**SQL Structure**import sqlite3

conn=sqlite3.connect("students.db")

curr=conn.cursor()

sql="query"

curr.execute(sql)

conn.commit()

## **Sample Sql Statements**

conn = sqlite3.connect("students.db")

cursor = conn.cursor()

cursor.execute('''

CREATE TABLE IF NOT EXISTS students (

Name TEXT,

RollNumber INTEGER PRIMARY KEY,

SAPID INTEGER UNIQUE,

Program TEXT,

Semester INTEGER,

Marks INTEGER,

Result TEXT

)

''')

conn.commit()

**//Adding elements**

students\_data = [

("Alice", 101, 5001, "BSc CS", 3, 85, "Pass"),

("Bob", 102, 5002, "BSc IT", 3, 40, "Fail"),

]

try:

cursor.executemany("INSERT INTO students VALUES (?, ?, ?, ?, ?, ?, ?)", students\_data)

conn.commit()

except sqlite3.Error as e:

print(f"Error inserting data: {e}")

**//Where clause**

try:

cursor.execute("SELECT \* FROM students WHERE Name=?", (name,))

conn.commit()

result = cursor.fetchall()

print(result)

except sqlite3.Error as e:

print(f"Error searching by name: {e}")

**//Updating**

try:

cursor.execute("UPDATE students SET Marks=?, Result=? WHERE Name=? AND SAPID=?", (35,"Fail", "Alice", 5001))

conn.commit()

except sqlite3.Error as e:

print(f"Error updating data: {e}")

**//Deleting**

try:

cursor.execute("DELETE FROM students WHERE SAPID=?", (5001,))

conn.commit()

except sqlite3.Error as e:

print(f"Error deleting data: {e}")

## **Importing Required Libraries**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

from sklearn.preprocessing import MinMaxScaler

## **Loading Data**

df = pd.read\_csv("dataset.csv")

df.head()

df.tail()

## **Data Inspection**

df.shape

df.info()

df.describe()

df.columns

df.isnull().sum()

df.duplicated().sum()

## **Handling Missing Values**

df.dropna(inplace=True)

df.fillna(df.mean(), inplace=True) **For Numerical**

df.fillna(df.mode().iloc[0], inplace=True) **For Categorical**

## **Handling Duplicates**

df.drop\_duplicates(inplace=True)

## **Outliers Quartile**

cat\_cols = df.select\_dtypes(include=["object"]).columns.tolist()

num\_cols = df.select\_dtypes(include=[np.number]).columns.tolist()

Q1 = df[num\_cols].quantile(0.25)

Q3 = df[num\_cols].quantile(0.75)

IQR = Q3 - Q1

lower\_bound = Q1 - 1.5 \* IQR

upper\_bound = Q3 + 1.5 \* IQR

outliers = ((df[num\_cols] < lower\_bound) | (df[num\_cols] > upper\_bound)).sum()

print("\nOutliers detected:\n", outliers)

# Removing Outliers – large datasets

df\_cleaned = df[~((df[num\_cols] < lower\_bound) | (df[num\_cols] > upper\_bound )).any(axis=1)]

print("\nShape after outlier removal:", df\_cleaned.shape)

# Capping Outliers – small dataset

df\_capped = df.copy() # Make a copy to avoid modifying the original data

for col in num\_cols:

df\_capped[col] = np.where(df\_capped[col] < lower\_bound[col], lower\_bound[col], df\_capped[col])

df\_capped[col] = np.where(df\_capped[col] > upper\_bound[col], upper\_bound[col], df\_capped[col])

print("\nData after capping extreme values:")

print(df\_capped.describe())

## **Normalization**

scaler = MinMaxScaler()

df\_cleaned[num\_cols] = scaler.fit\_transform(df\_cleaned[num\_cols])

print("\nData after Min-Max Scaling:\n", df\_cleaned.head())

## **Creating New Features**

df["New\_Column"] = df["Original\_Column"].apply(lambda x: x.split()[0])

## **Converting Data Types**

df["Column"] = pd.to\_numeric(df["Column"], errors='coerce')

df["Date\_Column"] = pd.to\_datetime(df["Date\_Column"])

# Extract year

df["Year"] = df["Date\_Column"].dt.year # Returns integers like 2023

# Extract month (as integer)

df["Month"] = df["Date\_Column"].dt.month # Returns integers 1-12

# Extract month name

df["Month\_Name"] = df["Date\_Column"].dt.month\_name() # Returns "January", "February", etc.

# Extract week of the year

df["Week"] = df["Date\_Column"].dt.isocalendar().week # Returns integers 1-53

# Extract day of month

df["Day"] = df["Date\_Column"].dt.day # Returns integers 1-31

# Extract day name

df["Weekday\_Name"] = df["Date\_Column"].dt.day\_name() # Returns "Monday", "Tuesday", etc.

# Extract day of week (0=Monday, 6=Sunday by default)

df["Weekday"] = df["Date\_Column"].dt.dayofweek # Returns integers 0-6

## **Extracting Numerical Values from Strings**

df["Column"] = df["Column"].str.extract(r'(\d+)').astype(float)

## **Feature Engineering: Calculating Age**

from datetime import date

today = date.today()

df['Age'] = today.year - df['Year\_Column']

## **Data Visualization**

### **Histogram & Boxplot for Numerical Data**

for col in num\_cols:

plt.figure(figsize=(12, 5))

plt.subplot(1, 3, 1)

df[col].hist(grid=False, color='blue', alpha=0.7)

plt.xlabel(col)

plt.ylabel("Count")

plt.title("Histogram")

plt.subplot(1, 3, 2)

sns.boxplot(x=df[col], color='red')

plt.title("Boxplot")

plt.subplot(1, 3, 3)

sns.kdeplot(df[col], shade=True, color='green')

plt.title("Density Plot")

plt.show()

### **Bar Plots and Pie Chart for Categorical Data**

fig, axes = plt.subplots(3, 2, figsize=(10, 10))

fig.suptitle("Bar Plot for Categorical Variables")

sns.countplot(ax=axes[0, 0], x='Column1', data=df, color='blue', order=df['Column1'].value\_counts().index)

sns.countplot(ax=axes[0, 1], x='Column2', data=df, color='blue', order=df['Column2'].value\_counts().index)

sns.countplot(ax=axes[1, 0], x='Column3', data=df, color='blue', order=df['Column3'].value\_counts().index)

sns.countplot(ax=axes[1, 1], x='Column4', data=df, color='blue', order=df['Column4'].value\_counts().index)

sns.countplot(ax=axes[2, 0], x='Column5', data=df, color='blue', order=df['Column5'].value\_counts().index)

sns.countplot(ax=axes[2, 1], x='Column6', data=df, color='blue', order=df['Column6'].value\_counts().index)

axes[1, 1].tick\_params(labelrotation=45)

axes[2, 0].tick\_params(labelrotation=90)

axes[2, 1].tick\_params(labelrotation=90)

#pie

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

df\_cleaned['Region'].value\_counts().plot.pie(autopct="%1.1f%%", colors=sns.color\_palette('pastel'))

plt.title("Region Distribution")

plt.ylabel('')

plt.show()

## **Crosstab Groupby** pd.crosstab([df.col1,df.col2],df.col3,margins=True) df.groupby(["col1","col2"])["col1"].count()

## **Log Transformation for Skewed Data**

def log\_transform(data, cols):

for colname in cols:

data[colname + '\_log'] = np.log1p(data[colname])

print(data.info())

log\_transform(df, ['Numerical\_Column1', 'Numerical\_Column2'])

### **Pair Plots & Correlation Heatmaps**

sns.pairplot(data=df[num\_cols])

plt.show()

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

sns.heatmap(df[num\_cols].corr(), annot=True)

plt.show()

### **Encoding Categorical Variables**

cc = cat\_cols.tolist()

df\_encoded = pd.get\_dummies(df[cc], columns=cc, drop\_first=True)

### **Final Heatmap of Encoded Data**

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

sns.heatmap(df\_encoded.corr(), annot=True)

plt.show()

### **Uni, bi, multi variate analysis**

df['Happiness Score'].hist() #uni

sns.scatterplot(x=df['Economy (GDP per Capita)'], y=df['Happiness Score']) #bi

sns.pairplot(df[['Happiness Score', 'Economy (GDP per Capita)', 'Family', 'Health (Life Expectancy)']]) #multi