**Practical No: 01**

**Aim: Creating Data Model using Cassandra**.

Create keyspace keyspace1 with replication = { 'class':'SimpleStratergy','replication\_factor': '3' };

Create table dept ( dept\_id int PRIMARY KEY, dept\_name text, dept\_loc text);

Create table emp ( emp\_id int PRIMARY KEY, emp\_name text, dept\_id int,email text, phone text );

Insert Queries for dept table

Insert into dept (dept\_id, dept\_name, dept\_loc) values (1001,'Accounts', 'Mumbai');

Insert into dept (dept\_id, dept\_name, dept\_loc) values (1002,'Marketing', 'Delhi');

Insert into dept (dept\_id, dept\_name, dept\_loc) values (1003, 'HR','Chennai');

Insert Queries for emp table

Insert into emp ( emp\_id, emp\_name, dept\_id, email, phone ) values

(1001, 'ABCD',1001, 'abcd@company.com', '1122334455');

Insert into emp ( emp\_id, emp\_name, dept\_id, email, phone ) values

(1002, 'DEFG',1001, 'defg@company.com', '2233445566');

Insert into emp ( emp\_id, emp\_name, dept\_id, email, phone ) values

(1003, 'GHIJ',1002, 'ghij@company.com', '3344556677');

Insert into emp ( emp\_id, emp\_name, dept\_id, email, phone ) values

(1004, 'JKLM',1002, 'jklm@company.com', '4455667788');

Insert into emp ( emp\_id, emp\_name, dept\_id, email, phone ) values

(1005, 'MNOP',1003, 'mnop@company.com', '5566778899');

Insert into emp ( emp\_id, emp\_name, dept\_id, email, phone ) values

(1006, 'MNOP',1003, 'mnop@company.com', '5566778844');

Select-all Query for emp table

select \* from emp

Select-all Query for dept table

select \* from dept

Update query for dept table where dept\_id=1003 set dept\_name is Human Resource

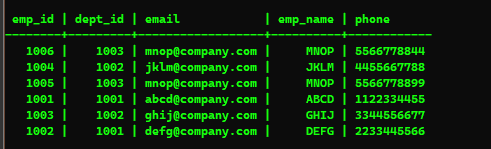
update dept set dept\_name='Human Resource' where dept\_id=1003;

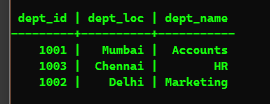
Delete query for emp table where emp\_id=1006 and Select-all from emp

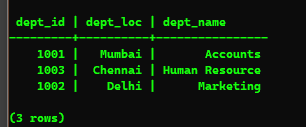
delete from emp where emp\_id=1006;

select \* from emp

**Output:**

****







**Practical No: 02**

**Aim: Write Python / R Program to convert from the following formats to HORUS format**

**2A. Text delimited CSV to HORUS format**

# Utility Start CSV to HORUS =================================

# Standard Tools

# =============================================================

import pandas as pd

# Input Agreement ============================================

sInputFileName = "D:/OneDrive - South Indian Education Society/Online

MSIT Classes/Practicals/Data Science/Practical 2/Country\_Code.csv"

InputData = pd.read\_csv(sInputFileName, encoding="latin-1")

print("Input Data Values ===================================")

print(InputData)

print("=====================================================")

# Processing Rules ===========================================

ProcessData = InputData

# Remove columns ISO-2-Code and ISO-3-CODE

ProcessData.drop("ISO-2-CODE", axis=1, inplace=True)

ProcessData.drop("ISO-3-Code", axis=1, inplace=True)

# Rename Country and ISO-M49

ProcessData.rename(columns={"Country": "CountryName"}, inplace=True)

ProcessData.rename(columns={"ISO-M49": "CountryNumber"}, inplace=True)

# Set new Index

ProcessData.set\_index("CountryNumber", inplace=True)

# Sort data by CurrencyNumber

ProcessData.sort\_values("CountryName", axis=0, ascending=False,

inplace=True)

print("Process Data Values =================================")

print(ProcessData)

print("=====================================================")

# Output Agreement ===========================================

OutputData = ProcessData

sOutputFileName = "D:/OneDrive - South Indian Education Society/Online

MSIT Classes/Practicals/Data Science/Practical 2/HORUS-CSV-

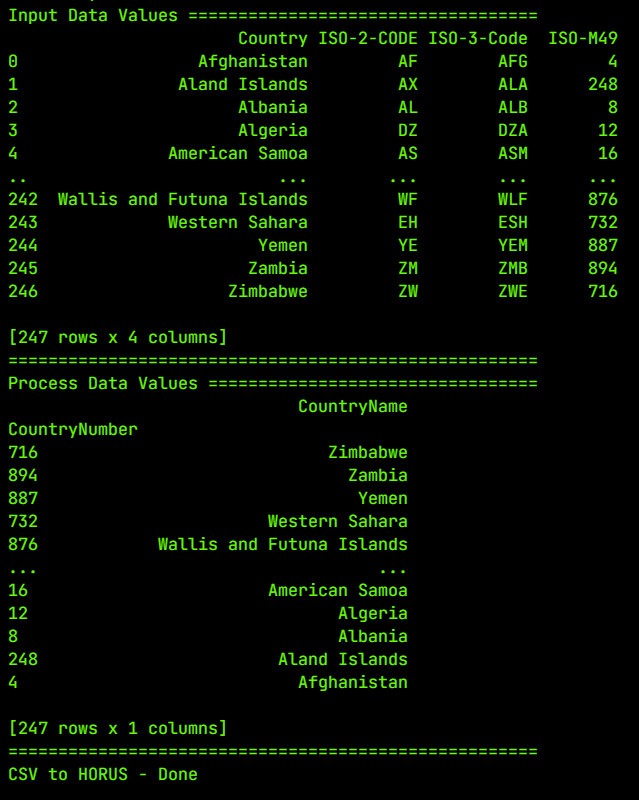
Country.csv"

OutputData.to\_csv(sOutputFileName, index=False)

print("CSV to HORUS - Done")

# Utility done ===============================================

**Output:**

****

**2B. XML to HORUS Format**

# Utility Start XML to HORUS =================================

# Standard Tools

import pandas as pd

import xml.etree.ElementTree as ET

def df2xml(data):

header = data.columns

root = ET.Element("root")

for row in range(data.shape[0]):

entry = ET.SubElement(root, "entry")

for index in range(data.shape[1]):

schild = str(header[index])

child = ET.SubElement(entry, schild)

if str(data[schild][row]) != "nan":

child.text = str(data[schild][row])

else:

child.text = "n/a"

entry.append(child)

result = ET.tostring(root)

return result

def xml2df(xml\_data):

root = ET.XML(xml\_data)

all\_records = []

for i, child in enumerate(root):

record = {}

for subchild in child:

record[subchild.tag] = subchild.text

all\_records.append(record)

return pd.DataFrame(all\_records)

sInputFileName = "D:/VKHCG/05-DS/9999-Data/Country\_Code.xml"

InputData = open(sInputFileName).read()

print("=====================================================")

print("Input Data Values ===================================")

print("=====================================================")

# print(InputData)

print("=====================================================")

# =============================================================

# Processing Rules ===========================================

# =============================================================

ProcessDataXML = InputData

# XML to Data Frame

ProcessData = xml2df(ProcessDataXML)

# Remove columns ISO-2-Code and ISO-3-CODE

ProcessData.drop("ISO-2-CODE", axis=1, inplace=True)

ProcessData.drop("ISO-3-Code", axis=1, inplace=True)

# Rename Country and ISO-M49

ProcessData.rename(columns={"Country": "CountryName"}, inplace=True)

ProcessData.rename(columns={"ISO-M49": "CountryNumber"}, inplace=True)

# Set new Index

ProcessData.set\_index("CountryNumber", inplace=True)

# Sort data by CurrencyNumber

ProcessData.sort\_values("CountryName", axis=0, ascending=False,

inplace=True)

print("=====================================================")

print("Process Data Values =================================")

print("=====================================================")

print(ProcessData)

print("=====================================================")

OutputData = ProcessData

sOutputFileName = "D:/VKHCG/05-DS/9999-Data/HORUS-XML-Country.csv"

OutputData.to\_csv(sOutputFileName, index=False)

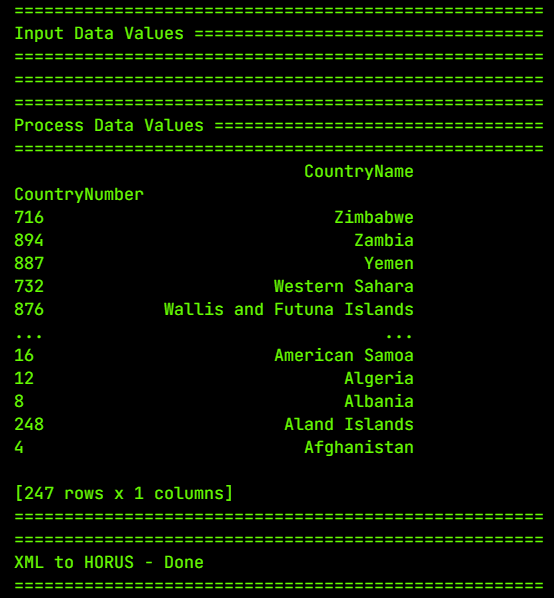
print("=====================================================")

print("XML to HORUS - Done")

print("=====================================================")

# Utility done ===============================================

**Output:**

****

**2C. JSON to HORUS Format**

# Utility Start JSON to HORUS =================================

# Standard Tools

# =============================================================

import pandas as pd

# Input Agreement ============================================

sInputFileName = "D:/VKHCG/05-DS/9999-Data/Country\_Code.json"

InputData = pd.read\_json(sInputFileName, orient="index",

encoding="latin-1")

print("Input Data Values ===================================")

print(InputData)

print("=====================================================")

# Processing Rules ===========================================

ProcessData = InputData

# Remove columns ISO-2-Code and ISO-3-CODE

ProcessData.drop("ISO-2-CODE", axis=1, inplace=True)

ProcessData.drop("ISO-3-Code", axis=1, inplace=True)

# Rename Country and ISO-M49

ProcessData.rename(columns={"Country": "CountryName"}, inplace=True)

ProcessData.rename(columns={"ISO-M49": "CountryNumber"}, inplace=True)

# Set new Index

ProcessData.set\_index("CountryNumber", inplace=True)

# Sort data by CurrencyNumber

ProcessData.sort\_values("CountryName", axis=0, ascending=False,

inplace=True)

print("Process Data Values =================================")

print(ProcessData)

print("=====================================================")

# Output Agreement ===========================================

OutputData = ProcessData

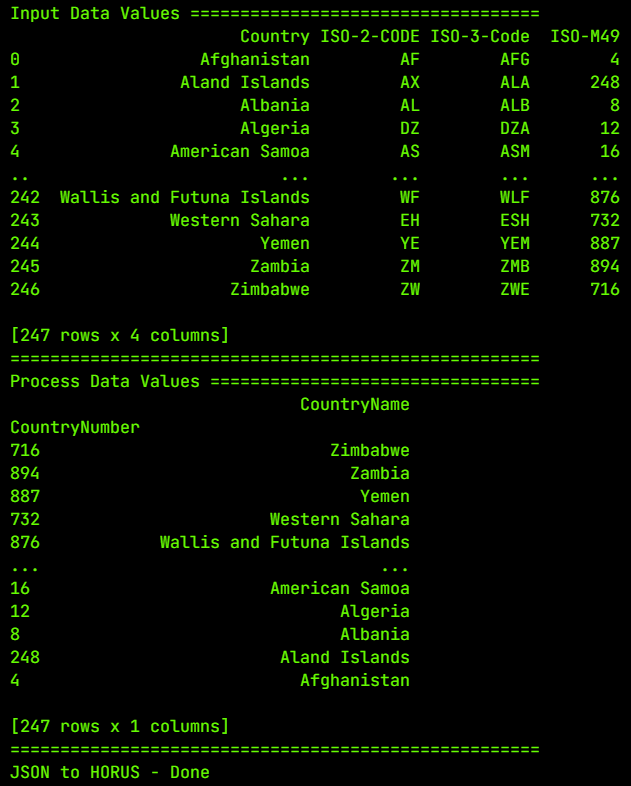
sOutputFileName = "D:/VKHCG/05-DS/9999-Data/HORUS-JSON-Country.csv"

OutputData.to\_csv(sOutputFileName, index=False)

print("JSON to HORUS - Done")

# Utility done ===============================================

**Output:**

****

**2D. MySql Database to HORUS Format**

# Standard Tools

# =============================================================

import pandas as pd

import sqlite3 as sq

# Input Agreement ============================================

sInputFileName = "D:/VKHCG/05-DS/9999-Data/utility.db"

sInputTable = "Country\_Code"

conn = sq.connect(sInputFileName)

sSQL = "select \* FROM " + sInputTable + ";"

InputData = pd.read\_sql\_query(sSQL, conn)

print("Input Data Values ===================================")

print(InputData)

print("=====================================================")

# Processing Rules ===========================================

ProcessData = InputData

# Remove columns ISO-2-Code and ISO-3-CODE

ProcessData.drop("ISO-2-CODE", axis=1, inplace=True)

ProcessData.drop("ISO-3-Code", axis=1, inplace=True)

# Rename Country and ISO-M49

ProcessData.rename(columns={"Country": "CountryName"}, inplace=True)

ProcessData.rename(columns={"ISO-M49": "CountryNumber"}, inplace=True)

# Set new Index

ProcessData.set\_index("CountryNumber", inplace=True)

# Sort data by CurrencyNumber

ProcessData.sort\_values("CountryName", axis=0, ascending=False,

inplace=True)

print("Process Data Values =================================")

print(ProcessData)

print("=====================================================")

# Output Agreement ===========================================

OutputData = ProcessData

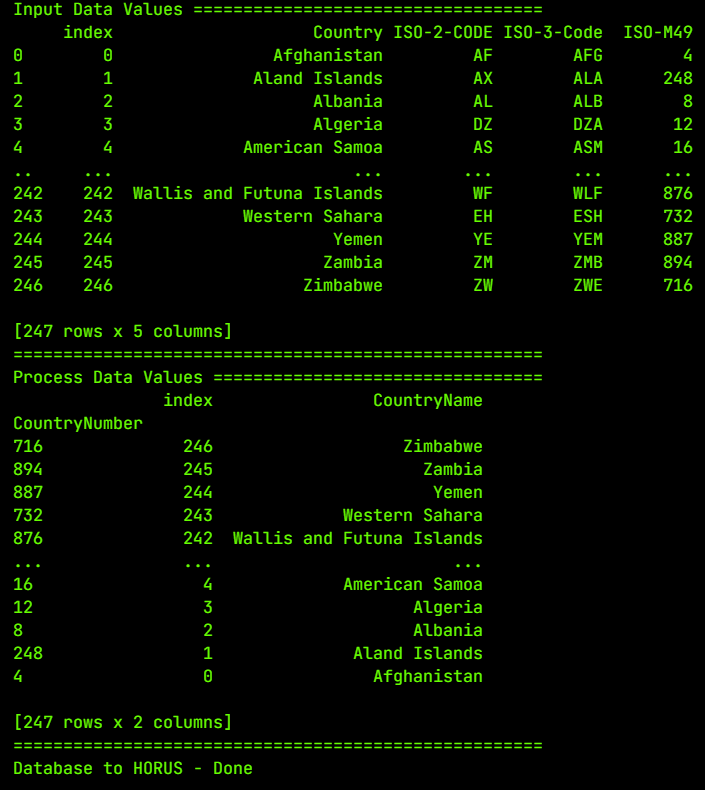
sOutputFileName = "D:/VKHCG/05-DS/9999-Data/HORUS-CSV-Country.csv"

OutputData.to\_csv(sOutputFileName, index=False)

print("Database to HORUS - Done")

# Utility done ===============================================

**Output:**

****

**Practical No: 03**

**Aim: Utilities and Auditing**

**3A. Fixers Utilities**

# ---------------------------- Program to Demonstrate Fixers utilities

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

import string

import datetime as dt

# 1 Removing leading or lagging spaces from a data entry

print("\n#1 Removing leading or lagging spaces from a data entry")

baddata = " Data Science with too many spaces is bad!!! "

print(">", baddata, "<")

cleandata = baddata.strip()

print(">", cleandata, "<")

# 2 Removing nonprintable characters from a data entry

print("#2 Removing nonprintable characters from a data entry")

printable = set(string.printable)

baddata = "Data\x00Science with\x02 funny characters is \x10bad!!!"

cleandata = "".join(filter(lambda x: x in string.printable, baddata))

print("Bad Data : ", baddata)

print("Clean Data : ", cleandata)

# 3 Reformatting data entry to match specific formatting criteria.

# Convert YYYY/MM/DD to DD Month YYYY

print("# 3 Reformatting data entry to match specific formatting

criteria.")

baddate = dt.date(2019, 10, 31)

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:**

****

**3B. Data Binning or Bucketing**

import matplotlib.mlab as mlab

import matplotlib.pyplot as plt

import numpy as np

import scipy.stats as stats

np.random.seed(0)

# example data

mu = 90 # mean of distribution

sigma = 25 # standard deviation of distribution

x = mu + sigma \* np.random.randn(5000)

num\_bins = 25

fig, ax = plt.subplots()

# the histogram of the data

n, bins, patches = ax.hist(

x,

num\_bins,

color="royalblue",

density=1,

alpha=1,

align="right",

orientation="vertical",

cumulative=True,

)

# add a 'best fit' line

y = stats.norm.pdf(bins, mu, sigma)

# mlab.normpdf(bins, mu, sigma)

ax.plot(bins, y, "--")

ax.set\_xlabel("Example Data")

ax.set\_ylabel("Probability density")

sTitle = (

r"Histogram "

+ str(len(x))

+ " entries into "

+ str(num\_bins)

+ " Bins: $\mu="

+ str(mu)

+ "$, $\sigma="

+ str(sigma)

+ "$"

)

ax.set\_title(sTitle)

fig.tight\_layout()

sPathFig = "D:/OneDrive - South Indian Education Society/Online MSIT

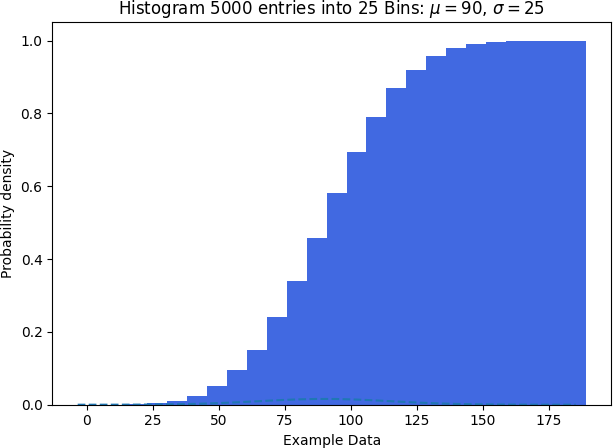
Classes/Practicals/Data Science/Practical 3/3B/DU-Histogram.png"

fig.savefig(sPathFig)

plt.plot()

plt.show()

**Output:**

****

**3C. Averaging of Data**

import pandas as pd

################################################################

InputFileName = "IP\_DATA\_CORE.csv"

OutputFileName = "Retrieve\_Router\_Location.csv"

Base = "D:/VKHCG"

print("################################")

print("Working Base :", Base, " using ")

print("################################")

sFileName = Base + "/01-Vermeulen/00-RawData/" + InputFileName

print("Loading :", sFileName)

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)

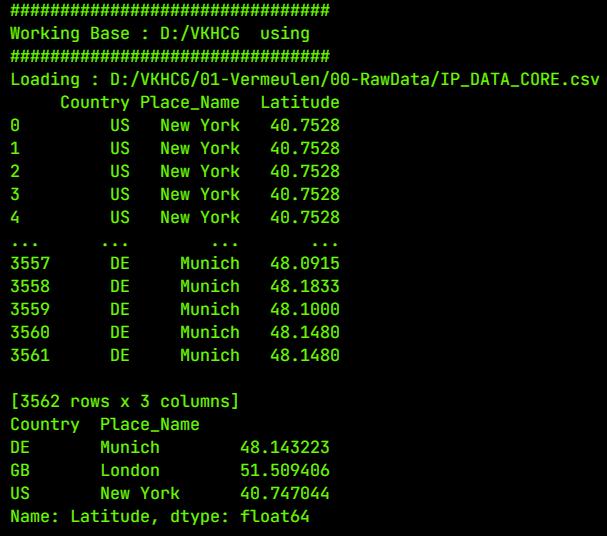
MeanData = AllData.groupby(["Country",

"Place\_Name"])["Latitude"].mean()

print(MeanData)

################################################################

**Output:**



**Practical No: 04**

**Aim: 4A. Data processing using R.**

library(readr)

IP\_DATA\_ALL <-

read\_csv("D:/VKHCG/01-Vermeulen/00-RawData/IP\_DATA\_ALL.csv")

View(IP\_DATA\_ALL)

spec(IP\_DATA\_ALL)

library(tibble)

set\_tidy\_names(IP\_DATA\_ALL, syntactic = TRUE, quiet = FALSE)

sapply(IP\_DATA\_ALL\_FIX, typeof)

library(data.table)

hist\_country = data.table(Country =

unique(IP\_DATA\_ALL\_FIX[is.na(IP\_DATA\_ALL\_FIX['Country']) ==

0,]$Country))

setorder(hist\_country, 'Country')

hist\_country\_with\_id = rowid\_to\_column(hist\_country, var =

"RowIDCountry")

View(hist\_country\_fix)

IP\_DATA\_COUNTRY\_FREQ = data.table(with(IP\_DATA\_ALL\_FIX,

table(Country)))

View(IP\_DATA\_COUNTRY\_FREQ)

hist\_latitude = data.table(Latitude =

unique(IP\_DATA\_ALL\_FIX[is.na(IP\_DATA\_ALL\_with\_ID['Latitude']) ==

0,]$Latitude))

setkeyv(hist\_latitude, 'Latitude')

setorder(hist\_latitude)

hist\_latitude\_with\_id = rowid\_to\_column(hist\_latitude, var = "RowID")

View(hist\_latitude\_with\_id)

IP\_DATA\_Latitude\_FREQ = data.table(with(IP\_DATA\_ALL\_FIX,

table(Latitude)))

View(IP\_DATA\_Latitude\_FREQ)

sapply(IP\_DATA\_ALL\_FIX[, 'Latitude'], min, na.rm = TRUE)

sapply(IP\_DATA\_ALL\_FIX[, 'Country'], min, na.rm = TRUE)

sapply(IP\_DATA\_ALL\_FIX[, 'Latitude'], max, na.rm = TRUE)

sapply(IP\_DATA\_ALL\_FIX[, 'Country'], max, na.rm = TRUE)

sapply(IP\_DATA\_ALL\_FIX[, 'Latitude'], mean, na.rm = TRUE)

sapply(IP\_DATA\_ALL\_FIX[, 'Latitude'], median, na.rm = TRUE)

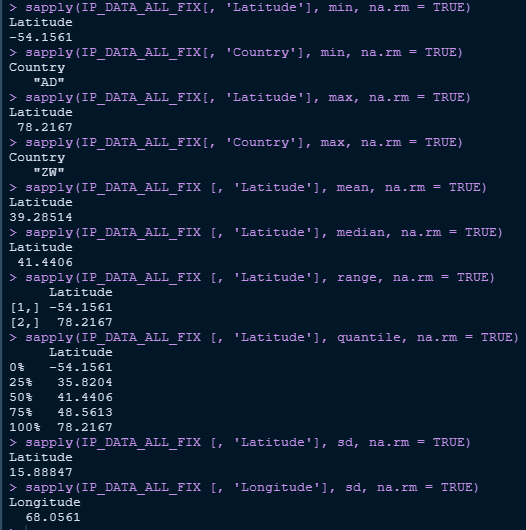
sapply(IP\_DATA\_ALL\_FIX[, 'Latitude'], range, na.rm = TRUE)

sapply(IP\_DATA\_ALL\_FIX[, 'Latitude'], quantile, na.rm = TRUE)

sapply(IP\_DATA\_ALL\_FIX[, 'Latitude'], sd, na.rm = TRUE)

sapply(IP\_DATA\_ALL\_FIX[, 'Longitude'], sd, na.rm = TRUE)

**Output:**

****

**4B. Program to retrieve different attributes of data**

import os

import pandas as pd

################################################################

Base = "D:/VKHCG"

################################################################

sFileName = Base + "/01-Vermeulen/00-RawData/IP\_DATA\_ALL.csv"

print("Loading:", sFileName)

IP\_DATA\_ALL = pd.read\_csv(sFileName, header=0, low\_memory=False,

encoding="latin-1")

################################################################

sFileDir = Base + "01-Vermeulen/01-Retrieve/01-EDS/02-Python"

if not os.path.exists(sFileDir):

os.makedirs(sFileDir)

print("Rows", IP\_DATA\_ALL.shape[0])

print("Columns", IP\_DATA\_ALL.shape[1])

print("### Raw Data Set #####################################")

for i in range(0, len(IP\_DATA\_ALL.columns)):

print(IP\_DATA\_ALL.columns[i], type(IP\_DATA\_ALL.columns[i]))

print("### Fixed Data Set #####################################")

IP\_DATA\_ALL\_FIX = IP\_DATA\_ALL

for i in range(0, len(IP\_DATA\_ALL.columns)):

cNameOld = IP\_DATA\_ALL\_FIX.columns[i] + ""

cNameNew = cNameOld.strip().replace(" ", ".")

IP\_DATA\_ALL\_FIX.columns.values[i] = cNameNew

print(IP\_DATA\_ALL.columns[i], type(IP\_DATA\_ALL.columns[i]))

################################################################

# print (IP\_DATA\_ALL\_FIX.head()

################################################################

print("Fixed Data Set with ID")

IP\_DATA\_ALL\_with\_ID = IP\_DATA\_ALL\_FIX

IP\_DATA\_ALL\_with\_ID.index.names = ["RowID"]

# print (IP\_DATA\_ALL\_FIX\_with\_ID.head()

sFileName2 = sFileDir + "/Retrieve\_IP\_DATA.csv"

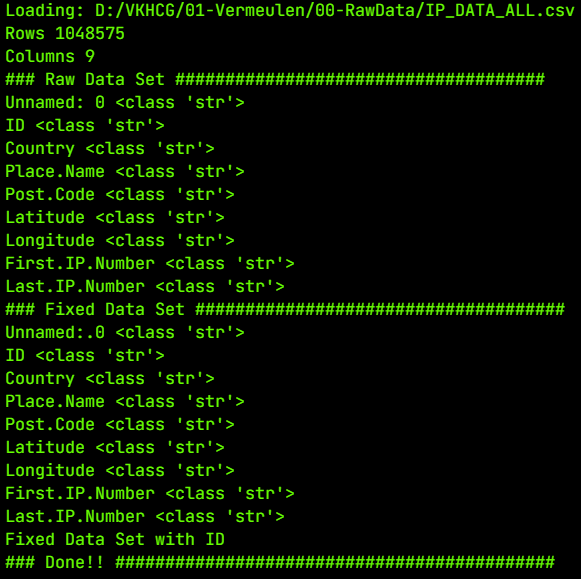
IP\_DATA\_ALL\_with\_ID.to\_csv(sFileName2, index=True, encoding="latin-1")

################################################################

print("### Done!! ############################################")

################################################################

**Output:**



**4C.Data Pattern**

library(readr)

library(data.table)

FileName = paste0('D:/VKHCG/01-Vermeulen/00-RawData/IP\_DATA\_ALL.csv')

IP\_DATA\_ALL <- read.csv(FileName)

hist\_country = data.table(Country = unique(IP\_DATA\_ALL$Country))

pattern\_country = data.table(Country = hist\_country$Country,

PatternCountry = hist\_country$Country)

oldchar = c(letters, LETTERS)

newchar = replicate(length(oldchar), "A")

for (r in seq(nrow(pattern\_country))) {

s = pattern\_country[r,]$PatternCountry

for (c in seq(length(oldchar))) {

s = chartr(oldchar[c], newchar[c], s)

}

s = chartr(" ", "b", s)

s = chartr(".", "u", s)

pattern\_country[r,]$PatternCountry = s

}

View(pattern\_country)

**Output:**



**Practical No: 05**

**Aim: Assessing Data**

**5A. Perform error management on the given data using pandas package**

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

import os

import sys

import pandas as pd

################################################################

Base = "D:/VKHCG"

################################################################

print("################################")

print("Working Base :", Base, " using ", sys.platform)

print("################################")

################################################################

sInputFileName = "Good-or-Bad.csv"

sOutputFileName = "Good-or-Bad-01.csv"

Company = "01-Vermeulen"

################################################################

Base = "D:/VKHCG"

################################################################

sFileDir = Base + "/" + Company + "/02-Assess/01-EDS/02-Python"

if not os.path.exists(sFileDir):

os.makedirs(sFileDir)

################################################################

# Import Warehouse

################################################################

sFileName = Base + "/" + Company + "/00-RawData/" + sInputFileName

print("Loading :", sFileName)

RawData = pd.read\_csv(sFileName, header=0)

print("################################")

print("## Raw Data Values")

print("################################")

print(RawData)

print("################################")

print("## Data Profile")

print("################################")

print("Rows :", RawData.shape[0])

print("Columns :", RawData.shape[1])

print("################################")

################################################################

sFileName = sFileDir + "/" + sInputFileName

RawData.to\_csv(sFileName, index=False)

################################################################

TestData = RawData.dropna(axis=1, how="all")

################################################################

print("################################")

print("## Test Data Values")

print("################################")

print(TestData)

print("################################")

print("## Data Profile")

print("################################")

print("Rows :", TestData.shape[0])

print("Columns :", TestData.shape[1])

print("################################")

################################################################

sFileName = sFileDir + "/" + sOutputFileName

TestData.to\_csv(sFileName, index=False)

################################################################

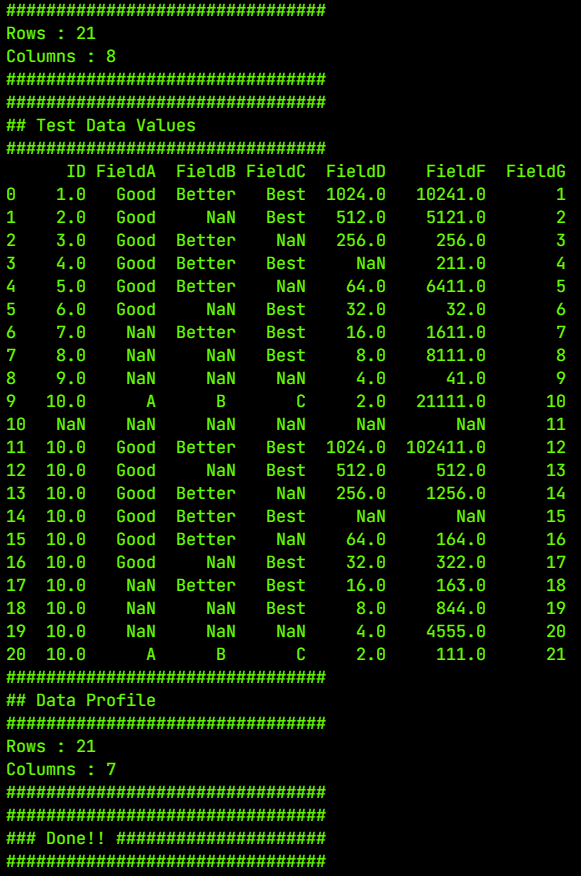
print("################################")

print("### Done!! #####################")

print("################################")

################################################################

**Output:**



**II. Drop the Columns Where Any of the Elements Is Missing Values**

import sys

import os

import pandas as pd

################################################################

Base = "D:/VKHCG"

sInputFileName = "Good-or-Bad.csv"

sOutputFileName = "Good-or-Bad-02.csv"

Company = "01-Vermeulen"

################################################################

print("################################")

print("Working Base :", Base, " using ", sys.platform)

print("################################")

################################################################

sFileDir = Base + "/" + Company + "/02-Assess/01-EDS/02-Python"

if not os.path.exists(sFileDir):

os.makedirs(sFileDir)

################################################################

# Import Warehouse

################################################################

sFileName = Base + "/" + Company + "/00-RawData/" + sInputFileName

print("Loading :", sFileName)

RawData = pd.read\_csv(sFileName, header=0)

print("################################")

print("## Raw Data Values")

print("################################")

print(RawData)

print("################################")

print("## Data Profile")

print("################################")

print("Rows :", RawData.shape[0])

print("Columns :", RawData.shape[1])

print("################################")

################################################################

sFileName = sFileDir + "/" + sInputFileName

RawData.to\_csv(sFileName, index=False)

################################################################

TestData = RawData.dropna(axis=1, how="any")

################################################################

print("################################")

print("## Test Data Values")

print("################################")

print(TestData)

print("################################")

print("## Data Profile")

print("################################")

print("Rows :", TestData.shape[0])

print("Columns :", TestData.shape[1])

print("################################")

################################################################

sFileName = sFileDir + "/" + sOutputFileName

TestData.to\_csv(sFileName, index=False)

################################################################

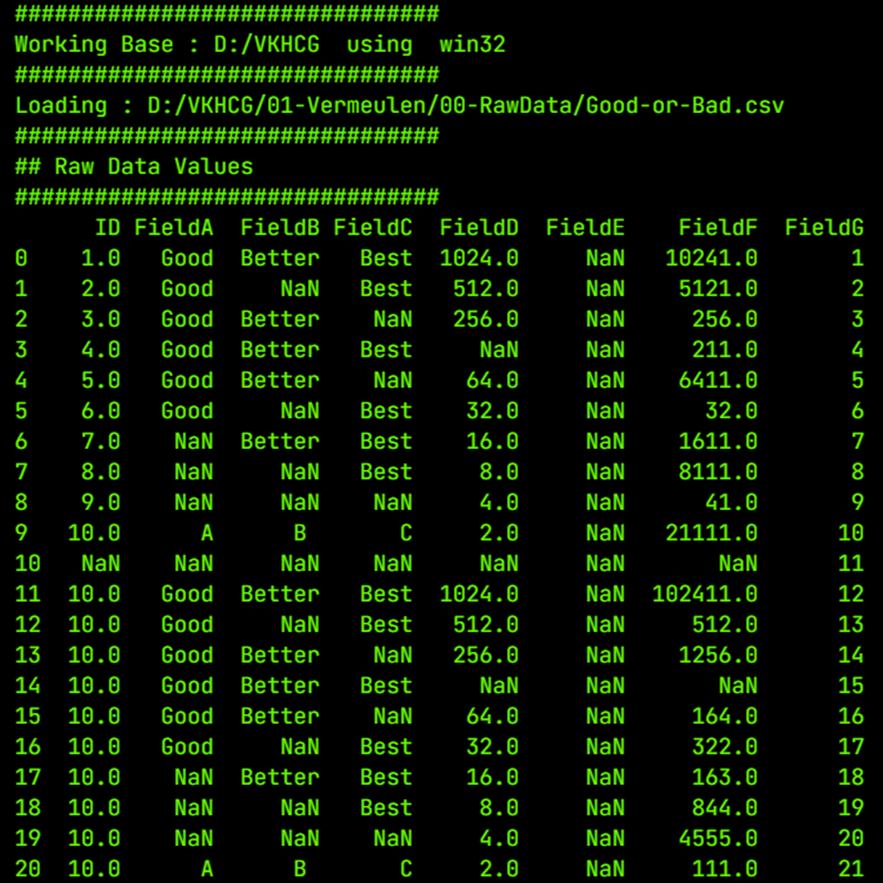
print("################################")

print("### Done!! #####################")

print("################################")

################################################################

**Output:**

****

**III. Keep Only the Rows That Contain a Maximum of Two Missing Values**

import sys

import os

import pandas as pd

################################################################

sInputFileName = "Good-or-Bad.csv"

sOutputFileName = "Good-or-Bad-03.csv"

Company = "01-Vermeulen"

Base = "D:/VKHCG"

################################################################

print("################################")

print("Working Base :", Base, sys.platform)

print("################################")

################################################################

sFileDir = Base + "/" + Company + "/02-Assess/01-EDS/02-Python"

if not os.path.exists(sFileDir):

os.makedirs(sFileDir)

################################################################

### Import Warehouse

################################################################

sFileName = Base + "/" + Company + "/00-RawData/" + sInputFileName

print("Loading :", sFileName)

RawData = pd.read\_csv(sFileName, header=0)

print("################################")

print("## Raw Data Values")

print("################################")

print(RawData)

print("################################")

print("## Data Profile")

print("################################")

print("Rows :", RawData.shape[0])

print("Columns :", RawData.shape[1])

print("################################")

################################################################

sFileName = sFileDir + "/" + sInputFileName

RawData.to\_csv(sFileName, index=False)

################################################################

TestData = RawData.dropna(thresh=2)

print("################################")

print("## Test Data Values")

print("################################")

print(TestData)

print("################################")

print("## Data Profile")

print("################################")

print("Rows :", TestData.shape[0])

print("Columns :", TestData.shape[1])

print("################################")

sFileName = sFileDir + "/" + sOutputFileName

TestData.to\_csv(sFileName, index=False)

################################################################

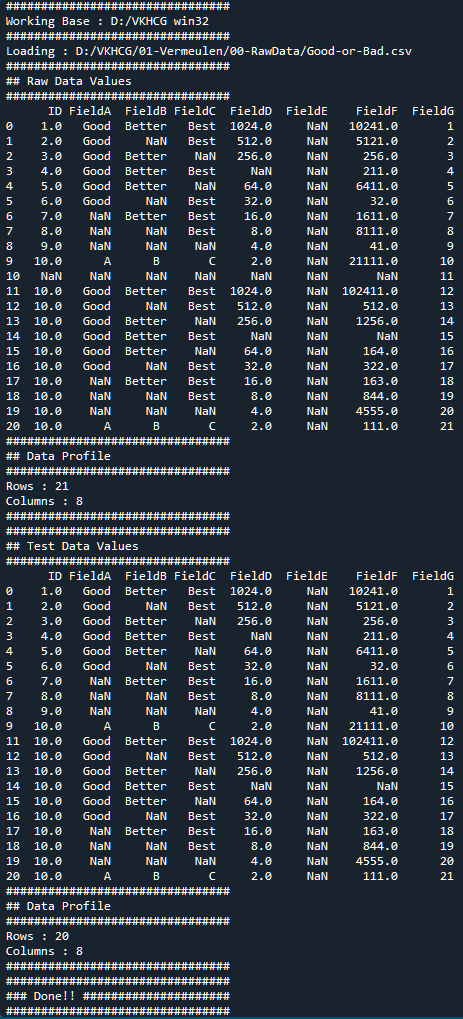
print("################################")

print("### Done!! #####################")

print("################################")

################################################################

**Output:**



**5B. Write Python / R program to create the network routing diagram from the given data on routers.**

**assess-network-routing-company.py**

########## Assess-Network-Routing-Company.py #####################

import os

import sys

import pandas as pd

################################################################

pd.options.mode.chained\_assignment = None

################################################################

Base = "D:/VKHCG"

################################################################

print("################################")

print("Working Base :", Base, sys.platform)

print("################################")

################################################################

sInputFileName1 = "01-Retrieve/01-EDS/01-R/Retrieve\_Country\_Code.csv"

sInputFileName2 = "01-Retrieve/01-EDS/02-

Python/Retrieve\_Router\_Location.csv"

sInputFileName3 = "01-Retrieve/01-EDS/01-R/Retrieve\_IP\_DATA.csv"

################################################################

sOutputFileName = "Assess-Network-Routing-Company.csv"

Company = "01-Vermeulen"

################################################################

################################################################

### Import Country Data

################################################################

sFileName = Base + "/" + Company + "/" + sInputFileName1

print("################################")

print("Loading :", sFileName)

print("################################")

CountryData = pd.read\_csv(sFileName, header=0, low\_memory=False,

encoding="latin-1")

print("Loaded Country:", CountryData.columns.values)

print("################################")

################################################################

## Assess Country Data

################################################################

print("################################")

print("Changed :", CountryData.columns.values)

CountryData.rename(columns={"Country": "Country\_Name"}, inplace=True)

CountryData.rename(columns={"ISO-2-CODE": "Country\_Code"},

inplace=True)

CountryData.drop("ISO-M49", axis=1, inplace=True)

CountryData.drop("ISO-3-Code", axis=1, inplace=True)

CountryData.drop("RowID", axis=1, inplace=True)

print("To :", CountryData.columns.values)

print("################################")

################################################################

################################################################

### Import Company Data

################################################################

sFileName = Base + "/" + Company + "/" + sInputFileName2

print("################################")

print("Loading :", sFileName)

print("################################")

CompanyData = pd.read\_csv(sFileName, header=0, low\_memory=False,

encoding="latin-1")

print("Loaded Company :", CompanyData.columns.values)

print("################################")

################################################################

## Assess Company Data

################################################################

print("################################")

print("Changed :", CompanyData.columns.values)

CompanyData.rename(columns={"Country": "Country\_Code"}, inplace=True)

print("To :", CompanyData.columns.values)

print("################################")

################################################################

################################################################

### Import Customer Data

################################################################

sFileName = Base + "/" + Company + "/" + sInputFileName3

print("################################")

print("Loading :", sFileName)

print("################################")

CustomerRawData = pd.read\_csv(sFileName, header=0, low\_memory=False,

encoding="latin-1")

print("################################")

print("Loaded Customer :", CustomerRawData.columns.values)

print("################################")

################################################################

CustomerData = CustomerRawData.dropna(axis=0, how="any")

print("################################")

print("Remove Blank Country Code")

print("Reduce Rows from", CustomerRawData.shape[0], " to ",

CustomerData.shape[0])

print("################################")

################################################################

print("################################")

print("Changed :", CustomerData.columns.values)

CustomerData.rename(columns={"Country": "Country\_Code"}, inplace=True)

print("To :", CustomerData.columns.values)

print("################################")

################################################################

print("################################")

print("Merge Company and Country Data")

print("################################")

CompanyNetworkData = pd.merge(CompanyData, CountryData, how="inner",

on="Country\_Code")

################################################################

print("################################")

print("Change ", CompanyNetworkData.columns.values)

for i in CompanyNetworkData.columns.values:

j = "Company\_" + i

CompanyNetworkData.rename(columns={i: j}, inplace=True)

print("To ", CompanyNetworkData.columns.values)

print("################################")

################################################################

################################################################

sFileDir = Base + "/" + Company + "/02-Assess/01-EDS/02-Python"

if not os.path.exists(sFileDir):

os.makedirs(sFileDir)

################################################################

sFileName = sFileDir + "/" + sOutputFileName

print("################################")

print("Storing :", sFileName)

print("################################")

CompanyNetworkData.to\_csv(sFileName, index=False, encoding="latin-1")

################################################################

################################################################

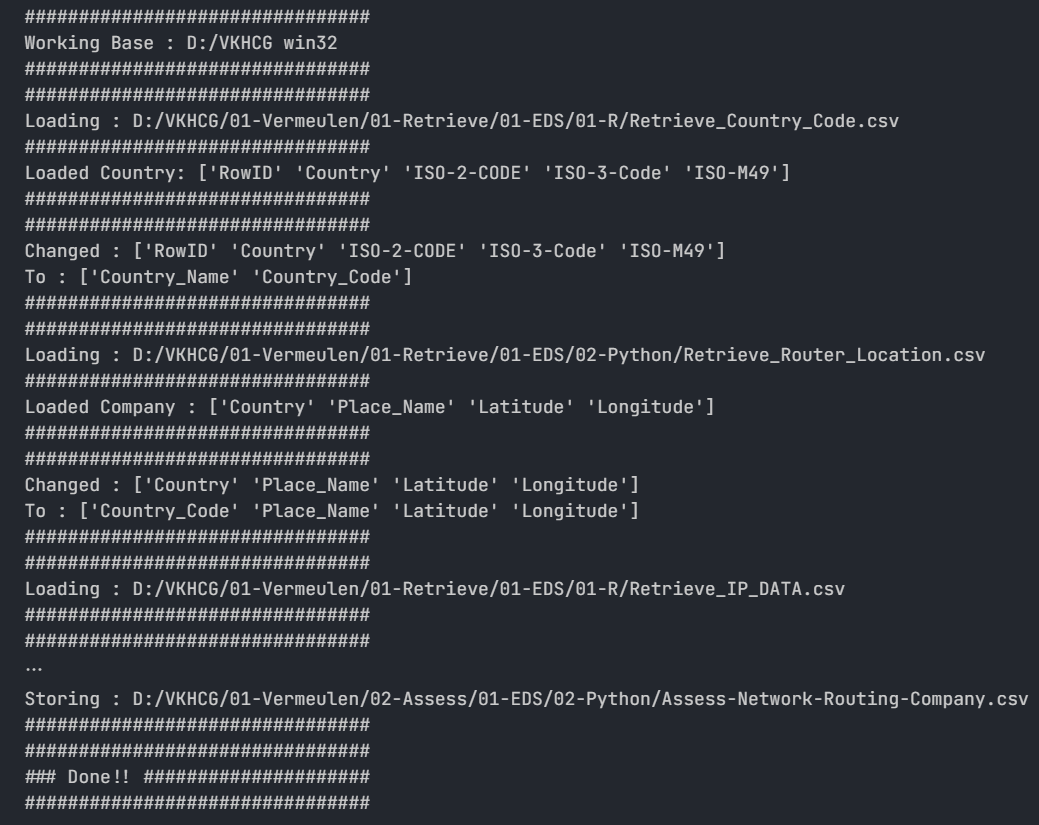
print("################################")

print("### Done!! #####################")

print("################################")

################################################################

**Output:**

****

**assess-network-routing-customer.py**

import os

import sys

import pandas as pd

################################################################

pd.options.mode.chained\_assignment = None

################################################################

Base = "D:/VKHCG"

print("################################")

print("Working Base :", Base, " using ", sys.platform)

print("################################")

################################################################

sInputFileName = (Base+ "/01-Vermeulen/02-Assess/01-EDS/02-Python/Assess-Network-

Routing-Customer.csv")

################################################################

sOutputFileName = "Assess-Network-Routing-Customer.gml"

Company = "01-Vermeulen"

################################################################

### Import Country Data

################################################################

sFileName = sInputFileName

print("################################")

print("Loading :", sFileName)

print("################################")

CustomerData = pd.read\_csv(sFileName, header=0, low\_memory=False,

encoding="latin-1")

print("Loaded Country:", CustomerData.columns.values)

print("################################")

print(CustomerData.head())

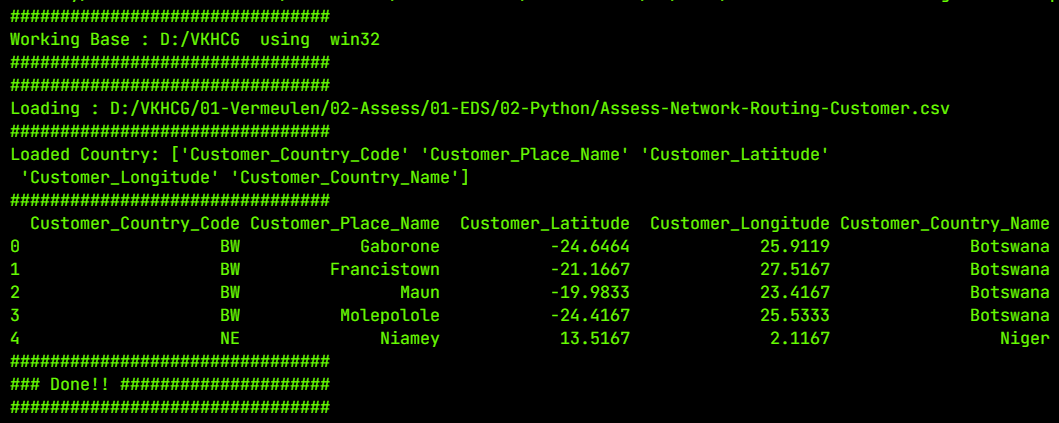
print("################################")

print("### Done!! #####################")

print("################################")

################################################################

**Output:**

****

**5C. Write a Python / R program to build directed acyclic graph**

**assess-DAG-location.py**

import os

import sys

import matplotlib.pyplot as plt

import networkx as nx

import pandas as pd

################################################################

Base = "D:/VKHCG"

################################################################

print("################################")

print("Working Base :", Base, " using ", sys.platform)

print("################################")

################################################################

sInputFileName = "01-Retrieve/01-EDS/02-

Python/Retrieve\_Router\_Location.csv"

sOutputFileName1 = "Assess-DAG-Company-Country.png"

sOutputFileName2 = "Assess-DAG-Company-Country-Place.png"

Company = "01-Vermeulen"

################################################################

#### Import Company Data

################################################################

sFileName = Base + "/" + Company + "/" + sInputFileName

print("################################")

print("Loading :", sFileName)

print("################################")

CompanyData = pd.read\_csv(sFileName, header=0, low\_memory=False,

encoding="latin-1")

print("Loaded Company :", CompanyData.columns.values)

print("################################")

################################################################

print(CompanyData)

print("################################")

print("Rows : ", CompanyData.shape[0])

print("################################")

################################################################

G1 = nx.DiGraph()

G2 = nx.DiGraph()

################################################################

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

G1.add\_node(CompanyData["Country"][i])

sPlaceName = CompanyData["Place\_Name"][i] + "-" +

CompanyData["Country"][i]

G2.add\_node(sPlaceName)

print("################################")

for n1 in G1.nodes():

for n2 in G1.nodes():

if n1 != n2:

print("Link :", n1, " to ", n2)

G1.add\_edge(n1, n2)

print("################################")

print("################################")

print("Nodes of graph: ")

print(G1.nodes())

print("Edges of graph: ")

print(G1.edges())

print("################################")

################################################################

sFileDir = Base + "/" + Company + "/02-Assess/01-EDS/02-Python"

if not os.path.exists(sFileDir):

os.makedirs(sFileDir)

################################################################

sFileName = sFileDir + "/" + sOutputFileName1

print("################################")

print("Storing :", sFileName)

print("################################")

nx.draw(

G1,

pos=nx.spectral\_layout(G1),

edge\_color="g",

with\_labels=True,

node\_size=8000,

font\_size=12,

)

plt.savefig(sFileName) # save as png

plt.show() # display

################################################################

print("################################")

for n1 in G2.nodes():

for n2 in G2.nodes():

if n1 != n2:

print("Link :", n1, " to ", n2)

G2.add\_edge(n1, n2)

print("################################")

print("################################")

print("Nodes of graph: ")

print(G2.nodes())

print("Edges of graph: ")

print(G2.edges())

print("################################")

################################################################

sFileDir = Base + "/" + Company + "/02-Assess/01-EDS/02-Python"

if not os.path.exists(sFileDir):

os.makedirs(sFileDir)

################################################################

sFileName = sFileDir + "/" + sOutputFileName2

print("################################")

print("Storing :", sFileName)

print("################################")

nx.draw(

G2,

pos=nx.spectral\_layout(G2),

edge\_color="b",

node\_color="g",

with\_labels=True,

node\_size=8000,

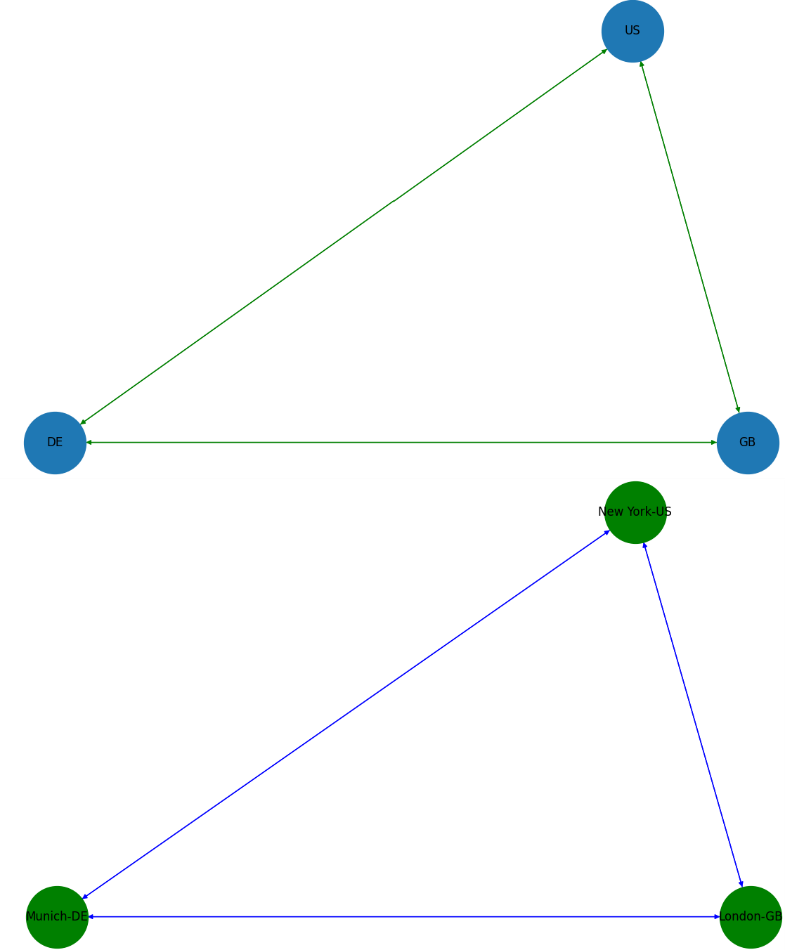
font\_size=12,

)

plt.savefig(sFileName) # save as png

plt.show()

**Output:**

****

**Practical No: 06**

**Aim: 6A. Build the time hub, links, and satellites**

import sys

import os

from datetime import datetime

from datetime import timedelta

from pytz import timezone, all\_timezones

import pandas as pd

import sqlite3 as sq

from pandas.io import sql

import uuid

pd.options.mode.chained\_assignment = None

################################################################

Base = "D:/VKHCG"

print("################################")

print("Working Base :", Base, " using ", sys.platform)

print("################################")

################################################################

Company = "01-Vermeulen"

InputDir = "00-RawData"

InputFileName = "VehicleData.csv"

################################################################

sDataBaseDir = Base + "/" + Company + "/03-Process/SQLite"

if not os.path.exists(sDataBaseDir):

os.makedirs(sDataBaseDir)

################################################################

sDatabaseName = sDataBaseDir + "/Hillman.db"

conn1 = sq.connect(sDatabaseName)

################################################################

sDataVaultDir = Base + "/88-DV"

if not os.path.exists(sDataBaseDir):

os.makedirs(sDataBaseDir)

################################################################

sDatabaseName = sDataVaultDir + "/datavault.db"

conn2 = sq.connect(sDatabaseName)

################################################################

base = datetime(2018, 1, 1, 0, 0, 0)

numUnits = 10 \* 365 \* 24

################################################################

date\_list = [base - timedelta(hours=x) for x in range(0, numUnits)]

t = 0

for i in date\_list:

now\_utc = i.replace(tzinfo=timezone("UTC"))

sDateTime = now\_utc.strftime("%Y-%m-%d %H:%M:%S")

print(sDateTime)

sDateTimeKey = sDateTime.replace(" ", "-").replace(":", "-")

t += 1

IDNumber = str(uuid.uuid4())

TimeLine = [

("ZoneBaseKey", ["UTC"]),

("IDNumber", [IDNumber]),

("nDateTimeValue", [now\_utc]),

("DateTimeValue", [sDateTime]),

("DateTimeKey", [sDateTimeKey]),]

if t == 1:

TimeFrame = pd.DataFrame.from\_dict(dict(TimeLine))

else:

TimeRow = pd.DataFrame.from\_dict(dict(TimeLine))

TimeFrame = TimeFrame.append(TimeRow)

################################################################

TimeHub = TimeFrame[["IDNumber", "ZoneBaseKey", "DateTimeKey",

"DateTimeValue"]]

TimeHubIndex = TimeHub.set\_index(["IDNumber"], inplace=False)

################################################################

TimeFrame.set\_index(["IDNumber"], inplace=True)

################################################################

sTable = "Process-Time"

print("Storing :", sDatabaseName, " Table:", sTable)

TimeHubIndex.to\_sql(sTable, conn1, if\_exists="replace")

################################################################

sTable = "Hub-Time"

print("Storing :", sDatabaseName, " Table:", sTable)

TimeHubIndex.to\_sql(sTable, conn2, if\_exists="replace")

################################################################

active\_timezones = all\_timezones

z = 0

for zone in active\_timezones:

t = 0

for j in range(TimeFrame.shape[0]):

now\_date = TimeFrame["nDateTimeValue"][j]

DateTimeKey = TimeFrame["DateTimeKey"][j]

now\_utc = now\_date.replace(tzinfo=timezone("UTC"))

sDateTime = now\_utc.strftime("%Y-%m-%d %H:%M:%S")

now\_zone = now\_utc.astimezone(timezone(zone))

sZoneDateTime = now\_zone.strftime("%Y-%m-%d %H:%M:%S")

print(sZoneDateTime)

t += 1

z += 1

IDZoneNumber = str(uuid.uuid4())

TimeZoneLine = [

("ZoneBaseKey", ["UTC"]),

("IDZoneNumber", [IDZoneNumber]),

("DateTimeKey", [DateTimeKey]),

("UTCDateTimeValue", [sDateTime]),

("Zone", [zone]),

("DateTimeValue", [sZoneDateTime]),]

if t == 1:

TimeZoneFrame = pd.DataFrame.from\_dict(dict(TimeZoneLine))

else:

TimeZoneRow = pd.DataFrame.from\_dict(dict(TimeZoneLine))

TimeZoneFrame = TimeZoneFrame.append(TimeZoneRow)

TimeZoneFrameIndex = TimeZoneFrame.set\_index(["IDZoneNumber"],

inplace=False)

sZone = zone.replace("/", "-").replace(" ", "")

#############################################################

sTable = "Process-Time-" + sZone

print("Storing :", sDatabaseName, " Table:", sTable)

TimeZoneFrameIndex.to\_sql(sTable, conn1, if\_exists="replace")

#################################################################

sTable = "Satellite-Time-" + sZone

print("Storing :", sDatabaseName, " Table:", sTable)

TimeZoneFrameIndex.to\_sql(sTable, conn2, if\_exists="replace")

#################################################################

print("################")

print("Vacuum Databases")

sSQL = "VACUUM;"

sql.execute(sSQL, conn1)

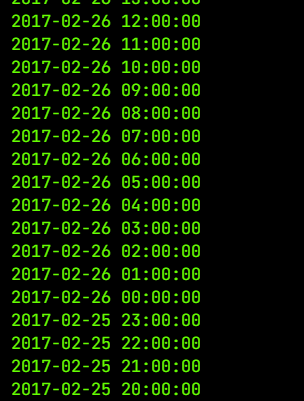
sql.execute(sSQL, conn2)

print("################")

print("### Done!! ############################################")

#################################################################

**Output:**



**6B. Write a program to load the vehicle data for Hillman Ltd into the data vault**

import sys

import os

import pandas as pd

import sqlite3 as sq

from pandas.io import sql

import uuid

pd.options.mode.chained\_assignment = None

################################################################

Base = "D:/VKHCG"

print("################################")

print("Working Base :", Base, " using ", sys.platform)

print("################################")

################################################################

Company = "03-Hillman"

InputDir = "00-RawData"

InputFileName = "VehicleData.csv"

################################################################

sDataBaseDir = Base + "/" + Company + "/03-Process/SQLite"

if not os.path.exists(sDataBaseDir):

os.makedirs(sDataBaseDir)

################################################################

sDatabaseName = sDataBaseDir + "/Hillman.db"

conn1 = sq.connect(sDatabaseName)

################################################################

sDataVaultDir = Base + "/88-DV"

if not os.path.exists(sDataBaseDir):

os.makedirs(sDataBaseDir)

################################################################

sDatabaseName = sDataVaultDir + "/datavault.db"

conn2 = sq.connect(sDatabaseName)

################################################################

sFileName = Base + "/" + Company + "/" + InputDir + "/" +

InputFileName

print("###########")

print("Loading :", sFileName)

VehicleRaw = pd.read\_csv(sFileName, header=0, low\_memory=False,

encoding="latin-1")

################################################################

sTable = "Process\_Vehicles"

print("Storing :", sDatabaseName, " Table:", sTable)

VehicleRaw.to\_sql(sTable, conn1, if\_exists="replace")

################################################################

VehicleRawKey = VehicleRaw[["Make", "Model"]].copy()

VehicleKey = VehicleRawKey.drop\_duplicates()

################################################################

VehicleKey["ObjectKey"] = VehicleKey.apply(

lambda row: str(

"("

+ str(row["Make"]).strip().replace(" ", "-").replace("/", "-

").lower()

+ ")-("

+ (str(row["Model"]).strip().replace(" ", "-").replace(" ", "-

").lower())

+ ")"

),

axis=1,

)

################################################################

VehicleKey["ObjectType"] = VehicleKey.apply(lambda row: "vehicle",

axis=1)

################################################################

VehicleKey["ObjectUUID"] = VehicleKey.apply(lambda row:

str(uuid.uuid4()), axis=1)

################################################################

### Vehicle Hub

################################################################

#

VehicleHub = VehicleKey[["ObjectType", "ObjectKey",

"ObjectUUID"]].copy()

VehicleHub.index.name = "ObjectHubID"

sTable = "Hub-Object-Vehicle"

print("Storing :", sDatabaseName, " Table:", sTable)

VehicleHub.to\_sql(sTable, conn2, if\_exists="replace")

################################################################

### Vehicle Satellite

################################################################

#

VehicleSatellite = VehicleKey[

["ObjectType", "ObjectKey", "ObjectUUID", "Make", "Model"]

].copy()

VehicleSatellite.index.name = "ObjectSatelliteID"

sTable = "Satellite-Object-Make-Model"

print("Storing :", sDatabaseName, " Table:", sTable)

VehicleSatellite.to\_sql(sTable, conn2, if\_exists="replace")

################################################################

### Vehicle Dimension

################################################################

sView = "Dim-Object"

print("Storing :", sDatabaseName, " View:", sView)

sSQL = "CREATE VIEW IF NOT EXISTS [" + sView + "] AS"

sSQL = sSQL + " SELECT DISTINCT"

sSQL = sSQL + " H.ObjectType,"

sSQL = sSQL + " H.ObjectKey AS VehicleKey,"

sSQL = sSQL + " TRIM(S.Make) AS VehicleMake,"

sSQL = sSQL + " TRIM(S.Model) AS VehicleModel"

sSQL = sSQL + " FROM"

sSQL = sSQL + " [Hub-Object-Vehicle] AS H"

sSQL = sSQL + " JOIN"

sSQL = sSQL + " [Satellite-Object-Make-Model] AS S"

sSQL = sSQL + " ON"

sSQL = sSQL + " H.ObjectType=S.ObjectType"

sSQL = sSQL + " AND"

sSQL = sSQL + " H.ObjectUUID=S.ObjectUUID;"

sql.execute(sSQL, conn2)

print("################")

print("Loading :", sDatabaseName, " Table:", sView)

sSQL = " SELECT DISTINCT"

sSQL = sSQL + " VehicleMake,"

sSQL = sSQL + " VehicleModel"

sSQL = sSQL + " FROM"

sSQL = sSQL + " [" + sView + "]"

sSQL = sSQL + " ORDER BY"

sSQL = sSQL + " VehicleMake"

sSQL = sSQL + " AND"

sSQL = sSQL + " VehicleMake;"

DimObjectData = pd.read\_sql\_query(sSQL, conn2)

DimObjectData.index.name = "ObjectDimID"

DimObjectData.sort\_values(["VehicleMake", "VehicleModel"],

inplace=True, ascending=True)

print("################")

print(DimObjectData)

#################################################################

print("################")

print("Vacuum Databases")

sSQL = "VACUUM;"

sql.execute(sSQL, conn1)

sql.execute(sSQL, conn2)

print("################")

#################################################################

conn1.close()

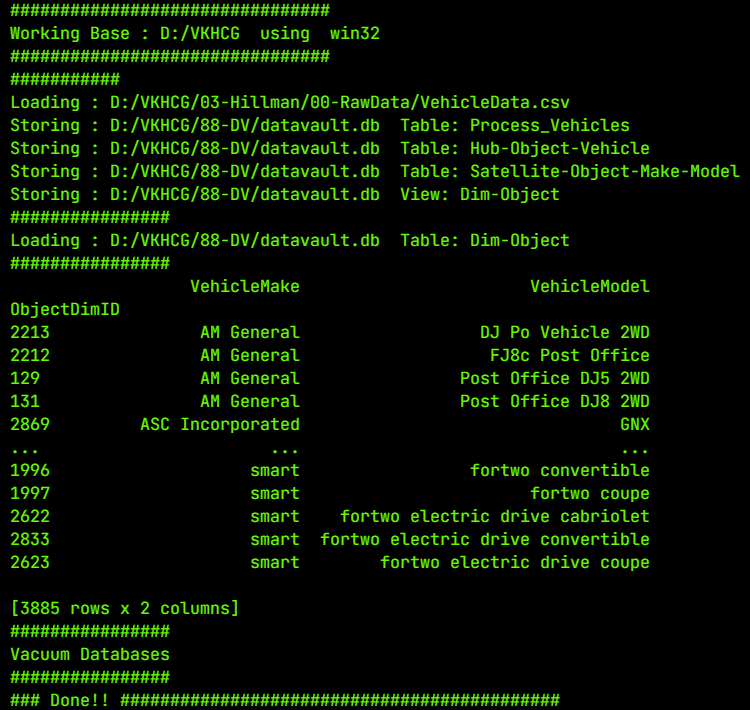
conn2.close()

#################################################################

print("### Done!! ############################################")

#################################################################

**Output:**



**6C. Write a program to build relationship between Process-Location and Hub-Location**

import sys

import os

import pandas as pd

import sqlite3 as sq

from pandas.io import sql

import uuid

################################################################

Base = "D:/VKHCG"

print("################################")

print("Working Base :", Base, " using ", sys.platform)

print("################################")

################################################################

Company = "01-Vermeulen"

InputAssessGraphName = "Assess\_All\_Animals.gml"

EDSAssessDir = "02-Assess/01-EDS"

InputAssessDir = EDSAssessDir + "/02-Python"

################################################################

sFileAssessDir = Base + "/" + Company + "/" + InputAssessDir

if not os.path.exists(sFileAssessDir):

os.makedirs(sFileAssessDir)

################################################################

sDataBaseDir = Base + "/" + Company + "/03-Process/SQLite"

if not os.path.exists(sDataBaseDir):

os.makedirs(sDataBaseDir)

################################################################

sDatabaseName = sDataBaseDir + "/Vermeulen.db"

conn1 = sq.connect(sDatabaseName)

################################################################

sDataVaultDir = Base + "/88-DV"

if not os.path.exists(sDataBaseDir):

os.makedirs(sDataBaseDir)

################################################################

sDatabaseName = sDataVaultDir + "/datavault.db"

conn2 = sq.connect(sDatabaseName)

t = 0

tMax = 360 \* 180

################################################################

for Longitude in range(-180, 180, 10):

for Latitude in range(-90, 90, 10):

t += 1

IDNumber = str(uuid.uuid4())

LocationName = ("L"+ format(round(Longitude, 3) \* 1000, "+07d")

+ "-"+ format(round(Latitude, 3) \* 1000, "+07d"))

print("Create:", t, " of ", tMax, ":", LocationName)

LocationLine = [

("ObjectBaseKey", ["GPS"]),

("IDNumber", [IDNumber]),

("LocationNumber", [str(t)]),

("LocationName", [LocationName]),

("Longitude", [Longitude]),

("Latitude", [Latitude]),]

if t == 1:

LocationFrame = pd.DataFrame.from\_dict(dict(LocationLine))

else:

LocationRow = pd.DataFrame.from\_dict(dict(LocationLine))

LocationFrame = LocationFrame.append(LocationRow)

################################################################

LocationHubIndex = LocationFrame.set\_index(["IDNumber"],

inplace=False)

################################################################

sTable = "Process-Location"

print("Storing :", sDatabaseName, " Table:", sTable)

LocationHubIndex.to\_sql(sTable, conn1, if\_exists="replace")

#################################################################

sTable = "Hub-Location"

print("Storing :", sDatabaseName, " Table:", sTable)

LocationHubIndex.to\_sql(sTable, conn2, if\_exists="replace")

#################################################################

print("################")

print("Vacuum Databases")

sSQL = "VACUUM;"

sql.execute(sSQL, conn1)

sql.execute(sSQL, conn2)

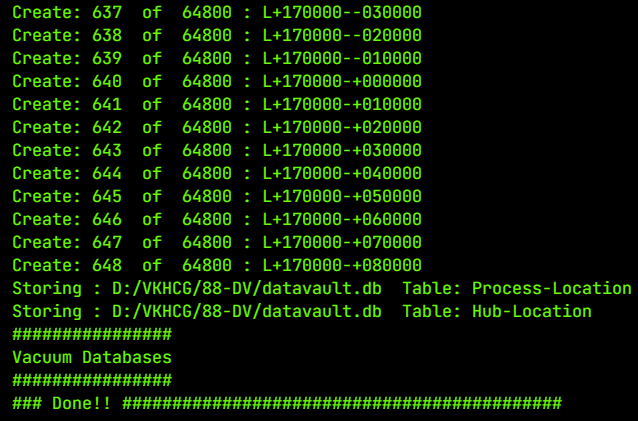
print("################")

################################################################

print("### Done!! ############################################")

################################################################

**Output:**

****

**Practical No: 07**

**Aim: 7A. Write a program to perform Transform Superstep on given data**

import sys

import os

from datetime import datetime

from pytz import timezone

import pandas as pd

import sqlite3 as sq

import uuid

pd.options.mode.chained\_assignment = None

################################################################

Base = "D:/VKHCG"

print("################################")

print("Working Base :", Base, " using ", sys.platform)

print("################################")

################################################################

Company = "01-Vermeulen"

InputDir = "00-RawData"

InputFileName = "VehicleData.csv"

################################################################

sDataBaseDir = Base + "/" + Company + "/04-Transform/SQLite"

if not os.path.exists(sDataBaseDir):

os.makedirs(sDataBaseDir)

################################################################

sDatabaseName = sDataBaseDir + "/Vermeulen.db"

conn1 = sq.connect(sDatabaseName)

################################################################

sDataVaultDir = Base + "/88-DV"

if not os.path.exists(sDataVaultDir):

os.makedirs(sDataVaultDir)

################################################################

sDatabaseName = sDataVaultDir + "/datavault.db"

conn2 = sq.connect(sDatabaseName)

################################################################

sDataWarehouseDir = Base + "/99-DW"

if not os.path.exists(sDataWarehouseDir):

os.makedirs(sDataWarehouseDir)

################################################################

sDatabaseName = sDataWarehouseDir + "/datawarehouse.db"

conn3 = sq.connect(sDatabaseName)

################################################################

print("\n#################################")

print("Time Category")

print("UTC Time")

BirthDateUTC = datetime(1960, 12, 20, 10, 15, 0)

BirthDateZoneUTC = BirthDateUTC.replace(tzinfo=timezone("UTC"))

BirthDateZoneStr = BirthDateZoneUTC.strftime("%Y-%m-%d %H:%M:%S")

BirthDateZoneUTCStr = BirthDateZoneUTC.strftime("%Y-%m-%d %H:%M:%S

(%Z) (%z)")

print(BirthDateZoneUTCStr)

print("#################################")

print("Birth Date in Reykjavik :")

BirthZone = "Atlantic/Reykjavik"

BirthDate = BirthDateZoneUTC.astimezone(timezone(BirthZone))

BirthDateStr = BirthDate.strftime("%Y-%m-%d %H:%M:%S (%Z) (%z)")

BirthDateLocal = BirthDate.strftime("%Y-%m-%d %H:%M:%S")

print(BirthDateStr)

print("#################################")

################################################################

IDZoneNumber = str(uuid.uuid4())

sDateTimeKey = BirthDateZoneStr.replace(" ", "-").replace(":", "-")

TimeLine = [

("ZoneBaseKey", ["UTC"]),

("IDNumber", [IDZoneNumber]),

("DateTimeKey", [sDateTimeKey]),

("UTCDateTimeValue", [BirthDateZoneUTC]),

("Zone", [BirthZone]),

("DateTimeValue", [BirthDateStr]),

]

TimeFrame = pd.DataFrame.from\_dict(dict(TimeLine))

################################################################

TimeHub = TimeFrame[["IDNumber", "ZoneBaseKey", "DateTimeKey",

"DateTimeValue"]]

TimeHubIndex = TimeHub.set\_index(["IDNumber"], inplace=False)

################################################################

sTable = "Hub-Time-Gunnarsson"

print("\n#################################")

print("Storing :", sDatabaseName, "\n Table:", sTable)

print("\n#################################")

TimeHubIndex.to\_sql(sTable, conn2, if\_exists="replace")

sTable = "Dim-Time-Gunnarsson"

TimeHubIndex.to\_sql(sTable, conn3, if\_exists="replace")

################################################################

TimeSatellite = TimeFrame[["IDNumber", "DateTimeKey", "Zone",

"DateTimeValue"]]

TimeSatelliteIndex = TimeSatellite.set\_index(["IDNumber"],

inplace=False)

################################################################

BirthZoneFix = BirthZone.replace(" ", "-").replace("/", "-")

sTable = "Satellite-Time-" + BirthZoneFix + "-Gunnarsson"

print("\n#################################")

print("Storing :", sDatabaseName, "\n Table:", sTable)

print("\n#################################")

TimeSatelliteIndex.to\_sql(sTable, conn2, if\_exists="replace")

sTable = "Dim-Time-" + BirthZoneFix + "-Gunnarsson"

TimeSatelliteIndex.to\_sql(sTable, conn3, if\_exists="replace")

################################################################

print("\n#################################")

print("Person Category")

FirstName = "Guðmundur"

LastName = "Gunnarsson"

print("Name:", FirstName, LastName)

print("Birth Date:", BirthDateLocal)

print("Birth Zone:", BirthZone)

print("UTC Birth Date:", BirthDateZoneStr)

print("#################################")

###############################################################

IDPersonNumber = str(uuid.uuid4())

PersonLine = [

("IDNumber", [IDPersonNumber]),

("FirstName", [FirstName]),

("LastName", [LastName]),

("Zone", ["UTC"]),

("DateTimeValue", [BirthDateZoneStr]),

]

PersonFrame = pd.DataFrame.from\_dict(dict(PersonLine))

################################################################

TimeHub = PersonFrame

TimeHubIndex = TimeHub.set\_index(["IDNumber"], inplace=False)

################################################################

sTable = "Hub-Person-Gunnarsson"

print("\n#################################")

print("Storing :", sDatabaseName, "\n Table:", sTable)

print("\n#################################")

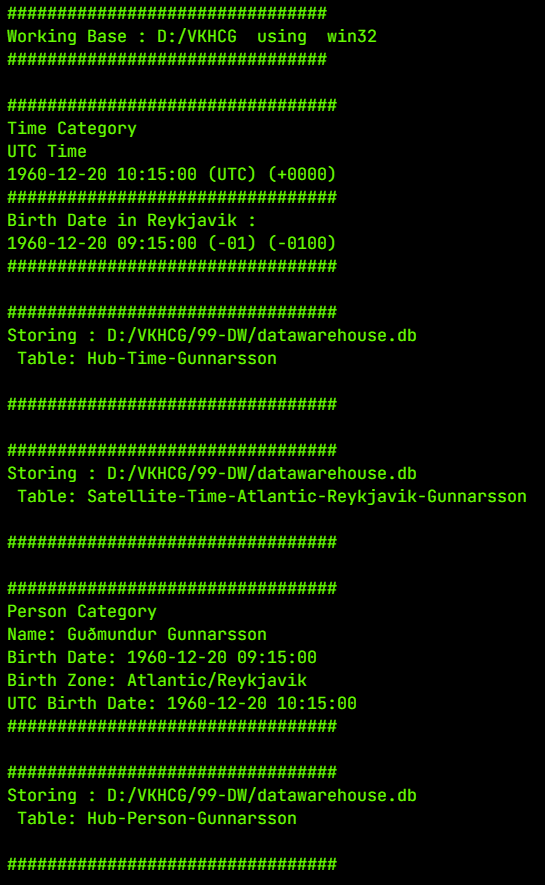
TimeHubIndex.to\_sql(sTable, conn2, if\_exists="replace")

sTable = "Dim-Person-Gunnarsson"

TimeHubIndex.to\_sql(sTable, conn3, if\_exists="replace")

################################################################

**Output:**

****

**7B. Write a program to build dimension Person, dimension Time, and factPersonBornAtTime**

import os

import sqlite3 as sq

import sys

import uuid

from datetime import datetime

import pandas as pd

from pytz import timezone

pd.options.mode.chained\_assignment = None

################################################################

Base = "D:/VKHCG"

print("################################")

print("Working Base :", Base, " using ", sys.platform)

print("################################")

################################################################

Company = "01-Vermeulen"

################################################################

sDataBaseDir = Base + "/" + Company + "/04-Transform/SQLite"

if not os.path.exists(sDataBaseDir):

os.makedirs(sDataBaseDir)

################################################################

sDatabaseName = sDataBaseDir + "/Vermeulen.db"

conn1 = sq.connect(sDatabaseName)

################################################################

sDataWarehousetDir = Base + "/99-DW"

if not os.path.exists(sDataWarehousetDir):

os.makedirs(sDataWarehousetDir)

################################################################

sDatabaseName = sDataWarehousetDir + "/datawarehouse.db"

conn2 = sq.connect(sDatabaseName)

################################################################

print("\n#################################")

print("Time Dimension")

BirthZone = "Atlantic/Reykjavik"

BirthDateUTC = datetime(1960, 12, 20, 10, 15, 0)

BirthDateZoneUTC = BirthDateUTC.replace(tzinfo=timezone("UTC"))

BirthDateZoneStr = BirthDateZoneUTC.strftime("%Y-%m-%d %H:%M:%S")

BirthDateZoneUTCStr = BirthDateZoneUTC.strftime("%Y-%m-%d %H:%M:%S

(%Z) (%z)")

BirthDate = BirthDateZoneUTC.astimezone(timezone(BirthZone))

BirthDateStr = BirthDate.strftime("%Y-%m-%d %H:%M:%S (%Z) (%z)")

BirthDateLocal = BirthDate.strftime("%Y-%m-%d %H:%M:%S")

################################################################

IDTimeNumber = str(uuid.uuid4())

TimeLine = [

("TimeID", [IDTimeNumber]),

("UTCDate", [BirthDateZoneStr]),

("LocalTime", [BirthDateLocal]),

("TimeZone", [BirthZone]),

]

TimeFrame = pd.DataFrame.from\_dict(dict(TimeLine))

################################################################

DimTime = TimeFrame

DimTimeIndex = DimTime.set\_index(["TimeID"], inplace=False)

################################################################

sTable = "Dim-Time"

print("\n#################################")

print("Storing :", sDatabaseName, "\n Table:", sTable)

print("\n#################################")

DimTimeIndex.to\_sql(sTable, conn1, if\_exists="replace")

DimTimeIndex.to\_sql(sTable, conn2, if\_exists="replace")

################################################################

print("\n#################################")

print("Dimension Person")

print("\n#################################")

FirstName = "Guðmundur"

LastName = "Gunnarsson"

###############################################################

IDPersonNumber = str(uuid.uuid4())

PersonLine = [

("PersonID", [IDPersonNumber]),

("FirstName", [FirstName]),

("LastName", [LastName]),

("Zone", ["UTC"]),

("DateTimeValue", [BirthDateZoneStr]),

]

PersonFrame = pd.DataFrame.from\_dict(dict(PersonLine))

################################################################

DimPerson = PersonFrame

DimPersonIndex = DimPerson.set\_index(["PersonID"], inplace=False)

################################################################

sTable = "Dim-Person"

print("\n#################################")

print("Storing :", sDatabaseName, "\n Table:", sTable)

print("\n#################################")

DimPersonIndex.to\_sql(sTable, conn1, if\_exists="replace")

DimPersonIndex.to\_sql(sTable, conn2, if\_exists="replace")

################################################################

print("\n#################################")

print("Fact - Person - time")

print("\n#################################")

IDFactNumber = str(uuid.uuid4())

PersonTimeLine = [

("IDNumber", [IDFactNumber]),

("IDPersonNumber", [IDPersonNumber]),

("IDTimeNumber", [IDTimeNumber]),

]

PersonTimeFrame = pd.DataFrame.from\_dict(dict(PersonTimeLine))

################################################################

FctPersonTime = PersonTimeFrame

FctPersonTimeIndex = FctPersonTime.set\_index(["IDNumber"],

inplace=False)

################################################################

sTable = "Fact-Person-Time"

print("\n#################################")

print("Storing :", sDatabaseName, "\n Table:", sTable)

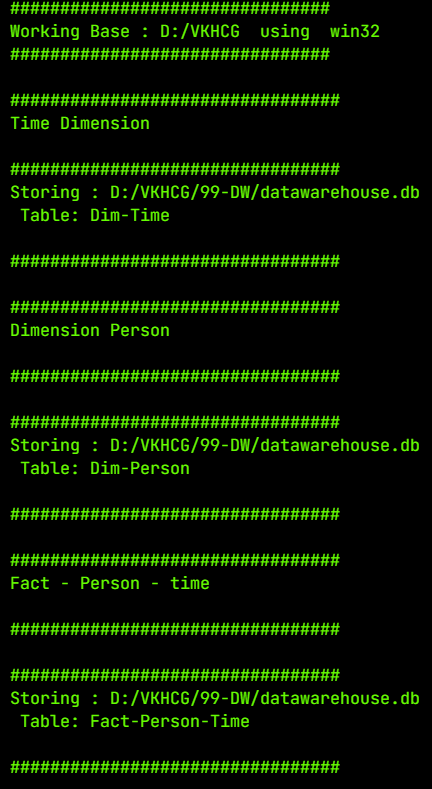
print("\n#################################")

FctPersonTimeIndex.to\_sql(sTable, conn1, if\_exists="replace")

FctPersonTimeIndex.to\_sql(sTable, conn2, if\_exists="replace")

################################################################

**Output:**



**Practical No: 08**

**Aim: 8A. Write a program to perform horizontal-style slicing orsubsetting of the data warehouse**

import sys

import os

import pandas as pd

import sqlite3 as sq

################################################################

Base = "D:/VKHCG"

print("################################")

print("Working Base :", Base, " using ", sys.platform)

print("################################")

################################################################

################################################################

Company = "01-Vermeulen"

################################################################

sDataWarehouseDir = Base + "/99-DW"

if not os.path.exists(sDataWarehouseDir):

os.makedirs(sDataWarehouseDir)

################################################################

sDatabaseName = sDataWarehouseDir + "/datawarehouse.db"

conn1 = sq.connect(sDatabaseName)

################################################################

sDatabaseName = sDataWarehouseDir + "/datamart.db"

conn2 = sq.connect(sDatabaseName)

################################################################

print("################")

sTable = "Dim-BMI"

print("Loading :", sDatabaseName, " Table:", sTable)

sSQL = "SELECT \* FROM [Dim-BMI];"

PersonFrame0 = pd.read\_sql\_query(sSQL, conn1)

print("################")

sTable = "Dim-BMI"

print("Loading :", sDatabaseName, " Table:", sTable)

sSQL = "SELECT PersonID, \

Height,\

Weight,\

bmi,\

Indicator\

FROM [Dim-BMI]\

WHERE \

Height > 1.5 \

and Indicator = 1 \

ORDER BY \

Height,\

Weight;"

PersonFrame1 = pd.read\_sql\_query(sSQL, conn1)

################################################################

DimPerson = PersonFrame1

DimPersonIndex = DimPerson.set\_index(["PersonID"], inplace=False)

################################################################

sTable = "Dim-BMI"

print("\n#################################")

print("Storing :", sDatabaseName, "\n Table:", sTable)

print("\n#################################")

# DimPersonIndex.to\_sql(sTable, conn2, if\_exists="replace")

################################################################

print("################")

sTable = "Dim-BMI"

print("Loading :", sDatabaseName, " Table:", sTable)

sSQL = "SELECT \* FROM [Dim-BMI];"

PersonFrame2 = pd.read\_sql\_query(sSQL, conn2)

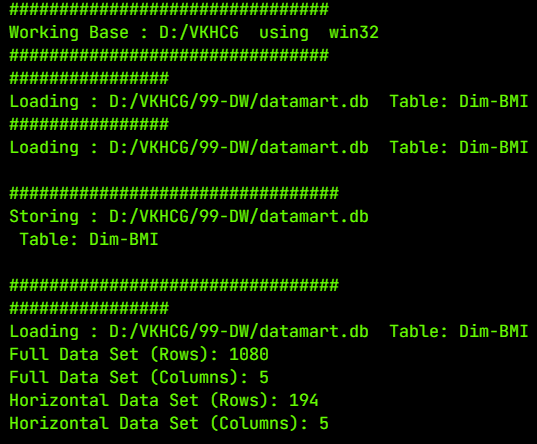
print("Full Data Set (Rows):", PersonFrame0.shape[0])

print("Full Data Set (Columns):", PersonFrame0.shape[1])

print("Horizontal Data Set (Rows):", PersonFrame2.shape[0])

print("Horizontal Data Set (Columns):", PersonFrame2.shape[1])

**Output:**



**8B. Write a program to perform association rule mining on the given data**

import os

import sys

import pandas as pd

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

################################################################

Base = "D:/VKHCG"

print("################################")

print("Working Base :", Base, " using ", sys.platform)

print("################################")

Company = "01-Vermeulen"

InputFileName = "Online-Retail-Billboard.xlsx"

EDSAssessDir = "02-Assess/01-EDS"

InputAssessDir = EDSAssessDir + "/02-Python"

################################################################

sFileAssessDir = Base + "/" + Company + "/" + InputAssessDir

if not os.path.exists(sFileAssessDir):

os.makedirs(sFileAssessDir)

################################################################

sFileName = Base + "/" + Company + "/00-RawData/" + InputFileName

################################################################

df = pd.read\_excel(sFileName)

print(df.shape)

################################################################

df["Description"] = df["Description"].str.strip()

df.dropna(axis=0, subset=["InvoiceNo"], inplace=True)

df["InvoiceNo"] = df["InvoiceNo"].astype("str")

df = df[~df["InvoiceNo"].str.contains("C")]

basket = (

df[df["Country"] == "France"]

.groupby(["InvoiceNo", "Description"])["Quantity"]

.sum()

.unstack()

.reset\_index()

.fillna(0)

.set\_index("InvoiceNo"))

################################################################

def encode\_units(x):

if x <= 0:

return 0

if x >= 1:

return 1

################################################################

basket\_sets = basket.applymap(encode\_units)

basket\_sets.drop("POSTAGE", inplace=True, axis=1)

frequent\_itemsets = apriori(basket\_sets, min\_support=0.07,

use\_colnames=True)

rules = association\_rules(frequent\_itemsets, metric="lift",

min\_threshold=1)

print(rules.head())

rules[(rules["lift"] >= 6) & (rules["confidence"] >= 0.8)]

################################################################

sProduct1 = "ALARM CLOCK BAKELIKE GREEN"

print(sProduct1)

print(basket[sProduct1].sum())

sProduct2 = "ALARM CLOCK BAKELIKE RED"

print(sProduct2)

print(basket[sProduct2].sum())

################################################################

basket2 = (df[df["Country"] == "Germany"]

.groupby(["InvoiceNo", "Description"])["Quantity"]

.sum()

.unstack()

.reset\_index()

.fillna(0)

.set\_index("InvoiceNo"))

basket\_sets2 = basket2.applymap(encode\_units)

basket\_sets2.drop("POSTAGE", inplace=True, axis=1)

frequent\_itemsets2 = apriori(basket\_sets2, min\_support=0.05,

use\_colnames=True)

rules2 = association\_rules(frequent\_itemsets2, metric="lift",

min\_threshold=1)

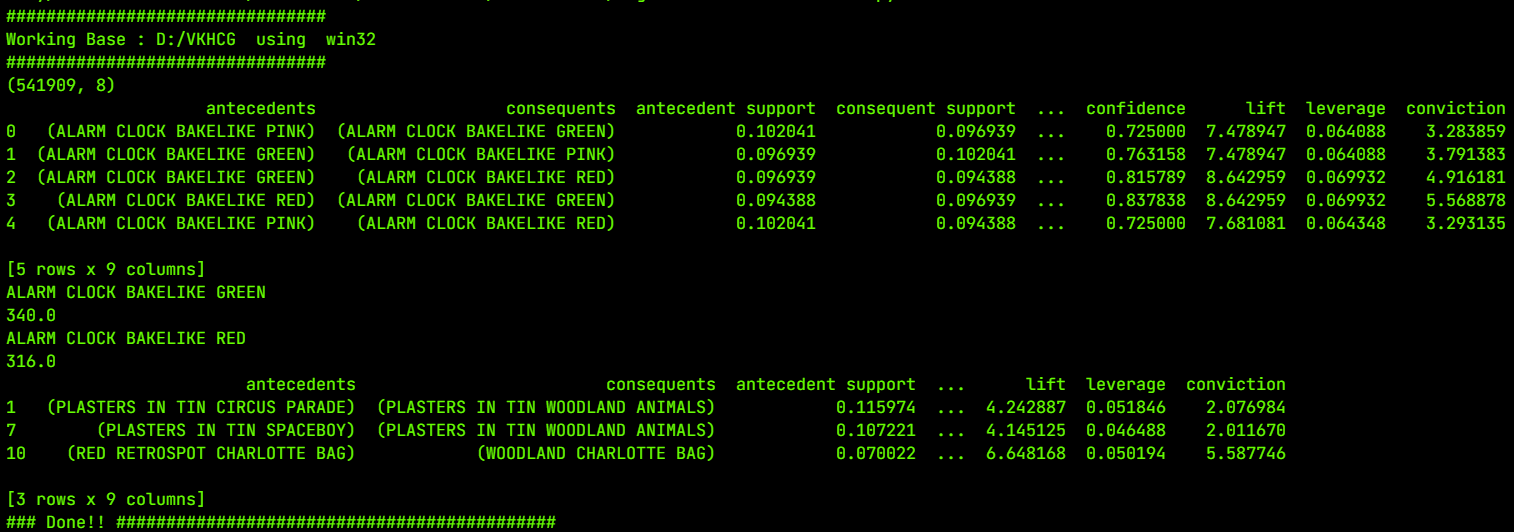
print(rules2[(rules2["lift"] >= 4) & (rules2["confidence"] >= 0.5)])

################################################################

print("### Done!! ############################################")

################################################################

**Output:**



**8C. Write a program to create a Network Routing Diagram using given data.**

import sys

import matplotlib.pyplot as plt

import networkx as nx

import pandas as pd

################################################################

pd.options.mode.chained\_assignment = None

################################################################

Base = "D:/VKHCG"

################################################################

print("################################")

print("Working Base :", Base, " using ", sys.platform)

print("################################")

################################################################

sInputFileName = "02-Assess/01-EDS/02-Python/Assess-Network-Routing-

Company.csv"

################################################################

sOutputFileName1 = "05-Organise/01-EDS/02-Python/Organise-Network-

Routing-Company.gml"

sOutputFileName2 = "05-Organise/01-EDS/02-Python/Organise-Network-

Routing-Company.png"

Company = "01-Vermeulen"

################################################################

################################################################

### Import Country Data

################################################################

sFileName = Base + "/" + Company + "/" + sInputFileName

print("################################")

print("Loading :", sFileName)

print("################################")

CompanyData = pd.read\_csv(sFileName, header=0, low\_memory=False,

encoding="latin-1")

print("################################")

################################################################

print(CompanyData.head())

print(CompanyData.shape)

################################################################

G = nx.Graph()

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

for j in range(CompanyData.shape[0]):

Node0 = CompanyData["Company\_Country\_Name"][i]

Node1 = CompanyData["Company\_Country\_Name"][j]

if Node0 != Node1:

G.add\_edge(Node0, Node1)

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

Node0 = CompanyData["Company\_Country\_Name"][i]

Node1 = (

CompanyData["Company\_Place\_Name"][i]

+ "("

+ CompanyData["Company\_Country\_Name"][i]

+ ")"

)

if Node0 != Node1:

G.add\_edge(Node0, Node1)

print("Nodes:", G.number\_of\_nodes())

print("Edges:", G.number\_of\_edges())

################################################################

sFileName = Base + "/" + Company + "/" + sOutputFileName1

print("################################")

print("Storing :", sFileName)

print("################################")

nx.write\_gml(G, sFileName)

################################################################

sFileName = Base + "/" + Company + "/" + sOutputFileName2

print("################################")

print("Storing Graph Image:", sFileName)

print("################################")

plt.figure(figsize=(15, 15))

pos = nx.spectral\_layout(G, dim=2)

nx.draw\_networkx\_nodes(G, pos, node\_color="k", node\_size=10,

alpha=0.8)

nx.draw\_networkx\_edges(G, pos, edge\_color="r", arrows=False,

style="dashed")

nx.draw\_networkx\_labels(G, pos, font\_size=12, font\_family="sans-

serif", font\_color="b")

plt.axis("off")

plt.savefig(sFileName, dpi=600)

plt.show()

################################################################

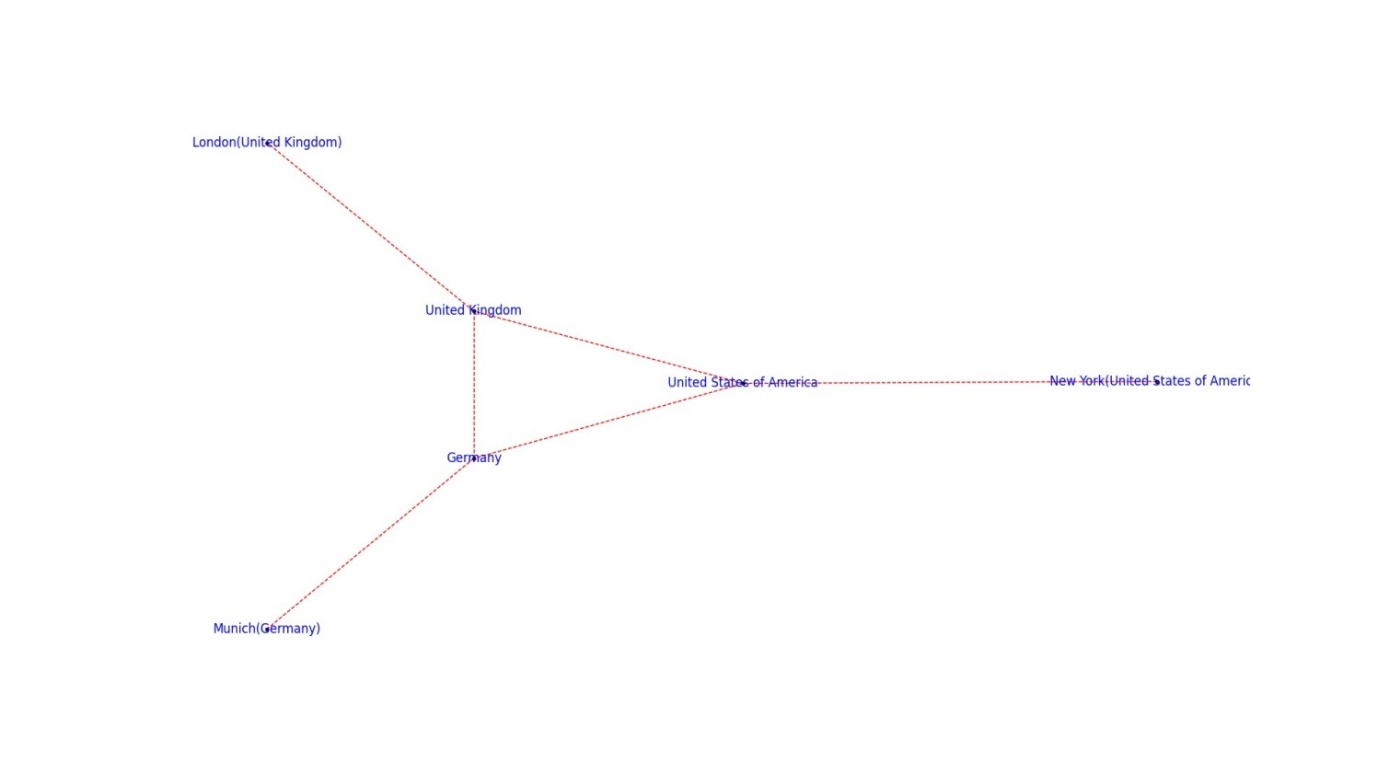
print("################################")

print("### Done!! #####################")

print("################################")

################################################################

**Output:**



**Practical No: 09**

**Aim: Generating Data**

**9A. Write a program to perform Report Superstep on Vermeulen PLC**

import sys

import pandas as pd

import networkx as nx

import matplotlib.pyplot as plt

################################################################

pd.options.mode.chained\_assignment = None

################################################################

Base = "D:/VKHCG"

################################################################

print("################################")

print("Working Base :", Base, " using ", sys.platform)

print("################################")

################################################################

sInputFileName = "02-Assess/01-EDS/02-Python/Assess-Network-Routing-

Customer.csv"

################################################################

sOutputFileName1 = "06-Report/01-EDS/02-Python/Report-Network-Routing-

Customer.gml"

sOutputFileName2 = "06-Report/01-EDS/02-Python/Report-Network-Routing-

Customer.png"

Company = "01-Vermeulen"

################################################################

################################################################

### Import Country Data

################################################################

sFileName = Base + "/" + Company + "/" + sInputFileName

print("################################")

print("Loading :", sFileName)

print("################################")

CustomerDataRaw = pd.read\_csv(sFileName, header=0, low\_memory=False,

encoding="latin-1")

CustomerData = CustomerDataRaw.head(100)

print("Loaded Country:", CustomerData.columns.values)

print("################################")

################################################################

print(CustomerData.head())

################################################################

G = nx.Graph()

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

for j in range(CustomerData.shape[0]):

Node0 = CustomerData["Customer\_Country\_Name"][i]

Node1 = CustomerData["Customer\_Country\_Name"][j]

if Node0 != Node1:

G.add\_edge(Node0, Node1)

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

Node0 = CustomerData["Customer\_Country\_Name"][i]

Node1 = (

CustomerData["Customer\_Place\_Name"][i]

+ "("

+ CustomerData["Customer\_Country\_Name"][i]

+ ")"

)

Node2 = (

"("

+ "{:.9f}".format(CustomerData["Customer\_Latitude"][i])

+ ")\

("

+ "{:.9f}".format(CustomerData["Customer\_Longitude"][i])

+ ")"

)

if Node0 != Node1:

G.add\_edge(Node0, Node1)

if Node1 != Node2:

G.add\_edge(Node1, Node2)

print("Nodes:", G.number\_of\_nodes())

print("Edges:", G.number\_of\_edges())

################################################################

sFileName = Base + "/" + Company + "/" + sOutputFileName1

print("################################")

print("Storing :", sFileName)

print("################################")

nx.write\_gml(G, sFileName)

################################################################

sFileName = Base + "/" + Company + "/" + sOutputFileName2

print("################################")

print("Storing Graph Image:", sFileName)

print("################################")

plt.figure(figsize=(25, 25))

pos = nx.spectral\_layout(G, dim=2)

nx.draw\_networkx\_nodes(G, pos, node\_color="k", node\_size=10,

alpha=0.8)

nx.draw\_networkx\_edges(G, pos, edge\_color="r", arrows=False,

style="dashed")

nx.draw\_networkx\_labels(G, pos, font\_size=12, font\_family="sans-

serif", font\_color="b")

plt.axis("off")

plt.savefig(sFileName, dpi=600)

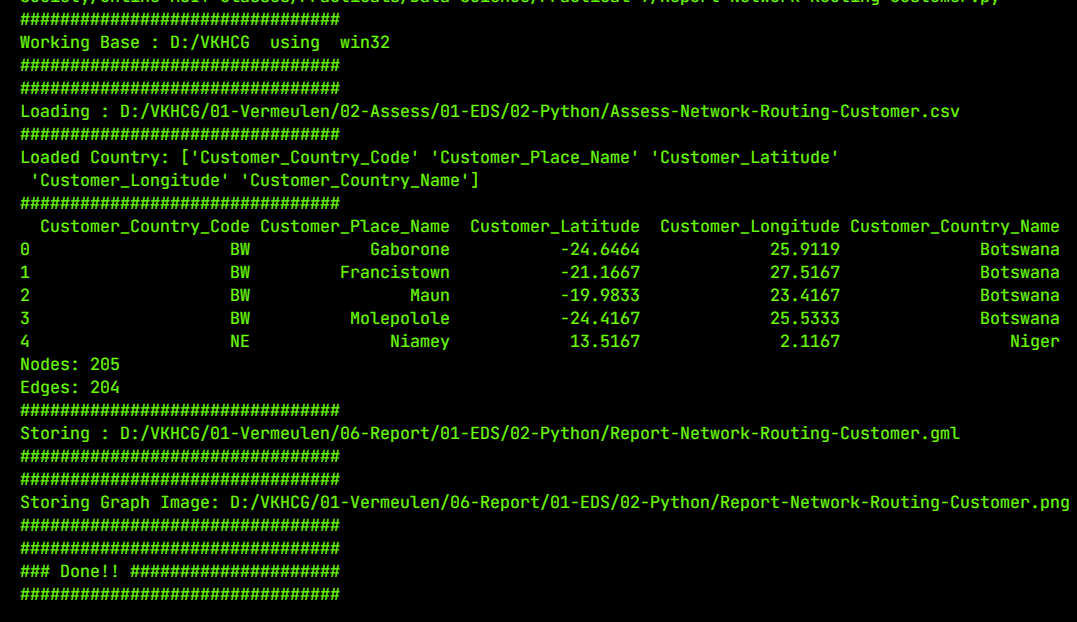
plt.show()

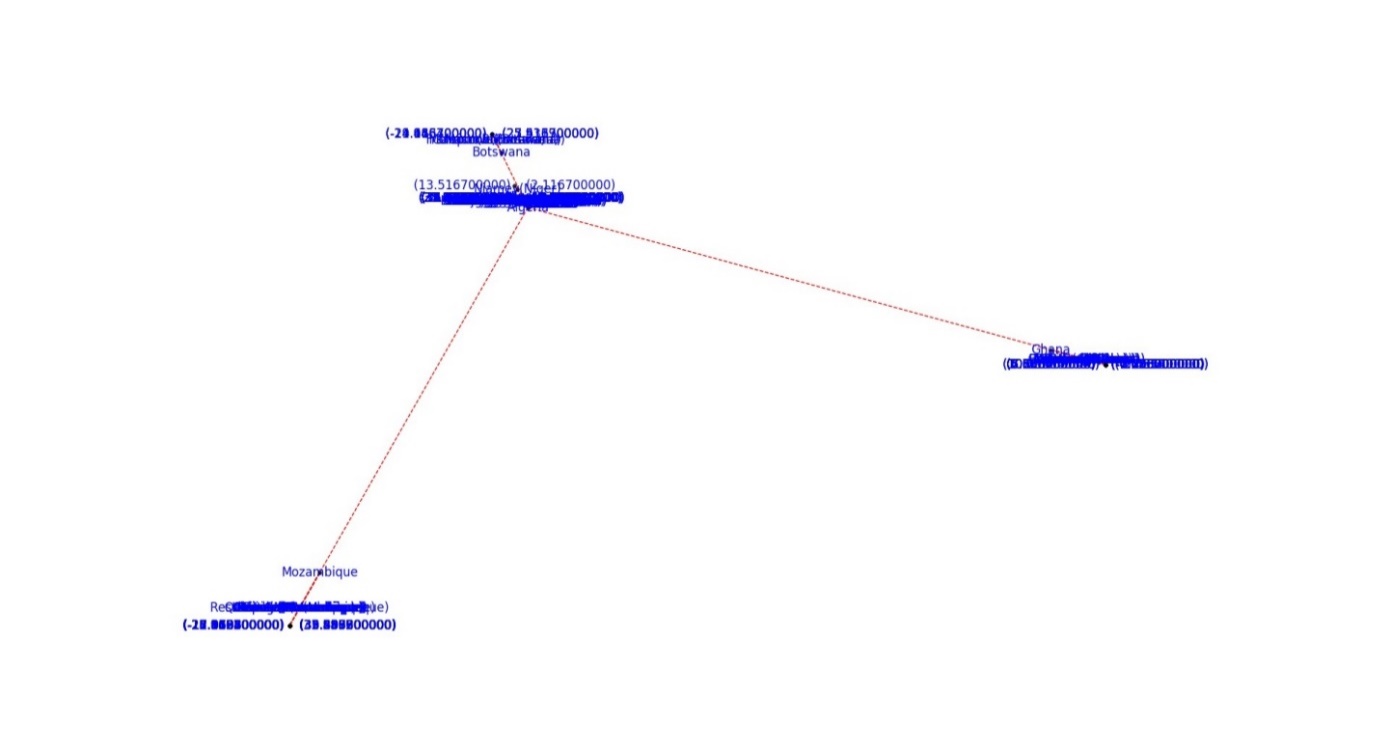
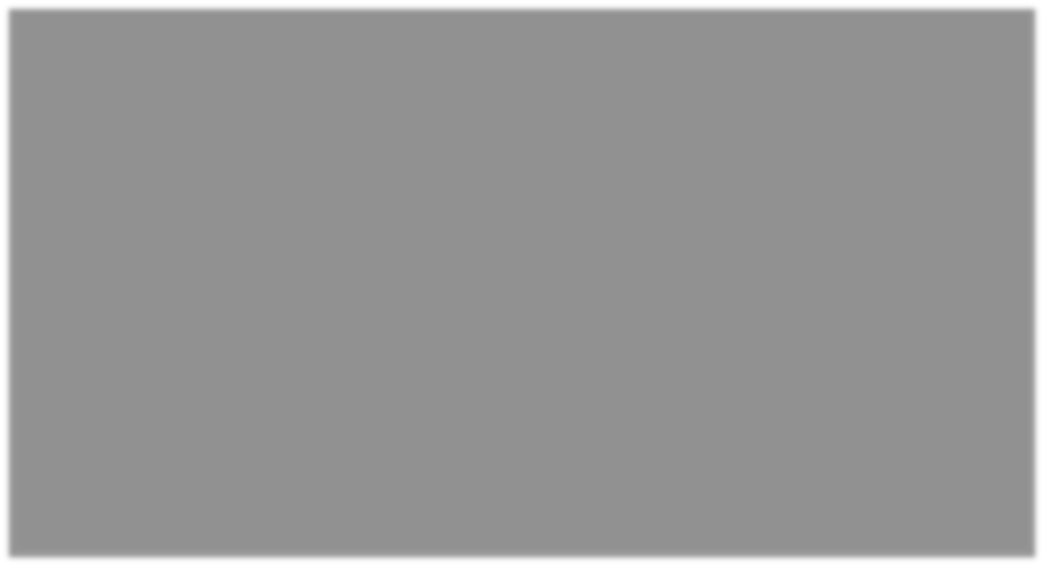
print("################################")

print("### Done!! #####################")

print("################################")

**Output:**

****



**9B. Write a program for Hillman Ltd to convert all numbers on thesides of containers into digits.**

from time import time

import matplotlib.pyplot as plt

import numpy as np

from matplotlib import offsetbox

from sklearn import (datasets, decomposition, discriminant\_analysis,

ensemble,

manifold, random\_projection)

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

X = digits.data

y = digits.targetō

n\_samples, n\_features = X.shape

n\_neighbors = 30

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

x\_min, x\_max = np.min(X, 0), np.max(X, 0)

X = (X - x\_min) / (x\_max - x\_min)

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(digits.target[i]),

color=plt.cm.Set1(y[i] / 10.0),

fontdict={"weight": "bold", "size": 9},

)

if hasattr(offsetbox, "AnnotationBbox"):

# only print thumbnails with matplotlib > 1.0

shown\_images = np.array([[1.0, 1.0]]) # just something big

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

dist = np.sum((X[i] - shown\_images) \*\* 2, 1)

if np.min(dist) < 4e-3:

# don't show points that are too close

continue

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

imagebox = offsetbox.AnnotationBbox(

offsetbox.OffsetImage(digits.images[i],

cmap=plt.cm.gray\_r), X[i]

)

ax.add\_artist(imagebox)

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

if title is not None:

plt.title(title)

n\_img\_per\_row = 20

img = np.zeros((10 \* n\_img\_per\_row, 10 \* n\_img\_per\_row))

for i in range(n\_img\_per\_row):

ix = 10 \* i + 1

for j in range(n\_img\_per\_row):

iy = 10 \* j + 1

img[ix : ix + 8, iy : iy + 8] = X[i \* n\_img\_per\_row +

j].reshape((8, 8))

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

plt.imshow(img, cmap=plt.cm.binary)

plt.xticks([])

plt.yticks([])

plt.title("A selection from the 64-dimensional digits dataset")

print("Computing random projection")

rp = random\_projection.SparseRandomProjection(n\_components=2,

random\_state=42)

X\_projected = rp.fit\_transform(X)

plot\_embedding(X\_projected, "Random Projection of the digits")

print("Computing PCA projection")

t0 = time()

X\_pca = decomposition.TruncatedSVD(n\_components=2).fit\_transform(X)

plot\_embedding(

X\_pca, "Principal Components projection of the digits (time

%.2fs)" % (time() - t0)

)

print("Computing Linear Discriminant Analysis projection")

X2 = X.copy()

X2.flat[:: X.shape[1] + 1] += 0.01 # Make X invertible

t0 = time()

X\_lda =

discriminant\_analysis.LinearDiscriminantAnalysis(n\_components=2).fit\_t

ransform(

X2, y

)

plot\_embedding(

X\_lda, "Linear Discriminant projection of the digits (time %.2fs)"

% (time() - t0)

)

print("Computing Isomap embedding")

t0 = time()

X\_iso = manifold.Isomap(n\_neighbors, n\_components=2).fit\_transform(X)

print("Done.")

plot\_embedding(X\_iso, "Isomap projection of the digits (time %.2fs)" %

(time() - t0))

print("Computing LLE embedding")

clf = manifold.LocallyLinearEmbedding(n\_neighbors, n\_components=2,

method="standard")

t0 = time()

X\_lle = clf.fit\_transform(X)

print("Done. Reconstruction error: %g" % clf.reconstruction\_error\_)

plot\_embedding(

X\_lle, "Locally Linear Embedding of the digits (time %.2fs)" %

(time() - t0)

)

print("Computing modified LLE embedding")

clf = manifold.LocallyLinearEmbedding(n\_neighbors, n\_components=2,

method="modified")

t0 = time()

X\_mlle = clf.fit\_transform(X)

print("Done. Reconstruction error: %g" % clf.reconstruction\_error\_)

plot\_embedding(

X\_mlle,

"Modified Locally Linear Embedding of the digits (time %.2fs)" %

(time() - t0),

)

print("Computing Hessian LLE embedding")

clf = manifold.LocallyLinearEmbedding(n\_neighbors, n\_components=2,

method="hessian")

t0 = time()

X\_hlle = clf.fit\_transform(X)

print("Done. Reconstruction error: %g" % clf.reconstruction\_error\_)

plot\_embedding(

X\_hlle,

"Hessian Locally Linear Embedding of the digits (time %.2fs)" %

(time() - t0),

)

print("Computing LTSA embedding")

clf = manifold.LocallyLinearEmbedding(n\_neighbors, n\_components=2,

method="ltsa")

t0 = time()

X\_ltsa = clf.fit\_transform(X)

print("Done. Reconstruction error: %g" % clf.reconstruction\_error\_)

plot\_embedding(

X\_ltsa, "Local Tangent Space Alignment of the digits (time %.2fs)"

% (time() - t0)

)

print("Computing MDS embedding")

clf = manifold.MDS(n\_components=2, n\_init=1, max\_iter=100)

t0 = time()

X\_mds = clf.fit\_transform(X)

print("Done. Stress: %f" % clf.stress\_)

plot\_embedding(X\_mds, "MDS embedding of the digits (time %.2fs)" %

(time() - t0))

print("Computing Totally Random Trees embedding")

hasher = ensemble.RandomTreesEmbedding(n\_estimators=200,

random\_state=0, max\_depth=5)

t0 = time()

X\_transformed = hasher.fit\_transform(X)

pca = decomposition.TruncatedSVD(n\_components=2)

X\_reduced = pca.fit\_transform(X\_transformed)

plot\_embedding(

X\_reduced, "Random forest embedding of the digits (time %.2fs)" %

(time() - t0)

)

print("Computing Spectral embedding")

embedder = manifold.SpectralEmbedding(

n\_components=2, random\_state=0, eigen\_solver="arpack"

)

t0 = time()

X\_se = embedder.fit\_transform(X)

plot\_embedding(X\_se, "Spectral embedding of the digits (time %.2fs)" %

(time() - t0))

print("Computing t-SNE embedding")

tsne = manifold.TSNE(n\_components=2, init="pca", random\_state=0)

t0 = time()

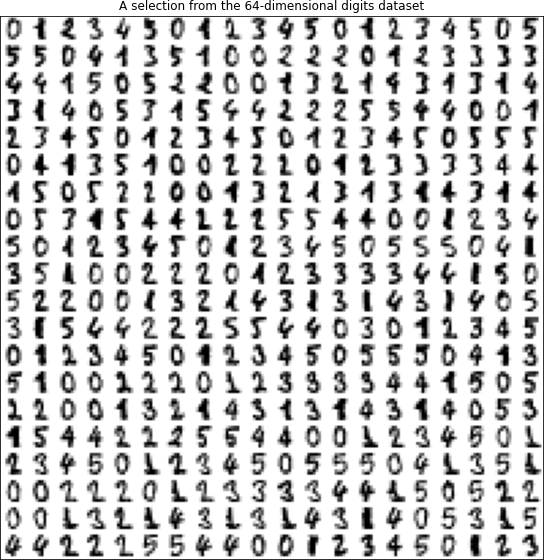
X\_tsne = tsne.fit\_transform(X)

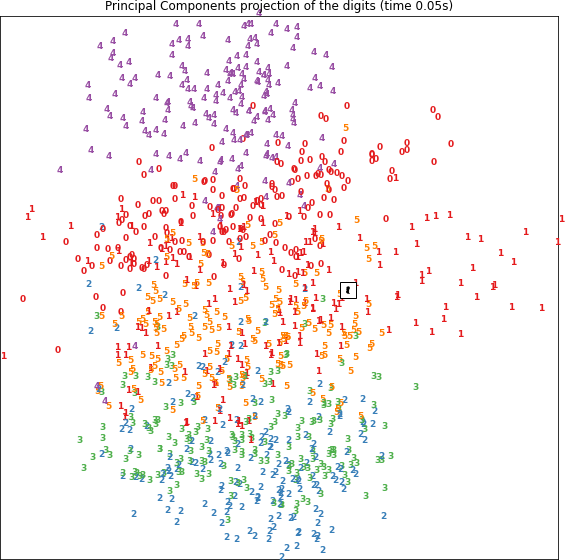
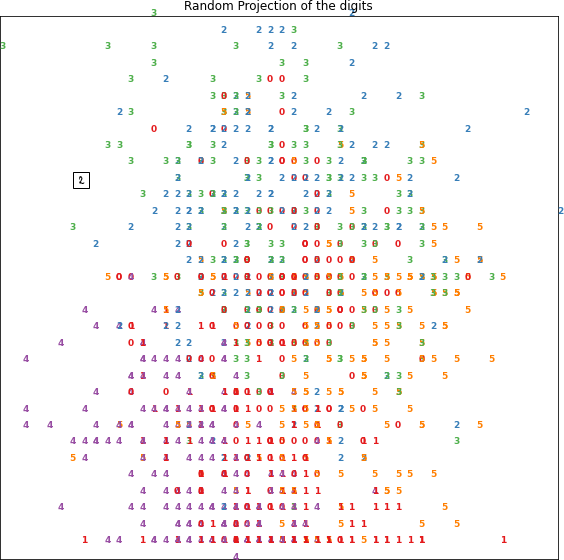
plot\_embedding(X\_tsne, "t-SNE embedding of the digits (time %.2fs)" %

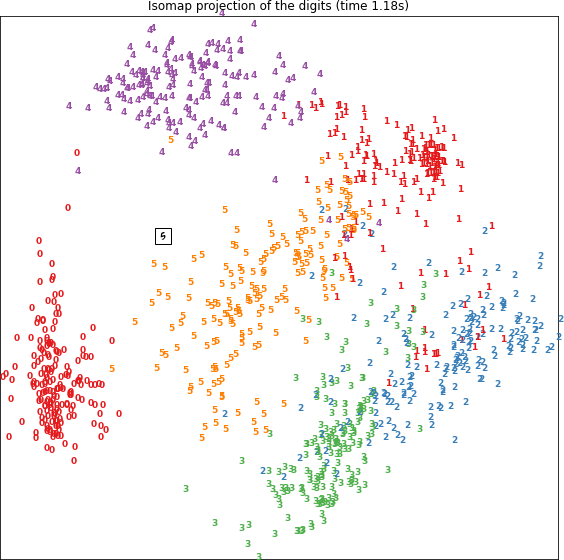
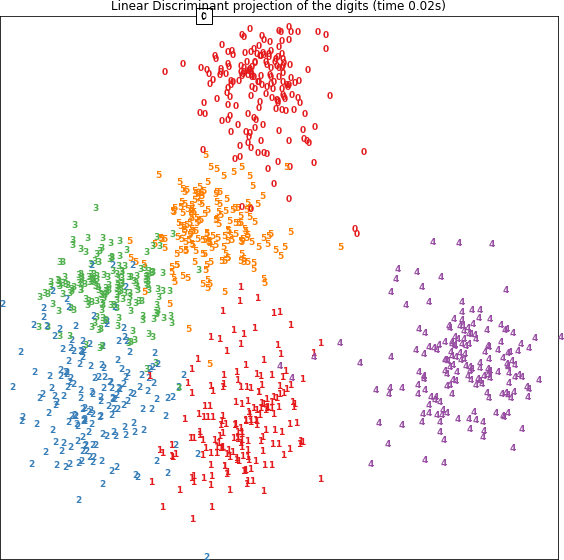
(time() - t0))

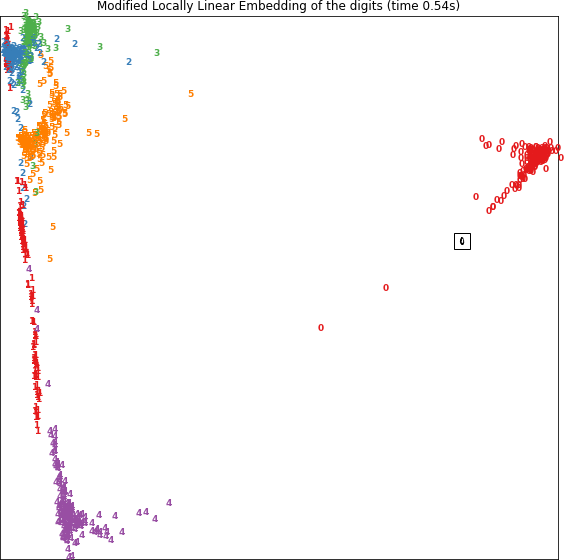
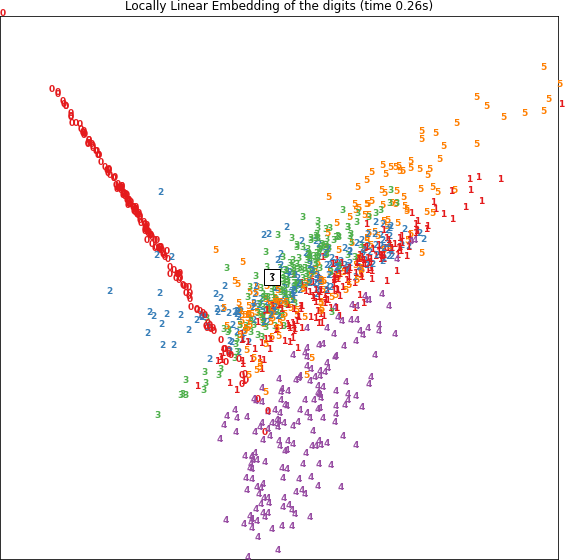
plt.show()

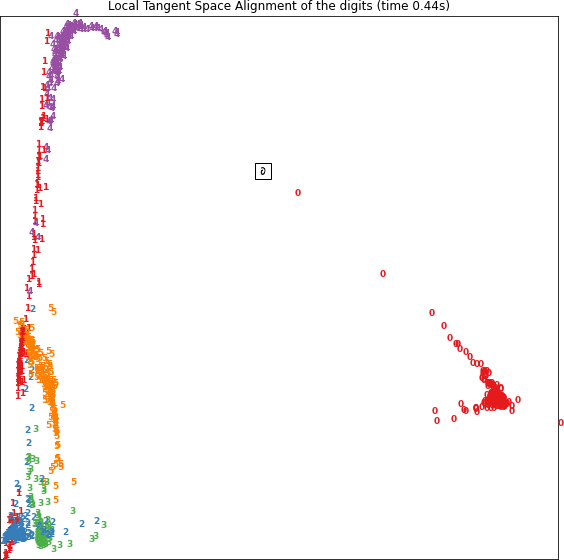
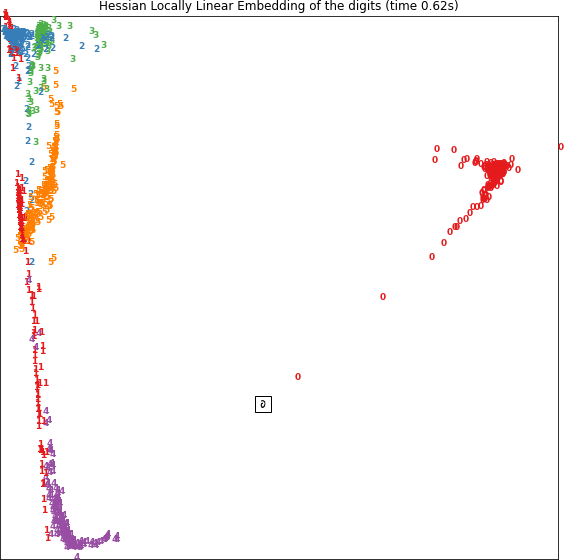
**Output:**

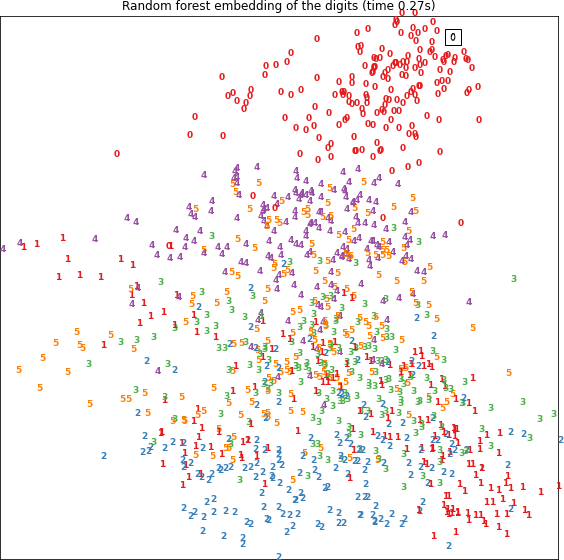
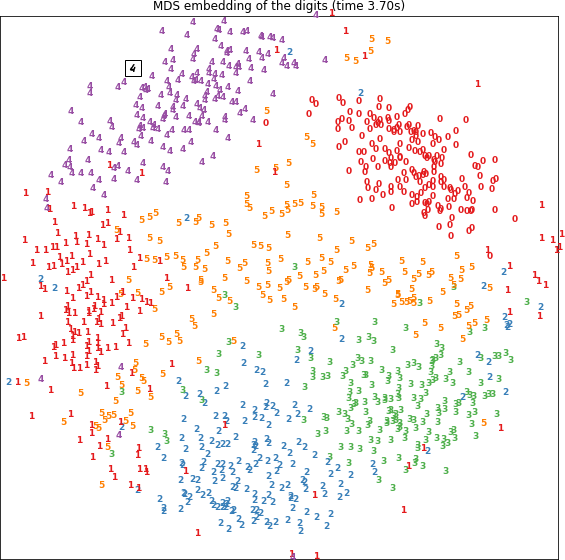


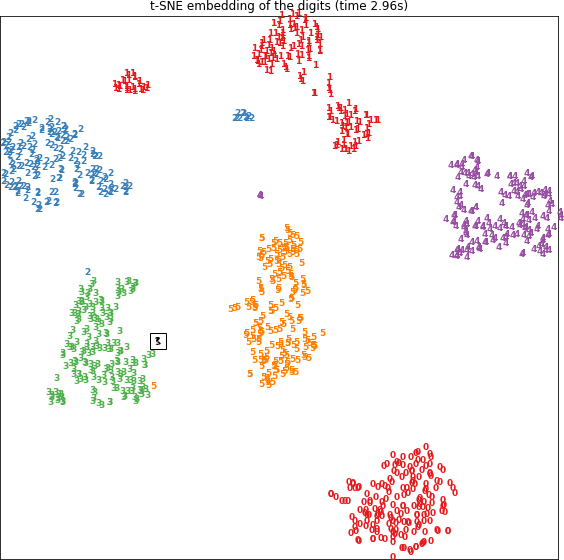
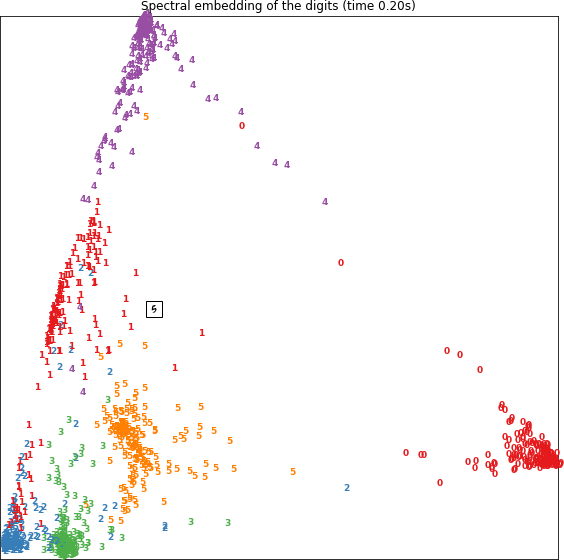












**9C. Write a program to perform Kernel Density**

import sys

import pandas as pd

import matplotlib as ml

import numpy as np

from matplotlib import pyplot as plt

################################################################

Base = "D:/VKHCG"

print("################################")

print("Working Base :", Base, " using ", sys.platform)

print("################################")

################################################################

ml.style.use("ggplot")

fig1 = plt.figure(figsize=(10, 10))

ser = pd.Series(np.random.randn(1000))

ser.plot(figsize=(10, 10), kind="kde")

sPicNameOut1 = Base + "/01-Vermeulen/06-Report/01-EDS/02-

Python/kde.png"

plt.savefig(sPicNameOut1, dpi=600)

plt.tight\_layout()

plt.show()

fig2 = plt.figure(figsize=(10, 10))

from pandas.plotting import scatter\_matrix

df = pd.DataFrame(

np.random.randn(1000, 5), columns=["Y2014", "Y2015", "Y2016",

"Y2017", "Y2018"]

)

scatter\_matrix(df, alpha=0.2, figsize=(10, 10), diagonal="kde")

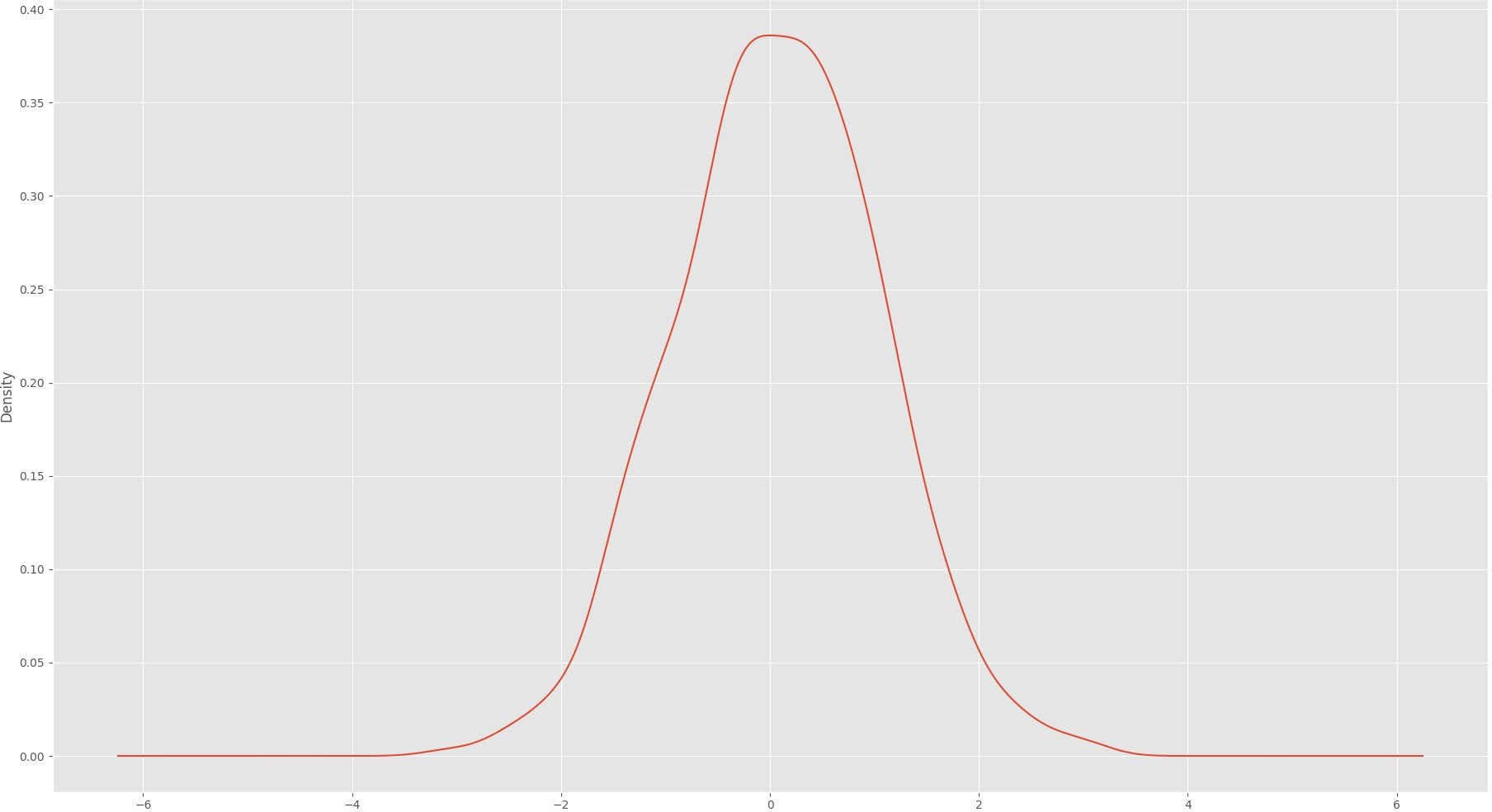
sPicNameOut2 = Base + "/01-Vermeulen/06-Report/01-EDS/02-

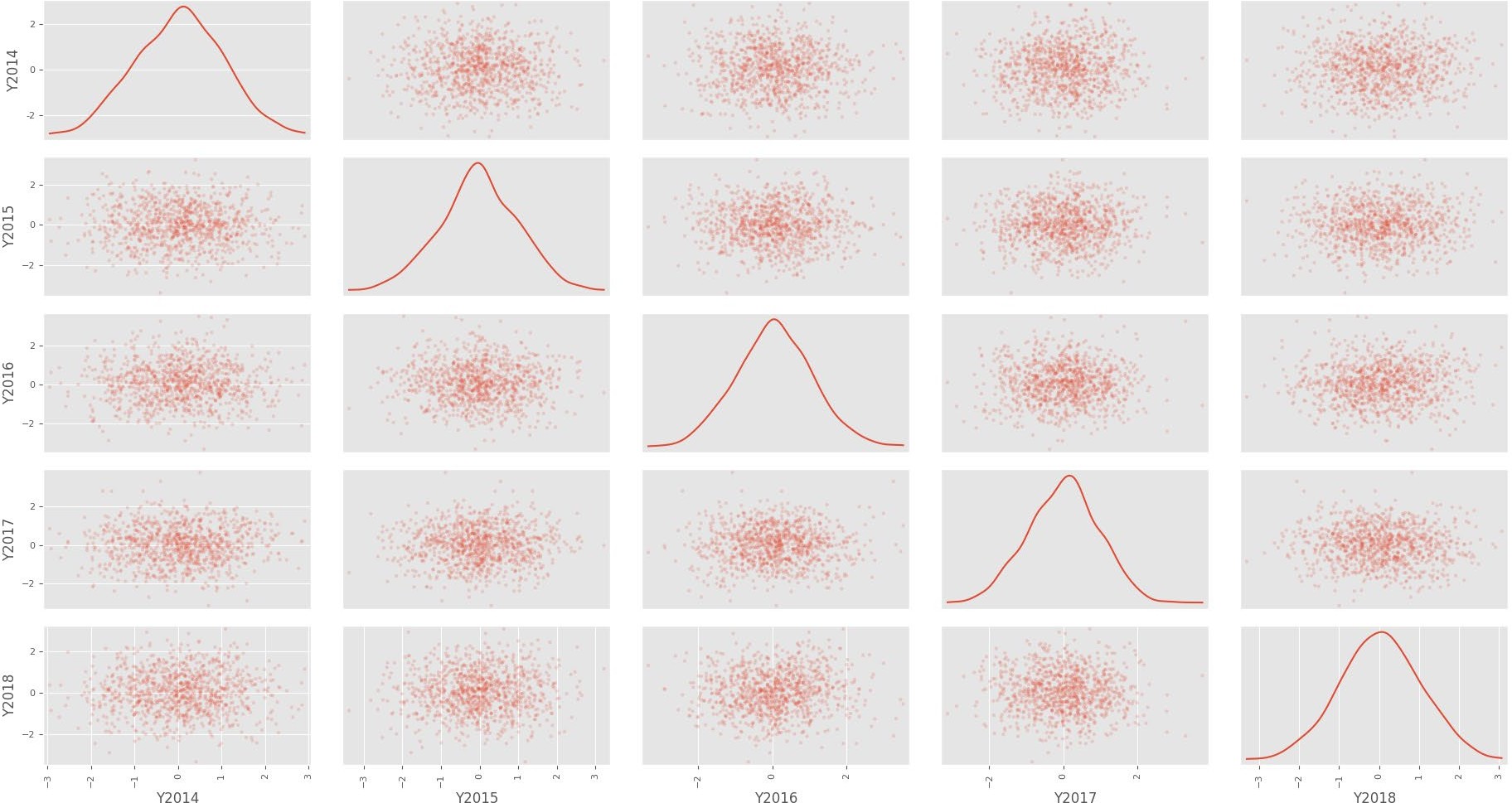
Python/scatter\_matrix.png"

plt.savefig(sPicNameOut2, dpi=600)

plt.tight\_layout()

plt.show()

 **Output:**



**9D. Write a program to plot Lag Plot, Autocorrelation, and Bootstrap Plot**

import sys

import pandas as pd

from matplotlib import style

from matplotlib import pyplot as plt

import numpy as np

################################################################

Base = "D:/VKHCG"

print("################################")

print("Working Base :", Base, " using ", sys.platform)

print("################################")

################################################################

style.use("ggplot")

from pandas.plotting import lag\_plot

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

data = pd.Series(

0.1 \* np.random.rand(1000)

+ 0.9 \* np.sin(np.linspace(-99 \* np.pi, 99 \* np.pi, num=1000))

)

lag\_plot(data)

sPicNameOut1 = Base + "/01-Vermeulen/06-Report/01-EDS/02-

Python/lag\_plot.png"

plt.savefig(sPicNameOut1, dpi=600)

plt.tight\_layout()

plt.show()

from pandas.plotting import autocorrelation\_plot

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

data = pd.Series(

0.7 \* np.random.rand(1000)

+ 0.3 \* np.sin(np.linspace(-9 \* np.pi, 9 \* np.pi, num=1000))

)

autocorrelation\_plot(data)

sPicNameOut2 = (

Base + "/01-Vermeulen/06-Report/01-EDS/02-

Python/autocorrelation\_plot.png"

)

plt.savefig(sPicNameOut2, dpi=600)

plt.tight\_layout()

plt.show()

from pandas.plotting import bootstrap\_plot

data = pd.Series(np.random.rand(1000))

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

bootstrap\_plot(data, size=50, samples=500, color="grey")

sPicNameOut3 = Base + "/01-Vermeulen/06-Report/01-EDS/02-

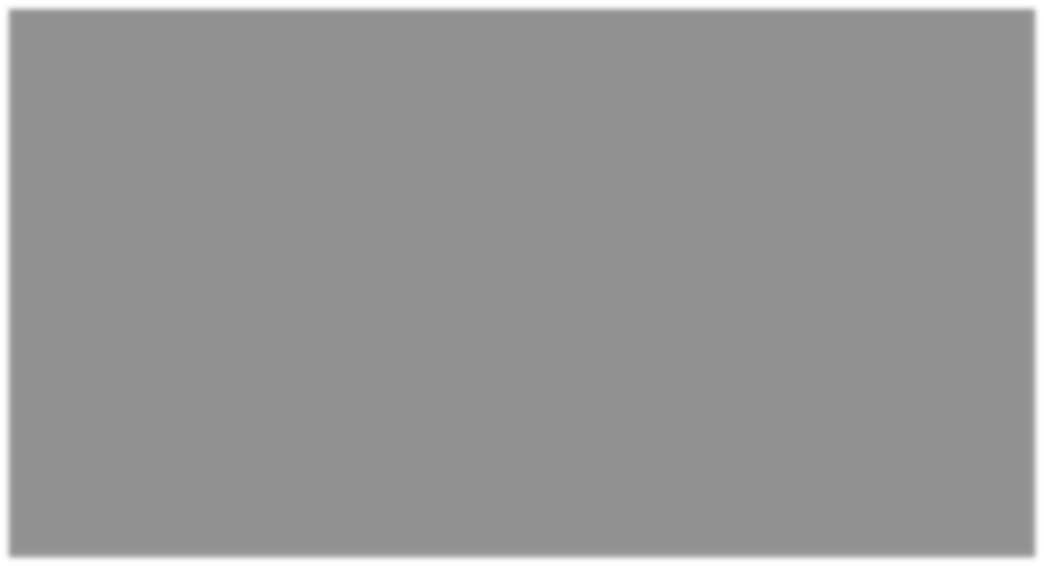
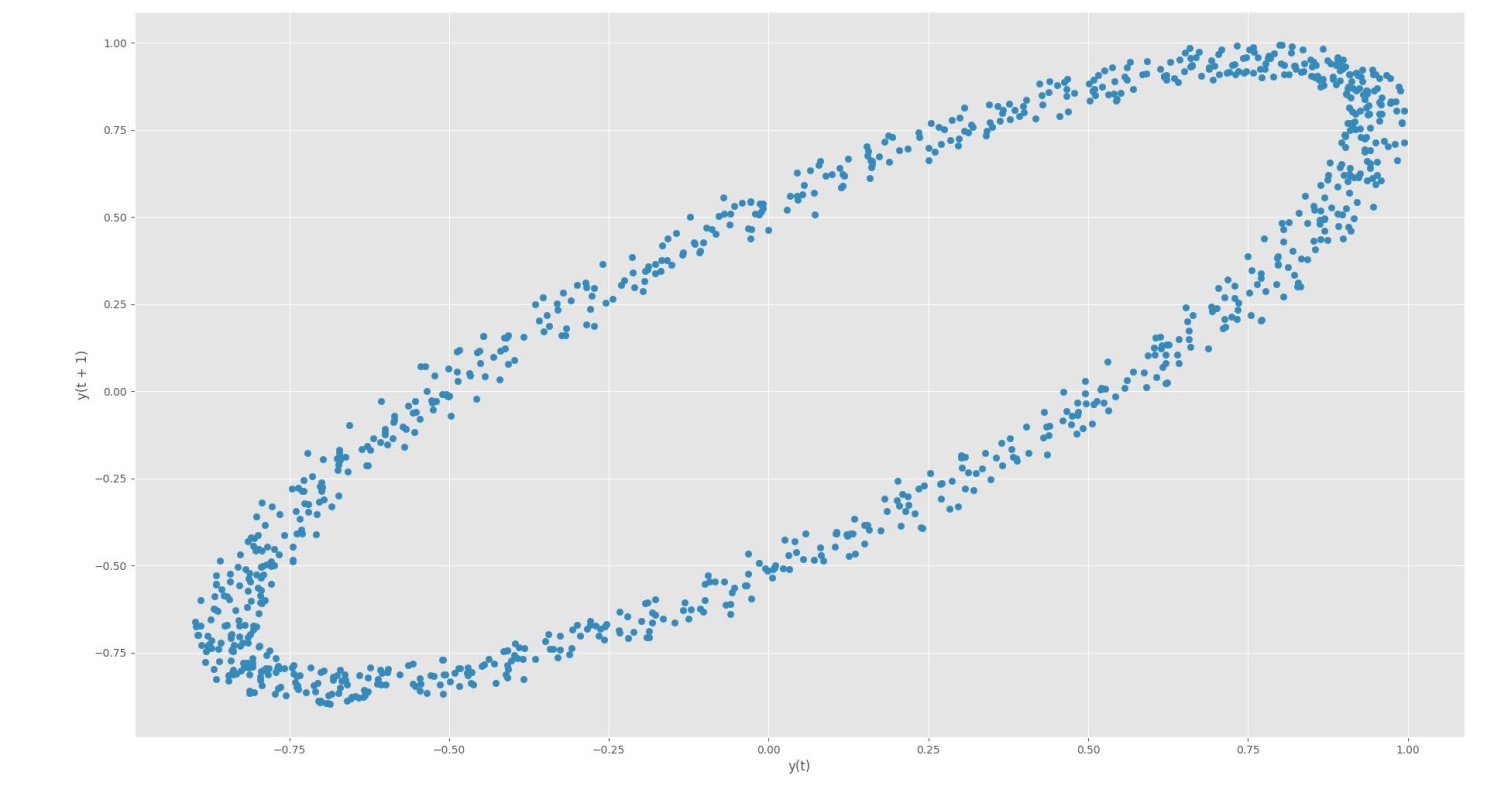
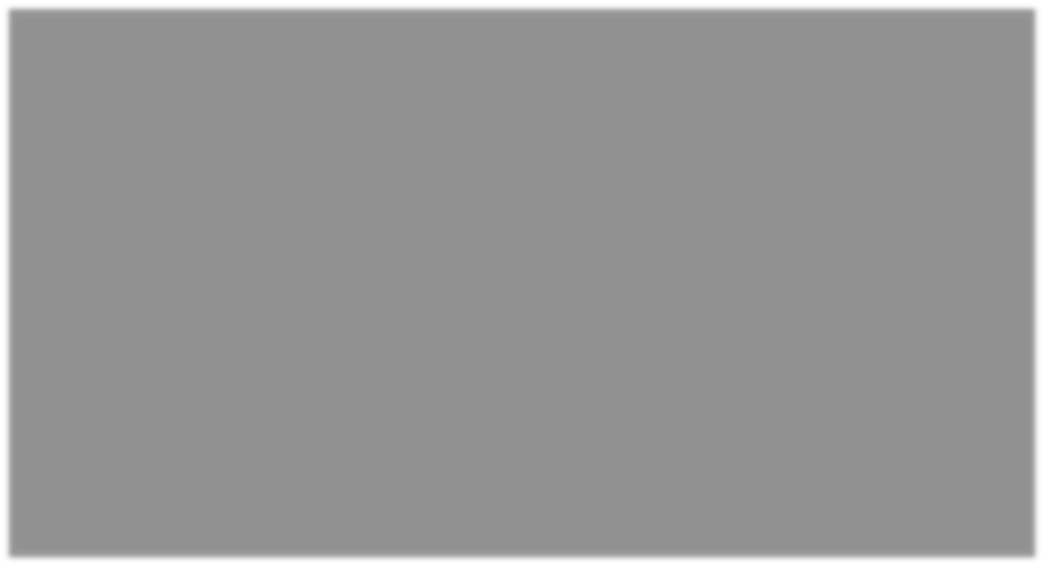
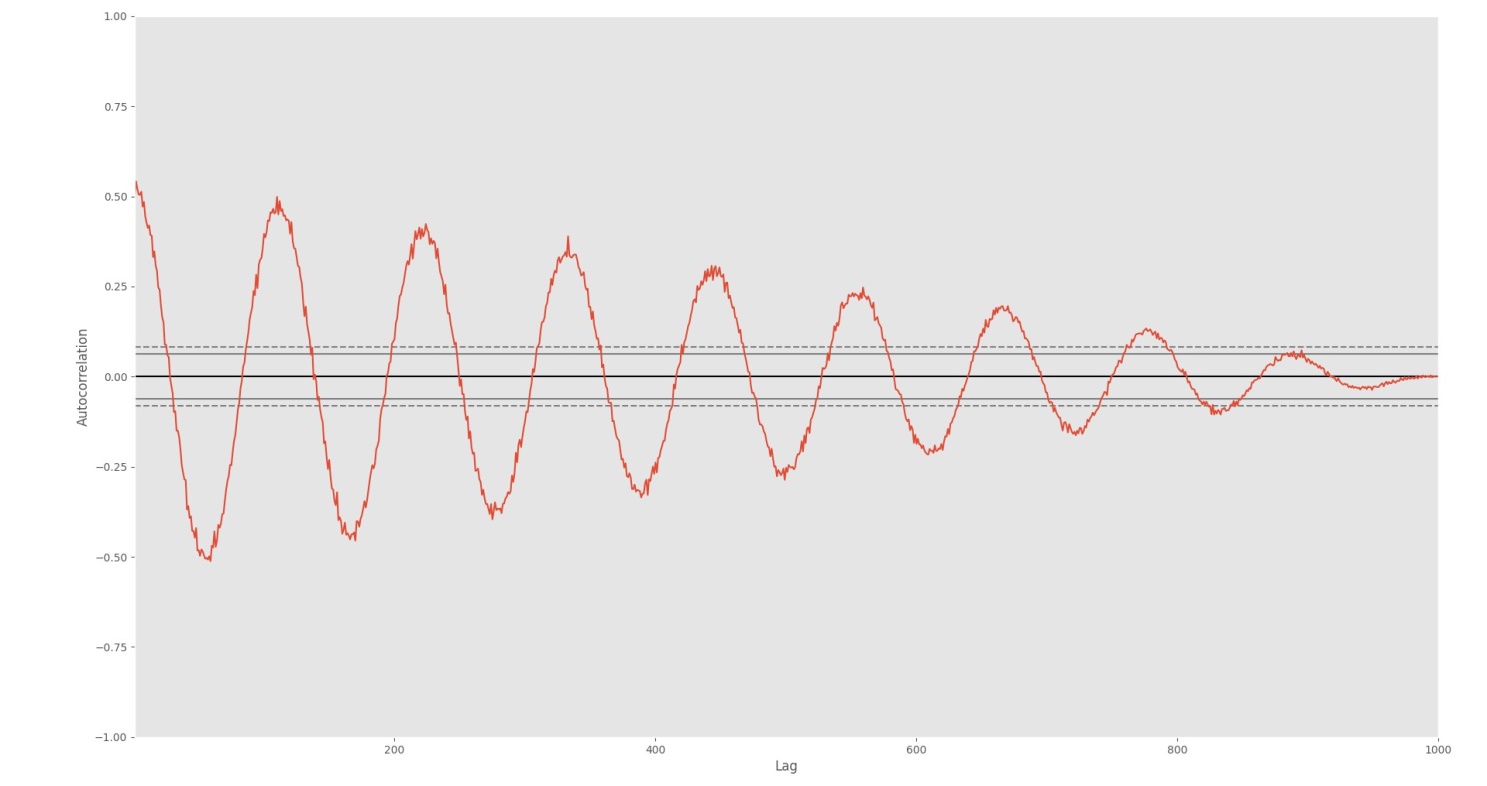
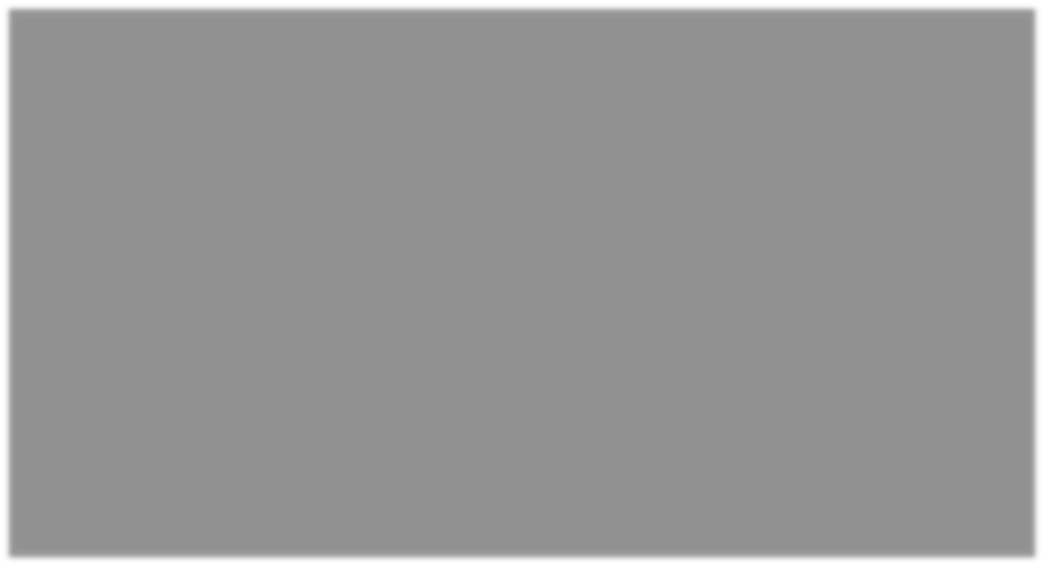
Python/bootstrap\_plot.png"

plt.savefig(sPicNameOut3, dpi=600)

plt.tight\_layout()

plt.show()

**Output:**

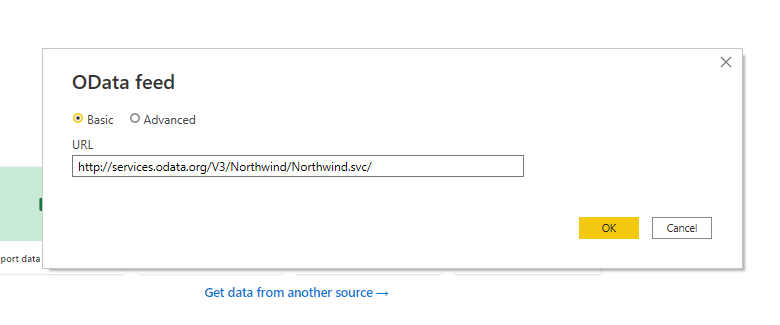


**Practical No: 10**

**Aim: Data Visualization with Power BI**

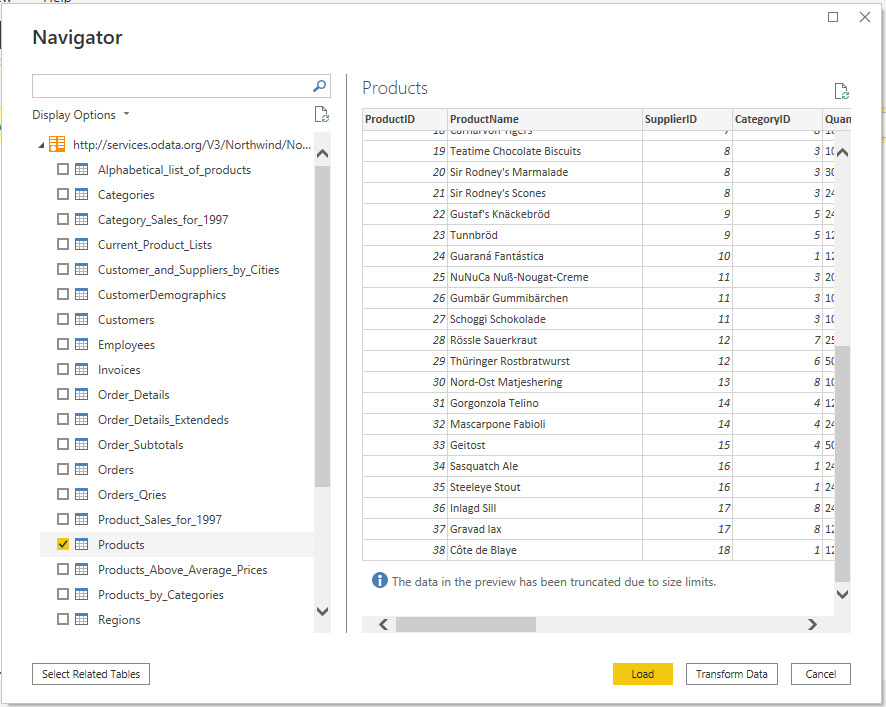
**Importing Data from OData Feed**

In this task, you'll bring in order data. This step represents connecting to a sales system. 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/>



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.
5. In the Navigator pane, select the Orders table, and then select Edit.



**ETL Process in Power BI**

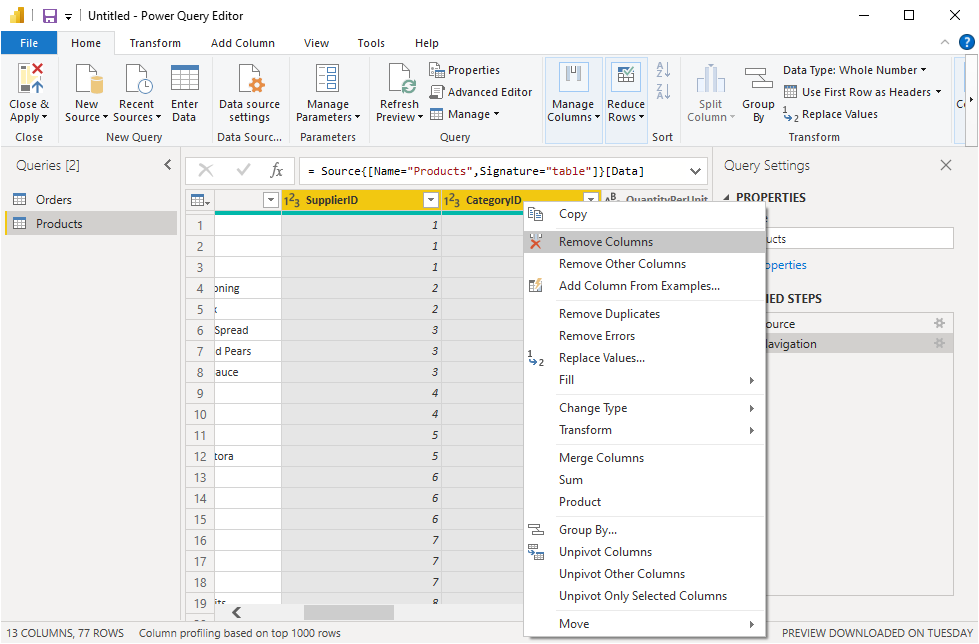
#### 1. Remove other columns to only display columns of interest

In this step you remove all columns except **ProductID**, **ProductName**, **UnitsInStock**, and

#### QuantityPerUnit

Power BI Desktop includes Query Editor, which is where you shape and transform your data connections. Query Editor opens automatically when you select **Edit** from Navigator. You can also open the Query Editor by selecting Edit Queries from the Home ribbon in Power BI Desktop. The following steps are performed in Query Editor.

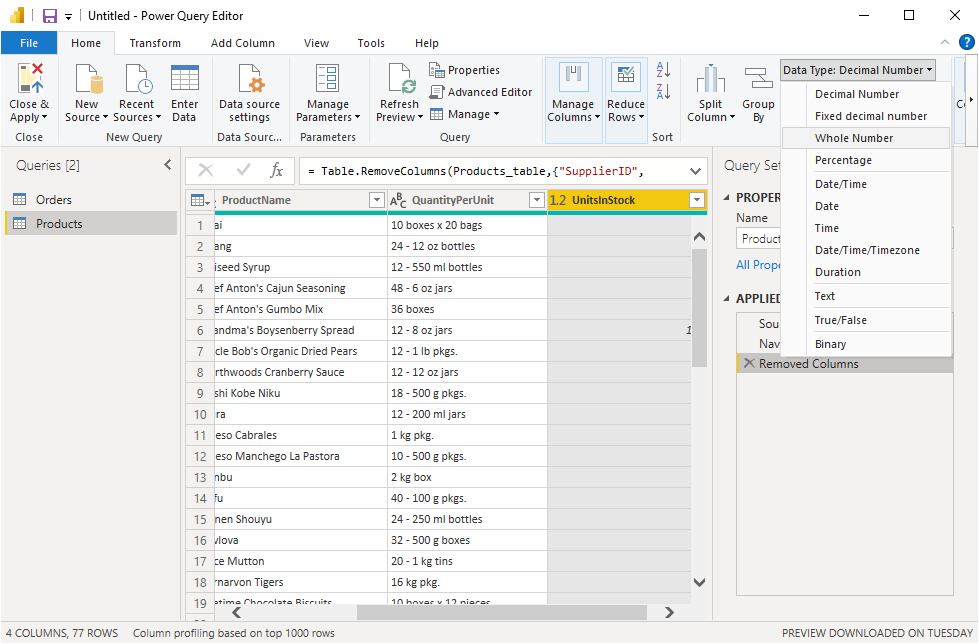
1. In **Query Editor**, select the **ProductID, ProductName**, **QuantityPerUnit,** and **UnitsInStock** columns (use **Ctrl+Click** to select more than one column, or **Shift+Click** to select columns that are beside each other).
2. Select **Remove Columns > Remove** Other Columns from the ribbon, or right-click on a column header and click Remove Other Columns.



#### 2. Change the data type of the UnitsInStock column

When Query Editor connects to data, it reviews each field and to determine the best data type. For the Excel workbook, products in stock will always be a whole number, so in this step you confirm the **UnitsInStock** column’s datatype is Whole Number.

1. Select the **UnitsInStock** column.
2. Select the **Data Type drop-down button** in the **Home ribbon**.
3. 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).



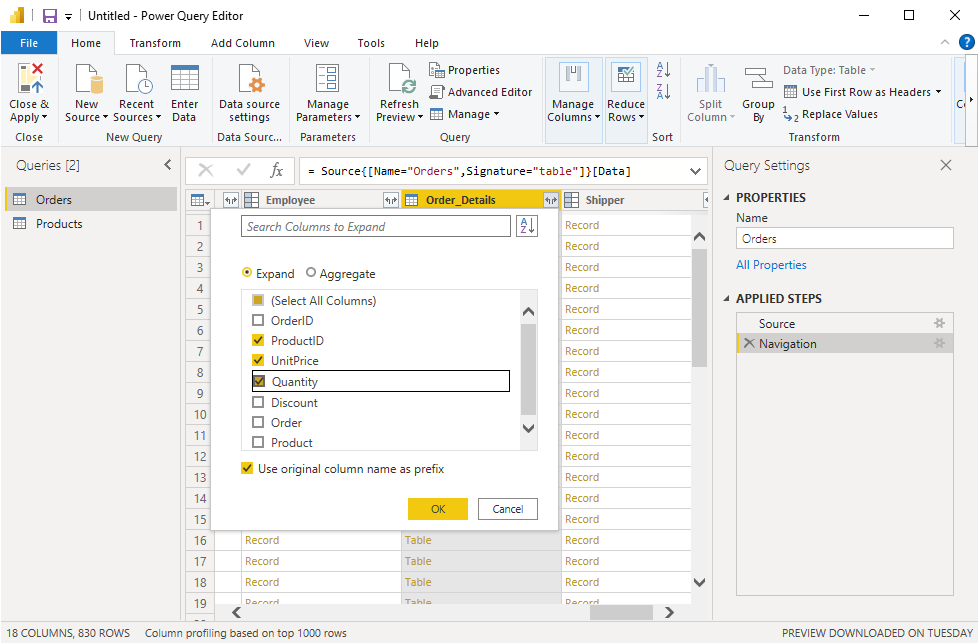
#### 3. Expand the Order\_Details table

The Orders table contains a reference to a Details table, which contains the individual products that were included in each Order. When you connect to data sources with multiples tables (such as a relational database) you can use these references to build up your query. In this step, you expand the **Order\_Details** table that is related to the Orders table, to combine the **ProductID**, **UnitPrice**, and **Quantity** columns from **Order\_Details** into the **Orders table**. This is a representation of the data in these tables:

The Expand operation combines columns from a related table into a subject table. When the query runs, rows from the related table (**Order\_Details**) are combined into rows from the subject table (**Orders**).

After you expand the Order\_Details table, three new columns and additional rows are added to the Orders table, one for each row in the nested or related 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:
4. Select (Select All Columns) to clear all columns.
5. Select ProductID, UnitPrice, and Quantity.
6. Click OK.

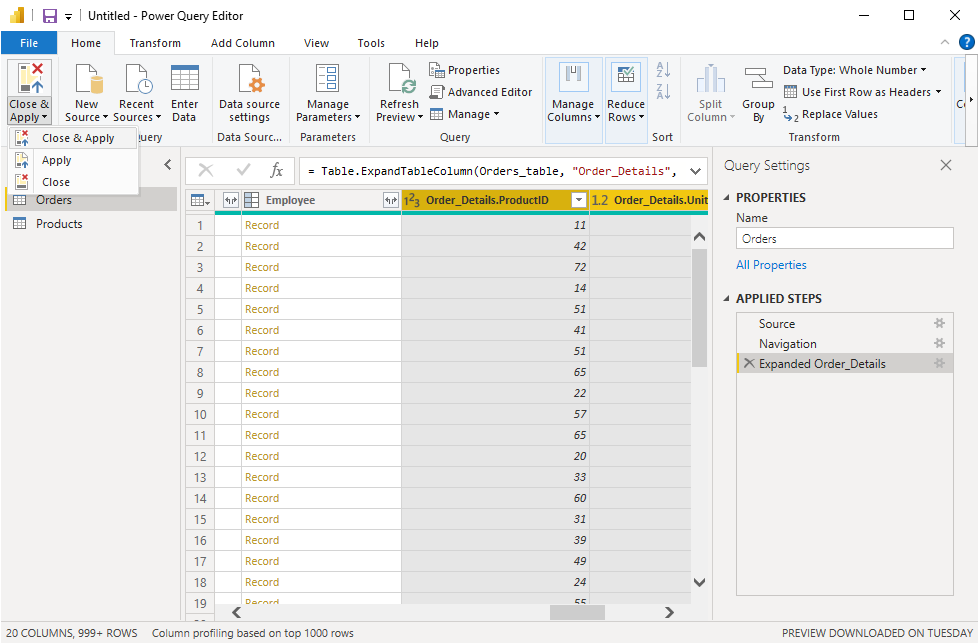


#### 4. Calculate the line total for each Order\_Details row

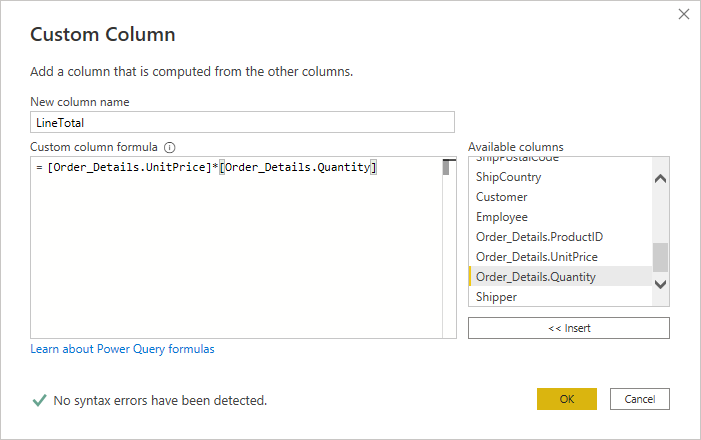
Power BI Desktop lets you to create calculations based on the columns you are importing, so you can enrich the data that you connect to. In this step, you create a Custom Column to calculate the line total for each Order\_Details row.

Calculate the line total for each Order\_Details row:

1. In the Add Column ribbon tab, click Add Custom Column.



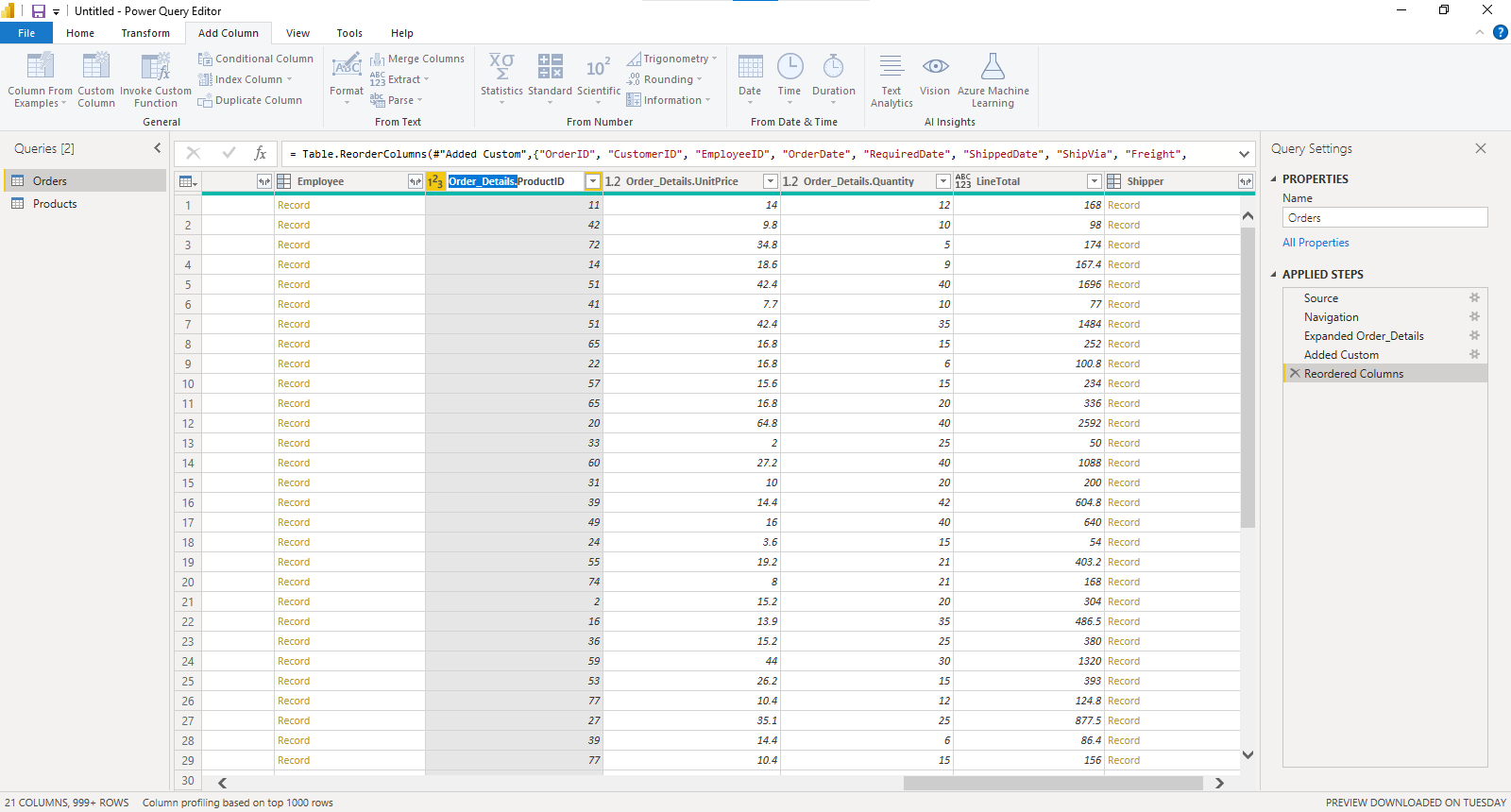
1. In the Add Custom Column dialog box, in the Custom Column Formula textbox, enter [Order\_Details.UnitPrice] \* [Order\_Details.Quantity].
2. In the New column name textbox, enter LineTotal.
3. Click OK.



**5. Rename and reorder columns in the query**

In this step you finish making the model easy to work with when creating reports, by renaming the final columns and changing their order.

1. In Query Editor, drag the LineTotal column to the left, after ShipCountry.
2. Remove the Order\_Details. prefix from the Order\_Details.ProductID, Order\_Details.UnitPrice and Order\_Details.Quantity columns, by double-clicking on each column header, and then deleting that text from the column name.



#### Combine the Products and Total Sales queries

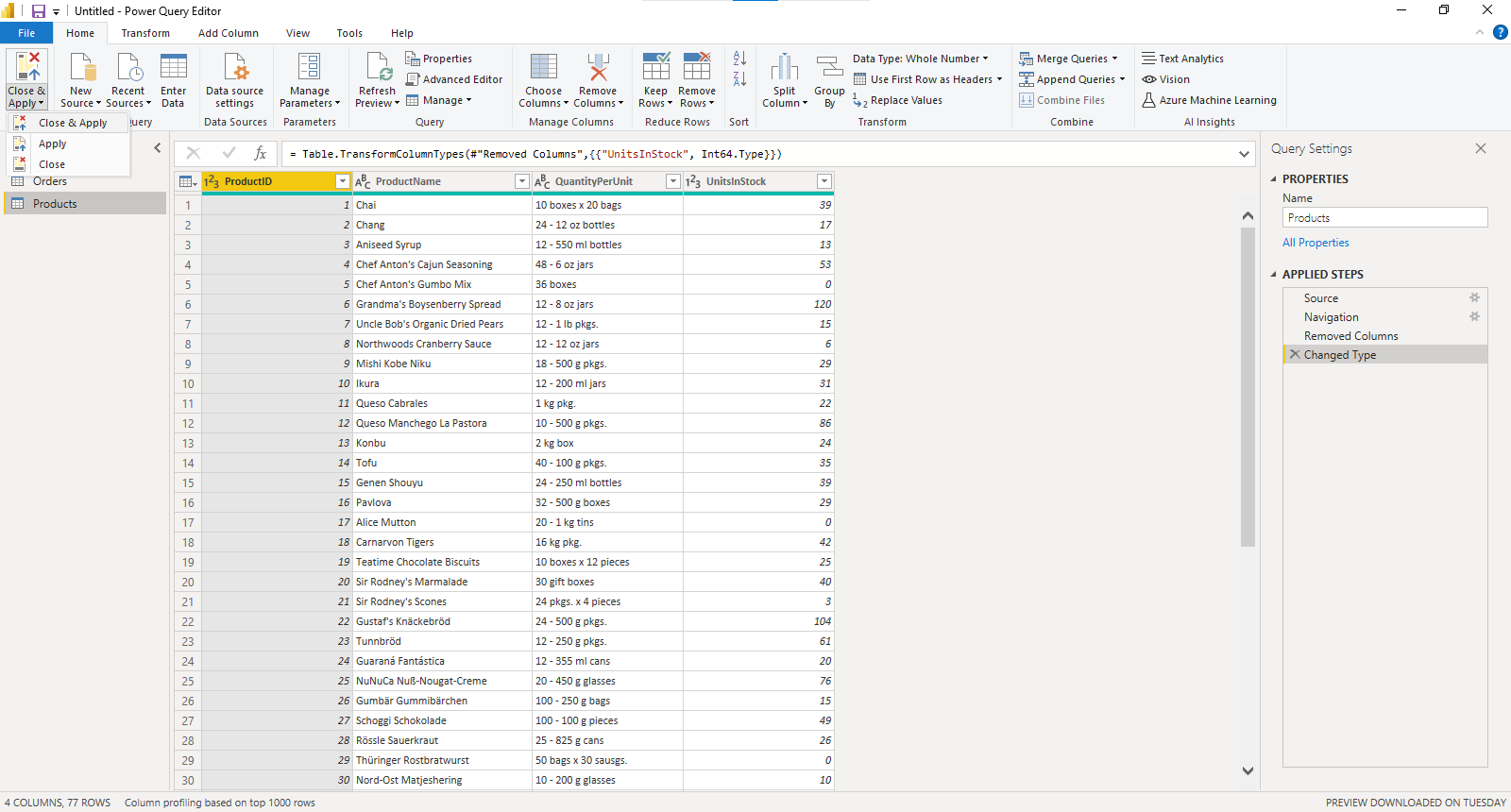
Power BI Desktop does not require you to combine queries to report on them. Instead, you can create Relationships between datasets. These relationships can be created on any column that is common to your datasets

we have Orders and Products data that share a common 'ProductID' field, so we need to ensure there's a relationship between them in the model we're using with Power BI Desktop. Simply specify in Power BI Desktop that the columns from each table are related (i.e. columns that have the same values). Power BI Desktop works out the direction and cardinality of the relationship for you. In some cases, it will even detect the relationships automatically.

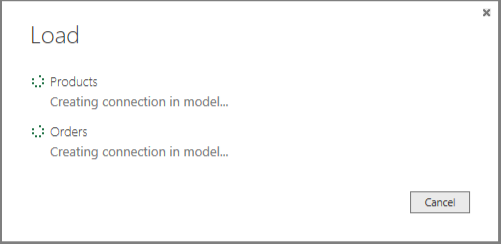
In this task, you confirm that a relationship is established in Power BI Desktop between the Products and Total Sales queries

Step 1: Confirm the relationship between Products and Total Sales

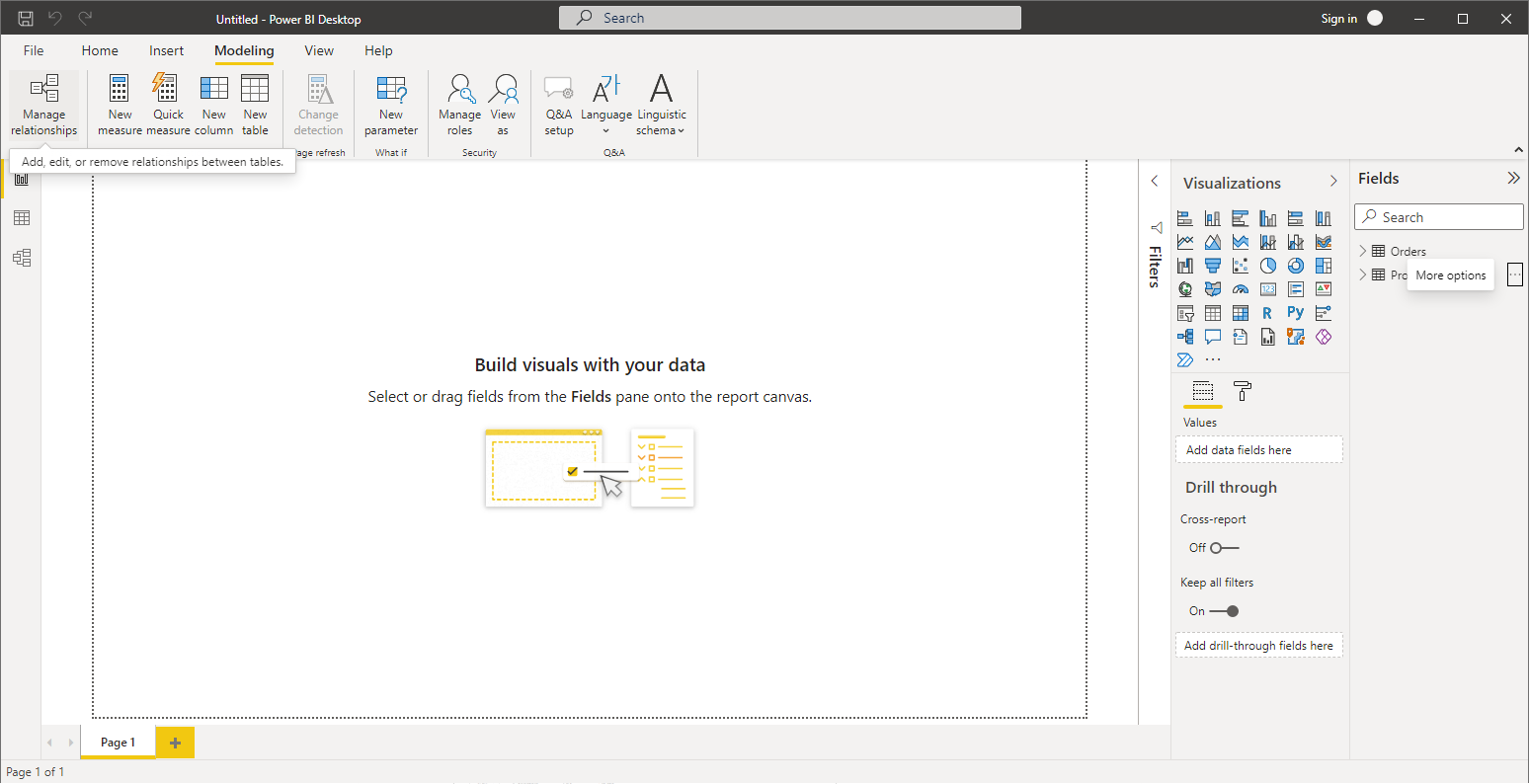
First, we need to load the model that we created in Query Editor into Power BI Desktop. From the Home ribbon of Query Editor, select Close & Load.



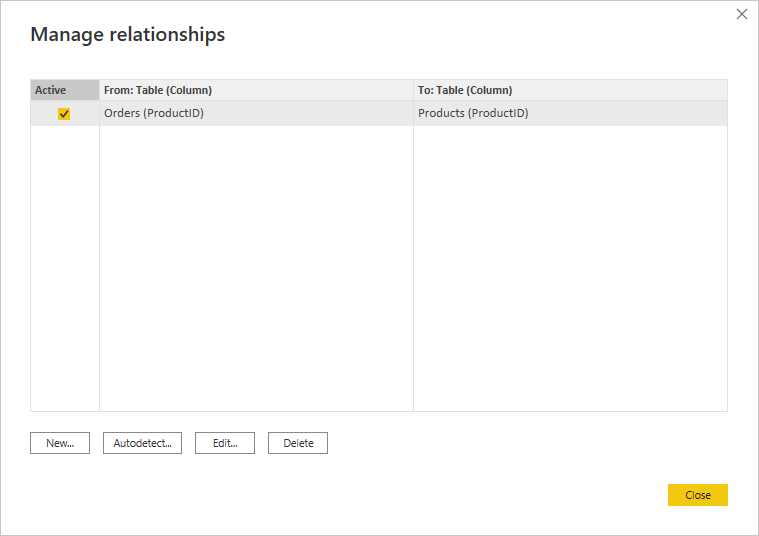
Power BI Desktop loads the data from the two queries.



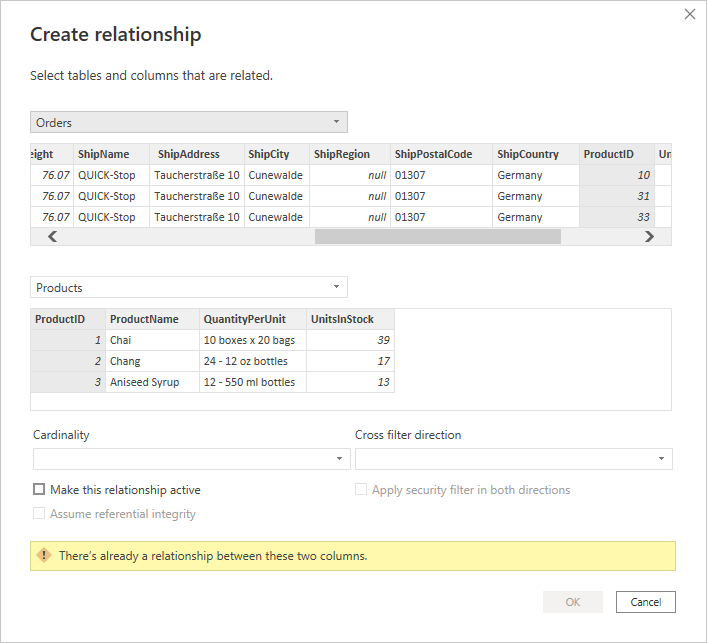
Once the data is loaded, select the Manage Relationships button Home ribbon.



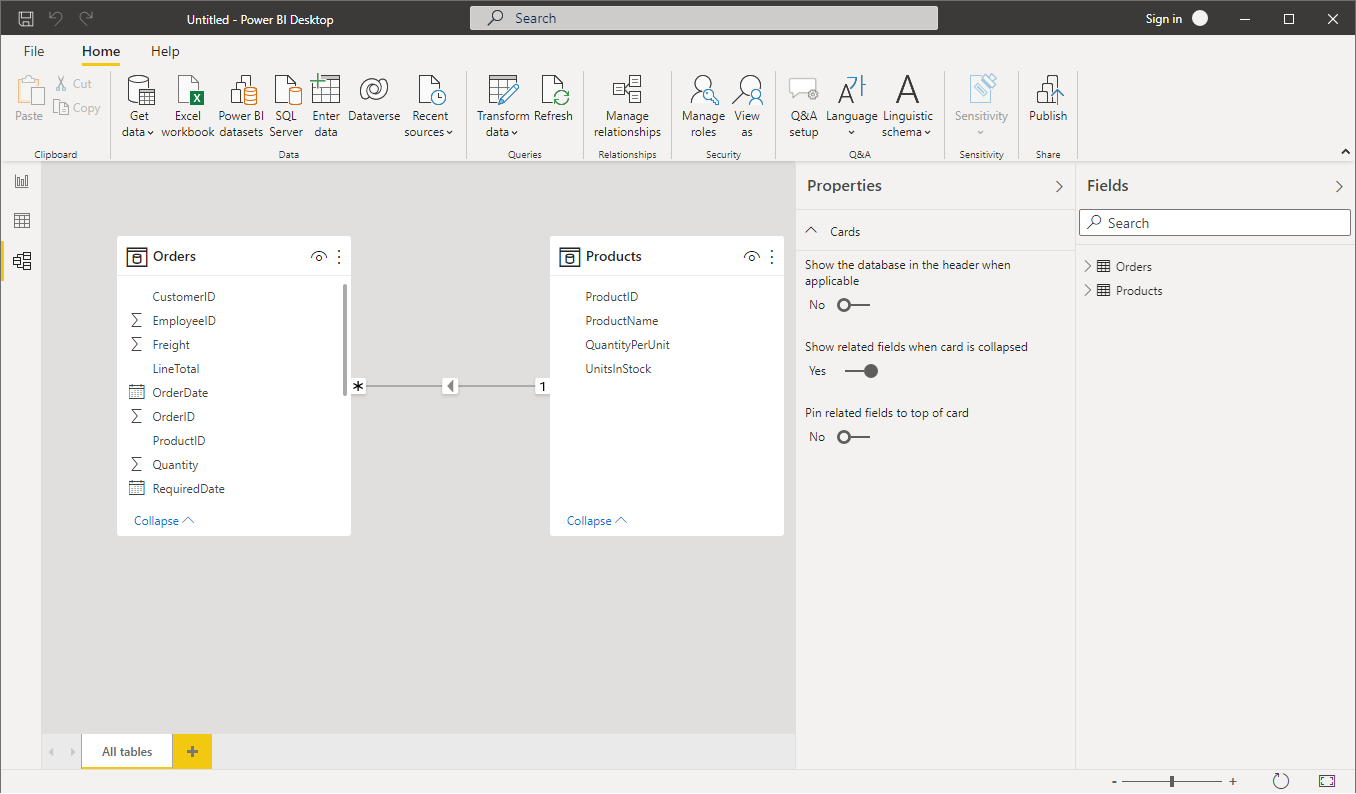
1. Select the New… button



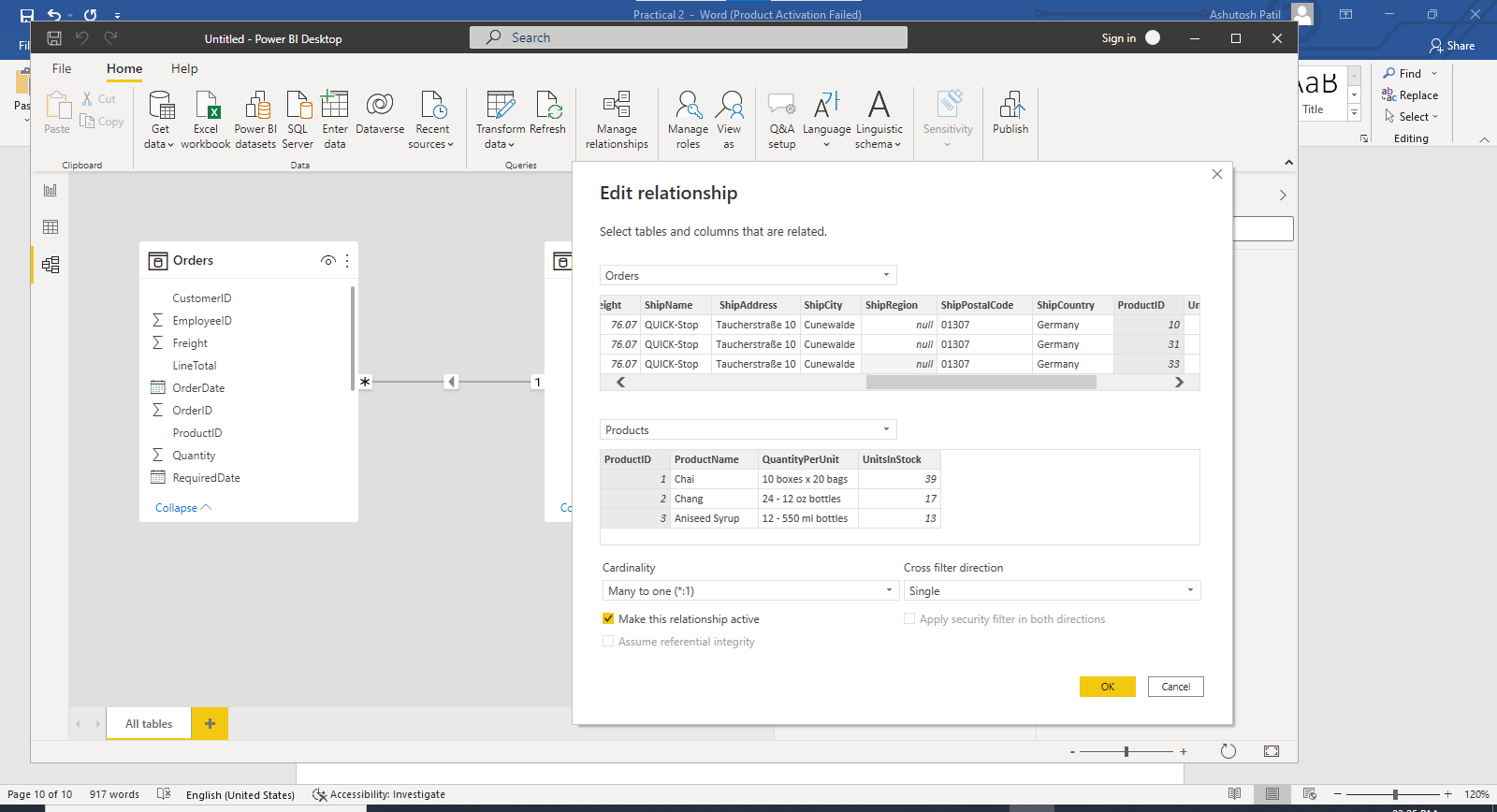
1. When we attempt to create the relationship, we see that one already exists! As shown in the Create Relationship dialog (by the shaded columns), the ProductsID fields in each query already have an established relationship.



1. Select Cancel, and then select Relationship view in Power BI Desktop.



1. We see the following, which visualizes the relationship between the queries
2. When you double-click the arrow on the line that connects the to queries, an Edit Relationship dialog appears.



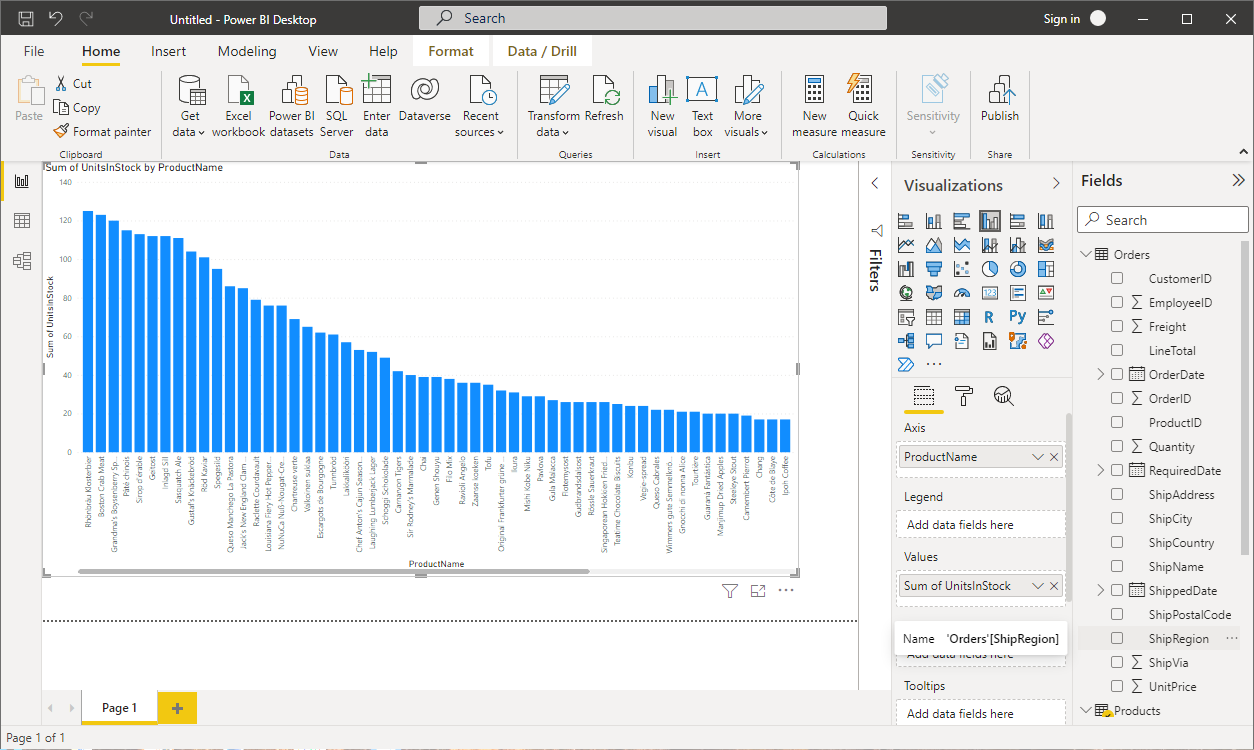
1. No need to make any changes, so we'll just select Cancel toclose the Edit Relationship dialog.

Power BI Desktop lets you create a variety of visualizations to gain insights from your data. You can build reports with multiple pages and each page can have multiple visuals. You can interact with your visualizations to help analyze and understand your data

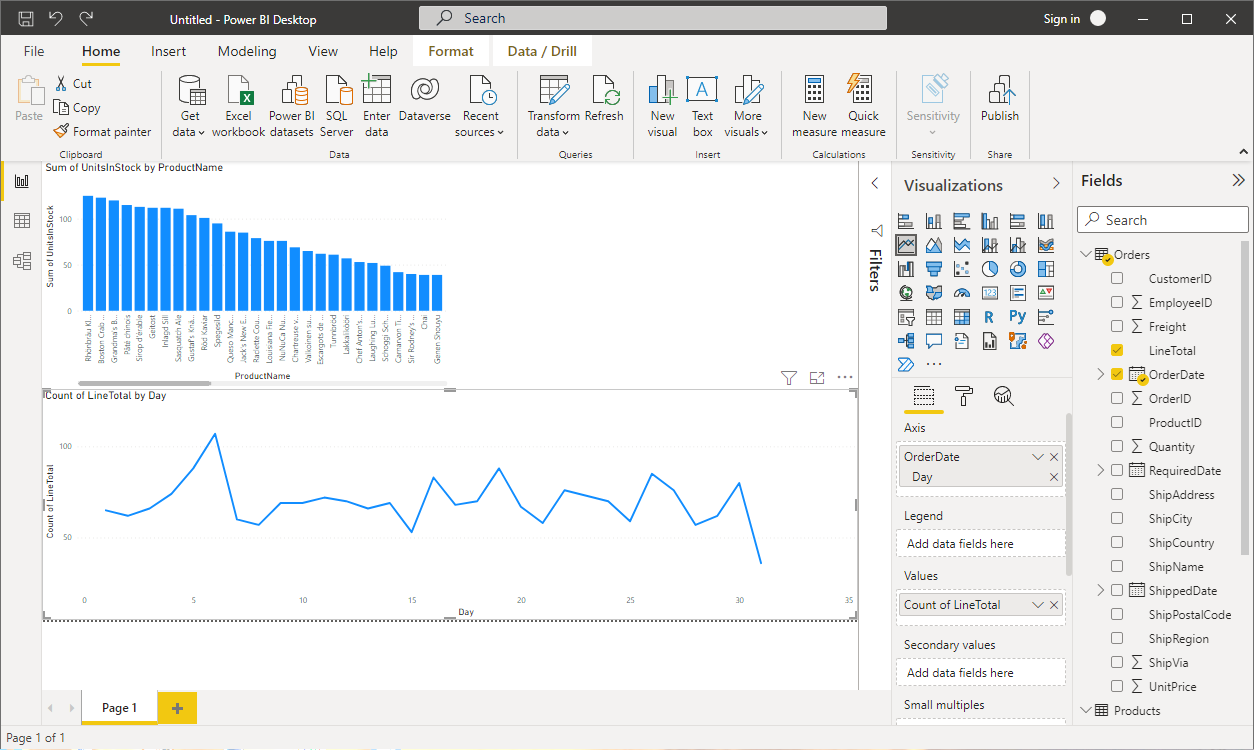
In this task, you create a report based on the data previously loaded. You use the Fields pane to select the columns from which you create the visualizations.

#### Step 1: Create charts showing Units in Stock by Product and Total Sales by Year

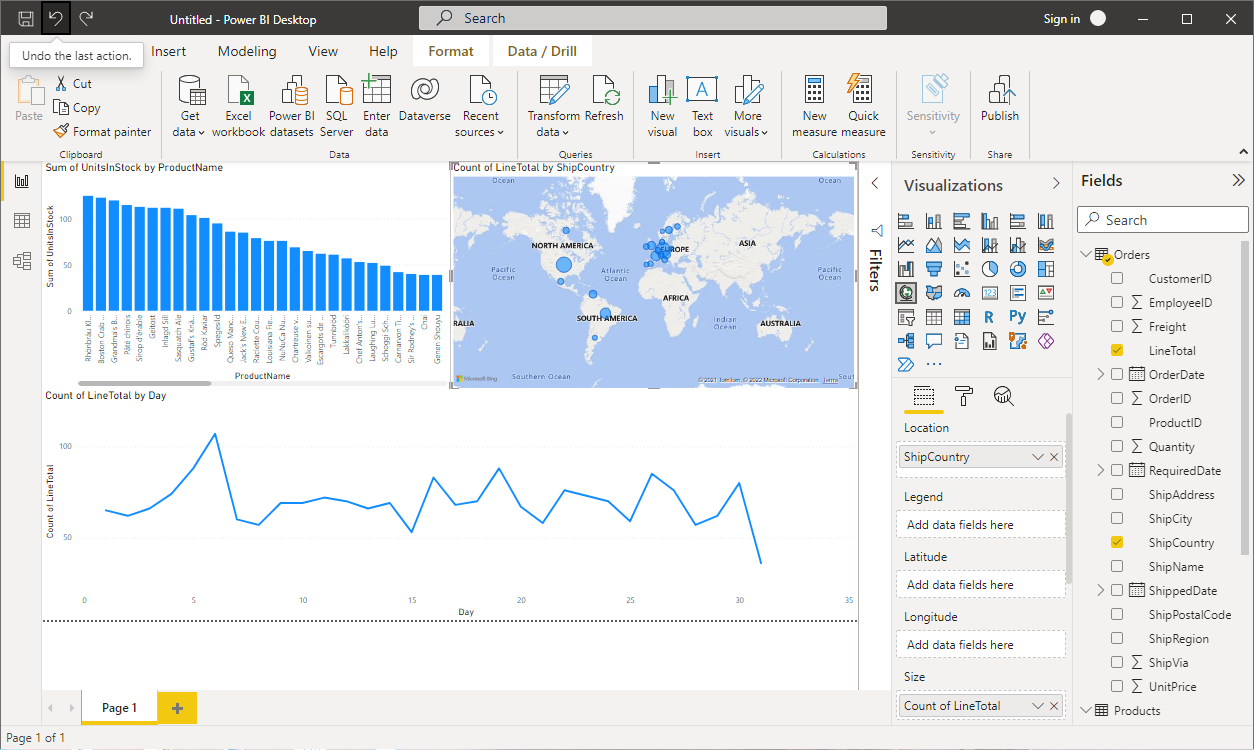
1. Drag UnitsInStock from the Field pane (the Fields pane is along the right of the screen) onto a blank space on the canvas. A Table visualization is created. Next, drag ProductName to the Axis box, found in the bottom half of the Visualizations pane. Then we then select Sort By > UnitsInStock using the skittles in the top right corer of the visualization.



1. Drag OrderDate to the canvas beneath the first chart, then drag LineTotal (again, from the Fields pane) onto the visual, then select Line Chart. The following visualization is created.



1. Next, drag ShipCountry to a space on the canvas in the top right. Because you selected a geographic field, a map was created automatically. Now drag LineTotal to the Values field; the circles on the map for each country are now relative in size to the LineTotal for orders shipped to that country.



#### Step 2: Interact with your report visuals to analyze further

Power BI Desktop lets you interact with visuals that cross-highlight and filter each other to uncover further trends.

1. Click on the light blue circle centered in Canada. Note how the other visuals are filtered to show Stock (ShipCountry) and Total Orders (LineTotal) just for Canada.

