland', 1271516, 5768, 'Europe'), ('Morocco', 1264664, 16276, 'Africa'), ('Bulgaria', 1248200, 37646, 'Europe'), ('Lithuania', 1226685, 9296, 'Europe'), ('Croatia', 1220490, 16806, 'Europe'), ('Lebanon', 1212815, 10647, 'Asia'), ('Tunisia', 1144824, 29238, 'Africa'), ('Slovenia', 1143235, 6794, 'Europe'), ('Guatemala', 1111191, 19679, 'North America'), ('Cuba', 1110918, 8530, 'North America'), ('Bolivia', 1106142, 22217, 'South America'), ('Costa Rica', 1066630, 8893, 'North America'), ('United Arab Emirates', 1020412, 2342, 'Asia'), ('Nepal', 998870, 12015, 'Asia'), ('Ecuador', 998202, 35876, 'South America'), ('Belarus', 994037, 7118, 'Europe'), ('Uruguay', 982846, 7462, 'South America'), ('Panama', 981822, 8480, 'North America'), ('Mongolia', 981200, 2179, 'Asia'), ('Latvia', 907831, 5957, 'Europe'), ('Azerbaijan', 817938, 9857, 'Asia'), ('Saudi Arabia', 814597, 9317, 'Asia'), ('Paraguay', 715806, 19530, 'South America'), ('Bahrain', 674303, 1518, 'Asia'), ('Sri Lanka', 670471, 16731, 'Asia'), ('Kuwait', 657745, 2563, 'Asia'), ('Dominican Republic', 641677, 4384, 'North America'), ('State of Palestine', 620371, 5402, 'Asia'), ('Myanmar', 617056, 19442, 'Asia'), ('Estonia', 598580, 2657, 'Europe'), ('Cyprus', 579899, 1173, 'Asia'), ('Moldova', 579110, 11783, 'Europe'), ('Venezuela', 543811, 5809, 'South America'), ('Egypt', 515645, 24613, 'Africa'), ('Libya', 506898, 6437, 'Africa'), ('Ethiopia', 493340, 7572, 'Africa'), ('Réunion', 467816, 879, 'Africa'), ('Honduras', 455011, 10989, 'North America'), ('Armenia', 439302, 8669, 'Asia'), ('Qatar', 436820, 682, 'Asia'), ('Oman', 397846, 4260, 'Asia'), ('Bosnia and Herzegovina', 397296, 16100, 'Europe'), ('North Macedonia', 341583, 9512, 'Europe'), ('Kenya', 338301, 5674, 'Africa'), ('Zambia', 333234, 4017, 'Africa'), ('Albania', 331053, 3585, 'Europe'), ('Botswana', 325931, 2786, 'Africa'), ('Luxembourg', 288658, 1123, 'Europe'), ('Montenegro', 277440, 2778, 'Europe'), ('Algeria', 270551, 6879, 'Africa'), ('Nigeria', 264450, 3154, 'Africa'), ('Zimbabwe', 256888, 5596, 'Africa'), ('China', 247078, 5226, 'Asia'), ('Uzbekistan', 243893, 1637, 'Asia'), ('Mozambique', 230174, 2221, 'Africa'), ('Brunei ', 223059, 225, 'Asia'), ('Martinique', 219529, 1036, 'North America'), ('Laos', 214982, 757, 'Asia'), ('Kyrgyzstan', 205920, 2991, 'Asia'), ('Iceland', 205009, 213, 'Europe'), ('El Salvador', 201785, 4228, 'North America'), ('Afghanistan', 196182, 7789, 'Asia'), ('Guadeloupe', 191997, 986, 'North America'), ('Maldives', 184966, 308, 'Asia'), ('Trinidad and Tobago', 181421, 4174, 'North America'), ('Uganda', 169396, 3628, 'Africa'), ('Namibia', 169253, 4065, 'Africa'), ('Ghana', 168616, 1459, 'Africa'), ('Jamaica', 150844, 3284, 'North America'), ('Cambodia', 137719, 3056, 'Asia'), ('Rwanda', 132474, 1466, 'Africa'), ('Cameroon', 121652, 1935, 'Africa'), ('Malta', 114283, 803, 'Europe'), ('Angola', 103131, 1917, 'Africa'), ('Barbados', 101899, 556, 'North America'), ('French Guiana', 93837, 409, 'South America'), ('DR Congo', 92751, 1422, 'Africa'), ('Channel Islands', 90153, 199, 'Europe'), ('Senegal', 88199, 1968, 'Africa'), ('Malawi', 87943, 2680, 'Africa'), ("Côte d'Ivoire", 86941, 822, 'Africa'), ('Suriname', 81057, 1384, 'South America'), ('French Polynesia', 76542, 649, 'Australia/Oceania'), ('New Caledonia', 73989, 314, 'Australia/Oceania'), ('Eswatini', 73374, 1422, 'Africa'), ('Guyana', 71192, 1279, 'South America'), ('Belize', 68473, 680, 'North America'), ('Fiji', 68207, 878, 'Australia/Oceania'), ('Madagascar', 66652, 1410, 'Africa'), ('Sudan', 63275, 4961, 'Africa'), ('Mauritania', 62777, 993, 'Africa'), ('Cabo Verde', 62344, 410, 'Africa'), ('Bhutan', 61419, 21, 'Asia'), ('Syria', 57172, 3163, 'Asia'), ('Burundi', 49370, 38, 'Africa'), ('Gabon', 48668, 306, 'Africa'), ('Seychelles', 46358, 169, 'Africa'), ('Andorra', 46113, 155, 'Europe'), ('Curaçao', 45127, 282, 'North America'), ('Papua New Guinea', 44915, 664, 'Australia/Oceania'), ('Aruba', 42914, 227, 'North America'), ('Mauritius', 40342, 1023, 'Africa'), ('Mayotte', 40161, 187, 'Africa'), ('Tanzania', 39168, 845, 'Africa'), ('Togo', 38649, 284, 'Africa'), ('Isle of Man', 38008, 116, 'Europe'), ('Guinea', 37652, 449, 'Africa'), ('Bahamas', 37146, 823, 'North America'), ('Faeroe Islands', 34658, 28, 'Europe'), ('Lesotho', 34287, 704, 'Africa'), ('Haiti', 33658, 851, 'North America'), ('Mali', 32263, 739, 'Africa'), ('Cayman Islands', 30380, 29, 'North America'), ('Saint Lucia', 28894, 391, 'North America'), ('Benin', 27490, 163, 'Africa'), ('Somalia', 27020, 1350, 'Africa'), ('Congo', 24837, 386, 'Africa'), ('Timor-Leste', 23217, 138, 'Asia'), ('Solomon Islands', 21544, 153, 'Australia/Oceania'), ('Burkina Faso', 21128, 387, 'Africa'), ('San Marino', 20500, 118, 'Europe'), ('Gibraltar', 20069, 108, 'Europe'), ('Grenada', 19403, 236, 'North America'), ('Liechtenstein', 19333, 86, 'Europe'), ('Nicaragua', 18491, 225, 'North America'), ('Bermuda', 18019, 148, 'North America'), ('South Sudan', 17823, 138, 'Africa'), ('Tajikistan', 17786, 125, 'Asia'), ('Equatorial Guinea', 16965, 183, 'Africa'), ('Tonga', 16182, 12, 'Australia/Oceania'), ('Samoa', 15889, 29, 'Australia/Oceania'), ('Djibouti', 15690, 189, 'Africa'), ('Marshall Islands', 15177, 17, 'Australia/Oceania'), ('Central African Republic', 14883, 113, 'Africa'), ('Dominica', 14852, 68, 'North America'), ('Monaco', 14436, 57, 'Europe'), ('Gambia', 12311, 371, 'Africa'), ('Greenland', 11971, 21, 'North America'), ('Saint Martin', 11941, 63, 'North America'), ('Yemen', 11932, 2155, 'Asia'), ('Vanuatu', 11908, 14, 'Australia/Oceania'), ('Caribbean Netherlands', 11176, 36, 'North America'), ('Sint Maarten', 10847, 87, 'North America'), ('Eritrea', 10163, 103, 'Africa'), ('Niger', 9931, 312, 'Africa'), ('Antigua and Barbuda', 8974, 145, 'North America'), ('Guinea-Bissau', 8796, 175, 'Africa'), ('Comoros', 8455, 161, 'Africa'), ('Liberia', 7929, 294, 'Africa'), ('Micronesia', 7856, 27, 'Australia/Oceania'), ('Sierra Leone', 7749, 126, 'Africa'), ('Chad', 7558, 193, 'Africa'), ('British Virgin Islands', 7305, 64, 'North America'), ('St. Vincent & Grenadines', 7112, 115, 'North America'), ('Saint Kitts & Nevis', 6532, 46, 'North America'), ('Cook Islands', 6386, 1, 'Australia/Oceania'), ('Turks and Caicos', 6372, 36, 'North America'), ('Sao Tome & Principe', 6193, 76, 'Africa'), ('Palau', 5430, 6, 'Australia/Oceania'), & Miquelon', 3131, 1, 'North America'), ('Falkland Islands', 1886, 0, 1276, 8, 'North ('Tuvalu', 20, 0, 'Australia/Oceania'), ('Western Sahara', 10, 1, 'Africa'), ('MS Zaandam', 9, 2, '')]