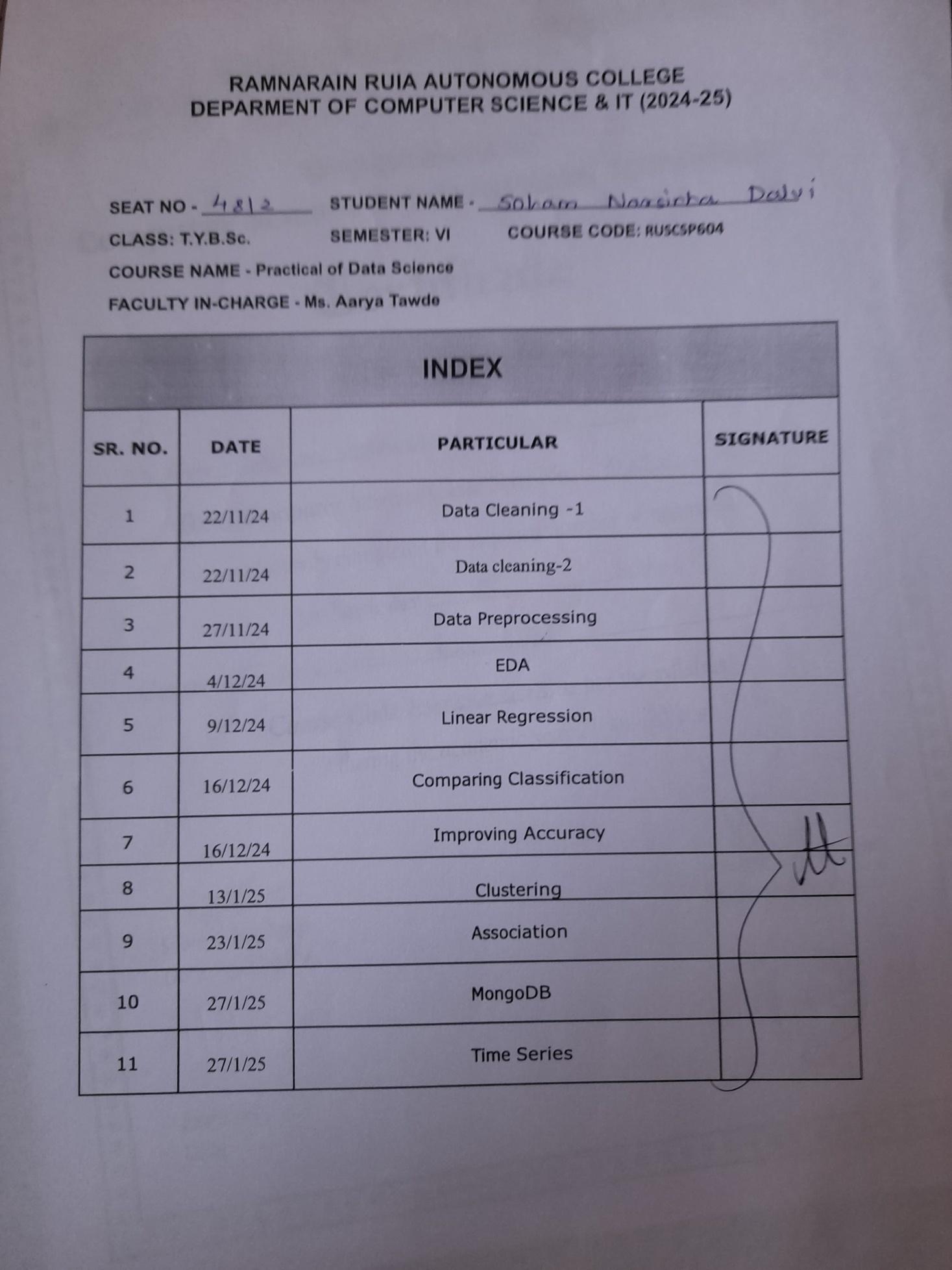
**Data Science Final-Ejournal**

**Name: Soham Narsinha Dalvi.**

**Roll No: 4812**

**Class: TYBSC.CS**





Practical No. 1

Data Cleaning - 1

**Q.1]** Do data clearing on the following dataset

1. Missing values

2. Removing duplicates

3. Finding and removing Noisy data-Bining,Boxplot

**Code:**

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

import seaborn as sns

df = pd.read\_csv('student\_data.csv')

df.replace('','NaN',inplace=True)

print(df.head(10))

df.fillna({'studentage': int(df['studentage'].mean())}, inplace=True)

df.fillna({'studentname': 'defaultname'}, inplace=True)

print("\nAfter removing missing values")

print(df.head(10))

df.drop\_duplicates(inplace=True)

df.drop\_duplicates(subset=["studentid"], inplace=True)

print("\nAfter removing duplicate values")

print(df.head(10))

plt.figure(figsize=(6,10))

sns.boxplot(data=df['studentid'])

plt.title("Boxplot data")

plt.ylabel("values")

plt.show()

Q1 = df['studentid'].quantile(0.25)

Q3 = df['studentid'].quantile(0.75)

IQR = Q3 - Q1

lower\_bound = Q1 - 1.5 \* IQR

upper\_bound = Q3 + 1.5 \* IQR

print(upper\_bound)

filter = (df['studentid'] >= lower\_bound)&(df['studentid'] <= upper\_bound)

print("\nOutliers are (for studentid):")

for index, row in df.iterrows():

if filter[index] == False:

print(row)

df = df[(df['studentid'] >= lower\_bound)&(df['studentid'] <= upper\_bound)]

print("\nAfter binning")

print(df.head(10))

**Code Explanation:**

pandas.fillna(): The fillna() method replaces the NULL values with a specified value.

Parameters:

1. Value to replace (any type)
2. inPlace (to create a dataframe copy or not)

pandas.drop\_duplicates(): The drop\_duplicates() method removes duplicate rows.

Use the subset parameter if only some specified columns should be considered when looking for duplicates.

Parameters:

1. Subset: which columns to consider
2. inPlace

sns.boxplot(): A box plot (or box-and-whisker plot) shows the distribution of quantitative data in a way that facilitates comparisons between variables or across levels of a categorical variable.

Parameters:

1. Data (any type)

**Dataset:**

studentid,studentname,studentage,studentcgp,studentpass,studentmobile

8627,Andrew Nelson,,3.41,True,6176967329

6777,Cameron Parsons,18,2.29,True,2695626827

6777,Cameron Parsons,18,2.29,True,2695626827

8627,Kristen Barnes,19,3.89,True,6124359911

8128,Paul Hardin,25,2.17,True,8375868679

47855,Jared Buckley,19,2.96,False,4624857797

6063,Sydney Mendoza,22,2.3,False,5571170750

8810,Henry Fisher,20,2.81,True,8353838882

2294,Jessica Scott,22,3.17,True,6695547433

3809,Alexa Miller,21,2.61,True,5804179625

8073,Damon Gonzalez,21,2.21,False,5254454523

6119,Andrea Richard PhD,21,2.87,False,7249349773

7082,Nicole Tran,22,2.04,False,1868329377

9607,David Park,24,3.32,True,4326816606

3938,Thomas Kennedy,22,2.82,True,1304563235

8284,Antonio Horton,19,3.46,True,5498692993

9166,Thomas Woodward,18,2.94,True,6275628927

3927,Brittany Roth,25,3.74,True,7147525727

2874,Christopher Rogers,20,3.35,False,3109025481

1496,Mary Wilson,20,2.63,False,3348030667

6133,Tracie Mckay,18,2.6,False,2282653263

7460,Daniel Lutz,20,2.35,False,7193854586

3256,Jonathan Adams,19,2.6,False,7418435153

9619,Alexandra Chavez,19,2.29,True,3333525493

7453,Michelle Washington,23,3.4,False,6257971024

4505,William Johnson,19,2.34,True,6185672386

8173,John Perez,20,3.88,False,9511307909

8516,David Johnson,21,3.0,True,3569921158

9256,Kimberly Schwartz,25,3.66,False,3024233431

4514,Eric Winters,18,2.73,True,2570938331

9697,Dawn Clarke,20,2.16,False,4357255804

5295,Tonya Guerrero,23,3.14,False,6768965128

5117,Dale Molina,22,3.6,False,7102951837

7101,Dawn Jones,21,2.69,False,7162375925

3007,Lisa Taylor,21,3.4,True,6680153792

6541,Nicole Spencer,20,3.58,False,2289769014

3917,Jake Martinez,24,2.77,False,3269719062

3319,Calvin Cooke,25,2.74,True,8813415708

1262,Wanda Thornton,24,2.84,False,4174846138

2515,Tara White,18,2.42,True,2939006416

2773,John Moore,19,2.35,True,9736918813

9865,Dalton Livingston,18,2.43,True,2589686756

6048,Brenda Pena,20,2.79,True,6245240364

2122,Jasmine Wood,19,3.74,True,6504776730

6653,Abigail Burgess,22,3.01,True,5759152251

3453,James Estes,20,3.4,True,1682274782

4838,Shawn Vance,21,2.63,False,4756641641

9945,Robert Schroeder,22,3.41,True,7463214857

6023,Dr. Joseph Rodgers Jr.,21,3.68,False,8115334966

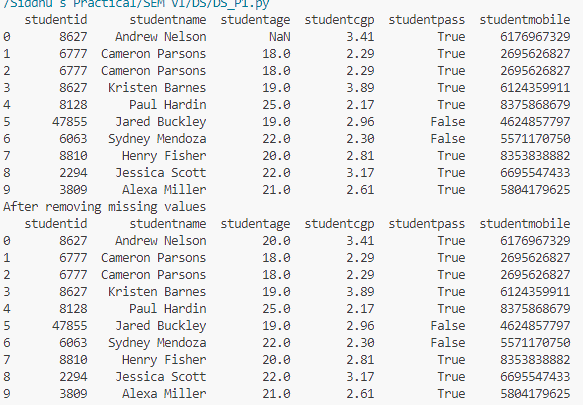
8841,Marcus Willis II,22,3.49,True,1302517401

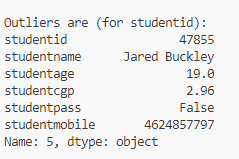
9353,Mark Gutierrez,24,3.4,False,8385105590

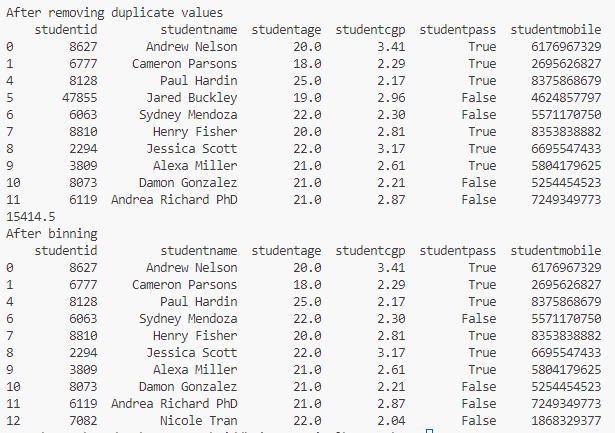
**Output Explanation:**

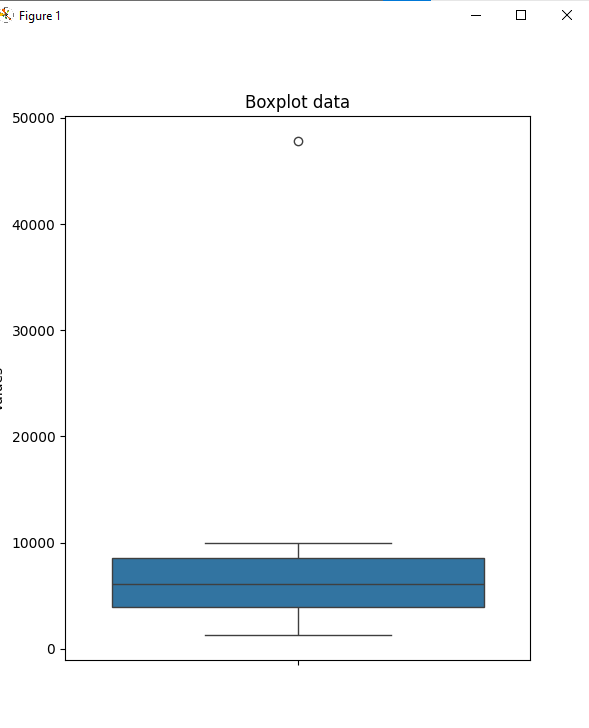
1. The original dataset is printed
2. Then missing values are filled and data is printed
3. Duplicate values are removed based defined columns and printed
4. We show the boxplot graph using sns
5. Calculate lower and upper bound for binning
6. Print Outliners

**Output:**









Practical No. 2

Data Cleaning - 2

**Q.1]** Do the following data cleaning

1. Data in wrong format
2. Scaling min-max and z-score normalization

**Data:**

studentid,studentname,studentage,studentcgp,studentpass,studentmobile,studentdob

6653,Abigail Burgess,22.0,3.01,True,5759152251,2003-12-01

3453,James Estes,20.0,9.8,True,1682274782,14/04/2004

4838,1234,21,2.63,False,4756641641,16 8 2006

9945,Robert Schroeder,22.3,7.8,True,7463214857,2004.9.30

6023,Dr. Joseph Rodgers Jr.,21,3.68,False,8115334966,2004-11-01

8841,Marcus Willis II,22,3.49,True,1302517401,20060915

9353,Mark Gutierrez,24,3.4,False,8385105590,19/08/2003

**Code:**

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

import seaborn as sns

df = pd.read\_csv('student\_data\_1.csv')

print(type(df['studentcgp']))

print("Before Date formating")

print(df.head(10))

df['studentdob'] = pd.to\_datetime(df['studentdob'], format='mixed', dayfirst=True)

print("After Date formating")

print(df.head(10))

df['studentage'] = df['studentage'].astype(int)

print("Afer studentage type formating as int")

print(df.head(10))

def convert\_cgpa\_10\_to\_4(cgpa\_10):

if 0 <= cgpa\_10 <= 10:

cgpa\_4 = (cgpa\_10 / 10) \* 4

return round(cgpa\_4, 1)

else:

raise ValueError("CGPA on a 10-point scale should be between 0 and 10")

for index, row in df.iterrows():

if row['studentcgp'] > 4.0:

df.loc[index, 'studentcgp'] = convert\_cgpa\_10\_to\_4(row['studentcgp'])

print("After correcting 10 point cgpa to 4 point cgpa")

print(df.head(10))

def min\_max\_scaling(col, new\_min, new\_max):

return (col - col.min()) / (col.max() - col.min()) \* (new\_max - new\_min) + new\_min

def z\_score\_normalization(col):

return (col - col.mean()) / col.std()

for indx, row in df.iterrows():

if str(row['studentname']).isdigit():

df.drop(indx, axis=0, inplace=True)

print("After removing invalid data in col studentname")

print(df.head(10))

df['cgp\_scaled'] = min\_max\_scaling(df['studentcgp'],1,10)

print("After Adding column for min-max scaling")

print(df.head(10))

df['cgp\_z\_score'] = z\_score\_normalization(df['studentcgp'])

print("After adding column for z-score normalization")

print(df.head(10))

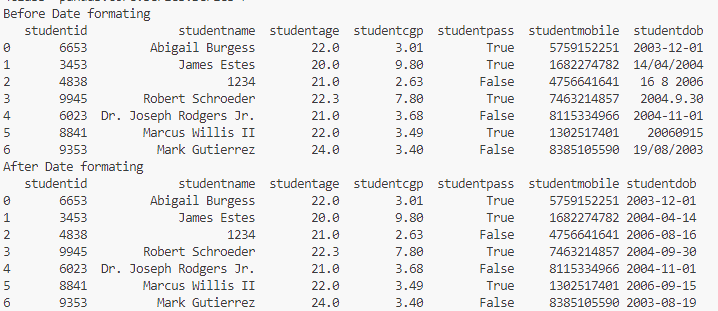
**Code Explanation:**

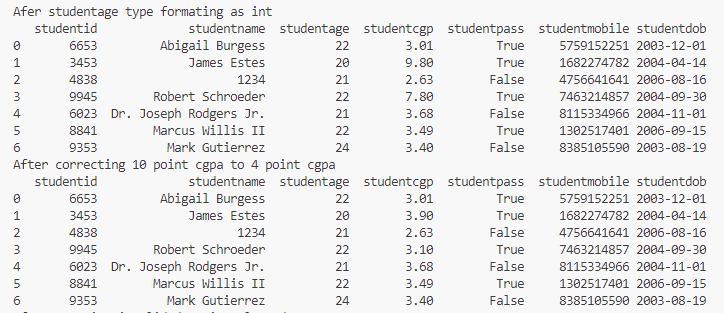
For Formatting the date column in the data we use the pd.to\_datetime() inbuilt function to convert all the mix date formats into one format

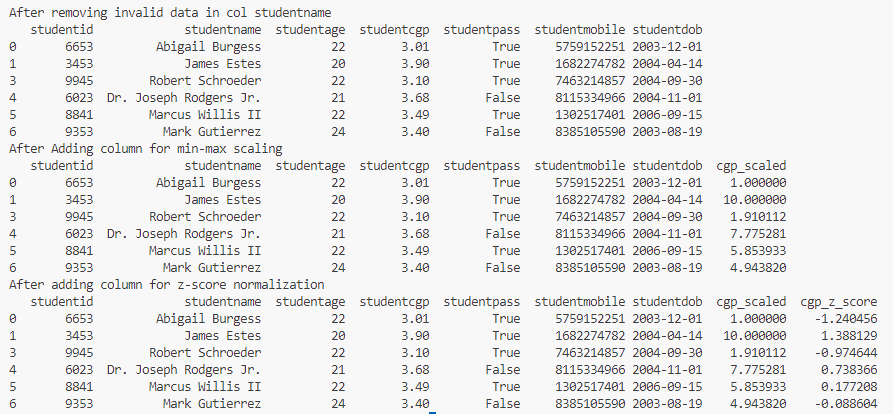
For correcting the data format in ‘studentage’ format, converting the values to int if some values are in float. For that we use astype(int) to convert the whole column

We define a custom function to convert the 10 point cgpa to 4 point cgpa. A loop iterates over the rows and check if the value is in 10 point scale and converts it.

**Output:**







**Output Explanation**:

First the data after correcting the date format is displayed.

After that student type is formatted

Then Grade is converted to 4 point scale

At last invalid data in studentname columns is removed (can be replaced with Default values)

Practical No. 3

Data Preprocessing

**Q.1]** Do the following on a dataframe

1. Correlation (Heatmap)
2. Chi-Square

**Data:**

studentid,studentname,studentage,studentcgp,studentpass,studentmobile,studentdob,studentgen

6653,Abigail Burgess,22.0,3.01,True,5759152251,2003-12-01,male

3453,James Estes,20.0,9.8,True,1682274782,14/04/2004,female

4838,1234,21,2.63,False,4756641641,16 8 2006,male

9945,Robert Schroeder,22.3,7.8,True,7463214857,2004.9.30,female

6023,Dr. Joseph Rodgers Jr.,21,3.68,False,8115334966,2004-11-01,female

8841,Marcus Willis II,22,3.49,True,1302517401,20060915,male

9353,Mark Gutierrez,24,3.4,False,8385105590,19/08/2003,male

**Code:**

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

import seaborn as sns

from scipy.stats import chi2\_contingency

df = pd.read\_csv('student\_data\_1.csv')

df['studentdob'] = pd.to\_datetime(df['studentdob'], format='mixed', dayfirst=True)

df['studentage'] = df['studentage'].astype(int)

def convert\_cgpa\_10\_to\_4(cgpa\_10):

if 0 <= cgpa\_10 <= 10:

cgpa\_4 = (cgpa\_10 / 10) \* 4

return round(cgpa\_4, 1)

else:

raise ValueError("CGPA on a 10-point scale should be between 0 and 10")

for index, row in df.iterrows():

if row['studentcgp'] > 4.0:

df.loc[index, 'studentcgp'] = convert\_cgpa\_10\_to\_4(row['studentcgp'])

def min\_max\_scaling(col, new\_min, new\_max):

return (col - col.min()) / (col.max() - col.min()) \* (new\_max - new\_min) + new\_min

def z\_score\_normalization(col):

return (col - col.mean()) / col.std()

for indx, row in df.iterrows():

if str(row['studentname']).isdigit():

df.drop(indx, axis=0, inplace=True)

print(df.head(10))

print(df.corr(numeric\_only=True))

dataplot = sns.heatmap(df.corr(numeric\_only=True),cmap="YlGnBu", annot=True)

plt.show()

contingency\_table = pd.crosstab(df['studentpass'],df['studentgen'])

chi2, p, dof, expected = chi2\_contingency(contingency\_table)

print("Chi-Square on studentpass and studentgender column")

print("Chi-Square Statistic:", chi2)

print("P-value:", p)

print("Degrees of Freedom:", dof)

print("Expected Frequencies:\n", expected)

alpha = 0.05

if p < alpha:

print("\nReject the null hypothesis: There is a significant relationship between student pass/fail and gender.")

else:

print("\nFail to reject the null hypothesis: No significant relationship between student pass/fail and gender.")

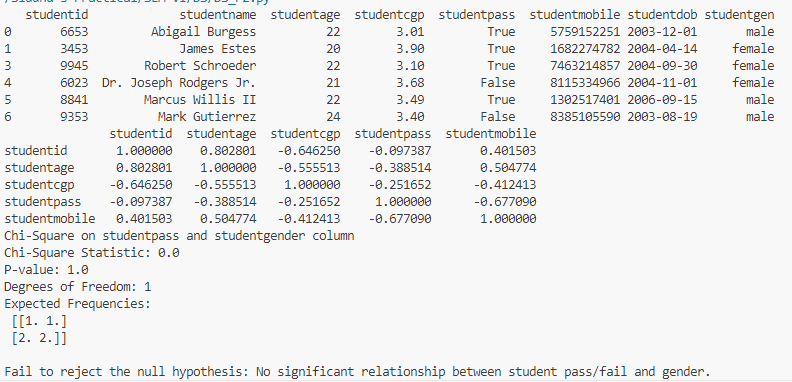
**Code Explanation:**

We use the sns.heatmap() function from the seaborn package to display the heatmap. The parameters include to only consider numeric values, colors of the graph, and whether to include annotation or not.

For the Chi-Square test we use the scipy library’s chi2\_contingency() function to perform the Chi-Square. The parameters we pass are the two columns for our dataframe to which we are performing the chi-square test.

**Output:**





**Output Explanation:**

The first output show heatmap of correlation between different columns of the dataframe

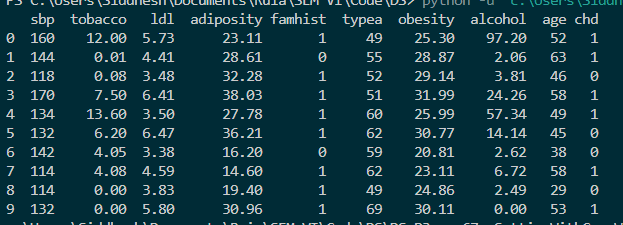
The Second output show the Chi-Square test performed on ‘studentpass’ and ‘studentgen’ column of the data

Practical No. 4

EDA

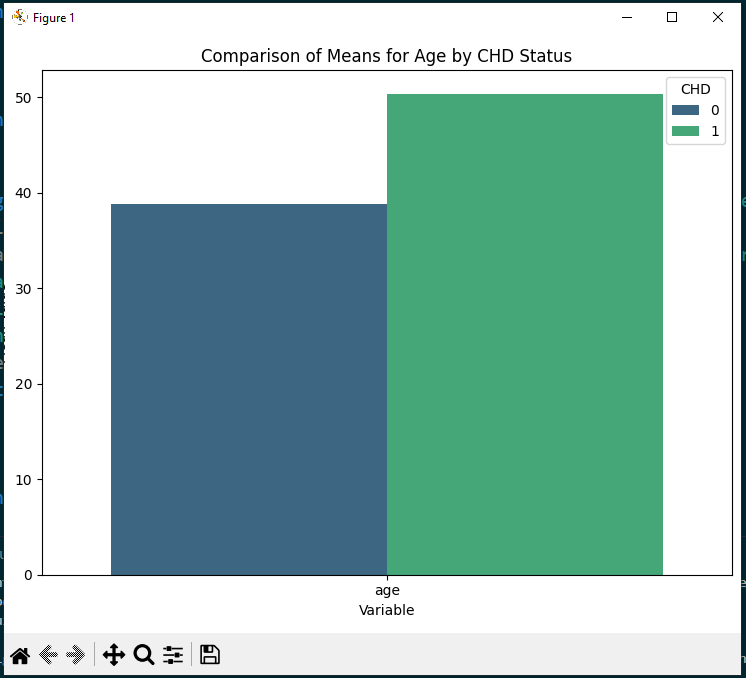
**Q.1]** Perform EDA on a dataset using Python.

**Dataset (df.head(10)):**



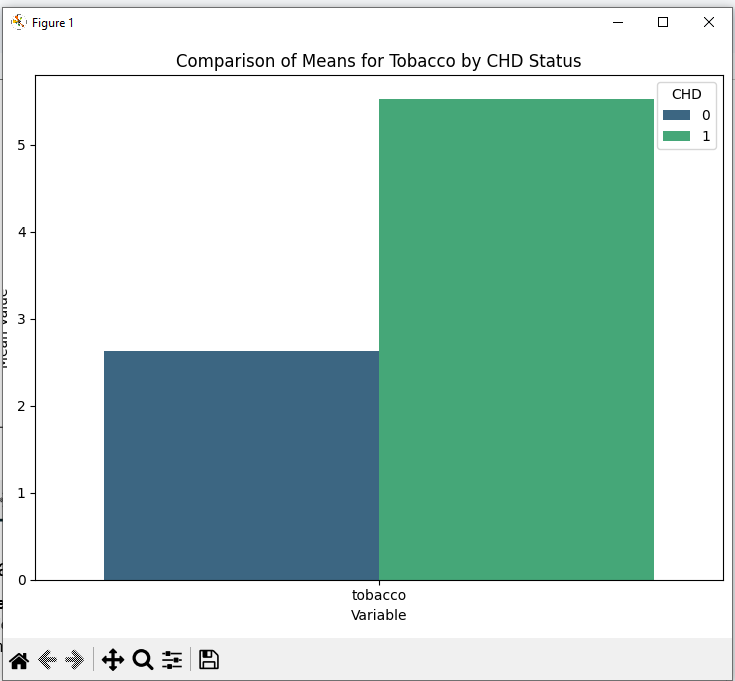
## Bar graph 1

**Inference:** The above chart shows that the population with less age mean has no CHD while the population with higher age mean has greater chances of CHD. Therefore, age correlates with CHD



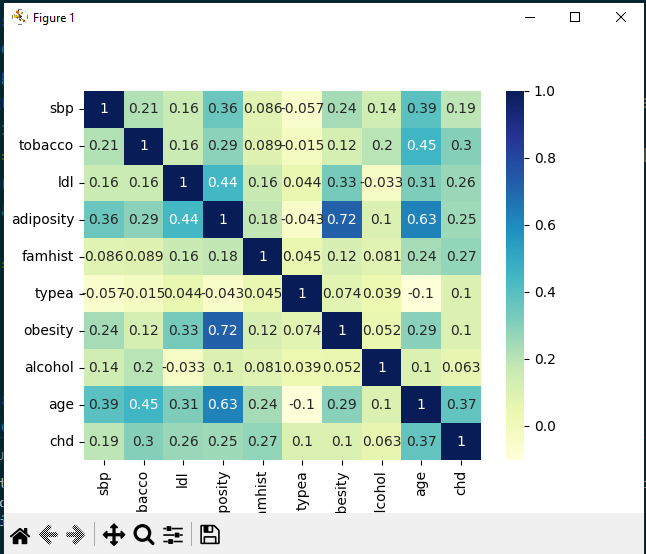
## Bar graph 2

**Inference:** Similarly the above chart shows that the population with less tobacco mean has no CHD while the population with higher tobacco mean has far greater chances of CHD. Therefore, tobacco highly correlates with CHD



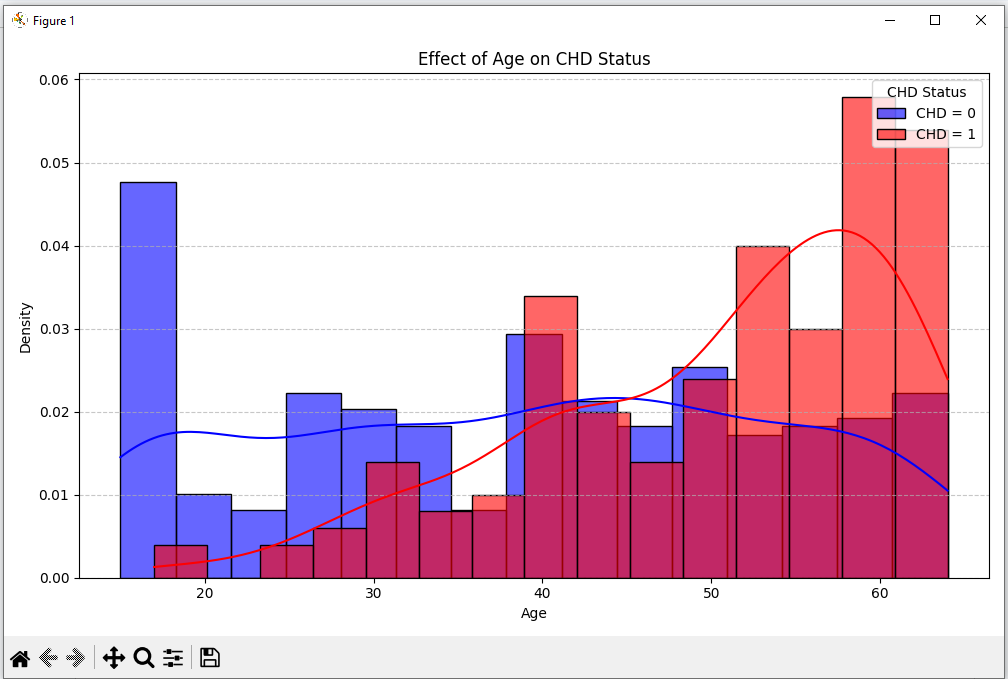
## Heatmap

**Inference:** The heatmap shows that age and tobacco highly correlates with CHD and also adiposity and obesity. It also shows that alcohol and tobacco are not as correlated as thought. Interestingly obesity is also not that correlated to CHD and famhist has no effect.



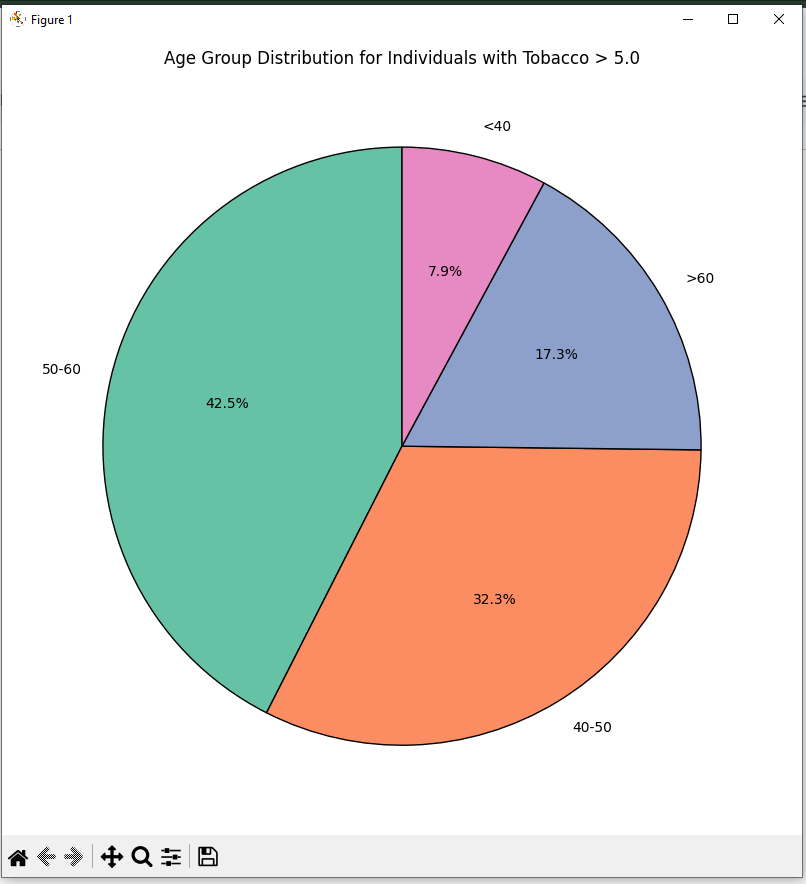
## Histogram

**Inference:** The age distribution skews older, suggesting a sample more at risk for CHD validating the information given by heatmap



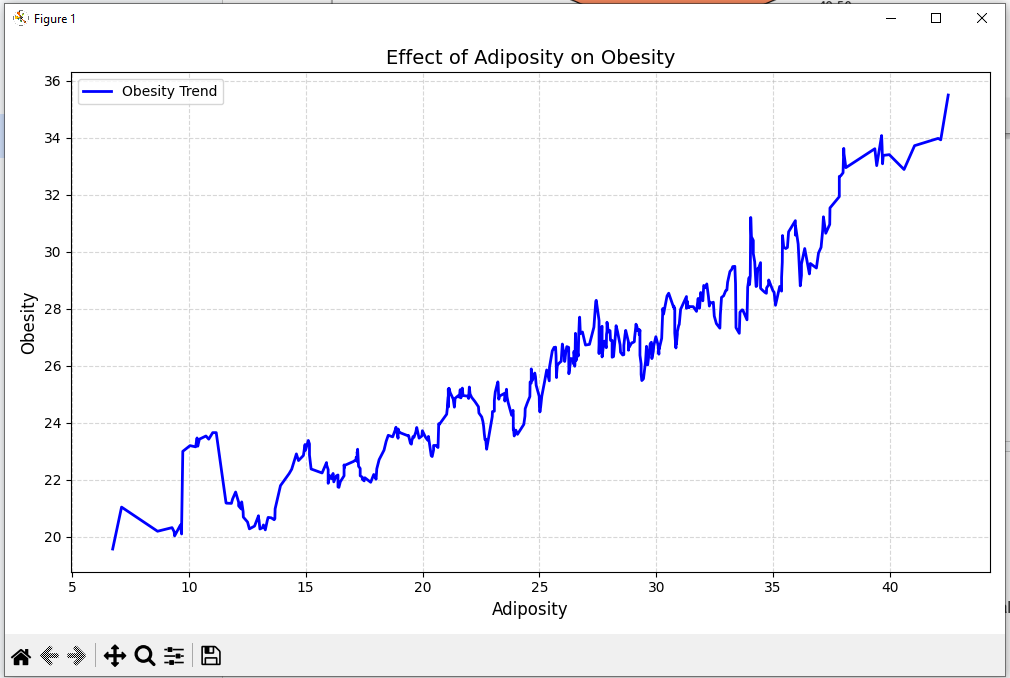
## Pie Chart

**Inference:** he majority of individuals with high tobacco consumption are between 50–70 years old and have high risk of CHD



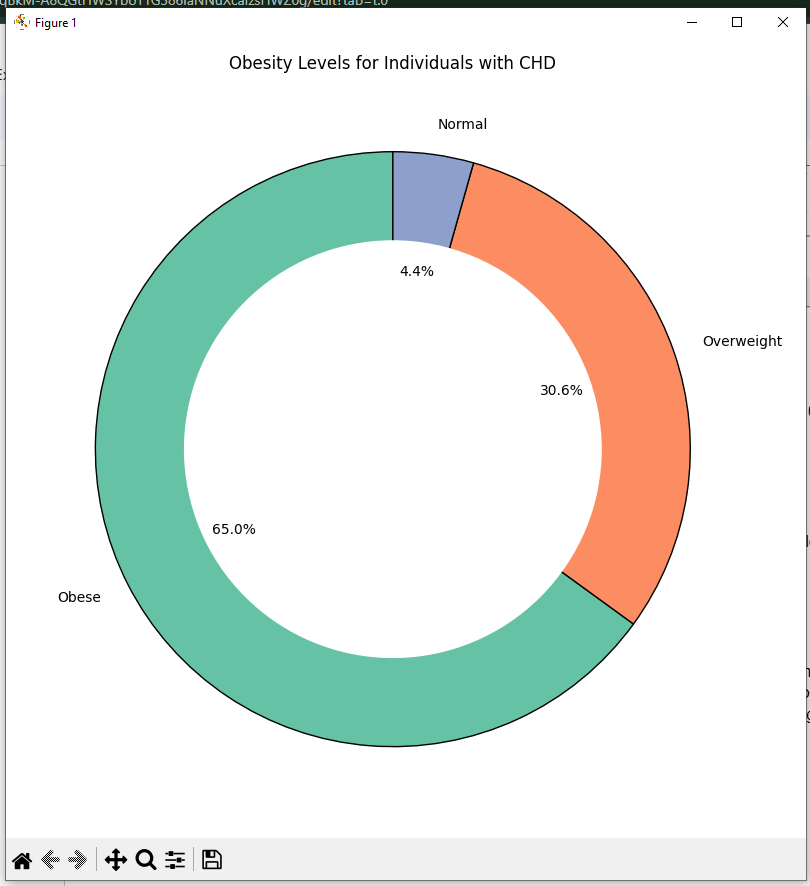
## Line Chart

**Inference:** There is a positive trend between adiposity and obesity, indicating that higher adiposity correlates with higher obesity levels.



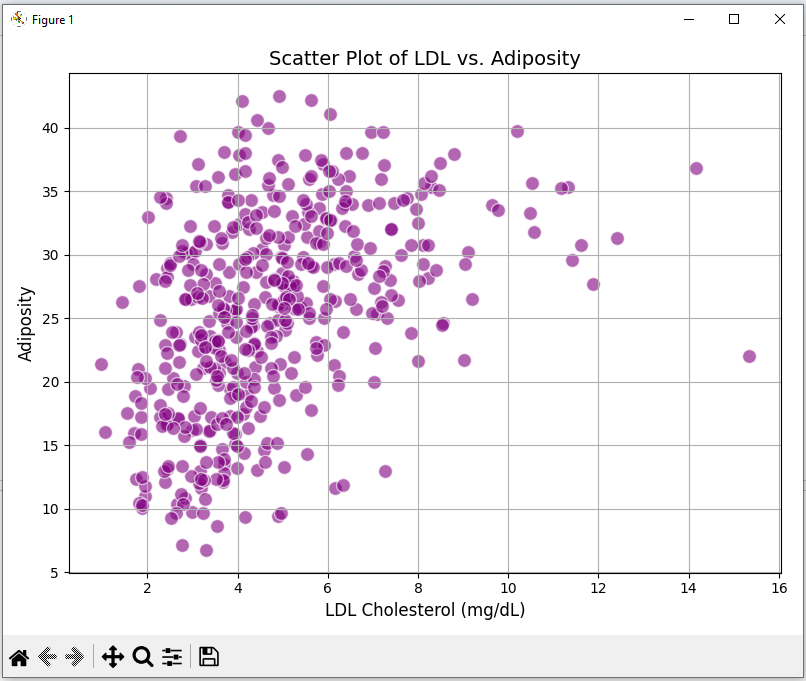
## Donut Chart

**Inference:** Among individuals with CHD, 60% are obese, 30% are overweight, and only 10% fall into the normal weight category, emphasizing obesity as a significant risk factor.



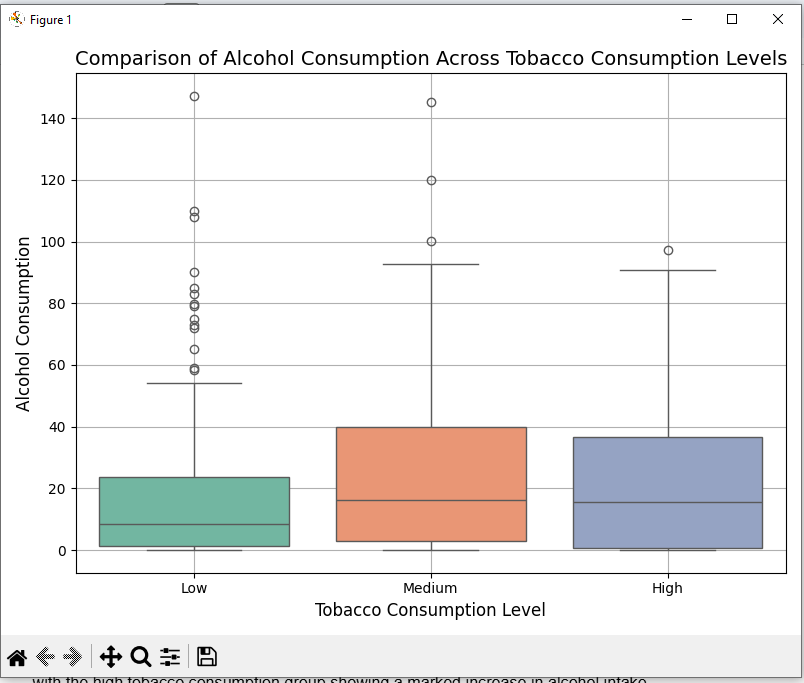
## Scatter Plot

**Inference:** There is a noticeable trend where higher adiposity correlates with higher LDL levels, indicating a potential relationship between body fat and cholesterol levels.



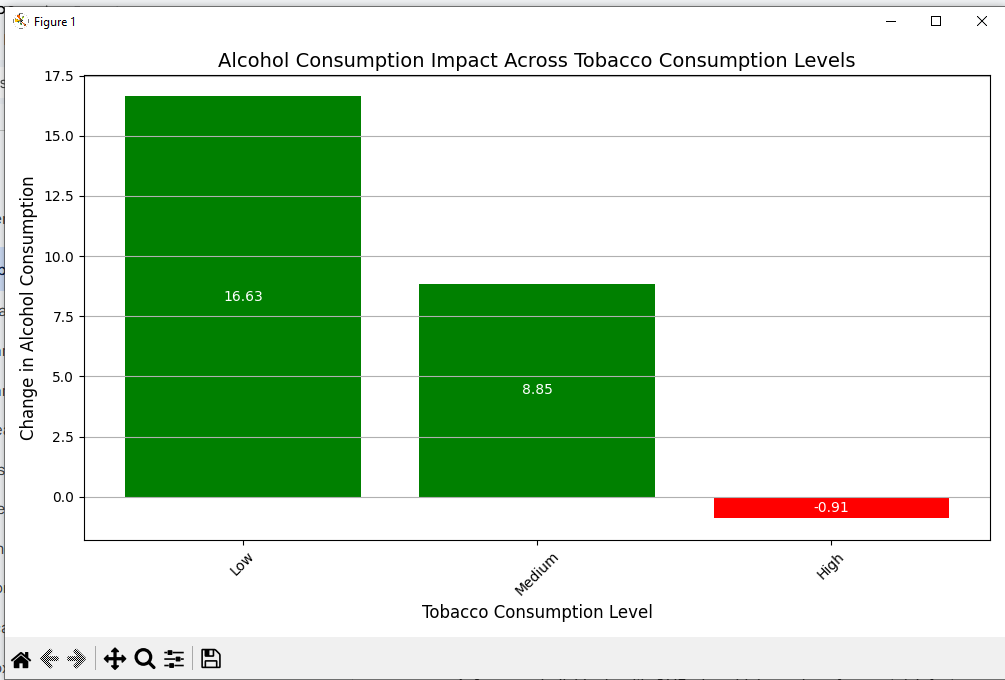
## Box Plot

**Inference:** Higher tobacco consumption is correlated with higher alcohol intake, as shown by the increasing median and spread of alcohol consumption in the high tobacco consumption group. But there are some outliers where individuals have low alcohol consumption but high tobacco consumption.



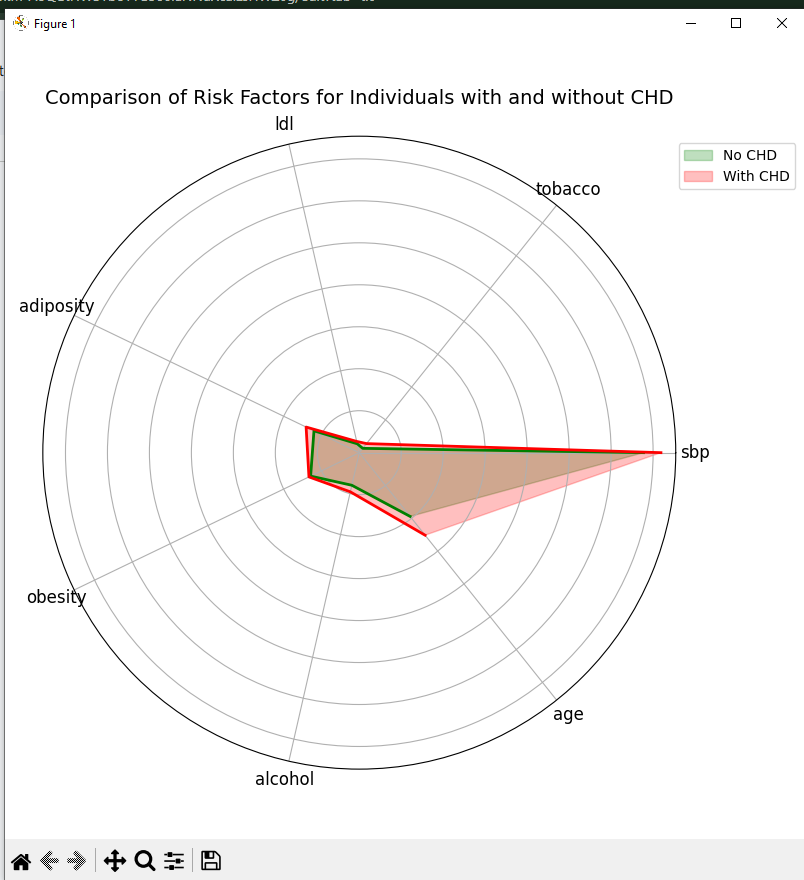
## Waterfall Chart

**Inference:** As tobacco consumption increases, alcohol consumption tends to increase as well, with the high tobacco consumption group showing a marked increase in alcohol intake.



## Radar Chart

**Inference:** Individuals with CHD show higher values for most risk factors, such as tobacco use, and age, compared to individuals without CHD.



**Code:**

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

import seaborn as sns

df = pd.read\_csv('SAheart.csv')

for index, row in df.iterrows():

if row['famhist'] == "Present":

df.loc[index, 'famhist'] = 1

else:

df.loc[index, 'famhist'] = 0

if row['chd'] == "Si":

df.loc[index, 'chd'] = 1

else:

df.loc[index, 'chd'] = 0

def min\_max\_scaling(col, new\_min, new\_max):

return (col - col.min()) / (col.max() - col.min()) \* (new\_max - new\_min) + new\_min

print(df.head(10))

dataplot = sns.heatmap(df.corr(),cmap="YlGnBu", annot=True)

plt.show()

# Bar chart 1

numerical\_columns = ['age']

grouped\_means = df.groupby('chd')[numerical\_columns].mean()

grouped\_means = grouped\_means.reset\_index()

melted\_means = grouped\_means.melt(id\_vars='chd', var\_name='Variable', value\_name='Mean Value')

plt.figure(figsize=(12, 6))

sns.barplot(data=melted\_means, x='Variable', y='Mean Value', hue='chd', palette='viridis')

plt.title('Comparison of Means for Age by CHD Status')

plt.xlabel('Variable')

plt.ylabel('Mean Value')

plt.legend(title='CHD', loc='upper right')

plt.tight\_layout()

plt.show()

# Bar chart 2

numerical\_columns = ['tobacco']

grouped\_means = df.groupby('chd')[numerical\_columns].mean()

grouped\_means = grouped\_means.reset\_index()

melted\_means = grouped\_means.melt(id\_vars='chd', var\_name='Variable', value\_name='Mean Value')

plt.figure(figsize=(12, 6))

sns.barplot(data=melted\_means, x='Variable', y='Mean Value', hue='chd', palette='viridis')

plt.title('Comparison of Means for Tobacco by CHD Status')

plt.xlabel('Variable')

plt.ylabel('Mean Value')

plt.legend(title='CHD', loc='upper right')

plt.tight\_layout()

plt.show()

# Histogram

plt.figure(figsize=(10, 6))

sns.histplot(data=df[df['chd'] == 0], x='age', bins=15, color='blue', label='CHD = 0', kde=True, stat="density", alpha=0.6)

sns.histplot(data=df[df['chd'] == 1], x='age', bins=15, color='red', label='CHD = 1', kde=True, stat="density", alpha=0.6)

plt.title('Effect of Age on CHD Status')

plt.xlabel('Age')

plt.ylabel('Density')

plt.legend(title='CHD Status', loc='upper right')

plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.tight\_layout()

plt.show()

# Pie chart

filtered\_df = df[df['tobacco'] > 5.0]

bins = [0, 40, 50, 60, 100]

labels = ['<40', '40-50', '50-60', '>60']

filtered\_df['Age Group'] = pd.cut(filtered\_df['age'], bins=bins, labels=labels)

age\_group\_counts = filtered\_df['Age Group'].value\_counts()

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

age\_group\_counts.plot.pie(

autopct='%1.1f%%',

colors=['#66c2a5', '#fc8d62', '#8da0cb', '#e78ac3'],

startangle=90,

wedgeprops={'edgecolor': 'black'}

)

plt.title('Age Group Distribution for Individuals with Tobacco > 5.0')

plt.ylabel('')

plt.tight\_layout()

plt.show()

# Line chart

df\_sorted = df.sort\_values(by='adiposity')

df\_sorted['Obesity Rolling Mean'] = df\_sorted['obesity'].rolling(window=10, min\_periods=1).mean()

plt.figure(figsize=(10, 6))

plt.plot(df\_sorted['adiposity'], df\_sorted['Obesity Rolling Mean'], color='blue', label='Obesity Trend', linewidth=2)

plt.title('Effect of Adiposity on Obesity', fontsize=14)

plt.xlabel('Adiposity', fontsize=12)

plt.ylabel('Obesity', fontsize=12)

plt.grid(alpha=0.5, linestyle='--')

plt.legend(loc='upper left')

plt.tight\_layout()

plt.show()

# Donut chart

bins = [0, 20, 25, 100]

labels = ['Normal', 'Overweight', 'Obese']

df['Obesity Level'] = pd.cut(df['obesity'], bins=bins, labels=labels)

chd\_df = df[df['chd'] == 1]

obesity\_counts = chd\_df['Obesity Level'].value\_counts()

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

colors = ['#66c2a5', '#fc8d62', '#8da0cb']

plt.pie(

obesity\_counts,

labels=obesity\_counts.index,

autopct='%1.1f%%',

colors=colors,

startangle=90,

wedgeprops={'edgecolor': 'black'}

)

center\_circle = plt.Circle((0, 0), 0.70, color='white')

plt.gca().add\_artist(center\_circle)

plt.title('Obesity Levels for Individuals with CHD')

plt.tight\_layout()

plt.show()

# Scatter plot

plt.figure(figsize=(8, 6))

plt.scatter(df['ldl'], df['adiposity'], color='purple', alpha=0.6, edgecolors='w', s=100)

plt.title('Scatter Plot of LDL vs. Adiposity', fontsize=14)

plt.xlabel('LDL Cholesterol (mg/dL)', fontsize=12)

plt.ylabel('Adiposity', fontsize=12)

plt.grid(True)

plt.tight\_layout()

plt.show()

# Boxplot

bins = [0, 5, 10, 20]

labels = ['Low', 'Medium', 'High']

df['Tobacco Level'] = pd.cut(df['tobacco'], bins=bins, labels=labels)

plt.figure(figsize=(8, 6))

sns.boxplot(x='Tobacco Level', y='alcohol', data=df, palette='Set2')

plt.title('Comparison of Alcohol Consumption Across Tobacco Consumption Levels', fontsize=14)

plt.xlabel('Tobacco Consumption Level', fontsize=12)

plt.ylabel('Alcohol Consumption', fontsize=12)

plt.grid(True)

plt.tight\_layout()

plt.show()

# Waterfall Chart

bins = [0, 5, 10, 20]

labels = ['Low', 'Medium', 'High']

df['Tobacco Level'] = pd.cut(df['tobacco'], bins=bins, labels=labels)

tobacco\_alcohol\_avg = df.groupby('Tobacco Level')['alcohol'].mean()

cumulative\_diff = tobacco\_alcohol\_avg.diff().fillna(tobacco\_alcohol\_avg)

tobacco\_levels = tobacco\_alcohol\_avg.index

alcohol\_diff = cumulative\_diff

plt.figure(figsize=(10, 6))

plt.bar(tobacco\_levels, alcohol\_diff, color=['green' if x >= 0 else 'red' for x in alcohol\_diff])

for i, v in enumerate(alcohol\_diff):

plt.text(i, alcohol\_diff[i] / 2, f'{v:.2f}', ha='center', va='center', color='white', fontsize=10)

plt.title('Alcohol Consumption Impact Across Tobacco Consumption Levels', fontsize=14)

plt.xlabel('Tobacco Consumption Level', fontsize=12)

plt.ylabel('Change in Alcohol Consumption', fontsize=12)

plt.xticks(rotation=45)

plt.grid(axis='y')

plt.tight\_layout()

plt.show()

# Radar Chart

risk\_factors = ['sbp', 'tobacco', 'ldl', 'adiposity', 'obesity', 'alcohol', 'age']

chd\_risk\_factors = df.groupby('chd')[risk\_factors].mean()

angles = np.linspace(0, 2 \* np.pi, len(risk\_factors), endpoint=False).tolist()

chd\_risk\_factors = chd\_risk\_factors.T

chd\_risk\_factors = chd\_risk\_factors[[0, 1]]

chd\_risk\_factors = chd\_risk\_factors.T

fig, ax = plt.subplots(figsize=(8, 8), subplot\_kw=dict(polar=True))

ax.fill(angles, chd\_risk\_factors.iloc[0], color='green', alpha=0.25, label='No CHD')

ax.plot(angles, chd\_risk\_factors.iloc[0], color='green', linewidth=2)

ax.fill(angles, chd\_risk\_factors.iloc[1], color='red', alpha=0.25, label='With CHD')

ax.plot(angles, chd\_risk\_factors.iloc[1], color='red', linewidth=2)

ax.set\_yticklabels([])

ax.set\_xticks(angles)

ax.set\_xticklabels(risk\_factors, fontsize=12)

plt.title('Comparison of Risk Factors for Individuals with and without CHD', size=14, color='black', va='bottom')

plt.legend(loc='upper right', bbox\_to\_anchor=(1.2, 1.0))

plt.tight\_layout()

plt.show()

""" first\_ten = df.head(10)

first\_ten = first\_ten[df.columns[[1, 2, 8, 9]]]

first\_ten = first\_ten.sort\_values(by=['chd'])

first\_ten['ldl'] = min\_max\_scaling(first\_ten['ldl'],0,10)

first\_ten['chd'] = min\_max\_scaling(first\_ten['chd'],0,10)

first\_ten.plot.bar()

plt.show()

print(df.corr(numeric\_only=True))

dataplot = sns.heatmap(df.corr(),cmap="YlGnBu", annot=True)

plt.show()

data\_his = df[df.columns[[8, 9]]]

for indx, row in data\_his.iterrows():

if row['chd'] == 0:

data\_his.drop(indx, axis=0, inplace=True)

data\_his.plot.hist(column='age')

plt.show()

pie\_data = df[df.columns[[4, 9]]]

for indx, row in pie\_data.iterrows():

if row['chd'] == 0:

pie\_data.drop(indx, axis=0, inplace=True)

hist, bins = np.histogram(pie\_data['famhist'], bins=2)

print(hist, bins)

pieDf = pd.DataFrame({'famhist': hist},index=['Present','Absent'])

pieDf.plot.pie(y='famhist',figsize=(5,5))

plt.show() """

Practical No. 5

Linear Regression

**Q.1]** Write a Python program to do Logic Regression to two columns of a dataset.

**Code:**

import numpy as np

import pandas as pd

from sklearn.linear\_model import LinearRegression

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

import matplotlib.pyplot as plt

import seaborn as sns

df = pd.read\_csv('SAheart.csv')

print(df.head(10))

X = df[['age']]

y = df['adiposity']

model = LinearRegression()

model.fit(X, y)

print(f"Coefficient: {model.coef\_[0]}") # Coefficients

print(f"Intercept: {model.intercept\_}") # intercept

main\_pred = model.predict(X)

age\_pred = np.array([15, 23, 31, 27]).reshape(-1, 1)

y\_pred = model.predict(age\_pred)

for index, val in enumerate(y\_pred):

print(f'{index} age: {age\_pred[index][0]} adiposity: {val}')

""" new\_ages = pd.DataFrame({'age': [25, 40, 60, 75]})

y\_pred = model.predict(new\_ages) """

plt.figure(figsize=(8, 6))

#sns.scatterplot(x='age', y='adiposity', data=df, color='blue', label='Data Points')

plt.plot(df['age'], main\_pred, color='green', label='Regression Line')

plt.plot(age\_pred, y\_pred, color='red', label='Predicted Line')

#sns.scatterplot(x='age', y='adiposity', data=pd.DataFrame({'age': age\_pred, 'adiposity': y\_pred}), color='orange', label='Data Points')

plt.xlabel('Age')

plt.ylabel('Adiposity')

plt.title('Linear Regression: Adiposity vs Age')

plt.legend()

plt.show()

mse = mean\_squared\_error(age\_pred, y\_pred) # Mean Squared Error

mae = mean\_absolute\_error(age\_pred, y\_pred) # Mean Absolute Error

r2 = r2\_score(age\_pred, y\_pred) # R-squared (R²)

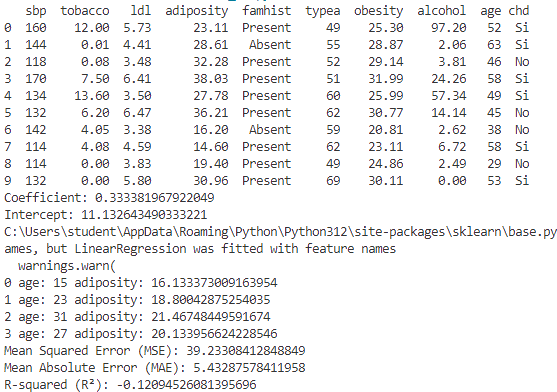
# Output the results

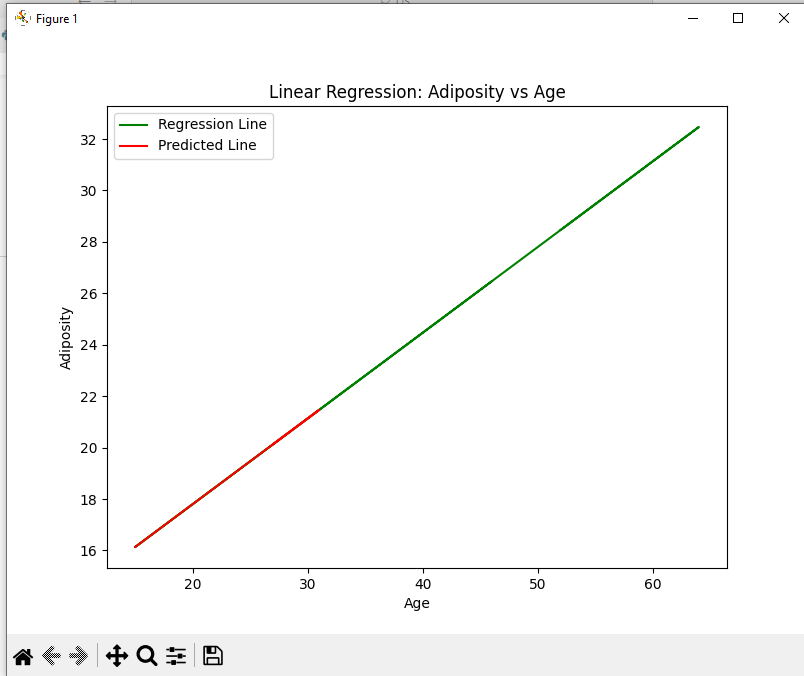
print(f"Mean Squared Error (MSE): {mse}")

print(f"Mean Absolute Error (MAE): {mae}")

print(f"R-squared (R²): {r2}")

**Output:**





**Inference:** The predicted values

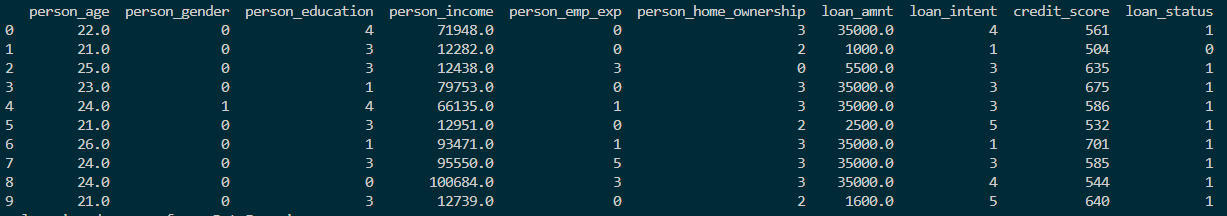
Practical No. 6

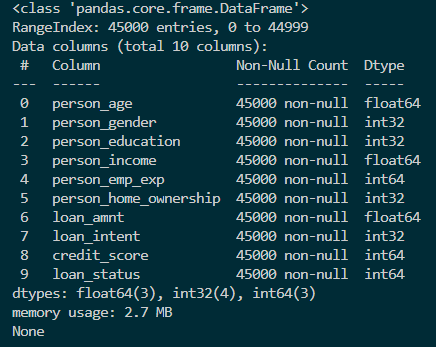
Comparing all Classification Algorithms

**Q.1]** Write Python code to compare all classification algorithms.

**Dataset:** [Link](https://www.kaggle.com/datasets/taweilo/loan-approval-classification-data)

**df.head(10)**





**Code:**

import numpy as np

import pandas as pd

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

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.naive\_bayes import GaussianNB, MultinomialNB

from sklearn.metrics import accuracy\_score

from sklearn.decomposition import PCA

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression, LogisticRegression

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

import seaborn as sns

df = pd.read\_csv('loan\_data.csv')

df = df.drop(['loan\_int\_rate', 'loan\_percent\_income', 'cb\_person\_cred\_hist\_length', 'previous\_loan\_defaults\_on\_file'],axis=1)

label\_encoder = LabelEncoder()

df['person\_gender'] = label\_encoder.fit\_transform(df['person\_gender'])

df['person\_education'] = label\_encoder.fit\_transform(df['person\_education'])

df['person\_home\_ownership'] = label\_encoder.fit\_transform(df['person\_home\_ownership'])

df['loan\_intent'] = label\_encoder.fit\_transform(df['loan\_intent'])

print(df.head(10))

print(df.info())

X =df[['person\_age','person\_gender','person\_education','person\_income','person\_emp\_exp','person\_home\_ownership','loan\_amnt','loan\_intent','credit\_score']]

y = df['loan\_status']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=1) # 70% training and 30% test

# Naives Bayes

print("\nNaives Bayes\n")

model = GaussianNB()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print(f'Accuracy: {accuracy \* 100:.2f}%')

sns.displot(df, x='person\_age', hue='loan\_status', kind='kde', fill=True)

sns.displot(df, x='person\_income', hue='loan\_status', kind='kde', fill=True)

plt.show()

# PCA

print("\nPCA\n")

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

pca = PCA(n\_components=4)

X\_train\_pca = pca.fit\_transform(X\_train\_scaled)

X\_test\_pca = pca.transform(X\_test\_scaled)

model = LogisticRegression(random\_state=42)

model.fit(X\_train\_pca, y\_train)

y\_pred = model.predict(X\_test\_pca)

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy with PCA-reduced features: {accuracy:.2f}")

# Variance explained by each principal component

explained\_variance = pca.explained\_variance\_ratio\_

print(f"Explained variance ratio by components: {explained\_variance}")

components = pd.DataFrame(pca.components\_, columns=['person\_age','person\_gender','person\_education','person\_income',

'person\_emp\_exp','person\_home\_ownership','loan\_amnt','loan\_intent',

'credit\_score'], index=[f"PC{i+1}" for i in range(pca.n\_components\_)])

plt.figure(figsize=(10, 6))

sns.heatmap(components, cmap='coolwarm', annot=True)

plt.title("Feature Contributions to Principal Components")

plt.xlabel("Features")

plt.ylabel("Principal Components")

plt.show()

# Linear Regression

print("\nLinear Regression\n")

model = LinearRegression()

model.fit(X\_train, y\_train)

print(f"Coefficient: {model.coef\_[0]}") # Coefficients

print(f"Intercept: {model.intercept\_}") # intercept

y\_pred = model.predict(X\_test)

mse = mean\_squared\_error(y\_test, y\_pred) # Mean Squared Error

mae = mean\_absolute\_error(y\_test, y\_pred) # Mean Absolute Error

r2 = r2\_score(y\_test, y\_pred) # R-squared (R²)

# Output the results

print(f"Mean Squared Error (MSE): {mse}")

print(f"Mean Absolute Error (MAE): {mae}")

print(f"R-squared (R²): {r2}")

print(f"Accuracy: {(1-r2)\*100}%")

sampled\_df = df.sample(n=100, random\_state=42)

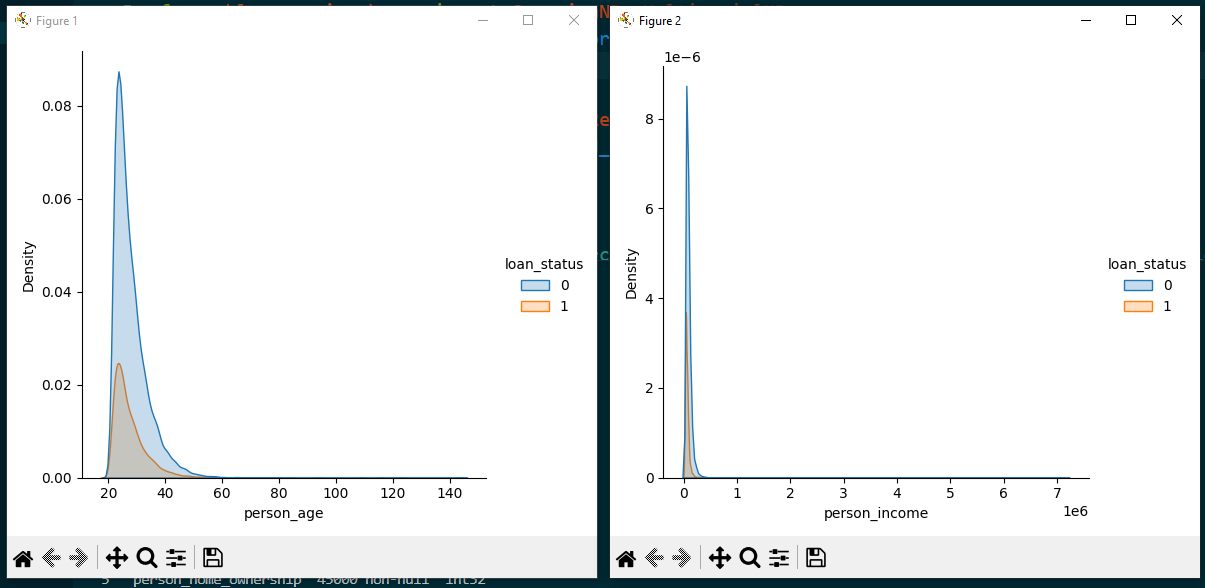
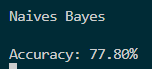
print(sampled\_df['person\_age'].median())

sns.pairplot(sampled\_df, x\_vars=['person\_age','person\_income'], y\_vars='loan\_status', height=5, aspect=0.8, kind='reg')

plt.show()

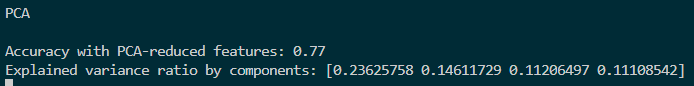
**Output:**

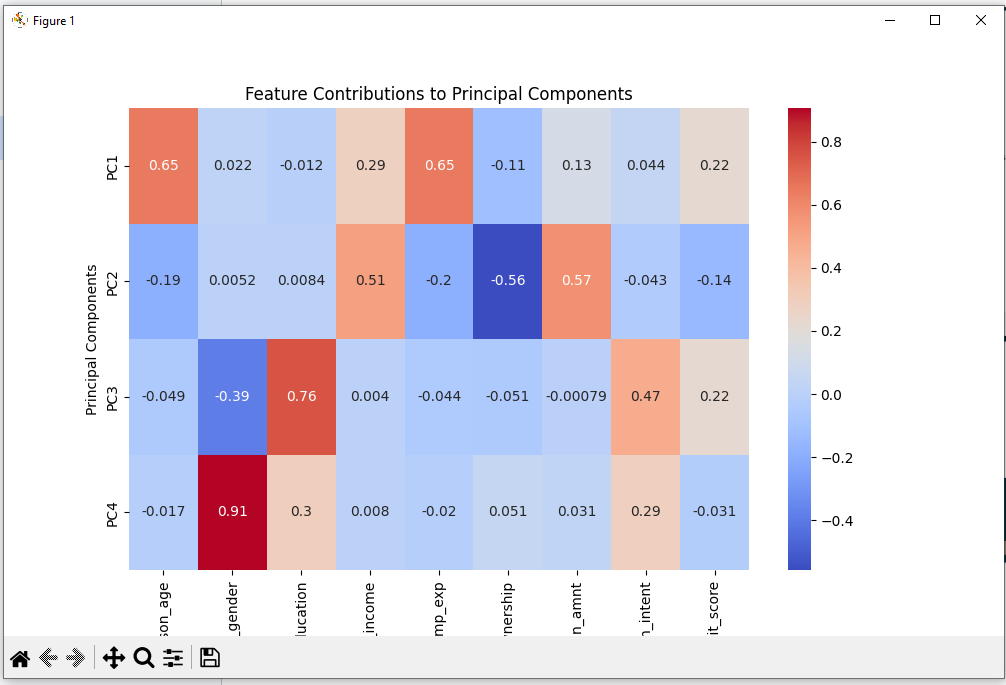
**Naives Bayes**



**Inference:** The above two graphs show the decision boundary used by Navies Bayes Model to make the predictions for the selected two features. From the first graph we can conclude that the young age people have higher chances of loan disapproval and constitute the most in the dataset.

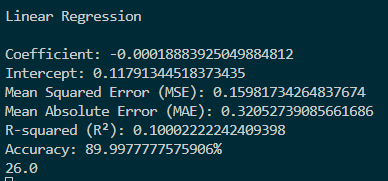
**PCA**

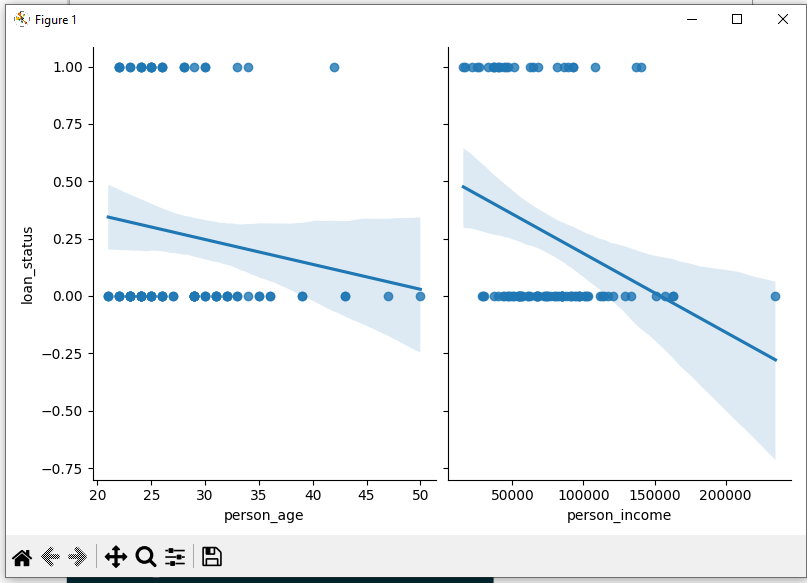




**Inference:** The above heatmap shows the correlationships between the reduced components and actual features. We can see that the person’s age, a person's income and a person's experience are the most contributing factors.

**Linear Regression**





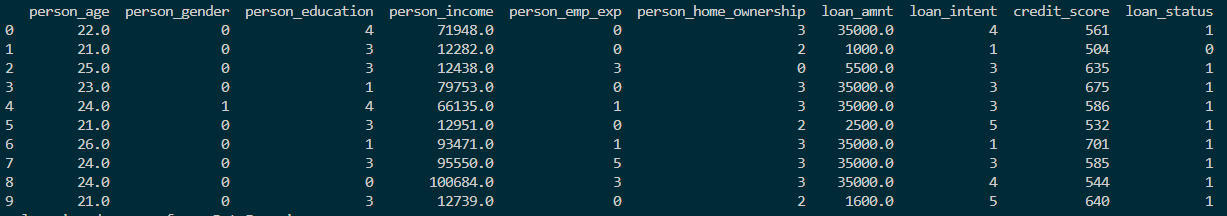
**Inference:** This a sns pairplot graph used to show multiple independent and one dependent variables linear regression. For each feature a separated graph is plotted, showing the regression line. For example, in the first graph, young age people have high rate of approval when compared to old age people.

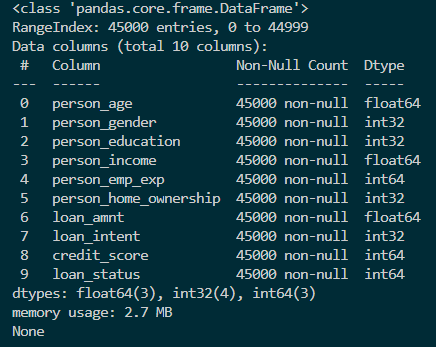
Practical No. 7

Improving Accuracy

**Q.1]** Use various techniques like Bagging, Boosting, Bootstrapping, and Cross-Validation to improve the model accuracy.

**Dataset:** [Link](https://www.kaggle.com/datasets/taweilo/loan-approval-classification-data)





**Code:**

import numpy as np

import pandas as pd

from sklearn import metrics

from sklearn.model\_selection import GridSearchCV, cross\_val\_score, train\_test\_split

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.metrics import accuracy\_score

import seaborn as sns

from sklearn.ensemble import BaggingClassifier

from sklearn.tree import DecisionTreeClassifier

import matplotlib.pyplot as plt

from sklearn.utils import resample

from sklearn.ensemble import AdaBoostClassifier

df = pd.read\_csv('loan\_data.csv')

df = df.drop(['loan\_int\_rate', 'loan\_percent\_income', 'cb\_person\_cred\_hist\_length', 'previous\_loan\_defaults\_on\_file'],axis=1)

label\_encoder = LabelEncoder()

df['person\_gender'] = label\_encoder.fit\_transform(df['person\_gender'])

df['person\_education'] = label\_encoder.fit\_transform(df['person\_education'])

df['person\_home\_ownership'] = label\_encoder.fit\_transform(df['person\_home\_ownership'])

df['loan\_intent'] = label\_encoder.fit\_transform(df['loan\_intent'])

print(df.head(10))

print(df.info())

X =df[['person\_age','person\_gender','person\_education','person\_income','person\_emp\_exp','person\_home\_ownership','loan\_amnt','loan\_intent','credit\_score']]

y = df['loan\_status']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=1) # 80% training and 30% test

clf = DecisionTreeClassifier(max\_depth=1, random\_state = 22)

clf2 = DecisionTreeClassifier(random\_state=22)

clf = clf.fit(X\_train,y\_train)

clf2 = clf2.fit(X\_train, y\_train)

print("\nMetrics")

y\_pred = clf.predict(X\_test)

clf\_accu = metrics.accuracy\_score(y\_test, y\_pred)

print(f"Accuracy: {clf\_accu \* 100:.2f}%")

# Bagging

estimator\_range = [4,8,12,24,36]

scores = []

models = []

for n\_estimators in estimator\_range:

bg = BaggingClassifier(estimator=clf2, n\_estimators = n\_estimators, random\_state = 22)

bg.fit(X\_train, y\_train)

models.append(bg)

scores.append(accuracy\_score(y\_true = y\_test, y\_pred = bg.predict(X\_test)))

plt.figure(figsize=(9,6))

plt.plot(estimator\_range, scores)

plt.xlabel("n\_estimators", fontsize = 18)

plt.ylabel("score", fontsize = 18)

plt.tick\_params(labelsize = 16)

plt.show()

print(f"\nAccuracy of 5th Bagging Modal: {scores[4] \* 100:.2f}%")

# Bootstrapping

accuracy = []

n\_iterations = 100

for i in range(n\_iterations):

X\_bs, y\_bs = resample(X\_test, y\_test, replace=True)

y\_hat = models[4].predict(X\_bs)

score = accuracy\_score(y\_bs, y\_hat)

accuracy.append(score)

sns.kdeplot(accuracy)

plt.title("Accuracy across 100 bootstrap samples of the held-out test set")

plt.xlabel("Accuracy")

plt.show()

# Boosting

ada = AdaBoostClassifier(estimator=clf, n\_estimators=50, learning\_rate=1.0, random\_state=22)

ada.fit(X\_train, y\_train)

ada\_accuracy = accuracy\_score(y\_test, ada.predict(X\_test))

print(f'\nAccuracy of the weak learner (Decision Tree): {clf\_accu \* 100:.2f}%')

print(f'Accuracy of AdaBoost model: {ada\_accuracy \* 100:.2f}%')

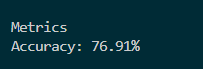
# Cross-Validation

cv\_scores = cross\_val\_score(clf2, X\_train, y\_train, cv=5, scoring='accuracy')

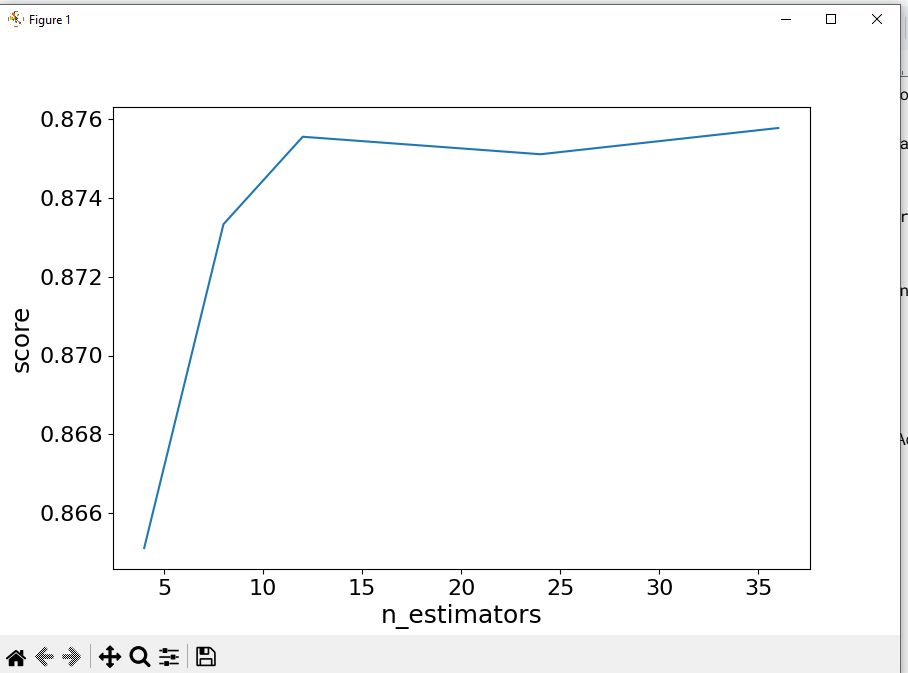
print(f'\nCross-Validation Results (Accuracy): {cv\_scores}')

print(f'Mean Accuracy: {cv\_scores.mean()}', '\n')

**Output:**

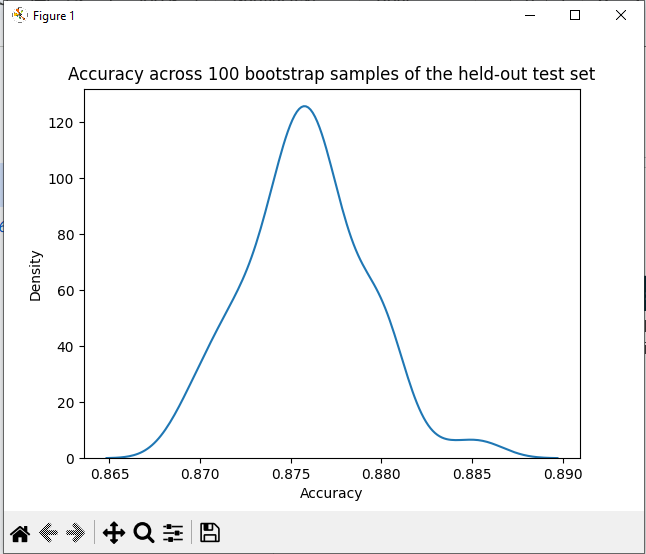


This is base accuracy for DTC model with max\_depth = 1 (for further AdaBoostClassifier use) and for model with no max\_depth the accuracy is around 81%

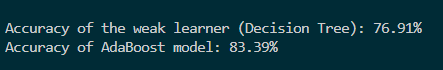




The graph shows the accuracies for the 5 estimators we choose and highest accuracy was for 5th Bagging modal which is 87.58%, improvement over previous 81%



The above graph shows the result of bootstrapping on the 5th Bagging model over 100 iterations. From that we can calculator the 95th percentile accuracy for the model



AdaBoosting has improvement over the DTC was max\_depth=1



Practical No. 8

Clustering

**Q.1]** Implement clustering in Python for a dataset using the following two methods.

1. Elbow method/graph
2. Silhouette coefficients

**Code:**

import numpy as np

import pandas as pd

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

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.metrics import accuracy\_score

from sklearn.decomposition import PCA

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.cluster import KMeans, AgglomerativeClustering

from sklearn.metrics import silhouette\_score

from scipy.cluster.hierarchy import dendrogram, linkage

df = pd.read\_csv('loan\_data.csv')

df = df.drop(['loan\_int\_rate', 'loan\_percent\_income', 'cb\_person\_cred\_hist\_length', 'previous\_loan\_defaults\_on\_file'],axis=1)

df = df.loc[:2500]

label\_encoder = LabelEncoder()

df['person\_gender'] = label\_encoder.fit\_transform(df['person\_gender'])

df['person\_education'] = label\_encoder.fit\_transform(df['person\_education'])

df['person\_home\_ownership'] = label\_encoder.fit\_transform(df['person\_home\_ownership'])

df['loan\_intent'] = label\_encoder.fit\_transform(df['loan\_intent'])

print(df.head(10))

print(df.info())

print(df.describe())

print(df.columns)

# Prepare data for clustering

new\_data = df.drop(columns=['loan\_status'])

df2 = new\_data.copy()

df2 = df2.loc[:200]

print(new\_data.head())

# K-Means clustering

kmeans = KMeans(n\_clusters=3, random\_state=42)

kmeans.fit(new\_data)

df['kmeans\_cluster'] = kmeans.labels\_

# Elbow method to find optimal k

wss = []

for k in range(1, 16):

kmeans = KMeans(n\_clusters=k, random\_state=42)

kmeans.fit(new\_data)

wss.append(kmeans.inertia\_)

plt.plot(range(1, 16), wss, marker='o')

plt.xlabel('Number of clusters k')

plt.ylabel('Total within-clusters sum of square')

plt.show()

# Standardize the data

scaler = StandardScaler()

X\_std = scaler.fit\_transform(new\_data)

# Reduce dimensions for visualization (you can skip this step if you have fewer features)

pca = PCA(n\_components=2)

X\_pca = pca.fit\_transform(X\_std)

# Visualize the clusters

#plt.scatter(X\_pca[:, 0], X\_pca[:, 1], c=kmeans.labels\_, cmap='viridis')

plt.scatter(new\_data['person\_income'], new\_data['loan\_amnt'], c=kmeans.labels\_, cmap='viridis')

plt.title('K-means Clustering')

plt.xlabel('person\_income')

plt.ylabel('loan\_amnt')

plt.show()

# Silhouette Score for K-Means

silhouette\_avg = silhouette\_score(new\_data, df['kmeans\_cluster'])

print("Mean Silhouette Width for K-Means Clustering:", silhouette\_avg)

# Hierarchical clustering

linkage\_matrix = linkage(df2, method='ward')

dendrogram(linkage\_matrix)

plt.title('Hierarchical Clustering Dendrogram')

plt.show()

# Agglomerative clustering

agglomerative = AgglomerativeClustering(n\_clusters=3)

df2['hierarchical\_cluster'] = agglomerative.fit\_predict(df2)

# Plot Hierarchical clusters

plt.scatter(df2['person\_income'], df2['loan\_amnt'], c=df2['hierarchical\_cluster'], cmap='viridis', alpha=0.6, s=50)

plt.xlabel('person\_income')

plt.ylabel('loan\_amnt')

plt.title('Hierarchical Clustering')

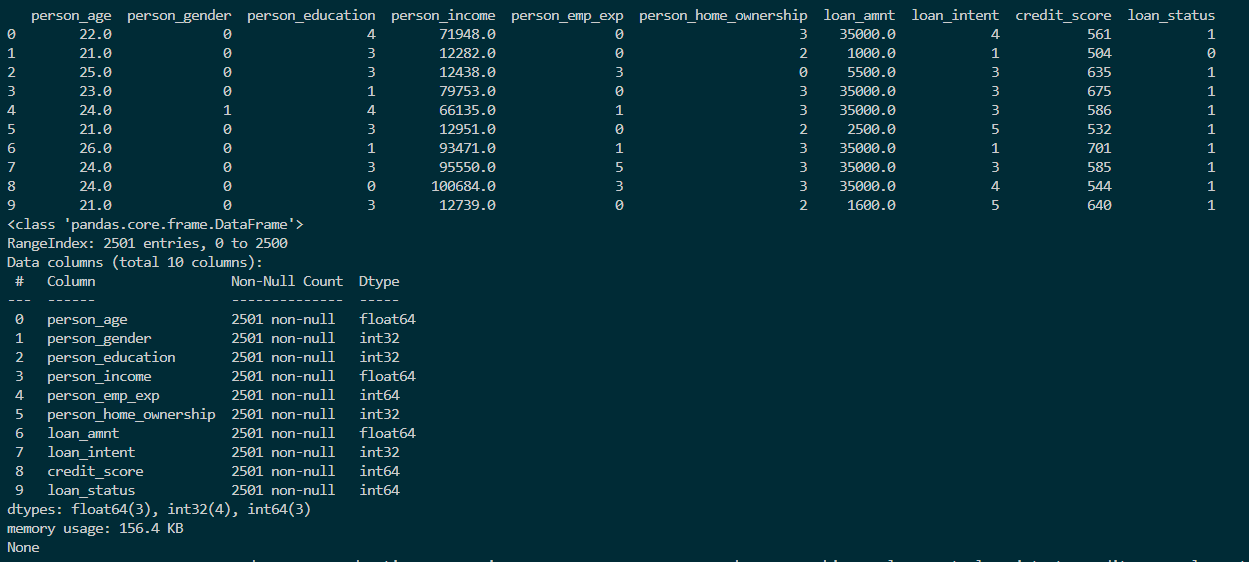
plt.show()

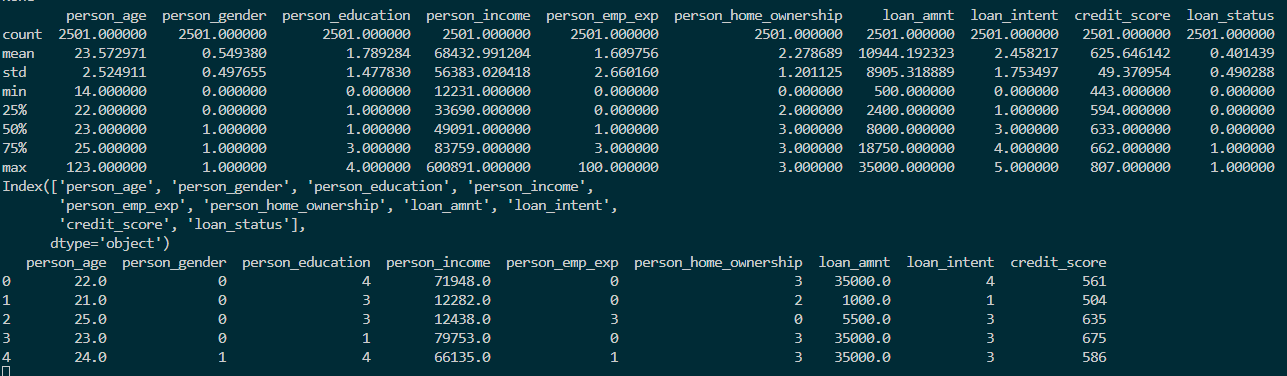
# Silhouette Score for Hierarchical Clustering

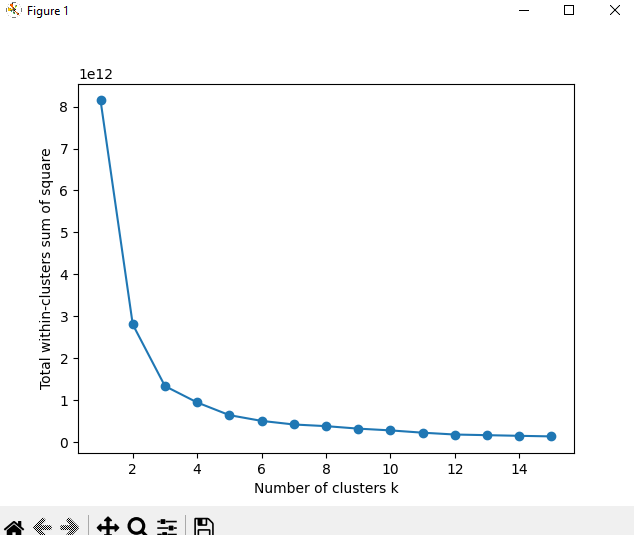
silhouette\_avg\_hierarchical = silhouette\_score(df2, df2['hierarchical\_cluster'])

print("Mean Silhouette Width for Hierarchical Clustering:", silhouette\_avg\_hierarchical)

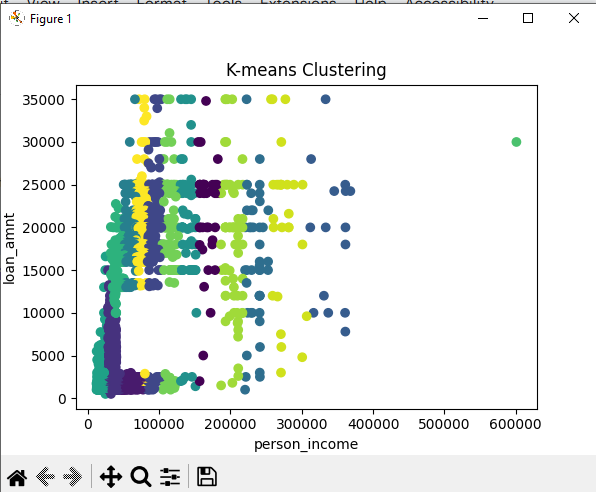
**Output:**



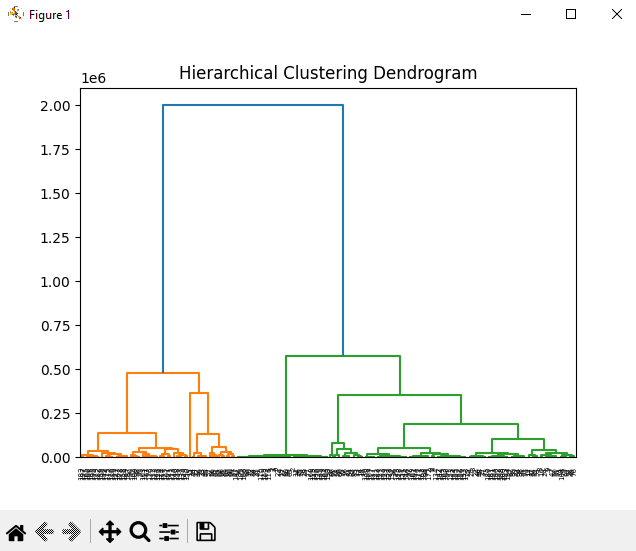


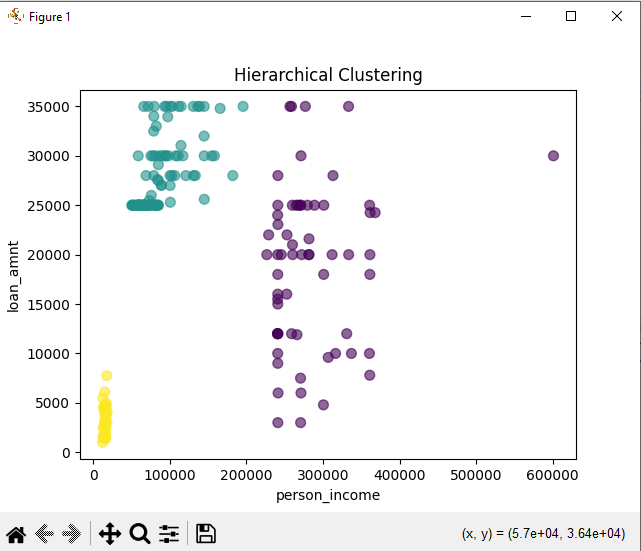


**Inference:** The above graph shows that there is not significant improvement in the inertia after 4 numbers of clusters. Hence, 4 is a good value for the number of clusters to be used.



**Inference:** The graph shows the distribution of different clusters between loan\_amnt and person\_income. There are visible distinguishable observations about the different clusters the data belonged to.





**Inference:** The graph shows the 4 different clusters for the same features (loan\_amnt and person\_income) when using Hierarchical Clustering. The clusters are formed quite properly and gives insights about a attribute have different effects on different clusters.



Practical No. 9

Association

**Q.1]** Do Association for shopping market cart dataset using Python.

**Code:**

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from mlxtend.preprocessing import TransactionEncoder

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

from sklearn.impute import SimpleImputer

with open("groceries.csv", 'r') as temp\_f:

# get No of columns in each line

col\_count = [ len(l.split(",")) for l in temp\_f.readlines() ]

### Generate column names (names will be 0, 1, 2, ..., maximum columns - 1)

column\_names = [i for i in range(max(col\_count))]

# Load the grocery transactions data

data = pd.read\_csv("groceries.csv",names=column\_names)

# Display the first few rows of the data and column names for inspection

print("First few rows of the data:\n", data.head())

print("Columns in the data:\n", data.columns)

# Convert the dataframe into a list of transactions (removing 'nan' values)

transactions = data.values.astype(str).tolist()

transactions = [[item for item in row if item != 'nan'] for row in transactions]

print("First 10 transactions:\n", transactions[:10])

# Initialize the TransactionEncoder to convert the transactions into a one-hot encoded format

transaction\_encoder = TransactionEncoder()

encoded\_transactions = transaction\_encoder.fit(transactions).transform(transactions)

# Convert the encoded transactions into a DataFrame for easier handling

encoded\_df = pd.DataFrame(encoded\_transactions, columns=transaction\_encoder.columns\_)

print("First 5 rows of the one-hot encoded data:\n", encoded\_df.head())

print("Shape of the encoded DataFrame:", encoded\_df.shape)

# Apply the Apriori algorithm to find frequent itemsets with a minimum support of 0.01

frequent\_itemsets = apriori(encoded\_df, min\_support=0.01, use\_colnames=True)

# Display the count of itemsets that meet the minimum support threshold

print("Number of frequent itemsets:\n", frequent\_itemsets.count()['itemsets'])

# Plot the top 15 frequent itemsets based on support

plt.figure(figsize=(12, 6))

plt.xticks(rotation=90)

color\_palette = ['#FFB6C1', '#ADD8E6', '#98FB98', '#FFD700', '#FFA07A',

'#87CEEB', '#FFC0CB', '#FF69B4', '#00FA9A', '#FF6347']

sns.barplot(x='itemsets', y='support', data=frequent\_itemsets.nlargest(n=15, columns='support'), palette=color\_palette)

plt.title("Top 15 Frequent Itemsets Based on Support")

plt.xlabel('Itemsets')

plt.ylabel('Support')

plt.show()

# Generate association rules based on the "lift" metric, with a minimum lift threshold of 1

association\_rules\_lift = association\_rules(frequent\_itemsets, metric="lift", min\_threshold=1,num\_itemsets=4)

# Sort and display the association rules based on support

print("Association rules sorted by support:\n", association\_rules\_lift.sort\_values(by=['support'], ascending=False))

# Add additional columns to the rules DataFrame to analyze the length of the antecedents and consequents

association\_rules\_lift["antecedent\_len"] = association\_rules\_lift["antecedents"].apply(lambda x: len(x))

association\_rules\_lift["consequent\_len"] = association\_rules\_lift["consequents"].apply(lambda x: len(x))

# Display the updated rules with antecedent and consequent lengths

print("Association rules with antecedent and consequent lengths:\n", association\_rules\_lift)

# Filter rules where the antecedent length is greater than or equal to 2

rules\_with\_multiple\_antecedents = association\_rules\_lift[association\_rules\_lift['antecedent\_len'] >= 2]

print("Rules with antecedent length >= 2:\n", rules\_with\_multiple\_antecedents)

# Filter rules where the antecedent length is at least 2, confidence is greater than 0.3, and lift is greater than 1

filtered\_rules\_1 = association\_rules\_lift[

(association\_rules\_lift['antecedent\_len'] >= 2) &

(association\_rules\_lift['confidence'] > 0.3) &

(association\_rules\_lift['lift'] > 1)

].sort\_values(by=['lift', 'support'], ascending=False)

print("Filtered rules with high confidence and lift:\n", filtered\_rules\_1)

# Filter rules where the consequent length is at least 2 and lift is greater than 1

filtered\_rules\_2 = association\_rules\_lift[

(association\_rules\_lift['consequent\_len'] >= 2) &

(association\_rules\_lift['lift'] > 1)

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

print("Filtered rules with high lift and consequent length >= 2:\n", filtered\_rules\_2)

# Recalculate the 'lift' column to ensure it represents the correct formula (support / (antecedent\_len \* consequent\_len))

association\_rules\_lift['lift'] = association\_rules\_lift['support'] / (association\_rules\_lift['antecedent\_len'] \* association\_rules\_lift['consequent\_len'])

print("Recalculated lift values:\n", association\_rules\_lift)

# ------------------- ACCURACY METRICS -------------------

# Generate and print association rules using different metrics for evaluation

print("-------------------- Accuracy Metrics --------------------")

# Association rules using 'lift' metric

rules\_by\_lift = association\_rules(frequent\_itemsets, metric="lift", min\_threshold=1,num\_itemsets=4)

print("Rules based on lift metric:\n", rules\_by\_lift)

# Association rules using 'confidence' metric

rules\_by\_confidence = association\_rules(frequent\_itemsets, metric="confidence", min\_threshold=1,num\_itemsets=4)

print("Rules based on confidence metric:\n", rules\_by\_confidence)

# Association rules using 'leverage' metric

rules\_by\_leverage = association\_rules(frequent\_itemsets, metric="leverage", min\_threshold=1,num\_itemsets=4)

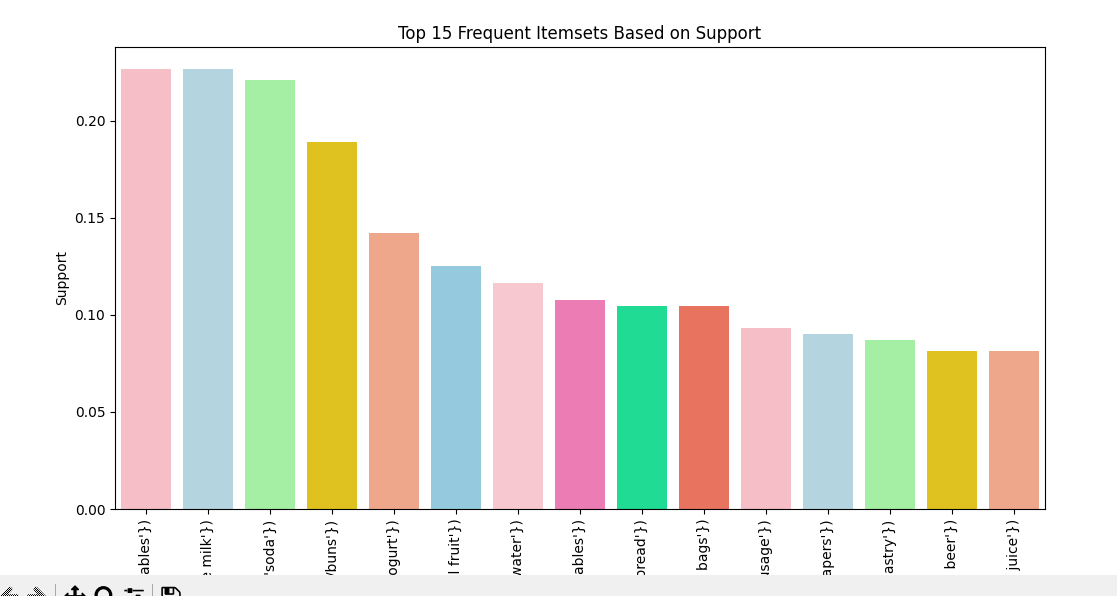
print("Rules based on leverage metric:\n", rules\_by\_leverage)

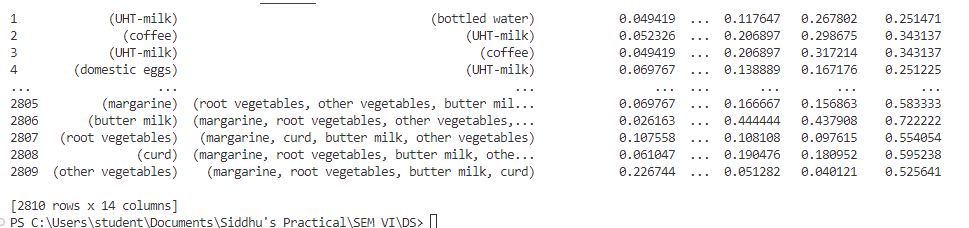
# Association rules using 'conviction' metric

rules\_by\_conviction = association\_rules(frequent\_itemsets, metric="conviction", min\_threshold=1,num\_itemsets=4)

print("Rules based on conviction metric:\n", rules\_by\_conviction)

**Output:**





**Q.2]** Generate FP Tree for shopping market cart dataset using Python.

**Code:**

import pyfpgrowth

##transactions = [[1, 2, 5],

## [2, 4],

## [2, 3],

## [1, 2, 4],

## [1, 3],

## [2, 3],

## [1, 3],

## [1, 2, 3, 5],

## [1, 2, 3]]

import csv

with open('groceries.csv', encoding="utf8", newline='') as f:

reader = csv.reader(f)

data = list(reader)

for i in range(len(data)):

if '' in data[i]:

data[i] = [x for x in data[i] if x]

data.pop(0)

print("some records",data[0:10])

patterns = pyfpgrowth.find\_frequent\_patterns(data, 2)

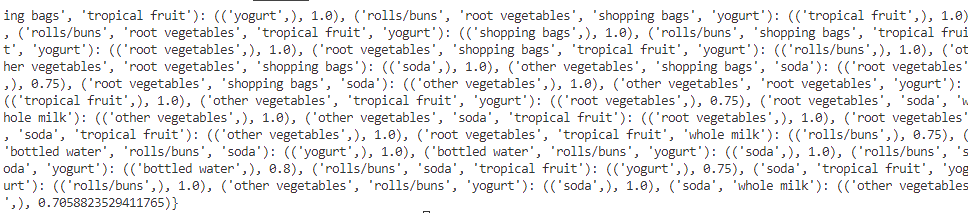
print("pattern",patterns)

print()

rules = pyfpgrowth.generate\_association\_rules(patterns, 0.7)

print("rules output",rules)

**Output:**



**Inference:**



The following lists shows the itemset showing probability that a person will buy another item when the person buys the supporting item. Here we use confidence threshold as 0.7 meaning only itemsets with probability >= than threshold will be included.

Practical No. 10

MongoDB

**Q.1]** Perform the following on some document databases other than specified in the doc attached.

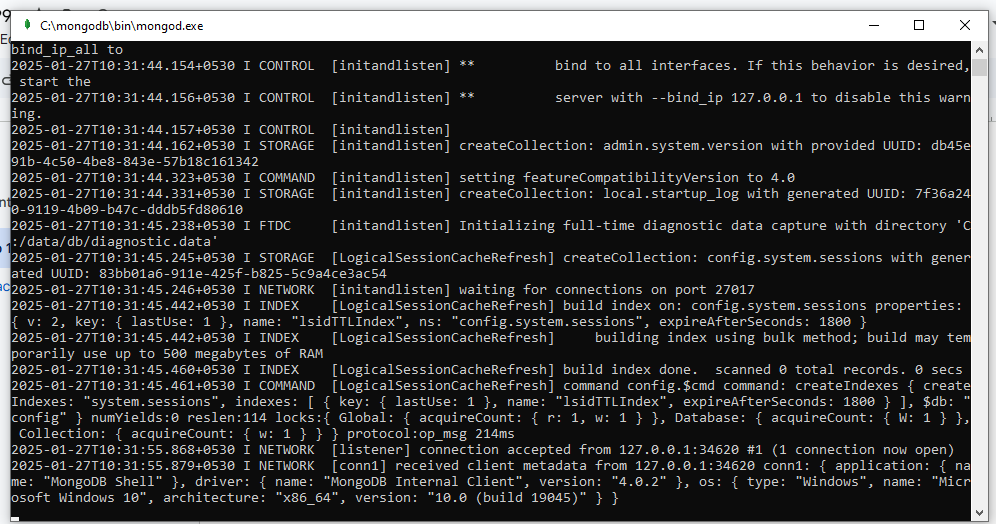
1. Create at least two collections
2. Insert five records in each
3. Querying-filtering,join at least six queries
4. Updating a record
5. Deleting a record
6. Dropping collection and databases

**Steps:**

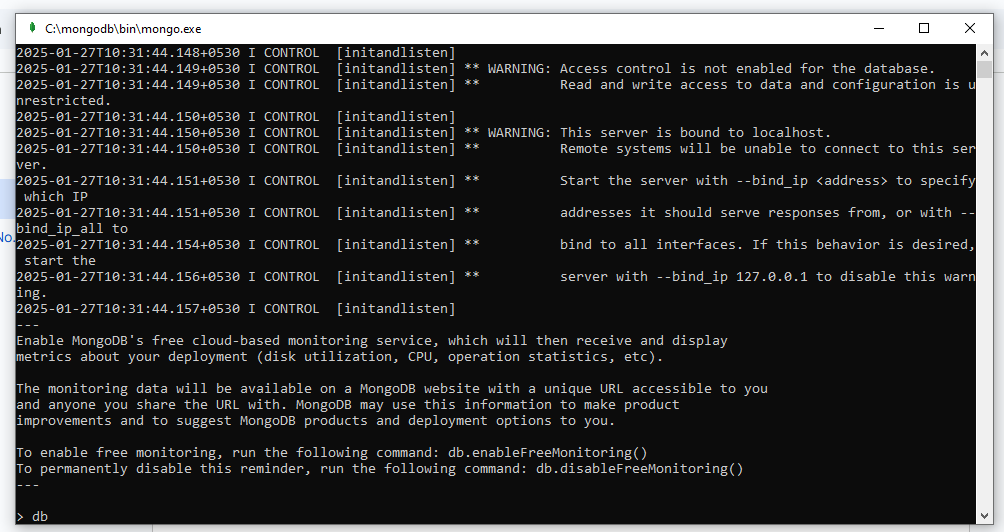
1. Extract mongodb zip C:
2. Create the following directory



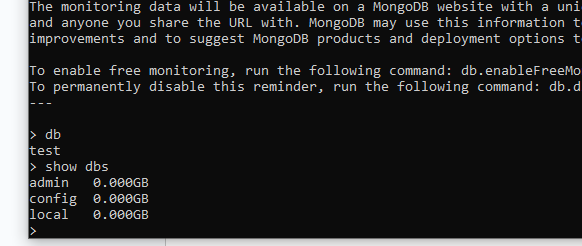
1. Goto C:\mongodb\bin and click on mongod.exe and keep server running



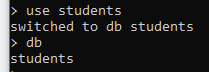
1. Click on mongo.exe



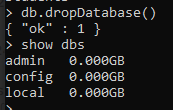
1. Write db to show current db



1. Create DB Command



1. Delete DB Command



1. Create Collection command



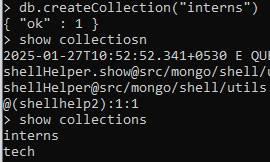
1. Command to check if the record was inserted



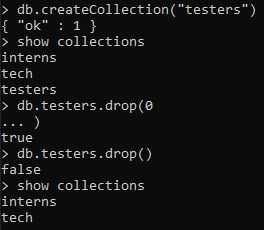
1. Display Collections command



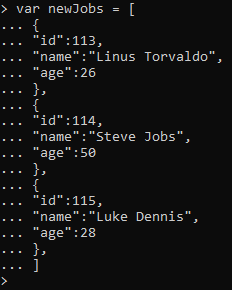
1. Alternate command to create collection



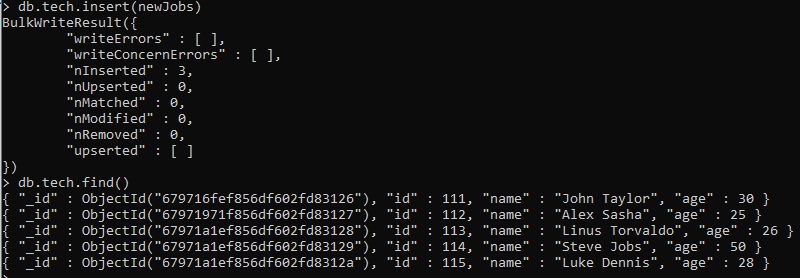
1. Drop Collection Command



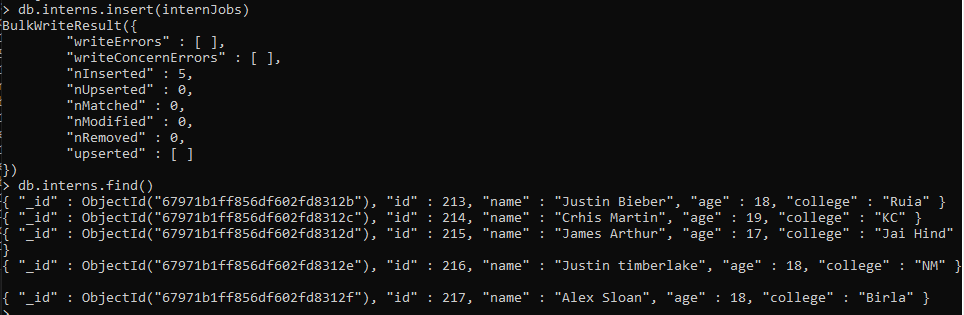
1. Creating a variable to insert multiple records



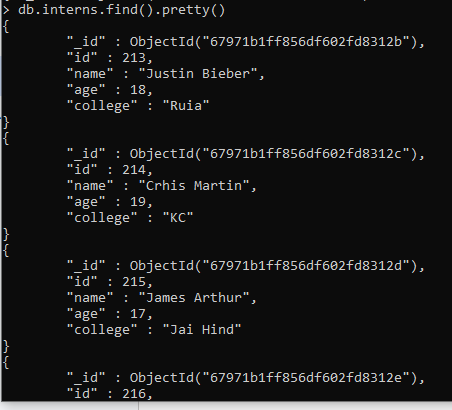
1. Inserting records



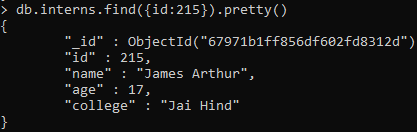
1. 5 Records in “interns” collection



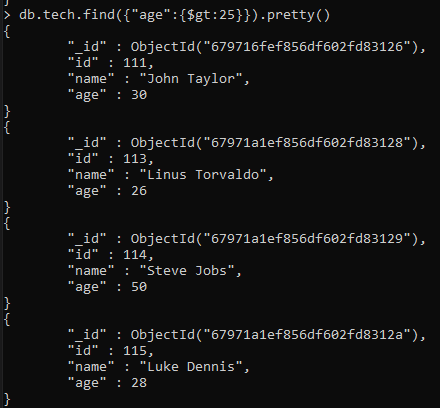
1. Querying in JSON format



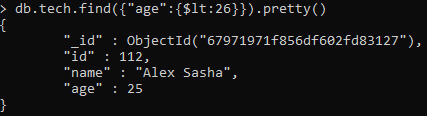
1. Querying based on id



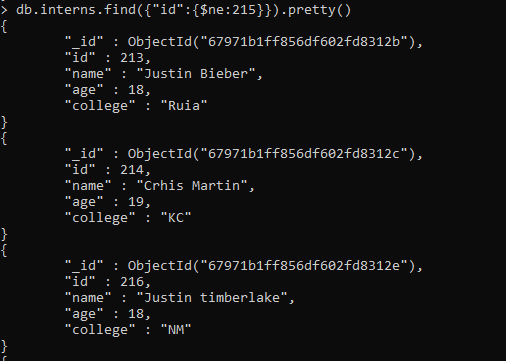
1. Greater than query



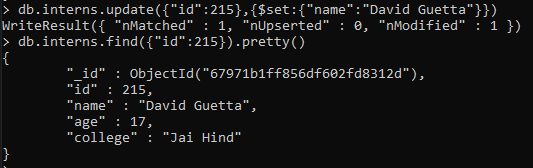
1. Less than query



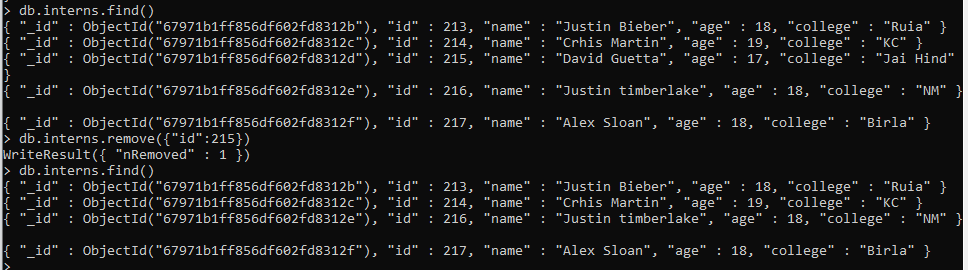
1. Not equal query



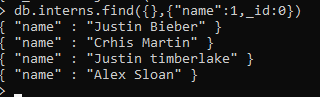
1. Update Query



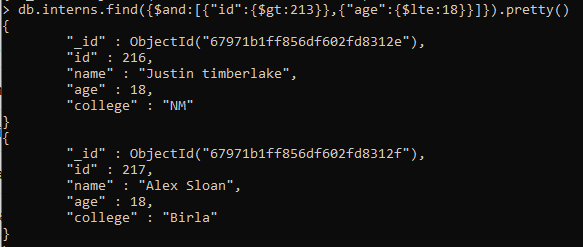
1. Delete Query



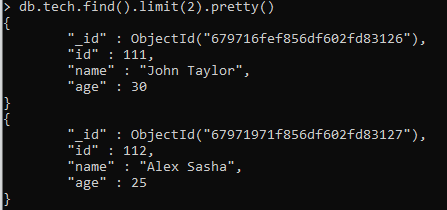
1. Only display a single column. Projection Command



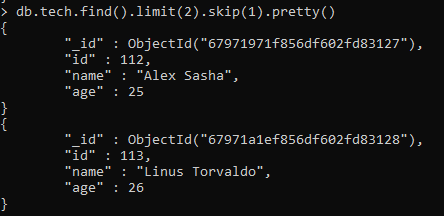
1. AND join query



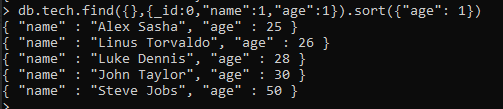
1. Limit Command



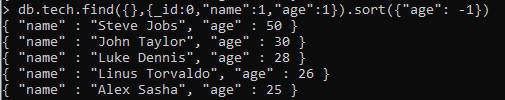
1. Skip Command (See the difference between above output, it will skip 1 record and limit to 2 records)



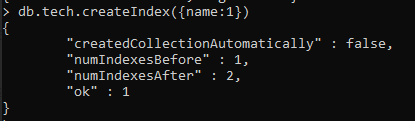
1. Sort in ascending order



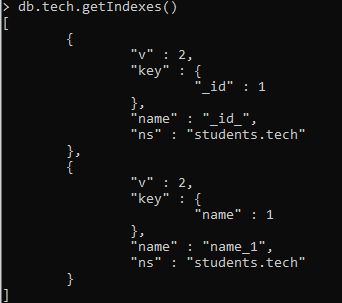
1. Sort in descending order



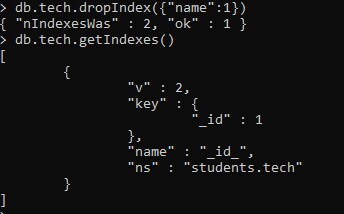
1. Create index command



1. Show indexes in a collection



1. Drop Index



Practical No. 11

Timer Series Forecasting

**Q.1]** Implement Time Series Forecasting in Python using the ARIMA model.

**Code:**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

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

from statsmodels.tsa.statespace.sarimax import SARIMAX

from statsmodels.tsa.arima.model import ARIMA

url = "https://raw.githubusercontent.com/facebook/prophet/main/examples/example\_air\_passengers.csv"

data = pd.read\_csv(url, header=0, parse\_dates=[0], index\_col=0,)

train = data.iloc[:-12]

test = data.iloc[-12:]

arima\_model = ARIMA(train, order=(5,1,0))

arima\_result = arima\_model.fit()

arima\_forecast = arima\_result.forecast(steps=12)

sarimax\_model = SARIMAX(train, order=(1,1,1), seasonal\_order=(1,1,1,12))

sarimax\_result = sarimax\_model.fit()

sarimax\_forecast = sarimax\_result.forecast(steps=12)

plt.figure(figsize=(10, 5))

plt.plot(train, label='Train')

plt.plot(test, label='Test')

plt.plot(arima\_forecast, label='ARIMA Forecast')

plt.plot(sarimax\_forecast, label='SARIMAX Forecast')

plt.legend()

plt.title('ARIMA and SARIMAX Forecasting')

plt.show()

#MAE

arima\_mae = mean\_absolute\_error(test, arima\_forecast)

sarimax\_mae = mean\_absolute\_error(test, sarimax\_forecast)

#RMSE

arima\_mse = mean\_squared\_error(test, arima\_forecast)

arima\_rmse = np.sqrt(arima\_mse)

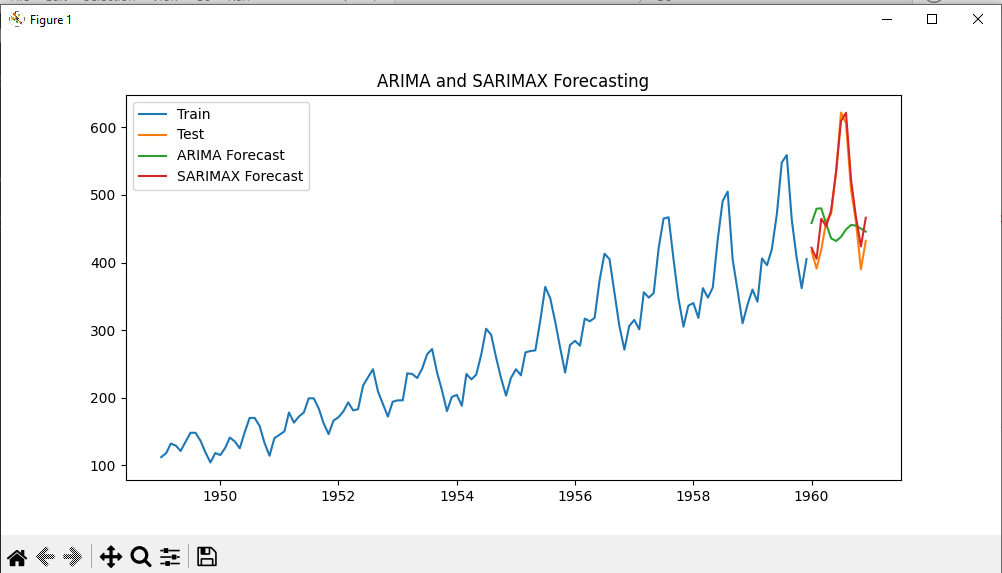
sarimax\_mse = mean\_squared\_error(test, sarimax\_forecast)

sarimax\_rmse = np.sqrt(sarimax\_mse)

print(f"ARIMA MAE: {arima\_mae:.2f}, SARIMAX MAE: {sarimax\_mae:.2f}" )

print(f"ARIMA RMSE: {arima\_rmse:.2f}, SARIMAX RMSE: {sarimax\_rmse:.2f}")

**Output:**





**Inference:** From the observations and the accuracy, we can say that the SARIMAX model is predicting more accurately than the ARIMA model. The dataset has more seasonal effect, so the SARIMAX model is better suited for this dataset.

// 1. Select the database

use student

// 2. Drop the current database

db.dropDatabase()

// 3. Insert a document into the 'tech' collection

db.tech.insert({id: 111, name: "soham"})

// 4. Retrieve all documents from the 'tech' collection

db.tech.find()

// 5. Show all collections in the database

show collections

// 6. Create a new collection named 'interns'

db.createCollection("interns")

// 7. Insert multiple documents into the 'tech' collection

var newjob = [{id: 111, name: "soham"}, {id: 112, name: "yash"}]

db.tech.insert(newjob)

// 8. Retrieve all documents from the 'interns' collection in a formatted way

db.interns.find().pretty()

// 9. Find a document in the 'interns' collection where id is 215

db.interns.find({id: 215}).pretty()

// 10. Find all documents in the 'tech' collection where age is greater than 25

db.tech.find({"age": {$gt: 25}}).pretty()

// 11. Find all documents in the 'tech' collection where age is less than 25

db.tech.find({"age": {$lt: 25}}).pretty()

// 12. Find all documents in the 'tech' collection where id is not equal to 215

db.tech.find({"id": {$ne: 215}}).pretty()

// 13. Update a document in the 'interns' collection where id is 215

db.interns.update({"id": 215}, {$set: {"name": "David Guetta"}})

// 14. Remove a document in the 'interns' collection where id is 215

db.interns.remove({"id": 215})

// 15. Retrieve only the 'name' field from all documents in the 'interns' collection (excluding \_id)

db.interns.find({}, {"name": 1, \_id: 0})

// 16. Find documents in the 'interns' collection where id > 213 and age <= 18

db.interns.find({$and: [{"id": {$gt: 213}}, {"age": {$lte: 18}}]}).pretty()

// 17. Limit the number of results to 2 in the 'tech' collection

db.tech.find().limit(2).pretty()

// 18. Skip the first document and limit results to 2 in the 'tech' collection

db.tech.find().limit(2).skip(1).pretty()

// 19. Retrieve only the 'name' and 'age' fields from the 'tech' collection and sort by 'age' in ascending order

db.tech.find({}, {\_id: 0, "name": 1, "age": 1}).sort({"age": 1})

// 20. Retrieve only the 'name' and 'age' fields from the 'tech' collection and sort by 'age' in descending order

db.tech.find({}, {\_id: 0, "name": 1, "age": 1}).sort({"age": -1})

// 21. Create an index on the 'name' field in the 'tech' collection

db.tech.createIndex({name: 1})

// 22. Retrieve all indexes in the 'tech' collection

db.tech.getIndexes()

// 23. Drop the index on the 'name' field in the 'tech' collection

db.tech.dropIndex({"name": 1})