**A**

**Journal on**

**Data Science**

**Soft Computing**

**DESIGNED BY**

**Mr. VINAY SUBHASH TIVAREKAR**

**SEAT NO.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Submitted To**

**K.M.S.P MANDAL’S**

**SANT RAWOOL MAHARAJ MAHAVIDYALAYA, KUDAL**

**(NAAC Accreditation “B+” Grade)**

**In the partial fulfilment of**

**M.Sc. Information Technology**

**(Part 1)**

**Under Guidance of**

**Asst. Prof. K. A. Kubal**

**Asst. Prof. P. S. Keravadekar**

**Through**

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**A**

**Journal on**

**Data Science**

**Soft Computing**

**DESIGNED BY**

**Mr. VINAY SUBHASH TIVAREKAR**

**SEAT NO.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Submitted To**

**K.M.S.P MANDAL’S**

**SANT RAWOOL MAHARAJ MAHAVIDYALAYA, KUDAL**

**(NAAC Accreditation “B+” Grade)**

**In the partial fulfilment of**

**M.Sc. Information Technology**

**(First Year)**

**Under Guidance of**

**Asst. Prof. K. A. Kubal**

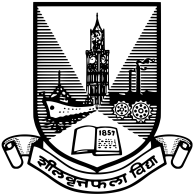
**Asst. Prof. P. S. Keravadekar**

**Through**

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**DATA SCIENCE**

**UNIVERSITY OF MUMBAI**

****

**K.M.S.P. Mandal’s**

**Sant Rawool Maharaj Mahavidyalaya**

**Kudal, Dist-Sindhudurg**

**DEPARTMEMT OF INFORMATION TECHNOLOGY**

# CERTIFICATE

**This to certify that,**

**Mr/Miss. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Exam Seat No. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ student of the M.Sc. (Information Technology, Part-I). He/ She has been successfully completed practical as prescribed by University of Mumbai in the Sant Rawool Maharaj Mahavidyalaya during the semester \_\_\_\_ of the academic year 2025-26**

**In the following topics**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Teacher In -Charge: External Examiner:**

**Date:**

**INDEX**

|  |  |  |
| --- | --- | --- |
| **Sr no.** | **Title** | **Sign** |
| **1.** | **Creating and using database in Cassandra** |  |
| **2.** | **Conversion from different formats to HOURS format.**   1. **Text** 2. **XML** 3. **JSON** 4. **Picture (JPEG)** 5. **Video** 6. **Audio** 7. **MySQL Database** |  |
| **3.** | **Write the program for following:**   1. **Fixer Utilities** 2. **Averaging** 3. **Outlier Detection** 4. **Logging** |  |
| **4.** | **Implement the Following**   * 1. **Data processing using R**   2. **Program retrieve different attributes of data**   3. **LOAD IP\_DATA\_ALL** |  |
| **5.** | **a. Perform error management on the given data using pandas package**  **b. Write python/R program to create the network routing diagram from the given data on routers**  **c. Write a python/R program to build acyclic graph.** |  |
| **6.** | **Build the time hub, links and satellites.** |  |
| **7.** | **Transforming data.** |  |
| **8.** | **Organizing Data** |  |
| **9.** | **Generating Data** |  |
| **10.** | **Data Visualization with power BI** |  |

|  |  |  |
| --- | --- | --- |
| K. M. S. P. Mandal’s  Sant Rawool Maharaj Mahavidyalaya, Kudal    **Department of Masters of Science**  **Information Technology** | Date:    Roll no: 17    Expt No: 01 | Signature |
| **Title: Creating and using database in Cassandra** | | |

Creating and Using a Database in Cassandra with Docker Desktop

Apache Cassandra is a distributed NoSQL database that can be run easily using Docker. Below is a step-by-step overview of setting up Cassandra and performing basic operations.

**1. Docker Desktop Setup**

Before using Cassandra, ensure Docker Desktop is installed and running on your system.

* Check Docker Version:  
  Open a terminal or command prompt and run:
* docker --version

This verifies that Docker is correctly installed and shows the version.

**2. Pull the Cassandra Docker Image**

To use Cassandra, pull its official Docker image from Docker Hub:

docker pull cassandra

This downloads the Cassandra image to your local machine.

3. Running Cassandra in Docker

Start a Cassandra container with a name (cassandra-db in this example):

docker run --name cassandra-db -d cassandra

* --name cassandra-db gives the container a name.
* -d runs it in detached mode.

**4. Check if Cassandra is Running**

Verify that the Cassandra container is active:

docker ps

This lists all running containers. Look for cassandra-db in the list.

**5. Accessing Cassandra Shell (CQLSH**)

CQLSH is the Cassandra Query Language shell. To access it inside the Docker container:

docker exec -it cassandra-db cqlsh

This opens a prompt where you can run Cassandra commands.

**B) Using Basic Cassandra Commands**

**1. Show Keyspaces**

Keyspaces in Cassandra are like databases in SQL. To list all keyspaces:

DESCRIBE KEYSPACES;

**2. Creating a Keyspace**

To create a new keyspace:

CREATE KEYSPACE mykeyspace

WITH replication = {'class':'SimpleStrategy', 'replication\_factor':1};

* SimpleStrategy is used for single-node setups.
* replication\_factor defines how many copies of data Cassandra keeps.

**3. Using a Keyspace**

To use the created keyspace:

USE mykeyspace;

**4. Creating a Table**

Tables store the actual data in Cassandra. Example:

CREATE TABLE users (

id int PRIMARY KEY,

name text,

age int

);

**5. Inserting Data into the Table**

Add records to the table:

INSERT INTO users (id, name, age) VALUES (1, 'Alice', 25);

INSERT INTO users (id, name, age) VALUES (2, 'Bob', 30);

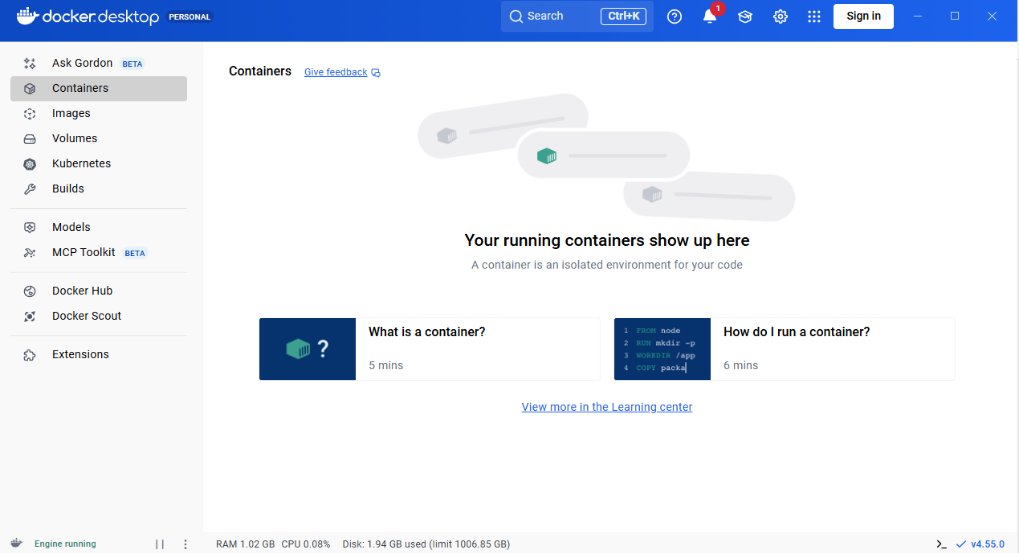
**6. Fetching the Data**

Retrieve data from the table:

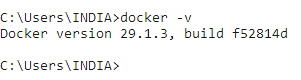
SELECT \* FROM users;

This displays all rows in the users table.

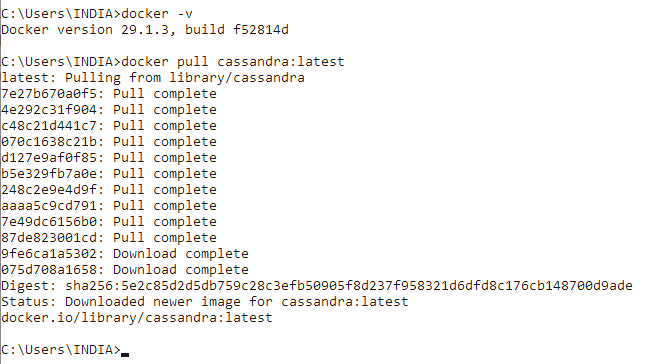
**Docker Desktop:**

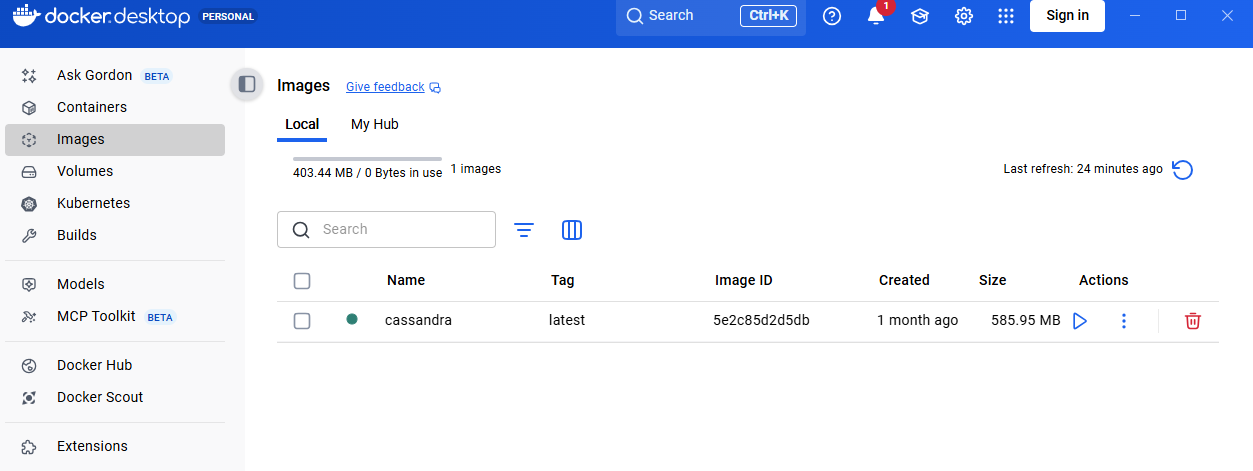


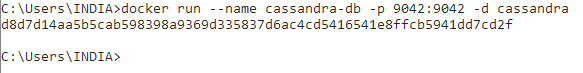
**Checking installed Docker Version:**

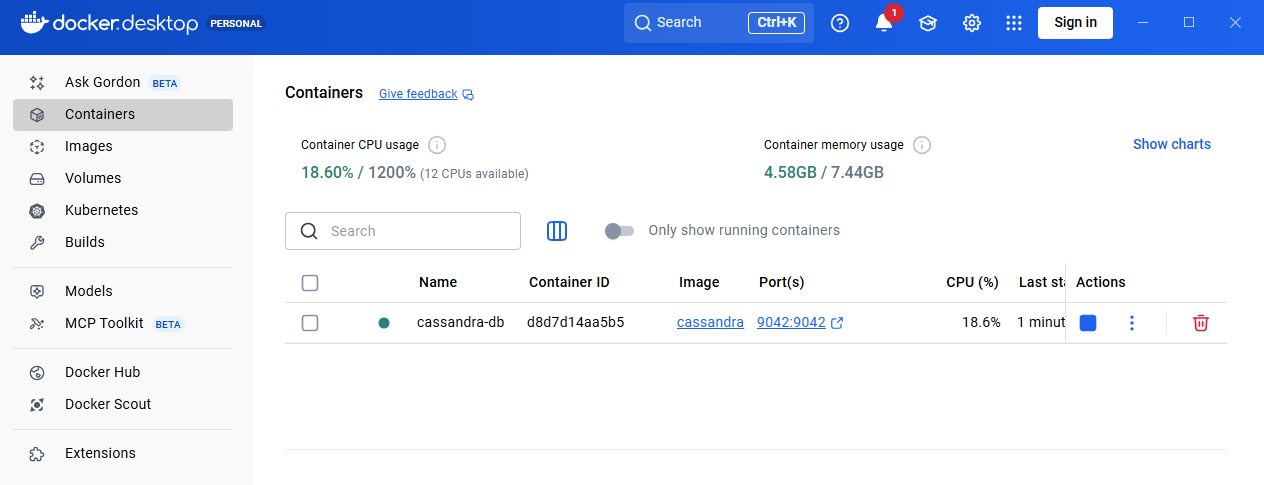
****

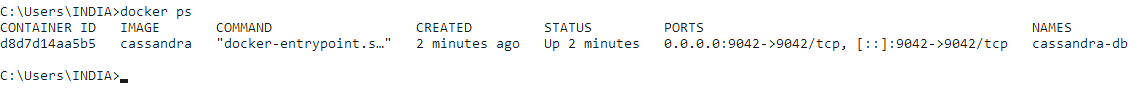
**Pull the Cassandra Docker Image:**

****

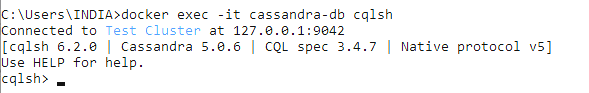




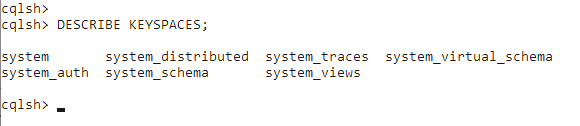
**Running Cassandra in a Docker:**

**Check if Cassandra in Running:**

**Accessing the Cassandra database shell (CQLSH) running inside a Docker container:**

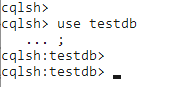
 **Command used: docker exec -it cassandra-db cqlsh**

**B) Using basic Cassandra Commands:**

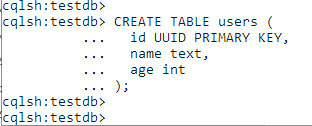
**Show Keyspace:**

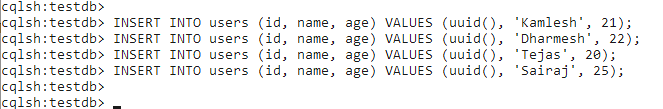
**Creating a Keyspace:**

**Using the Keyspace:**



**Creating a Table in Cassandra:**



**Inserting data into the Table:**

**Fetching the Data:**

****

|  |  |  |
| --- | --- | --- |
| K. M. S. P. Mandal’s  Sant Rawool Maharaj Mahavidyalaya, Kudal    **Department of Masters of Science**  **Information Technology** | Date:    Roll no: 17    Expt No: 02 | Signature |
| **Title: Write The programs for the following** | | |

**Data is available in different formats such as text files, XML, JSON, databases, images, audio, and video. To ensure uniform processing and storage, these heterogeneous data formats are converted into a common representation known as HORUS format. HORUS provides a standardized structure that preserves both data and metadata.**

**CSV to HORUS:**  
Delimited text files are parsed row-wise and column-wise, and the extracted tabular data is mapped into HORUS fields.

**XML to HORUS:**  
Hierarchical XML elements and attributes are parsed and converted into structured HORUS entities while preserving relationships.

**JSON to HORUS:**  
Key–value pairs and arrays in JSON are transformed into HORUS objects with proper data type mapping.

**MySQL to HORUS:**  
Database tables and records are retrieved using SQL queries and converted into structured HORUS format.

**Image (JPEG) to HORUS:**  
Image metadata and pixel information are extracted and stored in HORUS representation.

**Video to HORUS:**  
Video metadata, frames, and audio streams are processed and encapsulated in HORUS format.

**Audio to HORUS:**  
Audio properties and signal data are extracted and represented in a unified HORUS structure.

**Write The programs for the following**

1. **Convert CSV To Horus**

**Code:**

print("Text delimited CSV to HORUS format.")

import pandas as pd

inputfile="countries.csv"

myfile= pd.read\_csv(inputfile,encoding="latin-1")

print(myfile)

Pdata=myfile

# Remove columns

Pdata.drop('alpha3',axis=1,inplace=True)

#Rename column name

Pdata.rename(columns={'name':'CountryName'},inplace=True)

# Sort data by CountryName

Pdata.sort\_values('CountryName',axis=0,ascending=True,inplace=True)

print("Sorted Data Is")

print(Pdata)

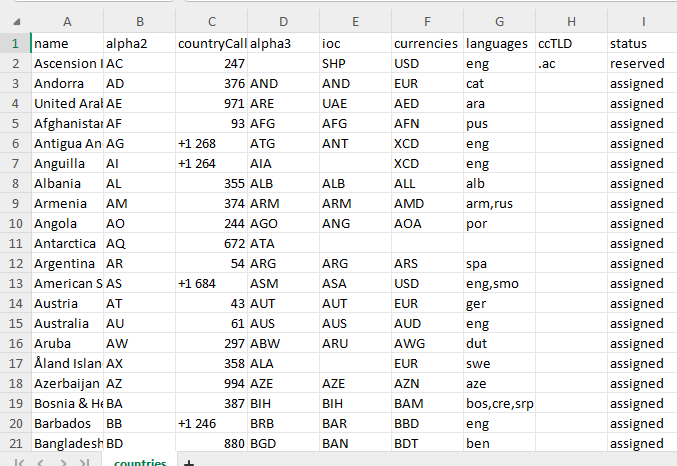
Output=Pdata

OutputFile ="CSV\_horus.csv"

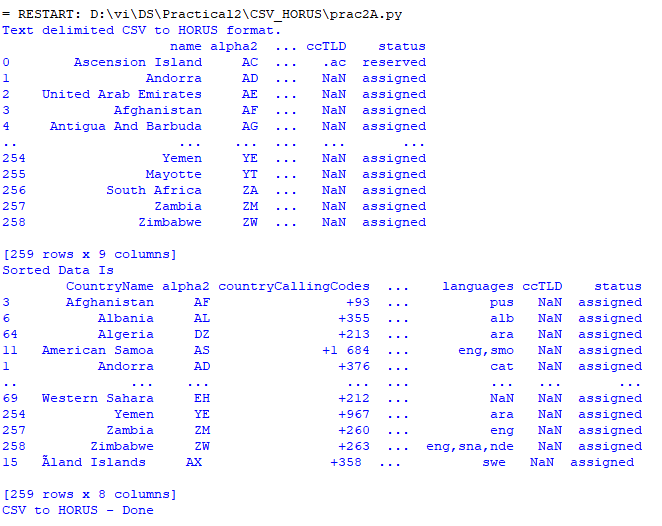
Output.to\_csv(OutputFile,index = False)

print('CSV to HORUS - Done')

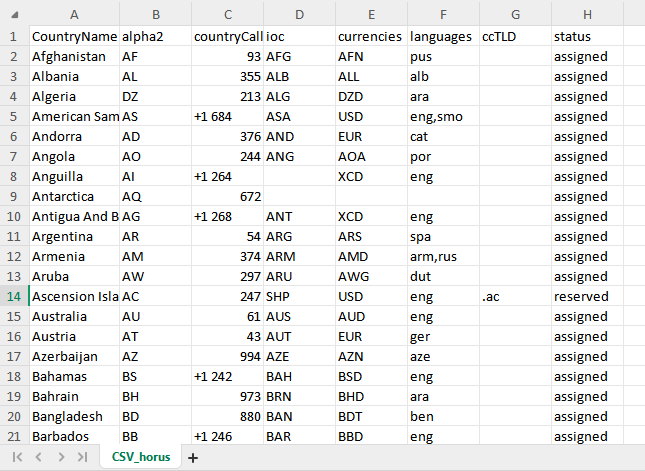
**Original File**

****

**Output**

****

**Converted File**

****

1. **Convert JSON to Horus**

print("JSON to HORUS format.")

import pandas as pd

inputfile="countries.json"

myfile= pd.read\_json(inputfile,encoding="latin-1")

print(myfile)

Pdata=myfile

# Remove columns

Pdata.drop('alpha3',axis=1,inplace=True)

print(Pdata)

#Rename column name

Pdata.rename(columns={'name':'CountryName'},inplace=True)

print(Pdata)

# Set new Index

Pdata.set\_index('CountryName',inplace=True)

print(Pdata)

# Sort data by CountryName

Pdata.sort\_values('CountryName',axis=0,ascending=True,inplace=True)

print("Sorted Data Is")

print(Pdata)

Output=Pdata

OutputFile ="Json\_horus.csv"

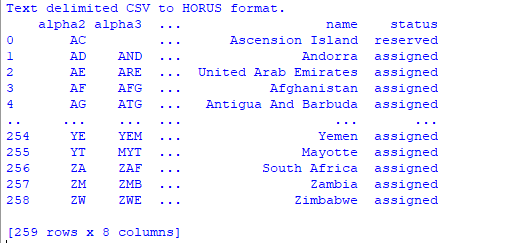
Output.to\_csv(OutputFile, index = False)

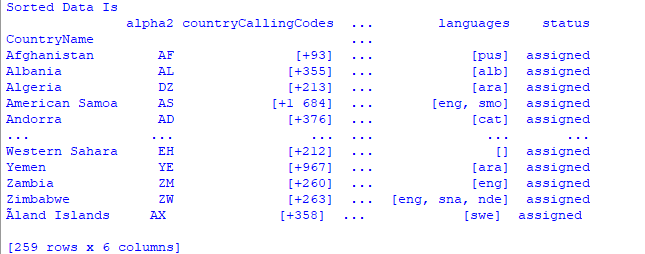
print('CSV to HORUS - Done')

**Original file**

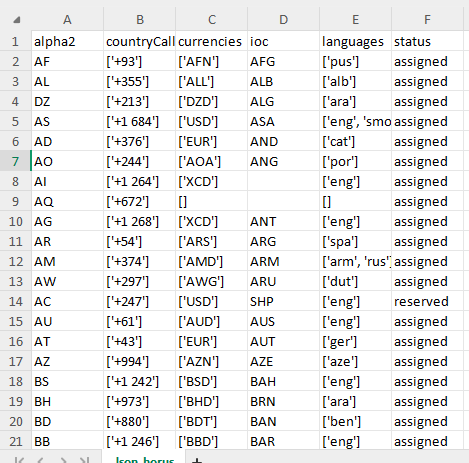
****

**Output**

****

****

**Converted File**

****

1. **Convert XML to Horus**

#install pandas lxml

import pandas as pd

#Read XML

df = pd.read\_xml("data.xml")

#Rename column to uppercase

df.columns = [col.upper() for col in df.columns]

print(df)

#Saving

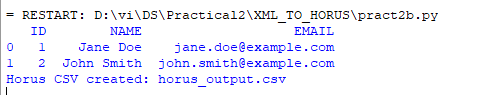
df.to\_csv("horus\_output.csv", index=False)

print("Horus CSV created: horus\_output.csv")

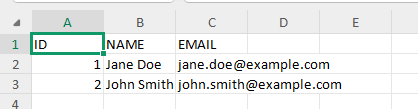
**Original File**



**Output**



**File Creation:**

****

1. **Convert Image(JPG) to Horus**

import cv2 as cv

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

InputFile = 'cat.jpg'

InputData = cv.imread(InputFile, cv.IMREAD\_COLOR)

print('Input Data Values ===================================')

height, width, channels = InputData.shape

print('X: ', height)

print('Y: ', width)

print('RGBA: ', channels)

print('=====================================================')

# Convert image to DataFrame

X = np.meshgrid(np.arange(width),

Y=np.arange(height))

R = InputData[:, :, 2].flatten() # OpenCV uses BGR

G = InputData[:, :, 1].flatten()

B = InputData[:, :, 0].flatten()

df = pd.DataFrame({

'XAxis': X.flatten(),

'YAxis': Y.flatten(),

'Red': R,

'Green': G,

'Blue': B

})

df.index.name = 'ID'

print('Rows: ', df.shape[0])

print('Columns :', df.shape[1])

print('=====================================================')

print('Process Data Values =================================')

print('=====================================================')

# Show the image

plt.imshow(cv.cvtColor(InputData, cv.COLOR\_BGR2RGB))

plt.axis('off')

plt.show()

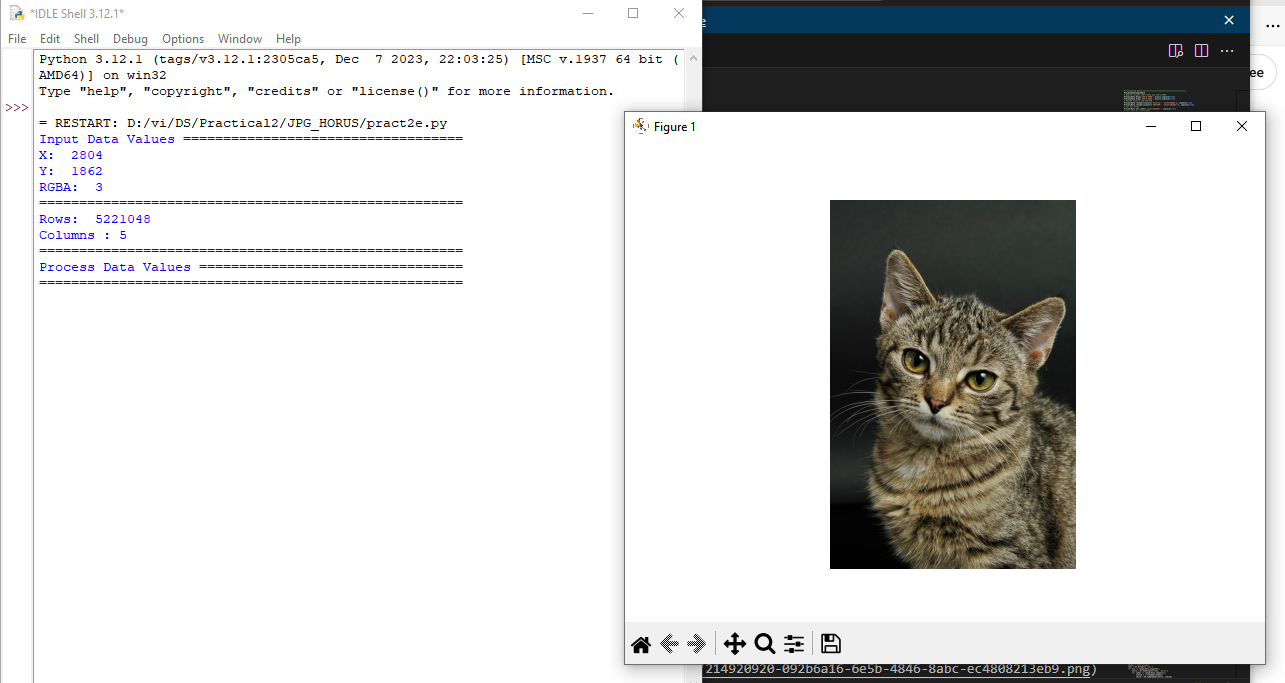
# Save to CSV

print('Storing File')

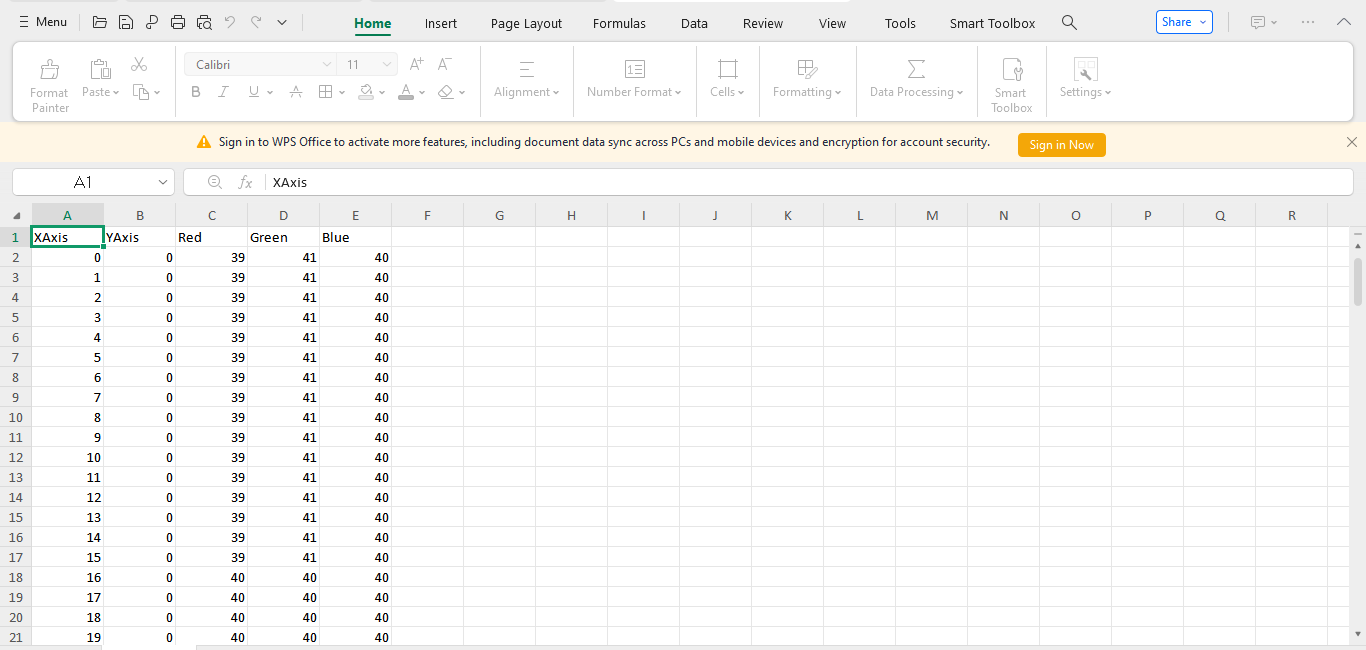
df.to\_csv('Image\_horus.csv', index=False)

print('Picture to HORUS - Done')

**Output**

****

**File conversion**



1. **Convert MYSQL Database to HORUS.**

**Code**

import pandas as pd

import mysql.connector

conn = mysql.connector.connect(

host='localhost',

user='root',

password='',

database='student'

)

query = "SELECT \* FROM stud"

df = pd.read\_sql(query, conn)

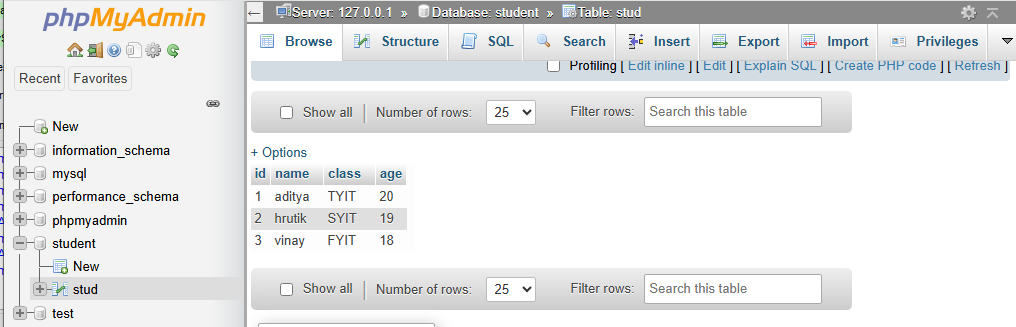
df.rename(columns={ 'fullname': 'name'}, inplace=True)

df.to\_csv("horus\_format\_output.csv", index=False)

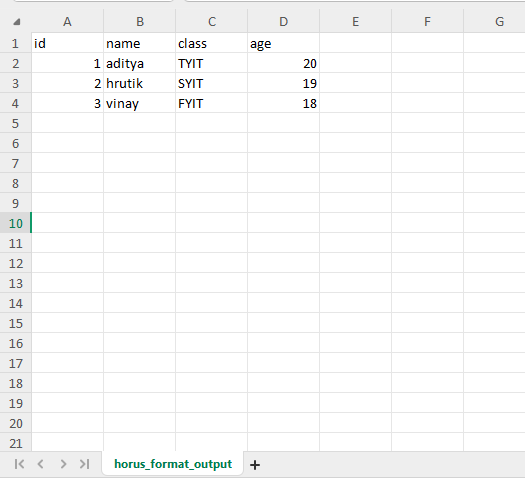
print("Sql\_horus file saved")

conn.close()

**Original MySQL Preview**

****

**Converted Horus File**

****

1. **Convert Audio to horus**

**Code:**

#Install mutagen library

from mutagen.easyid3 import EasyID3

from mutagen.mp3 import MP3

import pandas as pd

#Song file

file = 'iphone.mp3'

#Reading metadata

audio = MP3(file, ID3=EasyID3)

duration = round(audio.info.length, 2)

title = audio.get('title', [''])[0]

artist = audio.get('artist', [''])[0]

album = audio.get('album', [''])[0]

df = pd.DataFrame([{

'filename': file,

'title': title,

'artist': artist,

'album': album,

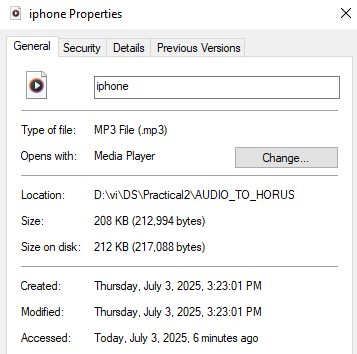
'duration (sec)': duration

}])

df.to\_csv('horus\_audio.csv', index=False)

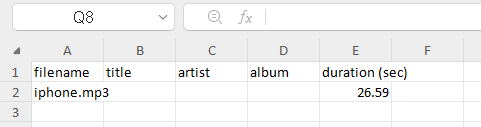
print("Audio to Horus file created")

**Original File**

****

**Audio to Horus Format(Extracting Metadata)**

**Horus\_audio.csv**

****

1. **Convert Video to horus.**

**Code:**

import cv2

import pandas as pd

#Reading Video file

file = 'video.mp4'

cap = cv2.VideoCapture(file)

# Extract basic metadata

width = cap.get(3) # CAP\_PROP\_FRAME\_WIDTH

height = cap.get(4) # CAP\_PROP\_FRAME\_HEIGHT

fps = cap.get(5) # CAP\_PROP\_FPS

frames = cap.get(7) # CAP\_PROP\_FRAME\_COUNT

duration = frames / fps

cap.release()

df = pd.DataFrame([{

'filename': file,

'width': int(width),

'height': int(height),

'fps': round(fps, 2),

'duration (sec)': round(duration, 2)

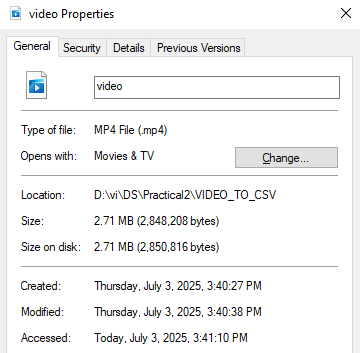
}])

print(df)

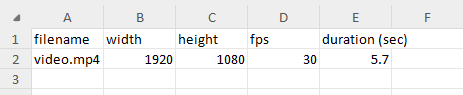
df.to\_csv('video\_horus.csv', index=False)

print("Saved to video\_horus.csv")

**Original File**

****

**Video To Horus Format (Extracting Metadata)**

****

|  |  |  |
| --- | --- | --- |
| K. M. S. P. Mandal’s  Sant Rawool Maharaj Mahavidyalaya, Kudal    **Department of Masters of Science**  **Information Technology** | Date:    Roll no: 17    Expt No: 03 | Signature |
| **Title: Write The programs for the following** | | |

**Fixers Utilities – Theory**

Fixers utilities are data preprocessing techniques used to improve data quality before analysis. They help in cleaning, organizing, and transforming raw data so that the data becomes accurate, reliable, and suitable for further processing.

**Data Binning or Bucketing**

Data binning or bucketing is a technique used to group continuous data into discrete intervals known as bins. Instead of handling individual values, data is represented using ranges, which reduces noise and improves data stability.

The purpose of data binning is to reduce the effect of minor observation errors, to simplify large datasets, and to improve visualization and analysis.

For example, age values can be grouped as 1–10, 11–20, and 21–30 into separate bins.

**Averaging of Data**

Averaging of data is a smoothing technique in which multiple data values are replaced by their mean value. This method is used to reduce fluctuations and noise in data.

The purpose of averaging is to remove random variations, to identify overall trends, and to improve data consistency.

The common types of averaging include simple average and moving average.

**Outlier Detection**

Outlier detection is the process of identifying data values that significantly differ from the majority of the dataset. Outliers may occur due to measurement errors, data entry mistakes, or rare events.

The purpose of outlier detection is to improve data accuracy, to prevent distortion of analysis results, and to detect anomalies or errors.

Common methods of outlier detection include statistical methods such as mean and standard deviation, and distance-based methods.

**Logging**

Logging is the process of recording system events, errors, and data changes during the execution of a program or system.

The purpose of logging is to track system behaviour, to debug errors, and to maintain audit trails.

The types of logs include error logs, activity logs, and system logs.

1. **Fixing Utilities**

**Code:**

import string

import datetime as dt

print('Removing leading or lagging spaces from a data');

mydata = " Hello My name is Mikka Singh "

print('Original Data:',mydata)

cleandata=mydata.lstrip()

print('After Cleaning:',cleandata)

print('Removing bad characters from a data')

charachter\_set = set(string.ascii\_letters + string.digits + ' ')

badlink=r'Data\+0+0Science with\+0+2 funny charac\*ters is \+10b+ad!!!'

cleanlink = ''.join([char for char in badlink if char in charachter\_set])

print('Bad Data : ',badlink);

print('Clean Data : ',cleanlink)

print('Convert YYYY/MM/DD to DD Month YYYY.')

baddate = dt.date(2004,4,3)

baddata=format(baddate,'%Y-%m-%d')

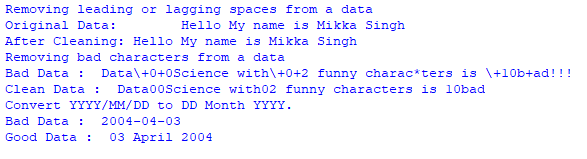
gooddate = dt.datetime.strptime(baddata,'%Y-%m-%d')

gooddata=format(gooddate,'%d %B %Y')

print('Bad Data : ',baddata)

print('Good Data : ',gooddata)

**Output:**

****

1. **Data Binning or Bucketing**

**Code:**

import numpy as np

import matplotlib.pyplot as plt

# Seed for reproducibility

np.random.seed(0)

# Generate sample data

mean = 90

sigma = 25

data = mean + sigma \* np.random.randn(50)

bins = 25

# Creation of histogram

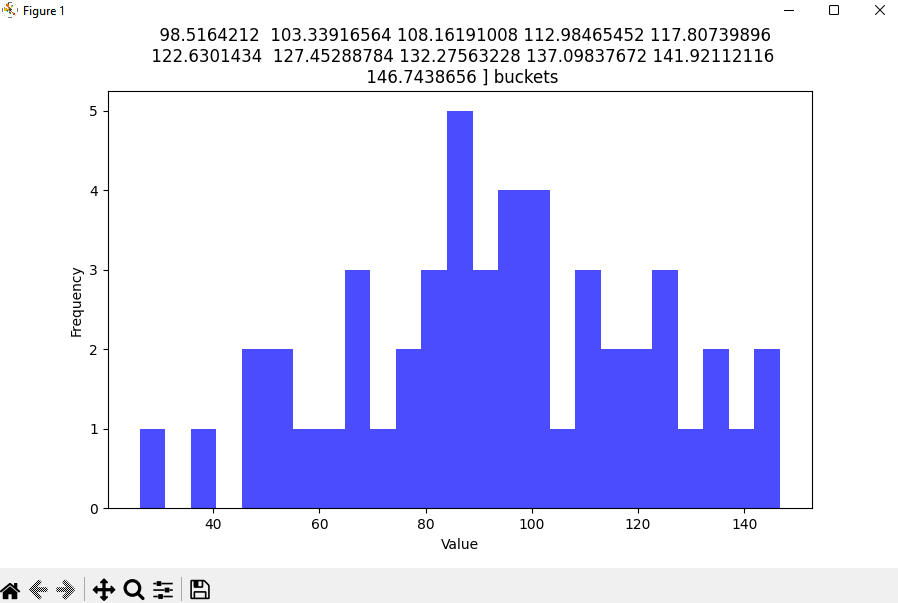
counts, bins, patches = plt.hist(data, bins=bins, density=False, alpha=0.7, color='blue')

plt.xlabel('Value')

plt.ylabel('Frequency')

plt.title(f'Histogram: {len(data)} entries binned into {bins} buckets')

plt.show()



1. **Averaging of data**

**Aim: This practical reads location data from a file. It picks country, place, and latitude columns and cleans the data. Then, it finds the average latitude for each place in every country.**

**Code:**

import pandas as pd

OutputFile='Retrieve\_Router\_Location.csv'

sFileName= ‘C:/VKHCG/01-Vermeulen/00-RawData/IP\_DATA\_CORE.csv '

IP\_DATA\_ALL=pd.read\_csv(sFileName,header=0,low\_memory=False,usecols=['Country','Place Name','Latitude','Longitude'], encoding="latin-1")

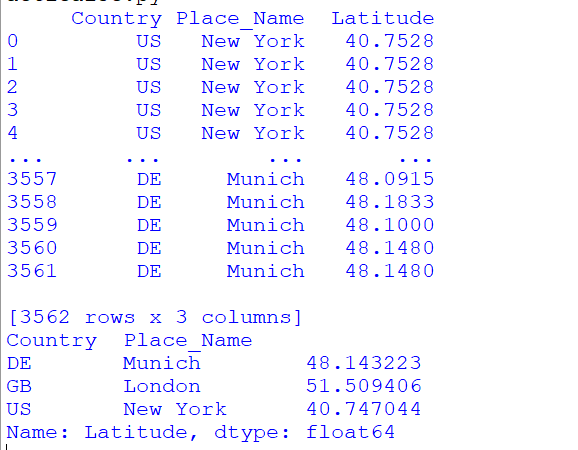
IP\_DATA\_ALL.rename(columns={'Place Name': 'Place\_Name'}, inplace=True)

AllData=IP\_DATA\_ALL[['Country', 'Place\_Name','Latitude']]

print(AllData)

MeanData=AllData.groupby(['Country', 'Place\_Name'])['Latitude'].mean()

print(MeanData)



1. **Outlier Detection**

**Code:**

import pandas as pd

# Read the file (replace with your actual path)

df = pd.read\_csv('C:/msc\_dataset/Practical\_3c/IP\_DATA\_CORE.csv', usecols=['Country', 'Place Name', 'Latitude'], encoding='latin-1')

df.rename(columns={'Place Name': 'Place\_Name'}, inplace=True)

# Filter for London

data = df[df['Place\_Name'] == 'London']

# Calculate mean and std

mean = data['Latitude'].mean()

std = data['Latitude'].std()

# Calculate bounds

lower, upper = mean - std, mean + std

# Filter data

outliers\_high = data[data['Latitude'] > upper]

outliers\_low = data[data['Latitude'] < lower]

not\_outliers = data[(data['Latitude'] >= lower) & (data['Latitude'] <= upper)]

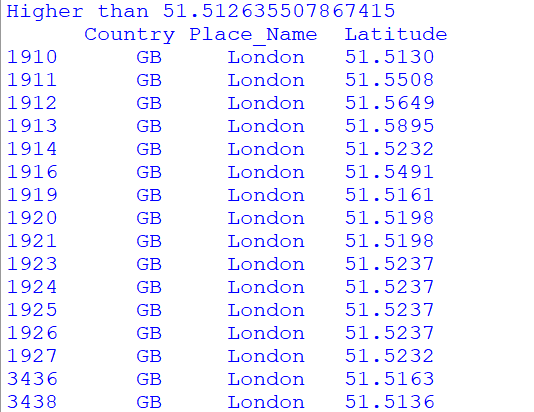
# Print results

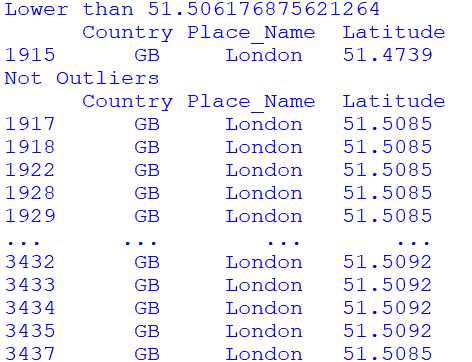
print('Higher than', upper, '\n', outliers\_high)

print('Lower than', lower, '\n', outliers\_low)

print('Not Outliers\n', not\_outliers)

**Output:**

****

****

1. **Logging.**

**Code:**

import os, logging, uuid, shutil

Base = 'C:/VKHCG'

sCompanies = ['01-Vermeulen','02-Krennwallner','03-Hillman','04-Clark']

sLayers = ['01-Retrieve','02-Assess','03-Process','04-Transform','05-Organise','06-Report']

sLevels = ['debug','info','warning','error']

for sCompany in sCompanies:

for sLayer in sLayers:

# Setup logging directory

sFileDir = os.path.join(Base, sCompany, sLayer, 'Logging')

if os.path.exists(sFileDir):

shutil.rmtree(sFileDir)

os.makedirs(sFileDir)

# Unique log file

sLogFile = os.path.join(sFileDir, f'Logging\_{uuid.uuid4()}.log')

print('Set up:', sLogFile)

# Configure logging

logging.basicConfig(

level=logging.DEBUG,

format='%(asctime)s %(name)-12s %(levelname)-8s %(message)s',

datefmt='%m-%d %H:%M',

filename=sLogFile,

filemode='w'

)

console = logging.StreamHandler()

console.setLevel(logging.INFO)

console.setFormatter(logging.Formatter('%(name)-12s: %(levelname)-8s %(message)s'))

logging.getLogger('').addHandler(console)

# Log messages

logging.info('Practical Data Science is fun!')

log\_methods = {'debug': logging.debug, 'info': logging.info,

'warning': logging.warning, 'error': logging.error}

for sLevel in sLevels:

logger\_name = f'Application-{sCompany}-{sLayer}-{sLevel}'

logger = logging.getLogger(logger\_name)

log\_methods[sLevel](f'Practical Data Science logged a {sLevel} message.')

**Output:**

****

|  |  |  |
| --- | --- | --- |
| K.M.S.P.Mandal’s  Sant Rawool Maharaj Mahavidyalaya, Kudal    **Department of Masters of Science**  **Information Technology** | Date:    Roll no:    Expt No: | Signature |
| **Title: Write The programs for the following** | | |

**Data Fixing and Statistical Analysis using R**

This program reads a CSV file containing IP-related data and performs basic data cleaning and exploratory analysis.

The dataset is loaded using the read.csv() function, and column names are standardized by replacing spaces with dots to make them easier to reference in R.

Duplicate rows are removed using the unique() function to improve data quality. The program then displays the total number of rows and columns to give an overview of the dataset size.

A frequency table is generated for the Country attribute to understand the distribution of records across different countries. Finally, descriptive statistics such as minimum, maximum, mean, median, and standard deviation are calculated for the Latitude and Longitude attributes, which helps in understanding the geographical spread of the data.

**Retrieving Different Attributes of Data using Python**

This program demonstrates how to retrieve and inspect dataset attributes using Python and the Pandas library. The CSV file is loaded into a DataFrame, and an output directory is created to store the processed results.

The program prints the total number of rows and columns, which provides an understanding of dataset dimensions. It then displays the raw column names and their data types.

To improve consistency, column names are cleaned by removing extra spaces and replacing spaces with dots.

The cleaned dataset is then indexed with a RowID and exported to a new CSV file. This program is useful for attribute inspection, metadata understanding, and data standardization.

**Data Pattern Analysis using R**

This program focuses on identifying data patterns in the Country attribute. After reading the CSV file, unique country values are extracted.

Pattern analysis is performed by replacing alphabetic characters with ‘A’, numeric characters with ‘N’, spaces with ‘b’, and special characters with ‘u’. This helps in identifying structural patterns in textual data and detecting inconsistencies.

The final output displays each country along with its derived pattern, which is useful for data validation and standardization.

**IP Routing and Distance Calculation using Python**

This program calculates geographical distances between different IP locations using latitude and longitude values.

The dataset is read with selected attributes such as Country, Place Name, Latitude, and Longitude. Duplicate records are removed, and a Cartesian join is performed to compare every location with every other location.

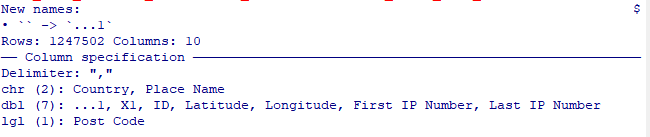
The Haversine formula is used to calculate distances between two geographical points in both kilometers and miles.

The resulting routing and distance information is stored in a CSV file. This program is useful for network routing analysis, geographical data analysis, and distance-based computations.

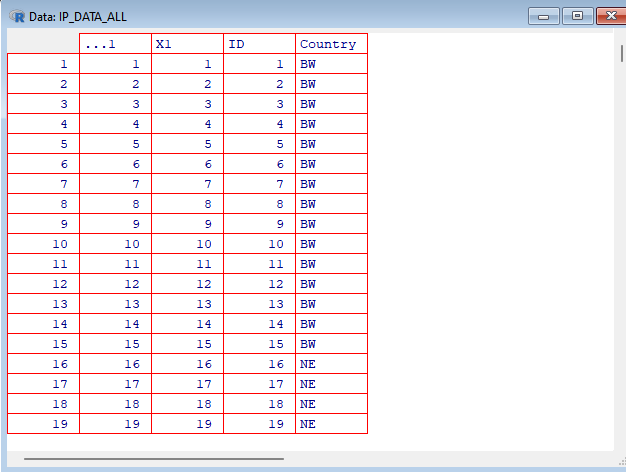
1. **Perform following data processing using R First Install R Library**

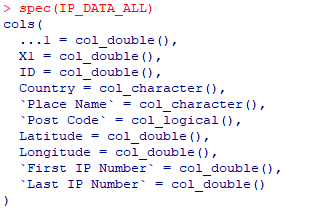
install.packages("readr")

IP\_DATA\_ALL=read\_csv(‘c:/VKHCG/01-Vermeulen/00-RawData/IP\_DATA\_ALL.csv’)



view(IP\_DATA\_ALL)



****

# Read the CSV file

FileName = "c:/VKHCG/01-Vermeulen/00-RawData/IP\_DATA\_ALL.csv"

IP\_DATA\_ALL = read.csv(FileName, stringsAsFactors = FALSE)

# Fix column names by replacing spaces with dots

names(IP\_DATA\_ALL) = gsub(" ", ".", names(IP\_DATA\_ALL))

# Remove duplicate rows

IP\_DATA\_ALL\_FIX = unique(IP\_DATA\_ALL)

# Print basic info about the dataset

cat("Rows:", nrow(IP\_DATA\_ALL\_FIX), "Columns:", ncol(IP\_DATA\_ALL\_FIX), "\n")

# Create a frequency table of countries

CountryFreq = as.data.frame(table(IP\_DATA\_ALL\_FIX$Country))

names(CountryFreq) = c("Country", "Frequency")

View(CountryFreq)

# Calculate and print statistics for Latitude

cat("Latitude -> Min:", min(IP\_DATA\_ALL\_FIX$Latitude, na.rm=TRUE),

"Max:", max(IP\_DATA\_ALL\_FIX$Latitude, na.rm=TRUE),

"Mean:", mean(IP\_DATA\_ALL\_FIX$Latitude, na.rm=TRUE),

"Median:", median(IP\_DATA\_ALL\_FIX$Latitude, na.rm=TRUE),

"SD:", sd(IP\_DATA\_ALL\_FIX$Latitude, na.rm=TRUE), "\n")

# Calculate and print statistics for Longitude

cat("Longitude -> Min:", min(IP\_DATA\_ALL\_FIX$Longitude, na.rm=TRUE),

"Max:", max(IP\_DATA\_ALL\_FIX$Longitude, na.rm=TRUE),

"Mean:", mean(IP\_DATA\_ALL\_FIX$Longitude, na.rm=TRUE),

"Median:", median(IP\_DATA\_ALL\_FIX$Longitude, na.rm=TRUE),

"SD:", sd(IP\_DATA\_ALL\_FIX$Longitude, na.rm=TRUE), "\n")

**Output:**

> # Read the CSV file

> FileName = "c:/VKHCG/01-Vermeulen/00-RawData/IP\_DATA\_ALL.csv"

> IP\_DATA\_ALL = read.csv(FileName, stringsAsFactors = FALSE)

>

> # Fix column names by replacing spaces with dots

> names(IP\_DATA\_ALL) = gsub(" ", ".", names(IP\_DATA\_ALL))

>

> # Remove duplicate rows

> IP\_DATA\_ALL\_FIX = unique(IP\_DATA\_ALL)

>

> # Print basic info about the dataset

> cat("Rows:", nrow(IP\_DATA\_ALL\_FIX), "Columns:", ncol(IP\_DATA\_ALL\_FIX), "\n")

Rows: 1247502 Columns: 9

>

> # Create a frequency table of countries

> CountryFreq = as.data.frame(table(IP\_DATA\_ALL\_FIX$Country))

> names(CountryFreq) = c("Country", "Frequency")

> View(CountryFreq)

>

> # Calculate and print statistics for Latitude

> cat("Latitude -> Min:", min(IP\_DATA\_ALL\_FIX$Latitude, na.rm=TRUE),

+ "Max:", max(IP\_DATA\_ALL\_FIX$Latitude, na.rm=TRUE),

+ "Mean:", mean(IP\_DATA\_ALL\_FIX$Latitude, na.rm=TRUE),

+ "Median:", median(IP\_DATA\_ALL\_FIX$Latitude, na.rm=TRUE),

+ "SD:", sd(IP\_DATA\_ALL\_FIX$Latitude, na.rm=TRUE), "\n")

Latitude -> Min: -54.2767 Max: 78.2167 Mean: 39.58496 Median: 41.9232 SD: 16.85403

>

> # Calculate and print statistics for Longitude

> cat("Longitude -> Min:", min(IP\_DATA\_ALL\_FIX$Longitude, na.rm=TRUE),

+ "Max:", max(IP\_DATA\_ALL\_FIX$Longitude, na.rm=TRUE),

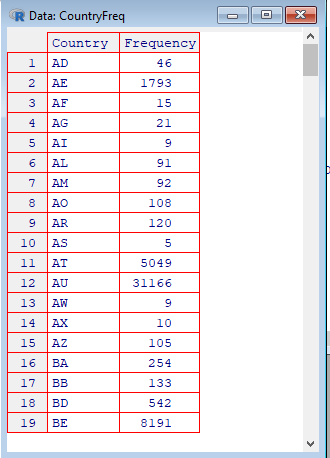
+ "Mean:", mean(IP\_DATA\_ALL\_FIX$Longitude, na.rm=TRUE),

+ "Median:", median(IP\_DATA\_ALL\_FIX$Longitude, na.rm=TRUE),

+ "SD:", sd(IP\_DATA\_ALL\_FIX$Longitude, na.rm=TRUE), "\n")

Longitude -> Min: -176.5 Max: 179.2167 Mean: -23.43114 Median: -2.4 SD: 71.78892

>

****

**B) Program retrieve different attributes of data**

**Code:**

import os

import pandas as pd

# Load the dataset

input\_file = ‘c:/VKHCG/01-Vermeulen/00-RawData/IP\_DATA\_ALL.csv '

df = pd.read\_csv(input\_file, header=0, low\_memory=False, encoding="latin-1")

# Create output folder if it doesn't exist

output\_dir = 'C:/msc\_dataset/Praqctical\_4c/4c-Python'

os.makedirs(output\_dir, exist\_ok=True)

# Step 3: Show dataset dimensions

print('Rows:', df.shape[0])

print('Columns:', df.shape[1])

# Step 4: Show raw column names

print('Raw Data Set')

for col in df.columns:

print(col, type(col))

# Step 5: Clean column names (replace spaces with dots)

df.columns = [col.strip().replace(" ", ".") for col in df.columns]

# Step 6: Show cleaned column names

print('Fixed Data Set')

for col in df.columns:

print(col, type(col))

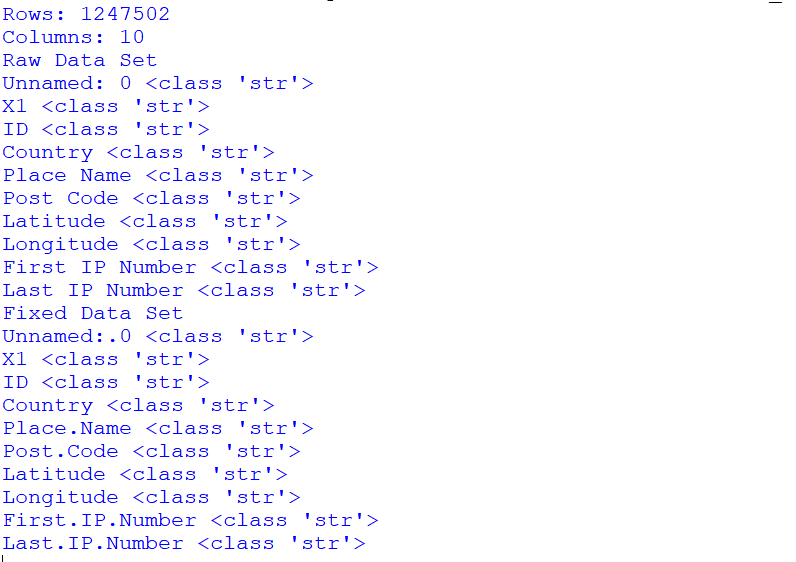
# Step 7: Set index name and export to CSV

df.index.name = 'RowID'

output\_file = os.path.join(output\_dir, 'Retrieve\_IP\_DATA.csv')

df.to\_csv(output\_file, index=True, encoding="latin-1")

**Output:**

****

1. **Data pattern.**

**Code:**

FileName = "c:/VKHCG/01-Vermeulen/00-RawData/IP\_DATA\_ALL.csv"

IP\_DATA\_ALL = read.csv(FileName, stringsAsFactors = FALSE)

Country = unique(IP\_DATA\_ALL$Country)

Pattern = gsub("[^A-Za-z0-9 ]", "u",

gsub("[0-9]", "N",

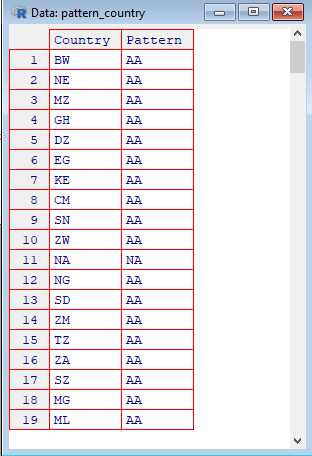
gsub("[A-Za-z]", "A",

gsub(" ", "b", Country))))

pattern\_country = data.frame(Country = Country, Pattern = Pattern)

View(pattern\_country)

**Output:**

****

1. **Load IP\_DATA\_ALL**

import os

import pandas as pd

# Input file

input\_file = ''C:/VKHCG'/01-Vermeulen/00-RawData/IP\_DATA\_ALL.csv'

print('Loading:', input\_file)

# Read CSV

df = pd.read\_csv(input\_file, encoding='latin-1', low\_memory=False)

# Fix column names

df.columns = [col.strip().replace(' ', '.') for col in df.columns]

# Add RowID

df.index.name = 'RowID'

# Output directory

output\_dir = BASE + '/01-Vermeulen/01-Retrieve/01-EDS/02-Python'

os.makedirs(output\_dir, exist\_ok=True)

# Save file

output\_file = output\_dir + '/Retrieve\_IP\_DATA.csv'

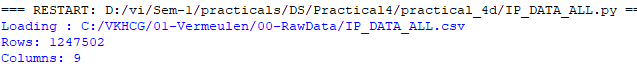
df.to\_csv(output\_file, encoding='latin-1')

# Info

print('Rows:', df.shape[0])

print('Columns:', df.shape[1])

print('Done!')

****

**Code: Retriew\_IP\_Routing.py**

import os

import pandas as pd

from math import radians, sin, cos, asin, sqrt

def haversine(lon1, lat1, lon2, lat2, km=True):

lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])

a = sin((lat2-lat1)/2)\*\*2 + cos(lat1)\*cos(lat2)\*sin((lon2-lon1)/2)\*\*2

r = 6371 if km else 3956

return round(2\*asin(sqrt(a))\*r, 3)

df = pd.read\_csv(

'C:/VKHCG/01-Vermeulen/00-RawData/IP\_DATA\_CORE.csv',

usecols=['Country', 'Place Name', 'Latitude', 'Longitude'],

encoding='latin-1'

)

os.makedirs('C:/VKHCG/01-Vermeulen/01-Retrieve/01-EDS/02-Python', exist\_ok=True)

df = df.drop\_duplicates().rename(columns={'Place Name': 'Place\_Name'})

df['K'] = 1

cross = df.merge(df, on='K').drop('K', axis=1)

cross['DistanceKM'] = cross.apply(

lambda r: haversine(r.Longitude\_x, r.Latitude\_x, r.Longitude\_y, r.Latitude\_y),

axis=1

)

cross['DistanceMiles'] = cross.apply(

lambda r: haversine(r.Longitude\_x, r.Latitude\_x, r.Longitude\_y, r.Latitude\_y, False),

axis=1

)

cross.to\_csv(

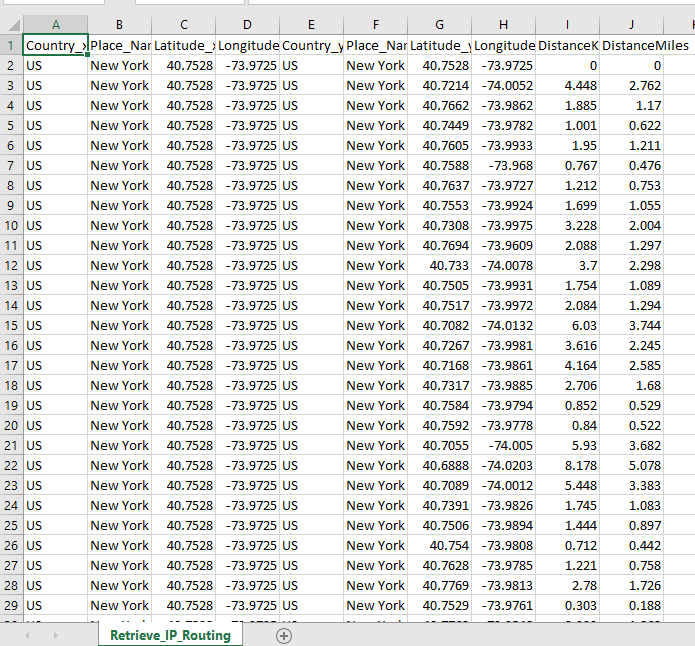
'C:/VKHCG/01-Vermeulen/01-Retrieve/01-EDS/02-Python/Retrieve\_IP\_Routing.csv',

index=False,

encoding='latin-1'

)

print('Done')

****

|  |  |  |
| --- | --- | --- |
| K. M. S. P. Mandal’s  Sant Rawool Maharaj Mahavidyalaya, Kudal    **Department of Masters of Science**  **Information Technology** | Date:    Roll no: 17    Expt No: 05 | Signature |
| **Title: Write The programs for the following** | | |

**Drop Columns Where All Values Are Missing**

**Aim**  
 To clean a dataset by removing columns in which all values are missing, ensuring that only columns with useful data are retained.

**Description**   
 The program starts by importing the pandas and os libraries. The input CSV file path and output file path are defined. The output directory is created if it does not exist using os.makedirs(). The dataset is loaded into a DataFrame using pd.read\_csv(). Columns where all values are missing are removed using data.dropna(axis=1, how='all'). The cleaned DataFrame is then saved to a new CSV file using to\_csv(). Finally, a message is printed to confirm successful processing.

Usage of Functions  
 pd.read\_csv() reads the CSV file into a DataFrame.

os.makedirs() ensures the output directory exists.

dropna(axis=1, how='all') removes columns where all values are missing.

to\_csv() saves the cleaned dataset to a CSV file.

**Drop Columns Where Any Value Is Missing**

**Aim**  
 To remove columns that contain even a single missing value, keeping only fully complete columns for analysis.

**Description**   
 The program imports the required libraries and defines input and output paths. The output directory is created if it does not exist. The dataset is loaded into a Pandas DataFrame. Columns with any missing values are dropped using data.dropna(axis=1, how='any'). The cleaned dataset is saved to a CSV file using to\_csv(). A message is printed to indicate that columns with missing values have been removed.

Usage of Functions  
 pd.read\_csv() loads the CSV file.

os.makedirs() ensures the output directory is present.

dropna(axis=1, how='any') removes columns with even a single missing value.

to\_csv() exports the cleaned dataset.

**Remove Rows with More Than Two Missing Values**

**Aim**  
To retain rows that have at most two missing values, removing rows that are mostly incomplete.

**Description**   
The program begins by importing pandas and os. Input and output file paths are defined, and the output directory is created if it does not exist. The dataset is loaded using pd.read\_csv(). Rows having more than two missing values are removed using data.dropna(thresh=len(data.columns)-2). The cleaned DataFrame is saved to a CSV file. A message is printed to confirm that rows with more than two missing values have been removed.

1. **Perform error management on the given data using pandas’ package.**

**i. Drop the Columns Where All Elements Are Missing Values**

**Code:**

import pandas as pd

import os

# Input and Output file paths

input\_file = "C:/VKHCG/01-Vermeulen/00-RawData/Country\_Currency\_edited.csv"

output\_file = "C:/VKHCG/01-Vermeulen/02-Assess/01-EDS/02-Python/drop\_columns\_all\_values\_are\_missing.csv"

# Create output folder if it does not exist

os.makedirs("C:/VKHCG/01-Vermeulen/02-Assess/01-EDS/02-Python", exist\_ok=True)

# Read Excel file (CORRECT FUNCTION)

data = pd.read\_csv(input\_file)

# Drop columns where all values are missing

clean\_data = data.dropna(axis=1, how='all')

# Save cleaned Excel file

clean\_data.to\_csv(output\_file, index=False)

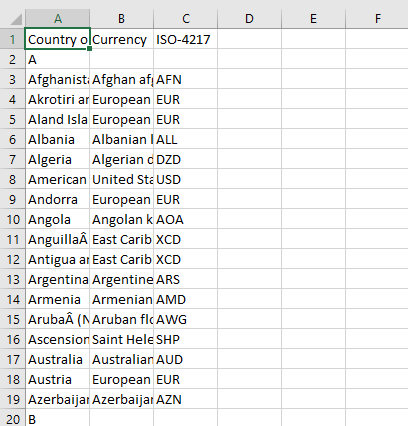
print("File processed successfully.")

****

**Output:**

****

**Final file after removing empty column**

****

**ii) Drop columns where any value is missing**

**Code:**

# Drop columns where any value is missing

import pandas as pd

import os

# Straight input and output paths

input\_file = "C:/VKHCG/01-Vermeulen/00-RawData/Country\_Currency\_edited.csv"

output\_dir = "C:/VKHCG/01-Vermeulen/02-Assess/01-EDS/02-Python"

output\_file = "C:/VKHCG/01-Vermeulen/02-Assess/01-EDS/02-Python/drop\_columns\_any\_value\_is\_missing.csv"

# Create output directory if it does not exist

os.makedirs(output\_dir, exist\_ok=True)

# Load data

data = pd.read\_csv(input\_file)

# Drop columns having any missing values

clean\_data = data.dropna(axis=1, how='any')

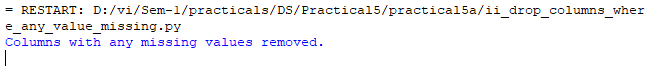
# Save cleaned file

clean\_data.to\_csv(output\_file, index=False)

print("Columns with any missing values removed.")

****

**Output**

****

****

**iii) Remove rows with maximum two missing values**

**Code:**

# Remove rows with maximum two missing values

import pandas as pd

import os

# Straight input and output paths

input\_file = "C:/VKHCG/01-Vermeulen/00-RawData/Country\_Currency\_edited.csv"

output\_dir = "C:/VKHCG/01-Vermeulen/02-Assess/01-EDS/02-Python"

output\_file = "C:/VKHCG/01-Vermeulen/02-Assess/01-EDS/02-Python/keep\_rows\_with\_maximum\_two\_missing\_values.csv.csv"

# Create output directory if it does not exist

os.makedirs(output\_dir, exist\_ok=True)

# Load data

data = pd.read\_csv(input\_file)

# Keep rows having at least (total columns - 2) non-missing values

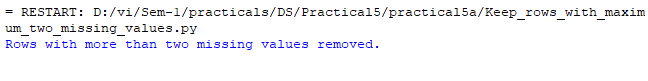
clean\_data = data.dropna(thresh=len(data.columns) - 2)

# Save cleaned data

clean\_data.to\_csv(output\_file, index=False)

print("Rows with more than two missing values removed.")

****

**Output: **

****

1. **Write python program to create the network routing diagram from the given data on routers.**

**Code:**

# Merge Country, Company, and Customer data

import pandas as pd

import os

# Disable chained assignment warning

pd.options.mode.chained\_assignment = None

# File paths (straight format)

country\_file = "C:/VKHCG/01-Vermeulen/01-Retrieve/01-EDS/01-R/Retrieve\_Country\_Code.csv"

company\_file = "C:/VKHCG/01-Vermeulen/01-Retrieve/01-EDS/02-Python/Retrieve\_Router\_Location.csv"

customer\_file = "C:/VKHCG/01-Vermeulen/01-Retrieve/01-EDS/01-R/Retrieve\_IP\_DATA.csv"

output\_dir = "C:/VKHCG/01-Vermeulen/02-Assess/01-EDS/02-Python"

output\_file = output\_dir + "/network\_routing\_output.csv"

# Create output folder

os.makedirs(output\_dir, exist\_ok=True)

# Load data files

country = pd.read\_csv(country\_file, encoding="latin-1",low\_memory=False)

company = pd.read\_csv(company\_file, encoding="latin-1",low\_memory=False)

customer = pd.read\_csv(customer\_file, encoding="latin-1",low\_memory=False)

# Clean and rename country data

country = country.rename(columns={

"Country": "Country\_Name",

"ISO-2-CODE": "Country\_Code"

}).drop(columns=["ISO-M49", "ISO-3-Code", "RowID"])

# Clean and rename company data

company = company.rename(columns={"Country": "Country\_Code"})

# Remove rows with missing values and rename customer column

customer = customer.dropna()

customer = customer.rename(columns={"Country": "Country\_Code"})

# Merge company and country data

merged\_data = pd.merge(company, country, on="Country\_Code", how="inner")

# Add prefix 'Company\_' to all column names

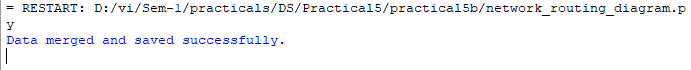
merged\_data = merged\_data.add\_prefix("Company\_")

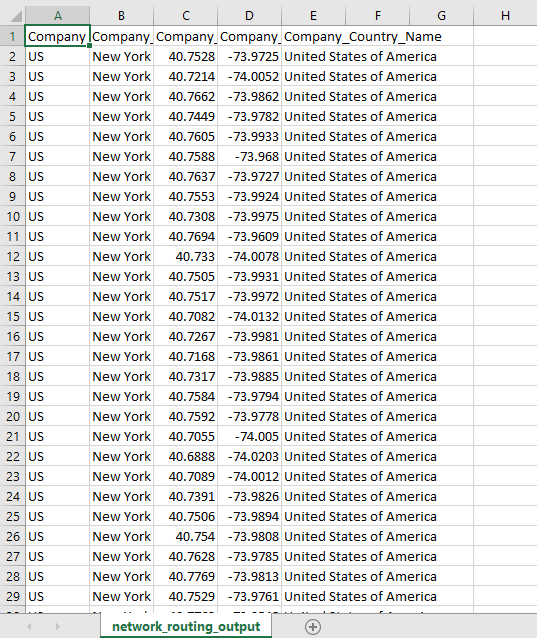
# Save final data

merged\_data.to\_csv(output\_file, index=False, encoding="latin-1")

print("Data merged and saved successfully.")

**Output**:

****

****

**ii) Asses network routing customer location.**

**Code:**

# ii) Access customers' location using network router location

import pandas as pd

import os

input\_file = "C:/VKHCG/01-Vermeulen/02-Assess/01-EDS/02-Python/Assess-Network-Routing-Customer.csv"

output\_dir = "C:/VKHCG/01-Vermeulen/02-Assess/01-EDS/02-Python"

output\_file = output\_dir + "/Assess-Network-Routing-Customer.gml"

# Create output directory if it does not exist

os.makedirs(output\_dir, exist\_ok=True)

# Load customer data

customer\_data = pd.read\_csv(input\_file, encoding="latin-1", low\_memory=False)

# Display loaded data (first 5 rows)

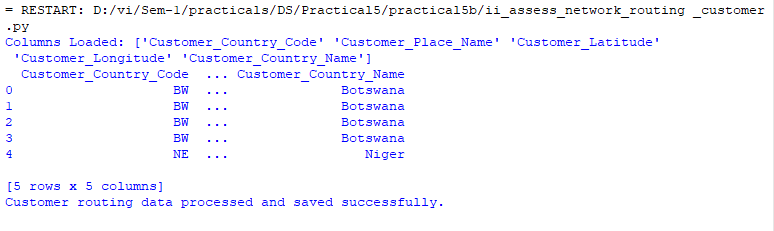
print("Columns Loaded:", customer\_data.columns.values)

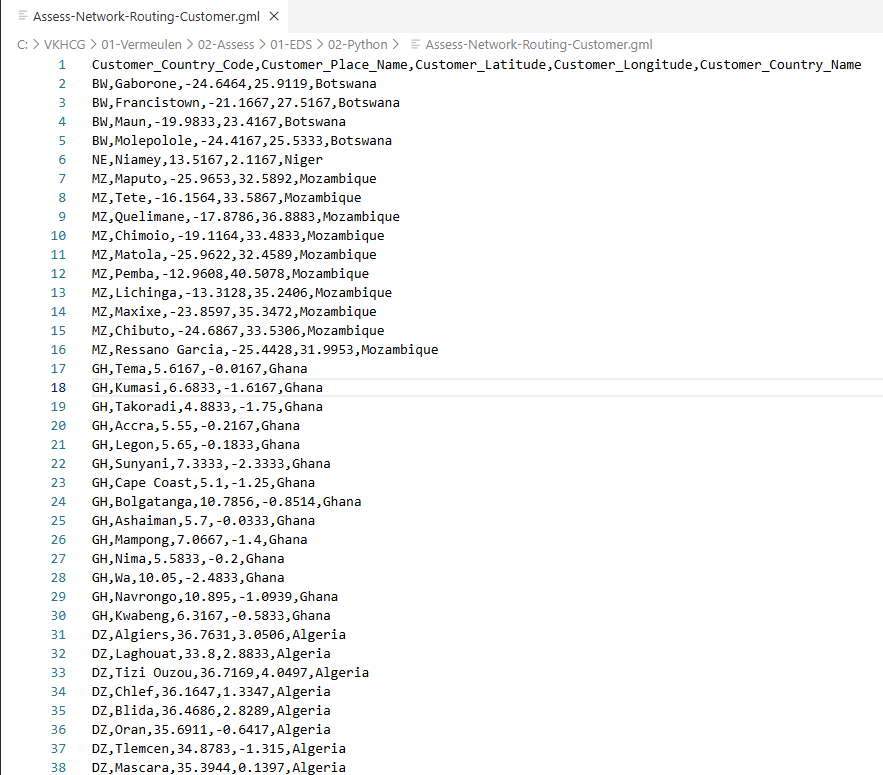
print(customer\_data.head())

# Save the data in GML format (Graph Modeling Language)

customer\_data.to\_csv(output\_file, index=False, encoding="latin-1")

print("Customer routing data processed and saved successfully.")

****

****

1. **Write a python program to build acyclic graph.**

**Code:**

#Directed Acyclic Graph.

import pandas as pd

import networkx as nx

import matplotlib.pyplot as plt

import os

# File paths

input\_file = "C:/VKHCG/01-Vermeulen/01-Retrieve/01-EDS/02-Python/Retrieve\_Router\_Location.csv"

output\_dir = "C:/VKHCG/01-Vermeulen/02-Assess/01-EDS/02-Python"

os.makedirs(output\_dir, exist\_ok=True)

# Load data

df = pd.read\_csv(input\_file, encoding="latin-1", low\_memory=False)

# Create and save DAGs

for col, color, fname in [(df['Country'], 'green', "Assess-DAG-Company-Country.png"),

(df['Place\_Name'] + '-' + df['Country'], 'blue', "Assess-DAG-Company-Country-Place.png")]:

G = nx.DiGraph()

G.add\_nodes\_from(col)

G.add\_edges\_from((n1, n2) for n1 in col for n2 in col if n1 != n2)

nx.draw(G, pos=nx.spring\_layout(G), node\_color='red', edge\_color=color, with\_labels=True,

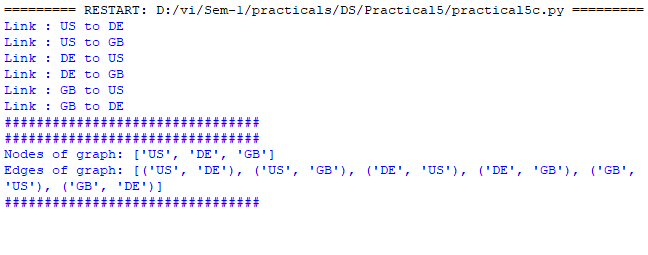
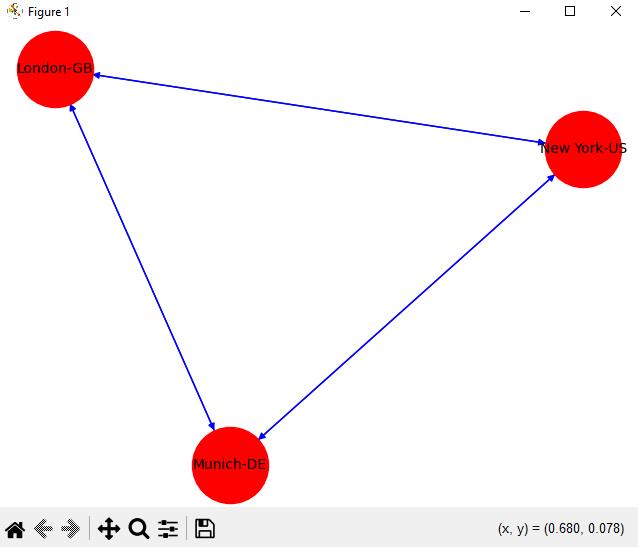
node\_size=3000, font\_size=10)

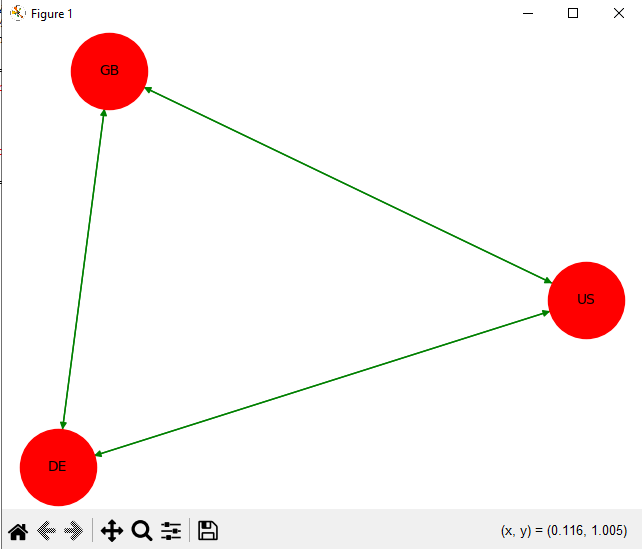
plt.savefig(os.path.join(output\_dir, fname))

plt.show()

print("DAGs created and saved successfully.")

**Output**

****

****

|  |  |  |
| --- | --- | --- |
| K.M.S.P.Mandal’s  Sant Rawool Maharaj Mahavidyalaya, Kudal    **Department of Masters of Science**  **Information Technology** | Date:    Roll no: 17    Expt No: 06 | Signature |
| **Title: Write The programs for the following** | | |

1. **Create a Time Hub** Generate a master list of UTC timestamps covering the required date and time range. This list acts as the central reference for all time-based data in the system. The timestamps are stored as a Hub-Time table in SQLite, ensuring a single source of truth for time and enabling consistent linking with other tables. Indexing the hub allows for faster queries when accessing or joining time-based data.
2. **Generate Time Zones (Satellites)**Convert the UTC timestamps from the Hub into all world time zones, including offsets for daylight saving time where applicable. Each time zone is stored as a separate Satellite table linked to the Hub-Time table. This structure allows efficient storage of derived time attributes such as local time, day, month, year, and weekday without modifying the central Hub. Satellites provide flexibility to add or update time zone conversions in the future without affecting the Hub.
3. **Implement Data Vault Concepts**Demonstrate the Data Vault modeling approach by separating unique identifiers (Hubs) from descriptive attributes (Satellites). The Hub-Time table stores unique UTC timestamps, while the satellite tables store related attributes such as local time and zone information. This approach ensures scalability, auditability, and flexibility, allowing historical tracking of changes in time attributes and supporting advanced analytics without redundancy.
4. **Verify and Store Data**Display the contents of the Hub and Satellite tables to ensure correct data generation and proper linking between tables. Store the tables in SQLite with appropriate indexes on key columns to enable fast querying and efficient analytics. Verification ensures data integrity, consistency between Hub and Satellites, and readiness for future time-based analysis and reporting**.**
5. **Build the time Hub, Link and Satellite**

**Code:**

import os, uuid, sqlite3 as sq, pandas as pd

from datetime import datetime, timedelta

from pytz import timezone, all\_timezones

print("Starting program")

os.makedirs("C:/VKHCG", exist\_ok=True)

con = sq.connect("C:/VKHCG/time.db")

print("Database opened")

print("Generating UTC time data")

df = pd.DataFrame([

[str(uuid.uuid4()),

(datetime(2018,1,1)-timedelta(h)).strftime("%Y-%m-%d-%H"),

(datetime(2018,1,1)-timedelta(h)).replace(tzinfo=timezone("UTC"))]

for h in range(24)

], columns=["ID","Key","UTC"]).set\_index("ID")

print("Storing Hub-Time table")

df[["Key"]].to\_sql("Hub-Time", con, if\_exists="replace")

print("Creating timezone tables")

for z in all\_timezones:

pd.DataFrame({

"ID":[str(uuid.uuid4()) for \_ in df.index],

"Key":df["Key"],

"Time":df["UTC"].apply(lambda x: x.astimezone(timezone(z)).strftime("%Y-%m-%d %H"))

}).set\_index("ID").to\_sql(f"Time-{z.replace('/','-')}", con, if\_exists="replace")

print("Showing tables in database")

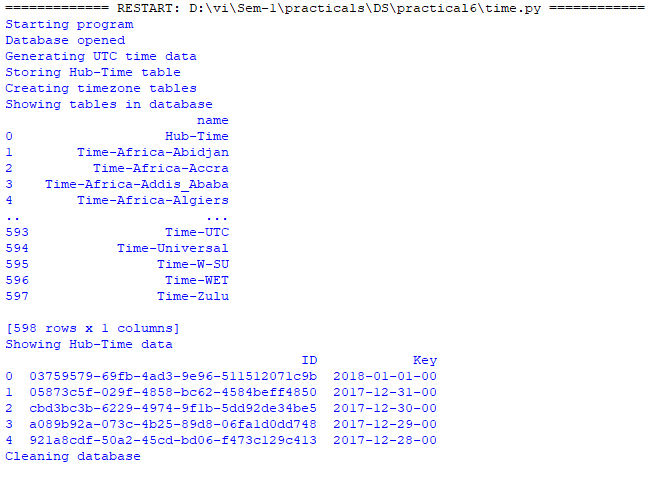
print(pd.read\_sql("SELECT name FROM sqlite\_master WHERE type='table'", con))

print("Showing Hub-Time data")

print(pd.read\_sql("SELECT \* FROM 'Hub-Time' LIMIT 5", con))

con.execute("VACUUM;")

con.close()

****

|  |  |  |
| --- | --- | --- |
| K.M.S.P.Mandal’s  Sant Rawool Maharaj Mahavidyalaya, Kudal    **Department of Masters of Science**  **Information Technology** | Date:    Roll no: 17    Expt No: 07 | Signature |
| **Title: Transform Data** | | |

Transforming data from a Data Vault to a Data Warehouse involves converting raw, historical data into dimensional structures that are optimized for reporting, analytics, and decision-making. The process ensures data consistency, standardization, and auditability, while supporting star schema design for efficient querying.

**a) Transform Time and Person from Data Vault to Data Warehouse**

* Convert UTC timestamps from the Time Hub into local times for accurate analysis.
* Load the Time Hub and its Satellite tables into the Data Vault to maintain historical consistency.
* Populate dimension tables in the Data Warehouse for reporting and analytics.
* Transform the Person entity into a Golden Nominal view to provide a single, authoritative source of person-related information.

**b) Build Time Dimension, Person Dimension, and Fact Table (Star Model)**

* Create Dim-Time using UTC timestamps and derived attributes such as year, month, day, weekday, and local time.
* Build Dim-Person using the consolidated Golden Nominal view with relevant personal attributes.
* Construct Fact-Person-Born-At-Time to link Dim-Time and Dim-Person, capturing events or time-related occurrences.
* Load these tables into the Transform Database and Data Warehouse following the star schema for fast queries and reporting.

**c) Data Vault to Data Warehouse**

* Transform historical data from Hubs, Links, and Satellites into dimension and fact tables.
* Preserve historical changes while optimizing data for analytics and reporting.
* Ensure that the data is clean, standardized, and consistent across the warehouse.

1. **Transform Time and Person from Data Vault to Data Warehouse**

**Purpose of this Transform**

* **Convert UTC time → local time**
* **Load Time Hub + Satellite into Data Vault**
* **Load Dimension tables into Data Warehouse**
* **Transform Person (Golden Nominal view)**

**Code:**

import os

import uuid

import pandas as pd

import sqlite3 as sq

from datetime import datetime

from pytz import timezone

COMPANY = '01-Vermeulen'

# Database paths

DV\_DB = 'C:/VKHCG/88-DV/datavault.db'

DW\_DB = 'C:/VKHCG/99-DW/datawarehouse.db'

# Ensure DW directory exists

os.makedirs('C:/VKHCG/99-DW', exist\_ok=True)

# Database connections

conn\_dv = sq.connect(DV\_DB)

conn\_dw = sq.connect(DW\_DB)

# TIME TRANSFORMATION

# Birth time in UTC

birth\_utc = datetime(1960, 12, 20, 10, 15, 0, tzinfo=timezone('UTC'))

birth\_zone = 'Atlantic/Reykjavik'

# Convert UTC to local zone

birth\_local = birth\_utc.astimezone(timezone(birth\_zone))

# Generate keys

time\_id = str(uuid.uuid4())

datetime\_key = birth\_utc.strftime("%Y-%m-%d-%H-%M-%S")

# Time Hub record

time\_hub = pd.DataFrame([{

"TimeID": time\_id,

"ZoneBaseKey": "UTC",

"DateTimeKey": datetime\_key,

"DateTimeValue": birth\_utc.strftime("%Y-%m-%d %H:%M:%S")

}]).set\_index("TimeID")

# Store Time Hub

time\_hub.to\_sql("Hub-Time-Gunnarsson", conn\_dv, if\_exists="replace")

time\_hub.to\_sql("Dim-Time-Gunnarsson", conn\_dw, if\_exists="replace")

# Time Satellite record

time\_sat = pd.DataFrame([{

"TimeID": time\_id,

"Zone": birth\_zone,

"DateTimeValue": birth\_local.strftime("%Y-%m-%d %H:%M:%S")

}]).set\_index("TimeID")

zone\_fix = birth\_zone.replace('/', '-')

# Store Time Satellite

time\_sat.to\_sql(f"Satellite-Time-{zone\_fix}-Gunnarsson", conn\_dv, if\_exists="replace")

time\_sat.to\_sql(f"Dim-Time-{zone\_fix}-Gunnarsson", conn\_dw, if\_exists="replace")

# PERSON TRANSFORMATION

first\_name = "Guðmundur"

last\_name = "Gunnarsson"

person\_id = str(uuid.uuid4())

# Person Hub / Dimension record

person = pd.DataFrame([{

"PersonID": person\_id,

"FirstName": first\_name,

"LastName": last\_name,

"BirthDateUTC": birth\_utc.strftime("%Y-%m-%d %H:%M:%S")

}]).set\_index("PersonID")

# Store Person Hub and Dimension

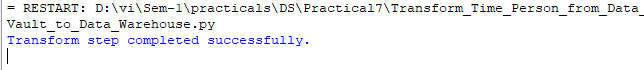
person.to\_sql("Hub-Person-Gunnarsson", conn\_dv, if\_exists="replace")

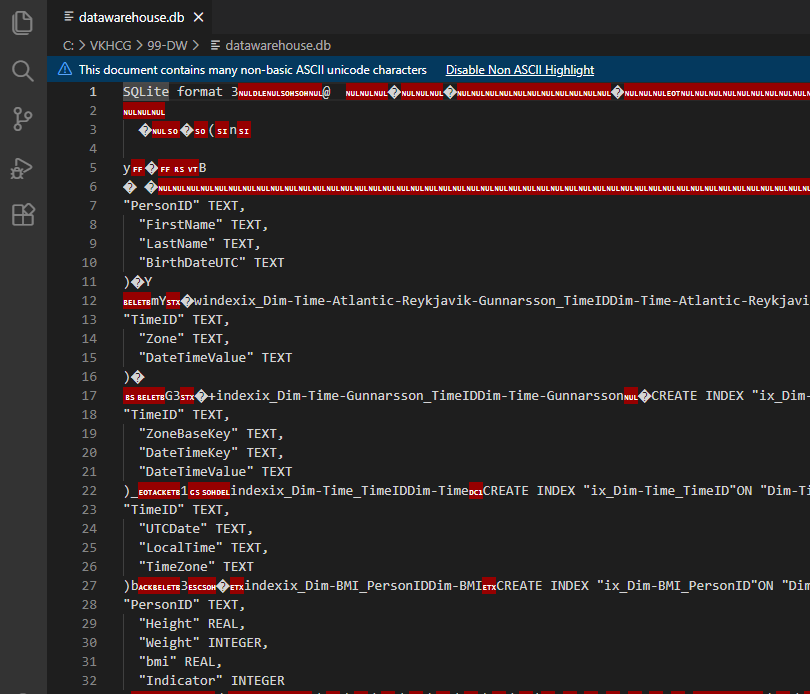
person.to\_sql("Dim-Person-Gunnarsson", conn\_dw, if\_exists="replace")

# Cleanup

conn\_dv.close()

conn\_dw.close()

print("Transform step completed successfully.") ****

****

1. **Build Time Dimension, Person Dimension and Fact table (Sun Model)**

**Purpose:**

* **Build Dim-Time**
* **Build Dim-Person**
* **Build Fact-Person-Born-At-Time**
* **Load into Transform DB and Data Warehouse**

**Code:**

# Build Time Dimension, Person Dimension and Fact table (Sun Model)

import os

import sys

import uuid

import pandas as pd

import sqlite3 as sq

from datetime import datetime

from pytz import timezone

# Base Configuration

BASE = os.path.expanduser('~/VKHCG') if sys.platform == 'linux' else 'C:/VKHCG'

COMPANY = '01-Vermeulen'

# Transform database

TRANSFORM\_DB = BASE + f'/{COMPANY}/04-Transform/SQLite/Vermeulen.db'

os.makedirs(os.path.dirname(TRANSFORM\_DB), exist\_ok=True)

# Data Warehouse database

DW\_DB = BASE + '/99-DW/datawarehouse.db'

os.makedirs(BASE + '/99-DW', exist\_ok=True)

# Database connections

conn\_transform = sq.connect(TRANSFORM\_DB)

conn\_dw = sq.connect(DW\_DB)

# DIMENSION : TIME

birth\_zone = 'Atlantic/Reykjavik'

birth\_utc = datetime(1960, 12, 20, 10, 15, 0, tzinfo=timezone('UTC'))

birth\_local = birth\_utc.astimezone(timezone(birth\_zone))

time\_id = str(uuid.uuid4())

dim\_time = pd.DataFrame([{

"TimeID": time\_id,

"UTCDate": birth\_utc.strftime("%Y-%m-%d %H:%M:%S"),

"LocalTime": birth\_local.strftime("%Y-%m-%d %H:%M:%S"),

"TimeZone": birth\_zone

}]).set\_index("TimeID")

dim\_time.to\_sql("Dim-Time", conn\_transform, if\_exists="replace")

dim\_time.to\_sql("Dim-Time", conn\_dw, if\_exists="replace")

# DIMENSION : PERSON

person\_id = str(uuid.uuid4())

dim\_person = pd.DataFrame([{

"PersonID": person\_id,

"FirstName": "Guðmundur",

"LastName": "Gunnarsson",

"BirthDateUTC": birth\_utc.strftime("%Y-%m-%d %H:%M:%S")

}]).set\_index("PersonID")

dim\_person.to\_sql("Dim-Person", conn\_transform, if\_exists="replace")

dim\_person.to\_sql("Dim-Person", conn\_dw, if\_exists="replace")

# FACT : PERSON BORN AT TIME

fact\_id = str(uuid.uuid4())

fact\_person\_time = pd.DataFrame([{

"FactID": fact\_id,

"PersonID": person\_id,

"TimeID": time\_id

}]).set\_index("FactID")

fact\_person\_time.to\_sql("Fact-Person-Born-At-Time", conn\_transform, if\_exists="replace")

fact\_person\_time.to\_sql("Fact-Person-Born-At-Time", conn\_dw, if\_exists="replace")

# Cleanup

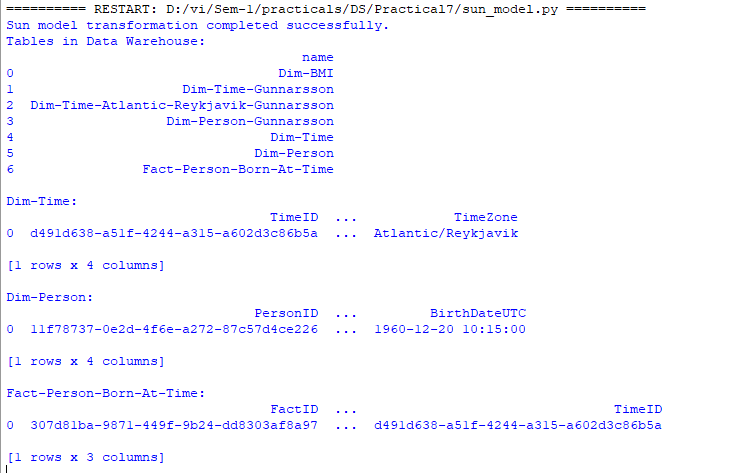
conn\_transform.close()

conn\_dw.close()

print("Sun model transformation completed successfully.")

**Output:**

**Tables created at C:/VKHCG/99-DW/datawarehouse.db**

****

1. **Data Vault To Data Warehouse.**

**Code:**

import os, sys, uuid, pandas as pd, sqlite3 as sq

from datetime import datetime

from pytz import timezone

# Paths

TRANSFORM\_DB = 'C:/VKHCG/04-Transform/SQLite/Vermeulen.db'

DV\_DB = 'C:/VKHCG/88-DV/datavault.db'

DW\_DB = 'C:/VKHCG/99-DW/datawarehouse.db'

os.makedirs(os.path.dirname(TRANSFORM\_DB), exist\_ok=True)

os.makedirs(os.path.dirname(DW\_DB), exist\_ok=True)

os.makedirs(os.path.dirname(DV\_DB), exist\_ok=True)

# Connections

conn\_transform = sq.connect(TRANSFORM\_DB)

conn\_dw = sq.connect(DW\_DB)

conn\_dv = sq.connect(DV\_DB)

#Create Data Vault tables if missing

conn\_dv.execute("""

CREATE TABLE IF NOT EXISTS [Hub-Time] (

DateTimeValue TEXT

);

""")

conn\_dv.execute("""

CREATE TABLE IF NOT EXISTS [Hub-Person] (

FirstName TEXT,

SecondName TEXT,

LastName TEXT,

BirthDateKey TEXT

);

""")

# Insert sample data if tables are empty

if conn\_dv.execute("SELECT COUNT(\*) FROM [Hub-Time];").fetchone()[0] == 0:

conn\_dv.execute("INSERT INTO [Hub-Time] VALUES ('1960-12-20 10:15:00');")

if conn\_dv.execute("SELECT COUNT(\*) FROM [Hub-Person];").fetchone()[0] == 0:

conn\_dv.execute("INSERT INTO [Hub-Person] VALUES ('Guðmundur','','Gunnarsson','1960-12-20 10:15:00');")

conn\_dv.commit()

#Build Time Dimension

time\_raw = pd.read\_sql\_query("SELECT DateTimeValue FROM [Hub-Time];", conn\_dv)

zones = ['Atlantic/Reykjavik', 'Europe/London', 'UTC']

time\_rows = []

for \_, row in time\_raw.iterrows():

utc\_time = datetime.strptime(row["DateTimeValue"], "%Y-%m-%d %H:%M:%S").replace(tzinfo=timezone("UTC"))

for z in zones:

time\_rows.append({

"TimeID": str(uuid.uuid4()),

"UTCDate": utc\_time.strftime("%Y-%m-%d %H:%M:%S"),

"LocalTime": utc\_time.astimezone(timezone(z)).strftime("%Y-%m-%d %H:%M:%S"),

"TimeZone": z

})

dim\_time = pd.DataFrame(time\_rows).set\_index("TimeID")

dim\_time.to\_sql("Dim-Time", conn\_transform, if\_exists="replace")

dim\_time.to\_sql("Dim-Time", conn\_dw, if\_exists="replace")

# Build Person Dimension

person\_raw = pd.read\_sql\_query("SELECT \* FROM [Hub-Person];", conn\_dv)

person\_rows = []

for \_, row in person\_raw.iterrows():

person\_rows.append({

"PersonID": str(uuid.uuid4()),

"FirstName": row["FirstName"],

"SecondName": row["SecondName"] if pd.notna(row["SecondName"]) else "",

"LastName": row["LastName"],

"Zone": "UTC",

"BirthDate": row["BirthDateKey"]

})

dim\_person = pd.DataFrame(person\_rows).set\_index("PersonID")

dim\_person.to\_sql("Dim-Person", conn\_transform, if\_exists="replace")

dim\_person.to\_sql("Dim-Person", conn\_dw, if\_exists="replace")

# Close connections

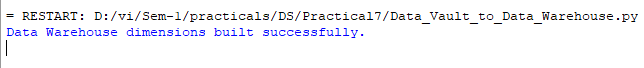
conn\_transform.close()

conn\_dw.close()

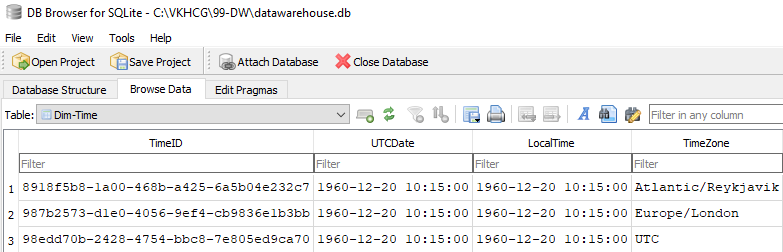
conn\_dv.close()

print("Data Warehouse dimensions built successfully.")

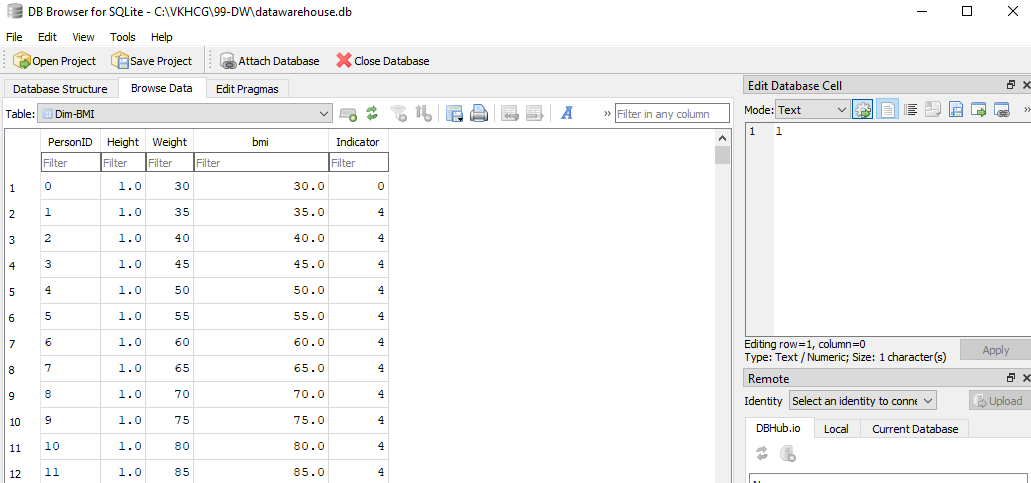
**Output:**

****

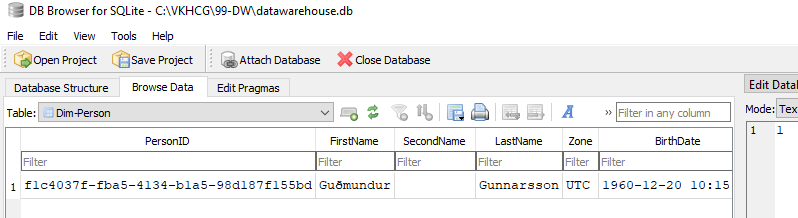
**Dim-Time**

****

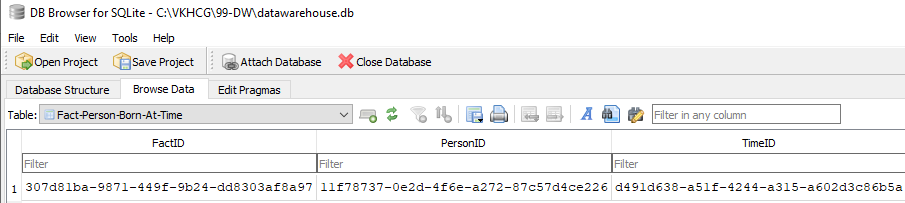
**Dim-BMI**

****

**Dim-Person**

****

**FactTable**

****

|  |  |  |
| --- | --- | --- |
| K.M.S.P.Mandal’s  Sant Rawool Maharaj Mahavidyalaya, Kudal    **Department of Masters of Science**  **Information Technology** | Date:    Roll no: 17    Expt No: 08 | Signature |
| **Title: Organizing Data** | | |

**The aim of this practical is to demonstrate different ways of organizing and retrieving data in a Data Warehouse, including horizontal, vertical, island-style, and secure vault organization, using SQL queries and Pandas in Python.**

**Description**

The program begins by importing the sqlite3 and pandas libraries. A connection to the SQLite database sample\_dw.db is established using sq.connect(). This database contains the Dim\_BMI table, which holds person-level health and demographic data.

**Horizontal Organization**  
 Horizontal organization involves selecting specific rows based on conditions, while including all columns. In the code, rows are filtered for individuals with Indicator = 1, Age between 20 and 40, and BMI >= 23. This approach is useful for retrieving a subset of data that meets particular criteria while keeping the full record attributes intact.

**Vertical Organization**  
 Vertical organization focuses on selecting specific columns (attributes) for all rows. The query extracts PersonID, Age, BMI, and Category for the entire dataset. This method is useful when only certain attributes are needed for analysis, reducing data volume and improving query efficiency.

**Island-Style Organization**  
 Island-style organization combines both row and column filtering with additional conditions. The query selects PersonID, Gender, Age, BMI, Category, and City for individuals meeting multiple conditions, excluding those from Delhi and ordering the results by BMI in descending order. This style is useful for focused analysis on a particular “island” of data that meets complex criteria.

**Secure Vault Organization**  
 Secure vault organization restricts access to sensitive data by masking or redacting certain attributes. In the code, the City column is replaced with 'REDACTED' for rows where Indicator = 1, while other non-sensitive columns are displayed. This approach ensures data privacy and complies with security and confidentiality requirements while allowing analysis on non-sensitive attributes.

**Usage of Functions**

**sq.connect()** establishes a connection to the SQLite database.  
**pd.read\_sql()** executes SQL queries on the database and loads the results into a Pandas DataFrame.  
**print()** displays the retrieved data in the console.  
SQL queries are used with SELECT, WHERE, AND, ORDER BY, and aliasing (AS) to filter, organize, and secure the data.

**Organizing Data**

**Code: organize\_all\_types.py**

import sqlite3 as sq

import pandas as pd

# Database connection

conn = sq.connect("sample\_dw.db")

# Horizontal Organization

# Select specific rows based on conditions (all columns)

horizontal\_df = pd.read\_sql("SELECT \* FROM Dim\_BMI WHERE Indicator = 1 AND Age BETWEEN 20 AND 40 AND BMI >= 23;", conn)

print("\nHORIZONTAL ORGANIZATION")

print(horizontal\_df)

# Vertical Organization

# Select specific columns (attributes) for all rows

vertical\_df = pd.read\_sql("SELECT PersonID, Age, BMI, Category FROM Dim\_BMI;", conn)

print("\nVERTICAL ORGANIZATION")

print(vertical\_df)

# Island-Style Organization

# Select specific rows and columns using multiple conditions

island\_df = pd.read\_sql("SELECT PersonID, Gender, Age, BMI, Category, City FROM Dim\_BMI WHERE Indicator = 1 AND Age BETWEEN 20 AND 40 AND BMI >= 23 AND City <> 'Delhi' ORDER BY BMI DESC;", conn)

print("\nISLAND STYLE ORGANIZATION")

print(island\_df)

#Secure Vault Organization

# Restrict and mask sensitive attributes for secure access

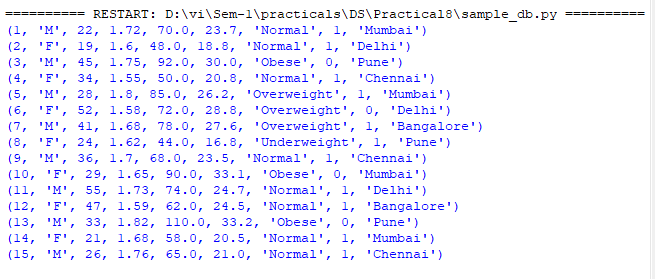
secure\_vault\_df = pd.read\_sql("SELECT PersonID, Age, BMI, Category, Indicator, 'REDACTED' AS City FROM Dim\_BMI WHERE Indicator = 1;", conn)

print("\nSECURE VAULT ORGANIZATION")

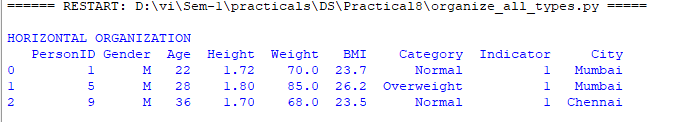
print(secure\_vault\_df)

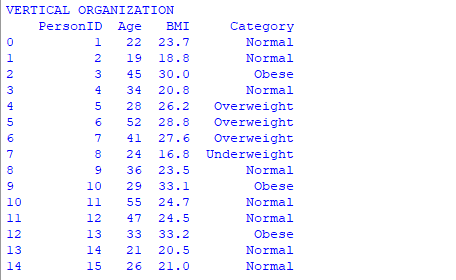
conn.close()

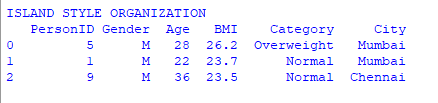
**Sample Data**

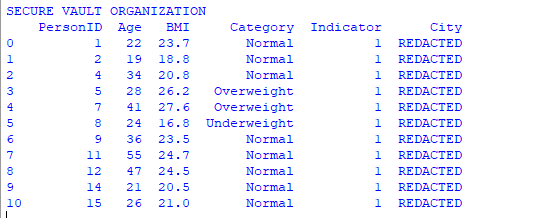


**Output**

****

****

****

****

|  |  |  |
| --- | --- | --- |
| K. M. S. P. Mandal’s  Sant Rawool Maharaj Mahavidyalaya, Kudal    **Department of Masters of Science**  **Information Technology** | Date:    Roll no: 17    Expt No: 09 | Signature |
| **Title: Generating Data** | | |

**The aim of this code is to generate two-dimensional representations of the high-dimensional handwritten digits dataset using multiple dimensionality reduction and embedding techniques, and to visualize the transformed data to understand how each algorithm organizes and preserves the inherent structure and clusters of the digits.**

**Embedding Techniques**

1. **Random Projection**  
   A linear technique that projects high-dimensional data into a lower-dimensional space using a random matrix while approximately preserving distances.  
   Output: 2D coordinates for each digit, showing approximate clustering. Execution time is printed.
2. **PCA (TruncatedSVD)**  
   Principal Component Analysis reduces dimensionality by projecting data onto directions of maximum variance. TruncatedSVD is used for sparse datasets.  
   Output: 2D scatter plot of digits, clusters aligned along the directions of highest variance.
3. **LDA (Linear Discriminant Analysis)**  
   A supervised linear technique that maximizes class separability in lower dimensions. Here, labels (y) are used.  
   Output: 2D projection where digits of the same class are separated more distinctly. Execution time is printed.
4. **Isomap**  
   A non-linear manifold learning method that preserves geodesic distances on a manifold. Uses a nearest-neighbor graph.  
   Output: 2D embedding showing how local distances are preserved, clustering digits based on intrinsic geometry.
5. **LLE (Locally Linear Embedding – Standard)**  
   Non-linear method that preserves local linear relationships in the data.  
   Output: 2D scatter plot preserving local neighborhoods of digits.
6. **Modified LLE**  
   A variant of LLE that improves robustness against noise in local neighborhoods.  
   Output: Similar 2D embedding with better local structure compared to standard LLE.
7. **Hessian LLE**  
   Uses Hessian eigenmaps to improve unfolding of the manifold in highly curved regions.  
   Output: 2D embedding showing local structures and manifold geometry in more detail.
8. **LTSA (Local Tangent Space Alignment**)  
   Aligns local tangent spaces of the manifold to compute a global low-dimensional embedding.  
   Output: 2D projection preserving local geometry while reducing dimensionality.
9. **MDS (Multidimensional Scaling)**  
   A distance-preserving technique that places points in lower dimensions such that pairwise distances approximate the original.  
   Output: 2D plot preserving overall distances between digits.
10. **Random Forest Embedding**  
    Uses an ensemble of decision trees to transform the data into a new space based on leaf node activations, then applies TruncatedSVD.  
    Output: 2D representation highlighting class separability learned by the trees.
11. **Spectral Embedding**  
    Uses the eigenvectors of the graph Laplacian constructed from data to embed points, preserving connectivity.  
    Output: 2D plot that emphasizes clusters based on similarity in the graph structure.
12. **t-SNE (t-distributed Stochastic Neighbor Embedding)**  
    Non-linear method that preserves local similarities by modeling high-dimensional pairwise similarities in lower dimensions.  
    Output: Highly interpretable 2D clusters where similar digits are grouped tightly. Often used for visualizing high-dimensional datasets.

**Code: report\_hillman**

from time import time

import numpy as np

import matplotlib.pyplot as plt

from matplotlib import offsetbox

from sklearn import manifold, datasets, decomposition, ensemble, discriminant\_analysis, random\_projection

digits = datasets.load\_digits(n\_class=6)

X, y = digits.data, digits.target

n\_neighbors = 30

def plot\_embedding(X, title=None):

X = (X - X.min(0)) / (X.max(0) - X.min(0))

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

ax = plt.subplot(111)

for i in range(X.shape[0]):

plt.text(X[i,0], X[i,1], str(y[i]), color=plt.cm.Set1(y[i]/10.), fontdict={'weight':'bold','size':9})

if hasattr(offsetbox,'AnnotationBbox'):

shown = np.array([[1.,1.]])

for i in range(X.shape[0]):

if np.min(np.sum((X[i]-shown)\*\*2,1))<4e-3: continue

shown = np.r\_[shown,[X[i]]]

ax.add\_artist(offsetbox.AnnotationBbox(offsetbox.OffsetImage(digits.images[i], cmap=plt.cm.gray\_r), X[i]))

plt.xticks([]); plt.yticks([]);

if title: plt.title(title)

# Show grid of digits

n\_img = 20; img = np.zeros((10\*n\_img,10\*n\_img))

for i in range(n\_img):

for j in range(n\_img):

img[10\*i+1:10\*i+9, 10\*j+1:10\*j+9] = X[i\*n\_img+j].reshape(8,8)

plt.figure(figsize=(10,10)); plt.imshow(img, cmap=plt.cm.binary); plt.title('Sample digits'); plt.xticks([]); plt.yticks([])

# Embeddings

for name, emb in [

("Random Projection", random\_projection.SparseRandomProjection(n\_components=2, random\_state=42)),

("PCA", decomposition.TruncatedSVD(n\_components=2)),

("LDA", discriminant\_analysis.LinearDiscriminantAnalysis(n\_components=2)),

("Isomap", manifold.Isomap(n\_neighbors=n\_neighbors, n\_components=2)),

("LLE", manifold.LocallyLinearEmbedding(n\_neighbors=n\_neighbors, n\_components=2, method='standard')),

("Modified LLE", manifold.LocallyLinearEmbedding(n\_neighbors=n\_neighbors, n\_components=2, method='modified')),

("Hessian LLE", manifold.LocallyLinearEmbedding(n\_neighbors=n\_neighbors, n\_components=2, method='hessian')),

("LTSA", manifold.LocallyLinearEmbedding(n\_neighbors=n\_neighbors, n\_components=2, method='ltsa')),

("MDS", manifold.MDS(n\_components=2, n\_init=1, max\_iter=100, init='random')),

("Random Forest", decomposition.TruncatedSVD(n\_components=2)),

("Spectral", manifold.SpectralEmbedding(n\_components=2, random\_state=0)),

("t-SNE", manifold.TSNE(n\_components=2, init='pca', random\_state=0))

]:

print(name); t0=time()

if name=="LDA": X2 = X.copy(); X2.flat[::X.shape[1]+1]+=0.01; X\_emb = emb.fit\_transform(X2,y)

elif name=="Random Forest":

X\_rf = ensemble.RandomTreesEmbedding(n\_estimators=200, random\_state=0, max\_depth=5).fit\_transform(X)

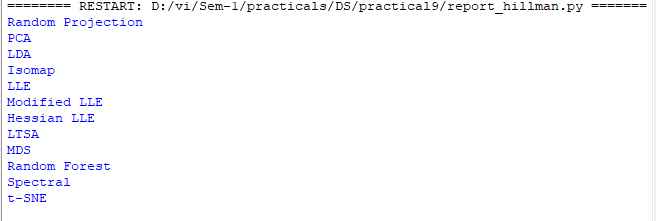
X\_emb = emb.fit\_transform(X\_rf)

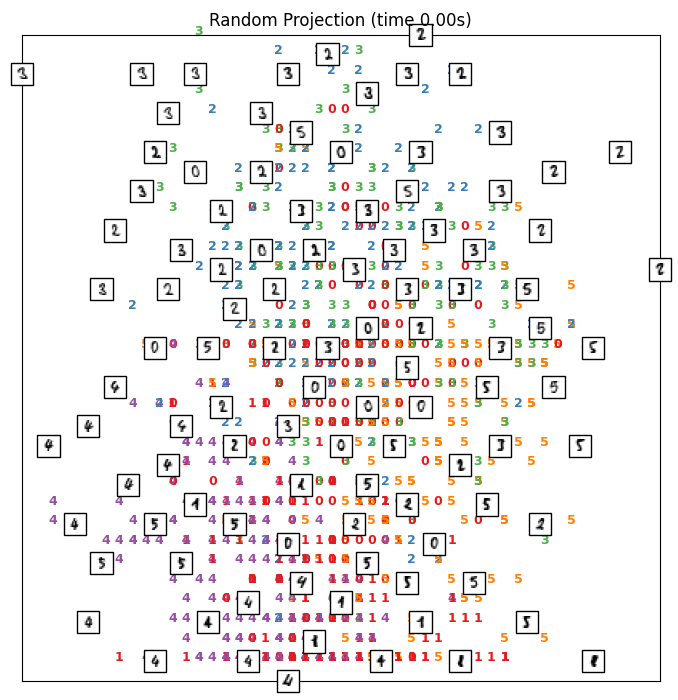
else: X\_emb = emb.fit\_transform(X)

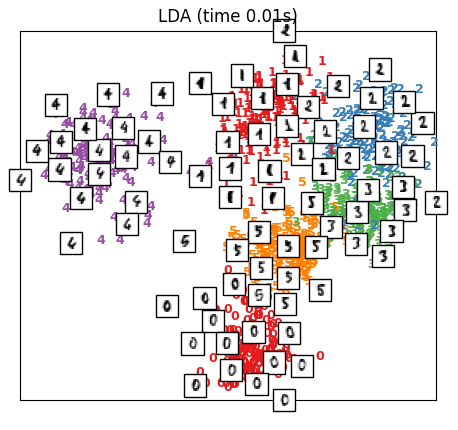
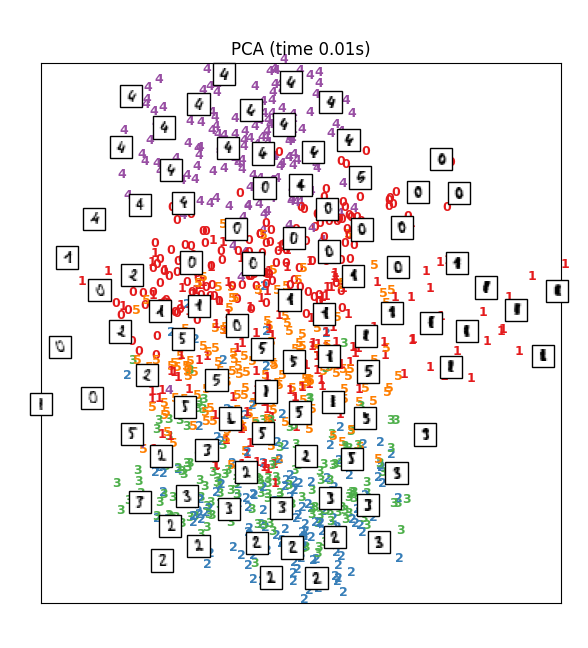
plot\_embedding(X\_emb, f"{name} (time {time()-t0:.2f}s)")

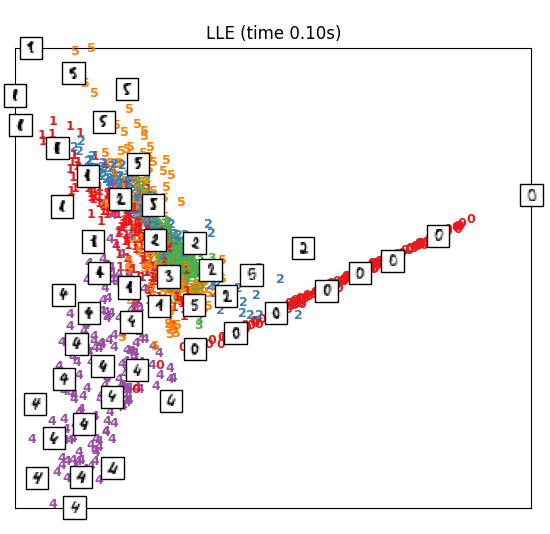
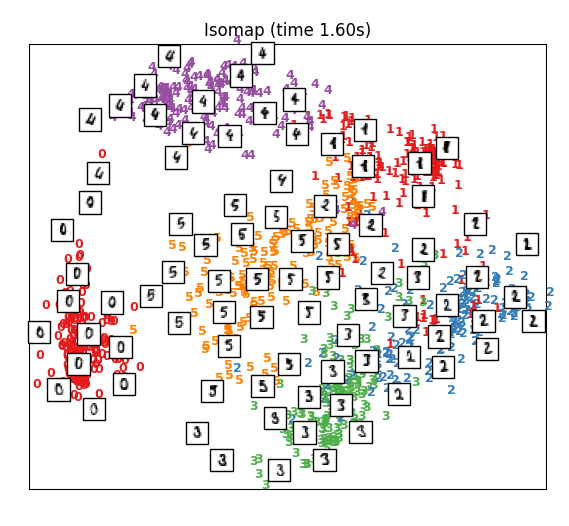
plt.show()

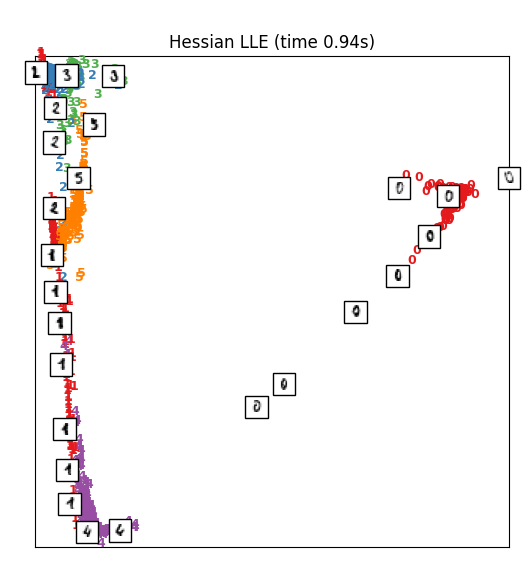
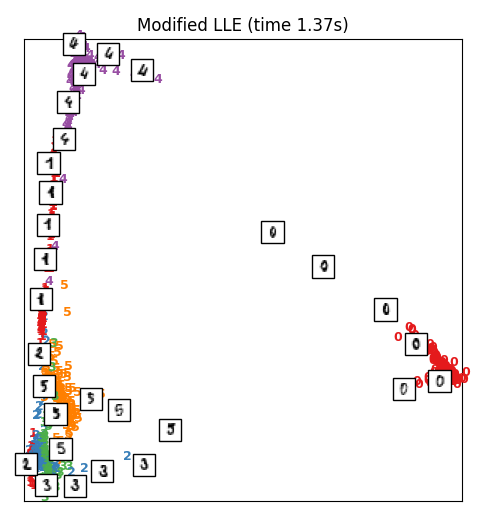
**Output:**

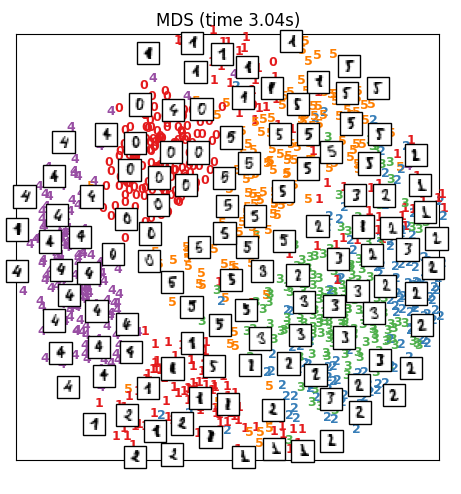
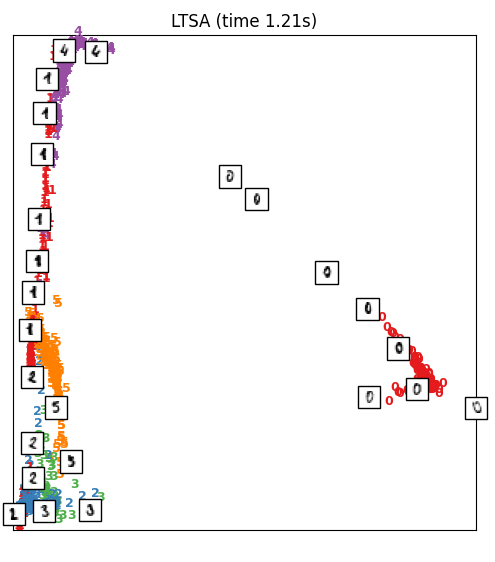
****

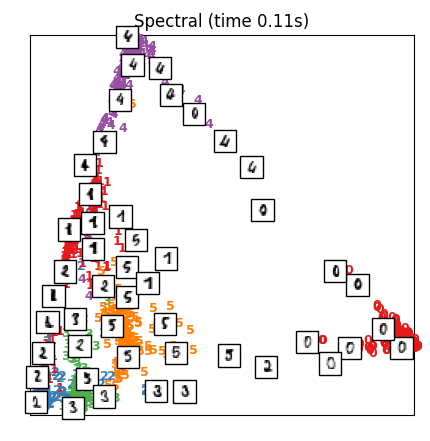
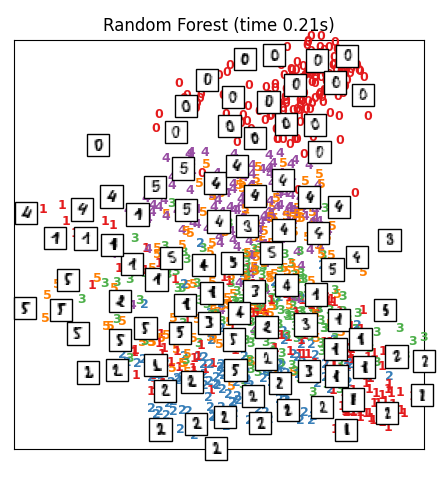
****

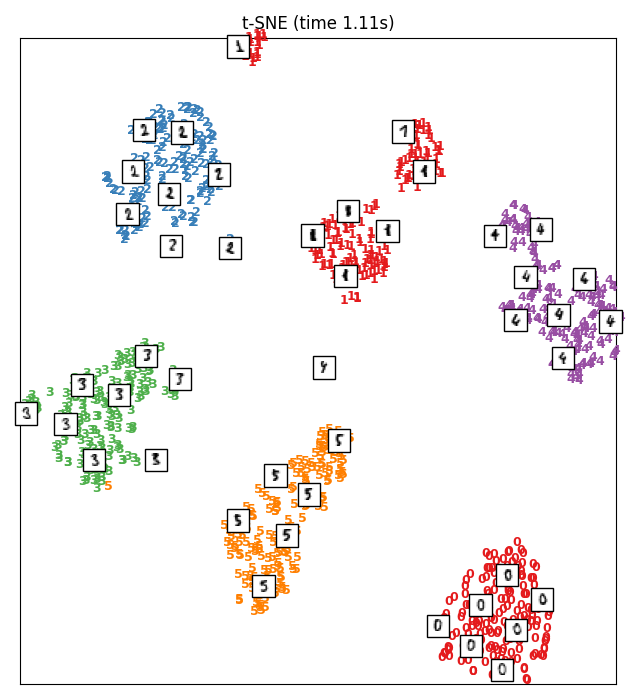
****

****

****

****

****

****

|  |  |  |
| --- | --- | --- |
| K. M. S. P. Mandal’s  Sant Rawool Maharaj Mahavidyalaya, Kudal    **Department of Masters of Science**  **Information Technology** | Date:    Roll no: 17    Expt No: 10 | Signature |
| **Title: Perform data visualization Using PowerBI** | | |

**Data Visualization in Power BI: Key Considerations**

Creating effective data visualizations in Power BI requires more than just dropping data onto charts. A well-designed visualization helps users quickly understand trends, patterns, and insights. Here are the main points to consider:

**1. Data Preparation**  
 Before creating visuals, ensure your data is clean and well-structured. Use Power Query to remove duplicates, handle missing values, and format data types correctly. Create calculated columns or measures using DAX for key metrics if needed.

**2. Choosing the Right Visuals**  
 Select visuals that match the type of data and the story you want to tell:

* Bar/Column charts: Compare categories or groups.
* Line charts: Show trends over time.
* Pie/Donut charts: Show proportions, but use sparingly.
* Scatter plots: Display relationships between variables.
* Maps: Visualize geographic data.
* Tables/Matrix: Present detailed data.

**3. Dashboard Design**  
 Keep your dashboard clean and simple. Prioritize the most important visuals at the top or center, and group related visuals together. Use consistent colors for similar categories to make it easier for users to interpret the data.

**4. Interactivity**  
 Power BI allows users to interact with the dashboard. Include slicers, cross-filtering, and drill-throughs to let users explore data in depth. This makes the report more dynamic and insightful.

**5. Labels and Tooltips**  
 Always provide clear titles, axis labels, and legends. Tooltips can give additional information when hovering over visuals, enhancing understanding without cluttering the dashboard.

**6. Performance**  
 Large datasets can slow down dashboards. Use aggregations, filters, and data reduction techniques like Top N filters to improve performance and maintain responsiveness.

**7. Storytelling and Insight**  
 Visualizations should tell a story. Highlight patterns, trends, or anomalies, and make it easy for users to grasp key insights at a glance. Avoid unnecessary clutter that can distract from the main message.

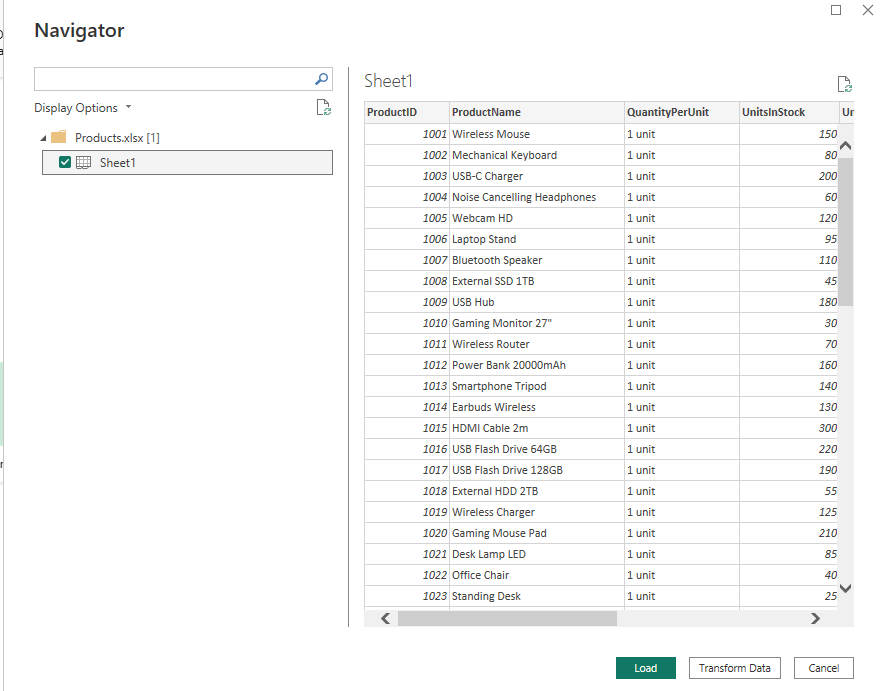
**8. Testing and Sharing**  
 Test your dashboard for interactivity, filter functionality, and clarity. Ensure visuals display correctly across devices and share your report via Power BI Service for collaboration and accessibility.

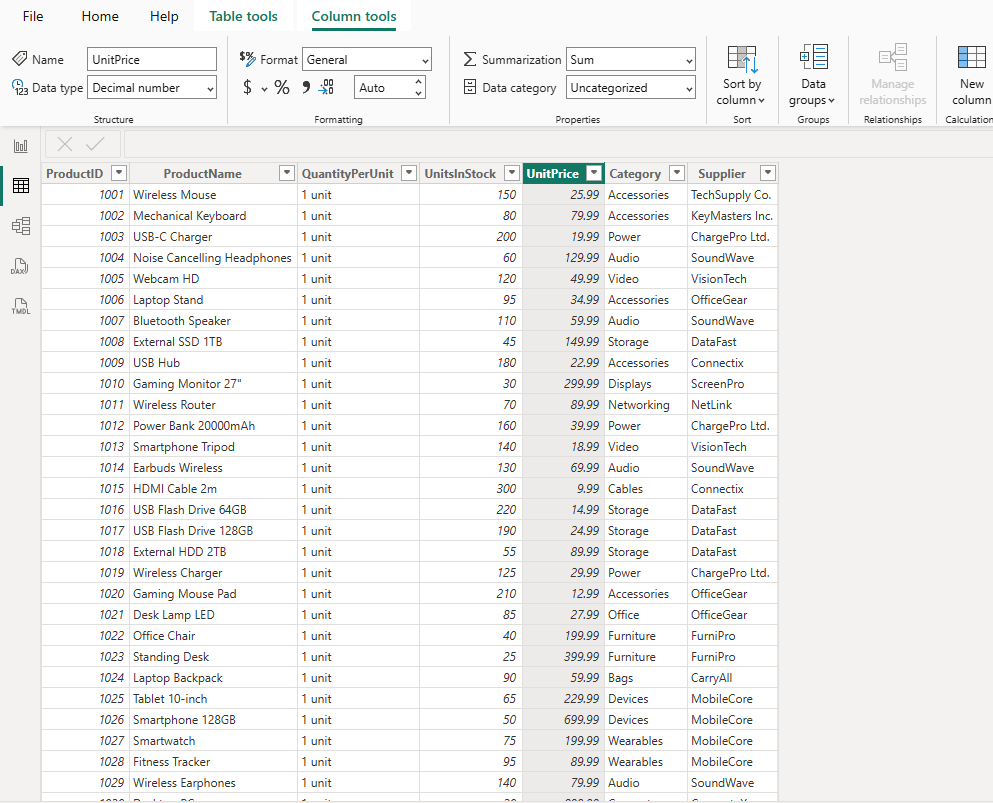
**Step 1: Connect to an Excel workbook**

**1. Launch power BI Desktop**

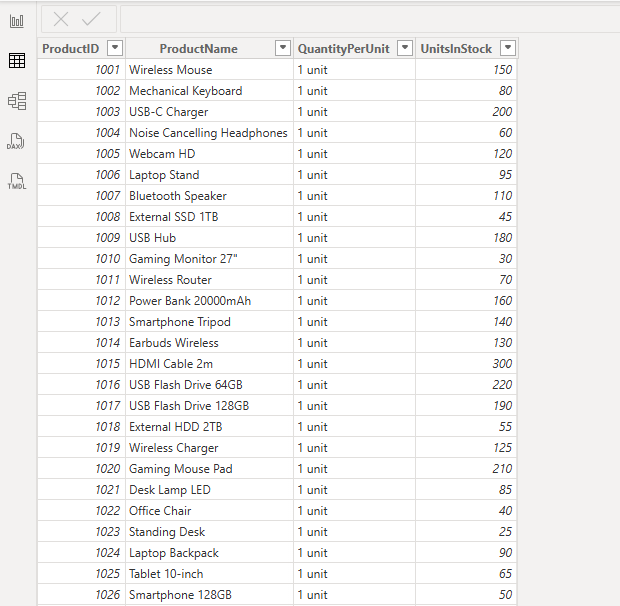
**2. Home -> Get Data -> Excel -> open product.xlsx**

**3. Right click on product -> Edit**

****

****

**4. In Query Editor, select the ProductID, ProductName, QuantityPerUnit, and UnitsInStock columns . Select Remove Columns -> Remove Other Columns from the ribbon, or right-click on a column header and click Remove Other Columns.**

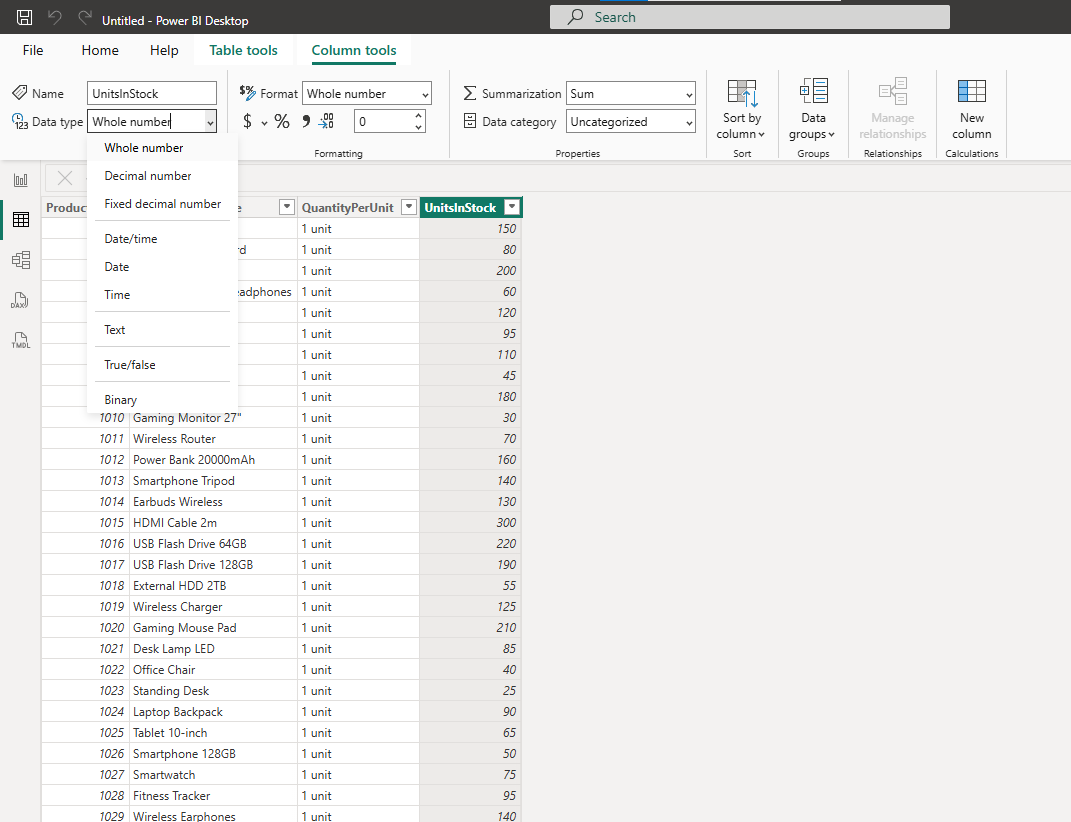
****

**5. Change the data type of the UnitsInStock column**

a. Select the UnitsInStock column.

b. Select the Data Type drop-down button in the Home ribbon.

c. If not already a Whole Number, select Whole Number for data type from the drop down (the Data Type: button also displays the data type for the current selection).



**Step 2:** **Import order data from an OData feed You import data into Power BI Desktop from the sample Northwind OData feed at the following URL, which you can copy (and then paste) in the steps below:** http://services.odata.org/V3/Northwind/Northwind.svc/

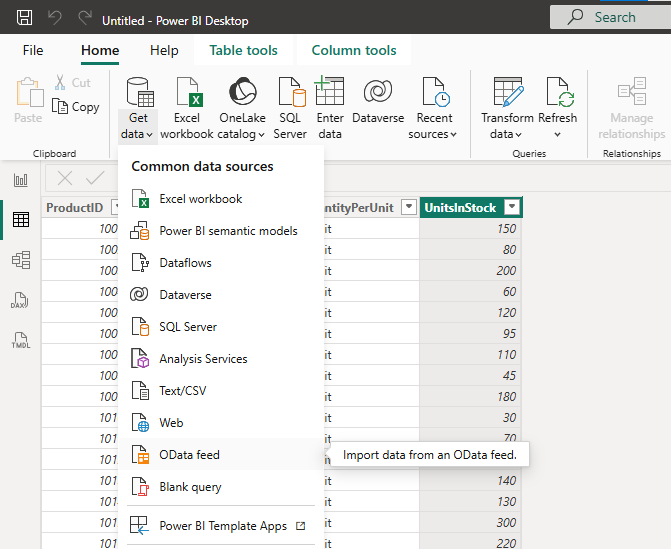
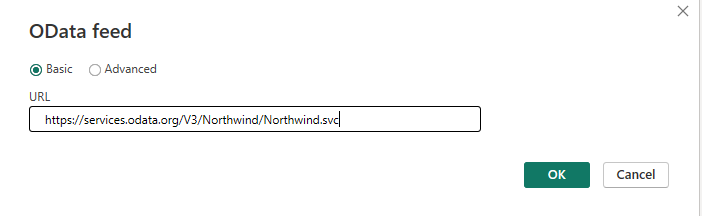
Step 1: Connect to an OData feed

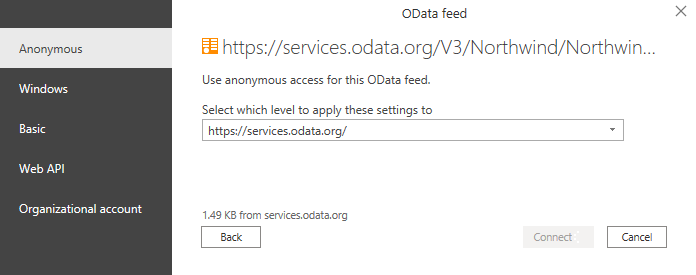
1. From the Home ribbon tab in Query Editor, select Get Data.

2. Browse to the OData Feed data source.

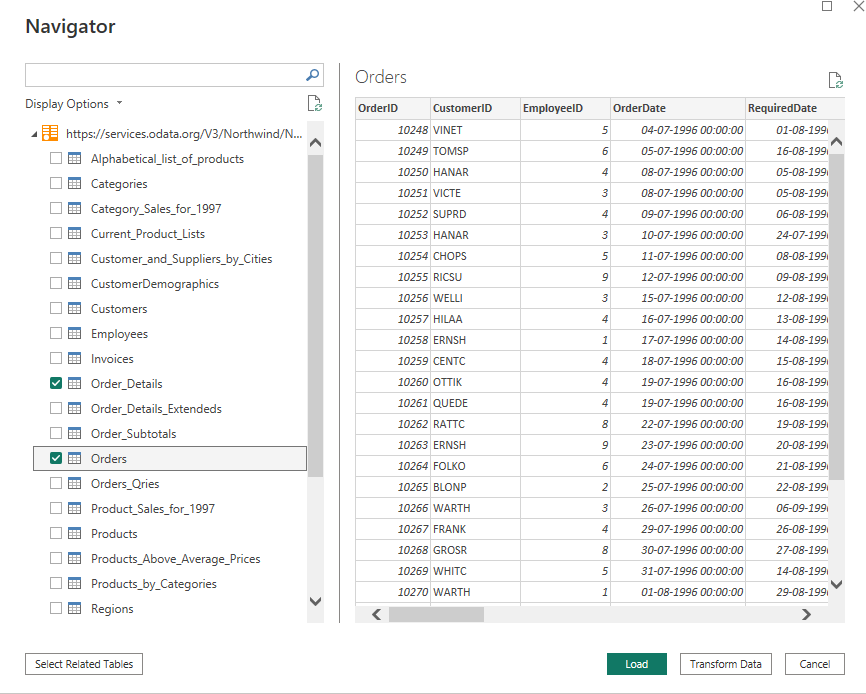
3. In the OData Feed dialog box, paste the URL for the Northwind OData feed.

4. Select OK.



Select Orders and Orders\_details.



**Step 3: Expand the Order\_Details table**

1. In the Query View, scroll to the Order\_Details column.

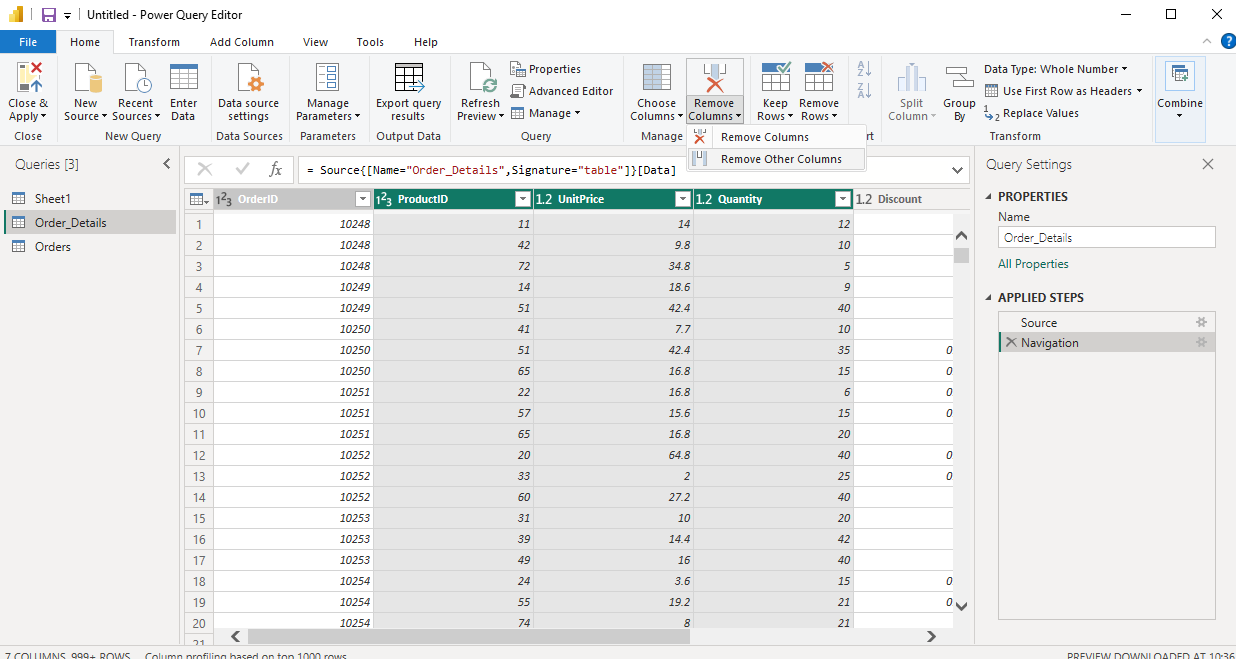
2. In the Order\_Details column, select the expand icon ().

3. In the Expand drop-down:

a. Select (Select All Columns) to clear all columns.

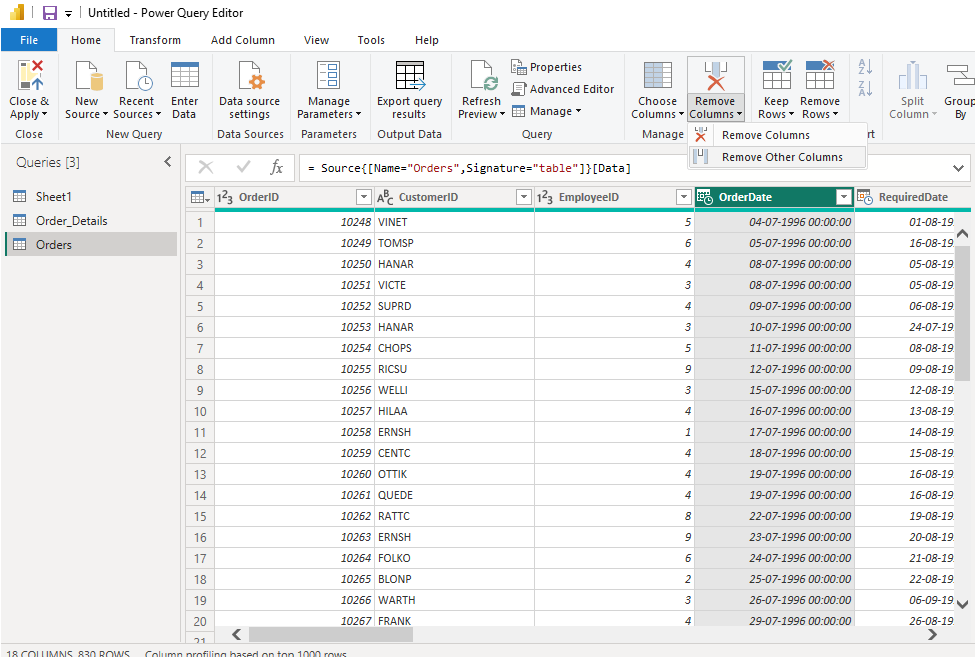
b. Select ProductID, UnitPrice, and Quantity.

c. click OK.



In this step you remove all columns except OrderDate, ShipCity, ShipCountry, from **orders**

and ProductID,.UnitPrice, and Quantity columns. From **order**\_**details**

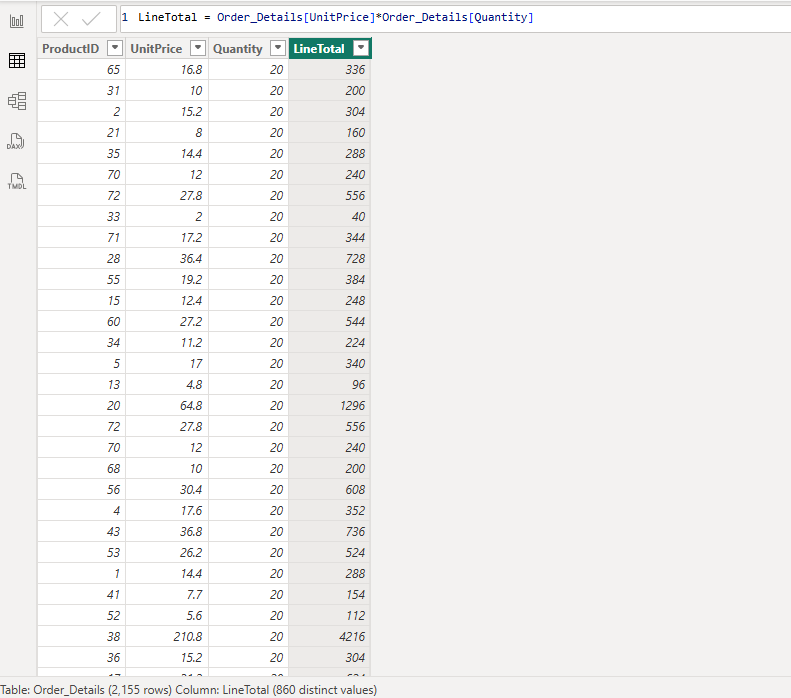


**Step 4: Calculate the line total for each Order\_Details row**

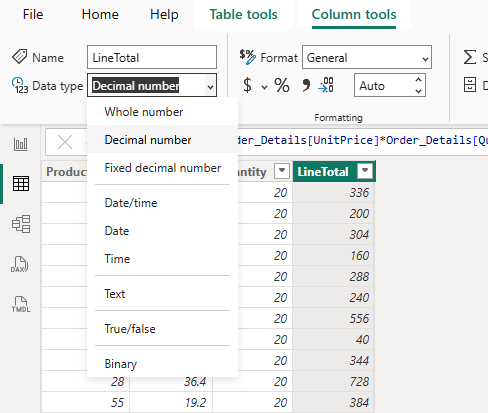
1. In the Add Column ribbon tab -> Add Custom Column.

2. In the Add Custom Column dialog box, in the Custom Column Formula textbox, enter [Order\_Details.UnitPrice] \* [Order\_Details.Quantity]

3. In the New column name textbox, enter LineTotal.

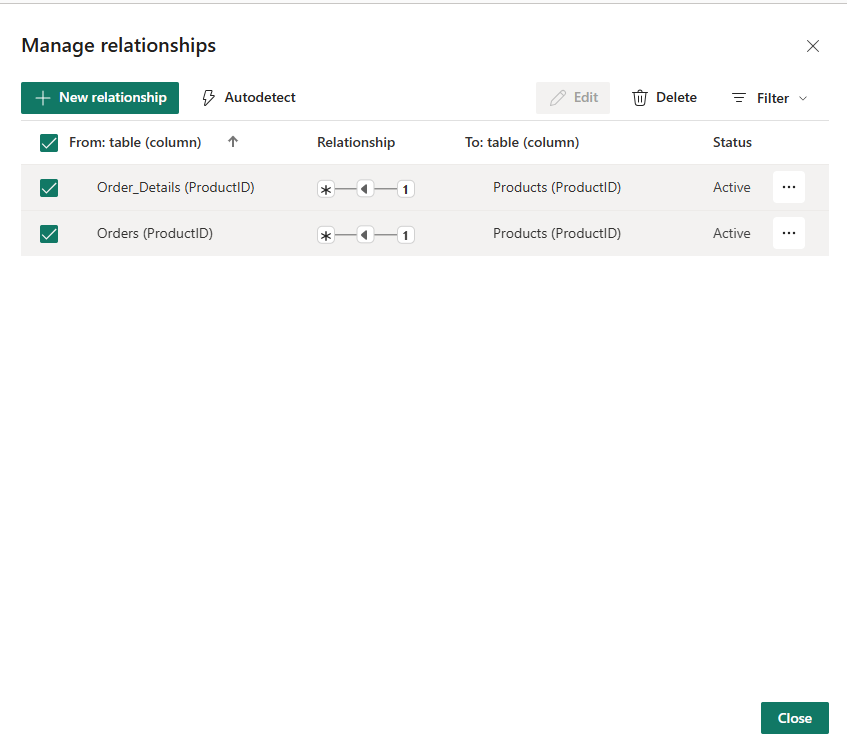


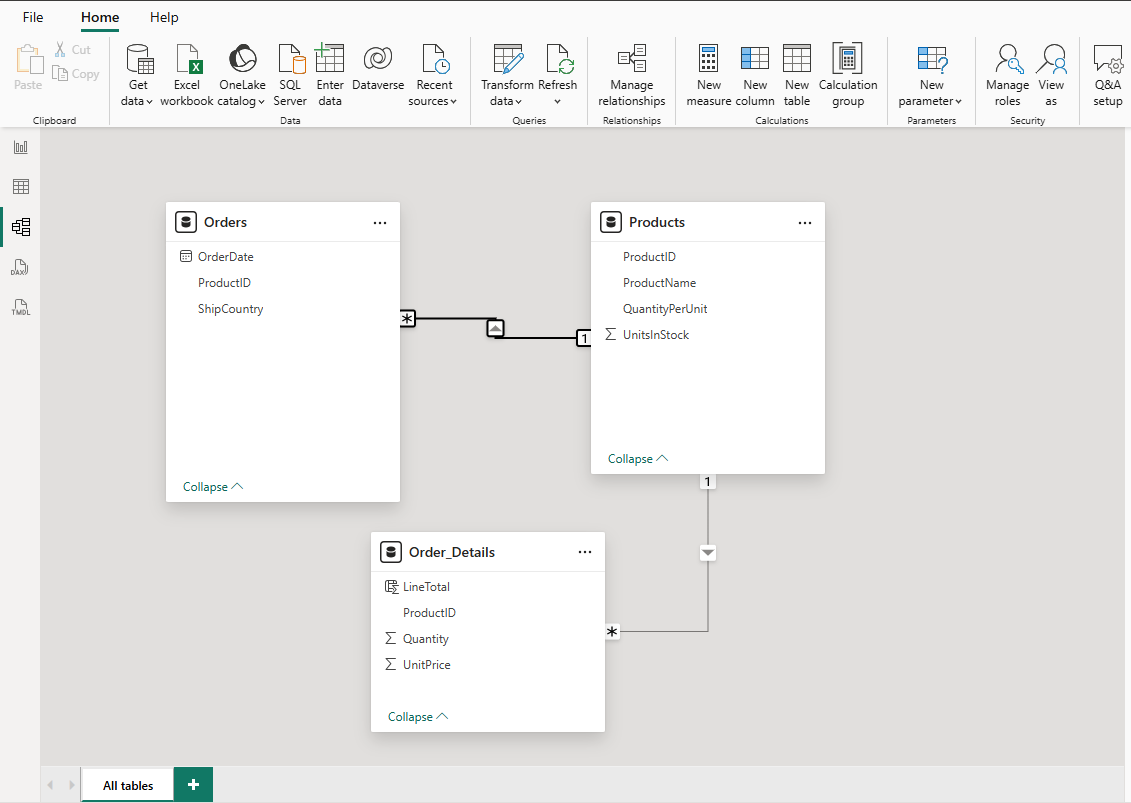
**Step 5: Set the datatype of the LineTotal field Right click the LineTotal column -> Change Type -> Decimal Number**



**Step 6: Combine the Products and Total Sales queries**

**1. Home tab -> Manage Relationships -> New**

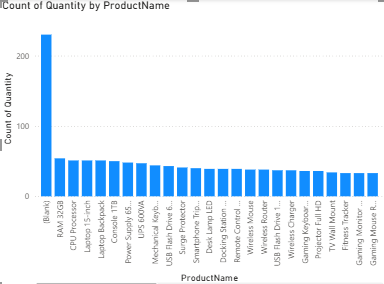
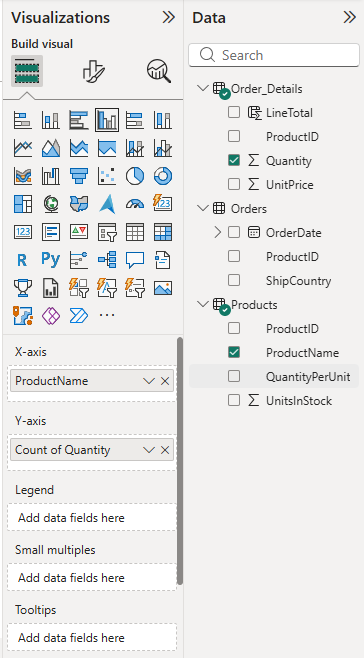




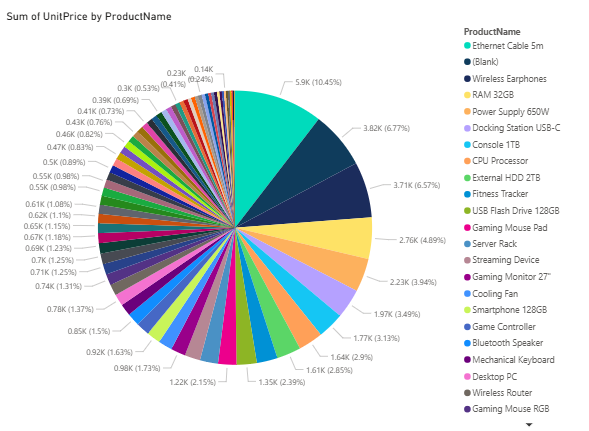
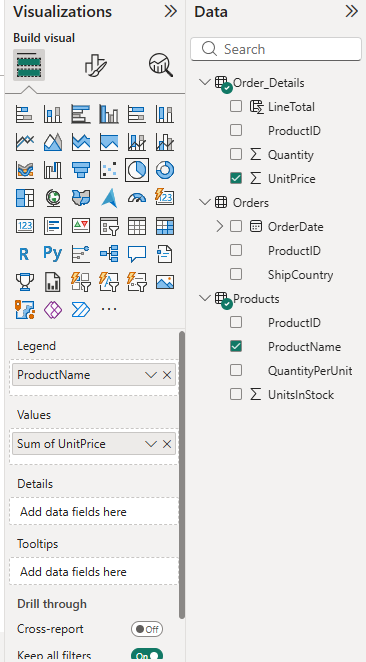
(changed EmployeeID to Product ID in Orders)

**Step 7: Visualization**

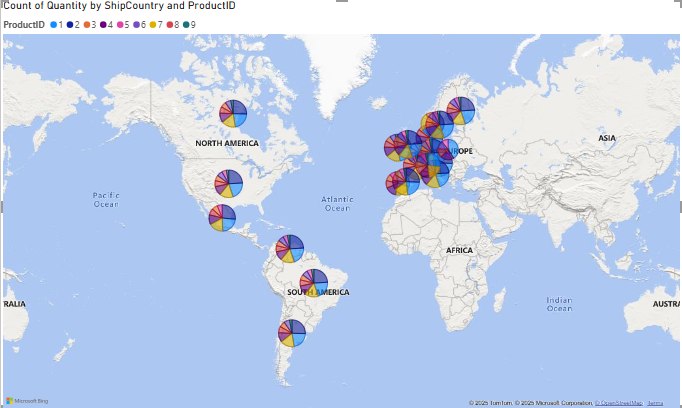
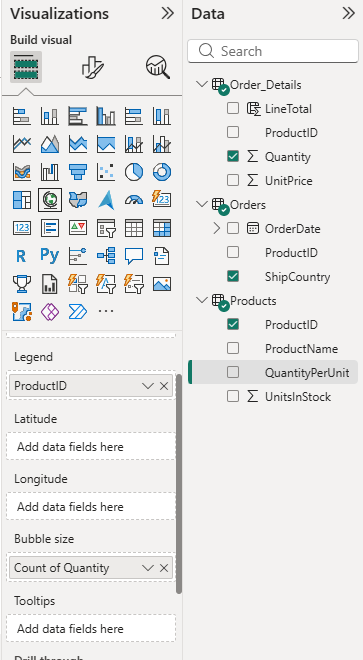
**Count Of Quantity by product name**

 ****

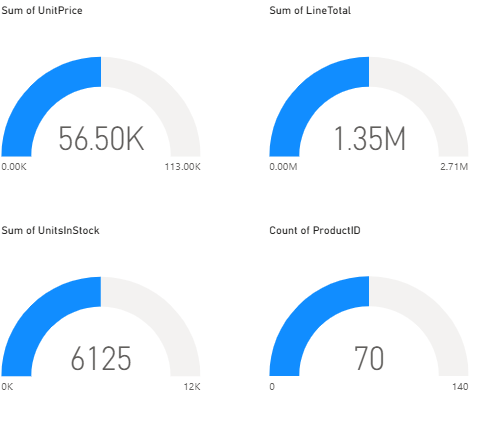
**Sum of Unit Price By Product name**

**** ****

**Count Of Quantity by Shipping Country and ProductID**

**** ****

**Count and sum of columns.**



**Interact with your report visuals to analyze further Final**

